HOW THE BRAIN MIGHT WORK: A HIERARCHICAL AND TEMPORAL MODEL FOR LEARNING AND RECOGNITION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Dileep George
June 2008

Acknowledgements

I am grateful to Jeff Hawkins for providing the intellectual framework behind this thesis and for financially supporting a part of my graduate research. Neuroscience is a vast field. Without the guidance provided by Jeff, I would have easily got lost. His immense knowledge and intensity of thought have always been a constant source of motivation. One of my favorite weekend routines has been leafing through Jeff's artificial intelligence and neuroscience library. Studying the astute observations that he scribbled on those books over a period of 20 years has been a source of inspiration and a reminder of the virtues of perseverance.

I would like to thank my official adviser, Bernard Widrow, for his guidance and support throughout the course of this work. He agreed to an unusual arrangement – I was allowed to spend all my time at the Redwood Neuroscience Institute (and later at Numenta) while working on my dissertation. He read my early manuscripts and patiently guided me to write with clarity. Anything readable in this thesis is because of him.

I am grateful to Nils Nilsson for encouraging me to work on this dissertation while working at Numenta. He read the early manuscripts of this thesis and made several corrections and helpful suggestions. This thesis would not have been possible without him. I would like to thank Brian Wandell for being my co-advisor and for the helpful suggestions that improved the neuroscience parts of this thesis. He directed me towards developing some of the theoretical aspects of hierarchical learning. Thanks to Vinod Menon, John Pauly and Tom Dean for serving on my orals committee.

I am grateful to all my colleagues at Numenta. Bobby Jaros collaborated with me in deriving the belief propagation equations for HTMs. I have always enjoyed the intense discussions with him. Jeff, Subutai and Jamie read early versions of the chapters and made very helpful comments and corrections. I am grateful to Subutai and Donna for giving me the flexibility to work on this thesis. Thanks to Tom Dean for introducing me to the concept of inductive bias during his stint at Numenta. His careful questioning brought some rigor to my thinking that I couldn't have got from anywhere else.

I would like to thank everybody at the erstwhile Redwood Neuroscience Institute. I was fortunate to have access to Bruno Olshausen, Tony Bell, Fritz Sommer, Pentti Kanerva and Bill Softky. Their insights, immense knowledge, and friendship have been crucial to the development of this work. I have always enjoyed my discussions with Kilian, Thomas and Matthias. I learned a great deal from the weekly seminars and journal clubs at RNI. Thanks to all the anonymous reviewers of my rejected and accepted papers. They led me in the right direction and introduced me to related areas of work.

Thanks to my friends at Stanford for the wonderful time I have had with them and for their constant encouragement. I was fortunate to have Jamie as a friend during my first year at Stanford. He listened patiently and encouragingly to my wild ideas for exploring neuroscience and helped me with the preparations for my PhD qualifiers. Thanks to Dua, Arpi, Sam, Dinesh, Dos, Jayanth, Mukul, Nikhil, Sahoo, Madhup, Navneet, Garima, Ashmi, Rishi, Raghav, Bipin, Ritesh and Kaustuv for the numerous hikes, dinners and games. Thanks to Nitin Bakshi for being a constant source of encouragement and inspiration right from my undergraduate days.

Thanks to the *Eureka* magazine for making me interested in science during my grade school days. I feel fortunate to have had wonderful school friends who shared this interest. Friends I haven't met – Mohammed Sherif, Anil Kumar, and many others – kept this interest alive by sending me books that I wouldn't otherwise have access to. Thanks to Priya for encouraging me to do a Ph.D., and thanks to Valyappachan for giving me access to his personal library.

I would like to thank Bapi for his helpful comments on my early manuscripts. Thanks to Deepa and Gigichettan for shouldering many responsibilities while I was away from home. I am grareful to Panki for all the help he provides to my mother.

Special thanks are due to my lovely wife, Sudeshna, for believing in me and for supporting me through my ups and downs at Stanford, RNI and Numenta. This thesis would not have been possible without her.

Anything good that I have become is because of my mother. To me, she has always been the epitome of perseverance, compassion, sacrifice and love. This dissertation is dedicated to her.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Suppose you are traveling to India for the first time. You see a new contraption that runs on three wheels. From your tourist guide you learn that it is called an *auto rickshaw*. After this incident, you notice that roads in India are filled with auto rickshaws. They come in different sizes, shapes, colors and adornments. After seeing a few examples of auto rickshaws, you are able to correctly recognize all the other auto rickshaws.

Our brains are learning systems. It is unlikely that evolution programmed in the knowledge of auto rickshaws. You had to learn about them by getting exposed to patterns. After seeing a few auto rickshaws, you are able to recognize novel auto rickshaws although the patterns that correspond to those novel ones are not identical or close to the patterns of the ones you first saw. This capability of learning systems to recognize new patterns is termed generalization. The generalization capabilities of human brains are yet to be matched by our algorithms.

There are two main goals of this thesis. One is to illuminate some principles required for the building of machines that learn, generalize and exhibit some form of 'intelligence'. The other is to study how the neocortex works. These are not independent goals. In the following sections, I will argue why both these goals go hand-in-hand; understanding the neocortex is required for building intelligent machines and having computational theories for building intelligent machines is required for understanding how the brain works.

1.1 Why study the neocortex?

There are philosophical and anthropological reasons for studying the neocortex; understanding the neocortex is part of understanding what it is to be human. However, there are also important computational and engineering reasons to study the neocortex. Understanding the principles of operation of the neocortex is a necessary step towards building intelligent machines.

The necessity of studying the brain can be argued from the point of view of machine learning. In machine learning, a researcher first constructs a parameterized adaptive model for the data to be modeled. This initial model, constructed based on domain knowledge, will have several parameters that can be tuned based on the training data. The researcher then devises algorithms that can adapt these parameters. Typically, the parameters are tuned by adapting them to minimize an error criterion [24, 10]. A model whose parameters are thus tuned can then be used to classify novel patterns, predict future patterns, compress, reconstruct etc. However, despite its mathematical elegance, this method hides some fundamental problems associated with learning.

One aspect of learning is the sample complexity – the number of examples required for training a model to a reasonable level of performance. If the hypothesis space for a learning problem is very large, then the construction of a learned model can take a large number of training examples and long training times. The No Free Lunch (NFL) theorem, a fundamental result on search and optimization, addresses aspects about the complexity of learning [50]. Intuitively, the NFL theorem for learning argues that no learning algorithm has an inherent superiority over another algorithm for all learning problems. If an algorithm is superior for a particular problem, it is only because that algorithm exploited assumptions that are suitable for that problem. The same algorithm will not do well on a problem that has assumptions different from that of the original one. This means that, to make the algorithms effective, the machine-learning researcher has to encode knowledge about the problem domain into the initial model structure. The more prior knowledge that is put in, the easier it is to learn. Does that mean we need to create a new model for each new problem that we try to solve using machine learning? Clearly, such an approach would be very human intensive.

On the other hand, humans and other mammals seem to solve these problems in a different way altogether. The fact that humans can learn and adapt to problems that did not exist when the initial model (the neocortex) was created is proof of the generic nature of the mechanisms used by the human brain. Moreover, a large number of researchers now conjecture that the neocortex is using fundamentally the same algorithm for learning different modalities. This means that the same algorithm is in operation for learning models for audition, vision, somatosensory perception and language.

Although not proven conclusively, there is increasing amount of evidence for a common cortical algorithm. The existence of a common cortical algorithm was conjectured first by a neuroscientist who studied cortical tissues of different mammals [68]. The remarkable similarity of the cortical connectivity across different modalities and even across different species was the reason for this conjecture. Experimental results on ferrets provided further support for this theory [103]. In these experiments, the optical nerve fibers in the ferret were re-routed to the auditory area. With exposure to visual patterns, the auditory area of the ferret developed visual receptive fields and was able to demonstrate behavior using this "audio-vision". Sensory substitution studies where the impression of visual patterns on to the somatosensory modality gives rise to visual perception also lends support to the existence of a common cortical algorithm [3, 2].

The combination of the common cortical algorithm and the NFL theorem produces important implications for machine learning and for our quest to build intelligent machines. On the surface, the NFL theorem seems to create problems for the idea of a common cortical algorithm. How can one mechanism/algorithm do very well on tasks as different as vision, audition and language? The answer comes from the part of the NFL theorem that talks about the assumptions that need to be exploited. The common cortical algorithm and the NFL theorem can be consistent with each other provided we can exploit the same basic assumptions to learn vision, audition, language etc. If the cortex is good at learning a wide variety of tasks using a common mechanism, then there must be something common about these seemingly different tasks. Evolution must have discovered this commonality and neocortex must be exploiting that.

From the NFL theorem we conclude that a universal theory of learning is, in essence, a universal theory of assumptions. The set of assumptions that a learner uses to predict outputs, given inputs that it has not encountered, is known as the *inductive bias* of that learning algorithm [66, 36]. The more assumptions we make, the easier it becomes to learn. However, the more assumptions we make, the fewer the number of problems we can solve. If we need to design an algorithm that can be applied to a large class of problems, the question we need to ask is: *What is the basic set of assumptions that are specific enough to make learning feasible in a reasonable amount of time while being general enough to be applicable to a large class of problems?* We are in luck if the same set of assumptions works for a large class of problems.

This brings us to the question we raised at the beginning of this section. We should study the neo-cortex because we need to find out the precise set of assumptions about the world that should be encoded in our learning algorithms. First, the neo-cortex is an existence proof that such a general set of assumptions exists. Second, studying the neo-cortex in the right way will tell us more about these assumptions.

1.2 How should we study the neocortex?

The problem of reverse-engineering the neocortex is a daunting one. How should we search for these universal assumptions? There are many anatomical and physiological details in the brain; how is one to know what is important and what is not? What is a mere implementation detail that biology has to employ because it has only neurons to work with, and what is an important computational principle that cannot be missed?

A good strategy would be to study the neo-cortex and the world at the same time. Studying the anatomy and physiology of the neocortex should give us important clues about the nature of the assumptions made by the neocortex. While studying the organization of the neocortex, we will have to look for general principles that can be relevant from a learning point of view. We should select only those principles for which we can find counterparts in the world. If an organization property in the neocortex is matched to an organization property of the world, we can be reasonably certain that we have found a principle that is relevant from a learning point of view.

1.3 The Memory-Prediction framework

Fortunately, Jeff Hawkins has already blazed the trails through his book entitled *On Intelligence* [43]. In this book, Hawkins proposed a theory for the operation of the cortex that is not only rooted in the assumptions of biology, but also makes sense from a computational point of view. The theory, named Memory-Prediction framework, is expressed in terms of computations done in biological circuits. The main points of the Memory-Prediction framework can be summarized as follows:

1. The neocortex is constructing a model for the spatial and temporal patterns that it is exposed to. The goal of this model construction is the prediction of the next pattern on the input.

2. The cortex is constructed by replicating a basic computational unit known as the canonical cortical circuit. From a computational point of view, this canonical circuit can be treated as a node that is replicated several times.

3. The cortex is organized as a hierarchy. This means that the nodes – the basic computational units – are connected in a tree shaped hierarchy.

4. The function of the cortex is to model the world that it is exposed to. This model is built using a spatial and temporal hierarchy by memorizing patterns and sequences at every node of the hierarchy. This model is then used to make predictions about the input.

5. The neocortex builds its model of the world in an unsupervised manner.

6. Each node in the hierarchy stores a large number of patterns and sequences. The pattern recognition method employed by the cortex is largely based on storing lots of patterns.

7. The output of a node is in terms of the sequences of patterns it has learned.

8. Information is passed up and down in the hierarchy to recognize and disambiguate information and propagated forward in time to predict the next input pattern.

The Memory-Prediction framework, as expressed in *On Intelligence* is a biological theory. In this thesis, we work on the foundation established by Hawkins and develop the algorithmic and mathematical counterparts of the Memory-Prediction framework. We will call this by the name Hierarchical Temporal Memory (HTM)[1]. We will also work back from the theory of HTM to biology in order to derive a mathematical model for the microcircuits of the neocortex.

1.4 Main contributions and organization

This thesis makes three main contributions. These contributions are organized in three chapters.

1. **Learning and invariant recognition algorithms for hierarchical-temporal data:** This thesis introduces new algorithms, collectively called Hierarchical Temporal Memory, that can be used to learn hierarchical-temporal models of data. Temporal continuity is used to learn multiple levels of a hierarchy in an unsupervised manner. The algorithms, when applied to a visual pattern recognition problem, exhibit invariant recognition, robustness to noise, and generalization. Inference in the hierarchy is done using Bayesian belief propagation equations that are adapted to this problem setting. (Chapter 2)

2. **Analysis of generalization in hierarchical-temporal models:** This thesis develops a generative model for HTMs. The data generated from this model is used to analyze and quantify the generalization properties of HTMs. (Chapter 3)

[1]This term was coined by Hawkins and the basic concepts behind HTM was described in [44]

3. **A mathematical model for cortical microcircuits:** This thesis introduces a mathematical model for cortical microcircuits based on the theory of Bayesian belief propagation in Hierarchical Temporal Memory networks. The proposed model has a laminar and columnar organization and matches many known anatomical and physiological data. The derived cortical microcircuit is used to explain the illusory contour response phenomenon, and the phenomenon of activity reduction in the primary visual cortex as a result of object recognition at higher levels. (Chapter 4)

Chapter 2

Hierarchical Temporal Memory: A Visual Pattern Recognition Example

2.1 Introduction

Vision is the primary sensory modality for humans and most mammals to interact with this world. In humans, vision related areas occupy about 30 percent of the neo-cortex. Light rays reflecting from physical objects enter through our eyes and form an image on the retina in our eyes. Our brains then interpret these images to make sense of the world. Although this seems virtually effortless and automatic, in the process our brains solve many problems that are not yet solved by a computer.

One such problem is the problem of invariant visual pattern recognition. Humans and most mammals can recognize images despite changes in location, size, lighting conditions and in the presence of deformations and large amounts of noise. Several attempts have been made to solve this problem on a computer. Many of the prior attempts tried to solve the invariance problem by defining it as a classification problem. In most cases, the problem was defined as of having labeled examples from a certain number of categories of objects. The examples within a category would include translations, deformations and size changes of objects belonging to that category. Typically, a classifier would then be trained with these labeled examples. For instance, multi-layer neural networks trained with back propagation were frequently used for image classification. While these techniques seemed adequate to memorize the training set, they often proved inadequate at generalizing what they have learned to patterns that they have never seen.

Why did these techniques have only limited success? Several clues can be obtained by analyzing how humans and other mammals solve the vision problem. Many of the early research ignored the role of time in vision. Humans can recognize an object from a single snap shot presentation of its image without integrating information over multiple time steps. This could be the main reason why many researchers ignored the temporal component of vision and thought of vision in terms of images and not videos. Although humans can recognize with a snapshot, we are presented with continuously varying data while we learn and we could be learning important generalization characteristics from that temporal information. Another important aspect of mammalian learning is its unsupervised nature. Mammals do not train with labeled data. Although the role of time in and the unsupervised nature of the vision problem seem to be two separate aspects, they are two sides of the same coin.

2.1.1 Time and hierarchy in the vision problem

Learning or classification problems where labels are available for all the training patterns are called *supervised learning* problems. Visual pattern recognition problems are frequently treated as supervised learning problems. This could be due to the tendency of humans to name all the objects in the world. However, a brief examination of how other mammals might learn to do invariant visual pattern recognition will reveal the unsupervised nature of the vision problem.

For example, consider a cat walking towards its bowl of milk. With each step the cat takes, a different image of the bowl falls on the cat's retina. The Euclidean distance between two consecutive images on the cat's retina can be quite large. However, the cat still knows that it is looking at the same bowl of milk even if it is not able to

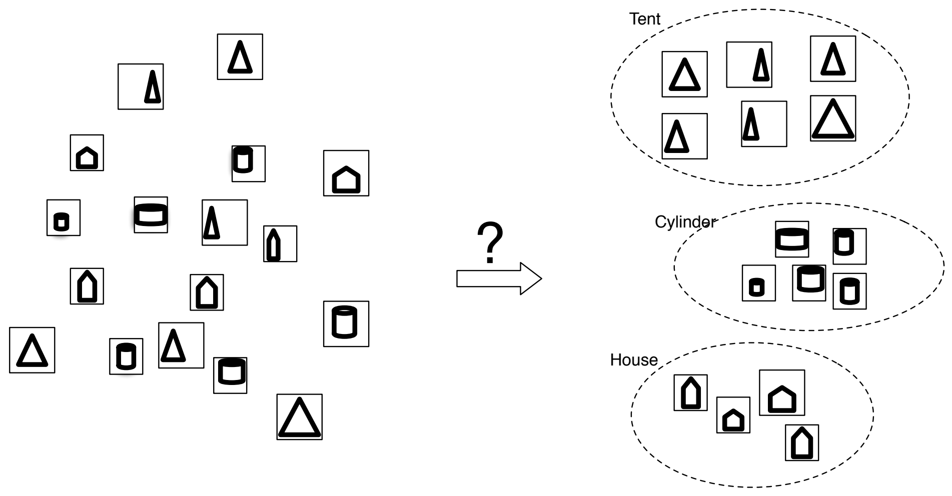


Figure 2.1: Unsupervised learning of object categories: Images belonging to three categories – "tent", "house" and "cylinder" – are shown on the left. Unsupervised learning of categories involves separating these images into their categories shown on the right.

name it as a bowl of milk. Nobody taught the cat about milk bowls by making it flip through pages of milk bowl pictures while shouting "milk bowl" in its ears. The cat had to learn in a completely unsupervised manner that the different images of milk bowl are actually caused by the same milk bowl.

The difficulty of this can be understood by looking at figure 2.1. On the left are images of different objects - "tent", "house" and "cylinder". How can one, in an unsupervised manner, cluster the objects that are perceptually similar to form groups as shown on the right in figure 2.1? A naive application of similarity metrics at the pixel level will not achieve the desired result because most often images from different categories have more overlap in the pixel space compared to images from the same category. Suppose one says that different images belonging to the same category are scaled and translated versions of each other. How is the cat to discover that fact? If scaling and translation are not pre-programmed within the cat's brain, then the cat will have to explore several other functions before it discovers that these images are related by scaling and translation – an impossible task.

Suppose that the number of pixels on the cat's retina is N. Then an input image

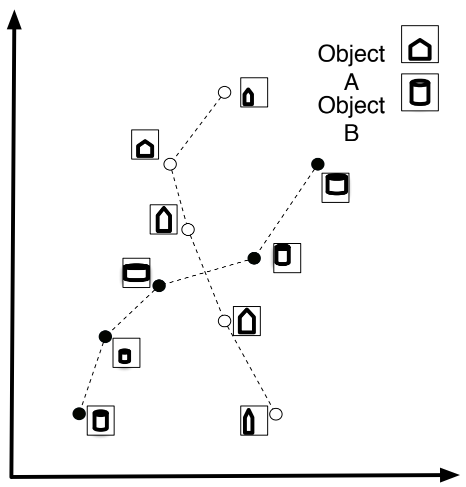


Figure 2.2: Visualization of object manifolds: Images generated by objects can be considered as points in a high-dimensional space. All the images generated by the same object can be thought to belong to the same *manifold* in this high-dimensional space. This figure represents two such manifolds - one belonging to object A ("House") and another belonging to object B ("Cylinder"). The points on these manifolds correspond to different images generated by the same object. The points belonging to the manifold of object A are shown using white-filled circles and the ones belonging to the manifold of object B are shown using black-filled circles. Intersecting points of manifolds correspond to identical views generated by two different objects.

can be considered as a vector of dimension N. Different images created by an object correspond to different points in an N-dimensional space. The set of points belonging to the same object can be considered to belong to a *manifold* in the N-dimensional space. These manifolds will contain an infinite number of points because they include infinitesimal variations between images of the same object. Different objects belong to different manifolds. The images of the milk-bowl that fall on the cat's retina can be thought of as samples from the milk-bowl manifold. This can be visualized as in figure 2.2. In figure 2.2, two manifolds, corresponding to objects A and B, are shown. The different points on the manifold A correspond to different images generated by the same object A. These images could, for example, correspond to viewing object A from different angles, distances, lighting conditions etc. Similarly, the points on manifold B correspond to different images generated by object B. If the cat has to learn that

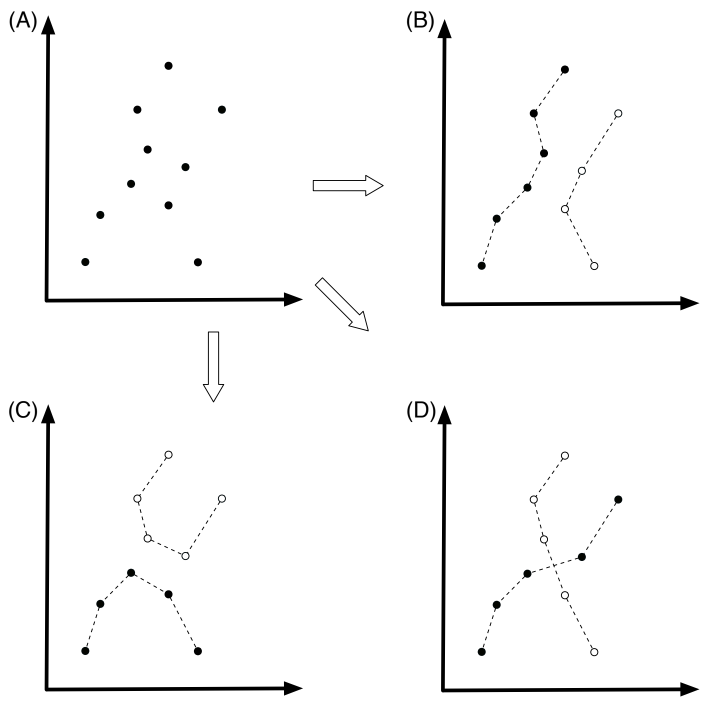


Figure 2.3: Unsupervised learning of objects would require separating the manifolds in a high dimensional space. Given just the points on the manifolds of different objects without their labels, there are several ways of splitting those points into manifolds. Figure (A) shows unlabeled points in a high dimensional space. Figures (B) to (D) illustrate several ways to split these points into two manifolds. Without further information than the one given in figure (A), it is impossible to prefer one segmentation over another.

all the images that are generated by object A are the same while being different from all the images generated by object B, it has to separate these two manifolds in an unsupervised manner. And all that the cat has to work with is samples from these manifolds.

The difficulty with separating these manifolds in an unsupervised manner is that the manifolds of different objects intersect or come arbitrarily close to one another. For example, some views of object A and object B can look the same. Those points will be common to both manifolds. If we are given points on two intersecting manifolds and not given the labels of these points, we will not be able to separate those manifolds without being provided additional information. In figure 2.3, we show three different

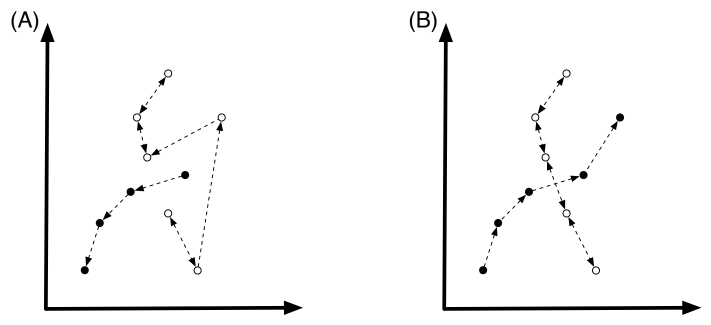


Figure 2.4: This figure shows how temporal information can be used to segment object manifolds. Temporal transitions are indicated using arrows. Figures (A) and (B) show two different temporal transition patterns. Segmenting the underlying manifolds involves identifying the points that are connected through temporal transitions. In both figures empty circles are used to denote the points of the first object and filled circles are used to denote the points of the second object.

segmentations of points from intersecting manifolds. We do not have any reason to favor a particular segmentation over another. Therefore, if we treat images as points in an N dimensional space, we cannot separate out the manifolds of objects in an unsupervised manner.

Fortunately, our cat does not have to solve this problem in this manner. For the cat, images in this world are presented continuously in time. When there is relative motion between the cat and the milk bowl, different images of the milk bowl are produced on the cat's retina. However, these images occur close by in time. This means that different images of the same object are likely to occur close by in time compared to different images of different objects. Also, since motion in the physical world obeys the locality and inertial laws of physics, two consecutive images are more likely to come from the same object than from different objects. This information is sufficient to unambiguously separate out the manifolds. Figure 2.4 shows how temporal information helps us to separate points belonging to different manifolds. Although two different images of the milk-bowl do not come labeled, the fact that they occur close by in time can be used to learn that they are produced by the same object. Thus, implicitly, time can be used as the supervisor to tell which patterns belong together and which patterns do not. [1]

The identity of a manifold can be thought of as an *invariant representation* for an object - different images of the object belong to the same manifold. When an object undergoes different transformations in the world, the identity of its manifold remains the same. Object recognition can be thought of as identifying the object manifold to which an input image belongs.

Note that once these manifolds are learned and separated using temporal information, recognition is easy for single snap-shots. It is only the separation of these manifolds in an unsupervised manner that required temporal information. That said, having temporal information during recognition could only help to improve the accuracy of recognition.

By using time as the supervisor we will be able to separate the manifolds of two objects A and B. However, learning the manifold of object A will not help us in learning the manifold for object B. Mammalian vision is extremely good at exploiting the learning of one object to learn another. In many cases we can generalize to novel views of an object that we have never seen before. Clearly, capturing this aspect of mammalian vision involves more than just unsupervised learning and more than just the use of time as the supervisor.

This is where hierarchy comes in to the picture. Most of the real world objects share the same building blocks. Suppose that our system can learn the invariant representations (i.e., manifolds) of these basic building blocks, and then learn the invariant representations of larger building blocks in terms of the invariant representations of the smaller building blocks. Then different objects can be learned as reconfigurations of the invariant representations of the basic building blocks. By learning one object, we also learn some building blocks for other objects. Moreover, the invariant representations of the building blocks apply equally well to learn a new object. If we follow this way of learning, the representations learned from one object can be used to learn another object.

The above discussion can be distilled into two main points:

[1]A recent experiment by Cox, Meier, Oertelt and DiCarlo lab showed that position invariant object recognition can be broken by changing the temporal statistics of presentation of the objects [18]. This could support the argument that neocortex uses time as a supervisor

- Use temporal information to see whether two different images (patterns) belong to the same object manifold or not. Use this information to form invariant representations

- Do the invariant learning in a hierarchy such that invariant representations of object components are learned first and invariant representations of larger objects are learned in terms of the invariant representations of these components

In addition to the ideas discussed above, there is another important aspect of mammalian vision that is largely ignored in current vision research. This is the sensori-motor nature of vision. In mammals, vision is actively linked to action and many vision problems can be solved only with self-generated actions. For example, our mental model of a milk bowl not only involves the visual part, but also includes the memory of how the visual part will change if we make particular movements. Although we are in complete agreement with the active vision approach, we think that significant (though ultimately limited) progress can be made on the invariant pattern recognition problem by exploiting the temporal and hierarchical aspect of vision. Therefore, we will focus on those aspects in this work.

2.1.2 Hierarchical temporal memory

Based on the discussion above, in this thesis we will develop a new method to attack the vision problem. In general this method can be thought of as a memory system that exploits the hierarchical structure of the visual world. Rather than thinking of the vision problem as a space-only problem, this method also exploits the temporal dimension to learn invariant representations corresponding to sub-objects at every level. Since this method is a memory system that exploits hierarchy and time, we will call it by the name Hierarchical Temporal Memory (HTM).

An HTM system is organized as a network of nodes. Typically this network has a tree shape. Figure 2.5 shows an HTM network with three levels. There are four nodes at level one, 2 nodes at level two and one node at level three. Inputs to the network are fed to the nodes at level one.

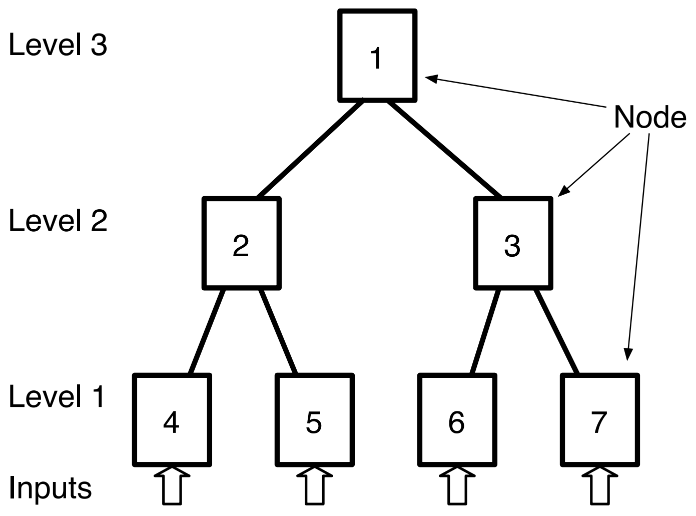


Figure 2.5: A simple HTM network that has 7 nodes arranged in a 3 level hierarchy. The levels are numbered from level one to level three, with level one at the bottom of the hierarchy and level three at the top. Inputs are fed to the nodes at the bottom level of the hierarchy.

We will frequently use the term *hierarchy* to refer to the arrangement of nodes in a network as shown in figure 2.5. The nodes that are closest to the inputs to the network are at the *bottom-level* of the hierarchy and the nodes that are furthest away from the sensory inputs are at the *top-level* of the network. In figure 2.5, the network-inputs go directly to level-1 nodes and hence the level-1 nodes are at the bottom-level. The top-level of the hierarchy has a single node – node-1. Thinking in terms of hierarchical levels helps us identify the relative positions of two nodes with respect to the input to the network. For example, a node that is hierarchically *below* another node is closer to the inputs than the other node. Therefore, as you ascend the hierarchy, you get farther and farther away from the inputs to the network.

The top level (level-3) of the network has a single node. The nodes which are directly connected to a node and below that node in the hierarchy are called the *child nodes* of that node. In the figure, nodes 4 and 5 are child nodes of node 2. Similarly, nodes that are directly connected to and hierarchically above a node are called *parent nodes* of that node. In the tree structure in figure 2.5, every node has at most one parent. Node 1 is the parent of node 2. Typically, inputs are fed to the bottom level nodes in the hierarchy.

The nodes in the HTM network are the basic algorithm and memory modules of the network. All nodes have inputs and outputs and contain algorithms that are similar in nature. Nodes also contain memory. This memory is used to store information about a node's environment. The learning algorithms within a node observe the input space and abstract some characteristics about it to be stored in the nodes memory. In rough words, a node learns invariant representations of its input space.

We will describe the mechanics of HTM using a simplified visual pattern recognition problem. This simplified setting will help us concentrate on the important aspects of HTM algorithms without getting bogged down by complexity. Using these algorithms as the basis, we will understand the generalizations that need to be made scale them to a general setting.

2.1.3 Organization of the rest of this chapter

This chapter is organized in eight sections. Section 2.2 defines the invariant pattern recognition problem and describes the data-set and the HTM network used for the rest of the chapter. Section 2.3 describes the operation of a node and section 2.4 describes how different nodes operate together in a hierarchy. Section 2.5 generalizes the simplified descriptions of the earlier sections to the case of noisy inputs. Section 2.6 examines the recognition performance of a trained network. Section 2.7 discusses related work and section 2.8 concludes the chapter with a summary and discussion.

2.2 The Pictures problem

In this section we will describe *Pictures*, our example problem for the rest of this chapter. Pictures is a simplified visual pattern recognition problem.

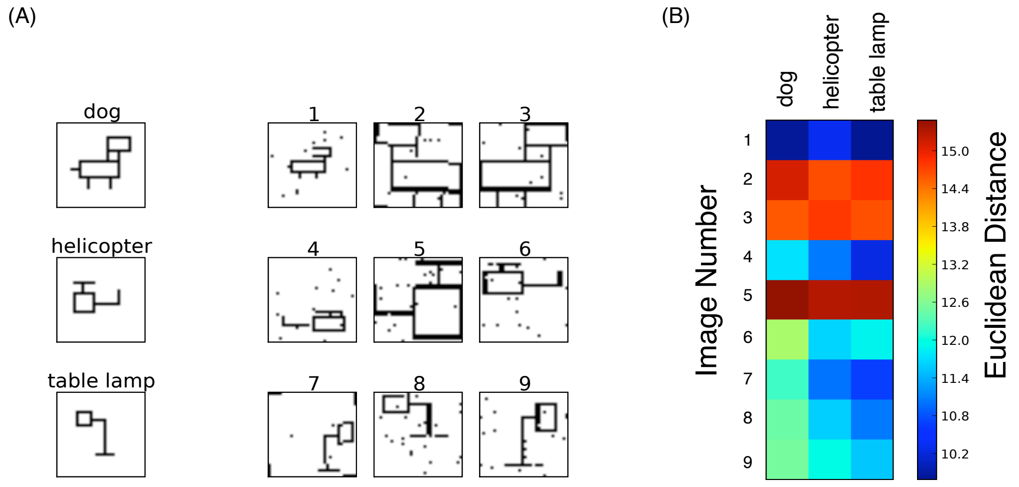


Figure 2.6: (a) The left column of this image shows prototypical examples of three categories - "dog", "helicopter" and "table lamp". On the right hand side are distorted versions of those categories. Humans are good at recognizing the correct categories of those distorted versions. (b) This figure shows the Euclidean distance between the prototype of each category and the distorted images of all categories. For example, row 5 is the distance of distorted image number 5 to the prototypes of "dog", "helicopter" and "table lamp". The closest prototype is the one that is at minimum distance from the image. If we use Euclidean distance as a measure of similarity, most of the images are misclassified. Image 4, which is actually a "helicopter", will be classified as a "table lamp". It can be concluded from this plot that a Euclidean distance measure does not inform us about perceptual similarity.

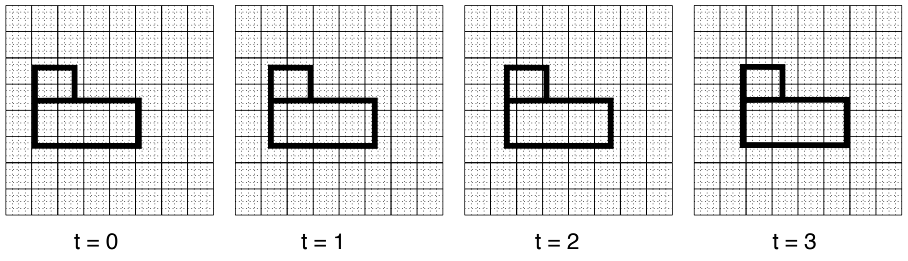


Figure 2.7: This figure shows 4 consecutive frames from a training video. These frames show the line-drawing of a "cat" moving from left to right. Each frame is of size 32 pixels by 32 pixels.

2.2.1 Pictures task

Our goal is to learn invariant representations of a simplified set of binary image categories using an HTM network. Figure 2.6(a) shows some examples of these images. These examples are of size 32 pixels by 32 pixels. The problem of invariant representation can be understood by looking at figure 2.6 (a) and (b). In figure 2.6 (a), on the left side, are prototypical line drawings of a "dog", "helicopter" and "table lamp". In the same figure, on the right hand side, are several deformations of these line drawings. Humans can readily recognize the correct categories of these images despite the different translations, scale changes and deformations. Although the images in the same category appear "similar" to humans, this notion of similarity is hard to quantify in order to be programmed in a computer. A measure of pixel similarity across different images here show that (figure 2.6 (b)) the images that fall within the same category can be more dissimilar in the pixel space compared to the ones that fall within different categories. We will take recognizing image categories under a wide variety of transformations, deformations and noise as our working definition of the invariant recognition problem.

2.2.2 Pictures training data

In the real visual world, transformations like translation, rotation and scaling occur due to relative motion between objects and the viewer. To create the effect of transformations created by continuous motions in our training data set, we need to create movies out of our binary images. These movies need to contain all the transformations that we want the learning system to be invariant to. Movies are created by animating binary images with smooth translations, rotations and scale variations. Figure 2.7 shows 4 frames of a sample movie sequence. In this sequence, the "cat" object moves to the right one pixel at a time. The visual world for our problem consists of such simulated movies of binary line drawings.

2.2.3 Pictures HTM network structure

The HTM network we use to model this visual world is organized in a three-level hierarchy. Figure 2.8 shows how our HTM network interfaces with this visual world. Each frame of the movies of the visual world is presented on a retina of size 32 pixels by 32 pixels. The nodes at level 1 are arranged in an 8x8 grid such that they cover the 32x32 retina without overlap. Each level-1 node receives its input from a 4x4 pixel patch of the retina as shown in figure 2.8. A node at level two receives its inputs from four level-1 nodes.

The effective input area from which a node receives its input is called its *receptive field*. The size of the receptive field for the level-1 nodes in the figure 2.8 is 4x4. The level-2 nodes have a receptive field of size 8x8 because their inputs come from 4 level-1 nodes and hence they indirectly get inputs from an 8x8 patch of the input image. In the kind of hierarchical arrangement shown in figure 2.8, the size of the receptive field of a node increases as we go up in the hierarchy. The single node at the top of the tree covers the entire visual field, by pooling inputs from all of the lower nodes.

2.2.4 Stages of operation

For the ease of description, we think of the HTM network to be operating in two distinct stages - *learning* and *sensing/inference*. During the learning stage, the network

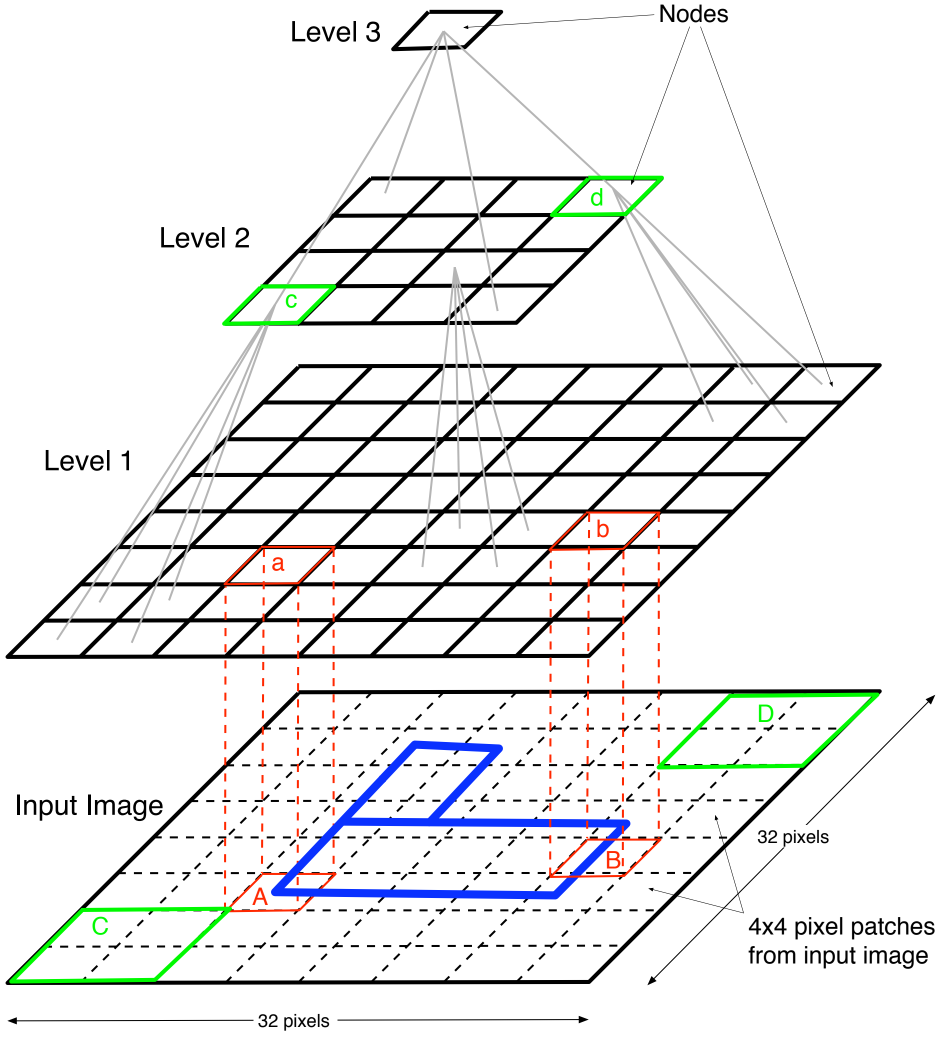


Figure 2.8: Structure of the HTM network for learning invariant representations for the binary images world. Level one of the network has 64 nodes arranged in an 8x8 grid. Each node is shown as a square. The input to a level-2 node comes from the outputs of 4 level-1 nodes as marked. The outputs from all the level-2 nodes go to a single level-3 node. Input is fed to the nodes at level-1. The input image, a frame from the training videos, is of size 32 pixels by 32 pixels. This image is divided into adjoining patches of 4 pixels by 4 pixels as shown. Each level-1 node's input corresponds to one such 4x4 patch. Two nodes at level-1 are marked (a) and (b). Squares marked (A) and (B) in the input image represent the respective receptive fields of these nodes. Each level-1 node 'sees' only a small portion of the input image. The squares marked by (C) and (D) correspond to the receptive fields of nodes at level-2 marked (c) and (d).

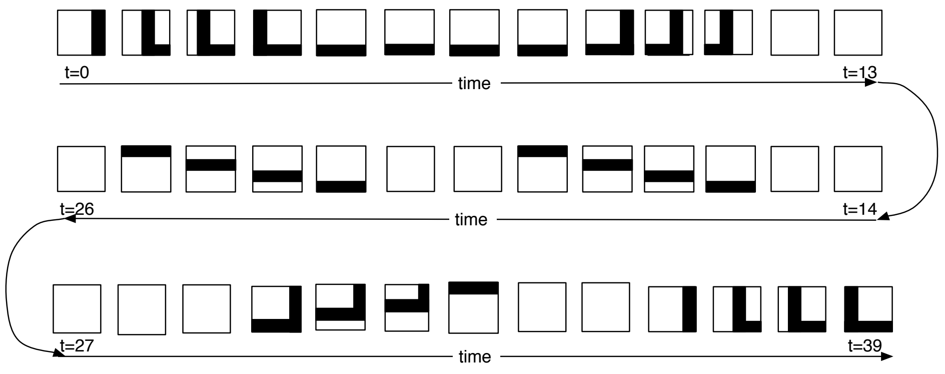


Figure 2.9: This figure shows a segment of a long input sequence received by a level-1 node. Each input pattern is a vector of 16 components and is shown as a 4x4 pixel array for easier interpretation. Short sub-sequences of patterns can repeat within this long sequence. For example, the 4-long sequence of a horizontal line moving up occurs twice within the shown sequence of inputs.

is exposed to movies as we described above and the nodes in the network form representations for the world using the learning algorithms we describe below. Once all the nodes in the network have finished learning, the whole network can be switched to the sensing mode. During the sensing/inference mode the HTM network recognizes the category of an input image. Though the learning stage involves exposing the network to movies from the visual world, we will restrict the sensing stage in our discussion to static image recognition. During this stage, we present an image (a single snap shot) and ask the network to identify the category of that image. This information will be read out at the top of the network.

All nodes in the network use the same learning and sensing/inference algorithms. Hence understanding the operation of the network involves two things – understanding the operations within a node and understanding how a node operates with other nodes in the hierarchy. The former is described in section 2.3 and the latter in section 2.4.

2.3 Operation of a node

We describe the operation of a node with respect to a level-1 node in figure 2.8. This will help us anchor our discussion using concrete examples.

The network in figure 2.8 is fed with movies of object motions. The input to the network at any time is a visual image – a frame from the training movie. The field of view of a level-1 node is a 4x4 pixel array from the visual image that is inputted to the network. Visual objects move under the field of view of this node. Objects are typically larger than the field of view. Therefore, the node "sees" only portions of the objects. When objects move, they create different patterns in the field of view of the node. Each pattern is a 16-pixel vector. Figure 2.9 shows a portion of an input sequence to a level-1 node.

The node has two phases of operations. During the *learning* phase, the node observes its input patterns and builds its internal representations. The node does not produce any outputs during this phase. A node that has finished its learning process is in the *sensing/inference* phase. During sensing/inference, the node produces an output for every input pattern. We now describe the details of the learning and sensing phases. We first describe this process for the case where the input patterns are noise-free.

2.3.1 Learning in a node

The input to the node during the learning process is a long sequence of patterns. The node operates on the input patterns one at a time. For every input pattern, the node does three operations:

1. Memorization of patterns
2. Learning transition probabilities
3. Temporal grouping

The details of these operations are described below.

Memorization of input patterns

The node has a memory that stores patterns within the field of view. This memory permanently stores patterns and gives each pattern a distinct label, i.e., a pattern number. Patterns that occur in the input field of view of a node are compared against the patterns that are already stored in the memory. If an identical pattern is not in the memory, then the input pattern is added to the memory and given a distinct pattern number. The number that is given to a pattern is arbitrary and does not reflect any property of the pattern.

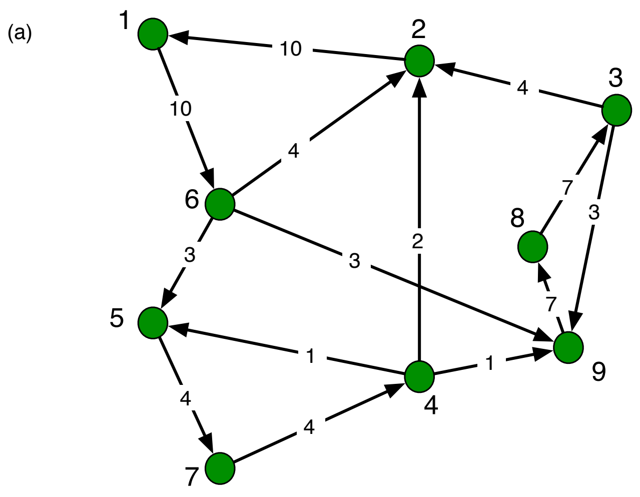
We assume that patterns are stored in a node as the rows of a matrix. We use C to represent this pattern memory matrix. Each row of the C matrix is a different pattern. The individual patterns will be referred to by c_1, c_2 etc., depending on the rows in which those patterns are stored.

Learning transition probabilities

The node constructs and maintains a Markov graph (figure 2.10). The vertices of this Markov graph correspond to the stored patterns. Each vertex is given the label of the pattern that it represents. As new patterns are added to the memory contents, new vertices are added to the Markov graph.

The link between two vertices is used to represent the number of transition events between the patterns corresponding to those vertices. When input pattern i is followed by input pattern j for the first time, a link is introduced between the vertices i and j and the number of transition events on that link is set to one. The number of transition counts on the link from i to j is then incremented whenever a transition from pattern i to pattern j is observed at the input of the node.

The Markov graph is normalized so that the links can become estimates of transition probabilities. Normalization is done by dividing the number of transition events on the outgoing links of each vertex by the total number of transition events from that vertex. This is done for all the vertices to obtain a normalized Markov graph. The total transition probability from each node of a normalized Markov graph should add to 1. The normalized Markov graph is updated continually to reflect the new



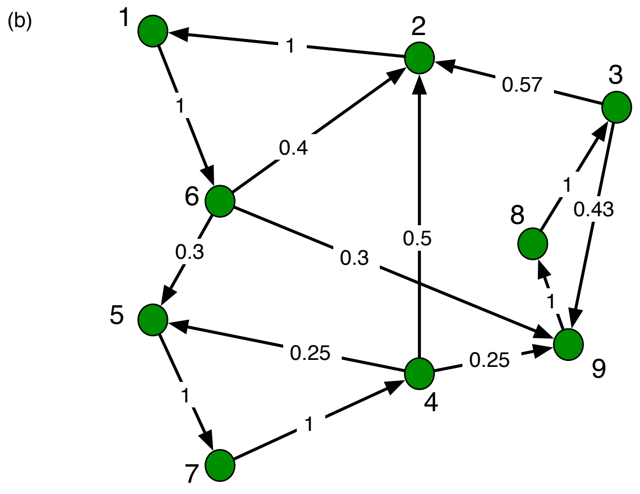


Figure 2.10: (a) Shows an un-normalized Markov graph. The vertices of the graph are numbered to correspond to patterns stored in the node. The arrangement of the vertices in space does not have any significance. The links are labeled using the number of transition events from one vertex to another. (b) The normalized Markov graph derived from (a). Estimated probabilities of transitions from one pattern to another are indicated using the labels on the links connecting the corresponding patterns.

probability estimates.

Temporal grouping

The node partitions the set of vertices of the Markov graph into a set of *temporal groups*. Each temporal group is a subset of the set of vertices of the Markov graph. The partitioning is done such that the vertices of the same temporal group are highly likely to follow one another.

The node uses a well-known clustering method called Agglomerative Hierarchical Clustering (AHC) [54] to form the temporal groups. The AHC algorithm takes a set of pattern labels and their pair-wise similarity measurements as inputs, and produces clusters of pattern labels such that patterns in the same cluster are similar to each other.

The probability of transition between two patterns is used as a measure of the similarity between those patterns for the AHC algorithm. Clustering based on this similarity metric puts patterns that are likely to follow one another into the same cluster. Patterns that are unlikely to follow each other fall into different clusters. Since each cluster is a set of patterns that are very likely to follow each other in time, we call each cluster by the name *temporal group*.

AHC starts with all the stored patterns in separate clusters and then recursively merges clusters with greatest similarity. This obtains a tree like structure called dendrogram with a single cluster (which contains all patterns) at the top of the tree and the individual patterns at the bottom (each pattern in its own cluster). The desired clustering for temporal grouping lie somewhere in between the bottom and the top of the dendrogram and can be obtained by defining a suitable criterion. For example, one criterion could be to cut the tree at a level where the size of the largest cluster does not exceed a particular value. The node has a design parameter that sets the maximum number of clusters or temporal groups of the node. We obtain the desired temporal groups by selecting a level of the dendrogram that gives the number of temporal groups closest to and less than the configured maximum allowed number of temporal groups.

We now illustrate the temporal grouping process on a simple input sequence to

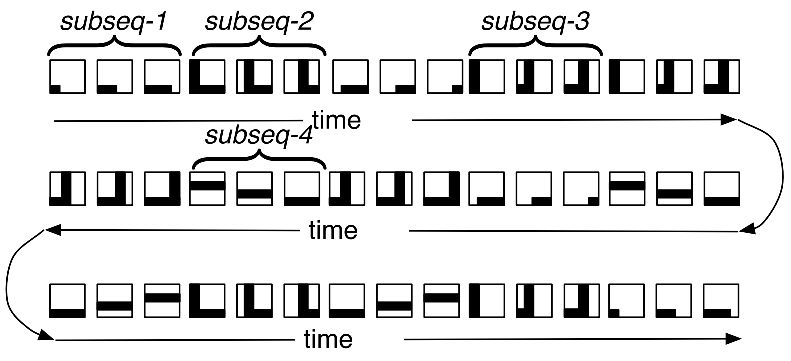


Figure 2.11: Part of the input sequence for the illustrative experiment described in the section on temporal grouping. The long input sequence for this experiment is constructed by switching randomly, with equal probabilities, between four three-long sub-sequences. The sub-sequences are marked using the label "subseq".

the node. The input sequence, shown in figure 2.11, is constructed by switching randomly and repeatedly (with uniform probabilities) between four sub-sequences of length three each as marked in the figure. The input sequence has 12 unique patterns. These were memorized by the node and given labels $c_1, c_2, \cdots, c_{12}$. The normalized Markov graph, obtained after exposing the node to a very long input sequence, is shown in figure 2.12. A dendrogram corresponding to this Markov graph is shown in figure 2.13. Figures 2.14 and 2.15 show the temporal groups obtained by cutting the dendrogram at two different levels. Figures 2.16 and 2.17 summarize the operations of the node during the learning phase.

The temporal groups within a node are updated as the markov transition probabilities are updated. The above steps – recording of patterns in memory, forming the normalized Markov graph, and temporal grouping – are performed for every time step during the learning phase. The learning process can be stopped when the temporal groups have sufficiently stabilized.

Another possibility would be to stage the three learning operations within the node. For example, it is possible to first finish the memorization of all the training patterns before forming the Markov graph. Similarly, the temporal grouping could be done just once, after the normalized Markov graph probabilities have stabilized.

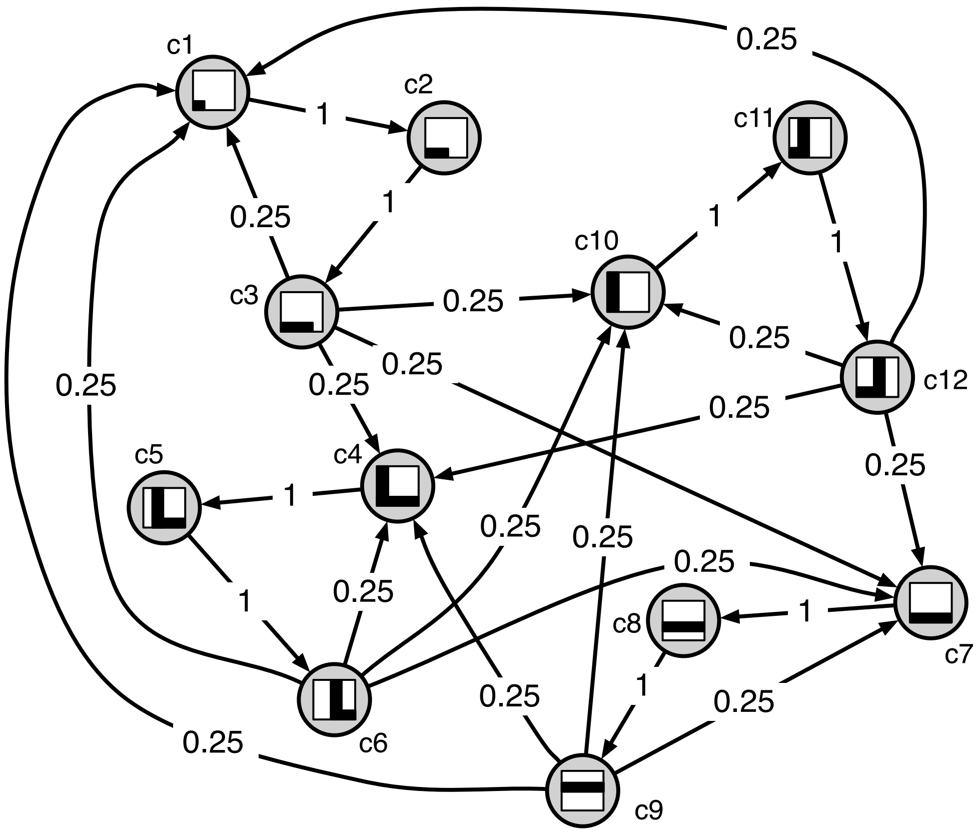


Figure 2.12: The normalized Markov graph that was learned for the input sequence described in figure 2.11. The pattern corresponding to each vertex of the Markov graph is shown within that vertex. The vertices are labeled c_1 to c_{12}. Note that this labeling is arbitrary and does not reflect any property of the pattern.

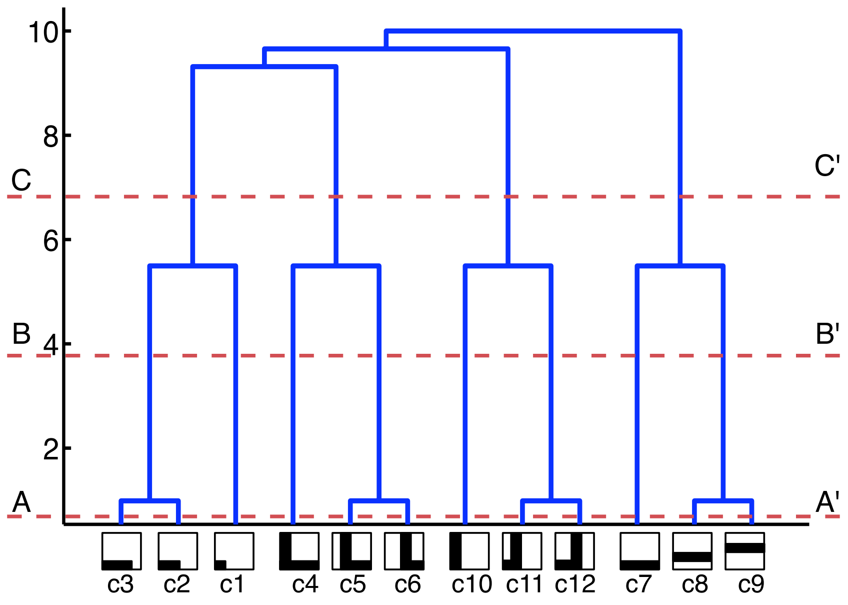


Figure 2.13: This figure shows the dendrogram that is constructed for the normalized Markov graph in figure 2.12. This dendrogram is constructed using the "linkage" routine from MATLAB. The inverses of the probability values are used as the distance between patterns.The Y axis of the dendrogram is a measure of the 'distance' between clusters. For example, the link from $c1$ to $[c3, c2]$ indicates the average 'distance' between $c1$ and the cluster $[c3, c2]$. These distances, calculated by the MATLAB linkage routine, cannot be directly interpreted in terms of the Markov graph edges. Several variants of linkage calculations exist. The obtained temporal groups are not sensitive to the particular method that is chosen. The dotted lines A–A', B–B' and C–C', correspond to cutting the dendrogram at different levels. Infinite distances (corresponding to probability of transition equal to zero) are replaced with the value 10 to make it feasible to display the dendrogram.

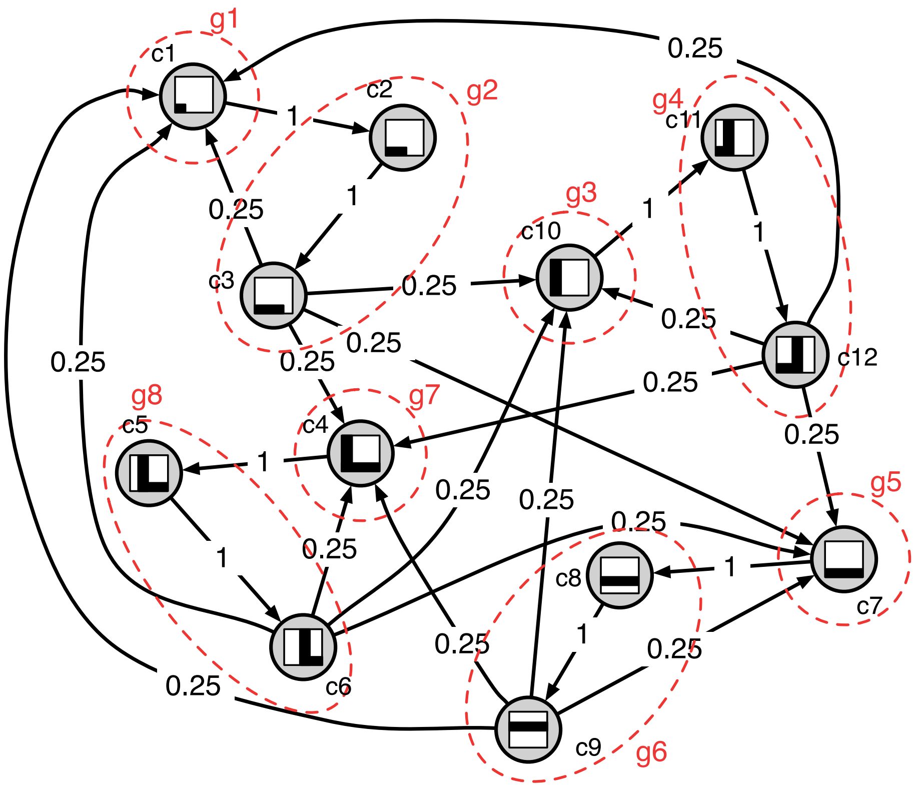


Figure 2.14: This figure shows the temporal groups obtained by cutting the dendrogram in figure 2.13 at the level marked B–B'. Eight temporal groups are obtained. These temporal groups are marked using dotted circles and given labels $g_1, \cdots, g_8$.

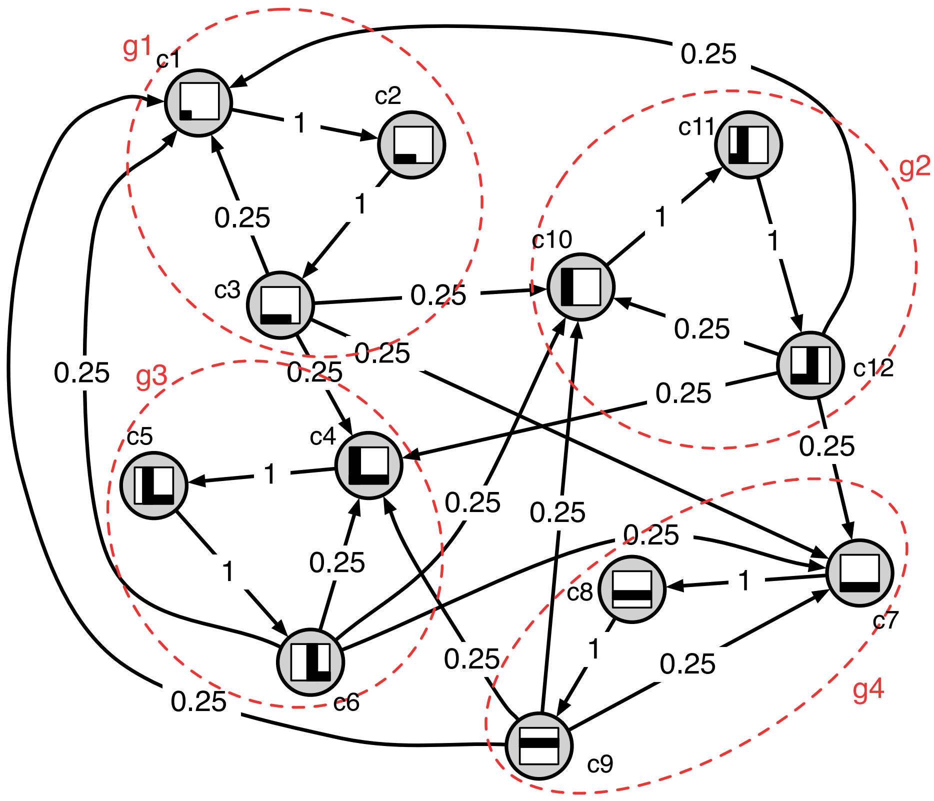


Figure 2.15: This figure shows the temporal groups obtained by cutting the dendrogram in figure 2.13 at the level marked C–C'. Four temporal groups are obtained. These temporal groups are marked using dotted circles and given labels $g_1, \cdots, g_4$.

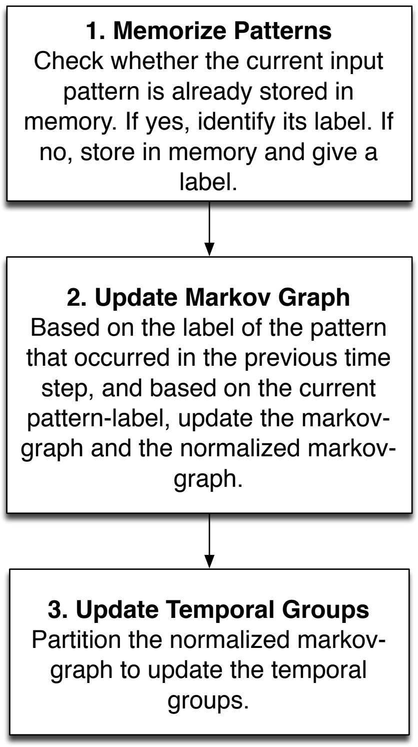

1. Memorize Patterns
Check whether the current input pattern is already stored in memory. If yes, identify its label. If no, store in memory and give a label.
2. Update Markov Graph
Based on the label of the pattern that occurred in the previous time step, and based on the current pattern-label, update the markov-graph and the normalized markov-graph.
3. Update Temporal Groups
Partition the normalized markov-graph to update the temporal groups.

Figure 2.16: Summary of the operations that a node performs during learning. These operations are performed for every time step during learning.

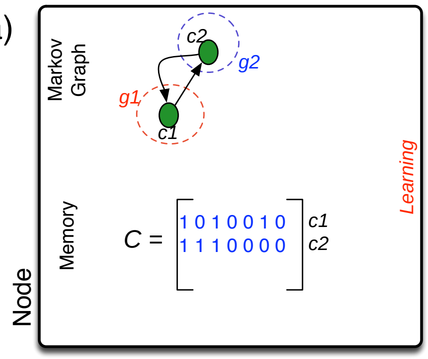
The second method is preferred as it involves less computation. This method will be used for the rest of this chapter.

2.3.2 Sensing/Inference in a node

A node that has completed its learning process can now be used for sensing/inference. The characteristics of the input to the node in sensing are identical to those during learning. Objects move under the field of view of the node and the node "sees" portions of those objects. The resulting patterns are the inputs to the node.

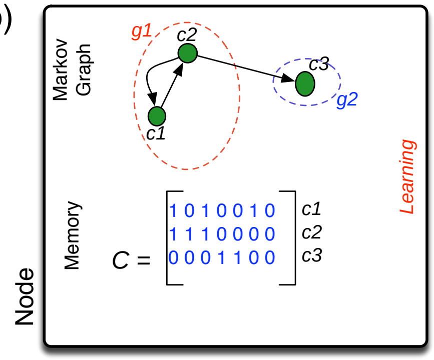
A node in sensing/inference produces an output for every input pattern. Although it is possible for the node to use a sequence of patterns to produce an output, we make the simplifying assumption that the outputs are produced based on instantaneous

(a)



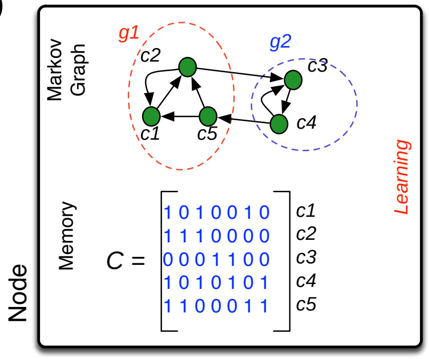
This figure shows an intermediate stage of the learning phase in a node. The inputs to this node are patterns of length seven. The node has stored two patterns in its memory. Although the patterns are shown as binary vectors, this need not be the case. The markov-graph within the node has two vertices corresponding to the two stored patterns. The temporal grouping step has put the two vertices in two different groups, *g1* and *g2*

(b)



At this point, the node has added another pattern to the memory making the total number of patterns equal to three. A new vertex, corresponding to the new pattern, is added to the markov-graph. The temporal grouping step has produced two temporal groups, with the 2 vertices in one group and one vertex in the other.

(c)



In this figure, the node is at the end of its learning process. The node has stored 5 patterns and the markov-graph has 5 vertices. The temporal grouping method produced 2 temporal groups. These temporal groups are marked using dotted-circles.

Figure 2.17: Illustration of the learning process in a node. Figures (a),(b) and (c) show different stages during the learning of a node. The node shown here is a fictitious one used only for illustrative purposes. The patterns and the Markov graph of this node do not correspond to any real problem situation.

inputs. One result of this assumption is that the Markov graph is not used during the sensing phase. It can be discarded once the temporal groups within the node are finalized.

For every input pattern, the node produces an output vector that indicates the degree of membership of the input pattern in each of its temporal groups. The current input pattern generally does not perfectly match any of the patterns stored in memory. Its closeness to every pattern stored in memory needs to be determined. Let d_i be the distance of the i^{th} stored pattern from the input pattern. The larger this distance, the smaller the match between the input pattern and the stored pattern. We assume that the probability that an input pattern matches a stored pattern falls off as a Gaussian function of the Euclidean distance. With this, the probability that the input pattern matches the i^{th} stored pattern can be calculated as being proportional to $e^{-d_i^2/\sigma^2}$, where σ is a parameter of the node. Calculating this for every stored pattern gives the closeness of the current input pattern to all the vertices of the Markov graph.

Degree of membership of the input pattern in each temporal group is determined by the maximum of its closeness to each of the vertices within the temporal group. This results in a vector of length equal to the number of temporal groups, with each component of the vector indicating the degree of membership of the input pattern in the corresponding temporal group. This vector is then normalized to sum to unity. The normalized memberships become estimates of probability of membership in each temporal group. The normalized degree of memberships is the output of the node. The output is a histogram giving estimates of probability of membership of the current input pattern in each temporal group of the node.

2.3.3 Experimental results of learning and sensing in a node

We now describe some experimental results of learning and sensing in a level-1 node – the node marked (a) in figure 2.8. The network in figure 2.8 is exposed to 300000 frames from the object-motion movies described earlier. Each frame of this movie is a 32-pixels by 32-pixels image that contains the picture of a whole object. The level-1 node's visual field, 4 pixels by 4 pixels, is much smaller than the whole image. Hence

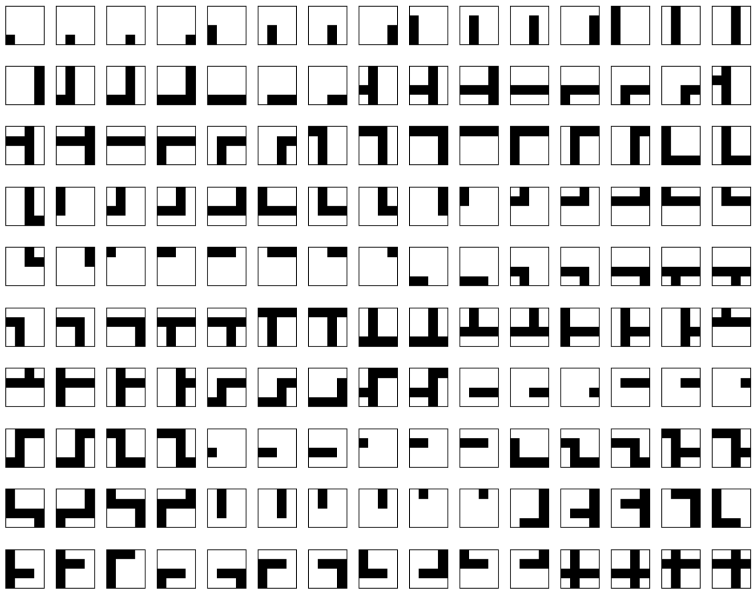


Figure 2.18: Patterns memorized by a level-1 node in figure 2.8 when exposed to videos of line drawing images. Each pattern is of length 16 and is shown as a 4x4 pixel array for easier interpretation. The node memorized 150 patterns.

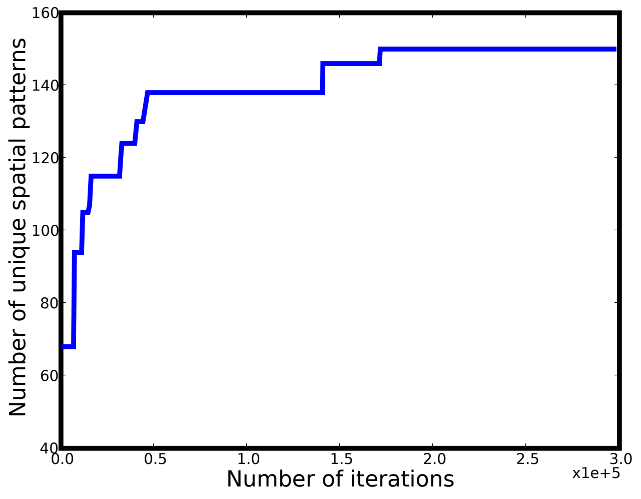


Figure 2.19: The number of patterns stored in a level-1 node as a function of the number of training images.

the node "sees" only portions of the objects.

Figure 2.18 shows the patterns memorized by the node during the training phase. From the 300000 frames, 150 unique patterns are memorized. Each pattern is a vector of 16 components and is shown as a 4x4 image array for easier interpretation. Figure 2.19 shows a plot of the number of unique patterns in the memory as a function of the number of frames of the training movie.

A Markov graph with 150 vertices is obtained at the end of the training period. It is not practical to show the whole Markov graph in a figure. In figure 2.20, we show a section of this Markov graph. Partitioning this Markov graph resulted in 58 temporal groups. Examples of the resultant temporal groups are shown in figure 2.21.

Figure 2.22 illustrates the sensing process in a node. Shown are the outputs of the node for 3 time steps during which a vertically translating horizontal line was the input sequence to the node. The top panel of the figure shows the calculated degree of match between the input patterns and the stored patterns for these three time steps. As the input patterns change, different stored patterns become the 'winners'. The bottom-panel of the figure (e to g) shows the output of the node for these time

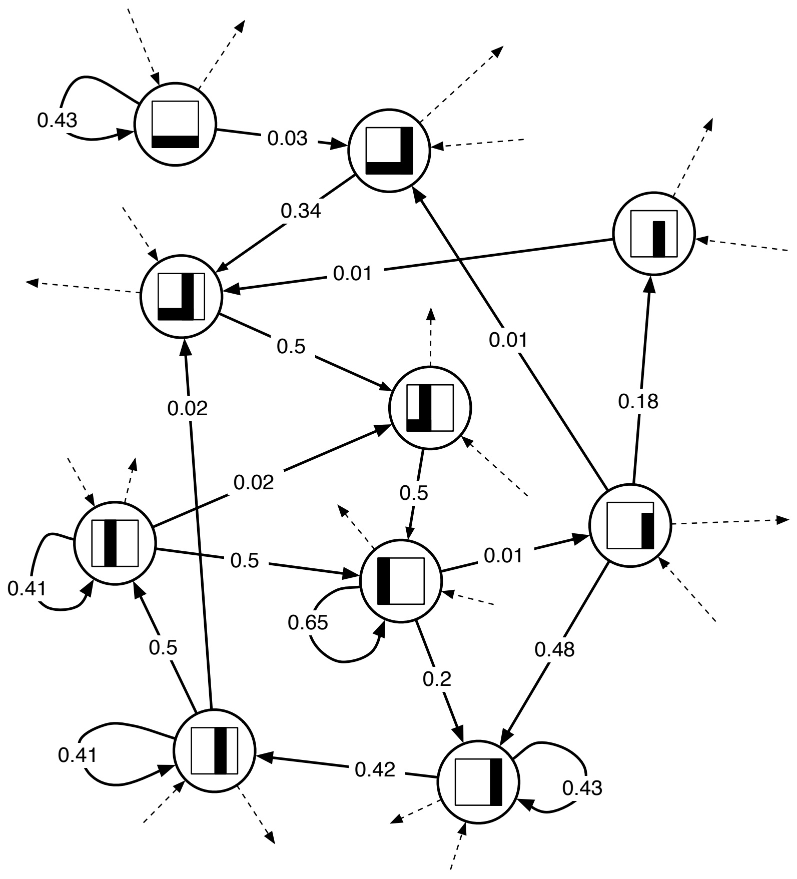


Figure 2.20: This figure shows a portion of the normalized Markov graph learned in a level-1 node. The original Markov graph has 150 vertices corresponding to the 150 stored patterns. The portion shown here has ten vertices. The vertices are shown as circles. For ease of interpretation, the patterns corresponding to these vertices are shown within the circles as 4x4 pixel arrays. Vertices corresponding to vertical lines have significant self-transition probabilities because vertical translations of a vertical line result in the same vertical line. In contrast, the corner-vertices have zero self-transition probabilities because a vertical or horizontal translation of a corner results in a different corner.

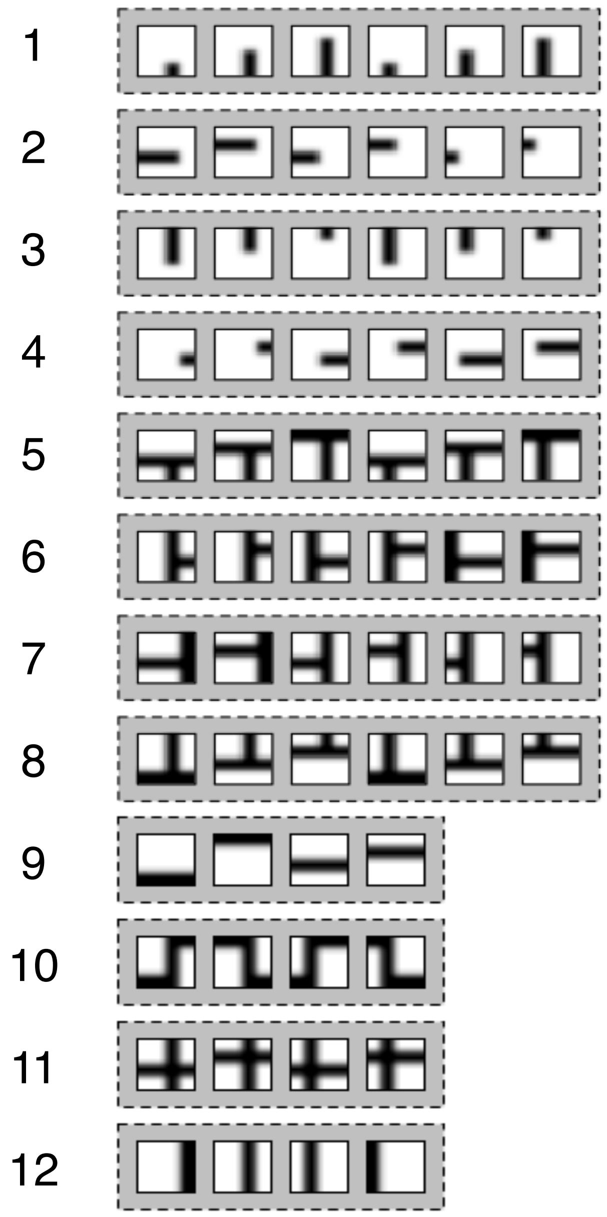


Figure 2.21: This figure shows 12 of the 58 temporal groups learned in a level-1 node. The patterns of the same temporal group are shown within a gray box. Each row is a different temporal group.

steps. The outputs have the same 'winner' – temporal group 9 – for the three time steps. The output is 'invariant' because the vertical translations of the horizontal line inputs are in the same temporal group.

A node in sensing has the following property. If the patterns of a temporal group of the node occur sequentially at the input of the node, then the output of the node will be invariant during that sequence. The sequential occurrence of patterns of the same temporal group will happen frequently if the temporal statistics of the inputs remain the same as during learning. Under this condition, the output of the node will, on average, change slower than the input to the node.

2.3.4 Temporal groups are zeroth order Markov chains

The temporal groups in a node are obtained by partitioning its Markov graph into sub-graphs. However, during the sensing/inference phase, each temporal group is considered as a set of patterns without regards to their within-temporal-group transition probabilities. In essence, each sub-graph is considered as a zeroth order Markov chain by discarding the sequential information stored within the sub-graph. In the general case, an HTM node can have variable order Markov chains that make use of sequential information during sensing/inference. For the rest of this thesis, we will use the name Markov chains when sequential information is important and the name temporal groups when it is not.

2.4 Operation of nodes in a hierarchy

The nodes in a network are trained one level at a time. The level-1 nodes are trained first. When the level-1 nodes are in training, they do not produce any outputs. At this time, the nodes at level two and higher are switched off. Once the level-1 nodes are trained, they are put into sensing/inference mode. In this mode, they produce outputs. These outputs are the inputs for the level-2 nodes. The level-2 nodes are trained next. This process is repeated in the hierarchy to train all the nodes in the network.

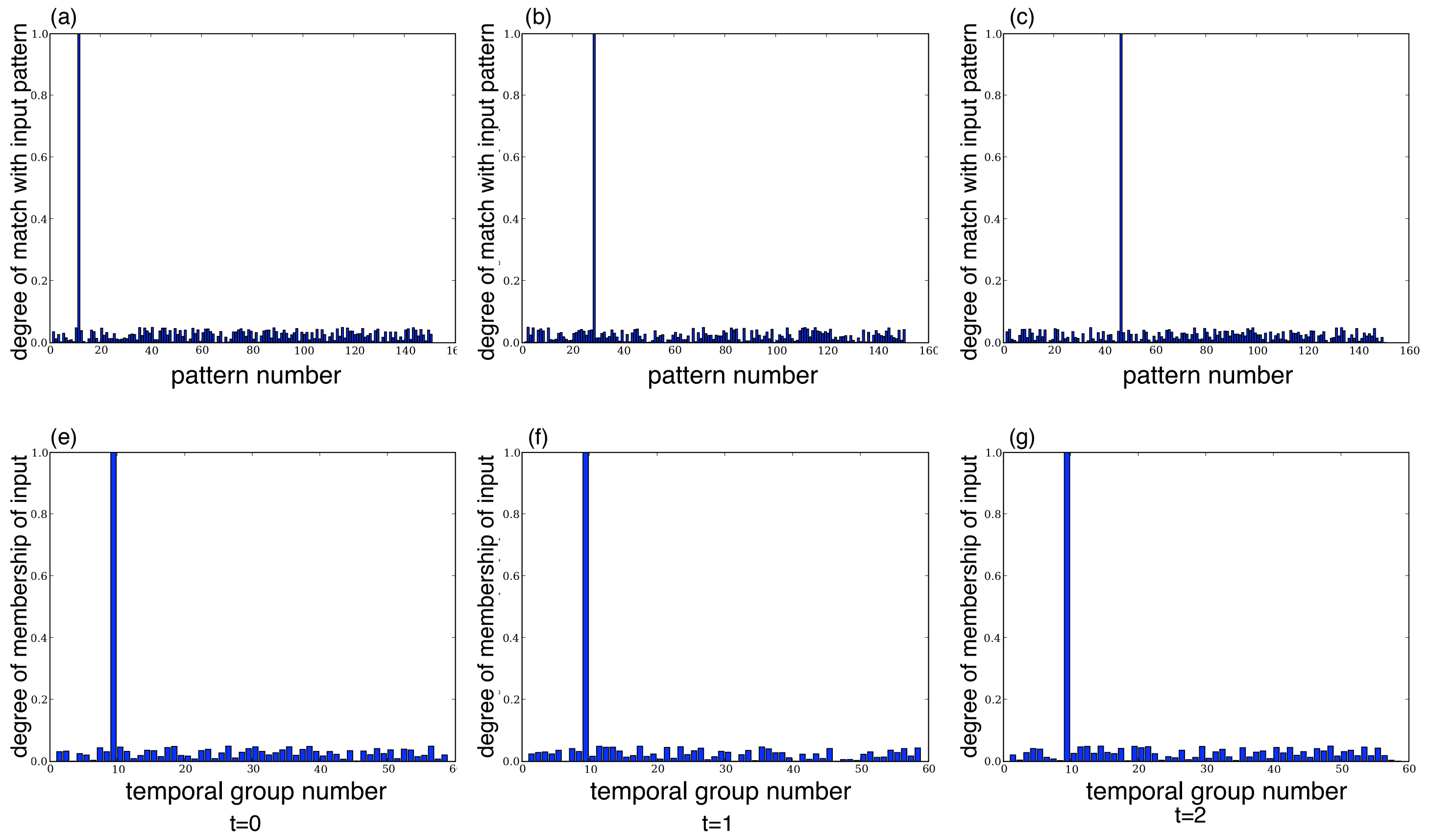


Figure 2.22: This figure illustrates the quantities computed during the sensing process of a node. The three columns correspond to three time steps. A vertically translating horizontal line was the input to the node during this time. The top panels of the figure (a to c) show the degree of match between the input pattern and the stored patterns as calculated in the node. Figures (e) to (g) show the corresponding node outputs.

2.4.1 Example two-level network

We use the simplified two-level hierarchical network shown in figure 2.23(a) to illustrate the operation of nodes in a hierarchy. This hierarchy has two nodes at level 1 and one node at level 2. The input to this network is an image of size 4 pixels by 8 pixels. One-half of this image, a pattern of size 4 pixels by 4 pixels, is fed to one node at level 1, and the other half is fed to the other node at level 1. The level-1 nodes are identical and have 12 patterns and four temporal groups. Both child nodes have identical numbering of patterns and temporal groups, although this need not be the case. The nodes at level 1 are in sensing/inference mode. The node at level 2 is in its learning phase.

2.4.2 Inputs for higher-level nodes

When a node is at the first level of the hierarchy, its input comes directly from the sensor, possibly after some pre-processing. The input to a node at a higher-level is the concatenation of the outputs of the nodes that are directly connected to it from a lower level. In figure 2.23, each level-1 node has an output of length 4. The input to the level-2 node is the concatenation of these outputs – a pattern of length 8 as shown in figure 2.23(a).

In the following subsection, we assume that the patterns that are presented to the level-1 nodes exactly match the patterns presented to them during their learning phase. This means that a level-1 node output, for any input pattern, will have exactly one '1' in the position corresponding to the temporal group to which the input pattern belongs, and zeros everywhere else . Therefore, the input patterns to the level-2 node in figure 2.23 has two '1's – one for each child's winner temporal group. The assumption of clean input patterns is relaxed in section 2.5.

2.4.3 Memorization of patterns at the level-2 node

We now describe the learning process in the level-2 node as a sequence of inputs is presented to the level-1 nodes. Figure 2.23 shows the two-level example network along with its inputs for times $t = 0$, $t = 1$ and $t = 2$. The sequence of inputs corresponds

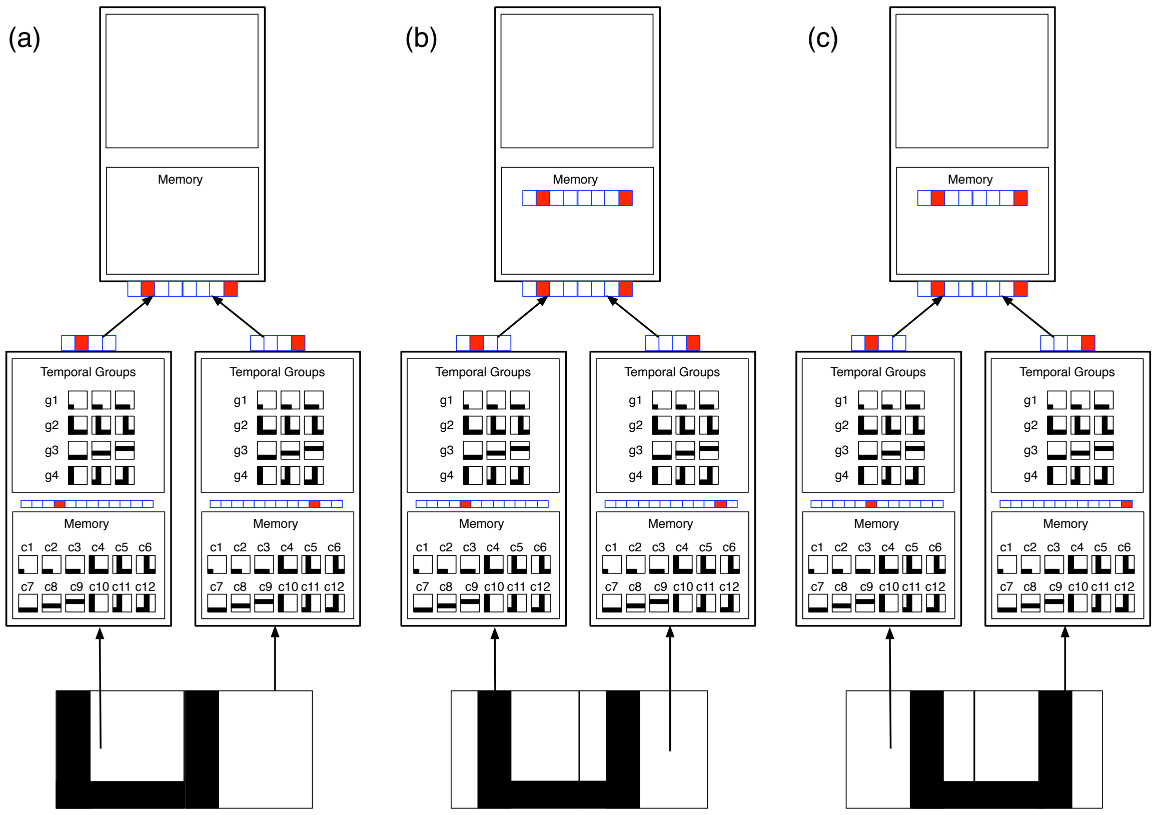


Figure 2.23: This figure illustrates how nodes operate in a hierarchy. We show a two-level network and its associated inputs for three time steps. This network is constructed for illustrative purposes and is not the result of a real learning process. The outputs of the nodes are represented using an array of rectangles. The number of rectangles in the array corresponds to the length of the output vector. Filled rectangles represent '1's and empty rectangles represent '0's. See text for details.

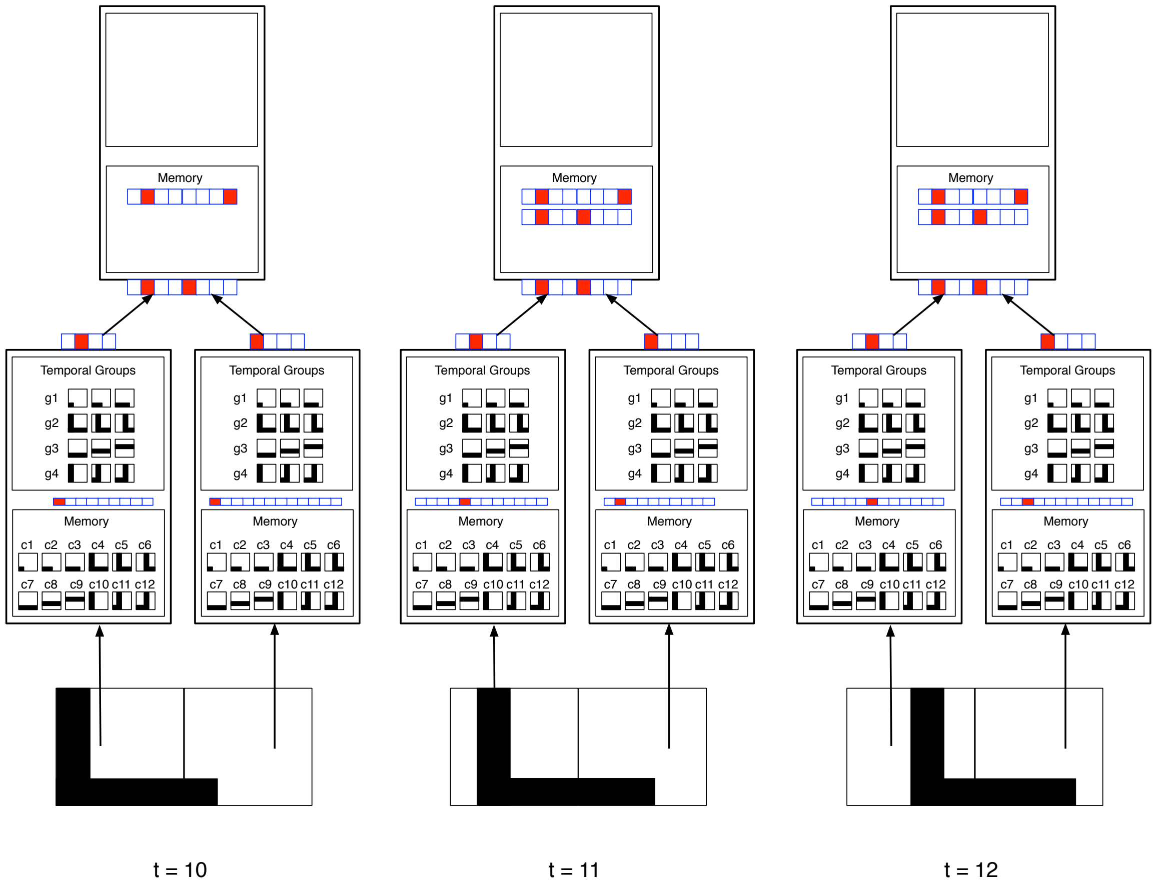


Figure 2.24: This figure uses the same setting as of figure 2.23 except for the input sequence. The input sequence is an "L" moving to the right. The level-2 node has already learned one pattern before the beginning of this input sequence. The new input sequence introduced one additional pattern to the level-2 node.

to a “U” shape moving in the combined visual field of these two nodes. No level-1 node sees the whole U pattern. The sequence of patterns seen by the left level-1 node, as the U moves in the combined field of view, can be described as left-corner, left-corner, left-corner. On the other hand, the right level-1 node sees the sequence vertical-line, right-corner, right-corner.

For the right-bottom node in Figure 2.23, the outputs for times $t = 0$ to time $t = 2$ are $[0, 0, 0, 1], [0, 0, 0, 1], [0, 0, 0, 1]$ respectively. The outputs do not change because the input patterns c_{10}, c_{11} and c_{12} are in the same temporal group (g_4) in the right level-1 node. Similarly, the left level-1 node outputs the vector $[0, 1, 0, 0]$ for these three time steps.

The input to the level-2 node, at any time, is the concatenation of the outputs from its children. For the above three time steps, this input is $[0, 1, 0, 0, 0, 0, 0, 1]$. This is obtained as the concatenation of $[0, 1, 0, 0]$ from the left level-1 node and $[0, 0, 0, 1]$ from the right level-1 node. While the network is exposed to 3 different images of U, the input to the level-2 node remains the same. This input is an invariant representation for the three different translations of the U.

The pattern memorization process at the level-2 node is identical to that of the level-1 nodes. A new pattern that is not already in the node’s memory is added to the memory and given a new number. During the sequence of inputs from $t = 0$ to $t = 2$, one new pattern was added to the memory of the level-2 node.

Figure 2.24 shows another sequence of patterns and the corresponding inputs/outputs for time period from $t = 10$ to $t = 12$.(The particular time indexing that we choose here is not important.) During this period an “L” moves across the combined visual field of the level-1 nodes. Note however, that the left level-1 node sees the same sequence of movement as it saw for the rightward motion of the U pattern. Therefore, its outputs for time steps $t = 10$ to $t = 12$ are identical to that for time steps $t = 0$ to $t = 2$ steps. The output of the right level-1 node during this time period is $[1, 0, 0, 0]$.

As we observed in the case for the U pattern, different shifts of the L produce the same input pattern to the level-2 node. Therefore, the input to the level-2 node is invariant to those translations of the L. However, the input pattern to the level-2 node for the L-sequence is different from that for the U-sequence. That is, the patterns at level 2 have maintained the selectivity between U and L while remaining invariant to their translations.

The learning of transition probabilities and temporal grouping in higher-level nodes are identical to that of level-1 nodes. This process can be repeated in the hierarchy to learn a multi-level network.

2.4.4 Patterns at higher-level nodes represent “coincidences”

We saw that a pattern of a higher-level node is the concatenation of the outputs from its children. An output from a child has a ‘1’ in the position corresponding to its currently active temporal group. Therefore, each pattern stored in a higher-level node has as many ones as the number of children of that node. The pattern [0100 0001] in the example we considered above describes that the 2nd temporal group of the left-child and the 4th temporal group of the right child occurred at the same time. In general, patterns in higher-level nodes represent particular co-occurrences or coincidences of the temporal groups of their children. Therefore, we will use the terms *coincidence*, *coincidence-pattern*, and *pattern* interchangeably for the rest of this thesis.

2.5 Dealing with noise and ambiguity

The descriptions in the previous sections assumed that the inputs to the network are not noisy. Dealing with noisy patterns require modifications to the learning and inference steps.

2.5.1 Learning with noisy inputs

The memorization process that was described earlier is not suitable for the case of noisy inputs because the node will have to store a very large number of patterns. Level-1 nodes and higher-level nodes require different modifications to deal with noise. We describe level-1 modifications first and then the higher-level modifications.

Learning with noise at level-1 nodes

When the inputs are noisy, an additional stage of clustering is performed on the inputs. This pre-clustering algorithm assumes that the patterns inputted to the node belong to a vector space. (See figure 2.25). The Euclidean distance between different patterns in this vector space is used to find clusters of patterns. The clusters are formed such that patterns close-by in Euclidean distance fall into the same cluster.

The node uses an online k-means clustering algorithm to cluster the input patterns into k clusters, where the number of clusters k is a design parameter of the node. The k-means algorithm is initialized with k randomly chosen input vectors as the initial cluster centers. This algorithm then assigns a new input vector to its closest cluster center and moves that cluster center towards the new input pattern by a small amount that is a fraction of the Euclidean distance between the input pattern and the closest cluster center. The amounts by which cluster centers are moved in response to new input patterns is reduced with time. This results in stabilization of the cluster centers.

The node obtains k cluster centers by applying online k-means algorithm to the input patterns. The cluster centers are recorded in permanent memory and given distinct labels. The rest of the learning process proceeds as above using the cluster centers as the memorized patterns. The vertices of the Markov graph now correspond to the cluster centers and they are labeled using the cluster center labels. The cluster center corresponding to an input pattern is the closest cluster to the input pattern in the Euclidean distance sense.

Learning with noise at higher-level nodes

The inputs to higher-level nodes are distributions that reflect the ambiguity in the input to the network. During learning, the higher-level nodes handle this ambiguity by sparsifying the inputs by taking a ‘winner’ from each child. The positions corresponding to the winners are set to one and the other positions are set to zero. With this operation, the patterns seen during learning match the ideal patterns that are seen for the noise-less case.

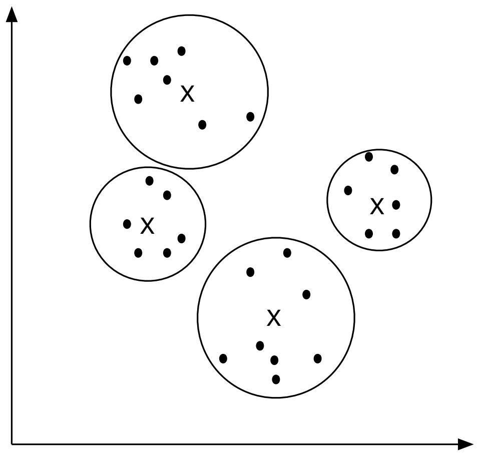


Figure 2.25: Idealized picture of clustering the noisy input patterns. The noise-free underlying patterns are represented as points in a vector space (represented by crosses). Noise causes different variations of these ideal patterns to appear at the input of the node (represented by dots). If the noise-level is small enough, approximations to the ideal patterns can be found by clustering the inputs. The cluster centers are used as approximations of the true patterns.

2.5.2 Sensing/Inference – general case

In our description of the sensing/inference process for the level-1 node in section 2.3.2, we assumed that the input patterns, in general, do not match the stored patterns. Therefore, the same sensing process can be used for the noisy case for all level-1 nodes. The output of a level-1 node reflects the noise in the input. Instead of signaling the membership of the input pattern in a group with complete certainty, the output is now a distribution that reflects the degree of membership of the noisy input pattern in each temporal group of the node.

The calculation of the outputs of higher-level nodes is slightly different from the computations done in a level-1 node. This difference can be understood with respect to the level-2 node from section 2.4. Consider the pattern shown in figure 2.23 within the level-2 node. The '1's in this pattern correspond to group-2 from the left child-node and group-4 from the right child-node. If the inputs from the children nodes are probability distributions, the probability that this particular coincidence-pattern is active would be proportional to the quantity obtained by multiplying together the second value from the left child-node's output and the fourth value from the right child-node's output.

In general, let the i^{th} coincidence-pattern be represented as $c_i = \left[m_1^i, \cdots, m_{N_c}^i\right]$ where the m_k^i's represent the positions where the coincidence-pattern has non-zero entries. N_c is the number of child nodes of this node. The probability that the i^{th} coincidence-pattern is active is calculated as

$$P(c_i) = \gamma \prod_{k=1}^{N_c} input(m_k^i) \tag{2.1}$$

where γ is a proportionality constant that normalizes the probability distribution.

Once the probability distribution over the stored patterns is obtained, the rest of the computations are identical to the level-1 node inference computations. From the probability distribution over the patterns, the probability that a particular group is active is calculated as the probability of the pattern that has the maximum probability among all the patterns belonging to that group.

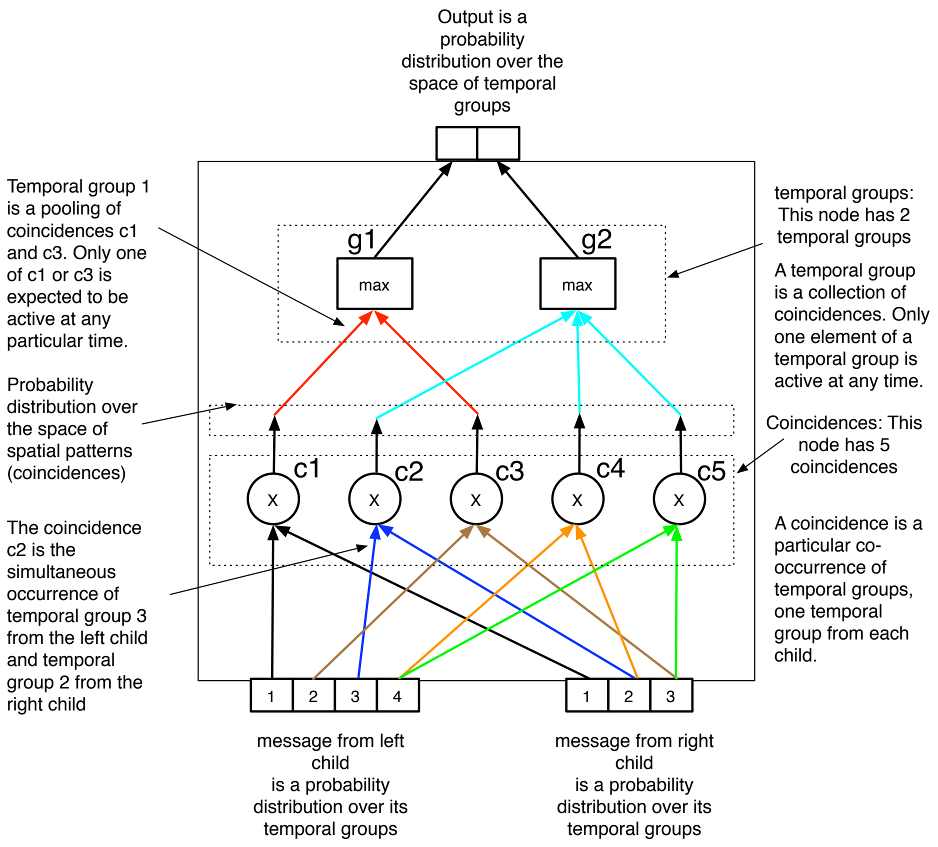

Output is a probability distribution over the space of temporal groups
Temporal group 1 is a pooling of coincidences c1 and c3. Only one of c1 or c3 is expected to be active at any particular time.
temporal groups: This node has 2 temporal groups
A temporal group is a collection of coincidences. Only one element of a temporal group is active at any time.
g1
g2
max
max
Probability distribution over the space of spatial patterns (coincidences)
Coincidences: This node has 5 coincidences
c1
c2
c3
c4
c5
x
The coincidence c2 is the simultaneous occurrence of temporal group 3 from the left child and temporal group 2 from the right child
A coincidence is a particular co-occurrence of temporal groups, one temporal group from each child.
1
2
3
4
message from left child is a probability distribution over its temporal groups
message from right child is a probability distribution over its temporal groups

Figure 2.26: This figure illustrates the structure of a higher-level node and the general inference mechanism. Each coincidence-pattern can be thought of as being represented by a neuron-like unit. The connections of this neuron to the inputs describe the coincidence-pattern represented by that neuron. For example, c_2 is has connections to the 3^{rd} location of the message from the left child and to the 2^{nd} location of the right child. This means that c_2 represents the coincidence of group three of the left child-node and group-2 of the right child-node. The coincidence-neurons do a multiplication of their inputs to obtain the outputs. Similarly, each temporal group can be thought of as a neuron. A temporal group is described by the connections it makes to the coincidence (or pattern) neurons. In the figure shown, temporal group g_1 consists of coincidences c_1 and c_3, while temporal group g_2 consists of coincidences c_2, c_4 and c_5. The output of a temporal group neuron is obtained by taking the max over the inputs.

This form of propagating the uncertainties provides better recognition compared to making winner-take-all decisions locally. Ambiguous information can get resolved higher up in the hierarchy because only some configurations of the temporal groups of lower levels are valid coincidence-patterns at higher levels. Bayesian belief propagation [75] is used to propagate the uncertainties in the network to obtain the optimum recognition performance. The derivation of belief propagation equations for HTM nodes is discussed in appendix A. Belief propagation techniques can also be used to propagate evidence down-stream in the hierarchy. This backward propagation is used to disambiguate and reconstruct information at the lower levels of the hierarchy. In the next section, we will consider the recognition performance of a fully trained hierarchy. Examples for feedback propagation and reconstruction using the same network can be found in appendix B.

2.6 Recognition performance of a fully trained hierarchy

Using the procedures described above, we trained a hierarchy of nodes. This hierarchy has 64 nodes in the bottom level, four nodes in the middle level and one node at the top level. The nodes at level 1 are arranged in an 8x8 grid and each level-1 node received a 4x4 pixel patch of the input. The input to a level-2 node came from 16 level-1 nodes organized in a 4x4 grid. The four level-2 nodes were connected to the one node at the top.

2.6.1 Training movies

The network is trained on 48 categories of line drawing images as shown in figure 2.27. The training set included several scaled examples of each category.

Levels 1 and 2 of the system are trained using videos consisting of vertical and horizontal translations of the training images. As the level-1 nodes do not distinguish between scaling and translation sequences, the level-1 temporal groups required for

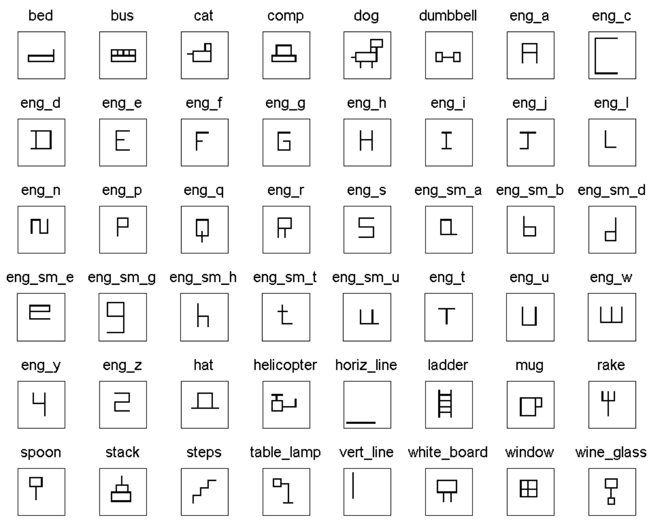


Figure 2.27: Training image examples: An example from each category is shown. Several examples at different scales were used to generate the training movies.

representing scale changes are learned from translation sequences. Therefore, continuous scale variations are not included in the training videos.

The top level was trained using supervision using a naive Bayes classifier [25, 20] attached to the pattern-probablity outputs of the level-3 node. During this process, the training images are presented at every position and the network is supervised at the classifier attached to the level-3 node. Training images are presented exhaustively at every position.

2.6.2 Testing images

Testing images are generated using a program that takes in a training image and generates several test images by arbitrarily scaling along x and y axes, adding noise, deleting pixels and adding small rotations.[2] Since the scales chosen are arbitrary, it is unlikely that an exact replica of a training image would be presented during testing.

[2]The training set did not have rotations in them

The system is tested for recognition accuracy by presenting these images at the bottom level and reading out the category from the classifier attached to the top-level node. The recognized category is set to be the one that produced the maximum probability at the classifier. This is then compared against the actual category index for the test image.

2.6.3 Recognition results

The recognition accuracy on training images is 99.73 percent. We tested the system on approximately 9000 test images. The system achieved 57 percent recognition accuracy on the test image set. The chance level is 2 percent. A nearest-neighbor classifier applied to the same problem obtained only 28 percent recognition accuracy. Figure 2.28 shows several test images that were recognized correctly. These test images are arranged in columns according to their correct category.

To understand the nature of the recognition performance, we gradually distorted an image in one category to an image in another category and recorded the strengths (proportional to probabilities) recorded at the top level of the network during the process. Figure 2.29 illustrates the result of this experiment.

2.6.4 How the system generalizes to translations, scaling and distortion

Suppose we take a trained network and try to teach it a new pattern – pattern 'A' as shown in figure 2.30. This network is trained on patterns other than A during its initial training phase. We now expose the network to pattern A at one position and supervise at the top of the network. The position of A to which the network is exposed is shown in figure 2.30 in black.

Figure 2.30 (a) also shows three horizontal translations of this A in gray, red and green. Notice the patterns within the input fields of the level-1 nodes, created due to translations of A. In the highlighted input field in figure 2.30 (a), these patterns correspond to four translations of a corner. If these patterns belong to the same temporal group in the node that is attached to this input field, then the output of

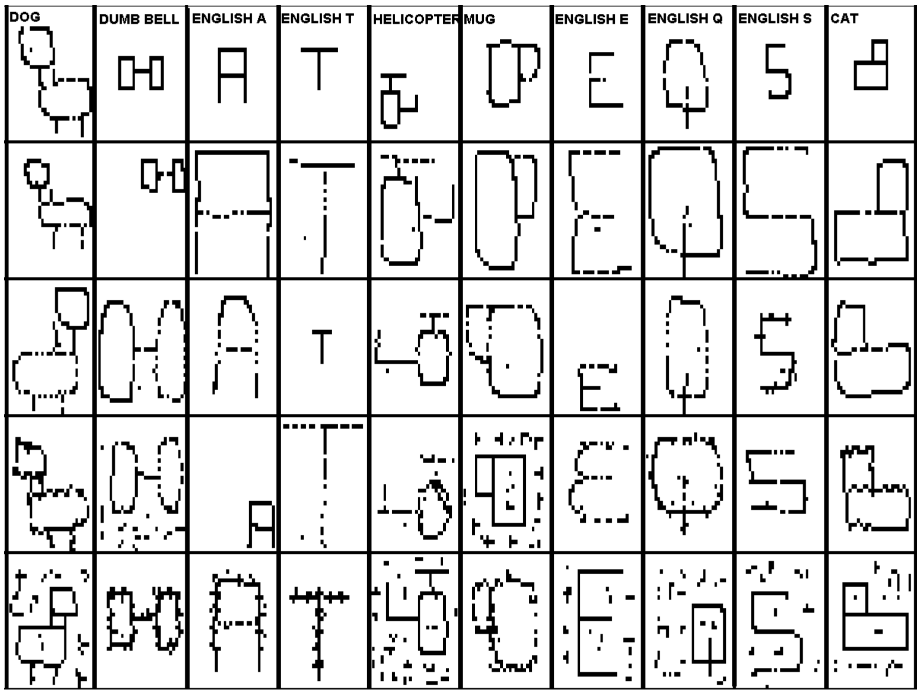


Figure 2.28: This figure shows examples of test images that were recognized correctly. Each column shows examples of a particular category. Each test image is of size 32 pixels by 32 pixels. Only 10 out of 48 categories are shown here. It can be seen from the test images that the system can recognize these images despite large amounts of translation, scale differences, distortions and noise.

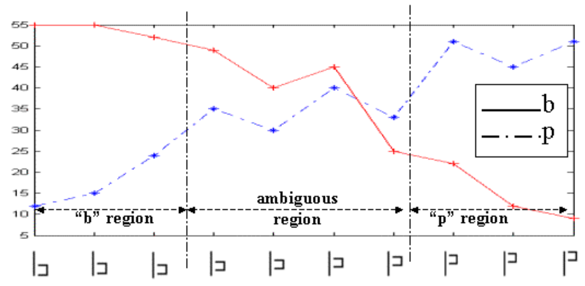


Figure 2.29: Variation of perceptual strength with pattern distortion : For this test we gradually distorted a test pattern of one category (category b) into a pattern of another category (category p). None of the test patterns were identical to the ones used for training the b and p categories. Plotted along the Y axis is the score obtained at L3 for categories b and p when patterns along the X axis were shown to the system. In the region marked as b *region* the pattern was identified as belonging to category b and similarly for the p *region*. In the ambiguous region the system classified the pattern as neither b nor p but (erroneously) identified it as various other categories.

that node will remain invariant during these translations of A. If this happens in all other nodes, the patterns seen by higher-level nodes will be invariant to local translations of A. The temporal groups required for this are similar to the learned temporal groups in section 2.3.1.

Figure 2.30 (c) shows the canonical A and several scaled versions of it. Note that the temporal groups required at the lowest level are similar to the ones required for translation invariance. In the input field of a level-1 node, both those transformations

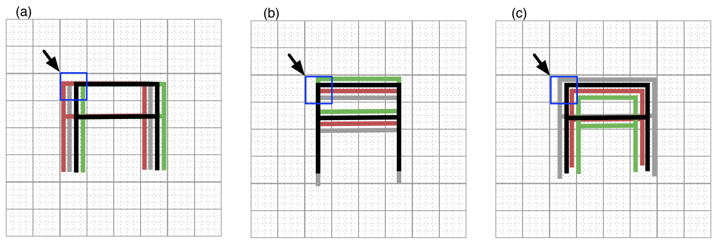


Figure 2.30: This figure is used to explain the generalization performance of our system. See text for details. (a) shows several horizontal translations of an "A" (b) shows several vertical translations of an "A" and (c) shows several scaled versions of an "A"

produce the same effect. Therefore, some of the temporal groups required for scale invariance can be learned by exposure to translating patterns.

Invariance across larger transformations is achieved by the combined effect of the temporal groups at the lower levels and higher levels. When the network learns a new object that is composed of parts of familiar objects, the invariances that are learned for the familiar objects automatically transfer to the new objects. This happens because the objects are learned in a hierarchy; new objects are learned as reconfigurations of the parts of familiar objects.

2.6.5 Generalization performance

The above discussion tells us that the generalization for a new object will depend on its "degree of similarity" with the familiar objects, at various levels of the hierarchy. Since this notion is hard to quantify, we use an experiment to illustrate this.

We trained the network on 48 categories as described before. We then exposed the network to three additional categories of images at 10 randomly chosen positions for each category while supervising at the top for the appropriate category. We tested the network for generalization by querying it for exposures to these images at different locations. The degree of success of the network in recognizing the images at other locations depended on the complexity of the novel image. Simple images generalized to more positions compared to complex images. Figure 2.31 shows the generalization performance of the network.

2.7 Related work

Several researchers have built pattern recognition systems with multiple levels. Neocognitron [32], convolutional neural networks [58], the HMAX model [84], and the fragments hierarchy from Ullman [100] are examples of models that have network architectures and pooling operations similar to the HTM model described here. However, the pooling mechanisms in these models were hardcoded and not learned. This is discussed in detail in section 3.7.

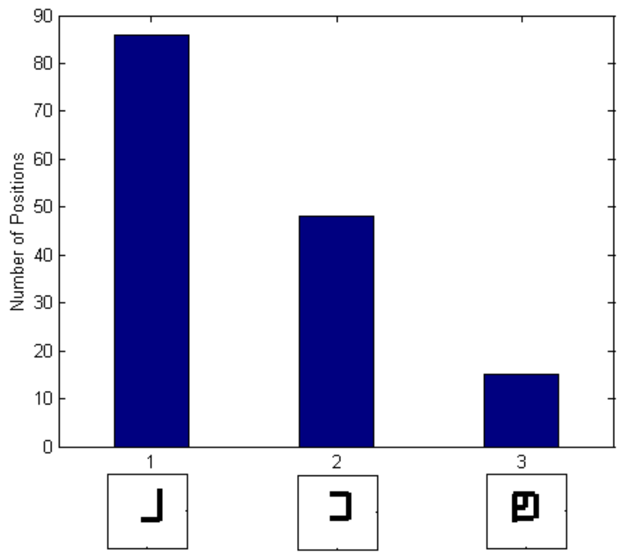


Figure 2.31: In this experiment we showed the system snapshots of 3 novel images at 10 randomly chosen positions. What is plotted is the number of positions to which the system generalized for each of these novel images (shown along the X axis). It can be seen that the generalization performance goes down with increasing complexity of the novel pattern.

Boltzmann machines [47], and deep belief networks [48] provide another set examples of networks with multiple levels of representations. However, these networks demonstrate limited translation and scale invariance since they do not have a pooling mechanism similar to the models described above. Moreover, the learning mechanisms in these models do not exploit the temporal nature of vision.

Visnet [91] and Slow Feature Analysis [109] are examples of networks that use temporal slowness as a principle for building invariant representations. These networks were built as proofs of concept for temporal learning; hence the algorithms in these models are unlikely to scale to larger problems. Moreover, these models do not use the formalism of Bayesian belief propagation for inference and do not have the capability of using feedback propagation for reconstruction of input images. Ullman and Bart [101] use image fragments, extended with motion tracking, as the features for invariant recognition. However, their work does not use a hierarchical structure.

2.8 Summary and bigger picture

Our attempt to achieve invariant pattern recognition considered the visual world as a spatio-temporal process. Our method involved creating a spatio-temporal abstraction hierarchy for the visual world. This abstraction hierarchy was created as a network of nodes where each node used essentially the same learning algorithm. Each node in the network contained patterns and temporal groups. We observed that a pattern in a higher-level node corresponds to the simultaneous occurrence of temporal groups in its child-nodes. The nodes higher up in the hierarchy have larger receptive fields – they account for larger amounts of space compared to nodes in lower levels. Since each node creating invariant representations for the temporal process that it is exposed to, the temporal variations become slower as we ascend in the hierarchy. The combined effect is that the nodes at the higher level (through the nodes at the lower level) end up representing larger spatial and temporal scales compared to the lower-level nodes. A "cause" that is represented at a higher level is influenced by a larger spatial extent, and is persistent for a larger duration of time compared to the lower level causes. Figure 2.32 gives an illustration of these concepts.

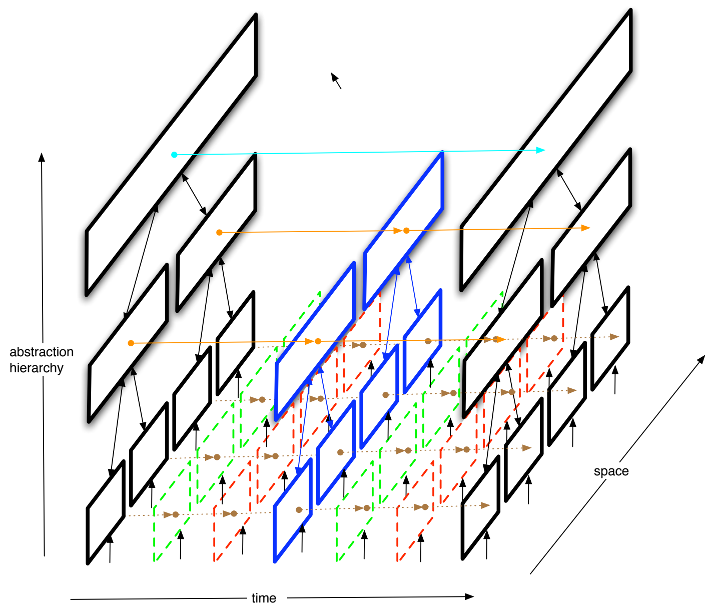


Figure 2.32: This figure shows the concept of a spatial and temporal abstraction hierarchy. The higher levels abstract larger areas of space and larger durations of time. The higher levels change their states slowly compared to the lower levels.

Instead of thinking of this as a recognition hierarchy, we can invert this hierarchy and think of it as the process by which the world generates images. The visual world can be thought of as a "generative model" that uses a hierarchical structure to construct the images. This model operates in space and time. When a particular cause is active at a higher level of the hierarchy, it causes multiple sub-causes to unfold in time simultaneously at various spatial locations. These sub-causes in turn act as causes to unfold further sub-causes in time at finer spatial scales. As we descend down in the world hierarchy, we see finer spatial and temporal scales. The causes that are active at the higher level restrict the dynamics of the "causes" at lower levels. The dynamics of the causes at lower level in turn restrict the flow of the causes in higher levels. This kind of image generation would generate several different images for a particular cause that is active at the top of the hierarchy. In the next chapter we analyze such spatio-temporal hierarchical systems from the point of view of learning and generalization.

Chapter 3

Towards an Understanding of Hierarchical Learning and Generalization

3.1 Introduction

The study of hierarchical models has a rich history and interest in them has waxed and waned in the past. Hierarchical architectures have recently regained attention in the machine learning literature [6, 48]. Many questions remain unanswered about the hierarchical nature of learning. Although hierarchical learning systems are known to work well in practice, computational learning theory does not answer why hierarchies are important for learning. Hierarchical systems are built routinely and are known to reduce sample complexity in practice. However, there is no theoretical explanation for this and no fundamental understanding of the parameters involved [78].

In this chapter we study the generalization properties of HTM networks by generating patterns from them. A *generative model* is specified to enable the generation of patterns from HTM networks. Analogous to the standard practice in machine learning where synthetic observations from a generative model are used to understand the characteristics of the recognition model, the HTM generative model is used primarily as a tool to analyze and characterize the generalization properties of HTMs.

Characterization of generalization in the HTM network is important because not all data domains and modeling techniques directly benefit from a hierarchical structure. For example, if the class structure for patterns in a particular domain can be defined in terms of nearest Euclidean neighbor relationships, then there is no generalization advantage in modeling those patterns using a hierarchy. Similarly, if there is no temporal structure in the data, application of an HTM to that data need not give any generalization advantage. The main goal of this chapter is to identify the characteristics of the data to which HTM models can be applied with successful generalization. This is accomplished by the analysis of the synthetic data generated from an HTM generative model.

Another advantage of understanding the source of generalization in HTMs is that it can be used to explain why some of the existing hierarchical models generalize. Neocognitrons [31], convolutional neural networks [58], and the HMAX model [84] of the visual cortex are examples of successful feed-forward hierarchical models of visual pattern recognition. However, the generalization characteristics of these models are not well understood. We show that these models have a mapping to the HTM model. In essence, these models are exploiting the same assumptions as specified in our generative model by hard-coding the effect of temporal learning. The hard-coded invariant features in these models can be learned using time as a supervisor. Therefore, the learning principles used in HTMs can be applied to problems where such hard-coding is not possible, in addition to the problems that are already tackled by these models.

The multi-scale properties of the generated data also lead us to draw some speculative connections to the perceived multi-scale structure of many physical, biological and information systems. Herbert Simon has argued that hierarchical-temporal organization could be a general property of physical and biological systems [89]. He conjectured that learning machines will need to replicate this hierarchical structure. We find tentative parallels between the organization of the visual cortex and the organization of some physical systems in the world.

The rest of this chapter is organized as follows. Section 3.2 describes how patterns are generated from HTMs. In section 3.3, we analyze the properties of these patterns.

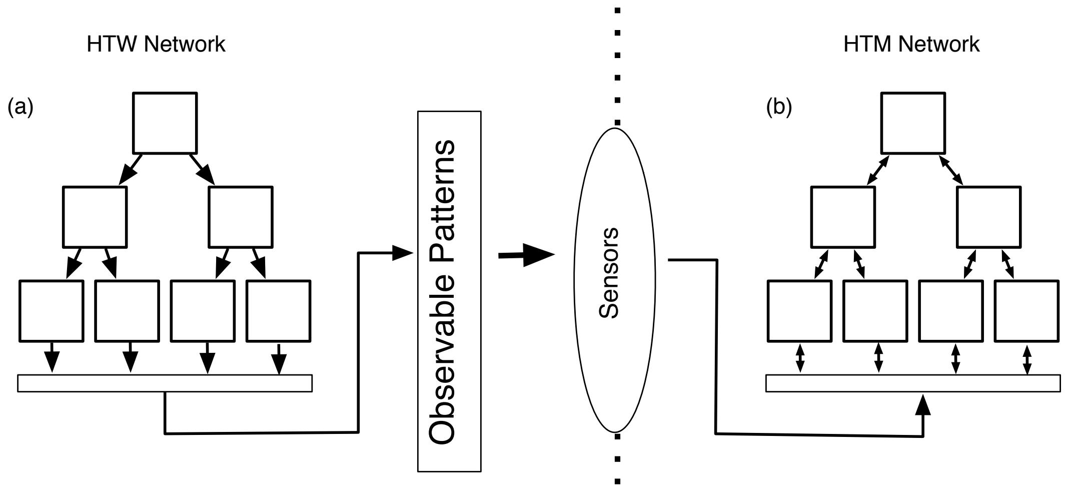


Figure 3.1: Patterns generated from an HTW network are used to analyze the generalization properties of an HTM network

In section 3.4, we describe the learning problems that are associated with hierarchical temporal data and in section 3.5 we describe some learning methods. In section 3.7, we show how convolutional neural networks and similar models can be considered as special cases of HTMs. The generalization properties of HTMs are analyzed in section 3.8. In section 3.9, we discuss related work. We conclude in section 3.10 by drawing some speculative connections between the organization of some systems in the world and the organization of the neocortex.

3.2 Generating patterns from HTMs

In the previous chapter, HTMs were introduced primarily as a recognition model. In this section we describe how patterns are generated from an abstractly defined HTM network. An HTM network used in the generative mode will be called an HTW network because the patterns generated from this network constitute a Hierarchical Temporal World (and also because 'W' is the reverse of 'M').

There are numerous advantages in having a generative model for HTMs. Since we have control over the generation of patterns, we can vary the parameters of the generation process to understand how the properties of generated patterns vary based on these parameters. Having a generative model also helps to answer questions about learning models that are not well matched to the generation model. For example, what would be the result of learning a four-level HTW generated patterns with a three-level HTM network? Having a mathematical model for pattern generation also helps us understand the source of generalization in recognition models.

3.2.1 HTW network

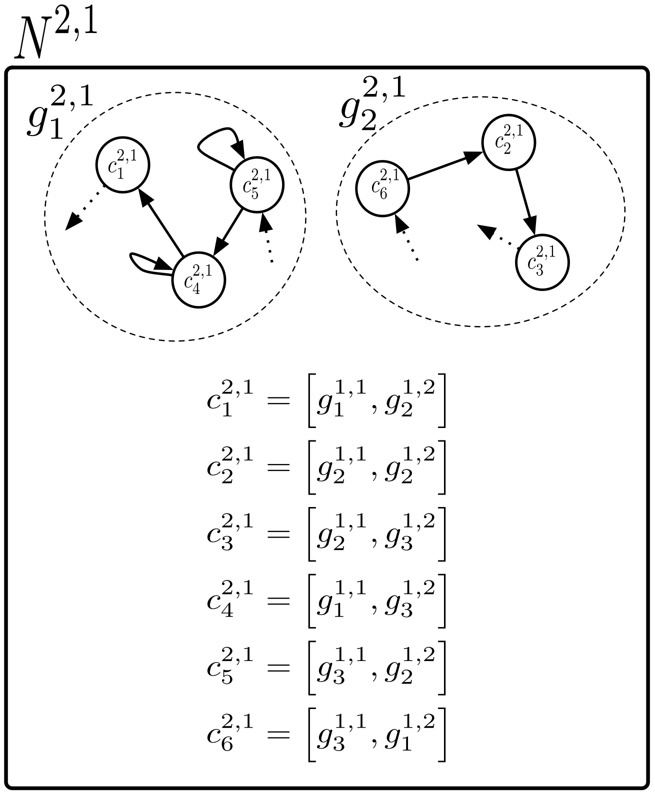
An HTW model is organized as a network of nodes as shown in figure 3.1(a). The outputs of the level-1 nodes constitute the observable patterns generated by the network. All the nodes in this network operate in the same way. Patterns are generated by the combined operation of the nodes in the hierarchy. Since all nodes work in an identical manner, we understand the pattern generation mechanism by first studying the mechanism in a node and then understanding how nodes work together in a hierarchy.

3.2.2 Structure of a node

A node in an HTW network consists of the following:

- A set of unique patterns labeled c_1 to $c_{|C|}$. Each pattern is a vector of the number of components equal to the number of children of that node. The set of pattern labels in a node can be considered as a finite alphabet.
- A set of Markov chains labeled g_1 to $g_{|G|}$. Each Markov chain is defined over a subset of the set of pattern labels in the node.

Each Markov chain g_i in the node is associated with an initial distribution Π_{g_i}. The initial distribution specifies the probability for each state of the Markov chain being the first state to be activated. Each node is also associated with a time constant τ that determines on average how many time steps a Markov chain will remain active. These two parameters control the generation of patterns from a node.



$$c_1^{2,1} = \left[g_1^{1,1}, g_2^{1,2}\right]$$
$$c_2^{2,1} = \left[g_2^{1,1}, g_2^{1,2}\right]$$
$$c_3^{2,1} = \left[g_2^{1,1}, g_3^{1,2}\right]$$
$$c_4^{2,1} = \left[g_1^{1,1}, g_3^{1,2}\right]$$
$$c_5^{2,1} = \left[g_3^{1,1}, g_2^{1,2}\right]$$
$$c_6^{2,1} = \left[g_3^{1,1}, g_1^{1,2}\right]$$

Figure 3.2: Structure of a node: Shown here is a node at level 2. Nodes are represented using their level index and index within the level. $N^{2,1}$ indicates that this node is the first node at level-2 of a network. All the variables within the node are superscripted by $(2,1)$. This node has two Markov chains, denoted by $g_1^{2,1}$ and $g_2^{2,1}$. There are 6 patterns in this node. They are labeled $c_1^{2,1}$ to $c_6^{2,1}$. Each pattern has two components corresponding to the two child nodes of this node. The child nodes are not shown. Also stored within the node is a time constant τ_2 that is the same for all nodes at this level of the hierarchy.

The components of a stored pattern in the node are the Markov chain labels of the child nodes – one component per child. A particular pattern represents a particular co-activation of the Markov chains in its children, with exactly one Markov chain being active in each child. The total number of such configurations is given by the product of the number of temporal groups in each child. However, the number of stored patterns in a node will be much less than this number and represents the number of allowed co-activations of the Markov chains in a node's children[1].

At this point, we need some notation to refer to particular nodes in a hierarchical arrangement of nodes. We will use two indices to uniquely identify a node in a network. The first index will correspond to the level of the network that the node is in. The second index will correspond to the number of the node within that level. For example, we will use $N^{2,1}$ to refer to first node at level-2 of the network. Whenever required, we will use the super-script indexing to refer to labels within the node as well. For example, $g_1^{2,1}$ represents the first Markov chain in node $N^{2,1}$.

Figure 3.2 shows a simple node with 6 patterns and 2 Markov chains. Markov chain g_1 has 3 pattern labels in it - pattern labels c_1, c_4 and c_5. Markov chain g_2 has 3 pattern labels in it - pattern labels c_2, c_3 and c_6.

3.2.3 Generation from a single node

A temporal sequence of patterns can be generated from a node by activating it with the identity of a Markov chain. The node will then generate a sequence of patterns by sampling from the activated Markov chain. The first pattern is emitted according to the initial distribution associated with the activated Markov chain. Subsequent patterns are emitted by following the transition probability table associated with this Markov chain. This process is repeated τ times so that a sequence of patterns that is τ long is generated. The pattern generation stops after τ steps. The node can

[1]The generative nodes have a one-to-one match to the HTM nodes that were introduced in chapter 1. In that chapter, we described HTM nodes that memorized patterns and learned a Markov graph over the space of pattern labels. This Markov graph was then partitioned to form temporal groups. After partitioning, the sub-graphs associated with the temporal groups were discarded. The Markov chains in our generative node correspond to the Markov subgraphs obtained after partitioning the Markov graph in the node during learning.

be started again with the identity of one of the Markov chains to generate further sequences of patterns.

Consider the node shown in figure 3.2. Assume that we start the node with the Markov chain g_1. The node produces the first pattern label by sampling according to the initial distribution of this Markov chain. Let this pattern label be c_5. The pattern corresponding to c_5 , $\left[g_3^{1,1}, g_3^{1,2}\right]$, is emitted. The next pattern label is generated according to the transition probability $P(.|c_5)$. This process is continued for 3 steps and then the node stops.

3.2.4 Generation of patterns from a hierarchy of nodes

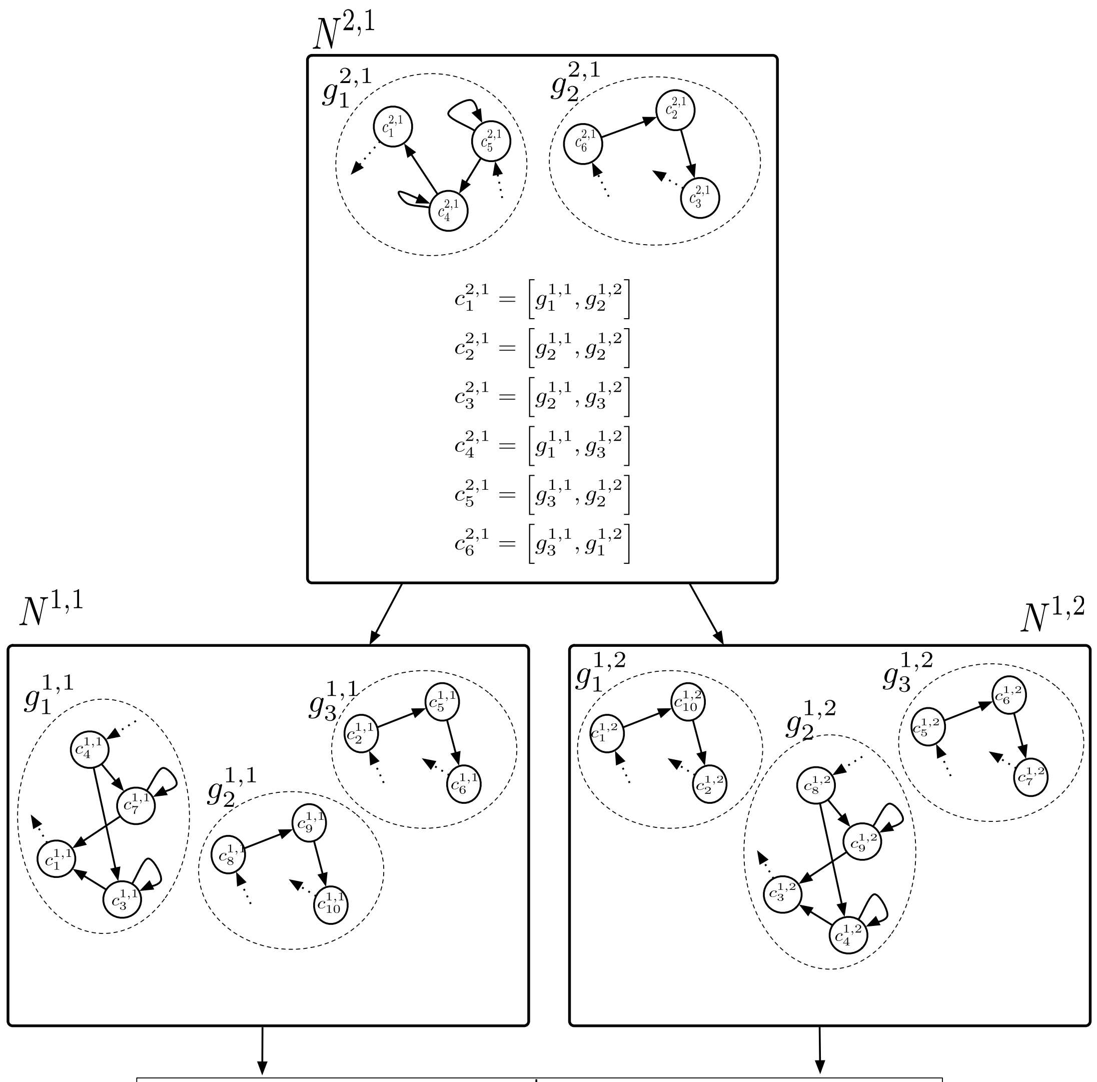
To understand the process of pattern generation in a hierarchy, let us connect two child nodes to the node that we studied in the previous section. These two child nodes are connected to the parent node as shown in the figure 3.3

Suppose we activate the parent node by passing it the identity of one of its Markov chains. Let this Markov chain be $g_2^{2,1}$. The parent node will generate the first pattern by sampling from the activated Markov chain according to its initial distribution. Let the pattern that is generated be $c_6^{2,1}$.

This pattern, $c_6^{2,1}$ has two components – $g_3^{1,1}$ and $g_1^{1,2}$. These components correspond to the Markov chain labels of the child nodes of node $N^{2,1}$. In general, the components of a pattern generated by a parent node are the Markov chain labels of the child nodes.

The pattern generated by the parent node activates the pattern generation in the child nodes by activating each child node with the Markov chain label from the pattern component corresponding to it. This can be thought of as the parent node passing the "control" down to the child nodes. Once the child nodes are activated, they will sample τ_1 times from their Markov chains before the control is returned to the parent. The parent node cannot generate another pattern till the child node has sampled τ_1 times from its Markov chain.

In the current case, the pattern $c_6^{2,1}$ generated by the parent node will activate the Markov chain $g_3^{1,1}$ in the left child node and $g_1^{1,2}$ in the right child node. These Markov


$N^{2,1}$
$g_1^{2,1}$
$g_2^{2,1}$
$c_1^{2,1} = \left[g_1^{1,1}, g_2^{1,2}\right]$
$c_2^{2,1} = \left[g_2^{1,1}, g_2^{1,2}\right]$
$c_3^{2,1} = \left[g_2^{1,1}, g_3^{1,2}\right]$
$c_4^{2,1} = \left[g_1^{1,1}, g_3^{1,2}\right]$
$c_5^{2,1} = \left[g_3^{1,1}, g_2^{1,2}\right]$
$c_6^{2,1} = \left[g_3^{1,1}, g_1^{1,2}\right]$
$N^{1,1}$
$g_1^{1,1}$
$g_2^{1,1}$
$g_3^{1,1}$
$N^{1,2}$
$g_1^{1,2}$
$g_2^{1,2}$
$g_3^{1,2}$

t=0	$c_2^{1,1}$	$c_8^{1,2}$
t=1	$c_5^{1,1}$	$c_4^{1,2}$
t=2	$c_6^{1,1}$	$c_3^{1,2}$
t=3	$c_4^{1,1}$	$c_5^{1,2}$
t=4	$c_7^{1,1}$	$c_6^{1,2}$
t=5	$c_1^{1,1}$	$c_7^{1,2}$

Figure 3.3: A two-level HTW network. Also shown are the patterns generated by this network from time $t = 0$ to time $t = 5$.

chains will be active concurrently. Each child node will then generate patterns from these Markov chains for τ_1 steps. Since these child nodes do not have any more children, the pattern label that is generated is emitted from these nodes. For every time step, we get two outputs - one corresponding to each child. Let the pattern labels emitted by the child nodes at time $t = 0$ be $c_2^{1,1}$ from the left child and $c_1^{1,2}$ from the right child. The concatenation of these two pattern labels, $\left[c_2^{1,1}, c_1^{1,2}\right]$ is a pattern with two components. This pattern is the output of the generative network at time $t = 0$.

The child nodes will continue the generation of patterns for τ_1 time steps ($\tau_1 = 3$). During this time, the parent node is not allowed to generate new patterns from its active Markov chain. Thus the output of the level-2 node will remain $c_6^{2,1}$ for these three time steps. At the end of τ_1 steps, the control is returned to the parent node and the parent node can generate the next pattern by sampling from its Markov chain (provided, the parent node hasn't already sampled τ_2 times from that Markov chain). The generation of a new pattern at the parent node once again initiates a pattern generation cycle at the child nodes.

This generation process can be extended to any number of levels. When a parent node generates a pattern and passes the control to its children, all the children possess the control simultaneously. Nodes lose control by either passing it to the parent or by passing it down to the children. Control is split as it passes down in the hierarchy - when a parent node passes the control down, all its children receive it. Control is aggregated as it passes up in the hierarchy - a parent node receives control only when all its children have passed their control to it. The nodes at the same level of the hierarchy generate patterns in synchrony and in parallel.

Whenever a node generates a pattern, the control is passed down to its children. The nodes at level-1 of a hierarchy do not have any children. Therefore, level-1 nodes do not pass down control and do not wait for the control to be returned. When control is passed to a level-1 node, the level-1 node emits pattern labels for τ_1 time steps and returns the control to the parent. All the level-1 nodes emit pattern labels concurrently for every time step. Therefore, at each time step we will get one pattern label from each level-1 node. At any time t, the concatenation of the outputs of the level-1 nodes is a vector of length equal to the number of nodes at level one. This vector is the observable pattern generated by the network at time t. Figure 3.3 shows the data generated from the two-level network for six time steps.

The data generated from an HTW network is both spatial and temporal. At each time point, the data has multiple components corresponding to the nodes at the bottom level of the hierarchy. This is the spatial or structural component of the data. Each of these components evolves in time. The generated data is a multi-variate time series with interleaved spatial and temporal properties.

3.3 Properties of hierarchical-temporal data

We now quantitatively analyze the data generated from and HTW network. Figure 3.4 shows a four-level HTW network with a fan-out of two at each level. Each node of this network has 20 patterns and 5 Markov chains. Each Markov chain contains 4 patterns. The same pattern does not participate in more than one Markov chain. The time constant of each level is set to an average value of 5. This was achieved by varying the time constant between 2 and 8 in a uniformly random manner after every cycle of generation. With this the active Markov chains at level 1 will change once in 5 time steps, on average, and the active Markov chains at level 2 will change once in 25 time steps, on average. In general, the active Markov chain at level i will change on average every 5^i time steps.

All the Markov chains are identical in their alphabet size and they use the same transition matrix, albeit with a different mapping of the states to the set of patterns. The transition matrix has a probability of transition value of 0.57 between its $(i \; mod \; 4)^{th}$ state and $(i+1 \; mod \; 4)^{th}$ state and a probability of transition value of 0.14 between every other pair. These quantities were chosen to ensure a strong sequential transition component with some amount of randomness. Any combination of values that ensures this would suffice.

Nodes in the network generate a pattern label for every time step. As the network is executed, each node generates a time-series. Figure 3.4 shows the arrangement and labeling of the generated data. Each level of the network creates a spatio-temporal

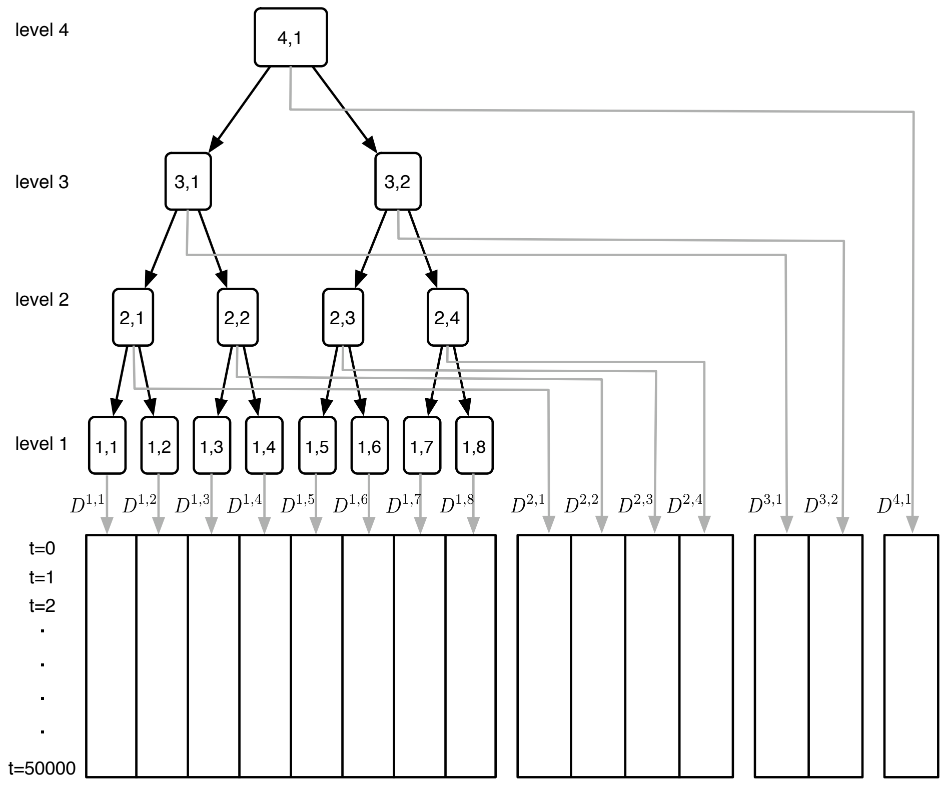


Figure 3.4: Structure of the HTW network used for the experiments in section 3.3.

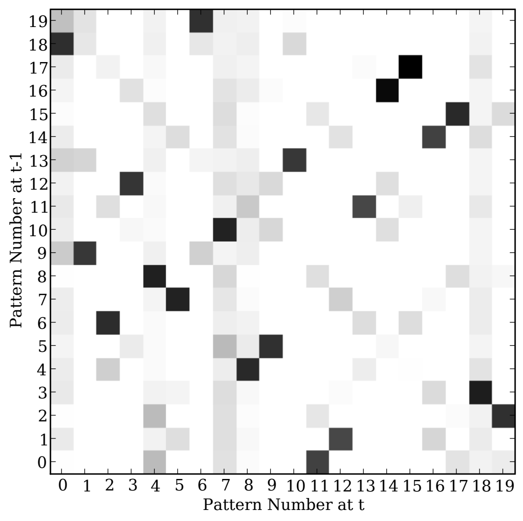


Figure 3.5: Transition probability matrix obtained from the data stream $D^{1,1}$ of the level-1 node $1, 1$

data set. We ran the network for 50000 time steps and stored the time-series corresponding to each node of the network. We then quantitatively analyzed the properties of the generated data.

3.3.1 Temporal structure of data streams from a single node

Figure 3.5 shows the first order transition matrix obtained by observing the data stream of node 1 at level 1. This transition matrix was obtained by counting the number of transitions in the data stream from one pattern label to another. The rows and columns of this matrix correspond to the pattern labels of node 1. The $(i, j)^{th}$ entry of this matrix gives the estimated probability that the next pattern is j given the current pattern is i.

Figure 3.6 shows the same transition matrix, with rows and columns now rearranged such that the pattern labels belonging to the same Markov chain occur adjacent to each other along the rows and columns[2]. This matrix has a block diagonal

[2]Clearly, this kind of re-arrangement is possible only when a pattern occurs only in one Markov

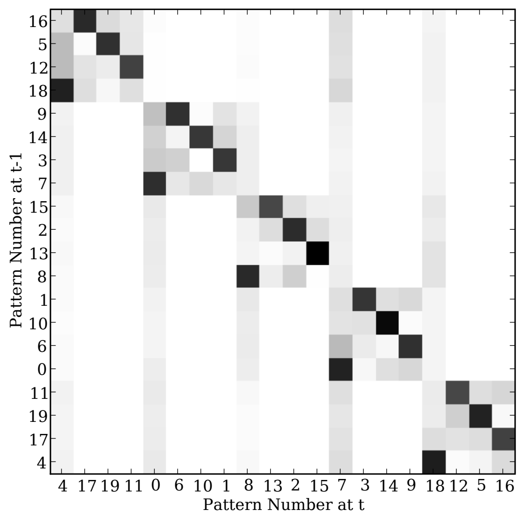


Figure 3.6: Reordered transition probability matrix.

structure. This structure arises because of the time constant τ_1 of the node. This time constant ensures that, once a Markov chain is started, patterns are generated from that Markov chain for the duration of τ_1, on average.

Suppose we did not know the Markov chain memberships of these patterns. Under benevolent conditions, we can use this temporal slowness property to recover those memberships and the underlying Markov chains. The transition matrix in figure 3.6 suggests that this can be done by identifying tightly connected sub-graphs of the graph representing the transitions between patterns of a node. The underlying sub-graphs, with appropriately normalized edge weights, will then correspond to the Markov chains. We will see in section 3.5.2 that the temporal slowness is a key property exploited during learning.

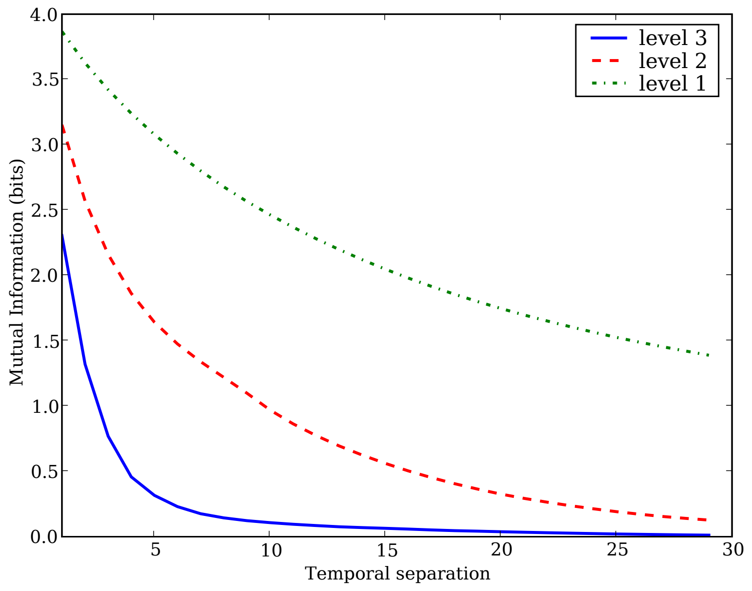


Figure 3.7: Temporal mutual information of nodes at level-1, level-2 and level-3.

3.3.2 Mutual information vs. temporal separation in the hierarchy

Consider the time-series $D^{1,1}$ generated by node 1 at level-1. For brevity, let us denote this stream by D. For any given time t, $D(t)$ is the pattern label generated by this node. Suppose we know the pattern label $D(t)$ at time t. The amount of information that this gives us about the pattern label $D(t+\Delta)$ at time $t+\Delta$ depends on the temporal properties of the data stream. We can use mutual information [17] to quantify this property [3]. Let D_Δ represent the data stream obtained by delaying the D data stream by Δ time steps. Then the mutual information $I(D, D_\Delta)$ between D and D_Δ quantifies the amount of information that on average can be obtained about a time step in stream D by measuring at a time step Δ steps preceding it.

chain

[3]Mutual information I between two random variables Y and Z is defined as $I(Y,Z) = H(Y) - H(Y|Z)$, where H represents the entropy. Intuitively, this can be thought of as the reduction in the uncertainty of one variable given the other.

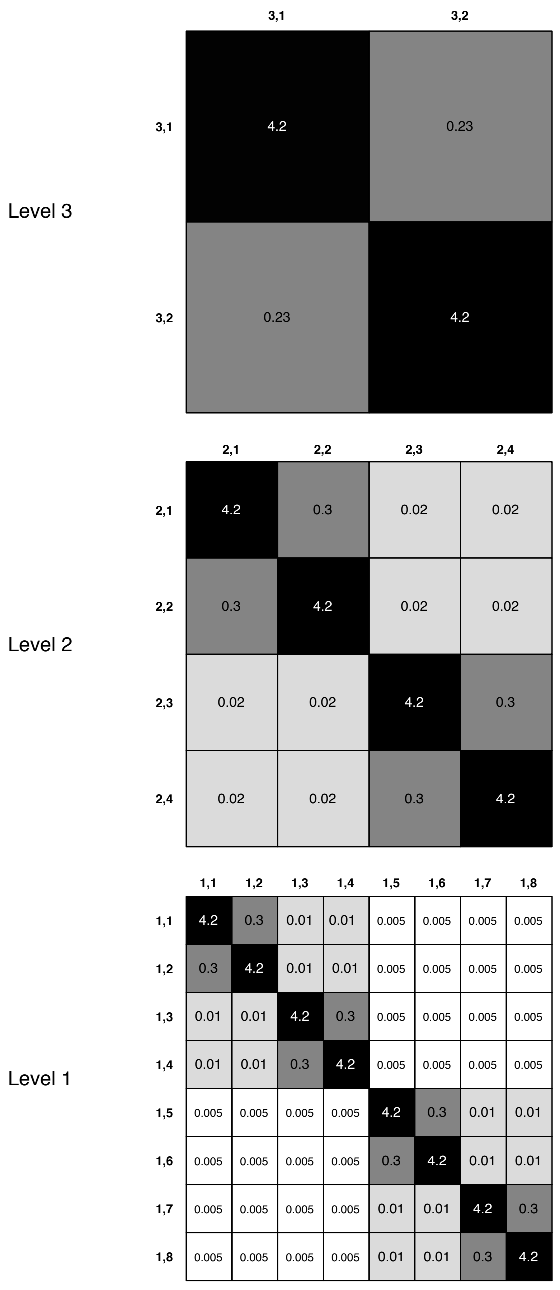
$I(D, D_\Delta)$ can be thought of as a generalization of the auto-correlation function and can be used to measure the amount of temporal dependency in a data stream D. (Auto-correlation cannot be used because the variables here are nominal.) For this reason, we will call this quantity the temporal mutual information. As all the nodes in our network have identical parameters, the temporal mutual information of the time-series generated by a node depends on the level of that node in the network.

Figure 3.7 plots the mutual information vs. temporal separation of nodes at levels one, two and three of the hierarchy. The plots clearly show the temporal dependency in the data streams. At level 1, knowing a pattern label at any time t gives on average 0.3 bits of information about the pattern label at time $t + 5$. The mutual information falls off with increasing temporal separation. It can also be observed that the temporal dependency of the states of the nodes at level 2 is larger than that of the nodes at level 1. One of the properties that we wanted our generative model to hold was the increasing temporal stability as we ascend in the hierarchy. This property is reflected in the plots. The temporal dependency of the data streams generated by a node increases as you move up in the hierarchy because the active Markov chains and patterns of the nodes at a higher levels change at a slower rate compared to that of a lower level node.

3.3.3 Mutual information vs. structural separation in the hierarchy

Suppose we know the pattern label that was emitted by a node at a particular time. How much does this tell us about the pattern label of another node within the same level at that time? This can be quantified by measuring the mutual information between the data streams of those nodes.

As shown in figure 3.4, the data streams generated by level-1 nodes are labeled $D^{1,1}, D^{1,2}, \cdots, D^{1,8}$. We measured the pair-wise mutual information between these data streams. The results of this measurement are shown in figure 3.8 (Level-1). Clearly, the nodes that are connected to the same immediate parent show high amount of mutual information. The nodes that are not connected to the same parent show



Level 3

	3,1	3,2
3,1	4.2	0.23
3,2	0.23	4.2

Level 2

	2,1	2,2	2,3	2,4
2,1	4.2	0.3	0.02	0.02
2,2	0.3	4.2	0.02	0.02
2,3	0.02	0.02	4.2	0.3
2,4	0.02	0.02	0.3	4.2

Level 1

	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8
1,1	4.2	0.3	0.01	0.01	0.005	0.005	0.005	0.005
1,2	0.3	4.2	0.01	0.01	0.005	0.005	0.005	0.005
1,3	0.01	0.01	4.2	0.3	0.005	0.005	0.005	0.005
1,4	0.01	0.01	0.3	4.2	0.005	0.005	0.005	0.005
1,5	0.005	0.005	0.005	0.005	4.2	0.3	0.01	0.01
1,6	0.005	0.005	0.005	0.005	0.3	4.2	0.01	0.01
1,7	0.005	0.005	0.005	0.005	0.01	0.01	4.2	0.3
1,8	0.005	0.005	0.005	0.005	0.01	0.01	0.3	4.2

Figure 3.8: The pair-wise mutual information plots between the data streams generated by the nodes at levels 1, 2 and 3. Mutual information between two variables X, Y is calculated using the formula $I(X,Y) = H(X) - H(X|Y)$. The mutual information (in bits) between pairs of data streams is indicated within the corresponding rectangle.

significantly lower mutual information. The same trend can be seen at levels 2 and 3 as shown in figure 3.8.

An important property of the hierarchy can be observed by examining how the measurements of mutual information between data streams vary in the hierarchy. Consider the data streams generated by level-1 nodes 1, 2, 3 and 4. Data streams $D^{1,1}$ and $D^{1,2}$ have low mutual information (0.01 bits) with data streams $D^{1,3}$ and $D^{1,4}$. However, there is high mutual information (0.3 bits) between the level-2 data streams $D^{2,1}$ and $D^{2,2}$.

Looking at the "raw" data streams $D^{1,1}, D^{1,2}$ and $D^{1,3}, D^{1,4}$, one might conclude that there is no advantage in jointly modeling $D^{1,1}, D^{1,2}$ and $D^{1,3}, D^{1,4}$. The data stream of the level-2 node $(2, 1)$ can be derived from the data streams of nodes level-1 nodes $(1, 1)$ and $(1, 2)$ by first finding the groups corresponding to the patterns and then finding the co-occurrences of the groups. Similarly, the data stream of the level-2 node $(2, 2)$ can be derived from the data streams of level-1 nodes $(1, 3)$ and $(1, 4)$. What we observe is that by doing one step of temporal and spatial abstractions on the set of level-1 nodes $(1, 1)$ and $(1, 2)$ and on the set of nodes $(1, 3)$ and $(1, 4)$, we obtain mutual information that was not visible at level-1 between $\{D^{1,1}, D^{1,2}\}$ and $\{D^{1,3}, D^{1,4}\}$. The same property holds good between nodes at levels 2 and 3.

The nodes that are at the same level of the hierarchy and not connected to the same parent act nearly independently. However, if we abstract out the details of these lower levels and go to a higher-level, we see patterns of interaction between these nodes. This *loose coupling* provides a way of learning these modules independently of each other, and abstracting the details at a lower level to see higher-level interactions. This is another property that we will exploit in learning hierarchical temporal memory systems.

3.3.4 Summary of the properties of HTW generated data

- Hierarchical organization in spatial (structural) and temporal dimensions
- Lower level nodes are associated with smaller portions of the patterns generated from the network. Higher level nodes influence larger portions of the generated

patterns, but with less detailed control

- Temporal predictability at the lower level nodes is smaller compared to the temporal predictability at higher levels. Temporal variations in lower level nodes occur faster compared to higher levels
- The nodes at any level that are not connected to the same parent are loosely coupled
- Temporal and spatial abstractions of one level reveal interesting structure that can be modeled by another level. Without making these abstractions, the dependency between different streams might not be visible at a lower levels.

3.4 Learning problems associated with hierarchical temporal data

Learning involves the construction of a model through the observation of hierarchical-temporal data. The complexity of the learning task can vary depending on the amount of prior knowledge that can be applied to the problem. Based on this, we can put the learning problems into two broad categories:

1. **Learn the network structure and node states from data:** The most general learning problem that can be defined on this data set involves learning the complete network structure that generated the data. This includes learning the hierarchical connectivity between the nodes and the internal states of each node in the hierarchy. If no assumptions are made about the data generation mechanism, solving this problem involves searching through a hypothesis space that is possibly infinite. However, under the assumption that the data was generated from an HTW network of unknown structure, practical algorithms can be devised to extract an approximate network structure.
2. **Assume the network structure. Learn the node parameters from data:** In many learning problems, the network structure can be designed using prior

domain knowledge. Then the only task that remains is of learning the states of the nodes in the network.

The complexity of learning and the details of what is learned will also depend on the task for which the model is built. Given below is a non-exhaustive categorization of the learning problem based on the task.

- **Classification problem:** The visible patterns generated by a network belong to different Markov chains at the top node of the network. The identity of the Markov chain that is active at the top-level node when a visible pattern is emitted by the network can be thought of as the class identity of that pattern. If we have a learned model, then for every input pattern, we can query the model to find out the Markov chain identity of that pattern at the top of the network. In this manner, a learned model can be used for classification.
- **Prediction problem:** The network can be queried to predict the next pattern based on a sequence of observed patterns.
- **Reconstruction problem:** The patterns that occur in the real world are noisy and ambiguous. A learned model can be used to clean up an input pattern.

3.5 Hierarchical temporal memory: a learning methodology for hierarchical-temporal data

We will now devise a learning methodology for hierarchical temporal data. This learning methodology satisfies the following two criteria:

1. Be a universal learning algorithm for the hierarchical-temporal world: This means that, we will not make any more assumptions about the world than the generic assumptions about the hierarchical and temporal world set-forth in the generative model. This will make our learning method applicable to a large class of problems that fall within these assumptions.

2. Be memory based. i.e, non-parametric: This means that we will not be making any assumptions about the shape of the probability distributions that can occur in the world. Instead, we will approximate the distributions by using explicit memorization. Temporal learning will be done using Markov models with multinomial distributions.

Since the methodology that we describe is hierarchical and temporal and memory-based (non-parametric), we refer to this by the name Hierarchical Temporal Memory (HTM).

Learning is the reverse process of generation. We are given a long stream of patterns that is generated from an HTW network. We do not know the structure of the network or the coincidence-patterns and Markov chains in any node in the generative network. By observing the data given to us, we want to get back the network-structure and the coincidence-patterns and Markov chains in the nodes of the network that generated the data.

We describe the learning process in two stages. First we consider the problem of learning the network structure and then we will consider the problem of learning the coincidence-patterns and Markov chains in the nodes.

3.5.1 Learning the structure of an HTM network

We are given multi-variate temporal data. The data have as many temporal streams as the number of variables. Let us assume that the number of streams is L_0. The pattern at any time point is a vector of L_0 components. By observing this data stream, we need to identify the hierarchical structure that generated this stream. In this section we will consider a greedy sub-optimal algorithm for doing this.

In section 3.3, we found that data streams that are generated from the nodes with the same immediate parents have high mutual information while data streams generated from nodes that are further separated in the hierarchy have low mutual information. Our network construction algorithm is based on this observation. The basic idea behind this algorithm involves using mutual information to find the data streams that are to be fed to the same node.

First, measure the pair-wise mutual information between the L_0 data streams. These pair-wise measurements can be thought of as a measure of similarity between the data streams. Then divide the streams into L_1 clusters based on these mutual information measurements. The number of clusters could be a parameter or could be discovered by the clustering algorithm. The first level (level-1) of the network is constructed using L_1 nodes. All the data streams that fell into the same cluster are fed to the same node at level-1.

Next, train all the level-1 nodes by feeding in the data streams to these nodes according to their clustering assignment. (In the next section we will describe the method for training the nodes. For the time being, we will assume the existence of such a method.) After the level-1 nodes are trained, those nodes will start producing outputs. An output is produced for every input. Since there are L_1 nodes at level-1, we will have L_1 data streams. At this stage, we have constructed and trained the first level of the network.

For the next stage of network construction, we treat these L_1 data streams the same way we treated the original L_0 data streams. That is, we measure the pair-wise mutual information between these L_1 data streams and put them into L_2 clusters. We assign one node for each cluster and feed those data streams that fall into the same cluster into those nodes. This process is repeated until we reach a single node at the top.

3.5.2 Learning the states of a node

The process involved in learning the states of an HTM node was discussed in detail in section 2.3. For convenience, we summarize that discussion in this section.

A node that is not trained is a tabula rasa. Through the learning process we hope to create within the node the coincidence patterns and the Markov chains of the corresponding hierarchical temporal world that generated the patterns.

The input to a node in the network is a sequence of input patterns, presented one pattern at a time. The node, when it starts the learning process, is a tabula-rasa. It has nothing stored in its memory and it hasn't formed any Markov chains/temporal groups. At the end of the learning process, the node will have memorized several patterns and formed several Markov chains. The node achieves this learning in a two-step process - (1) Memorizing the coincidence-patterns and (2) Learning the temporal groups.

Memorizing the coincidence patterns

The input to a node at an intermediate level in the hierarchy is the concatenation of the outputs of its child nodes. If the node has M children, this is a pattern of M components. Each component of this pattern is the "name" of the Markov chain/temporal group from the child node. The whole pattern represents a co-occurrence of the temporal groups in the child nodes.

In the first step of the learning process, the node memorizes the unique patterns that it sees in its inputs and labels those patterns. We will assume that the node observes the input stream long enough to see all the coincidence patterns in the data. The memorized patterns are labeled $c_1, c_2, \cdots, c_{N_c}$, where N_c is the number of memorized patterns.

Learning the Markov chains/temporal groups using time as the supervisor

Once the first stage of learning is finished, a sequence of input patterns produce a sequence of pattern labels within the node. In the second stage of the learning process, the node learns the Markov chains within the node. The node does this by observing the transitions of pattern labels in time. By learning the Markov chains, the node is modeling the transitions of pattern labels using a mixture of Markov models. Only one of these Markov models will be active at a particular time. Each Markov model is responsible for modeling a temporal duration approximately equal to the time constant of the node.

If the input stream of patterns came pre-segmented with their assignments to Markov chains, then the problem of learning the Markov chains is a trivial one – learn the transition probabilities in a Markov chain only using those input sequence segments that are assigned to that Markov chain. However, the input stream of pattern labels does not come pre-segmented. The node has to, on its own, figure out the segmentation of the input sequence and learn the Markov chains based on the appropriate segments.

Without any further information, this segmentation problem is under-specified and therefore not solvable in practice. In section 3.3 we described how temporal slowness is a key assumption in the generation of patterns from an HTW network. This property ensures that a Markov chain that is generating patterns tends to be active for some duration of time. In learning the Markov chains, the node assumes that the pattern labels that persistently follow each other in time are likely to come from the same Markov chain.

A simple algorithm that can be used for learning the Markov chains involve learning a transition matrix between the complete set of pattern labels and then partitioning that transition matrix into smaller transition matrices corresponding to the individual Markov chains. We can consider the original transition matrix as a graph. Then, with the assumption of temporal stability, the transition matrices of the individual Markov chains will correspond to tightly connected sub-graphs of this original graph. This method of learning the Markov chains will work when the pattern labels belonging to one Markov chain are mostly disjoint from the pattern labels belonging to a different Markov chain.

3.6 Classification problems

Many machine-learning theories are constructed around the concept of classifying patterns in a supervised manner [25]. In supervised classification problems, a training data set is available that has a set of patterns and their category labels. A classifier is then trained on this training set data. The goal is to learn the boundaries between these classes so that novel patterns can be assigned to their correct classes. The performance of these classifiers is measured using a test set.

In contrast to the standard clustering mechanisms and discriminative learning mechanisms that are distance-based, the classification in HTMs is model-based. The patterns are assumed to come from a Hierarchical Temporal World (HTW) model.

In the HTW generative model, for every pattern that is generated from the network, there is a Markov chain at the top-node of the network that is active. The identity of this Markov chain is the class label for that pattern.

In HTMs, the classification problem is the problem of identifying the identity of the Markov chain at the top node that corresponds to the input pattern. In a tree structured hierarchical network, this can be solved using Bayesian belief propagation techniques. See appendix A for the description of belief propagation in HTM networks.

3.7 Convolutional neural networks, HMAX model, Neocognitron

Convolutional neural networks [58] and feed-forward models of the visual cortex like Neocognitron [32] and the HMAX model [84] have been very successful on visual pattern recognition problems. Despite the wide variety of learning algorithms employed in these models, they all share the same structural properties. All these models have feed-forward hierarchies with alternating layers of *feature selection* and *feature-pooling levels*.

We will use the illustration in figure 3.9 to understand the basic mechanism behind these models. The input image in the figure is a two dimensional array of pixel values. The segment of the network shown in the figure consists of feature-detection neurons (shown as circles) and feature-pooling neurons (shown as rounded rectangles). As illustrated, the feature detection neurons have receptive fields that are much smaller in size compared to the size of the whole input image. In this example, we have two kinds of feature detectors - a vertical line detector and a horizontal line detector. These feature detector neurons respond to the presence of a vertical/horizontal line in a small patch of the input image that corresponds to their receptive fields.

Now, consider the feature pooling neuron 'a' in figure 3.9. This neuron pools across the outputs of two feature detector neurons of the vertical line type. The output of the feature pooling neurons is the maximum of the output of these neurons. This

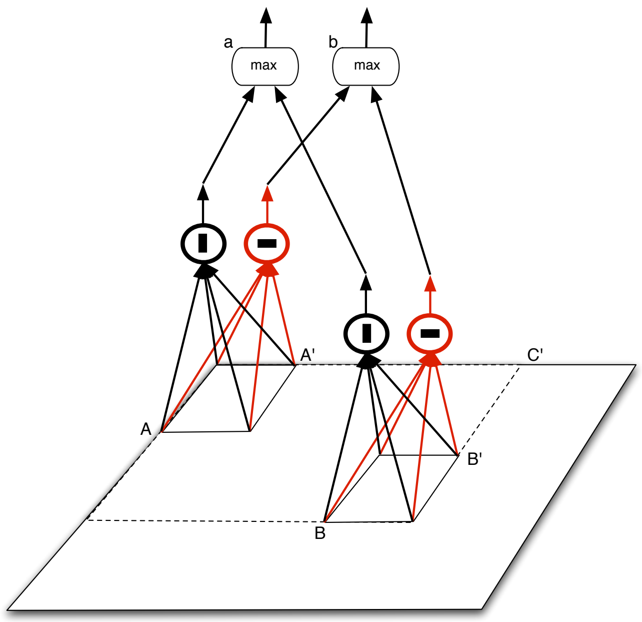


Figure 3.9: This figure shows the basic mechanism of feature-detection and feature-pooling employed by convolutional neural networks and by the HMAX model.

means that this feature pooling neuron will respond if a vertical line is present either in the input field A–A', or in the field B–B' or in both. By max-pooling the responses of the vertical line feature detectors in A–A' and B–B', the feature pooling neuron has become an invariant detector of the vertical line in the receptive fields A–A' and B–B'.

If we tile the rectangular input field C–C' with several vertical line detectors and then pool their responses using a max-pooling neuron, then the response of that max-pooling neuron will be invariant to the translation of a small vertical line segment within the C–C' rectangular input field. Using the same mechanism, we can build translation invariance to any kind of feature. All we need to do is pool across translated versions of that feature type using a max-pooling mechanism. This is the basic mechanism employed by Neocognitron, convolutional neural networks, and the HMAX model. By building invariant pooling over small receptive fields and then stacking them gradually with alternating layers of feature detection and feature pooling, these networks gain invariance to translations and small scale variations of the same image category, while maintaining selectivity to images belonging to different categories.

It is important to note that the invariant pooling is *hard-coded* in convolutional neural networks and in the HMAX model. In these models, the feature pooling neuron did not learn to pool across the outputs of translated feature detector neurons of the same type. Instead, the pooling was pre-determined by the designer of the network.

How did the network designer know what features to pool across? It is quite intuitive that in the visual world, we want features invariant to translations and scale variations. Therefore, in the case of the visual world, it is possible to hard-code translation invariance and scale invariance into the network. However, the technique of hard-coding the invariance will not transfer to other domains. Even in the vision domain, it will become increasingly difficult to hard-code rotations and other deformations. Learning these invariances is the only general strategy that we can apply to build models for any domain. In the visual world, those learned invariances will correspond to translation, scaling etc. In a different domain, we will have to invent new words to describe the invariances that are learned.

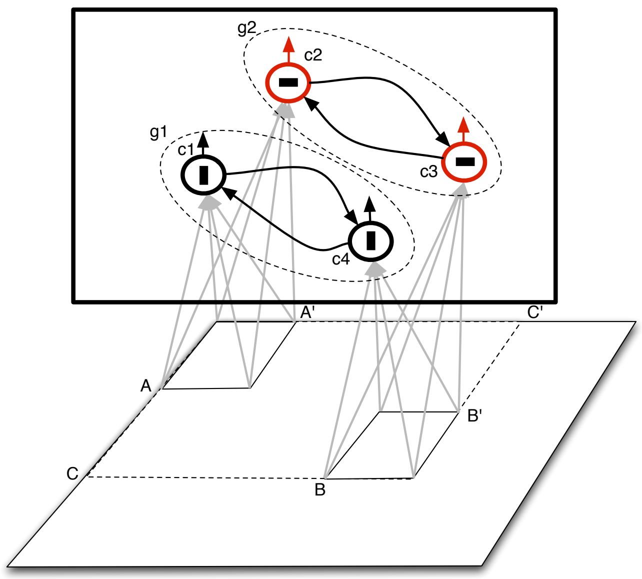


Figure 3.10: Mapping of the feature-detection and feature pooling layers of convolutional neural nets to the HTM node.

3.7.1 Mapping to hierarchical temporal memory

Convolutional neural nets and models like Neocognitron and HMAX can be mapped to HTMs in a straightforward manner. The simple cell layers in these models map to the coincidence-patterns in HTMs. A simple cell in the convolutional network remembers a configuration of features from the feature maps at lower levels. Similarly, a coincidence pattern in an HTM node, remembers a configuration of the Markov chain "features" from the lower level nodes.

The feature-pooling levels of these models map to the Markov chains of an HTM node. The max pooling operation maps to the max-propagation mode of Bayesian belief propagation used in HTMs. Figure 3.10 shows how the segment of convolutional network in figure 3.9 can be mapped to an HTM node.

Having the feature-pooling layer as a learned temporal group gives HTMs several advantages over convolutional neural networks and similar models. In HTMs, the feature pools are learned using temporal continuity. This lets the feature-pooling levels in HTMs pool together patterns that are otherwise very different in a spatial similarity metric. Moreover, this lets HTMs to be invariant to the transformations that are present in the data rather than hard-coding the transformations. In general, learning these feature pools is the only viable strategy to apply the same modeling techniques for data from different domains.

Feature pooling in convolutional neural networks and similar models correspond to treating the Markov chains in an HTM node as zeroth order models by discarding their transition probability tables. HTMs in general have the capability of using these Markov chains as a variable order temporal model. This means that HTMs can exploit the temporal continuity not only during learning, but also during inference. For example, movement of an input picture gives more information about the transformation of features. Since an HTM has encoded these transformations as part of its Markov chains, HTMs can exploit this additional information by integrating evidence over time using the dynamic programming equations described in appendix A.

3.8 Generalization

Generalization is the capability of pattern recognition systems to correctly classify patterns that were never seen during training. In non-parametric methods like Nearest Neighbors, generalization is achieved through the Euclidean distance function. A new pattern is assigned to the same class of its nearest prototype. More sophisticated classifiers like SVMs find the "boundaries" between different classes in a higher dimensional space. The boundaries are parameterized and then those parameters are fit using the training set. However, generalization can suffer in these models because the actual class boundaries in the high dimensional space can be very discontinuous and fast changing.

3.8.1 Generalization in HTMs

It is the combined effect of the Markov chains and the hierarchy that gives rise to generalization in hierarchical temporal memory systems. The sequential information stored in the Markov chains is not required for understanding the generalization

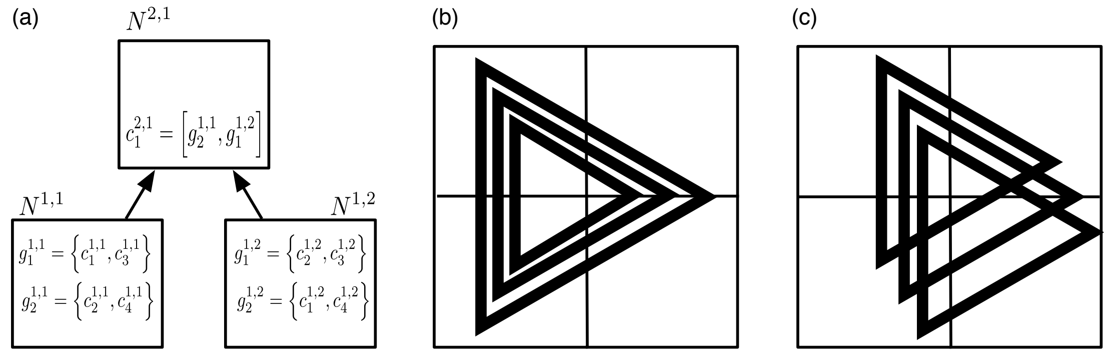

(a)
$N^{2,1}$
$c_1^{2,1} = \left[g_2^{1,1}, g_1^{1,2}\right]$
$N^{1,1}$
$N^{1,2}$
$g_1^{1,1} = \left\{c_1^{1,1}, c_3^{1,1}\right\}$
$g_2^{1,1} = \left\{c_2^{1,1}, c_4^{1,1}\right\}$
$g_1^{1,2} = \left\{c_2^{1,2}, c_3^{1,2}\right\}$
$g_2^{1,2} = \left\{c_1^{1,2}, c_4^{1,2}\right\}$
(b)
(c)

Figure 3.11: (a) A two-level hierarchy to illustrate the source of generalization in HTMs. (b) Invariance to translation (c) Invariance to scale variation

properties. Therefore, for the rest of this discussion, we will consider each Markov chain as a zeroth order model. A Markov chain that is considered to be a zeroth order model will be called a *temporal group*.

To explain the generalization mechanisms in HTMs, we use a simple two-level network with two nodes at level 1 and one node at level 2, as shown in figure 3.11(a). Consider a particular coincidence pattern at level 2. This coincidence pattern specifies the co-occurrence of a temporal group (zeroth order Markov chain) from the left child and a temporal group/Markov chain from the right child.

Let us assume that the co-occurrence pattern $c_1^{2,1}$ at level-2 belongs to "category-1". This pattern represents the co-occurrence of $g_2^{1,1}$ and $g_1^{1,2}$ from the children. By definition, all the patterns that correspond to the cross-product space of $g_2^{1,1}$ from the left child and $g_1^{1,2}$ from the right child will belong to "category-1" because, all these input patterns to the network will co-activate g_1 from the left child and g_2 from the right child. However, to learn this co-occurrence, the level-2 node had to see only one such instance. For example, suppose that the network was first exposed to the input pattern $\left[c_2^{1,1}, c_4^{1,2}\right]$ which it memorized to be category 1. By the nature of the temporal at level-1, the network will generalize patterns $\left[c_2^{1,1}, c_1^{1,2}\right]$, $\left[c_4^{1,1}, c_4^{1,2}\right]$ and $\left[c_4^{1,1}, c_1^{1,2}\right]$ as belonging to this category.

The temporal groups/Markov chains in the children are not specific to any higher-level category of patterns. The temporal groups in these nodes could be learned without the network having seen even one pattern of category 1. This is possible because the same temporal groups appear as part of different categories. Once the temporal groups in the lower level nodes are learned, a new category can be learned as a new co-occurrence of the temporal groups. This new configuration can be learned by exposing the network to just one input pattern.

Though simple, this mechanism is very powerful. Consider figure 3.11(b) and (c). In this figure, assume that, the level-1 nodes have all learned different translations of vertical lines, 45^o lines and -45^o degree lines as their temporal groups. The second level memorized a triangle as a particular configuration of these temporal groups, by getting exposed to one image of the triangle. Now the network will be able to generalize to different scale variations and translations of this triangle based on the property that we described above. Any transformation of the triangle where the local variations remain within the current temporal groups (Markov chains) of the corresponding nodes will be recognized as the triangle itself.

3.8.2 Generalization: numerical example

We will illustrate the generalization properties using an artificial example. First, we generate data from a three level HTW network. Each node in the network has three temporal groups (zeroth order Markov chains) and six coincidence patterns. Each temporal group has two coincidence patterns each. As before, we treat the concatenated outputs of level-1 nodes as the output pattern of the network. This network can create a total of 384 different patterns, each a vector of four components. See appendix C for an analysis of the number of patterns that can be generated from an HTM hierarchy. The patterns that are generated by the network are divided into three categories corresponding to the three temporal groups at the highest level. Therefore there are 128 patterns per category.

We are interested in training an HTM network to classify these patterns. Suppose that we are given the network structure that generated the data, but not given the

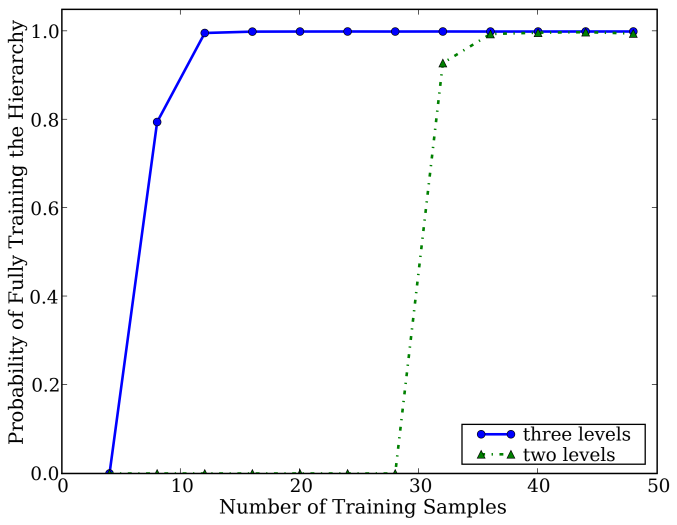


Figure 3.12: Results of the example to illustrate the generalization power that we gain from using a hierarchy.

mapping of patterns to temporal groups in each node. Also assume that instead of using time as the supervisor at each node, we are allowed to query a friendly oracle that will tell us how to map patterns to Markov chains. For any pattern that is picked from the set of possible patterns, the friendly oracle tells us which of the 3 top-level categories it came from. The oracle also helps with queries about a node. For example, take a 4-component pattern and split it up into the 4 components corresponding to the 4 level-1 nodes. The oracle will then give information about the Markov chain to which the pattern-segments belong in their corresponding level-1 nodes.

The friendly oracle here is mimicking the effect of using time as the supervisor. In practical cases, an HTM node will need to see sequences of occurrences of these patterns to carve out the Markov chains. This means that several of these patterns need to be seen repeatedly. However, for simplifying our analysis of generalization, we assume that the friendly oracle tells us the Markov chain of a pattern even if it is seen only once.

If we are allowed to use answers from a friendly oracle, then we can build up the network level by level by first mapping out the level-1 nodes and then using the level-1 node outputs to map out level-2 nodes and so on using the following strategy. We pick a pattern from the set of all possible patterns. This pattern has 4 components corresponding to the 4 level-1 nodes. Each component is a pattern label of the corresponding node. If the pattern label within a node was not observed before, we query the oracle to find out the pattern-label to Markov chain mapping. If the pattern label in a node was observed earlier, then we already know the pattern-label to Markov chain mapping for that node. We produce the outputs of level-1 nodes under this mapping.

Now we look at the inputs to the level-2 nodes. The input pattern to a level-2 node is the concatenation of the outputs of its child level-1 nodes. Following the above strategy of asking the oracle or looking up a previously learned mapping, we map the patterns at level-2 nodes to outputs. Repeating the same strategy, we can find the category output of an input pattern to the network.

Once all the patterns in all the nodes are observed, the hierarchy is fully mapped out. With a fully trained hierarchy, we do not need the oracle to correctly classify the rest of the patterns. Fewer the patterns required for training the hierarchy, greater the generalization.

From the complete set of 384 patterns, we can randomly pick a particular number of patterns as training set. Given the number of patterns in the training set, we can calculate the probability that the hierarchy will be fully trained with that number of training patterns. For this numerical example, we did this calculation by simulating the process several times and counting the number of times the hierarchy was fully trained. We varied the number of examples in the training set and repeated this process to obtain the graph shown in figure 3.12. We also calculated this curve for a two-level hierarchy. For this case, we assumed that the network had four nodes at level-1 all feeding into the same single level-2 node.

The plots shown in figure 3.12 illustrate the generalization capabilities of this hierarchical learning. For a three level training hierarchy, the probability of fully training the system goes to 1 for a randomly selected training set of about 12 patterns. That is, out of the 384 patterns we need to only observe about 12 patterns as part of the training process to correctly classify all the other 372 patterns. A two-level training hierarchy applied to the same learning problem will become fully trained after observing about 40 training patterns. Although this is considerably worse than the performance of a three level hierarchy, the benefit of using a hierarchical training is still achieved even when the training hierarchy does not perfectly match the generation hierarchy.

The probability that an HTM hierarchy will be fully trained with a given number of patterns can be calculated theoretically. Equations for this calculation are derived in appendix C.

3.9 Related work

The work done in this chapter has several connections to the work done by Rivest and Sloan on the topic of Hierarchical Concept Learning. In [85], they proved that hierarchical concepts could be learned efficiently if a teacher is available at every level of the hierarchy. We are making use of this fundamental result by using time as the teacher at every level of the hierarchy. However we haven't analyzed the additional complexity involved in using time as a supervisor.

Hierarchical systems have been the studied in different contexts. Herbert Simon [89] described the structural and temporal dimensions of the hierarchical organization of the world. In the book titled *Hierarchy Theory* [74], several authors described the hierarchical organization of the world from different points of view.

Hierarchies have also been studied from the views of scaling laws and compositionality. In [33], Geman et al described the nature of structural hierarchies and their compositional advantages. In [15], Changizi identifies a set of universal scaling laws for compositional hierarchical systems. In both these works the nature of structural hierarchy is studied in detail while the temporal hierarchy is left unexamined. Their hierarchies did not have alternating temporal and spatial abstractions. Therefore, the unique generalization source provided by such abstractions was not identified.

Harrison studied algorithms for discovering compositional structure in the world and identified and analyzed spatial and temporal abstraction algorithms [41]. However, they were not put together in a hierarchical manner and their generalization properties were not analyzed.

The work in this thesis was influenced by the work of Riesenhuber and Poggio [84]. Although their network hard-coded the effect of temporal learning, it was a proof of concept that hierarchical systems generalize well. Although they have started analyzing the hierarchy from a theoretical perspective [90], their approach is in terms of defining a distance measure in the hierarchy. In [79], Poggio clearly identifies a connection between generalization and hierarchy and argues that generalization will depend more on the architecture than on the detailed nature of the algorithms.

In contrast, some of the recent work on Deep Belief Networks has emphasized clever learning algorithms for multi-layered networks [48]. However, it is unclear what advantages these networks offer in terms of sample complexity of learning and in terms of generalization. Although Deep Belief Networks have been applied to temporal sequences as well [94], this was accomplished without any constraints for discovering slow features. Therefore, it is not clear that those networks learned spatial and temporal hierarchies similar to that of HTMs.

Hierarchical Hidden Markov Models (HHMMs) [29], are another class of hierarchical models that share some properties with HTMs. However, these models are hierarchical in just one dimension – usually the temporal dimension. Therefore, those models do not get the advantage of the compositional generalization provided by HTMs. In [77], Karl Pfleger described the construction of sequence hierarchies with bottom-up chunking. Although Pfleger describes the hierarchy as compositional, it is not compositional in the sense described in this chapter. Similar to HHMMs, Pfleger's hierarchies are only in the temporal dimension.

Memory based learning was first described by Albus in his CMAC architecture for motor learning and control [1]. Inspired by the success of this memory-based approach, Lin and Vitter explored a theoretical framework for memory-based learning [62]. However, their examination was restricted to single level systems and did not include the effect of temporal learning. Consequently, the generalization properties of these mechanisms were limited to nearest neighbor search, space decomposition techniques and clustering [62]. In contrast, Poggio and Girosi explored a mechanism where approximations are made using basis functions instead of stored prototypes [80]. However, the generalization mechanisms in these techniques ended up being a modified form of nearest neighbor search with spatial decomposition. Hierarchical vector quantization [14] is related, but the hierarchy in this case helps only in the reduction of storage and computation and does not result in any interesting generalization properties.

A number of researchers [109, 7, 91] have built hierarchical networks with slowness in time with increasing levels of hierarchy as the guiding principle to build invariant representations. This body of work comes closest to the hierarchies described in this chapter. However, their networks were built as a proof of concept for biological systems. These models did not have an associated generative model and did not use Bayesian belief propagation for inference. Moreover, the generalization properties that arise from the interplay of spatial and temporal hierarchies were not analyzed.

3.10 Speculations on the parallels between the organization of the world and the organization of the neocortex

The No Free Lunch (NFL) theorems [110, 50] for learning teach us that no learning algorithm has an inherent superiority over another learning algorithm for all learning problems. If an algorithm is superior for a particular problem, it is only because the algorithm exploits assumptions that are suitable for that problem. On the other hand, results from neuroscience tell us that the neocortex might be using the same algorithm for different tasks like visual, auditory and somatosensory perception [43]. It is also well known that the neocortex is organized as a hierarchy [28]. A plausible explanation for these observations is that data from different domains and sensory modalities, despite their apparent superficial differences, can have the same underlying statistical properties. This could be because the data generation mechanisms of the world have an underlying hierarchical structure due to the laws of physics and self-organization. The neocortex, through evolution, might have discovered this fact and could be exploiting it to learn efficient models for hierarchically structured data. If that is the case, then the secrets behind the efficiency of hierarchical learning can be unlocked by studying the organization of the world. In this section, we present some parallels between the organization of some systems in the world and the organization of the neocortex.

3.10.1 Hierarchical organization of systems in the world

Natural and man-made dynamic systems tend to have a nested multi-scale organization. In such systems, there are large-scale system-level variables and small-scale sub-system-level variables. Often, the larger system level variables are slower compared to the smaller sub-system level variables [105]. If we consider weather systems, for example, winter is a high-level system variable that affects the whole country. During winter, there will be local system variables like snow storms that affect the weather in, say, Minnesota. While the winter lasts for several months, the local variations usually last only for days. The weather system can be thought of as organized as a hierarchy. This hierarchy is spatial as well as temporal. The higher-level variables like winter correspond to large spatial area and slow temporal variations. Local variations like snow storms are faster compared to the large-scale variables but influence smaller spatial areas.

It is important to note that the hierarchical organization is in both time and space. The hierarchy in space can be thought of as the construction of a complex system using sub-systems. The sub-systems provide stable re-usable components that can be assembled together to build larger systems. Herbert Simon was among the first to point out this arrangement [89]. According to Simon, the hierarchy in time can be thought of as an ordering of characteristic frequencies of these sub-systems. Different sub-systems will have different characteristic frequencies associated with their dynamics. In general there is mapping between the sub-system ordering and the frequency ordering. It is generally observed that smaller subsystems (small spatial scales) are associated with faster frequencies of vibrations and larger subsystems are associated with slower frequencies of vibrations [74].

The sub-systems at different levels of the hierarchy interact in both directions. State variables at higher-level are not mere averaging of the state variables at lower levels [74]. The higher-level variables provide context to the lower-level variables which control the rate of dynamics of the higher-level variables.

3.10.2 Spatial and temporal hierarchies in the neocortex

It is well established that the visual is organized as an anatomical hierarchy [28]. The structural hierarchy of the primate visual cortex has been well studied. The visual information leaving the retina first enters the cortical area V1 after passing through the lateral geniculate nucleus [45]. In the ventral stream that is generally accepted as responsible for object recognition, information gets passed successively to visual areas V2, V4 and IT. The neurons in region V1 see only a small portion of the visual field. The neurons in region V2 receive their inputs from several neurons in area V1. Therefore, the effective receptive field of a V2 neuron is several times bigger than that of a V1 neuron. In general the receptive field sizes increase as you go up in the hierarchy. A neuron in area IT has a receptive field that is almost equal to the size of the whole visual field. If we present an object as input to the visual cortex, the neurons in area V1 will respond to small local features like oriented lines in that object. The pattern of neuron firings in area IT will correspond to the identity of the object itself. This kind of organization of the neurons gives rise to a structural hierarchy.

The information flow in the visual cortex is not just from bottom to top. Both feed-forward and feedback connections exist between the levels of the hierarchy. The different levels of the hierarchy interact in a recurrent manner [45] with the higher levels providing context for the lower levels [60].

Interspersed with the structural hierarchy of the visual cortex is a temporal hierarchy. To see this, imagine that we slowly move the object presented to the visual cortex, while still keeping the whole object within its receptive field. The neurons in area V-1 will see local features moving in and out of their receptive fields. That is, their responses will change quickly as we move the object. Since the neurons in area IT code for the identity of the object itself, their responses will remain largely stable as the object is moved. This corresponds to a temporal slowness as we ascend in the hierarchy. The idea of a temporal hierarchy is consistent with the idea of a structural hierarchy where the neurons specialize for object identities as you ascend in the hierarchy.

The temporal hierarchy in the visual cortex hasn't received as much experimental verification as the spatial hierarchy. Recently, Hasson et al [42] studied the nature of temporal hierarchies using functional magnetic resonance imaging. They concluded that, similar to the known cortical hierarchy of spatial receptive fields, there is a hierarchy of progressively longer temporal receptive windows in the visual cortex.

Chapter 4

Towards a Mathematical Theory of Cortical Microcircuits

4.1 Introduction

Understanding the computational and information processing roles of the cortical microcircuits is one of the outstanding problems in neuroscience. The microcircuits of the cortex are bewildering in their complexity and anatomical detail. Although enormous progress has been made in the collection and assimilation of data about the physiological properties and interconnectivity of cortical neurons, the data are not sufficient to derive a computational theory in a purely bottom-up fashion.

The theoretical setting of hierarchical Bayesian inference is gaining acceptance as the framework for understanding cortical computation [60, 45, 83, 19]. Tai Sing Lee and David Mumford [60] suggested that algorithms for Bayesian belief propagation might model the interactive feed-forward and feedback cortical computations. They based this suggestion on an understanding of neurophysiological evidence in the light of Bayesian computations. However, they did not propose mechanisms for hierarchical and temporal learning or propose how these computations could be implemented in cortical circuits. In a recent review, Hegde and Felleman pointed out correctly that the "Bayesian framework is not yet a neural model. [Bayesian] framework currently helps explain the computations that underlie various brain functions, but not how the brain implements these computations" [45]. The work in this chapter is an attempt to fill this gap by deriving a computational model for cortical circuits based on the mathematics of Bayesian belief propagation in Hierarchical Temporal Memory (HTM) networks.

The starting point for this work is the Memory-Prediction framework described by Jeff Hawkins in his book entitled *On Intelligence* [43]. In the Memory-Prediction framework, Hawkins provided a plausible computational and biological framework for understanding the operations of the cortex. Hawkins also provided biological instantiations for his theory. Hawkins' theory was a remarkable synthesis and introduced several novel ideas for the computations performed by cortical circuits. However, these instantiations were incomplete and lacked the rigor provided by a mathematical framework. The theory of Hierarchical Temporal Memory, as expressed in [44] and further expanded in the previous chapters of this thesis, was developed by starting with the Memory-Prediction theory and by formalizing it using a mathematical framework. This chapter works back from this formal expression and derives cortical microcircuits by matching the computational specifications of the theory with known biological data. The resultant biological circuit is complete in the sense that it supports the Bayesian computations required for temporal, feed-forward and feedback inference. The various elements of the circuits are also consistent with each other in the sense that they operate under the same set of assumptions and also work together in a hierarchy.

This work differs in many aspects from the prevalent modeling methods in computational neuroscience. In neuroscience, it is common to construct models based on bottom-up information without synthesizing it in to a larger framework. Many models are constructed on an as-needed basis to explain particular phenomena. A model that explains one phenomenon typically does not explain another. Such models typically do not achieve any significant pattern recognition or prediction. In contrast, the cortical circuit models in this chapter are not constructed to explain any particular phenomenon. These circuits do the same computation as the pattern-recognition HTM networks in chapter 2, and follow the principles of hierarchical-temporal learning and generalization established in chapter 3.

Several researchers have proposed models for cortical circuits [38, 37, 65]. Some of these models exhibit interesting pattern recognition properties and some have been used in the explanation of physiological phenomena. However, none of these models are derived from a general Bayesian framework and they do not incorporate the concepts of hierarchical temporal learning that were described in the earlier chapters. To our knowledge, the work in this chapter is the first instantiation of the theory of Bayesian belief propagation and hierarchical and temporal learning and generalization into a biologically plausible cortical circuit. (Partial details of this work have been published earlier [35, 34].)

The circuits derived in this chapter can provide a hypothesis driven framework for examining the neural connectivity. Deciphering the functional connectivity of the cortical circuits is a formidable task and is associated with the perils involved in the reverse engineering of a complex system. Several insights can be obtained by comparing it with the reverse engineering of an electronic circuit. Although a single transistor can function as an amplifier, an amplifier for real world use is seldom constructed using a single transistor. A good construction involves a biasing circuitry, which makes sure that the amplifying transistor works properly despite changing temperature conditions, different device characteristics and feedback instabilities. It is reasonable to expect that a similar situation exists within the cortical sheet where a multitude of neurons are involved in biasing a canonical cortical circuit to function. If the circuit is tested for connectivity when it is not properly biased (for example, when an animal is not in the normal behaving conditions), one could end up missing some important connections and logging some spurious ones. Hence, deciphering the functional connectivity from an increasing amount of anatomical data requires theories about cortical functions and how they map on to anatomy. This chapter can be considered as a contribution in that direction.

As with any other theory, it is expected that the particular instantiation described here will need to be revised as more data is obtained and more aspects of cortical computations like attention, timing and motor action are incorporated. The circuits derived here could act as a basis for such explorations. The biological circuit gives us a mapping from the intermediate values in belief propagation computations to column and lamina specific neural responses. The anatomical evidence for the derived circuits is examined and some of its variations are explored. In addition, the derived circuits are used to model well known physiological phenomena. The phenomenon of neurons in V1 responding to illusory contours, and the phenomenon of higher-level recognition reducing the activity of lower levels are observed in the same network. Modeling of these phenomena can also serve as examples of how to use the derived circuits for hypothesis driven physiological experiments.

This chapter is organized as follows. First, the general organization of the cortex is described and the concepts behind canonical cortical circuits are reviewed (Sections 4.2 and 4.3). Section 4.4 establishes a mapping between the cortical hierarchy and the HTM hierarchy and between cortical regions and HTM nodes. This establishes the context for looking at the computations done within an HTM node. The Bayesian belief propagation equations for the computations in an HTM node are described in section 4.5. The mapping from these equations to biological circuits is established in two stages. In the first stage (section 4.6), an abstract neural implementation of the belief propagation equations is proposed and described in detail. In the second stage (section 4.7), this abstract neural implementation is mapped to a laminar biological cortical circuit by comparing the computational specifications with anatomical data. Section 4.8 provides example applications of this circuit in the modeling of physiological phenomena. Variations and extensions of the derived circuits are discussed in section 4.10. The chapter is concluded with a discussion in section 4.11.

4.2 Neocortex primer and working assumptions

The main purpose of this section is to describe the biological background for the sections that follow. The anatomical features of neocortex are described primarily to establish the working assumptions for the rest of the chapter. Although many features of cortical organization are well understood, these are not without exceptions. Many others are less well understood and remain controversial. We will revisit and discuss some of these assumptions in section 4.11. A good review of the organization of the neocortex can be found in the book entitled *The Computational Brain*, authored by

Churchland and Sejnowski [16].

The neocortex is the seat of intelligence in humans and other mammals. Physically, the neocortex looks like a sheet. This sheet is several millimeters thick and contains several layers of neurons. Sensory and motor processing in the neocortex is topographically organized. This means that, nearby neurons, in general, represent related things. For example, neurons in the visual areas of the cortex V1 are arranged topographically and adjacent neurons have adjacent receptive fields on the retina. Different such regions corresponding to different modalities and different stages of information processing can be identified on the neocortical sheet [104].

Cortical areas have a laminar organization. This means that the neo-cortical sheet is constructed from many different layers of neurons. Neurons in a given layer of the neo-cortical sheet conform to a highly regular pattern of where each connects to and from where each receives connections. In general, the neo-cortical sheet is considered as having six different laminae (layers). Although several exceptions can be found, the six layers provide a commonly agreed upon framework with which to explore models of cortical circuits [22]. These laminae are numbered 1 to 6. Layer-1 is the layer closest to the skull and farthest from the white matter. Layer-6 is the layer closest to the white matter. Typically, the laminae are described with respect to a horizontal orientation with layer-1 at the top and layer-6 at the bottom, as shown in figure 4.1(B). With this picture in mind, layers 1-3 are typically referred to as the 'superficial' layers or top layers. Layers 5 and 6 are referred to as the deep layers of cortex.

In addition to the 'horizontal' organization seen in the laminae, cortical regions also have a 'vertical' organization. This vertical organization is typically described as cortical columns [69]. The characteristic of this vertical organization is a high degree of commonality between neurons in the same vertical column crossing different laminae. There is a high degree of anatomical connectivity between neurons in the same column. This is also reflected in the physiological response properties of neurons in the same column. The orientation columns in the primary visual cortex typically do not result in sharp boundaries [16]. In higher level cortices like IT, individual columns represent component visual features of an object [99]. Although the columnar organization is generally accepted, its function is still debated [52].

It is now well accepted that the different regions in the visual cortex connect in a hierarchical manner [28]. For example, the primary visual cortex, known as area V1, is the first cortical processing stage for visual signals. The result of the processing of V1 is sent to cortical area V2 and then to cortical area V4. Physically, the connections between different areas of the cortex travel through the white matter or through the cortical sheet itself. It is also known that the connections between different regions of cortex are reciprocal. If area A connects to area B, then area B also sends feedback connections to area A. In fact, the hierarchical positions of different regions of the visual cortex were anatomically established based on the differences in feed-forward and feedback projection patterns [28].

4.3 Canonical cortical microcircuits

Cortical neurons, organized in different laminae and columns, form microcircuits that connect vertically in a column as well as horizontally across columns. Many properties of these microcircuits are stereotypical and remain the same across age, cortical area and species. This suggests that neocortical microcircuits are variations of a common microcircuit template. A major challenge for computational neuroscience is to understand the computational function of lamina-specific synaptic connections in stereotypical cortical microcircuits [40].

There are a large number of anatomical and physiological studies about the intra and inter-laminar connections of neurons. Many researchers have provided summary diagrams for these connections by synthesizing the vast amount of anatomical data. Although highly idealized, these diagrams provide templates from which to develop a detailed theory of cortical microcircuits. Figure 4.2 shows an example of one such cortical circuit template [4]. In this circuit, the excitatory feed-forward pathways derived from anatomical evidence of axonal projection patterns are shown. Other such summary diagrams are available from [97, 98, 22, 23].

The goal of this chapter is to provide a detailed model for the canonical cortical microcircuit based on the theory of hierarchical temporal memory and Bayesian belief

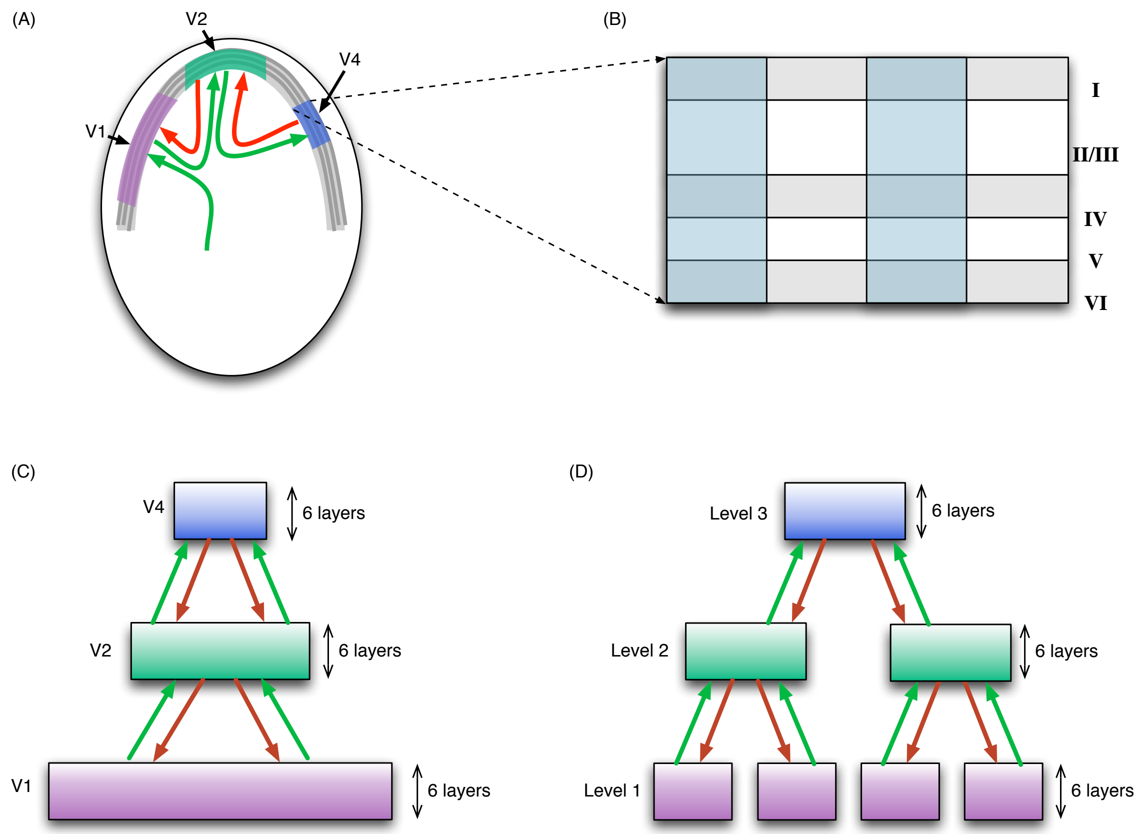


Figure 4.1: (A) Schematic of neocortex inside the skull. The neocortex is a thin sheet of several layers of neurons. Different areas of the neocortex sheet process different information. Three areas – V1, V2 and V4 – are marked on this sheet. These correspond to areas in the visual cortex. Sensory information entering V1 gets processed through multiple stages. The connections between cortical areas are reciprocal. The feed-forward connections are represented using green arrows and the feedback connections are represented using red arrows. (B) A slice of the neo-cortical sheet showing its six layers and its columnar organization. The cortical layers are numbered 1 to 6. Layer-1 is the outer layer (close to the skull) and layer 6 is the inner layer (close to the white matter). (C) Areas in the visual cortex are connected in a hierarchical manner. This diagram shows the logical hierarchical arrangement of the areas which are physically organized as shown in (A). (D) An HTM network that corresponds to the logical cortical hierarchy shown in (C)

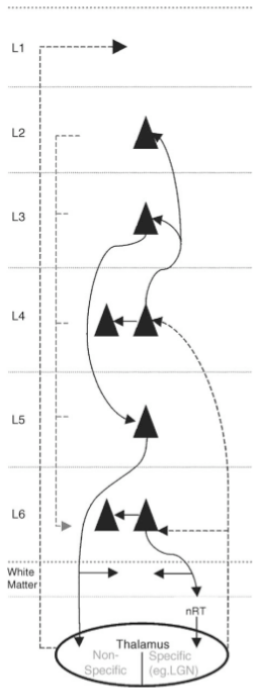


Figure 4.2: An example of a summary diagram for the cortical circuits. Image source: Bannister [4]

propagation. At this point, the biological background and working assumptions necessary for understanding such a mapping have been reviewed. The next section will describe how different organizational levels of an HTM map to different organization properties of the neocortex.

4.4 Mapping between neocortex hierarchy and HTM network hierarchy

The goal of this chapter is to match the theory and operations of hierarchical temporal memory to the organizational details of neocortex at different levels ranging from the hierarchy to the detailed microcircuitry. This section establishes the high-level mapping from the cortical hierarchy, cortical areas and cortical columns to HTM hierarchy, set of HTM nodes, and coincidence patterns in HTM nodes.

Logically, an idealized version of the hierarchical connectivity of areas of the visual cortex can be drawn as shown in figure 4.1(C). In the logical hierarchy shown in figure 4.1(C), the areas V1, V2 and V4 are shown as single contiguous regions consisting of several cortical columns. However, it is clear that at the lower levels of the hierarchy, multiple columns can be ‘active’ independent of one another. For example, in V1, the left part of V1 could be seeing a vertical line while the right side of V1 could be seeing a horizontal line. Since these hypotheses are not mutually exclusive, the column representing vertical line will be active on the left side of V1 while the column representing horizontal line will be active on the right side of V1 [43]

An area of cortex can be thought of as encoding a set of patterns and sequences in relation to the patterns and sequences in regions hierarchically above and below it. The patterns correspond to the coincidence-patterns in an HTM node and the sequences correspond to the Markov chains. An area like V1 can encode several mutually non-exclusive patterns. In an idealized model, these sets of patterns can be partitioned into several sets where the patterns belonging to the same set are mutually exclusive and the patterns belonging to different sets are mutually non-exclusive. (Mutually exclusive means that the strengthening of one hypothesis decreases the strength for another.)

A set of mutually exclusive patterns can be represented by a random variable. In our idealized version, a cortical column represents a particular value of this random variable. At every time instant, the activity of a set of cells in a column represents the probability that a particular coincidence-pattern is active. The feed forward and feed back connections to a set of cortical columns carry the belief propagation messages. Observed information anywhere in the cortex is propagated to other regions through these messages and can alter the probability values associated with the hypotheses maintained by other cortical-column sets.

An HTM Node, in its simplified form, encodes a set of mutually exclusive patterns and Markov chains. A region of cortex that has several patterns simultaneously active will be implemented using several HTM nodes. Figure 4.1(D) shows the HTM implementation of the logical cortical hierarchy shown in 4.1(C). In this mapping, the area V1 is implemented using 4 HTM nodes while area V2 is implemented using 2 HTM nodes. Typically, the number of non-exclusive patterns that needs to be maintained decreases as you ascend in the hierarchy. Therefore, higher-level cortical regions can possibly be modeled using a fewer number of HTM nodes. Note that this is a representative diagram. A cortex-equivalent implementation of V1 and V2 could require several thousand HTM nodes for each cortical area.

4.5 Belief propagation in HTM nodes

Intuitively, a learned HTM node can be thought of as having memorized a set of patterns (the coincidence-patterns) and a set of sequences over patterns (the Markov chains). For example, the different coincidence-patterns in an HTM node that corresponds to part of area V1 can represent different translations of a vertically oriented edge and different translations of a horizontally oriented edge. The Markov chains in that node can correspond to a Markov chain that represents the sequence of movements of the vertical edge and another Markov chain that represents the sequence of movements of the horizontal edge.

In general, the messages that come into an HTM node from its children are the

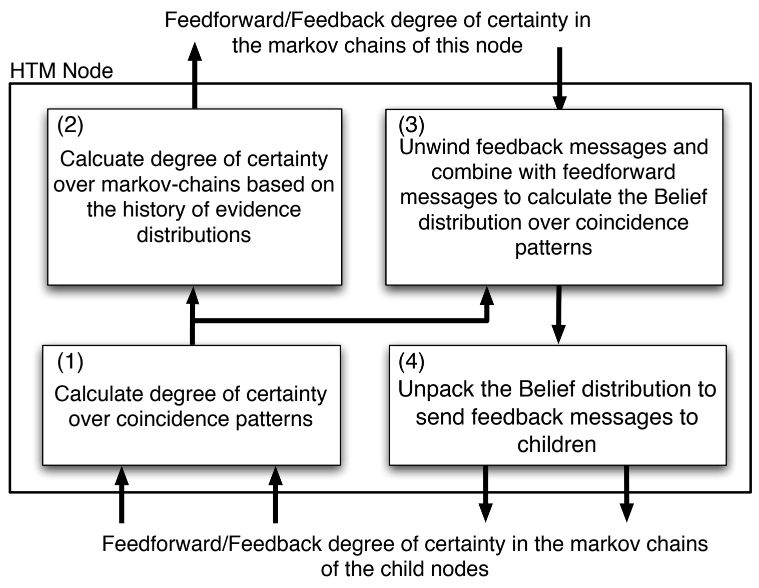
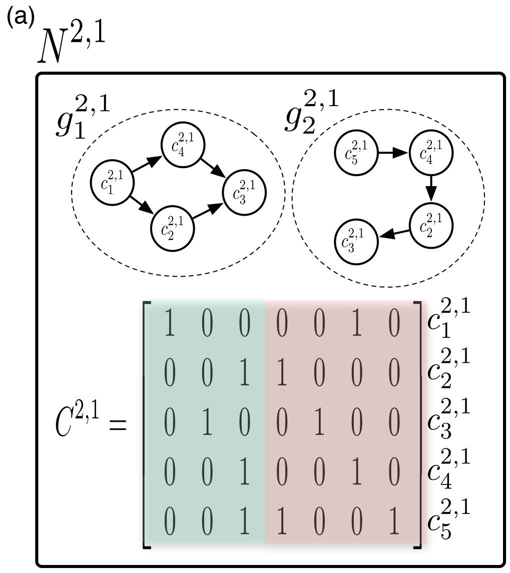

Feedforward/Feedback degree of certainty in the markov chains of this node
HTM Node
(2) Calcuate degree of certainty over markov-chains based on the history of evidence distributions
(3) Unwind feedback messages and combine with feedforward messages to calculate the Belief distribution over coincidence patterns
(1) Calculate degree of certainty over coincidence patterns
(4) Unpack the Belief distribution to send feedback messages to children
Feedforward/Feedback degree of certainty in the markov chains of the child nodes

Figure 4.3: An intuitive depiction of the inference operations of an HTM node.

children's degree of certainty over their Markov chains. The node converts these messages to its own degree of certainty over its coincidence-patterns. Based on the history of messages received, it also computes a degree of certainty in each Markov chain. This is then passed up to the higher level node. What the node receives from the parent is the parent's degree of certainty over this HTM node's Markov chains. This is then 'unwound' in a step-by-step manner to find the top-down probability distribution over coincidence patterns. The top-down distribution over coincidence-patterns is then 'unpacked' to derive the node's degree of certainty over its child node Markov chains. Figure 4.3 represents this intuitive picture.

4.5.1 Belief propagation equations for HTM nodes

Table 4.1 summarizes the computation of belief propagation messages in an HTM node. We will now describe the notation and meaning of these equations using the reference HTM node shown in figure 4.4. Detailed derivations of these equations are given in appendix A.



$$C^{2,1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{array}{c} c_1^{2,1} \\ c_2^{2,1} \\ c_3^{2,1} \\ c_4^{2,1} \\ c_5^{2,1} \end{array}$$

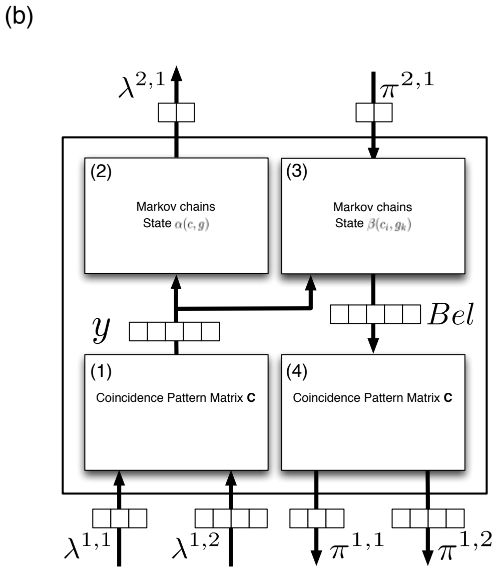

(b)
$\lambda^{2,1}$
$\pi^{2,1}$
(2) Markov chains State $\alpha(c,g)$
(3) Markov chains State $\beta(c_i,g_k)$
y
Bel
(1) Coincidence Pattern Matrix **C**
(4) Coincidence Pattern Matrix **C**
$\lambda^{1,1}$
$\lambda^{1,2}$
$\pi^{1,1}$
$\pi^{1,2}$

Figure 4.4: (a) Structure of the reference node with 5 coincidence patterns and 2 markov chains. This is an HTM node that has finished its learning process. It is assumed that this is a node at level-2 of a network and is labeled as $N^{2,1}$. This node has 2-children. Child 1 has 3 markov chains and child-2 has 4 markov chains. Each coincidence pattern represents a co-occurrence of the markov-chains of the children. The portions of the coincidence pattern coming from the first and second child are shown in different shades. (b) Information flow in the reference node for the computation of the belief propagation equations shown in table 4.1. The rectangles inside the node are processing units for the equations in the rows corresponding to the number displayed in each rectangle. We will use 'feed-forward' or 'bottom-up' to qualify messages received from children and messages sent up to the parent of this node. We will use 'feedback' or 'top-down' to qualify messages received from the parent and messages sent to the child nodes of this node. The node shown in figure has two bottom-up input messages corresponding to the two children and has two top-down outputs which send messages to these children. The arrows show vectors of inputs, outputs and intermediate computational results. The number of components of each vector is represented using an array of rectangular boxes placed on these arrows.

In these equations, the coincidence-patterns are referred to using c_i's and the Markov chains are referred to using g_i's. The HTM node shown in figure 4.4(a) contains 5 coincidence-patterns and 2 Markov chains. The transition probability matrix of the Markov chain g_r is denoted by $P(c_i(t)|c_j(t-1), g_r)$. This term appears in equations 4.3 and 4.6. Each coincidence-pattern in the node represents a co-occurrence of the temporal groups from its children. These are represented implicitly in equations 4.1 and 4.8.

The node receives feed-forward messages from its children and sends feed-forward messages to its parent. The feed-forward messages are denoted by $\lambda^{source\ node\ index}$. Similarly, the node receives feedback messages from its parent and sends feedback messages to its child nodes. These messages are denoted by $\pi^{destination\ node\ index}$. The equations shown in table 4.1 describe how the output messages are derived from the input messages. From the view point of the node, the feed-forward messages carry information about the *evidence* from below. Evidence from below at any time point t is denoted by ${}^{-}e_t$. Similarly evidence from the parent is denoted by ${}^{+}e_t$.

Equation 4.1 describes how the node calculates its likelihood of coincidence patterns, from the messages it gets from the children. The bottom-up likelihood of coincidence-pattern c_i at time t is represented by $y_t(i) = P({}^{-}e_t|c_i(t))$. The likelihood of each coincidence pattern is calculated as the product of the message components corresponding to that coincidence pattern.

In equation 4.2, the bottom-up likelihood of Markov chain g_r at time t is denoted by $P({}^{-}e_0^t|g_r(t))$, where the term ${}^{-}e_0^t$ represents the sequence of bottom-up evidences from time 0 to time t. This reflects that the likelihood of the Markov chains depends on the sequence of inputs received by the node. The variables α and β defined in equations 4.3 and 4.6 are state variables that are updated in a recursive manner at every time instant. These are dynamic programming variables, each variable defined over all pairwise combinations of coincidence-patterns and Markov chains. For example, $\alpha_t(c_i, g_r)$ is value of the feed-forward dynamic programming variable at time t corresponding to coincidence c_i and Markov chain g_r. In equations 4.3 and 4.6, the states are updated every time step by passing the state from the previous time step through the Markov transition matrices and by combining them with bottom-up/top-down evidence.

A detailed derivation of these equations can be found in appendix A. We examine these equations in more detail in the next section as we consider how to implement them using neuron-like elements.

4.6 Neuronal implementation of HTM belief propagation

This section describes an implementation of the HTM belief propagation equations using neuron like elements. The implementation will be described with respect to the reference HTM node in figure 4.4 . Two different views of this reference node are shown. In the first view, the coincidence patterns and the Markov chains are shown explicitly. The reference node has two children - child 1 with three Markov chains and child 2 with four Markov chains. Therefore, a coincidence pattern stored in the node is a 7-element vector. Both Markov chains in the reference node have 4 states each.

The second view shows the message flow within this node for the implementation of the belief propagation equations. The rectangles inside the node correspond to the equations shown in table 4.1 and are numbered according to the row number of the table. The neuronal implementation of the equations in table 4.1 is described in the following subsections. The subsection numbers below correspond to the table row numbers.

4.6.1 Calculating the bottom-up likelihood of coincidence-patterns

The bottom-up input to the HTM node is the feed-forward output messages from its children. These output messages carry the information about the degree of certainty of the child nodes in their Markov chains/temporal groups. Each message is a vector of length equal to the number of Markov chains in the corresponding child. The

1) Calculate likelihood over coincidence-patterns.	$$y_t(i) = P(^-e_t \| c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j}) \quad (4.1)$$ where coincidence-pattern c_i is the co-occurrence of $r_i^{m_1}$'th Markov chain from child 1, $r_i^{m_2}$'th Markov chain from child 2, $\cdots$, and $r_i^{m_M}$'th Markov chain from child M.
2) Feed-forward probability over Markov chains (temporal groups) using dynamic programming	$$\lambda_t^k(g_r) = P(^-e_0^t \| g_r(t)) \propto \sum_{c_i(t) \in C^k} \alpha_t(c_i, g_r) \quad (4.2)$$ $$\alpha_t(c_i, g_r) = P(^-e_t \| c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t) \| c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r) \quad (4.3)$$ $$\alpha_0(c_i, g_r) = P(^-e_0 \| c_i(t=0))P(c_i(t=0) \| g_r) \quad (4.4)$$
3) Calculate the belief distribution over coincidence patterns	$$Bel_t(c_i) \propto \sum_{g_r \in G^k} P(g_r \| ^+e_0)\beta_t(c_i, g_r) \quad (4.5)$$ $$\beta_t(c_i, g_r) = P(^-e_t \| c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t) \| c_j(t-1), g_r)\beta_{t-1}(c_j, g_r) \quad (4.6)$$ $$\beta_0(c_i, g_r) = P(^-e_0 \| c_i(t=0))P(c_i(t=0) \| g_r, ^+e_0) \quad (4.7)$$
4) Calculate the messages to be sent to child nodes.	$$\pi^{child}(g_m) \propto \sum_{i \forall i} I(c_i)Bel(c_i) \quad (4.8)$$ where $$I(c_i) = \begin{cases} 1, & \text{if } g_m^{child} \text{ is a component of } c_i \\ 0, & \text{otherwise} \end{cases} \quad (4.9)$$

Table 4.1: Belief propagation equations for an HTM node

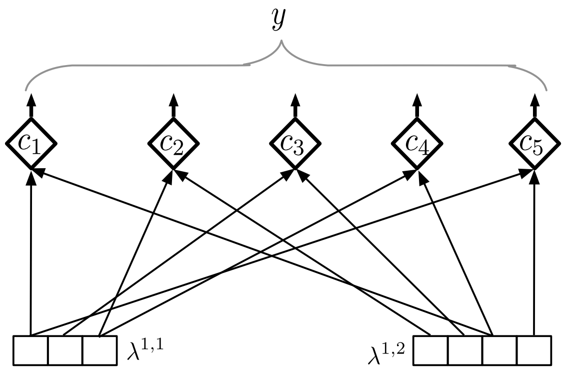


Figure 4.5: Circuit for calculating the bottom-up probability over coincidence-patterns. Coincidence-pattern neurons are represented by diamond-shapes. The inputs to the circuit are the messages from the children – these are denoted by $\lambda^{1,1}$ and $\lambda^{1,2}$. The output of the circuit is y, as calculated by equation 4.1. The input connections to the neuron represent its coincidence-pattern. For example, c_2 is the co-occurrence of Markov chain 3 from the left child and Markov chain 1 from the right child. The probabilities are calculated by a multiplication of the inputs to each neuron.

likelihood of coincidences is derived from these input messages according to equation 4.1. This operation is performed by the rectangle marked 1 in figure 4.4(b).

Figure 4.5 shows neuronal implementation of this calculation for the reference HTM node. Each stored coincidence pattern is represented by a neuron. For example, the coincidence pattern c_1 is represented by the neuron labeled c_1. The pattern corresponding to the co-occurrence is stored in the connection this neuron makes to the messages from the child input nodes. The neuron c_1 has connections to the first position of the message from the first child and the second position of the message from the second child. Note that this corresponds to first row of the coincidence-pattern matrix $C^{2,1}$ in figure 4.4(a). Similarly, it can be verified that the other neurons in figure 4.5 match up the corresponding rows of the coincidence-pattern matrix $C^{2,1}$.

It is the output of these neurons that represent the result of the calculation performed by equation 4.1. Each neuron calculates its output by multiplying it inputs. For example, the output of neuron c_3 is proportional to $\lambda^{1,1}(2) \times \lambda^{1,2}(3)$. The output, denoted by y in figure 4.4(b), is a vector of 5 components – one component corresponding to each coincidence pattern. This vector represents the degree of certainty

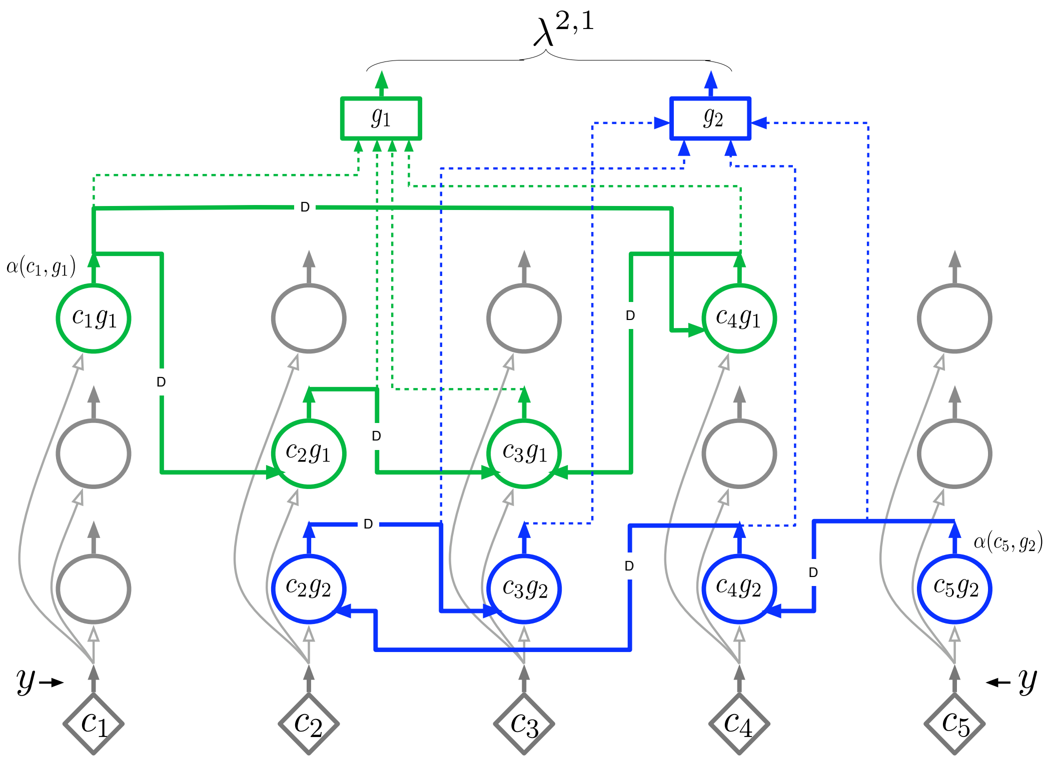


Figure 4.6: The circuit for calculating the likelihoods of Markov chains based on the sequence of inputs. This circuit implements the dynamic programming equations 4.3. There are three types of neurons in this picture. See text for details

of the node in its currently active coincidence pattern, based on the messages received from its child nodes.

4.6.2 Efficiently calculating the bottom-up likelihood of Markov chains

The next step in the computation of feed-forward messages is the calculation of the degree of certainty of the HTM node in each of its Markov chains. This computation is done by the rectangle marked 2 in figure 4.4(b). The input for this calculation is the probability distribution over coincidences – the 5-component vector (y) that was calculated in the previous section. The output depends not only on the current input

y, but also on the sequence of inputs received up till this time point. The output is a likelihood distribution over the Markov chains of the node. For the reference node, this is a vector with 2 components as shown in figure 4.4(b).

For the belief propagation calculation, the quantity that we need to calculate is $P(^-e_0, ^-e_1, \cdots, ^-e_t|g_i)$ for each Markov chain g_i where $^-e_0, ^-e_1, \cdots, ^-e_t$ represent the bottom-up evidence distributions received from time 0 to time t. This forward degree of certainty in a Markov chain depends on the sequence of messages that the node has received from its children. Done naively, computation of this quantity requires computing the probability of all possible paths from time 0 to time t. To calculate $P(^-e_0^t|g_i)$ efficiently, all the past evidence needs to be collapsed into a state variable that can be updated recursively every time instant. This is done using a technique called dynamic programming [5, 53, 9]. Equation 4.3 represents this dynamic programming computation. The derivation of this equation is described in appendix A.

Now we will show that this equation has a very efficient, simple and intuitive implementation using neuron-like elements.

Consider the neuron circuit shown in figure 4.6. There are three kinds of neurons in this circuit. The diamond-shaped neurons at the bottom are the same as the neurons that were described in the previous section. The outputs y of these neurons are one of the inputs to the 'circle' neurons as shown in figure 4.6. The circle neurons implement the sequence memory of the Markov chains in an HTM node. Figure 4.6 shows the implementation of the two Markov chains of the reference node in figure 4.4(a). The connections between the circle neurons implement the transition probabilities of the Markov chain. The 'axons' between these neurons encode a one time-unit delay. This means that the output of a circle neuron is available at the input of the circle neuron that it connects to after one time step.

Note that all the circle-neurons in the same column have the same bottom-up input. They are all driven by the same pattern-likhelihood neuron from below. Each column, considering only bottom-up input, can be thought of as representing one particular coincidence pattern. But the same coincidence pattern can be part of different Markov chains. The circle neurons represent the coincidence pattern in the context of different Markov chains. For this reason, we have labeled the circle neurons as c_1g_1, c_3g_2 etc. In this notation, c_1g_1 represents the coincidence pattern c_1 in the context of the Markov chain g_1.

In addition to the bottom-up input, these circle neurons also have a 'lateral' input. These come from other circle-neurons in the same Markov chain. It is this lateral connection that specifies the meaning of a neuron in a sequence. For example, the circular neuron c_1g_1 will be activated only in the context of Markov chain g_1, whereas the circular neuron c_3g_2 will be activated only in the context of Markov chain g_2.

Each circle neuron in this circuit does the same computation. Each circle-neuron has a bottom-up input that represents the strength of activation of the coincidence pattern that is being represented by that neuron. In the figure, the bottom-up inputs are represented by white arrow heads. Each neuron also has a set of lateral inputs. These lateral inputs are weighted by synapse strengths. Each neuron calculates its output by taking the weighted sum of its lateral inputs and then multiplying that quantity by the bottom-up input. Let us denote the output of a circle neuron using $\alpha(\textit{coincidnce number}, \textit{markov chain number})$. For example, $\alpha(c_3, g_2)$ is the output of circle neuron c_3g_2. With this, the output of any circle neuron c_ig_r is calculated as

$$\alpha_t(c_i, g_r) = y(i) \sum_j w(i,j) * \alpha_{t-1}(c_j, g_r) \tag{4.10}$$

That is, the output of the a circle neuron at any time is the weighted sum of the outputs of the neurons in the same Markov chain at the previous time step multiplied by the current bottom-up activation. The axonal delays in the neuron circuits ensure that it is the previous outputs of the circle neurons that are used to calculate the current output in the above equation.

Examine equation 4.3. Note that what is described above is exactly this equation if we replace $w(i,j)$ by $P(c_i|c_j, g_r)$. Therefore, the circle-neuron circuits shown in figure 4.6 implement equation 4.3 and the weights on the lateral time-delayed connections correspond to the transition matrix entries in each Markov chain.

Now consider the third kind of neurons – the 'rectangle' neurons – in figure 4.6. The rectangle neuron marked g_1 receives its inputs from the outputs of all the circle neurons in the Markov chain g_1. In general, the rectangle neurons pool the outputs of all the coincidence-pattern neurons in the context of a Markov chain. Therefore, in a node, there will be as many rectangle neurons as there are Markov chains. The input connections to these neurons are instantaneous – they do not have a delay. At any time point, the output of a rectangle neuron is calculated as the sum (or maximum) of the inputs to that neuron.

Note that the operation of the square neurons correspond to pooling over the activations of all the circle neurons of the same Markov chain. It is easy to verify that this is the operation involved in the calculation of the message this node sends to its parent according to equation 4.2. The concatenated outputs of the square neurons is the message $\lambda^{2,1}$ that this node sends to its parent. As noted in figure 4.4(b), this message is a vector of two components, corresponding to the two Markov chains in the reference node in figure 4.4(a). This completes the description of the neuronal implementation of equations in the second row of table 4.1 and of the operations performed by the rectangle marked (2) in figure 4.4(b).

4.6.3 Calculating the belief over coincidence patterns

An HTM node calculates its degree of belief in a coincidence pattern by combining bottom-up, top-down and temporal evidences according to the equations on the third row of table 4.1. This corresponds to the operations of the rectangle marked (3) in figure 4.4(b). The top-down input to the node is a vector of length equal to the number of Markov chains, two in this case, of the node. The output of this computation is the belief-vector over the coincidence-patterns – in this case, a vector of length 5.

The belief calculation, described in equation 4.5, has almost the same form of the forward dynamic programming equations 4.3. The state variable β has the same form as the state variable α and a very similar update equation. The only difference between these two is the multiplication by a top-down factor $P(g_k|^{+}e_0)$ in the belief calculation equations. Therefore, the neuronal implementation of the dynamic programming part of the belief calculation equation is very similar to that of the forward dynamic programming variable α. This implementation is shown in figure 4.7. The

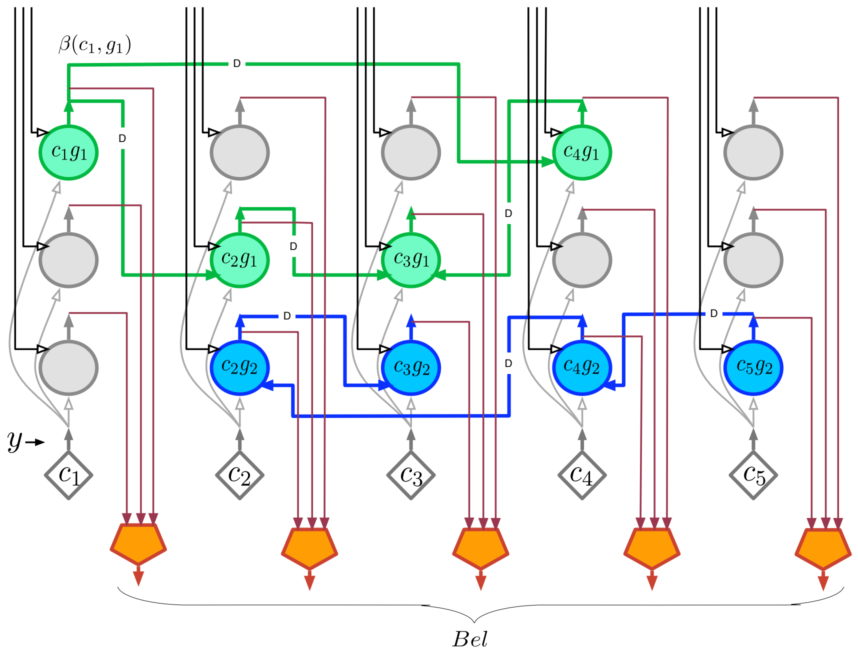


Figure 4.7: Circuit for calculating the belief distribution over coincidence patterns. The belief is calculated by integrating the sequence of bottom-up inputs with top-down inputs according to equation 4.5. See text for details.

filled-circle neurons correspond to the circle-neurons in the forward calculation. Note that, in contrast to the circle neurons in figure 4.6, the filled-circle neurons also have a top-down multiplicative input that corresponds to $P(g_k|^{+}e_0)$.

In the place of the rectangle neurons that calculated the $\lambda^{2,1}$ message in figure 4.6, we have pentagon neurons in figure 4.7. The pentagon-neurons in figure 4.7 are the belief neurons. These neurons pool over the activities of the same coincidence neuron in different Markov chains to calculate the belief value for each coincidence pattern. This operation corresponds to the $\sum_{g_k \forall k}$ operation in equation 4.5. Note that the operation of the pentagon neuron is different from that of the rectangle neuron in figure 4.6. The rectangle neuron pools over different coincidence patterns in the same Markov chain. The pentagon neuron pools over the same coincidence-pattern in different Markov chains.

Incorporating duration models in belief calculation

One of the drawbacks of the equations in section 4.5 is the lack of explicit duration models associated with the states of Markov chains. As expressed in the equations, the Markov chains are forced to change their state with every time step. Self-loops in the Markov chains will lead to an exponential duration probability density that is inappropriate for most physical signals [81].

Several techniques exist for incorporating explicit duration models into Markov chains [61, 81]. We assume that durations are signaled to a node by an external timing unit that determines the rate of change of the signals using some system level measurements. This means that the belief calculation will have two components. The component that we described above calculates the belief distribution without considering when exactly that belief distribution is going to be active. The second component, the external timing signal, determines when the belief distribution is going to be active without regards to the nature of the distribution itself.

Figure 4.8 shows the arrangement of this circuit where the outputs of the 'rounded rectangle neurons' are looped through an external variable delay unit. The rounded rectangle neurons act as a gate that opens only when a timing signal and a belief value are both available at its inputs. The activation of these neurons trigger the

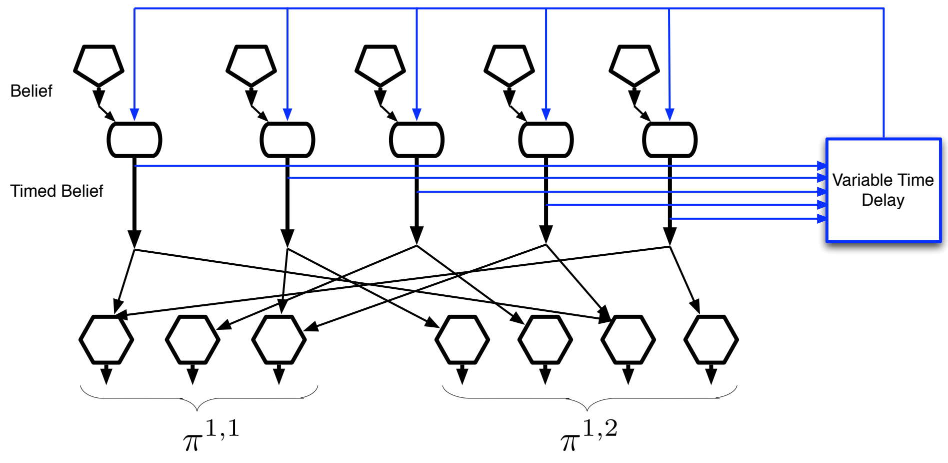


Figure 4.8: The circuit for the timing of the belief distribution and for computing the messages to be sent to children according to equation 4.8. The two sets of hexagonal neurons correspond to the two children of the reference node. See text for more details on the operation of this circuit.

next timing cycle.

4.6.4 Calculating the messages that are to be passed to child nodes

The step that remains to be explained is the conversion of the belief messages to the messages that a node sends to its children. This step is described by equation 4.8 and corresponds to the operations performed by the rectangle marked (4) in figure 4.4(b). The input for this operation is the belief vector. The outputs are the π messages that are sent to the child nodes. A message is sent to each child and the message describes the degree of certainty this node has about the child nodes' Markov chains.

Figure 4.8 shows how this equation can be implemented using neurons. The pentagon neurons shown in this figure are the same as the belief neurons we discussed in the previous section. The outputs of the pentagon neurons are gated through the timing loop using the rounded rectangle neurons. The outputs from these rounded rectangle neurons are fed to 'hexagonal neurons'. Figure 4.8 shows two sets of hexagonal neurons corresponding to the two child nodes of this node. Each hexagonal neuron corresponds to a Markov chain of the child node. The left child node has 3 Markov chains and the right child node has 4 Markov chains. The outputs of these hexagonal neurons are the messages that are sent to the respective children.

The connections between the rounded rectangle neurons and the hexagonal neurons encode the constituents of coincidence patterns. For example, the first rounded-rectangle neuron is connected to the first Markov chains corresponding to both children. This is because the coincidence pattern c_1 is defined as the co-occurrence of the first Markov chain from the left child and the first Markov chain from the right child. The hexagonal neurons calculate their outputs as a sum of their inputs. This operation corresponds to equation 4.8.

The operation of the hexagonal neurons shown in figure 4.8 can be thought of as the reverse of the operations performed by the diamond neurons that were described in figure 4.5. The weights on the inputs to both these kinds of neurons define the coincidence-patterns. In the case of the diamond neurons, they are calculating the probability over coincidences from the probability distribution over Markov chains from each child. The hexagonal neurons are doing the reverse – they are calculating the probability distributions over the Markov chains from each child from the probability distribution over coincidence patterns.

4.7 A detailed proposal for the computations performed by cortical layers

In section 4.3, we gave an overview of the laminar and columnar arrangement of cortical circuits. We also described the correspondence between the HTM node and cortical areas. We then set out on a goal to provide a detailed model for the cortical circuits. We are now at that point. In this section, we propose a detailed computational model for the laminar cortical microcircuits.

Anatomical data give us important constraints on input/output layers, intra/inter-laminar connections and placement of cell bodies and dendrites. The HTM belief propagation equations provide computational specifications. The combination of the anatomical constraints and the computational specifications provides a reasonable framework to develop a detailed model of the cortical circuit. Assignment of a particular function to a particular layer imposes constraints on what functions can be performed by another layer. The challenge is to find an organization that is self-consistent in the sense that it implements the belief propagation equations while conforming to the constraints imposed by biology.

Our working hypothesis can be stated simply: The cortical microcircuits are implementing the HTM belief propagation equations described in table 4.1. The neuronal implementation of these equations was described in the previous section. Under the assumption that the cortical circuits are implementing these equations, what remains to be explained is how the neuronal implementation of the previous section is physically and spatially organized in the layers and columns of cortical microcircuits. This is accomplished by comparing the abstract implementations from the previous section with anatomical data.

Our proposal for the function, connectivity and physical organization of cortical layers and columns is shown in figure 4.9. This figure corresponds to the laminar and columnar cortical microcircuit implementation of the belief propagation equations for the reference HTM node in figure 4.4. Figure 4.9 was created by arranging the neurons of the abstract neuronal implementation of HTM belief propagation into columns and laminae in such a way that the resultant circuit matched most of the prominent features of anatomical data. In the following sections we will de-construct this picture and examine the anatomical and physiological evidences for the specific proposals. This will also illuminate the process that we went through to arrive at the circuit shown in figure 4.9.

The circuits in figure 4.9 provide an exemplar instantiation of the Bayesian computations in laminar and columnar biological cortical circuits. Several plausible variations and exceptions of this circuit can be found because of the degrees of freedom in the implementation of the belief propagation equations and because of the paucity

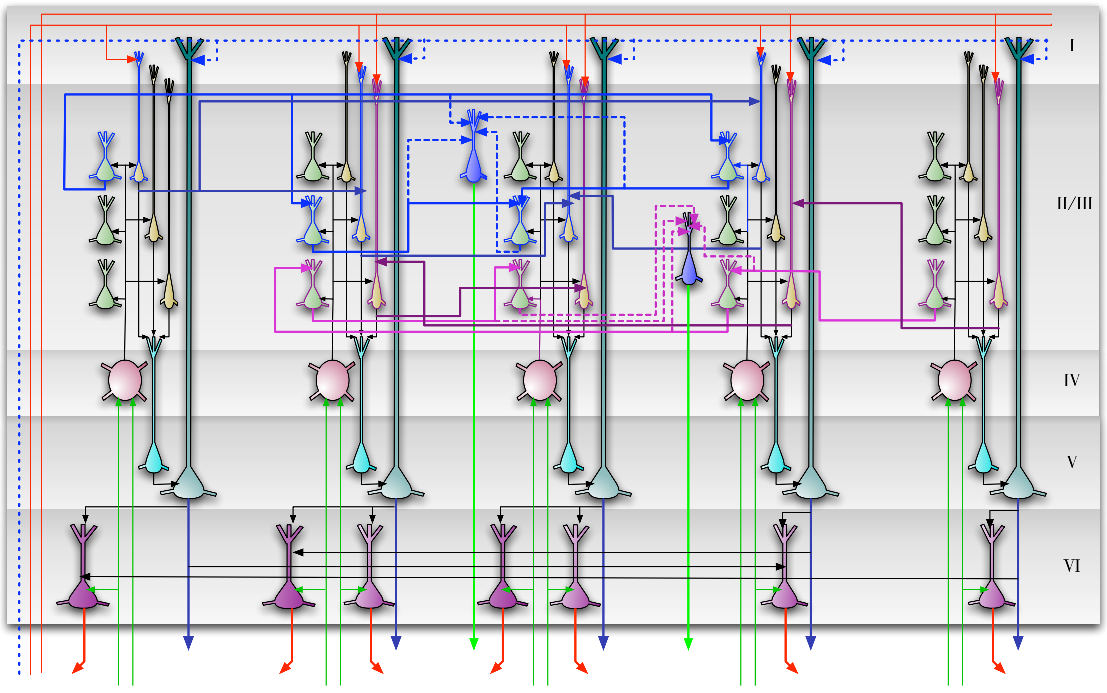


Figure 4.9: A laminar biological instantiation of the Bayesian belief propagation equations used in the HTM nodes. The circuit shown here corresponds exactly to the instantiation of the reference HTM node shown in figure 4.4. The 5 'vertical columns' in the circuit correspond to the 5 coincidence patterns stored in the reference node. Layers 1 to 6 are marked according to the standard practice in neuroscience. Emphasis is given to the functional connectivity between neurons and the placement of the cell bodies and dendrites. Detailed dendritic morphologies are not shown. Axons are shown using thin arrow-tipped lines. Feed-forward inputs and outputs are shown using green axons and feedback inputs and outputs are shown using red axons. Whether an axon is an input or output can be determined by looking at the direction of the arrows. The blue axons entering and exiting the region represent timing-duration signals. 'T' junctions represent the branching of axons. However axonal crossings at 'X' junctions do not connect to each other. Inter-columnar connections exist mostly between neurons in layer 2/3 and between layer-5 cells and layer-6 cells. The inter-columnar connections in layer 2/3 that represent sequence memories are represented using thicker lines. See text for more details.

of anatomical data. We will tackle some of these exceptions and variations as they come up in the appropriate context.

4.7.1 Columnar organization

The cortical circuit shown in figure 4.9 is organized as 5 columns corresponding to the 5 coincidence-patterns in the reference HTM node that we started with. The neurons in each column represent some aspect of the coincidence pattern that the column represents. For example, the neurons in layer 2/3 represent the coincidence pattern in the context of different sequences, whereas the neurons in layer-6 represent the participation of the coincidence-pattern in the calculation of feedback messages. The 5 cortical columns shown represent a set of 5 mutually exclusive hypotheses about the same input space. For example, these columns can correspond to a set of columns in the primary visual cortex V1 that receives input from a small area of the visual field. The 5 coincidence-patterns can correspond to different orientations of a line. If the receptive field is small enough, the different orientations can be considered mutually exclusive - the activity of one reduces the activity of the other. We saw that this kind of columnar organization is typical in biology [99],[16].

In the idealized cortical column model, each different aspect that needs to be represented about a coincidence pattern is represented using a single neuron each. For example, there is exactly one neuron representing coincidence-pattern 1 in the context of Markov chain 1. This means that there is no redundancy in this idealized cortical representation. Clearly, nothing about the computation or the representation changes if we replicate each neuron in this circuit a few times, while maintaining their connectivity. A coincidence that is represented by a single neuron in our cortical column can be represented by a cluster or laterally interconnected neurons.

The reason why this observation is important is that this might explain the intra-laminar lateral connections between the neurons within the same cortical-column[1]. Figure 4.10(b) shows how an idealized cortical column is replicated three times to provide physical redundancy. In this replicated circuit, the neurons with the same

[1]Since our equations are derived from the logical computation, connections required for physical redundancy are not addressed by it at all.

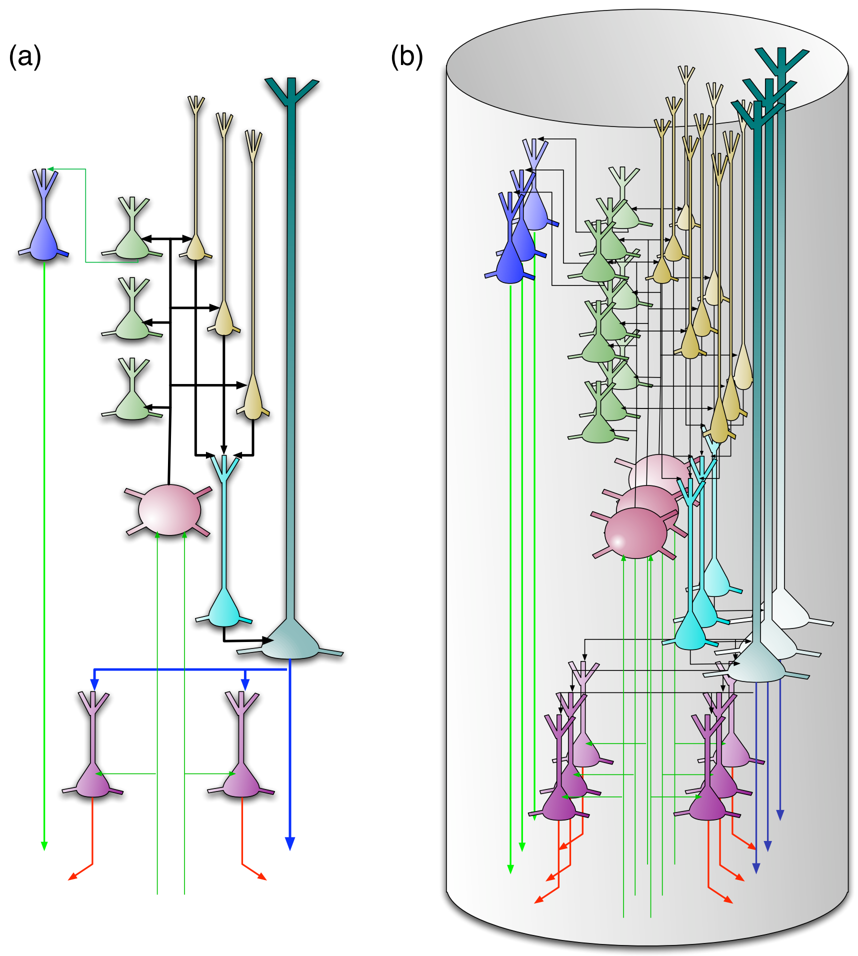


Figure 4.10: (a) An idealized cortical column. This idealization could correspond to a biological mini-column. (b) A cortical column that contains several copies of the mini-column in (a). See text for more details.

meaning can be laterally connected without changing the meaning of the computation. Such lateral connections can provide better redundancy, better transmission ability and even some kind of active transmission ("amplification") [22] along a transmission line[2]. Note also that in such an arrangement, the intensity of response will be greater at the center of the physical column and will decrease towards the periphery. This is observed in real biological columns [16].

Our idealized cortical column in figure 4.10(a) can correspond to what is known as the *mini-columns* of the cortex. Mini-columns are developmental units that contain about 80 to 100 neurons. By the 26th gestational week, the human neocortex is composed of a large number of mini-columns in parallel vertical arrays [70]. It is interesting to note that most of the connections in the idealized vertical column in figure 4.10(a) can be established without any learning. The connections that can be established a-priori are shown in black color in figure 4.10(a). This feature makes our idealized cortical column a good candidate to be a developmental feature like the mini-column. Cortical columns of the type we show in figure 4.10(b) are formed by binding together many mini-columns using common input and short-range horizontal connections [70]. The number of mini-columns per cortical column varies from 50 to 80 [70].

For the rest of the discussion, it is assumed that the short-range, within cortical column, intra-laminar excitatory connections found in biology are explained by the need for physical redundancy as described above. The rest of the discussion will focus on the idealized cortical column and the idealized cortical circuit with no redundancy.

4.7.2 Layer 4 stellate neurons implement the feed-forward probability calculation over coincidence-patterns

The excitatory neurons in Layer-4 of the cortex consist primarily of star-shaped neurons called stellate neurons and pyramidal neurons [96]. Layer-4 is generally accepted

[2]This is analogous to doubling the number of wires to increase the current carrying capacity and using twisted strands of wire instead of a single core

as the feed forward input layer to cortical regions [45]. In the cat primary visual cortex, the outputs from the retina pass through the lateral geniculate nucleus (LGN) of the thalamus and then terminate on layer-4 stellate cells. Most of these connections are known to be proximal to the cell body and can drive the cells. The major projection (output) of layer-4 stellate neurons is to layer-3 cells [22].

We propose that the layer-4 stellate cells are implementing the probability calculation described in equation 4.1 and shown in figure 4.5. This means that layer-4 neurons are coincidence detectors and that the synapses of the layer-4 neurons represent co-occurrence patterns on its inputs.

In figure 4.9, the layer-4 neurons are shown as star-shaped and red in color. The inputs to these neurons are the outputs of lower levels of the cortical hierarchy, possibly routed through the thalamus. It is easy to verify that the connections of these neurons correspond to the 'diamond' neurons in our belief propagation implementation shown in figures 4.5 , 4.6 and 4.7. Note that, in the implementation of the belief propagation equations, shown in figures 4.6 and 4.7, the neurons that calculate the probability distribution on coincidence patterns (the diamond neurons) have only feed-forward inputs. This is in contrast to many other neurons that receive feed-forward, feedback and lateral inputs. In neuroscience, it is accepted that the feedback inputs to a cortical region generally avoid layer-4 [45]. This is consistent with our proposal for the function of layer-4 neurons.

Making layer-4 correspond to the feed-forward computation of the probability over coincidence-patterns impose some constraints on the computational roles for other layers. For example, the major projection of layer-4 is to layer-3. This means that any computation that requires major inputs from layer-4 will need to be done at layer-3 and should match the general characteristics of layer-3. The proposals for layer-3 computations, described in section 4.7.4, match these constraints.

4.7.3 Layer 1: The broadcast layer for feedback information and timing information

Feedback connections from higher levels of the cortex rise to layer 1. The recipients of these connections are the cells with apical dendrites in layer 1. Layer 1 comprise mostly of axons carrying feedback from higher level of cortex and non-specific thalamus and apical dendrites and a minor concentration of cell bodies [96] .

To remain consistent with this biological data, the layer-1 in our mapping will be the 'broadcast' layer for feedback information. The axons carrying feedback information $P(G|e^{+})$ will be available at layer-1 and accessed by the apical dendrites of neurons that require this information. In addition, the availability of a timing signal at layer-1 is assumed. The purpose of this timing signal is discussed in section 4.7.5.

4.7.4 Layer 2/3 pyramidal cells: sequence memory, pooling over sequences, incorporating feedback information

The primary inter-laminar excitatory input to layer 2/3 is from the stellate cells of layer 4. In addition, the layer 2/3 neurons receive excitatory inputs from other layer 2/3 neurons (Lateral connections) [4] . Many layer 2/3 neurons also project to higher levels of cortex and to layer 5 [95].

We propose three different roles for the layer 2/3 pyramidal neurons in cortical circuits: (1) Calculation of feed-forward Markov chain (sequence) states, (2) Projection of Markov chain information to higher level cortices and (3) Computation of sequence states that incorporate feedback information. We now consider each proposal in detail and then examine anatomical evidence in support of these circuits.

1. **Pyramidal cells for feed-forward sequence states:** The pyramidal neurons shown in green color in figure 4.9 implement the Markov chain sequence memories and the dynamic programming computations for feed-forward sequential inference. These neurons correspond to the 'circle neurons' that we described in section 4.6.2 and implement the dynamic programming equation 4.3 in table

4.1. These pyramidal neurons receive 'vertical' excitatory inputs from the outputs of layer-4 stellate neurons and 'lateral' inputs from other pyramidal cells within layer 2/3. Circuits in the layer 2/3 of 4.9 show how sequence memories (Markov chains) are implemented in biology. The green pyramidal neurons with blue outlines and blue axons correspond to Markov chain g_1 in figure 4.6 and the green pyramidal neurons with magenta outlines correspond to Markov chain g_2 in figure 4.6. The axons from these pyramidal cells cross column boundaries and connect to other pyramidal neurons that belong to the same sequence circuit. Since these connections correspond to sequence memories, they will be very precise about which columns they target.

2. **Pyramidal cells that project to higher order cortex:** To compute the feed-forward output that needs to be sent to the higher level cortical regions, we propose a second set of pyramidal cells. These pyramidal cells correspond to the Markov chain identities and get excitatory inputs from the previous set of pyramidal cells that belong to the same Markov chain. This second set of pyramidal neurons in layer 2/3 corresponds to the rectangle neurons in figure 4.6. These neurons send their outputs to higher-level cortices. Therefore, their axons project through layer 4, layer-5 and layer-6 towards the white-matter. In figure 4.9, these pyramidal neurons are shown in blue color in layer 2/3. Note that they receive excitatory projections from other layer 2/3 pyramidal neurons and send axons to the white matter.

3. **Pyramidal cells for feedback sequence computation:** In section 4.6.3, we saw that a second set of dynamic programming states were required for the calculation of the belief in coincidence patterns and as an intermediate step in deriving the feedback messages to be sent to the children. These set of neurons do the sequence computations while integrating feedback information from the higher layers. We propose a third set of pyramidal neurons in layer 2/3 for this purpose. These neurons correspond to the filled-circle neurons in figure 4.7. In figure 4.9, these neurons are represented using yellow colored pyramidal neurons in layer 2/3. The lateral connections of these neurons are identical to the lateral

connections of the layer 2/3 green pyramids that we just described. However, these yellow layer 2/3 pyramids also integrate feedback information from layer 1 using their apical dendrites in layer-1 as shown in figure 4.9.

Now, let us examine the anatomical evidence that led us to these proposals. The major bottom-up input required for the above calculations is the feed-forward probability over coincidence patterns that was assigned to layer-4 neurons in section 4.7.2. The major excitatory projection of layer-4 neurons is to layer 2/3 neurons [97]. For example, L4 spiny neurons in the barrel cortex of the mouse are characterized by mainly vertically oriented, predominantly intra-columnar axons that target Layer 2/3 pyramidal cells [63]. Note that the green and yellow neurons in figure 4.9 receive inputs from layer-4 neurons that are in the same column.

Cells in layer 2/3 are known to be 'complex' cells that respond to sequence of motion or cells that respond invariantly to different translations of the same feature. Unlike cells in layer 4 that respond to impoverished stimuli, cells in layer 2/3 of the visual and barrel cortices strongly prefer richer stimuli, such as motion in the preferred direction [49]. This is consistent with our proposal for the layer 2/3 cells that represent different coincidence patterns in the context of different Markov chains (sequences). They become active only in the context of the correct sequence. In biology, it is found that axons of the layer 2/3 pyramidal neurons travel several millimeters in parallel to the layer 2/3–layer-4 boundary and re-enter layer 2/3 to make excitatory connections to pyramidal cells there [4, 64]. This is akin to some of the blue and magenta axons that we show in figure 4.9 and is self-consistent with the implementation of sequence memories and dynamic programming computations. The green neurons and the yellow neurons in figure 4.9 correspond to this description.

For the computation of sequence information that incorporates feedback information, the sequence neurons will also need access to the feedback information present at layer-1. This means that there should be a class of neurons that are part of the sequence circuits as described above and receive excitation from layer 1. Pyramidal cells of this type, with their apical dendrites in layer-1, are found in the cortex [97, 4]. These are the yellow pyramidal neurons shown in figure 4.9.

To calculate the belief in a coincidence-pattern, the outputs of all the yellow neurons in the same column have to be summed up. This corresponds to pooling the evidence for that coincidence-pattern from all the different Markov chains (sequences) in which the coincidence participates. We will see that layer-5 is ideally suited for doing this. It is well known that layer 2/3 pyramidal cell axons have two distinct projection fields: one horizontal, with long-range (2-4 mm) projecting axon collaterals and one vertical, confined to the column [63]. The horizontal, trans-columnar connections target other layer 2/3 pyramidal cells [27, 51] and correspond to the sequence memory circuits that were described above. Both the green neurons and the yellow neurons in figure 4.9 take part in these circuits, with the yellow neurons receiving feedback information as well. It is known that the trans-laminar projections of layer 2/3 neurons are to a class of cells known as layer 5-B[95]. It is also known that it is the layer-3 pyramidal cells with their apical dendrites in layer-1 that connect to layer-5 cells. These projections are confined to the same column [63]. In the next section we will see that this is consistent with our proposal for the belief calculation cells in layer 5.

4.7.5 Layer 5: implementation of belief calculation

We propose that a class of layer-5 pyramidal neurons in cortical circuits calculate the belief over coincidence patterns according to equation 4.5. This corresponds to the computations performed by the pentagonal neuron in figure 4.7. In the biological implementation shown in figure 4.9, these neurons are shown in light cyan color in layer-5. These neurons receive inputs from the yellow-neurons in layer 2/3. Logically, the operation of these belief neurons corresponds to the pooling of evidence for a particular coincidence from the different sequences that this coincidence participates in. These neurons calculate the belief without regards to the exact time at which the belief is going to be active.

Layer-5 pyramidal cells for duration models

In the proposed circuit, a second set of pyramidal cells are involved in representing the precise time at which the beliefs are going to be active. These neurons, shown as the dark cyan neurons in the layer 5 of of figure 4.9, correspond to the rounded-rectangle neurons in figure 4.8. The timing signal, assumed to be broadly available at layer-1, is shown as blue colored axons. The dark cyan timing neurons have their apical dendrites in layer-1 to access this timing signal. It is assumed that the belief-timing neurons project to the non-specific thalamus, which acts as a variable delay mechanism that projects back to layer-1 to complete a timing loop, as shown in figure 4.8. This is identical to the timing loop proposed by Hawkins [43] and similar to the timing loops in the cortical circuits proposed by Granger [37].

Now let us examine the anatomical evidence for these neurons and connections. It is now well established that there are primarily two kinds of pyramidal neurons in layer-5 of the cortex. Type 1, is called the 'regular-spiking' (RS) neurons and type 2 is called the 'intrinsically bursting' (IB) neurons. This classification is done based on the difference in the nature of the spike-trains generated by these neurons. The IB cells are larger, and they have their apical dendrites in layer -1. The RS cells are smaller, and their apical dendrites are mostly in layer 2/3. It is also known that the RS cells are mostly pre-synaptic to the IB cells. That is, the RS-cells send their outputs to IB cells. In our mapping in figure 4.9, the RS-cells are the light-cyan colored neurons in layer-5. Note that their inputs are from layer 2/3. Similarly, the IB cells are the dark-cyan colored neurons in layer-5 with their apical dendrites in layer-1. The output of the RS-cell goes to the IB-cell. These mappings are consistent with anatomical data [96, 97].

However, there is a significant difference between anatomical data and the circuit for layer-5 neurons shown in figure 4.9. Anatomical data says that most of the excitatory connections from the layer-2/3 pyramidal cells (the yellow neurons in figure 4.9) to layer-5 go to the IB-cells in layer-5. Whereas, in our mapping, they go first to the RS-cells, and then to the IB-cells. In our implementation, we have separated the pooling of evidence and the integration of durations into the RS-cells and IB-cells respectively. Nothing prevents the IB-cells from doing both these functions. For example, the apical dendrites of the IB-cells can be involved in the duration models, while the connections close to its cell body from the layer 2/3 cells can be involved in the pooling of evidence. However, that will leave the role of RS-cells in question. In our survey, we could not find detailed information about the inputs to RS-cells, except for the knowledge that their dendrites are in layer 2/3.

The existence of RS-cells can be justified if there is some utility for the belief in coincidence patterns that does not incorporate the precise timing information. A bit of introspection leads us to believe that there is indeed the need for such a neuron. Consider the case of listening to music. We know which note is going to happen next, well before it happens. If only the IB-cells were present, then they would fire only when the belief and the timing both match. There is clearly some utiility to having the information about the belief, available before the precise time at which the coincidence pattern corresponding to the belief occurs. We ascribe this role to the RS-cells in layer-5. They are belief-cells that ‘anticipate’ the occurrence of the belief. The IB-cells represent that same belief at a precise time point.

In some sense, the RS cells can be thought of as the amber lights at a traffic intersection. They come on before the red lights come on and represent the anticipation of the red light. The timing difference between the amber light coming on and the red light coming on depends on the typical speed of the traffic on that street.

The RS neurons fire *tonically.* That is, when activated, they fire at a relatively same spike-rate that is maintained over some duration. In contrast, the IB cells fire with bursts. The contrasting physiological properties of RS and IB neurons might correspond to their complementary roles in the calculation of a timed belief distribution.

With this description, we can make sense of the connections from layer 2/3 cells to the IB-cells. The IB-cells can get this information either directly from the layer 2/3 cells, or through the RS, cells. Both these circuits are plausible and biology could be using both these mechanisms. The fact that IB cells can receive the information directly from layer 2/3 cells do not obviate the RS cells because RS cells have a utility of their own that cannot be represented by the IB-cells. The fact that RS cells are predominantly pre-synaptic to IB cells is also consistent with this proposal.

The IB-neurons of layer 5 project to sub-cortical areas and to motor areas. If a cortical area is to influence actions, it makes good sense that the signals for that should be based on the temporally precise belief of that cortical area, because the belief represents the best possible information about the coincidence patterns represented in that cortical area. Therefore, the fact that layer-5 IB neurons project to sub-cortical areas that influence motor actions is self-consistent with the proposal that they compute the belief. The timing loop requires the projection of the IB neurons to an external timing circuit. Hawkins [43] and Granger [37] have proposed the projections of IB cells to the non-specific thalamus as the mechanism for generating a variable timing signal.

Another role of the computed belief is in the calculation of the messages that are sent to the child nodes. This calculation requires the pooling over the belief from different coincidence-patterns. This means that the axons of layer-5 cells that are involved in this computation should cross column boundaries and travel long distances. There is anatomical evidence for this [96]. In the next section we will see that these connections are self-consistent with the assignment of the role of feedback message computation to layer-6 neurons.

4.7.6 Layer 6: computing the feedback messages for children

We assign to layer 6 pyramidal neurons the role of computing the feedback messages that are to be sent to regions that are hierarchically below. This corresponds to the hexagonal neurons in figure 4.8 and equation 4.8 in table 4.1. In figure 4.9, this is shown as the purple colored neurons in layer 6.

Feedback messages are derived from the results of belief calculation from a set of columns. This means that the layer-6 neurons will receive its inputs from the layer-5 neurons involved in the calculation of belief. This is shown in figure 4.9. A given set of columns will send feedback messages to all its ‘child regions’. The feedback message sent to one child is not the same as the feedback message sent to the other child. In figure 4.9, some of the layer-6 neurons project to the left child while the rest project to the right child.

The input connections to a layer-6 cell come from multiple columns depending on all the coincidence patterns in which a child-nodes Markov chain participated. Note that in figure 4.9, the axonal inputs to layer-6 neurons from layer-5 neurons cross several columns.

Layer-6 is known to be a primary source of cortical feedback connections [22]. There is a class of pyramidal neurons in layer-6 that have short dendritic tufts extending primarily to layer-5. The axons of these neurons project cortico-cortically [96]. Hence they are appropriately situated for calculating the feedback messages and is consistent with our proposals for other layers.

There are many other neuron-types that are identified in layer-6. We do not explain the functions of those neurons. Our conjecture is that some of those neurons are involved in the driving of context dependent, covert attention. Examining these roles is beyond the scope of this chapter.

It is well know that the feed-forward axons from the thalamus make synapses in layer-6 on their way to layer-4 stellate cells. In figure 4.9, these axons are shown as the green axons that synapse in layer-6 and layer-4. The belief propagation equations in table 4.1 do not directly explain the purpose of these layer-6 synapses of these rising neurons. One plausible explanation is that this connection is required for the learning of the synaptic inputs to layer-6 from layer-5. The layer-6 neurons pool the evidence from the different coincidences that a child's Markov chains participate in. To do this, the output of all the coincidence neurons that the child's Markov chain is part of should get wired to the input of the layer-6 feed back neuron that corresponds to that child's Markov chain. This is possible only if, the layer-6 feedback neuron fires at the same time as the layer-5 belief-neuron. This means that, there should be a mechanism to fire the layer-6 feedback neuron corresponding to a child's Markov chain, when that Markov chain is active in the feed-forward message from that child. This can be accomplished by making the feed-forward inputs fire neurons in layer-6 during learning. This could be an explanation for the *en passage* synapses in layer-6 of axons rising to layer-4. Another plausible explanation is that these connections are required for the divisive normalization of the feedback messages as shown by the dotted rectangle in figure A.2 of appendix A.

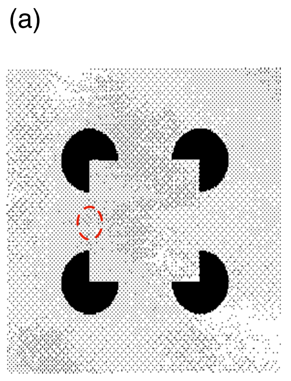
4.8 Applications of the derived cortical microcircuit

The mapping between Bayesian belief propagation computations and cortical anatomy enables us to view the variables in the calculation in an HTM node as neural responses arranged in a laminar and columnar manner that corresponds to the cortical arrangement. The columnar arrangement corresponds to the coincidence-patterns in the node and the laminar arrangement corresponds to the different stages involved in the belief propagation computations. With this mapping, the HTM node can be treated as a piece of cortical tissue. Neural responses that follow a particular pattern of firing can be searched for and located. This, in combination with the hierarchy of an HTM network, can be used to model and predict different physiological phenomena.

4.8.1 A model for the illusory contour effect

The illusory contour effect is a well-known cognitive and physiological phenomenon. Figure 4.11(a) shows a Kanizsa diagram that produces the illusory contour effect. When viewing this figure, humans perceive the edges of a square even in regions where there is no direct visual evidence for them. Lee and Nguyen [59] found that neurons in area V1 responded to such illusory contours even though their feed-forward receptive fields do not have any evidence supporting the presence of a line. In [60] , Lee and Mumford suggested that, this could be the result of Bayesian computations. Their argument was that the presented stimulus, according to the statistics of the visual world, is adequate to create a high-level hypothesis of the rectangle even though the edges are missing. The activation of this global hypothesis at areas V2 and above in turn constrains the activity of lower level neurons through the feedback messages.

In addition to finding the neurons in V1 that respond to the illusory contours, Lee and Nguyen also studied the temporal dynamics of their responses. Figure 4.11(b)



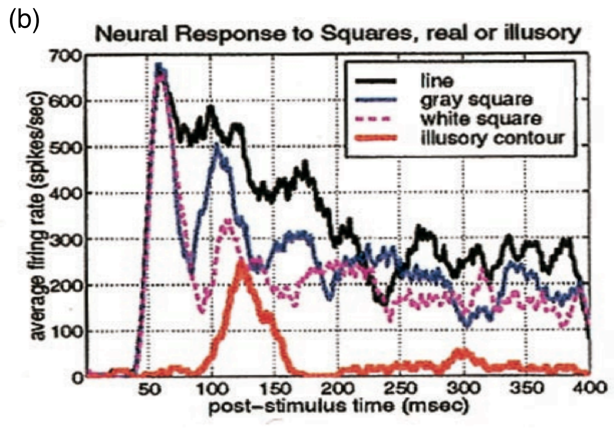


Figure 4.11: (a) A Kanizsa diagram that produces the illusory contour effect. While viewing this figure, vertical edges of a square are perceived even where there is no direct evidence for those edges – for example within the dotted circle. This image is adapted from [59] (b) The dynamics of illusory contour response as observed in monkeys. Note the onset of the response to illusory contours occur about 50 milliseconds later compared to the onset of the response to a real contour. Image source: Nguyen and Lee [59].

shows the temporal dynamics of the neurons responding to illusory contours as observed in the monkey visual cortex [59]. The summary of their findings is that the population averaged response to illusory contours emerged 100 milliseconds after stimulus onset in the superficial layers of V1 and at approximately 120 to 190 millisecond in the deep layers. The responses to illusory contours in area V2 occurred earlier, at 70 msec in the superficial layers and at 95 msec in the deep layers.

The hypothesis that illusory contours are the result of the higher levels imposing their knowledge on to the lower levels is tested using a special input pattern (stimulus) presented to the Pictures HTM network. The Pictures HTM network was introduced in chapter 2. The input pattern, an incomplete 'A' as shown in figure 4.12(1), is created by deleting a small segment from the training pattern 'A'. This pattern is presented at the input to the trained Pictures HTM network and feed-forward and feedback inference is done using Bayesian belief propagation equations. (See appendix B for examples of feedback based reconstruction of inputs to the Pictures network).

As the belief propagation messages are propagated up and down in the hierarchy,

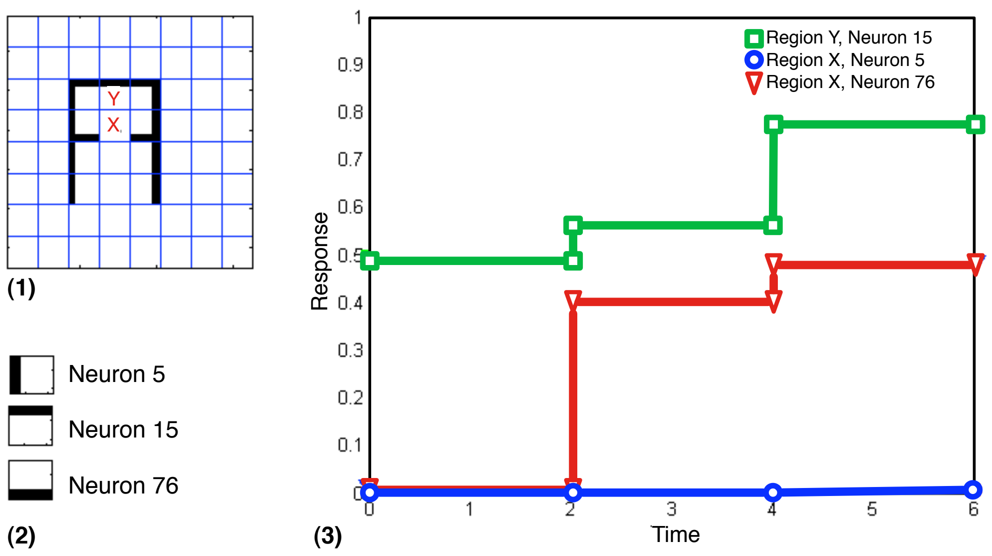


Figure 4.12: (1) Input pattern – an incomplete ‘A’– for replicating the illusory contour effect in the Pictures HTM network introduced in chapter 2. The Pictures HTM network was trained on complete ‘A’s and other patterns (See section 2.6.1 for details). (2) The ‘feed-forward’ receptive fields of the neurons recorded from nodes receiving inputs from the areas marked X and Y in the first figure. (3) Dynamics of the neural responses to illusory contours and real contours as observed in nodes X and Y of the Pictures network. See text for details.

we recorded the activity of a selected set of ‘neurons’ in two nodes X and Y at the level 1 of the Pictures network. These nodes correspond respectively to the nodes at level 1 that have their input fields as the regions marked X and Y in figure 4.12(1). The neurons that were selected for this study were the belief neurons in nodes X and Y corresponding to the coincidence patterns shown in figure 4.12(b). Neuron 76, recorded from the level-1 node corresponding to the input field X, was the belief neuron in the idealized cortical-column representing a horizontal line-segment aligned with the bottom-edge as its coincidence pattern. Neuron 5, observed from the same node, was in a cortical column that represented a vertical line. Neuron 15 was recorded from the node corresponding to the input field Y and represented the belief in a horizontal line-segment aligned with the top-edge.

The input pattern in figure 4.12(1) is presented to the Pictures HTM network at time $t = 0$. The time unit in figure 4.12(3) represents the delay in propagating a belief propagation message one level in the hierarchy. For example, a time-step of 1 is the time taken by a feed-forward message from a level-1 node to reach its level-2 parent and is same as the time taken by a feedback message from a level-3 node to reach its level-2 child. The message passing between all the nodes at all the levels occur concurrently. The delays in calculating the intermediate variables within a node are ignored. (See appendix B for more details on the timing of the belief propagation messages.)

Consider neuron 15 from node Y. This neuron shows a robust response at time $t = 0$ because the input field to this node receives a perfect input that is tuned to neuron 15. Therefore, at time $t = 0$, there is a large amount of support for the coincidence pattern represented by neuron 15 and hence its activity is high. In contrast, neuron 76 of region X does not show any response at time $t = 0$. This is because there is no bottom-up evidence in the input field to node X to support its belief in the coincidence pattern represented by neuron 76.

At time $t = 2$ the messages from the level-1 node have propagated one level up and feedback messages from level-2 nodes have arrived at the level-1 nodes. These feedback messages incorporate the level-2 nodes’ interpretations, based on their past experience, of the feed-forward messages they received from multiple level-1 nodes.

The neuron responses of the level-1 nodes at this time integrate this feedback messages into consideration. At time $t = 2$, the response of neuron 15 from node Y has increased because, the evidences from the nodes surrounding Y, through the mechanism of belief propagation, have increased the evidence for node Y to believe in the horizontal-line coincidence pattern that is represented by neuron 15. This is because the combination of the evidence from these neighboring nodes is consistent with what the network has been trained on.

Now consider the response at time $t = 2$ of Neuron 76 from node X. The response of this neuron went up from near zero at time $t = 0$ to a robust response at time $t = 2$. Clearly, at this time point, this neuron is responding as if there is a horizontal line in the input field of node X, even though there is no bottom-up evidence for this. The response of Neuron 76 is the illusory contour response.

The reason why the response of neuron 76 went up at time $t = 2$ can be understood in terms of the learned memory of the network and in terms of the belief propagation computations. The network was trained on patterns similar to A. The nodes at level-2 have learned that the occurrence of a horizontal line in one of the children is usually coincident with the occurrence of a horizontal line in its neighboring child. There is strong evidence for line-segments on either sides of the input field of node X. The parent node of node X, from the memory of its past experience and based on the messages received from the children, infers that a continuous line is likely to be the input pattern. This is because there is sufficient evidence for the existence of a line even though node X did not report the presences of a horizontal line segment in its input field. This inference is reflected in the feedback message to node X from its parent and tells node X that there is strong top-down evidence for a horizontal line. Since neuron 76 is the belief neuron that represents the combination of top-down and bottom-up evidence, its response goes up with the increase in top-down evidence for the horizontal line.

At time $t = 4$, feedback messages incorporating global information all the way from the level-3 node arrive at the level-1 nodes. These messages further reinforce the beliefs of neuron 76 and neuron 15. The response of neuron 5 from node X is also plotted as a control to show that feedback messages do not increase the responses of all neurons. The response of neuron 5 does not increase with the incorporation of feedback information because, the contextual information, in light of the past experience of level-2 nodes, do not support the belief in a vertical line in node X.

The delayed onset of the illusory contour response in figure 4.12(3) corresponds to the delayed onset of illusory contour response observed by Lee and Nguyen in 4.11(b). The delayed onset occurs because of the delays in propagating messages up and down in the hierarchy. Lee and Nguyen also showed that the illusory contour response occurs first in the superficial layers and then in the deep layers. This is also consistent with the cortical circuit model in figure 4.9 because the feedback information first reaches the Markov chain neurons in layer 2/3 (the yellow-neurons in figure 4.9) and then is then integrated into the layer-5 neurons.

4.9 Shape perception reduces activity in the lower levels

Using functional MRI, Murray et al [72] showed that perception of objects in higher levels of the visual cortex reduces the level of activity in the primary visual cortex. They postulated that inferences of high-level areas are subtracted from incoming sensory information in lower areas through cortical feedback.

The model we described in this chapter offers an alternative explanation for this phenomenon compared to the subtraction theory. Reduction in activity occurs because the recognition of a global percept by a higher-level node, narrows the hypotheses space maintained by a lower-level node as a direct result of Bayesian belief propagation in the hierarchy . This was verified in the Pictures HTM network using an experiment.

In this experiment, a highly noisy picture of a helicopter is given as the input to the network. The noisy image ensured that the input pattern received by the level-1 nodes did not exactly match any of the coincidence-patterns stored in those nodes. Figure 4.13 shows the activity of the belief-cells recorded from a level-1 node receiving its input from the rectangular area (marked with an arrow) in the helicopter images.

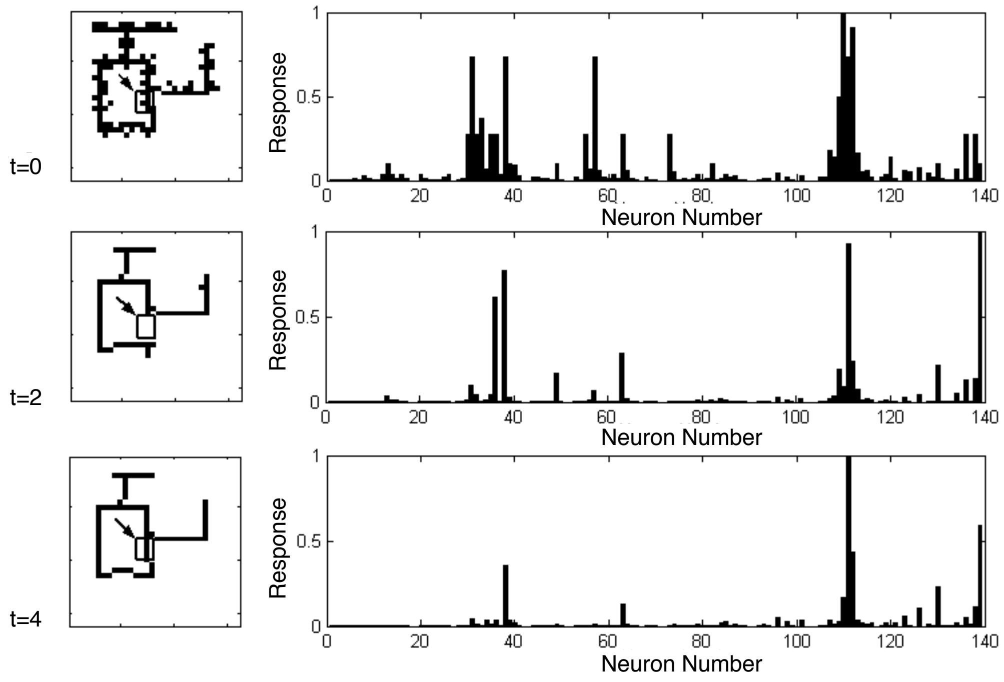


Figure 4.13: **Higher-level perception reduces the activity of lower-levels:** Left panel top shows the noisy helicopter input image that is presented to the network. The right panel shows the recorded activities of the belief neurons from a level-1 node that received its input from the input field marked by the arrow in the left panel figures. The three rows correspond to the responses at times $t = 0, t = 2$ and $t = 4$. The left panel figures for $t = 2$ and $t = 4$ show the reconstructed input images at those times. See text for details.

At t = 0, the input is highly ambiguous as shown and hence the belief of the region is highly spread out. This spread reflects the ambiguity in the bottom-up evidence to this node. At t = 1 the level 2 nodes integrate the information from multiple level 1 nodess and feed back information to level 1 nodes. At t = 2, the level 1 nodes use this information to update their beliefs. Figure shows that this reduces the spread of the belief as compared to t = 0. The corresponding picture of the helicopter is the reconstruction at this stage if you take the best guesses from all level 1 regions. Although the belief distribution is narrower, the best guess based on that belief distribution does not produce the correct reconstruction in the marked input field. At t = 4, the level 1 nodes get feedback messages which incorporate the global information. This further narrows the belief distribution. Note also that the reconstruction based on the incorporation of this global information reconstructs a vertical line in the marked area.

In addition to replicating this effect, out model also offers a prediction. The prediction is that the reduction in activity will be observed mostly in the deep layers and partly in the superficial layers of the cortical circuitry. The reduction in the deep layers corresponds to the narrowing of the belief distribution represented by the layer-5 neurons in figure 4.9. The reduction in the superficial layers corresponds to the reduction in activity of the yellow neurons in layer 2/3 of figure 4.9 that integrate the feedback information.

4.10 Variations, omissions and extensions

In this section we consider the omissions and variations of the proposed cortical circuits and identify research areas that could extend them.

Learning

The microcircuits are derived based on the belief propagation equations in a learned HTM node. The learning algorithms described in chapter 2 have not been analyzed for their biological plausibility. Incorporation of learning into the circuits would require neural mechanisms for competition, inhibition and online learning. This is beyond the scope of this thesis.

Inhibitory neurons

Inhibitory neurons are missing from the derived circuits. The assumption we make is that excitatory neurons provide the prominent information processing pathway that is supported by inhibitory neurons. Most of the inter-laminar connections within a cortical area are provided by the spiny excitatory neurons, whereas the smooth inhibitory interneurons principally connect locally within their layer of origin. It is the excitatory cells that connect long distance in both vertical and lateral dimensions and their activity is then moulded by local inhibitory neurons [22]. Several computational and physical functions can be attributed to the inhibitory neurons. They are required for competition during learning and inference. It is well known that belief propagation requires normalization of the messages to avoid numerical underflows. Inhibitory neurons could also be required for avoiding instabilities produced by positive feedback loops.

Continuous cortical regions

The HTM nodes are discrete entities with abrupt boundaries. This does not correspond to biology because the visual cortex is not a collection of discrete nodes. Although the idealized HTM node instantiation gives us the flexibility to create mathematical abstractions that can be analyzed, it needs to be modified to make a biological correspondence.

One way to accommodate this could be to use HTM nodes with heavily overlapped input fields to construct a region. Consider an HTM node that represents coincidence-patterns that correspond to orientations $10^o,\ 30^o,\ 50^o, \cdots$ and another HTM node representing orientations $20^o,\ 40^o,\ 60^o, \cdots$. These two nodes, when physically interleaved, can produce more continuos variation of the receptive fields.

These modifications introduce the problem that the resultant hierarchical network, viewed as a Bayesian network, has cycles in it. Although theoretical guarantees do not exist for the convergence of belief propagation in such hierarchies, successful systems have been built based on belief propagation in loopy graphs [71, 8, 30].

Sparse distributed representations

Many computational neuroscientists now believe that the neocortex uses sparse-distributed representations to encode information [87, 107, 106]. The HTM model here uses sparse-distribted representations when the representations of a hierarchical level are considered. However they do not use sparse-distributed representations within a node. In domains where a node is exposed to data that has rich characteristics, this model would require modifications to include sparse distributed representations within an HTM node.

Although several algorithms have been researched for the learning of sparse representations, the recently discovered theory of Compressed Sensing (CS) theory might provide a biologically plausible and simple method for sparse-distributed representations in the cortex [21, 13]. The CS theory argues that the information in signals that have a sparse representation in some basis is well preserved with random projections. This means that sparse representations might be achieved by representing an incoherent set of randomly selected coincidence patterns, while letting multiple coincidence-patterns to be active at the same time to represent any input. From a biological stand point, this would correspond to the layer 4 cells representing randomly selected coincidence patterns from their input space.

Cortical maps

The circuits derived in this chapter provide no explanation for the cortical maps [104, 92]. The reason for this is that the circuits derived here attempt to explain only the information processing in a learned model. Any arrangement of the circuit that preserves the connections would still do the same information processing. However this leaves a major organization property of the neocortex unexplained. There could be two reasons for the cortical maps. One is that organizing the columns in a particular manner in space could reduce wiring length or some other resource that needs to be conserved. Another reason could be that a topographical organization of "similar" patterns could reduce the search space for coincidence-detection and sequence learning algorithms. Circuits for implementing self organizing map [55] algorithms need to be incorporated into the theory. This is left as future work.

Thalamic relays and attentional/motor pathway

The microcircuits derived in this chapter do not incorporate any mechanism for attention control. Hypothesis-driven attention control is an important aspect of perception. Attention plays an important role in belief propagation as well [75]. It is known that thalamus plays an important role in corticocortical communication, acting as a dynamic control of information that is passed from one cortical area to another [88, 73]. The thalamic pathway is an alternate pathway in the hierarchy in addition to the direct cortico-cortical projections. There are computational reasons why two pathways might exist in the cortex. In belief propagation, the messages required for attention control are different from that of a standard feedback message. The attention control messages instantiate variables at intermediate levels and therefore affect the results of feed-forward propagation, where as the standard feedback messages in belief propagation do not interact with feed-forward messages. Some forms of attention can also be considered as an internal motor action. In that sense, the attention control mechanism can also be thought of as analogous to the *do* operator proposed by Pearl [76] to model the effect of actions in causal Bayesian networks. It is a tantalizing clue that Guillery and Sherman [39] found that the layer-5 pyramidal cells that project to the pulvinar of the thalamus also project to motor structures. Incorporation of the attention pathway into the derived circuits is left for future research.

4.11 Discussion

Is there a common cortical algorithm?

The work in this chapter is based on the assumption of a common cortical algorithm. We provided some evidence in support of the common cortical algorithm in the introduction to this thesis [3, 2, 103]. However, the existence of a common cortical algorithm is anything but a settled matter. One argument that can be made against the common cortical algorithm is the heterogeneity of neuron types. For example, researchers have revealed substantial variation in pyramidal cell structure in different cortical areas [26]. Some of these variations are systematic – for example, the increase in dendritic arbor size with ascendance in the hierarchy – and can be resolved with the idea of a common algorithm. Until we understand and explain the computational reason behind a large majority of such variations, the common cortical algorithm will have to be considered as a working hypothesis.

Is there a hierarchy?

Although there seems to be no debate that the visual system is hierarchically organized according to principled anatomical criteria [45], the hierarchy in the visual system is not a clean hierarchy as the one we considered here. There are many connections that skip levels and multiple internally consistent hierarchies can be derived based on the same anatomical criteria [46, 102]. Hierarchical organization/processing has also been found in the auditory cortex [108] and in the motor cortex [57].

Are cortical columns important?

The columnar organization of the cortex was discovered half a century ago [67] and has survived as a fundamental principle of brain organization. Columnar organization was one of the features of the derived circuits and this was based on the presumed columnar organization of the neocortex. Horton and Adams[52] raised several issues regarding the function of columnar organization and its value. Our mapping could help resolve some of the issues that were raised by these authors.

One seeming point of contention is the delineation of the roles of minicolumns and macrocolumns [82, 12]. Horton and Adams argue that defining two different structures as the fundamental unit of cortex produces confusion. Although our mapping does not consider columns as 'independent' fundamental units, it can offer some clarity to this dilemma. In our mapping, minicolumns are *physical* units of cortical computations and macrocolumns are *logical* units of cortical computation. The minicolumns establish the basic vertical wiring necessary and common to belief propagation computations. Consider a cortical area consisting of 1000 such minicolumns. If that cortical area is exposed to a world where only 10 patterns need to be represented, then it can be expected that approximately 100 minicolumns will be *allocated* to represent the concept of any particular pattern – a macrocolumn. This also addresses another concern that the authors raised – the variation in the number of minicolumns per macrocolumn across areas. The variation occurs due to a combination of genetics (the number of available minicolumns) and learning (the richness of the statistics of the world).

Another point raised by the Horton and Adams is the near-continuous variation of orientation responses in the visual cortex without any apparent discrete boundaries. We discussed in section 4.10 how the continuous variation in the cortex could be handled using overlapping nodes. The visual world has continuous orientations and this could be the reason why the visual cortex tries to preserve that continuous gradual variation. On the other hand, when the world is more discrete as in the case of stimulation received from whisker follicles, the cortical organization is predominantly columnar to the extend that it is called *barrel cortex*. If we lived in a visual world that had only four orientations, then we would probably have seen a 'barrel' organization in the visual cortex.

Are connections random or precise?

There are two views in neuroscience about the nature of connections between neurons. Many researchers argue that connections maybe precise with respect to cell classes but are likely to be highly random with respect to individual neurons within a class [11]. On the other hand, there are at least some cases of highly specific and precise connections [56], even to the extent of the relative position of the targets.

The circuit models derived here support both these view points. We saw that some of the connections in the minicolumn can be pre-wired without learning. Those connections are likely to be highly specific. Moreover, the sequence learning connections that travel long distances to target another column (coincidence-pattern) in the same Markov chain are also likely to be specific. On the other hand, the coincidence-patterns represented by the layer-4 neurons can be randomly selected from the set of input coincidences as we discussed in the context suggested by the Compressed Sensing theory.

Appendix A

Belief Propagation in HTM Networks

In this section we describe Bayesian belief propagation as implemented in HTMs. Bayesian belief propagation was pioneered by Judea Pearl and more details on this can be obtained from his book entitled *Probabilistic Reasoning in Intelligent Systems* [75]. The belief propagation equations for HTM networks are derived by modifying the belief propagation equations for Bayesian networks. Like Bayesian networks, HTMs can be thought of as encoding relationships between random variables. In our vision example, these random variables correspond to the coincidence-patterns and Markov chains learned at multiple levels of the network.

An input to the network is typically termed *evidence*. For example, a new image that is presented to the network is the evidence for the network. The network propagates this evidence in the hierarchy to adjust the *belief* states of each node in the network. The belief states are defined as the posterior probabilities of coincidence patterns in each node.

A useful picture to have in mind is that of the HTM network being in equilibrium with the current state of the world. In our case, the current state of the world is the input image. The belief states in the nodes of the network reflect this evidence. When a new input image is presented, this information is passed up and down the hierarchy using belief propagation and the belief states of the nodes change to match the new evidence.

Two popular variants of belief propagation algorithms exist and they answer different queries based on the same model. One variant, called *sum-prop*, computes the belief states of each node given the evidence. Another variant of the belief propagation algorithm is called *max-prop*. This is also known as belief revision. This algorithm finds the best possible combination of assignments of states of nodes in an HTM network, given the evidence. The answer that belief revision computes is known as the *Most Probable Explanation* for the current evidence [75]. The difference between these two queries can be subtle. In the sum-prop case, the belief states calculated at a node is the degree of belief calculated without assuming that other nodes have committed to any particular state. In the max-prop case the belief states correspond to the degree of belief of a node's state being part of the best possible configuration. This assumes the commitment of other nodes to particular states.

We use the following notation to describe the belief propagation mechanisms in HTM networks. We follow the notations used by Pearl [75]. Equations are described for two cases. In the first case, all the evidence is restricted to the current time instance. This case is applicable to the recognition of an image that is presented to the network and also for reconstruction of that image. This is described in the next section. The equations for incorporating evidence from past time steps – the dynamic case – is described in section A.2.

A.1 Belief propagation equations for the static case

A.1.1 Notation

C^k	Random variable representing the coincidence patterns of the k^{th} node. Depending on the context, C^k is also used to represent the set of coincidence patterns in the k^{th} node.

c_i^k	The i^{th} coincidence in the k^{th} node.
G^k	The random variable representing the Markov chains/temporal groups of the k^{th} node. Depending on the context, this can also be used to represent the set of temporal groups of the k^{th} node.
g_i^k	The i^{th} Markov chain/temporal group in the k^{th} node.
e^- and ^-e	The evidence from below, from a node's view point. This is comprised of all the evidence observed by all the descendants of this node.
e^+ and ^+e	The evidence from above, from a node's view point. This is comprised of all the evidence observed by all the nodes that are not descendants of this node.
$P(e^-\|G^k)$	Probability of evidence from below given the Markov chains/temporal groups in node k. This is a vector of length equal to the number of groups.
$P(e^-\|g_i^k)$	The probability of evidence from below given the i^{th} group in node k. This is a scalar.
$P(e^-\|C^k)$	The probability of evidence from below given the coincidence patterns in node k. This is a vector of length equal to the number of coincidence patterns in node k.
$P(e^-\|c_i^k)$	The probability of evidence from below given the i^{th} coincidence pattern in node k. This is a scalar.
$P(C^k\|G^k)$	The rows of this matrix correspond to the Markov chains/temporal groups and the columns correspond to the coincidences. The $(i,j)^{th}$ entry of this matrix, $P(c_j\|g_i)$, gives the probability of seeing the coincidence c_j given that we have seen the Markov chain/temporal group g_i.

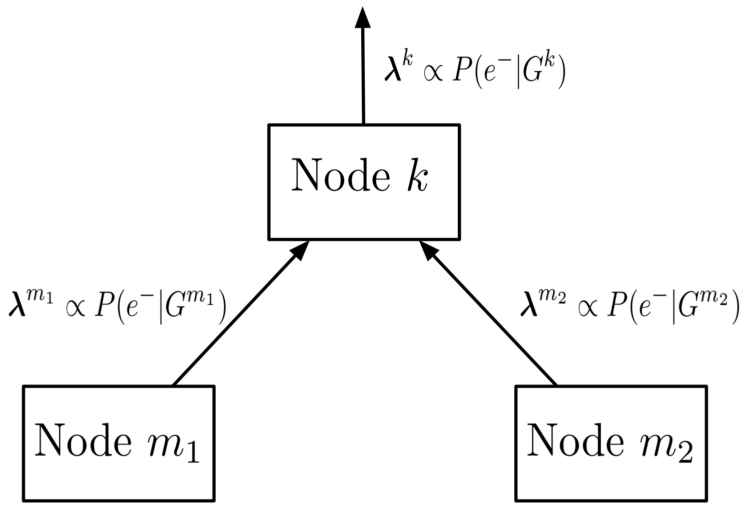

$\lambda^k \propto P(e^-|G^k)$
Node k
$\lambda^{m_1} \propto P(e^-|G^{m_1})$
$\lambda^{m_2} \propto P(e^-|G^{m_2})$
Node m_1
Node m_2

Figure A.1: A segment of an HTM network.

λ^k The message that the k^{th} node sends to its parent. This is a vector and is proportional to $P(e^-|G^k)$. $\lambda^k(r)$ is the r^{th} component of this vector and is proportional to $P(e^-|g_r^k)$

For all the variables defined above, superscripts indicate the membership of the variables in a particular node. To reduce clutter, the superscript will be dropped when the node-membership of the variables are clear from the context.

A.1.2 Learned memory of a node

The coincidence-patterns and the Markov chains/temporal groups form the learned memory structures of the node on which belief propagation operates.

A coincidence-pattern c_i defines a particular co-occurrence of the Markov chains/temporal groups of its child nodes. It can be thought of as a vector $[r_i^{m_1}, \cdots, r_i^{m_M}]$, where the r_i's correspond to the indices of the groups of its child nodes. For example, if $M = 2$ and c_4 is the coincidence of group 2 from child node m_1 and group 5 from child node m_2, then the coincidence pattern represented by c_4 is $[2, 5]$.

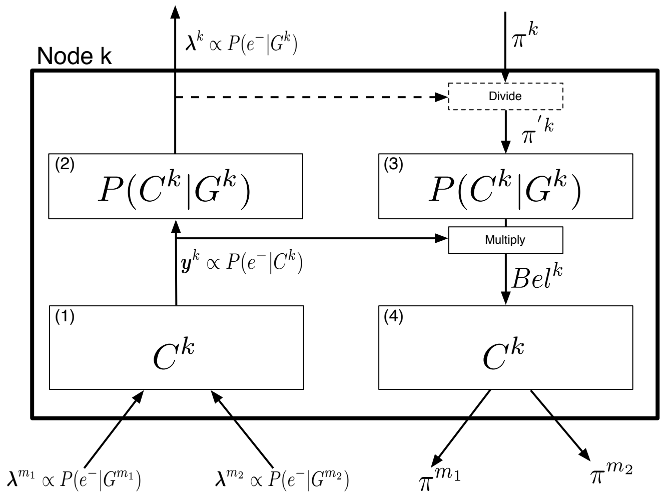


Figure A.2: Block diagram of belief propagation operations in an HTM node for the static case.

For belief propagation in the static case, we need a conditional probability matrix that can be derived from the Markov chains/temporal groups. This matrix is denoted by $P(C|G)$. The i^{th} row of this matrix corresponds to the i^{th} group in the node and the j^{th} column of the matrix corresponds to the j^{th} coincidence pattern. The i^{th} row of the $P(C|G)$ matrix is calculated as follows. The positions corresponding to the coincidences that belong to this group are populated by their frequencies of occurrence. The rest of the positions are populated by zeros. This row is then normalized to sum to one to obtain the i^{th} row of $P(C|G)$. C and $P(C|G)$ constitute the memory of an HTM node. These memories are affected only through learning.

A.1.3 Belief propagation computations

Figure A.1 shows a segment of an HTM network with 3 nodes. We describe the belief propagation computations with node k as the reference. This node has two child nodes - node m_1 and node m_2. For generality we assume that this node has M children. We use $m_1, m_2, \cdots, m_M$ as the indices of the child nodes.

This node gets M messages from its child nodes. The message from node m_i will be λ^{m_i} where,

$$\lambda^{m_i} \propto P(e^-|G^{m_i}) \tag{A.1}$$

This message is proportional to the probability of evidence from below given the temporal groups G^{m_i} in node m_i. This is a vector of length equal to the number of groups of node m_i.

Feed-forward computations

From these messages, first the degree of certainty over coincidence-patterns is computed. This quantity is represented by y and is proportional to $P(e^-|C)$. The i^{th} component of the y vector corresponds to the coincidence c_i. In general, this coincidence can be thought of as represented as a vector of M numbers $[r_i^{m_1}, r_i^{m_2}, \cdots, r_i^{m_M}]$ where the r's represent the group indices of its children that constitute this coincidence. The i^{th} component of y is then calculated as

$$y(i) = \alpha_1 \prod_{j=1}^{M} \lambda^{m_j}(r_i^{m_j}) \tag{A.2}$$

where α_1 is an arbitrary scaling constant. This scaling constant is usually set to a value so that the messages do not encounter floating point underflow.

The above calculation assumes that the evidences from the children can be combined independently, given the coincidence-patterns of the node. Since, $P(e^-|c_i) = \prod_{j=1}^{M} P(e^-|g_{r_{m_i}}^{m_i})$ and since λ^{m_i} are proportional to $P(e^-|G^{m_i})$ for all i, this calculation ensures that y is proportional to $P(e^-|C)$

The node calculates its output y. This output λ is proportional to $P(e^-|G)$ and is a vector of length N_g, the number of temporal groups within the node. The i^{th} component of this vector is calculated as,

$$\lambda^k(i) = \sum_{j=1}^{N_c} P(c_j|g_i)y(j) \tag{A.3}$$

Since $P(e^-|g_i) = \sum_{j=1}^{N_c} P(c_j|g_i)P(e^-|c_j)$, the above computation ensures that λ^k is proportional to $P(e^-|G^k)$.

Feedback computations

The feedback computations take the top-down message π^k from the parent and computes the messages that need to be sent to the children. In the process, the *belief* of the node is calculated as the product of the feed-forward probability over coincidences and the feed-back probability over coincidences. The first step in this computation is the calculation of π' as

$$\pi'^k(i) = \pi^k(i)/\lambda^k(i) \tag{A.4}$$

From π', the top-down probability over coincidence-patterns is calculated as

$$z(i) = \sum_{g_j \in G} P(c_i|g_j)\pi'(j) \tag{A.5}$$

The *belief* of the node in its coincidence-patterns is the product of the top-down and bottom-up probabilities over coincidences:

$$Bel(c_i) = y(i) \times z(i) \tag{A.6}$$

The π messages that are sent to the children are calculated from the *belief* distribution and the coincidence-pattern descriptions.

$$\pi^{child}(g_m^{child}) = \sum_{c_i \in C} I(c_i) Bel(c_i) \tag{A.7}$$

where,

$$I(c_i) = \begin{cases} 1, \text{ if} g_m^{child} \text{ is a component of } c_i \\ 0, \text{ otherwise} \end{cases} \tag{A.8}$$

The belief revision computations can be obtained by replacing the summations in equations A.3, A.5 and A.7 with maximizations.

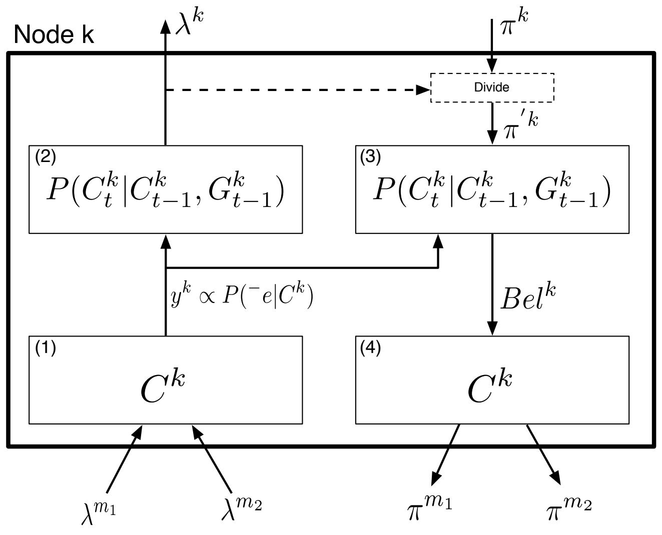


Figure A.3: Block diagram of belief propagation computations within a node for the dynamic case.

A.2 Belief propagation equations for the dynamic case

The equations we described in the previous section is a special case of the belief propagation equations required if sequential evidence is taken into account. In this section, we use dynamic programming [5, 53, 93]methods to derive the equations for sequential inference under some simplifying assumptions.

A.2.1 Notation

We need some new notation to take care of temporal indexing in addition to the notation described in the previous section. Figure A.2 shows the block diagram of the node for belief propagation computations for the dynamic case.

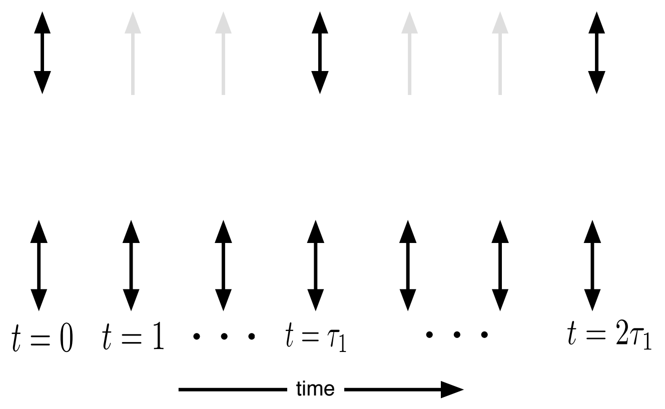


Figure A.4: Timing of input and output messages to Node k. The message passing between the node and its children occur at intervals of 1 time step, where as the message passing between the node and its parents occur at intervals of τ_k time steps.

$c_0^t, {}^-e_0^t, {}^+e_0^t$	These denote all the possible sequence combinations of the underlying variable from time 0 to time t.
$\alpha_t, Bel_t, y_t, \beta_t$	All the internal variables are updated with every time step. Their value at any particular time are denoted using the subscript t.
$c(t)$	Denotes all possible values the coincidence can take at time t. Similar notation is used for the Markov chains.
$c_i(t)$	Denotes the coincidence-pattern c_i at time t.
$P(C_t^k \mid C_{t-1}^k, G_{t-1}^k)$	This describes all the within Markov chain transition probabilities for the set of Markov chains in the node.

A.2.2 Coincidence patterns and the Markov chains

The coincidence-patterns for the dynamic case are the same as the coincidence-patterns for the static case. The temporal-groups in the static case are zeroth order Markov chains, whereas in the dynamic case we use first-order Markov chains. The set of Markov chain transition probabilities is a learned data structure and is denoted using $P(C_t^k|C_{t-1}^k, G_{t-1}^k)$.

A.2.3 Dynamic programming equations

We derive the dynamic programming equations under simplifying assumptions. Figure A.2 shows the timing of the messages, as seen by node k. We assume that the bottom-up messages to the node arrive synchronized to time-steps $t = 0, t = 1$ etc. The messages from the parent node arrives at intervals that are multiples of τ_k, where τ_k is the time constant of node-k. Similarly, it is assumed that bottom-up messages are passed at an interval of τ_k, although they are calculated for every time step $t = 0, t = 1, \cdots$.

Assume that a top-down message arrived at time $t = 0$, synchronous with bottom-up messages from the children. We derive the update equations for calculating the internal states and outputs of the node from $t = 0$ to $t = \tau_k$, as more bottom-up messages arrive.

$$\begin{aligned}
Bel_t(c_i) &= P(c_i(t)|^{-}e_0^t, ^{+}e_0) \\
&= (1/P(^{-}e_0^t|^{+}e_0^t)) \sum_{g_r \in G^k} \sum_{c_0^{t-1}} P(^{-}e_0^t|c_0^t, g_r, ^{+}e_0) P(c_0^t, g_r|^{+}e_0) \\
&= (1/P(^{-}e_0^t|^{+}e_0^t)) \sum_{g_r \in G^k} P(g_r|^{+}e_0) \sum_{c_0^{t-1}} P(^{-}e_0^t|c_0^t, g_r, ^{+}e_0) P(c_0^t, g_r|^{+}e_0) \\
&\propto \sum_{g_r \in G^k} P(g_r|^{+}e_0) \beta_t(c_i, g_r) \qquad \text{(A.9)}
\end{aligned}$$

Where the dynamic programming variable β_t is defined as

$$\beta_t(c_i, g_r) = \sum_{c_0^{t-1}} P(^{-}e_0^t|c_0^t, g_r, ^{+}e_0) P(c_0^t, g_r|^{+}e_0) \qquad \text{(A.10)}$$

Then, the update equation becomes

$$\beta_t(c_i, g_r) = P(^{-}e_t|c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t)|c_j(t-1), g_r) \beta_{t-1}(c_j, g_r) \qquad \text{(A.11)}$$

The initial state is $\beta_0(c_i, g_r) = P(^{-}e_0|c_i(t = 0)) P(c_i(t = 0)|g_r, ^{+}e_0)$ and can be understood as the initial distribution of each Markov chain.

The messages for bottom-up transmission are calculated as follows:

$$\begin{aligned}
P({}^{-}e_0^t|g_r(t)) &= \sum_{c_0^t} P({}^{-}e_0^t, c_0^t|g_r) \\
&= \sum_{c_0^t} P({}^{-}e_0^t|c_0^t)P(c_0^t|g_r) \\
&= \sum_{c_0^t} P({}^{-}e_0^{t-1}|c_0^{t-1})P({}^{-}e_t|c_t)P(c_0^{t-1}, c_t|g_r) \\
&= \sum_{c_0^t} P({}^{-}e_t|c_t)P(c_t|c_{t-1}, g_r)P({}^{-}e_0^{t-1}|c_0^{t-1})P(c_0^{t-1}|g_r) \\
&= \sum_{c_i \in C^k} P({}^{-}e_t|c_i(t)) \sum_{c_j \in C^k} P(c_i(t)|c_j(t-1), g_r) \\
&\quad \sum_{c_0^{t-2}} P({}^{-}e_0^{t-1}|c_0^{t-1})P(c_0^{t-1}|g_r) \\
&= \sum_{c_i \in C^k} P({}^{-}e_t|c_i(t)) \sum_{c_j \in C^k} P(c_i(t)|c_j(t-1), g_r)\alpha_{t-1}(c_i, g_r)
\end{aligned} \tag{A.12}$$

Where α is the dynamic programming variable whose update equation is given by

$$\alpha_t(c_i, g_r) = P({}^{-}e_t|c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t)|c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r) \tag{A.13}$$

The bottom-up output message is calculated as

$$\lambda(g_r) = P({}^{-}e_0^t|g_r(t)) \propto \sum_{c_i(t) \in C^k} \alpha_t(c_i, g_r) \tag{A.14}$$

Appendix B

Feedback Propagation in HTM Networks

This section describes the results of feedback propagation in the Pictures HTM network that was described in chapter 2 and trained on the Pictures data set described in section 2.6.1. The recognition results of this network was discussed in section 2.6. Recognition is achieved by presenting an image as input to the network and propagating the evidence up in the hierarchy using Bayesian belief propagation. For any image that is presented, it is also possible to propagate information down in the hierarchy. This is done using the belief propagation equations that are described in appendix A. Feedback propagation can be used to clean up a noisy images and to drive attention. The figures that follow describe and demonstrate the operation of feedback propagation in an HTM network.

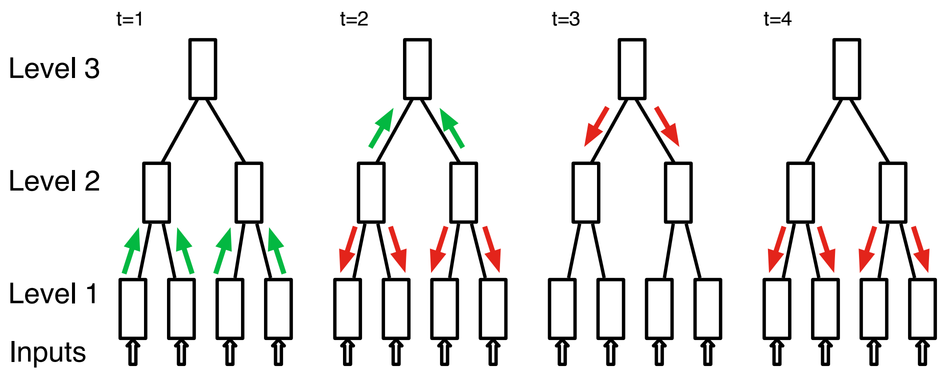


Figure B.1: Belief propagation messages can be thought of as perturbations propagating in a network that is in equilibrium. At time $t = 0$, a new image is presented to the network. This causes a perturbation in the equilibrium of the level-1 nodes and messages are propagated up in the hierarchy. At time $t = 1$, these messages arrive at the level-2 nodes. The arrival of these messages trigger feed-forward and feedback messages from the level-2 nodes. At time $t = 2$, the feed-forward messages reach the top of the three-level hierarchy and the feed-back messages from level-2 reach the level-1 nodes. At time $t = 3$, the feedback messages from the level-3 node reach the level-2 nodes and trigger another feedback propagation from the level-2 nodes. At time $t = 4$, these feedback messages, incorporating the effect of evidence from the whole hierarchy, arrive at the level-1 nodes. See [75] for more details on the scheduling of belief propagation messages.

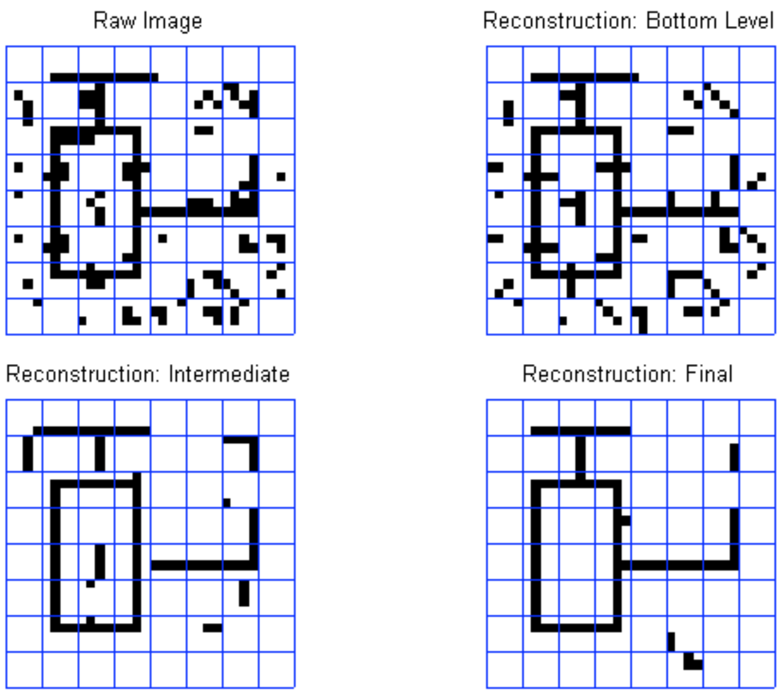


Figure B.2: Feedback reconstruction of the noisy image of a 'helicopter': Top-left: the noisy input image. Top-right: Reconstruction of the image using local information from just the level-1 nodes This reconstruction is done by choosing the coincidence-patten that best matches the input in every 4x4 patch. Bottom-left: Reconstruction of the image at time $t = 2$ after the first set of feedback messages are incorporated. This reconstruction is done by choosing the coincidence that has the maximum *belief* in each level-1 node. Bottom-right: The final reconstruction at time $t = 4$. This reconstruction incorporates evidence from the whole hierarchy.

Appendix C

Calculation of Generalization Properties of HTM Networks

C.1 Definitions

We will be dealing with HTM networks where all the nodes at the same level have similar parameters. It is easy to generalize the computations to the general case. The total number of levels in the network will be denoted by L. Level 1 will be the first level of the hierarchy and level L will be the L^{th} level.

For all the calculations in this section, we abstract the HTM nodes using three parameters. $N_{C,l}$ denotes the number of coincidence-pattern in a node at level-l and $N_{G,l}$ denotes the number of temporal groups (zeroth order Markov chains) in a node at that level. The number of children of nodes at levels $l = 2, 3, \cdots L$ are denoted using B_l, where B stands for branching. We assume that the number of coincidences are evenly distributed into the groups, i.e, each group has the same number of coincidences. Therefore, we consider only those cases where the number of coincidences per group is an integer.

C.2 Number of patterns that can be generated from a network

The number of patterns that can be generated from a network can be calculated easily by first analyzing a small network of two levels and then generalizing it to L levels. Consider a network with one level-2 node and B_2 level-1 nodes that are children of the level-2 node. For any particular coincidence pattern that is generated by the level-2 node, the total number of patterns that can be generated by the network (defined as the concatenation of the coincidence-pattern labels of level-1 nodes) is given by $(N_{C,1}/N_{G,1})^{B_2}$. Therefore, the total number of patterns that are generated by this two-level network is $N_{C,2} \times (N_{C,1}/N_{G,1})^{B_2}$.

Now, consider a three-level network. For any fixed coincidence pattern generated by the node at level-3, we will get $N_{C,3} \times (N_{C,2}/N_{G,2})^{B_3}$ patterns at the output of the level-2 nodes. Each of this pattern has B_3 components and they each independently operate to generate patterns at level-1. The number of patterns generated by the three-level network would then be $N_{C,3} \times (N_{C,2}/N_{G,2})^{B_3}(N_{C,1}/N_{G,1})^{B_3 B_2}$. Generalizing this, we obtain the number of patterns that can be generated form an L-level network as

$$N_w(L) = N_{C,L} \times \left(\frac{N_{C,L-1}}{N_{G,L-1}}\right)^{B_L} \times \left(\frac{N_{C,L-2}}{N_{G,L-2}}\right)^{B_L B_{L-1}} \times \cdots \times \left(\frac{N_{C,1}}{N_{G,1}}\right)^{B_L B_{L-1} \cdots B_2} \tag{C.1}$$

C.3 Probability of fully training the network

Using the training-oracle that we introduced in section 3.8, an HTM node becomes fully trained when it has observed all the possible coincidence-patterns in its corresponding HTW counter part.

Consider a node at level 1. Assume that the world has N_w patterns that were generated from an HTW network of L levels. Then, $N_w/N_{C,1}$ of those patterns will have the first coincidence from the first node at level-1 as their first component. Similarly, any particular coincidence selected from this node will have $N_w/N_{C,1}$ patterns in the world with that coincidence as the corresponding component. If we are selecting patterns from the world without replacement, then any selected pattern is equally likely to have one of the $N_{C,1}$ coincidence pattern in the component corresponding to this node.

Assume that we have drawn N_D patterns from the world without replacement. To calculate the probability that we have all types of coincidence-patterns from our level-1 node in this set of N_D patterns, first define events $A_1, A2, \cdots, A_{N_{C,1}}$ where A_j is the even that coincidence-pattern j was not present in the set of N_D patterns. The probability of event A_j is $\left(\frac{N_{C,1}-1}{N_{C,1}}\right)^{N_D}$.

Then, the probability that all coincidence-patterns are present in the set of N_D patterns can be calculated as

$$p_1 = 1 - P(\cup_{j=1}^{N_{C,1}} A_j) \tag{C.2}$$

This probability can be calculated using the inclusion-exclusion formula (see [86]) as

$$p_1 = 1 - \left(\sum_{i=1}^{N_{C,1}-1} \binom{N_{C,1}}{i} \left(\frac{N_{C,1}-1}{N_{C,1}} \right)^{N_D} (-1)^{i+1} \right) \tag{C.3}$$

and the probability that all the level-1 nodes will have all their coincidence-patterns in a draw of N_D patterns from the world can be calculated by raising p_1 to the power of the number of nodes at level-1. Continuing this reasoning, the probability of fully training the hierarchy in N_D draws can be calculated as

$$P_{train} = \prod_{l=1}^{L} \left(1 - \left[\sum_{i=1}^{N_{C,l}-1} \binom{N_{C,l}}{i} \left(\frac{N_{C,l}-1}{N_{C,l}} \right)^{N_D} (-1)^{i+1} \right] \right)^{B_{l+1}B_{l+2}\cdots B_L} \tag{C.4}$$

Bibliography

[1] James S. Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, 97:270–277, 1975.

[2] Paul Bach-y Rita. Tactile sensory substitution studies. *Ann N Y Acad Sci*, 1013:83–91, 2004.

[3] Paul Bach-y Rita and Stephen W Kercel. Sensory substitution and the human-machine interface. *Trends Cogn Sci*, 7(12):541–546, 2003.

[4] Peter A. Bannister. Inter- and intra-laminar connections of pyramidal cells in the neocortex. *Neurosci Res*, 53(2):95–103, 2005.

[5] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[6] Yoshua Bengio and Yann LeCun. *Scaling Learning Algorithms towards AI.* Large Scale Kernel Machines. MIT Press, Cambridge, MA, 2007.

[7] Pietro Berkes and Laurenz Wiskott. Slow feature analysis yields a rich repertoire of complex cell properties. *Journal of Vision*, 5(6):579–602, 2005.

[8] G. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error-correcting coding: Turbo codes. In *Proceedings International Conference on Communications*, pages 1064–1070, May 1993.

[9] Dimitri P. Bertsekas. *Dynamic Programming: Deterministic and Stochastic Models.* Prentice-Hall, Englewood Cliffs, N.J., 1987.

[10] Christopher M. Bishop. *Pattern Recgonition and Machine Learning*. Springer, 2006.

[11] V. Braitenberg and A. Schuz. *Cortex: Statistics and Geometry of Neuronal Connections*. Springer-Verlag, Berlin, 1991.

[12] Daniel P Buxhoeveden and Manuel F Casanova. The minicolumn hypothesis in neuroscience. *Brain*, 125(Pt 5):935–951, 2002.

[13] E.J. Candes and M.B. Wakin. An introduction to compressive sampling [a sensing/sampling paradigm that goes against the common knowledge in data acquisition]. *Signal Processing Magazine, IEEE*, 25(2):21–30, March 2008.

[14] N. Chaddha, S. Mehrotra, and R.M. Gray. Finite state hierarchical table-lookup vector quantization for images. *icassp*, 4:2024–2027, 1996.

[15] Mark A. Changizi. Universal scaling laws for hierarchical complexity in languages, organisms, behaviors and other combinatorial systems. *J Theor Biol*, 211(3):277–295, 2001.

[16] Patricia S. Churchland and Terrence J. Sejnowski. *The Computational Brain*. MIT Press, Cambridge, Massachusetts, 1992.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, 1991.

[18] David D. Cox, Philip Meier, Nadja Oertelt, and James J. DiCarlo. 'breaking' position-invariant object recognition. *Nat Neurosci*, 8(9):1145–1147, 2005.

[19] Sophie Deneve. *Bayesian inference in spiking neurons*, pages 353–360. Advances in Neural Information Processing Systems 17. MIT Press, 2005.

[20] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[21] D.L. Donoho. Compressed sensing. *Information Theory, IEEE Transactions on*, 52(4):1289–1306, April 2006.

[22] Rodney J. Douglas and Kevan A. Martin. Neuronal circuits of the neocortex. *Annual Review of Neuroscience*, 27:419–451, 2004. LR: 20041117; JID: 7804039; RF: 176; ppublish.

[23] Rodney J. Douglas and Kevan A. C. Martin. Mapping the matrix: the ways of neocortex. *Neuron*, 56(2):226–238, 2007.

[24] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley and Sons, New York, 1973.

[25] R. O. Duda, P. E. Hart, and David G. Stork. *Pattern Classification.* Second edition, John Wiley and Sons, New York, NY, 2001.

[26] Guy N Elston. Cortical heterogeneity: implications for visual processing and polysensory integration. *J Neurocytol*, 31(3-5):317–335, 2002.

[27] Dirk Feldmeyer, Joachim Lubke, and Bert Sakmann. Efficacy and connectivity of intracolumnar pairs of layer 2/3 pyramidal cells in the barrel cortex of juvenile rats. *J Physiol*, 575(Pt 2):583–602, 2006.

[28] D. J. Felleman and D. C. Van Essen. Distributed hierarchical processing in the primate cerebral cortex. *Cereb Cortex*, 1(1):1–47, 1991.

[29] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden Markov model: Analysis and applications. *Machine Learning*, 32(1):41–62, 1998.

[30] Brendan J. Frey and David J. C. MacKay. A revolution: Belief propagation in graphs with cycles. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

[31] K. Fukushima. Neocognitron: A self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernnetics*, 36(4):93–202, 1980.

[32] Kunihiko Fukushima, Sei Miyake, and Takayuki Ito. Neocognitron: A neural network model for a mechanism of visual pattern recognition. In *Artificial neural networks: theoretical concepts*, pages 136–144. IEEE Computer Society Press, 1988.

[33] S. Geman, D. Potter, and Z. Chi. Composition systems. *Quarterly of Applied Mathematics*, LX:707–736, 2002.

[34] Dileep George and Jeff Hawkins. Belief propagation and wiring length optimization as organization principles for cortical microcircuits. Technical report, Redwood Neuroscience Institute, 2005.

[35] Dileep George and Jeff Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks*, volume 3, pages 1812–1817. IEEE, 2005.

[36] Diana F. Gordon and Marie desJardins. Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1-2):5–22, 1995.

[37] Richard Granger. Engines of the brain: The computational instruction set of human cognition. *AI Magazine*, 27(2):15–31, Summer 2006.

[38] Stephen Grossberg. Towards a unified theory of neocortex: laminar cortical circuits for vision and cognition. *Prog Brain Res*, 165:79–104, 2007.

[39] R W Guillery and S M Sherman. The thalamus as a monitor of motor outputs. *Philos Trans R Soc Lond B Biol Sci*, 357(1428):1809–1821, 2002.

[40] Stefan Haeusler and Wolfgang Maass. A statistical analysis of information-processing properties of lamina-specific cortical microcircuit models. *Cereb Cortex*, 17(1):149–162, 2007.

[41] Mathew Harrison. *Discovering Compositional Structures.* PhD thesis, Brown University, Providence, RI, 2005.

[42] Uri Hasson, Eunice Yang, Ignacio Vallines, David J Heeger, and Nava Rubin. A hierarchy of temporal receptive windows in human cortex. *J Neurosci*, 28(10):2539–2550, 2008.

[43] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Henry Holt and Company, New York, 2004.

[44] Jeff Hawkins and Dileep George. Hierarchical temporal memory: Concepts, theory and terminology. `http://www.numenta.com/Numenta_HTM_Concepts.pdf`, 2006.

[45] Jay Hegde and Daniel J Felleman. Reappraising the functional implications of the primate visual anatomical hierarchy. *Neuroscientist*, 13(5):416–421, 2007.

[46] C C Hilgetag, M A O'Neill, and M P Young. Indeterminate organization of the visual system. *Science*, 271(5250):776–777, 1996.

[47] G.E. Hinton and T.J. Sejnowski. Learning and relearning in Boltzmann machines. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume I: Foundations*, pages 282–317. MIT Press, Cambridge, Massachusetts, 1986.

[48] Geoffrey E. Hinton. Learning multiple layers of representation. *Trends Cogn Sci*, 11(10):428–434, 2007.

[49] Judith A. Hirsch and Luis M. Martinez. Laminar processing in the visual cortical column. *Curr Opin Neurobiol*, 16(4):377–384, 2006.

[50] Y. C. Ho and D. L. Pepyne. Simple explanation of the no-free-lunch theorem and its implications. *Journal of Optimization Theory and Applications*, V115(3):549–570, December 2002.

[51] Carl Holmgren, Tibor Harkany, Bjorn Svennenfors, and Yuri Zilberter. Pyramidal cell communication within local networks in layer 2/3 of rat neocortex. *J Physiol*, 551(Pt 1):139–153, 2003.

[52] Jonathan C Horton and Daniel L Adams. The cortical column: a structure without a function. *Philos Trans R Soc Lond B Biol Sci*, 360(1456):837–862, 2005.

[53] Ronald A. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, Massachusetts, 1960.

[54] Stephan C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[55] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[56] J Kozloski, F Hamzei-Sichani, and R Yuste. Stereotyped position of local synaptic targets in neocortex. *Science*, 293(5531):868–872, 2001.

[57] O E Krigolson and C B Holroyd. Evidence for hierarchical error processing in the human brain. *Neuroscience*, 137(1):13–17, 2006.

[58] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series, 1995.

[59] T. S. Lee and M. Nguyen. Dynamics of subjective contour formation in the early visual cortex. *Proc Natl Acad Sci U S A*, 98(4):1907–1911, 2001.

[60] Tai Sing Lee and David Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, 2(7):1434–1448, July 2003.

[61] S. Levinson. Continuously variable duration hidden markov models for speech analysis. *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '86.*, 11:1241–1244, Apr 1986.

[62] Jyh-Han Lin and Jeffrey Scott Vitter. A theory for memory-based learning. *Mach. Learn.*, 17(2-3):143–167, 1994.

[63] Joachim Lubke and Dirk Feldmeyer. Excitatory signal flow and connectivity in a cortical column: focus on barrel cortex. *Brain Struct Funct*, 212(1):3–17, 2007.

[64] Jennifer S. Lund, Alessandra Angelucci, and Paul C. Bressloff. Anatomical substrates for functional columns in macaque monkey primary visual cortex. *Cerebral cortex (New York, N.Y. : 1991)*, 13(1):15–24, 2003.

[65] Kevan A C Martin. Microcircuits in visual cortex. *Curr Opin Neurobiol*, 12(4):418–425, 2002.

[66] Tom M. Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, New Brunswick, New Jersey, 1980.

[67] Berman A.L Mountcastle, V. B. and P.W. Davies. Topographic organization and modality representation in first somatic area of cat's cerebral cortex by method of single unit analysis.

[68] V B Mountcastle. An organizing principle for cerebral function: the unit module and the distributed system. In *The Mindful Brain*. MIT Press, 1978.

[69] Vernon B. Mountcastle. The columnar organization of the neocortex. *Brain*, 120(4):701–722, 1997.

[70] Vernon B. Mountcastle. Introduction to the special issue on computation in cortical columns. *Cerebral Cortex*, 13(1):2–4, January 2003.

[71] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy-belief propagation for approximate inference: An empirical study. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufmann, 2000.

[72] S. O. Murray, D. Kersten, B. A. Olshausen, P. Schrater, and D. L. Woods. Shape perception reduces activity in human primary visual cortex. *Proceedings of the*

National Academy of Sciences of the United States of America, 99(23):15164–15169, Nov 12 2002. LR: 20041117; DEP: 20021104; JID: 7505876; 2002/11/04 [aheadofprint]; ppublish.

[73] B. A. Olshausen, A. Anderson, and D. C. Van Essen. A neurobiological model of visual attention and pattern recognition based on dynamic routing of information. *Journal of Neuroscience*, 13(11):4700–4719, 1993.

[74] H. H Pattee. *Hierarchy theory: the challenge of complex systems.* The International library of systems theory and philosophy. G. Braziller, New York, 1973.

[75] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Francisco, California, 1988.

[76] Judea Pearl. *Causality : Models, Reasoning, and Inference.* Cambridge University Press, March 2000.

[77] Karl Pflegerr. *Online Learning of Predictive Compositional Hierarchies.* PhD thesis, Stanford University, 2002.

[78] T Poggio and S Smale. The mathematics of learning: Dealing with data. *Notices of the American Mathematical Society (AMS*, (50):2003, 2003.

[79] Tomaso Poggio and Emilio Bizzi. Generalization in vision and motor control. *Nature*, 431(7010):768–774, 2004.

[80] Tomaso Poggio and Federico Girosi. A theory of networks for approximation and learning. Technical Report AI Memo No. 1140, MIT AI Laboratory, 1989.

[81] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[82] P Rakic. Specification of cerebral cortical areas. *Science*, 241(4862):170–176, 1988.

[83] Rajesh P. N. Rao. Hierarchical Bayesian inference in networks of spiking neurons. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[84] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2(11):1019–1025, November 1999.

[85] R. L. Rivest and R. Sloan. A formal model of hierarchical concept learning. *Information and Computation*, 114(1):88–114, 1994.

[86] Sheldon M. Ross. *A First Course in Probability, Fourth Edition*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[87] Phil Sallee and Bruno A. Olshausen. Learning sparse multiscale image representations. In *Advances in Neural Information Processing Systems*, volume 16, pages 1327–1334, 2004.

[88] S Murray Sherman and R W Guillery. The role of the thalamus in the flow of information to the cortex. *Philos Trans R Soc Lond B Biol Sci*, 357(1428):1695–1708, 2002.

[89] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, Cambridge, Massachusetts, 1981.

[90] Andrea Caponnetto Steve Smale, Tomaso Poggio and Jake Bouvrie. Derived distance: towards a mathematical theory of visual cortex. Technical report, MIT CBCL, 2007.

[91] Simon M Stringer and Edmund T Rolls. Invariant object recognition in the visual system with novel views of 3d objects. *Neural Comput*, 14(11):2585–2596, 2002.

[92] N V Swindale. How many maps are there in visual cortex? *Cereb Cortex*, 10(7):633–643, 2000.

[93] Joseph A. Tatman and Ross D. Shachter. Dynamic programming and influence diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(2):365–379, 1990.

[94] G W Taylor, G E Hinton, and S Roweis. Modeling human motion using binary latent variables. In *Advances in Neural Information Processing Systems*, page 2007. MIT Press, 2006.

[95] A M Thomson and A P Bannister. Postsynaptic pyramidal target selection by descending layer iii pyramidal axons: dual intracellular recordings and biocytin filling in slices of rat neocortex. *Neuroscience*, 84(3):669–683, 1998.

[96] Alex M. Thomson and A. Peter Bannister. Interlaminar connections in the neocortex. *Cerebral cortex (New York, N.Y. : 1991)*, 13(1):5–14, 2003.

[97] Alex M. Thomson and Christophe Lamy. Functional maps of neocortical local circuitry. *Front. Neurosci.*, 1(1):19–42, 2007.

[98] Kevan A.C. Martin Tom Binzegger, Rodney J. Douglas. *Brain, Vision, and Artificial Intelligence*, chapter Cortical Architecture, pages 15–28. Springer Berlin / Heidelberg, 2005.

[99] K Tsunoda, Y Yamane, M Nishizaki, and M Tanifuji. Complex objects are represented in macaque inferotemporal cortex by the combination of feature columns. *Nat Neurosci*, 4(8):832–838, 2001.

[100] Shimon Ullman. Object recognition and segmentation by a fragment-based hierarchy. *Trends Cogn Sci*, 11(2):58–64, 2007.

[101] Shimon Ullman and Evgeniy Bart. Recognition invariance obtained by extended and invariant features. *Neural Netw*, 17(5-6):833–848, 2004.

[102] L G Ungerleider and J V Haxby. 'what' and 'where' in the human brain. *Curr Opin Neurobiol*, 4(2):157–165, 1994.

[103] L von Melchner, S L Pallas, and M Sur. Visual behaviour mediated by retinal projections directed to the auditory pathway. *Nature*, 404(6780):871–876, 2000.

[104] Brian A Wandell, Serge O Dumoulin, and Alyssa A Brewer. Visual field maps in human cortex. *Neuron*, 56(2):366–383, 2007.

[105] BT Werner. Complexity in natural landform patterns. *Science*, 284(5411):102–104, 1999.

[106] H. Wersing and E. Korner. Learning optimized features for hierarchical models of invariant object recognition. *Neural computation*, 15(7):1559–1588, Jul 2003. JID: 9426182; ppublish.

[107] Heiko Wersing, Julian Eggert, and Edgar Körner. Sparse coding with invariance constraints. pages 385–392, 2003.

[108] C M Wessinger, J VanMeter, B Tian, J Van Lare, J Pekar, and J P Rauschecker. Hierarchical organization of the human auditory cortex revealed by functional magnetic resonance imaging. *J Cogn Neurosci*, 13(1):1–7, 2001.

[109] Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[110] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1):67–82, 1997.