

The HTM Learning Algorithms

Dileep George and Bobby Jaros, Numenta Inc.

1 Introduction

This paper describes the learning algorithms and concepts behind Hierarchical Temporal Memory (HTM). This description is applicable to the Research Release of the Numenta Platform for Intelligent Computing. The fundamental concepts and terminology behind HTMs were introduced in an earlier white paper from Numenta.¹ This document assumes familiarity with that material and is intended for readers who want to learn more about the nuts and bolts of HTMs.

Successful development of HTM applications requires understanding at several levels. In addition to understanding the software platform, a developer will require an intuitive understanding of the learning algorithms within HTM nodes. Developers will also require an understanding of how these concepts hold together to achieve system-level results. Although these ideas are simple, we find that it is hard to commit them to memory when they are introduced in a vacuum. For this reason, we chose to describe our algorithms using visual pattern recognition as an illustrative example.

The rest of this paper is organized in ten sections and can be considered in three logical blocks. The first block, covered by sections 2 and 3, is about the vision problem. In these sections, we will introduce the reader to the nature of the vision problem, describe how HTMs are good for solving it and define a simple pattern recognition problem that will act as our vehicle for the rest of the paper. The second logical block is the mechanics of HTMs. These are covered in sections 4 to 7. In these sections, we describe in detail the learning algorithms within a node and how nodes operate in a hierarchy to achieve system level results. The third block, covered in sections 8 through 9, is about HTMs in practice. In these sections we will describe experimental results. We will also describe the bigger picture behind this paper and connect the vision problem to other problems in the world. We conclude in section 10 and summarize the key points.

Some sections of this paper can be easily skipped on a first reading. Those less concerned with the implementation details might want to scan over sections 4.1 and 4.2. Readers who are familiar with the vision problem can skip to section 3. The description throughout the main body is fairly qualitative. In the appendix, we have a more rigorous description of Belief Propagation in HTMs.

¹This white paper, entitled *Hierarchical Temporal Memory: Concepts, Theory and Terminology* can be obtained from Numenta's website at www.numenta.com

2 The Vision Problem

Vision is the primary sensory modality for humans and most mammals to perceive the world. In humans, vision related areas occupy about 30 percent of the neocortex. 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, our brains solve many problems in the process that can not yet be solved by a computer.

Perhaps the most imposing of these problems is that 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. In most cases, the problem is defined as having labeled examples from a certain number of categories of objects. Given these training examples, most approaches either pre-program or learn low-level statistical patterns (“features”) of the images, then try to map combinations of these features to the correct categories using a simple classifier. Some level of invariance is achieved when trained with translations, deformations, and size changes of the original objects. However, these techniques are not yet able to adequately generalize from patterns they were trained on to patterns they have not seen.

How can we improve on these approaches? 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. Since humans can recognize objects from a single snapshot of its image without integrating information over multiple time steps, many researchers have ignored the temporal component of vision and thought of it in terms of still images, not videos. We believe this approach is misguided. Although humans can recognize with a snapshot, we *learn* with continuously varying data and use this temporal information to ascertain important generalization characteristics. 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 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 reveals the unsupervised nature of the vision problem.

For example, consider a dog walking towards its water bowl. With each step the dog takes, a different image of the bowl falls on the dog’s retina: we can imagine the retinal image as a pixelated image. The pixel-wise distance between two consecutive images on the dog’s retina can be quite large. However, the dog still knows that it is looking at the same bowl even if it is not able to name it as a bowl. Nobody taught the dog about bowls by making it flip through pages of bowl pictures while shouting “bowl” in its ears. The dog had to learn in a completely unsupervised manner that the different images of the bowl are actually caused by the same bowl.

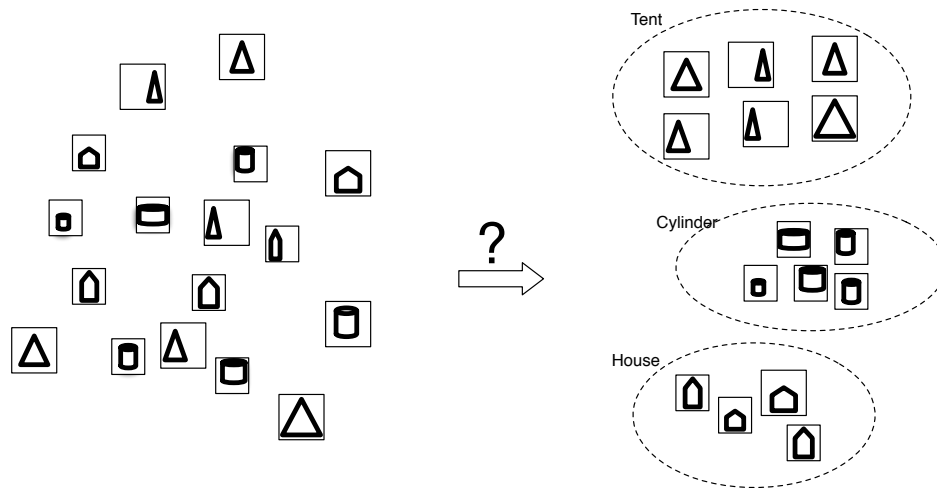


Figure 1: Unsupervised learning of object categories: On the left, we show images belonging to three categories - “tent”, “house” and “cylinder”. Unsupervised learning of categories involve separating these images into their categories shown on the right.

The difficulty of this problem can be understood by looking at figure 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 the right in figure 1? A naive application of similarity metrics at the pixel level will not achieve the desired result because typically images from different categories have more overlap in the pixel space than images from the same category. (See Section 3)

There are many other options for mapping a given image to a category than with pixel-wise distance metrics. Just as pixel-wise distance fails, however, so too will most other mapping functions. Although we can subjectively evaluate the different functions, there is no reason to prefer, *a priori*, one of them over any other – just as there is no reason to think that pixel-wise distances would fail so badly until we try it on real problems. We also cannot hope to get lucky, given that only a miniscule fraction of mapping functions would remotely match the correct mapping from visual pattern to object name. (In figure 1, the grouping we show on the right side is just one of a combinatorially large number of possible groupings.)

Fortunately, our dog does not have to solve this problem in this manner. For the dog, images in this world are presented continuously in time. When there is relative motion between the dog and the bowl, different images of the bowl are produced on the dog’s retina. Although these images can be very distant from one another by most metrics, the very fact that they occurred close in time hints that they are the same object, since motion in the physical world obeys the locality and inertial laws of physics.

Although two different images of the 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. ²

The identity of an object can be thought of as an *invariant representation* for that object - different images of the object have the same identity. When an object undergoes different transformations in the world, its identity remains the same. Learning to recognize objects involves learning invariant representations.

Note that once the invariant representations are learned using temporal information, recognition is easy for single snapshots. It is only the learning and separation of the invariant representations for different objects in an unsupervised manner that requires temporal information.

By using time as the supervisor we are able to separate the representations for two objects A and B. However, learning the invariant representation of object A does not help us in learning the invariant representation 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 into the picture. Most of the real-world objects share the same building blocks. Suppose that a visual recognition system can learn the invariant representations 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 re-configurations 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, what we learned about one object can be used to learn another object.

The above discussion gives two main insights on how to create a system for invariant visual pattern recognition:

- Use temporal information to determine whether two different images (patterns) belong to the same object or not. This allows us to create invariant representations.
- Perform this learning in a hierarchy such that invariant representations of simpler object components are learned first and invariant representations of larger objects are learned in terms of these invariant representations of simpler components.

²A recent experiment showed that monkeys can be fooled into believing two different objects are the same by making those objects follow one another in time for several presentations. This supports the argument that the cortex uses time as a supervisor. Cox DD, Meier P, Oertelt N, and DiCarlo JJ. Breaking position invariant object recognition. *Nature Neuroscience* 8:1145-1147 (2005).

3 The Pictures Example

In this section we will describe *Pictures*, our example for the rest of this paper. *Pictures* is 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.

3.1 Task

Our goal is to use an HTM network to learn invariant representations of a simplified set of binary image categories, of size 32 pixels by 32 pixels. Figure 2(a) shows some examples of these images. The problem of invariant representation can be understood by looking at figure 2 (a) and (b). In figure 2 (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, scalings 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 (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*.

3.2 Training Data

In the real visual world, transformations like translation, rotation and scaling occur due to relative motion between objects and the viewer. In section 2.1, we described that HTMs exploit the temporal continuity of these transformations to learn representations that are invariant to transformations. 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 are created by simulating frames of smooth translations, rotations and scaling. We need to include in the movies all the transformations that we want the learning system to be invariant to. Figure 3 shows 4 frames of an example movie sequence. In this sequence, the “cat” object moves to the right one pixel at a time. The visual world for our problem will consist of such simulated movies of binary line drawings. We refer to this simplified visual problem set-up (the 32-pixel by 32-pixel drawings, turned into movies) as the *Pictures World*.

3.3 Network Structure

The HTM network we use to model this visual world is organized in a 3 level hierarchy. Figure 4 shows how our HTM network interfaces with the 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 the lowest level (Level 1) of the network receive inputs from the retina. Each node receives its input from a 4x4 pixel patch of the retina as shown in figure 4. The nodes at the lowest level are arranged in an 8x8

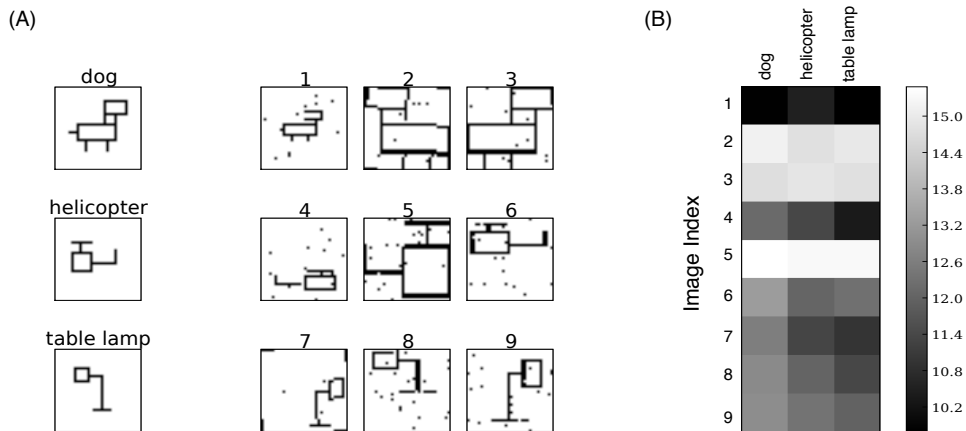


Figure 2: (a) The left column of this image shows prototypical examples of three categories - “dog”, “helicopter” and “table lamp”. On the right side are distorted versions of those categories. (b) This figure shows the Euclidean distance between the prototype of each category and the distorted images 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”, is classified as a “table lamp”. It can be concluded from this plot that a Euclidean distance measure does not inform us about perceptual similarity.

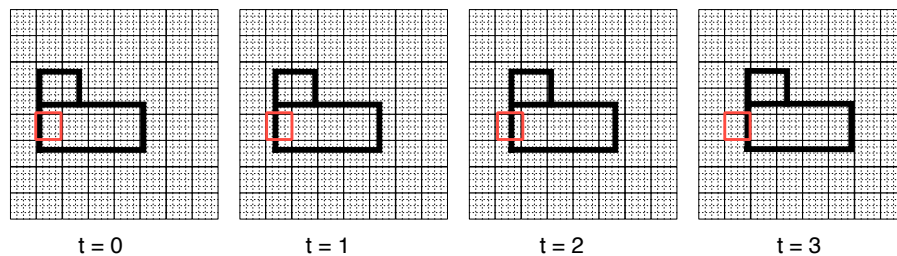


Figure 3: This figure shows 4 consecutive frames from a training video for the network. These frames show the line drawing of a “cat” moving from left to right. Each frame is of 32 pixels by 32 pixels.

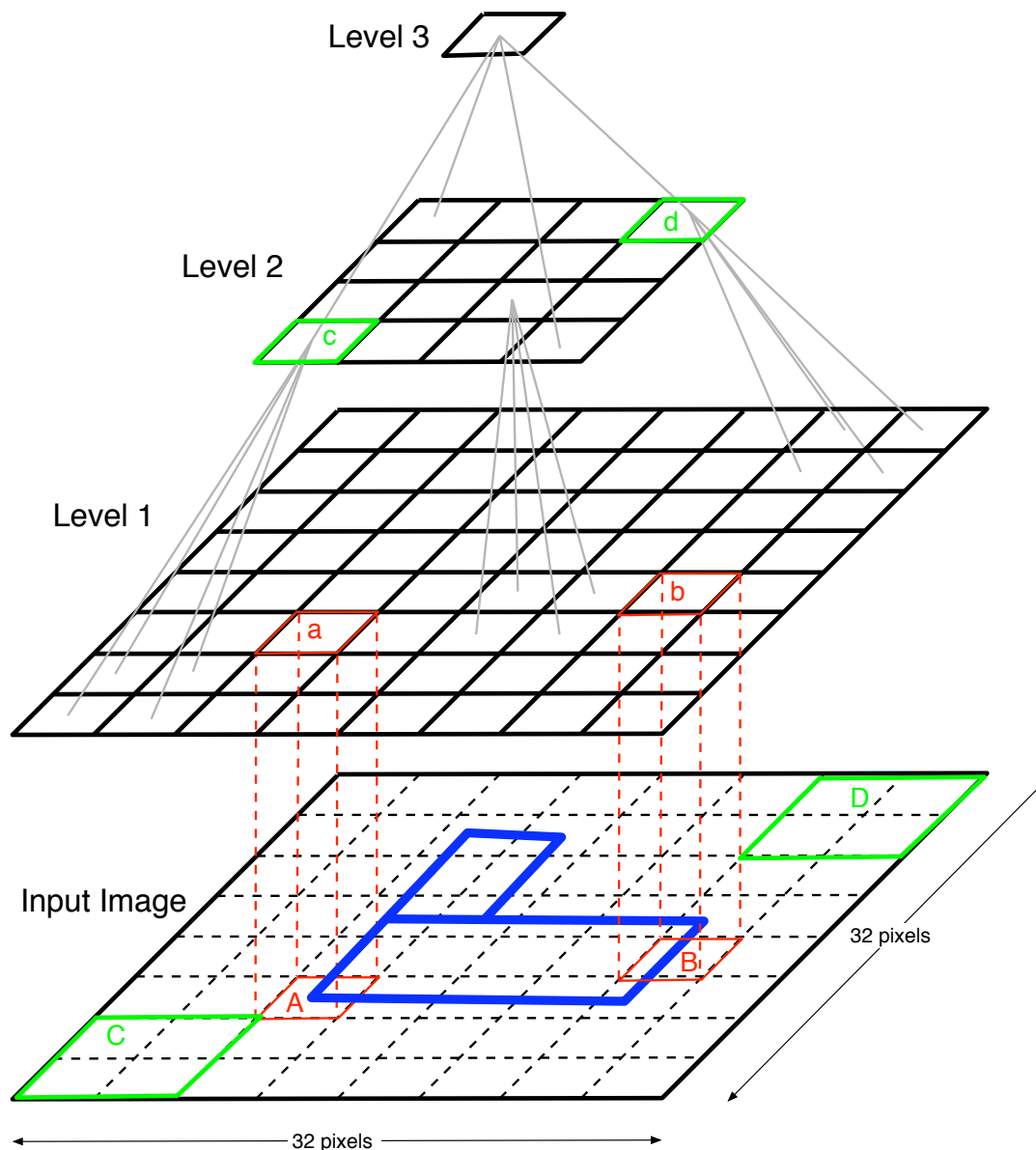


Figure 4: Structure of an HTM network for learning invariant representations for the binary images world. This network is organized in 3 levels. Input is fed in at the bottom level. Nodes are shown as squares. The top level of the network has one node, the middle level has 16 nodes and the bottom level has 64 nodes. The input image is of size 32 pixels by 32 pixels. This image is divided into adjoining patches of 4 pixels by 4 pixels as shown. Each bottom-level node's input corresponds to one such 4x4 patch. The figure shows how image patches are mapped to the bottom-level nodes. 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. Similarly, the squares marked (C) and (D) correspond to the receptive fields of nodes at level-2 marked (c) and (d).

grid so that the whole 32x32 retina is covered by these nodes without overlap. A node at a higher level receives its input from several nodes at the lower level. In figure 4 a level 2 node receives its input from the outputs of 4 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 4 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 4, the size of the receptive field of a node increases as we go up in the hierarchy. The node at the root of the tree covers the entire visual field, by pooling inputs from its child nodes.

3.4 Stages of Learning

The HTM network operate in two distinct stages - *training* and *inference*.³ During the training stage, the network is exposed to movies as described above, and the nodes in the network form representations of the world using the learning algorithms we describe below. When learning is complete, the network is switched to inference mode. During 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 restrict the inference stage in our discussion to static image recognition. During this stage, we present an image (a single snapshot) and ask the network to identify the category of that image. This information is read out at the top of the network.

All nodes in the network use learning and inference algorithms that are based on the same principles. Hence understanding the operation of the network involves only two things:

- understanding the operations within a single node (section 4)
- understanding how a node operates with other nodes in the hierarchy (sections 5 and 7)

4 Structure and Operation of a Node

The input to a node, irrespective of its position in the hierarchy, is a temporal sequence of patterns. Consider the Level-1 node marked (a) in figure 4. The input to this node is an image patch of size 4 pixels by 4 pixels. In figure 4, this input corresponds to a “corner”. The node considers this input as a pattern of length 16. If the “cat” in the picture moves to the right in the next frame of the movie, this node gets a different pattern - a shifted corner - as input in the next time instant. The input space of a node is a temporal sequence of such patterns.

Figure 5(a) shows examples of several patterns this node observes as movies of different objects are played through its receptive field. The goal of this node is to form pools or groups of these patterns such that patterns belonging to the same group are likely to be variations of the same thing in the

³For a node, the equivalent stages are *learning* and *inference*. The reason for the terminology difference is that typically some nodes are in inference while other parts of the network are still being trained. See the introduction to section 5

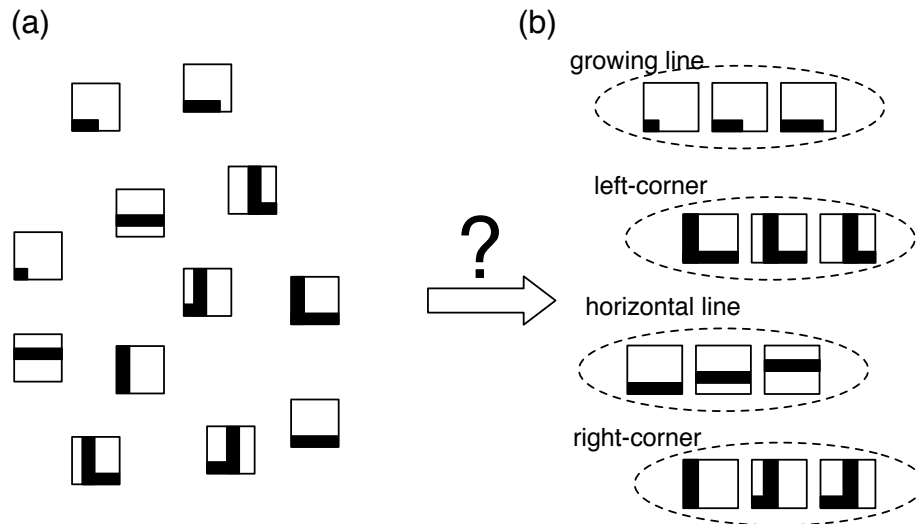


Figure 5: The goal of the node is to form pools or groups of patterns on the left hand side such that patterns belonging to the same group are likely to be variations of the same thing.

visual world. One source of variation in the visual world is the relative motion between objects and the viewer. Another source of variation is random noise. If the node can group together visual patterns that correspond to variations of the same original pattern, then these groups will be invariant to such variations. Figure 5(b) shows some groups that are invariant to relative motion: the different motions of “corners” are all considered to be the same thing. (In fact we name all these things “corners” precisely because our brains have learned that these variations correspond to the same thing.) Once it has formed these groups (formed an invariant representation), the node can start producing outputs – summarizing each input it receives by which invariant group to which it belongs.

A node uses two kinds of pooling mechanisms to form its invariant representations. (These two kinds of pooling mechanism roughly correspond to the two kinds of variations we discussed above: variations due to noise and variations due to motion). The first kind of pooling mechanism pools patterns according to their pixel-wise similarity. We call this mechanism *spatial pooling*. Spatial pooling is useful for removing noise from a pattern, but ineffective at solving other invariances for reasons described in section 2.1 and visualized in figure 2. Spatial pooling can be thought of as a quantization process that maps a potentially infinite number of input patterns to a finite number of quantization centers.

The second pooling mechanism used by a node is *temporal pooling*. This mechanism pools together patterns using their temporal proximity. If pattern *A* is frequently followed by pattern *B*, this mechanism can assign them to the same group. This kind of pooling mechanism is very powerful because it can pool together patterns that are very dissimilar from a pixel-similarity perspective. For example, this mechanism can group together a corner and a shifted corner. Temporal pooling can learn a wider class of invariances than spatial pooling, but cannot operate on an infinite number

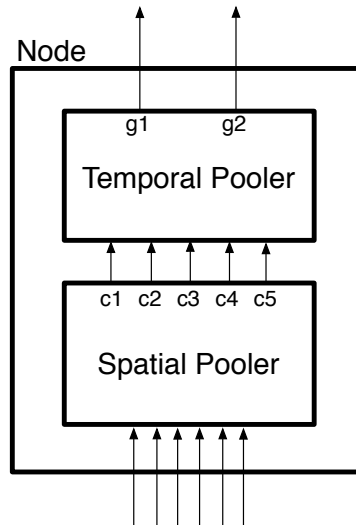


Figure 6: Structure of a node: A node has a spatial pooler and a temporal pooler. The input to the node is connected to the input of the spatial pooler. The output of the spatial pooler is the input to temporal pooler. The output of the temporal pooler is the output of the node. The node shown in this figure has an input pattern size 6. The spatial pooler within this node has 5 quantization centers marked c_1 to c_5 . The output of the spatial pooler is a vector of length 5. The temporal pooler within this node has 2 groups marked g_1 and g_2 . The output of the temporal pooler, and hence of the node, is a vector of size 2.

of points; for this reason, spatial pooling precedes temporal pooling to quantize the input space and offer a finite alphabet to the temporal pooler.

A node contains two modules to perform the above two steps, as shown in figure 6:

1. **spatial pooler:** Learns a mapping from a potentially infinite number of input patterns to a finite number of *quantization centers*. The output of the spatial pooler is in terms of its quantization centers. We discuss spatial pooler in detail in section 4.1
2. **temporal pooler:** Learns *temporal groups* - groups of quantization centers - according to the temporal proximity of occurrence of the quantization centers of the spatial pooler. The output of the temporal pooler is in terms of the temporal groups that it has learned. We discuss temporal pooler in detail in section 4.2

These modules do not implement a pre-determined function, but must learn an output mapping by observing their inputs. Therefore, both these modules must undergo a learning phase before they switch to an inference phase. The stages of operation of a node are illustrated in figure 7. We will come back to this figure while we look at the operation of spatial pooler and temporal pooler in detail in the following sections.

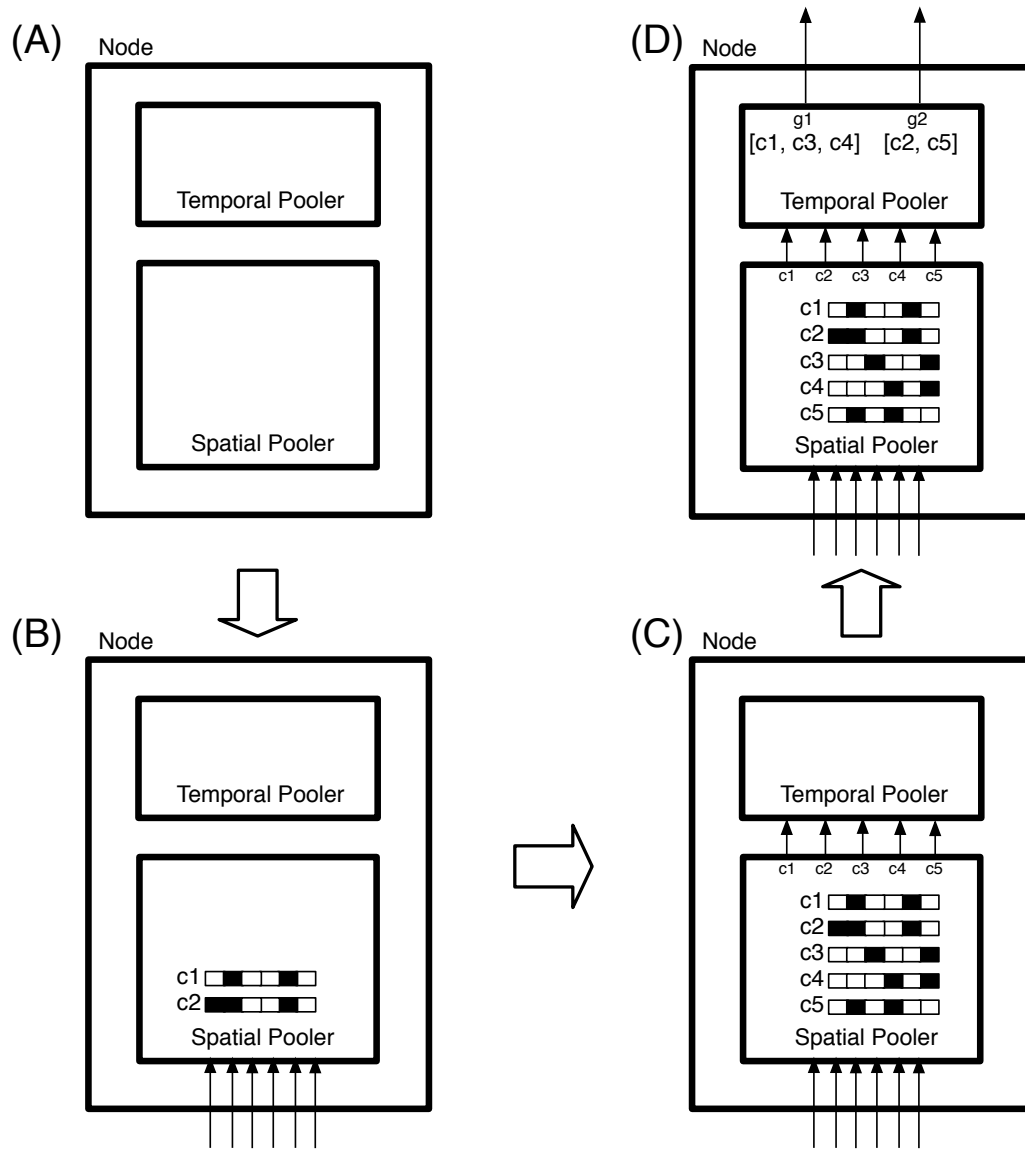


Figure 7: Different stages of a node: Illustration of the different stages of node. (1) shows and an initial node - a node that has not started its learning process. In (2), the spatial pooler of the node is in its learning phase. At this time, it is receiving inputs and has formed 2 quantization centers. In (3) the spatial pooler has finished its learning process and is in the inference stage. The spatial pooler now has $N_c = 5$ quantization centers. Each quantization center is a pattern of length 6. At this stage, the temporal pooler is receiving inputs and learning the time-adjacency matrix. (4) shows a fully learned node where both the spatial pooler and temporal pooler have finished their learning processes. The temporal pooler now has $N_g = 2$ temporal groups. The node, at this stage, is ready to produce outputs.

4.1 Operation of the spatial pooler

The spatial pooler has two stages of operation.

- During the learning stage it quantizes the input patterns and memorizes the quantization centers.
- Once these quantization centers are learned, it produces outputs in terms of these quantization centers. This is the inference stage.

4.1.1 Learning quantization points in the spatial pooler

Consider again the node marked (a) at level-1 of the hierarchy in figure 4. The input to this node is a patch of size 4 pixels by 4 pixels from the overall input image. The node considers this input as a pattern of length 16. As the network is exposed to frames from a movie, this node gets exposed to a series of patterns. These patterns are the inputs to the spatial pooler and the spatial pooler learns a quantization of these input patterns. We describe a simple algorithm for doing this. This algorithm takes in a maximum distance parameter D which corresponds to the minimum separation at which a pattern is considered different from the existing quantization patterns.

For every input pattern, check whether there is a quantization center that is within Euclidean distance D from this pattern.

- If yes, do nothing.
- If no, add the new pattern to the list of quantization centers.

With this method, the spatial pooler stores a small subset of its input patterns as quantization centers. Each quantization center is a vector of length equal to the length of input patterns. Note that, in this simple quantization scheme, the center does not move from its original value - a pattern is fixed upon storage.

The parameter D in the spatial pooler should be large enough to counter the variations in input patterns that are caused by noise. If D is too small, the spatial pooler will generate too many quantization centers. The parameter D should also be small enough so that it doesn't group together unrelated patterns. For example, if D is too large, it might group together a line and a corner.

At the beginning of learning, the node rapidly adds new quantization centers. As more quantization centers are added the amount of space that is not mapped to an existing quantization center decreases. Therefore, over time, the number of new quantization centers that a spatial pooler forms within a unit of time decreases. This is shown in figure 8. The quantization process can be stopped when the number of new quantization centers per time period falls below some threshold.

Figure 9 shows some of the quantization centers that the lower level node learned when it was exposed to the line-drawing movies described earlier. The figure shows a total of 150 quantization centers arranged in a 10x15 grid. Each quantization center is a vector of size 16 which corresponds

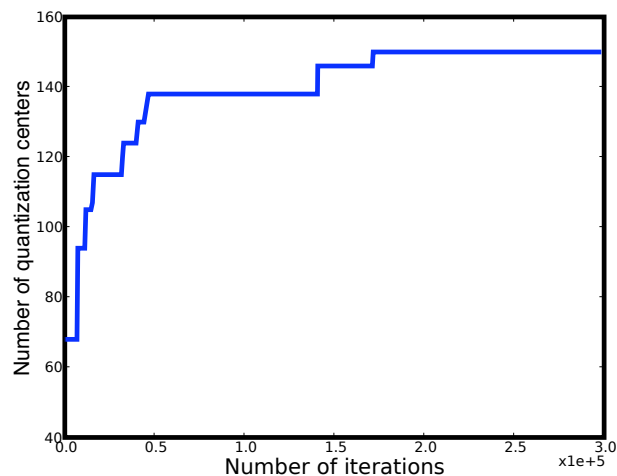


Figure 8: The number of quantization centers stored in a bottom-level node as a function of the number of images to which the network is exposed. Note the steep rise in the number of quantization centers initially and the gradual rise towards the end

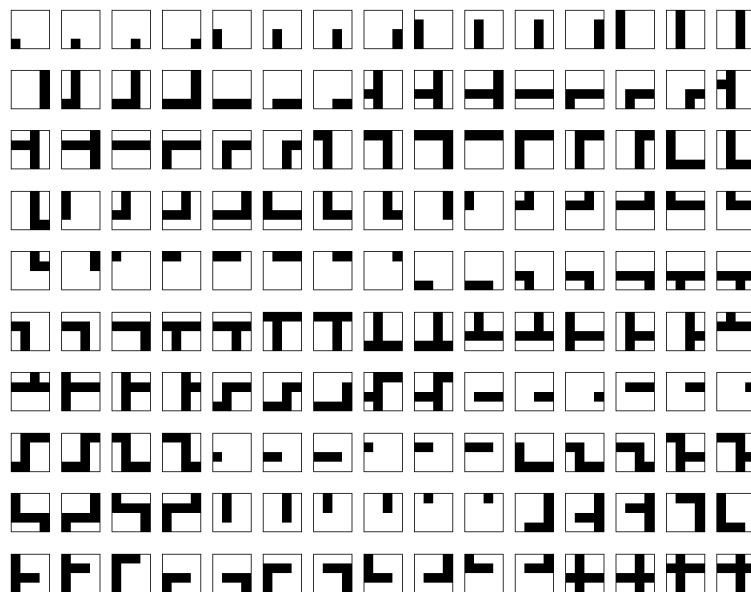


Figure 9: Quantization centers learned by a bottom-level node when it was exposed to videos of line drawing images. Note that, in the Pictures World, these can be obtained with $D=0$.

to a 4x4 pixels patch of visual input. Because the training inputs in the Pictures World are noise-free, we set $D = 0$ for this experiment.

4.1.2 Inference in the spatial pooler

Once the quantization process is finished, the spatial pooler can be switched to inference mode.⁴ In this mode, the spatial pooler produces outputs for every input pattern.

We use N_c to represent the total number of quantization centers that are stored within a node and c_1, c_2, \dots, c_{N_c} to represent the individual quantization centers. The output of the spatial pooler is a vector of length N_c . The i^{th} position in this vector corresponds to c_i , the i^{th} quantization center stored within the node's spatial pooler. In general, this output is a probability distribution on the space of quantization centers. This distribution can be thought of as indicative of the degree of match of the input pattern to the stored quantization centers. In section 5.3 we show that the quantization centers of a higher-level node has some special meaning that can be used to calculate this degree of match. In this section we describe how to calculate the degree of match of an input to the stored quantization centers.

The first step in calculating the degree of match between an input and the quantization center is to calculate the Euclidean distance between the input and the quantization centers. Let d_i be the distance of the i^{th} quantization center from the input. The larger this distance, the smaller the match between the input and the quantization center. If we assume that the probability that a pattern belongs to a quantization center falls off as a Gaussian function of the Euclidean distance,⁵ then the probability that the input pattern belongs to quantization center i can be calculated as proportional to $e^{-d_i^2/\sigma^2}$. The parameter σ describes how likely we expect variations to be from the center: a higher σ value is appropriate when we expect higher noise levels in the system.

The outputs of the spatial pooler are fed to the temporal pooler. It is on these inputs that the temporal pooler performs learning and, eventually, inference. This corresponds to stage (C) in figure 7.

4.2 Operation of the temporal pooler

A particular input to the temporal pooler carries information about the relative strengths of the currently active quantization centers in the spatial pooler. The temporal pooler's operation can be divided into three stages. First, it learns the patterns of temporal transitions between the quantization centers of the spatial pooler. Then, based on these temporal transitions, it forms groups of quantization centers according to their temporal proximity. Once these two steps are completed,

⁴The spatial pooler can produce outputs even while it is learning the quantization centers. If the spatial pooler has learned n quantization centers at time t , it can produce outputs in terms of those n quantization centers, while simultaneously learning new quantization centers. This scheme is more efficient because the temporal pooler can start its learning process simultaneously with the spatial pooler. Indeed, this is the way spatial pooling is implemented in our Research Release

⁵Clearly, other assumptions can be made about how the probability falls off as a pattern moves away from a quantization center. A Gaussian assumption is a reasonable starting point.

the temporal pooler starts producing outputs in terms of its learned temporal groups. We describe these three steps - learning temporal transitions, forming temporal groups and producing outputs in the sections that follow.

4.2.1 Learning the temporal transitions of quantization centers

Let the input to the temporal pooler at time t be denoted as $y(t)$. As we mentioned before, this is a vector of length N_c and can be viewed as a probability distribution over the space of the quantization centers of the spatial pooler. Let $c(t)$ be the index of the maximum of $y(t)$, i.e., $c(t) = \arg \max y(t)$. In other words, $c(t)$ is the index of the quantization center that is most active at time t , and $c(t-1)$ can be thought of as the index of the quantization center that was maximally active at time $t-1$. In order to learn the patterns of temporal transitions of the quantization centers, the temporal pooler observes the $c(t)$'s as they occur through time. There are different ways to learn and represent temporal transitions. Again with the goal of simplicity, we assume a model which learns a first-order time-adjacency matrix between quantization centers.

Learning the time-adjacency matrix involves observing consecutive occurrences of quantization centers. During the beginning of this learning process, the temporal pooler initializes a matrix which has N_c rows and N_c columns to all zero values. (N_c is the number of quantization centers that the node has stored). The rows and columns of the matrix correspond to the quantization centers occurring at times $t-1$ and t respectively. The time-adjacency matrix within a temporal pooler will be denoted by T .

Let $x(t-1)$ and $x(t)$ be the inputs to the node (and spatial pooler) at time $t-1$ and t . The corresponding outputs of the spatial pooler (and hence the inputs to the temporal pooler) are $y(t-1)$ and $y(t)$. The temporal pooler calculates $c(t)$ and $c(t-1)$ as $c(t) = \arg \max y(t)$ and $c(t-1) = \arg \max y(t-1)$. At time t , the node updates its time-adjacency matrix by incrementing $T(c(t-1), c(t))$.

During this process, the temporal pooler periodically normalizes its time-adjacency matrix along its rows to obtain the T_{norm} matrix. The T_{norm} matrix is a true first-order transition matrix. The $(i, j)^{th}$ entry of this matrix gives the probability of observing c_j at time t , given it was c_i that was observed at time $t-1$. The node can stop this learning process when the T_{norm} matrix has sufficiently stabilized.

Figure 10(a) shows the transition matrix that was learned at the lowest level of the hierarchy in figure 4 when we exposed the network to movies of the kind we described in section 3. A section of this transition matrix is expanded in figure 10(b) to show the kinds of temporal relations that were learned from the movie.

4.2.2 Forming temporal groups by partitioning the time-adjacency matrix

After the time-adjacency matrix has been formed, it must be partitioned into groups. Unlike the process of learning the matrix itself, this process is done only once.⁶ The goal is to partition

⁶Strictly speaking, this is not a learning stage because the node is not observing its inputs during this stage.

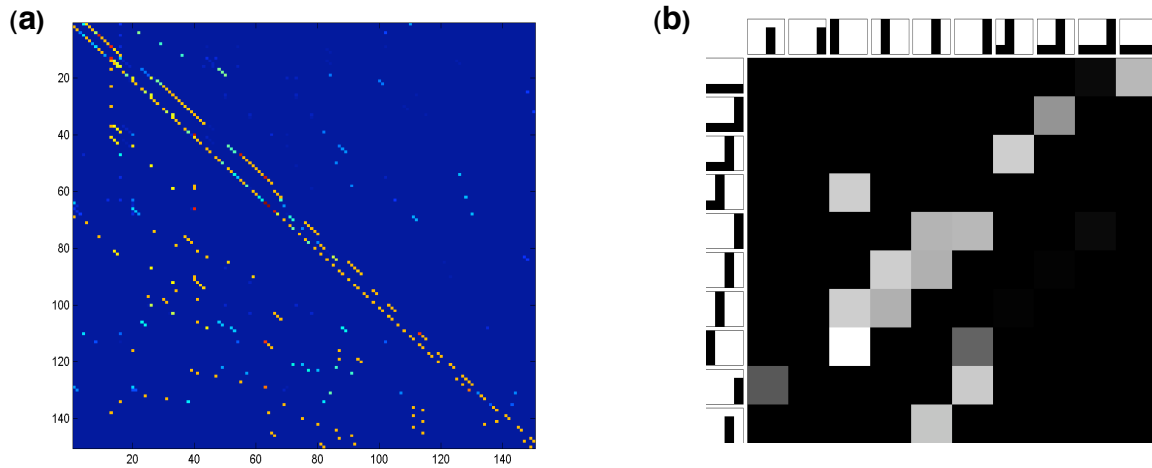


Figure 10: Temporal adjacency matrix. (a) shows the full time-adjacency matrix learned within the temporal pooler of a level-1 node in figure 4. This is a 150x150 matrix with rows corresponding to the quantization centers at time t , and columns corresponding to the quantization centers at $t + 1$. (b) zooms in on a portion of the time-adjacency matrix. In this figure, the associated quantization centers are shown along the rows and columns. The rows correspond to time t and the columns correspond to time $t + 1$. The values are indicated by a gray scale with white being the highest and black being the lowest. As an example, an examination of the 5th row shows that a vertical line at time t is very likely to move to the left, or remain at the same location at time $t - 1$. In contrast the third row shows that a corner is very likely to move to the left and not at all likely to stay in the same position.

the set of quantization centers into *temporally coherent* subgroups. This means that we want to form groups such that all the quantization centers within a group are highly likely to follow each other and all the quantization centers within different groups are unlikely to follow each other. Once again, the fundamental idea is that quantization centers which are highly likely to follow one another in time are likely to be linked to the same cause in the world.

There are several ways to formalize the idea of forming temporal groups from the time-adjacency matrix. One is to think of this as a graph partitioning problem, where the quantization centers are the vertices and the time-adjacency values are the edges; in this case, the goal is to find highly connected sub graphs of a graph. Another way is to think of this as a clustering problem where the similarity between two quantization centers is measured as the time-adjacency between those two quantization centers. Different methods can lead to different groups, but need not highly affect the overall performance. In this section we explore one approach to partition the transition matrix.

First, let us define a mathematical notion of what we would consider “good” groups. Let D_1, \dots, D_{N_g} , where D_i are disjoint subsets of the set of quantization centers, be a candidate partitioning of the quantization centers c_1, \dots, c_{N_c} . Then, we can define a time-adjacency measure t_i on D_i as

$$t_i = \frac{1}{n_i^2} \sum_{k \in D_i} \sum_{m \in D_i} T(k, m) \quad (1)$$

where n_i is the number of elements of the partition D_i and $T(k, m)$ denotes the $(k, m)^{th}$ entry of the time-adjacency matrix T . In the above equation, t_i can be thought of as a measure of the average temporal similarity of the quantization centers within the cluster D_i . If the spatial patterns within that cluster frequently occur adjacently in time, their corresponding values in the time-adjacency matrix would be high. This would lead to a high value for t_i .

Based on this per-partition similarity measure, we can define a global objective J where

$$J = (1/2) \sum_{i=1}^{N_g} n_i t_i \quad (2)$$

Ideally, we would want to find the set of partitions $\{D_i\}_{i=1 \dots n}^*$ that maximizes this objective function. However, given a set of quantization centers, the number of candidate partitions of that set is large, and it is not computationally feasible to solve this problem through an exhaustive search. Greedy, approximate methods, on the other hand, are computationally feasible, but can lead to local optima far inferior to the global optimum. Fortunately, the overall HTM architecture is robust to sub-optimal groupings, and even decent solutions in any node work quite well in the complete HTM network.

We now present one such greedy method, which is fast and extremely simple. The grouping algorithm repeats the following process until all quantization points belong to a group:

1. Find the most-connected quantization point that is not yet part of a group.
 - The most-connected quantization point is simply the quantization point whose corresponding row in the time-adjacency matrix has the greatest sum.

2. Pick the N_{top} most-connected quantization points to this quantization point.
 - N_{top} is a parameter specified in the algorithms by *topNeighbors*. The pooler adds these quantization points to the current group. Only quantization points that are not already part of a group are considered.
3. Repeat step 2 for each of the newly-added quantization points,
 - With well-behaved groups, this process will terminate automatically, because all of point X's closest N_{top} neighbors are also likely to have point X as one of their N_{top} closest neighbors.
 - If the group size does not automatically close off by the time the group reaches a certain size (*maxGroupSize*), the process is terminated and no new points are added.
4. The resulting set of quantization points is added as a new group. Return to step 1 until all quantization points have been grouped.

This process is illustrated in figures 11 - 13, with the time-adjacency matrix visualized as a graph. Thicker links represent higher values in the matrix. The role of the pooler is to segment this graph in order to group together quantization points that belong to the same cause. We assume that quantization points with stronger links are more likely to represent the same cause than those with weaker links or no links. In this example, *topNeighbors* (N_{top}) is set to 2.

After every quantization points belongs to a group, learning is complete. The time-adjacency matrix is retained for debugging purposes, but it is not used again.

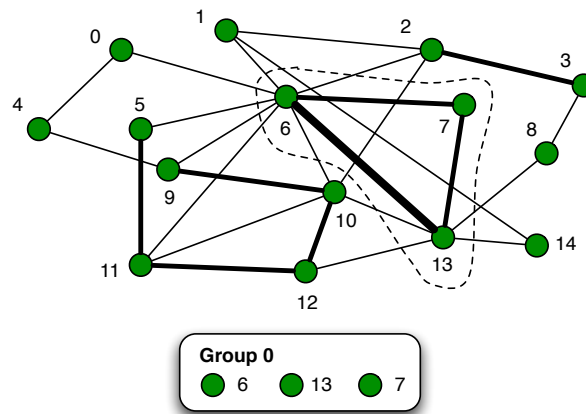


Figure 11: First iteration of temporal grouping algorithm: group formed using the most frequent quantization point, 6, as a seed. The pooler finds the 2 quantization points which are most strongly connected to point 6, point 13 and 7, and adds them to the group. The pooler then examines the top neighbors of point 13, which happen to be point 6 and 7, but they already have been pulled into the group. It does the same for point 7, and finds quantization centers 6 and 13. Since no unexamined neighbors are left, the pooler stops and group 0 is complete, with three points. These points are removed from the graph

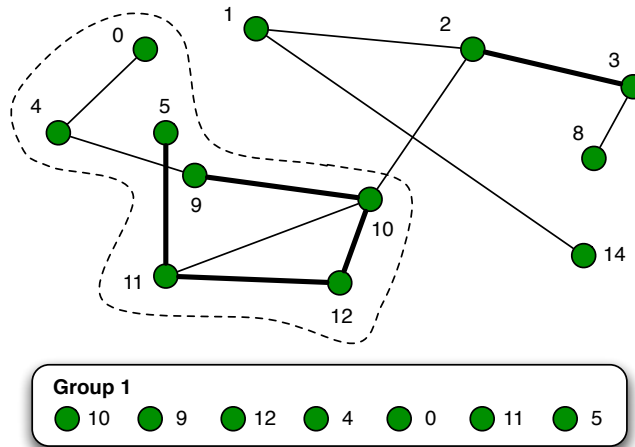


Figure 12: Second iteration of temporal grouping algorithm: group formed using the most frequent remaining quantization center, 10, as a seed. Point 10's top neighbors are points 9 and 12, which it adds to the group. Point 9's top neighbors are points 10 (already in the group) and 4. Point 4's top neighbors are 9 (already in the group) and 0. Point 0 has only one neighbor, which is already in the group. Point 12 brings a new top neighbor, 11, which brings a new top neighbor, 5, which does not bring any new neighbors. The group is complete, and these points are removed from the graph.

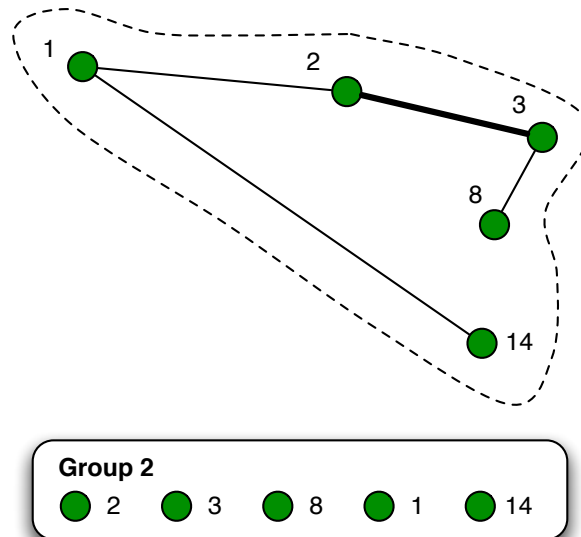


Figure 13: Third and final iteration of temporal grouping algorithm. It is easy to see that all remaining points are pulled into the next group, because no point has more than 2 links.

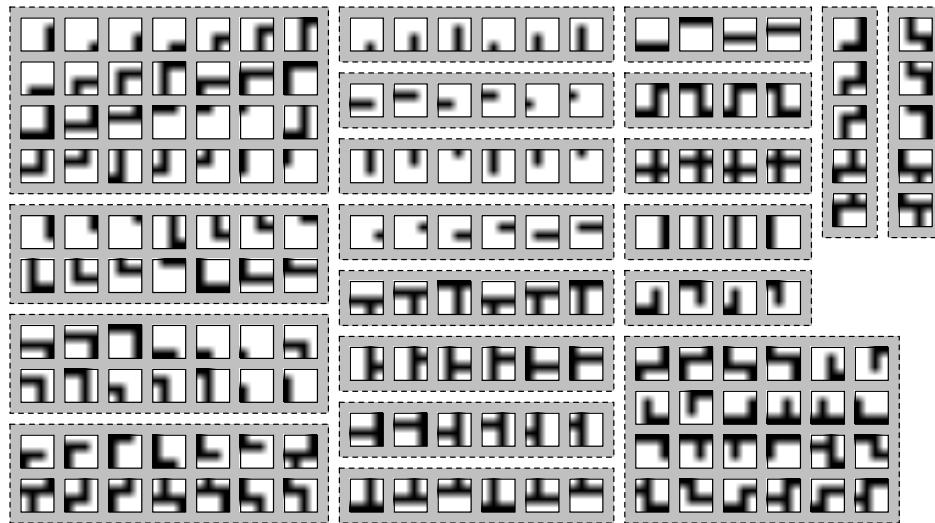


Figure 14: The groups produced by the grouping algorithm in a level 1 node in the Pictures World. The individual groups have a shaded background and a dotted-line boundary: the boxes have been manually arranged to show similar groups, although the order of the groups does not matter in any of the algorithms. The algorithm is successful in grouping together patterns that are perceptually similar. For this example, $topNeighbors = 3$ and $maxGroupSize$ is infinite.

Figure 14 shows the results when applying this algorithm to a level 1 node in the Pictures World. Spatial patterns which a human finds perceptually similar – “corners”, “lines”, “T’s” – have been grouped together.

4.2.3 Producing outputs from the temporal pooler: noiseless case

Once the time-adjacency matrix is used to partition the set of quantization centers into different temporal groups, the temporal pooler can start producing outputs. Let the number of temporal groups be N_g . Then, the output of the temporal pooler is a vector of size N_g . The output can be considered as a probability distribution over the space of the temporal groups. We describe how this probability distribution is calculated for the general case in section 7. In this section we consider the special case where the input to the temporal pooler is assumed to contain no uncertainty about the current quantization center. This case also corresponds to the noise-less inference case of the spatial pooler.

The output of the temporal grouper is easily explained for the noiseless case. The input patterns to the node match the quantization centers in the spatial pooler exactly. The spatial pooler finds the index of the currently active quantization center by taking the $argmax$ of its current input. The temporal pooler then finds the index of the temporal group to which this quantization index belongs. The output of the temporal pooler is all zeros except for one at the position corresponding to the index of the temporal group to which the quantization center belongs.

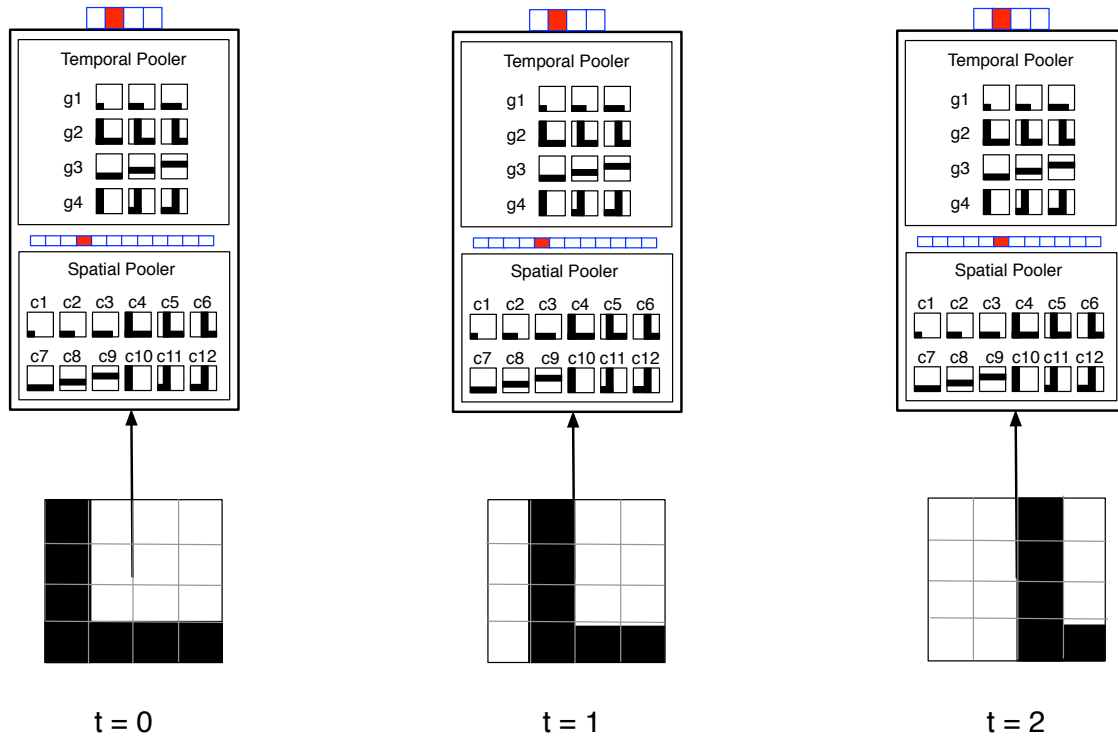


Figure 15: This figure shows a fully-learned node in inference mode for 3 time steps. The fully learned node has 12 quantization centers within its spatial pooler and 4 temporal groups within its temporal pooler. The quantization centers are shown as 4x4 pixel patches. The temporal groups are shown in terms of their quantization centers. The input to the node is a 4x4 pixel patch. The output of the spatial pooler is shown as a vector of length 12 with a filled square indicating a 1 and an empty square indicating a zero. The output of the temporal pooler (and hence the output of the node) is shown as a vector of length 4. See the text for more details.

4.3 Supervised pooling at the top level

We saw how the temporal pooler uses temporal proximity information to pool quantization centers into temporal groups. In cases where an external supervisory signal is available, that information can be used to pool quantization centers instead. This type of pooling is typically done at the highest level to associate known labels to the quantization centers of the top node. It replaces the temporal pooler in these nodes.

Suppose for every quantization center at the top node, we have a separate signal indicating the category to which this quantization center belongs. A simple supervised learning scheme associates the quantization center index with the category index. This can be done in a matrix form where the rows of the matrix correspond to the category index and the columns of the matrix correspond to the quantization centers. This matrix is initialized to zero at the beginning of the supervised learning process. For a particular iteration of the learning process, let the category index be i , and let the index corresponding to the quantization center that is activated at this node be j . The $(i, j)^{th}$ entry of the matrix is then incremented.

4.4 Structure of a fully learned node

Once the spatial pooler and the temporal pooler within a node have finished their learning processes the node can be considered to be fully-learned. A fully learned node stores the following information;

- A set of quantization centers numbered 1 to N_c in its spatial pooler.
- A time-adjacency matrix of size $N_c \times N_c$ storing the first-order temporal relations between the quantization centers. This matrix is stored in the temporal pooler.
- A set of temporal groups numbered 1 to N_g . Each temporal group can be thought of as a subset of the set of spatial patterns. These temporal groups are stored in the temporal pooler.

The different learning stages a node passed through before becoming fully learned are summarized in figure 7. A fully learned node is shown in figure 7(D).

Remember that we use the time-adjacency matrix to form the temporal groups. Although it is possible to use the time-adjacency matrix during the inference (recognition) process, in this paper we deal only with “flash” image recognition, i.e, recognition based on a single snap shot of the image. Therefore, once the temporal groups are formed, we can ignore the time-adjacency matrix when we discuss the inference case.

4.5 Fully learned node in inference

Once a node is fully learned, it can be switched to the inference (recognition) stage . From a node’s point of view, the inference consists of identifying the temporal group to which the input spatial pattern belongs. In this section we describe how this is done. We again restrict our attention to the noiseless case where the input pattern corresponds exactly to one of the stored quantization centers. This makes it easy to visualize what is happening. The general case is explained in section-7.

Figure 15 shows an example of a fully learned node in operation for three time steps. Assume that, through exposure to a series of patterns, the node has learned the structure shown in figure 15. The input to this node is a pattern of length 16 which corresponds to a 4x4 image patch. The node has learned 12 quantization centers. These quantization centers are shown as 4x4 patches within the spatial pooler of the node. Since there are 12 quantization centers, the output of the spatial pooler is of length 12. This node also has 4 temporal groups. Each temporal group has 3 quantization centers each. The temporal group g_1 represents a line growing to the right and the temporal group g_2 represents a corner moving to the right. The output of the node is of size 4 and is equal to the number of temporal groups in the temporal pooler.

In figure 15, the input pattern at time $t = 0$ matches the quantization center c_4 stored within the spatial pooler. Therefore, the output of the spatial pooler at $t = 0$ has a 1 in the fourth location and zero everywhere else. (In the figure, the 1 is indicated by a filled rectangle and zero by an empty rectangle). The fourth quantization center c_4 belongs to the temporal group g_2 . Therefore the output of the temporal pooler at $t = 0$ has a 1 in the location corresponding to g_2 .

At time $t = 1$, the node gets different input pattern - a shifted corner. This input matches quantization center c_5 and the spatial pooler output reflects this fact. The quantization center c_5 belongs to the same temporal group (g_2) as before. Therefore, the output of the temporal pooler did not change. Due to the nature of the temporal groups that were learned, the three different input patterns (three different shifts of a corner) generated the same output temporal group. This is because group g_2 is an invariant representation for various shifts of a corner. In section 6 we examine the nature of invariant representations at various levels of the hierarchy.

5 Operation of Nodes in a Hierarchy

We follow a level-by-level strategy to train all the nodes in our HTM network. First, only the nodes at the bottom level of the hierarchy are trained. During this time, the nodes at the higher levels are disabled - they do nothing. When the nodes at the bottom level are in their learning phase, they do not send any outputs to the higher levels.

Once all the nodes at level 1 are finished with their learning process the nodes at level 2 can start learning: the nodes at the bottom level are switched from learning mode to inference mode, and the nodes at the second level are enabled and put into learning mode. This process is then repeated in the hierarchy until all nodes in the network are fully trained.

We use the simplified two level hierarchical network shown in figure 16 to illustrate the operation of nodes in a hierarchy. This figure shows a parent node at level 2 with 2 children at level 1. Each bottom-level node is fully learned and in inference stage. The structure of each bottom-level node is the same as the one we described in the previous section, with 12 quantization centers and 4 temporal groups each. (Both child nodes have identical numbering of quantization centers and temporal groups, although this need not be the case). Also shown is how these level 1 nodes receive their inputs from a section of the input image. This section of the input image corresponds to 2 adjacent non-overlapping patches of 4x4 pixels. The parent node at level 2 hasn't started its learning process and hence it is in the initial stage.

5.1 Inputs for nodes higher in the hierarchy

When a node is at the bottom level of the hierarchy, its input comes directly from the sensor (possibly after some pre-processing). For higher-level nodes, the inputs are the outputs from their child nodes. For any given parent node, its input is a concatenation of the outputs of its children.

In figure 16, each level 1 node has an output of length 4. The input to the level 2 node is a concatenation of these outputs. Therefore the input to the level 2 node is a pattern of length 8 as shown in figures 16 and 17. This input goes to the spatial pooler of the level 2 node.

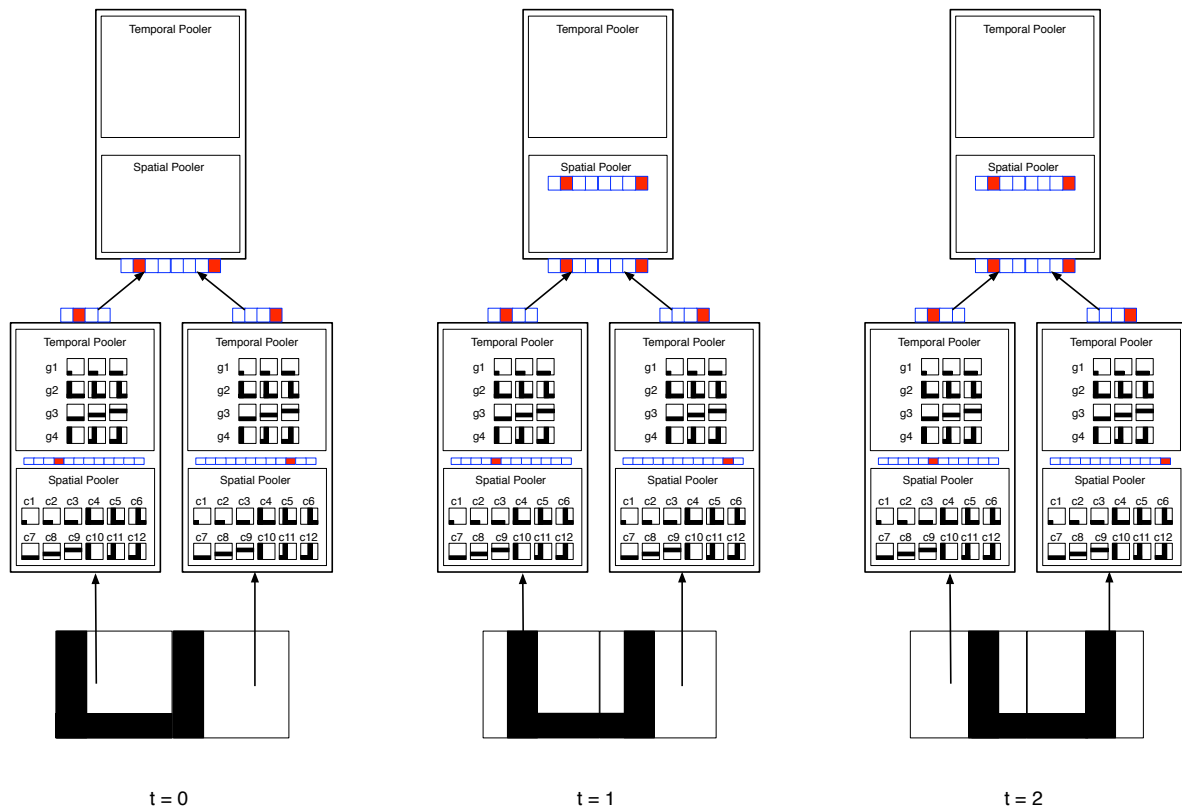


Figure 16: Operation of nodes in a hierarchy: This figure illustrates how nodes operate in a hierarchy. The bottom-level nodes have finished learning and are in inference mode. The input to these nodes is a 4x4 input image patch i.e., a pattern of length 16. The sequence of inputs from time $t = 0$ to time $t = 2$ correspond to a U pattern moving to the right. The corresponding spatial pooler and node outputs are shown. These level-1 nodes have learned 12 quantization centers each. The quantization centers are shown within their spatial poolers as 4x4 pixel patches. Each bottom-level node has 4 temporal groups. Each temporal group has 3 quantization centers. Temporal group g_2 , for example, represents a left corner moving to the right. The outputs of the spatial pooler is shown as a vector of length 12. The output of the temporal pooler (and the node) is shown as a vector of length 4. The top-level node has just begun its learning process. The input to this node is represented as a vector of length 8. This input is obtained by concatenating the outputs of the bottom-level nodes. The input pattern to this network segment at time $t = 0$ is a U pattern. The left half of this U pattern (left-corner) is the input to the level-1 node at the left. This input pattern matches quantization center c_4 within the spatial pooler of this node. This is reflected in the output of the spatial pooler, with a one in position 4 and zeros elsewhere. The quantization center c_4 belongs to the temporal group g_2 in this node. Therefore the output of the node has a 1 in position 2 and zeros elsewhere.

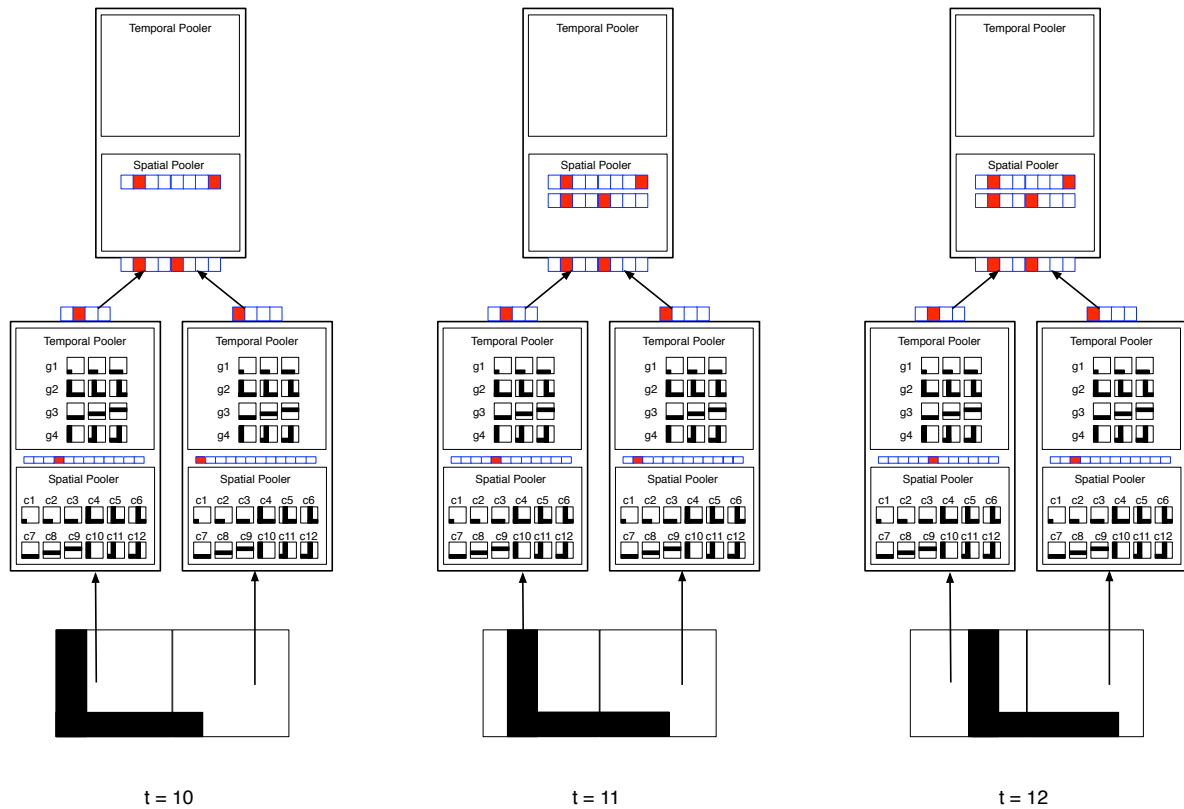


Figure 17: This figure uses the same setting as of figure 16 except for the input sequence. The input sequence is an “L” moving to the right. The second level node’s spatial pooler is in the learning state. It has already learned one quantization center before the beginning of this input sequence. The new input sequence introduced one additional quantization center to the second level node.

5.2 Learning at a Level 2 node

As described earlier, while the level 2 node is in learning mode, all its children are in the inference mode. Consider now a sequence of inputs as shown in figure 16. This figure shows the network and the input for times $t = 0$ to times $t = 10$. The sequence of inputs correspond to a “U” shape moving in the combined visual field of these two nodes.

Note that none of the lower-level nodes see the whole “U” pattern. The sequence of patterns seen by the left-bottom node, as the U moves in the combined field of view can be described as left-corner, left-corner, left-corner. This input sequence is the same as the case we examined in section 4.5. The right-bottom node, on the other hand, sees the sequence vertical-line, right-corner, right-corner as the whole “U” pattern moves in the visual field. These patterns cause internal spatial pooler outputs at each of these time instants. These outputs are of length 12 and correspond to the quantization centers closest to the input patterns. For the right-bottom node, the quantization centers that are closest to the input patterns at $t = 0, t = 1$ and $t = 2$ are c_{10}, c_{11} and c_{12} respectively. The corresponding outputs of the spatial pooler in this node are $[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0]$, $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]$, $[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]$ respectively. The outputs for this node times $t = 0$ to time $t = 2$ are $[0, 0, 0, 1]$, $[0, 0, 0, 1]$, $[0, 0, 0, 1]$ respectively. i.e, the outputs do not change during these time steps. The outputs of the other lower-level node are also shown in the figures.

The input to the level 2 node at any time is a concatenation of the patterns from its children. For the 3 time steps of “U” moving to the right as shown in figure 16, this input is the same and is equal to $[0, 1, 0, 0, 0, 0, 0, 1]$ and is obtained by concatenating the output $[0, 1, 0, 0]$ of the left child node with the output $[0, 0, 0, 1]$ of the right child node. During these three time steps, the network was exposed to 3 different images of “U” at the bottom level. However, the input to the level 2 node remained the same for these three time steps. Thus the input to the level 2 node can be thought of as an invariant representation for three different “U” images which in this case corresponds to three different shifts of a “U”.

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

Figure 17 shows another sequence of patterns and the corresponding outputs of the bottom-level nodes and inputs of the level 2 node. Lets assume that this corresponds to the inputs to the node from $t = 10$ to $t = 12$. (The choice of time points is arbitrary). During this period an “L” moves across the combined visual field of the bottom nodes. Note however, that the bottom-left node sees the same sequence of movement as it saw for the rightward motion of the “U” pattern. That is, its output for the times $t = 10$ to $t = 12$ is identical to that of time steps $t = 0$ to $t = 2$. The bottom-right node sees a horizontal line that is growing towards the right and this belongs to temporal group ‘1’ within that node. Therefore, the output of the bottom-right node from $t = 10$ to $t = 12$ is $[1, 0, 0, 0]$. The concatenation of these two outputs is the input to the level 2 node.

As we observed in the case of the “U” pattern, the different shift of “L” at the bottom level produces the same spatial pattern input to the level 2 node. This means that the input to the level 2 node is invariant to those translations of “L”. Note however that the spatial pattern input to the level 2 node for the “L” sequence is different from that for the “U” sequence. Thus, the spatial patterns at level 2 have maintained the selectivity between “U” and “L” while forming invariant representations for shifts of “U” and shifts of “L”. In the next section we examine how this system learns invariance to some transformations while retaining selectivity.

5.3 Quantization centers represent “coincidences”

We saw that a quantization center of a higher-level node is a concatenation of outputs from its children. An output from a child has a ‘1’ in the position corresponding to its currently active group. Therefore, each quantization center has as many ones as there are children. A quantization center represents a particular co-occurrence or coincidence of the temporal groups of its children. Consider the case of the quantization center [0100 0001] in the example we considered above. This quantization center describes that the second temporal group of the left-child and the fourth temporal group of the right child occurred at the same time.

Since a quantization center in a node is essentially a co-occurrence or coincidence of temporal groups in that node’s child nodes, we use “coincidence” and quantization center interchangeably for the rest of this paper.

6 Representations at various levels of the hierarchy

In the previous section, we observed that the output of the level 1 node remained stable for several time steps while the input changed. Note that, for three different appearances of the “left-corner” (different shifted versions), the node produced the same output - the output corresponding to g_2 , the left corner group. This is not accidental. This happened because the node, by virtue of its learning, had grouped together different spatial patterns to form temporal groups. These temporal groups are invariant representations for the portion of space to which that the node is exposed. If we examine the temporal groups of the bottom-level node, we can see that g_1 represents a “growing line”, that g_2 represents a “left corner”, that g_3 represents a “horizontal line” and g_4 represents a “right corner”.

For three positions of the U, the left-bottom node output represented the “left-corner” group g_2 and the right-bottom node output represented a “right-corner” group g_4 . These groups are invariant representations for different shifts of a corner. The quantization pattern that the level 2 node learned is a combination of the invariant representation of left-corner (g_2) from the node on the left and the invariant representation of right-corner (g_4) from the right node. This quantization center (“coincidence”) can be thought of as representing U invariant to the three positions. This was obtained by representing the simultaneous occurrence of the invariant representation left-corner from the left child node and the invariant representation right-corner from the right child node.

The coincidence that was learned at level 2 for the “L” pattern consisted of representing the simultaneous occurrence of the left-corner group from the left-child node and the growing-line group from the right-child node. This is a different coincidence of the invariant representations of the child nodes. U and L are represented at the level 2 node as different configurations of the invariant representations of its child node. High-level object parts are represented using different configurations of the invariant representations of its parts.

In general the following observations can be made about the two learning mechanisms at every higher-level node. By pooling multiple quantization centers into one group, the temporal grouping scheme produces invariant representations. The spatial pooler learning process learns coincidences of the invariant representations of its child nodes. By learning particular coincidences of these invariant representations, the spatial-pattern learning process learns “selectivity”. The strategy used by this system is to alternately learn invariances and selectivity in a hierarchy to enable the hierarchy to recognize various objects despite their transformations.

A coincidence at a higher-level node represents things that occur at the same time in the visual world, whereas temporal groups represent things that happen one after another.

7 Inference - General Case

In an earlier section we described the inference process in the case where the input patterns of a node are perfect replicas of the quantization centers it has learned during the training process. In reality, the input patterns are always associated with noise. The general case of node inference involves dealing with noisy input patterns.

7.1 Spatial pooler inference - general case

We look at inference with noisy patterns as it happens in the second-level node. We saw earlier that the input to the second level node is a concatenation of the outputs of the child nodes. The child nodes’ outputs carry information about the temporal groups they are in. Each child node’s output length is equal to the number of temporal groups in that child node. In the noiseless case, these child nodes were certain about what group they were in and their outputs had a ‘1’ in the place of the temporal group that they were currently in and ‘0’ everywhere else. In the noisy case, the child nodes output will have an uncertainty about the group they are in and the output of a child node will be a probability distribution over its groups.

We can understand this process by looking at a particular quantization center in the level 2 node. Consider the quantization center shown in figure 16 within the level 2 node. This node has two child nodes and therefore every spatial pattern has two ‘1’s in it, each ‘1’ representing a particular temporal group from a child. In this particular case, the ‘1’s happen to be for group 2 from the left child node and group 4 from the right child node. Now, if the input from the children nodes are probability distributions, the probability that this particular coincidence is active is roughly proportional to the quantity obtained by multiplying together the second value from the left child nodes output and the fourth value from the right child nodes output.

In general, let the i^{th} quantization center (or coincidence) be represented as $c_i = [m_1^i, m_2^i, \dots, m_{N_c}^i]$ where the m_k^i 's represent the positions where the spatial patterns have a non-zero and N_c is the number child nodes of this node. Then, the probability that the i^{th} quantization center (or coincidence) is active can be calculated as

$$P(c_i) = \gamma \prod_{k=1}^{N_c} input(m_k^i) \quad (3)$$

where γ is a proportionality constant. The probability distribution over the space of quantization centers can be calculated by calculating $P(c_i)$ for all i and the normalizing that vector.

7.2 Temporal pooler inference - general case

Now, the node has to calculate its output using the probability distribution over its quantization centers calculated as above. In the noiseless case, we calculated this simply as the index of the group to which the current quantization center belongs. But now, with noisy inputs, we have a probability distribution over all quantization centers. From this, the probability that a particular group is active can be calculated as the probability of that spatial pattern that has the maximum probability among all the spatial patterns belonging to that group. Doing this for every group, we obtain a distribution that is over the space groups. This is then set as the output of the node.

The inference process for a node that is trained using supervision is similar to the case discussed above. During the supervised learning process, we develop a description of a category in terms of the spatial patterns received in that node. This is similar to the temporal groups case where a temporal group consists of several spatial patterns. Similarly a category consists of several spatial patterns and the inference process is similar to that of other nodes.

This form of propagating the uncertainties provides better recognition compared to making winner-take-all decisions locally. The ambiguous information can be resolved higher up in the hierarchy because only some configurations of temporal groups of lower levels are allowed. The mathematics of propagating such uncertainties to obtain the optimum recognition performance is called Bayesian Belief Propagation. We formally describe the Belief Propagation mechanism used in HTM nodes in appendix A. Belief Propagation techniques also allow us to propagate information downstream in the hierarchy. This can be used to disambiguate information at lower levels.

8 Performance of a Fully Trained Hierarchy

We trained an HTM network using the procedures described above. Unlike the hierarchy in figure 4, this network had 4 levels: 64 nodes in the bottom level, 16 nodes in the second level, 4 nodes in the third level, and one node on top. Each node received inputs from 2-by-2 patch of nodes, and there was no overlap in the bottom-level nodes' receptive fields.

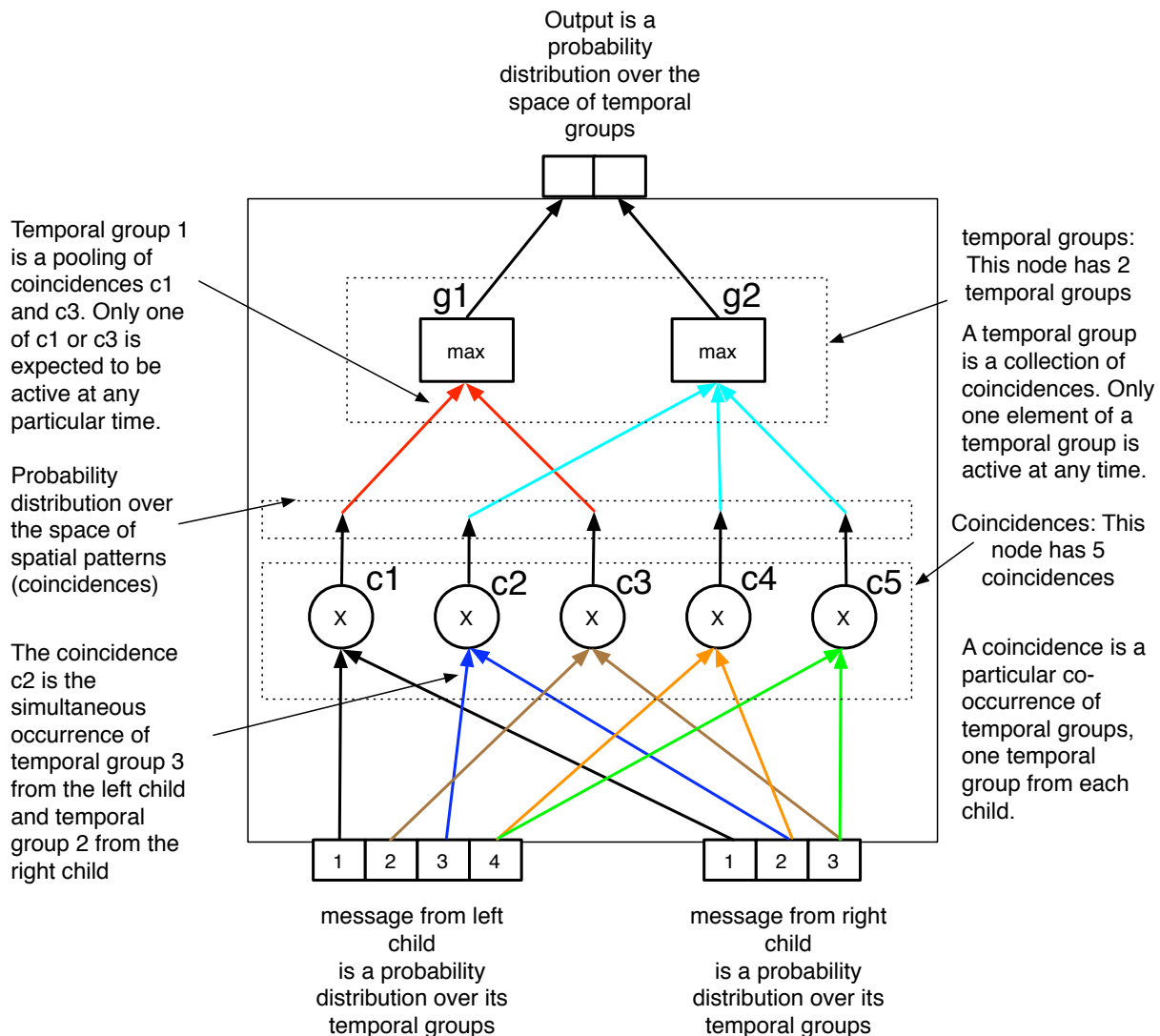


Figure 18: This figure illustrates the structure of a node and the general inference mechanism. Each quantization center (or coincidence) can be thought of as being represented by a neuron-like unit. The connections of this neuron to the input describes the quantization center or coincidence. For example, c_2 is has connections to the 3rd location of the message from the left child and to the 2nd location of the right child. This means that c_2 represents the coincidence of group 3 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 quantization center) 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 outputs of a temporal group neuron is obtained by taking the maximum over the inputs.

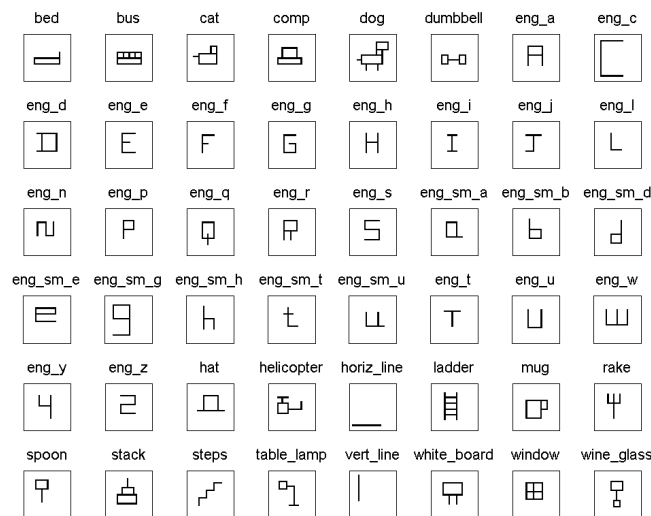


Figure 19: Training image category examples: An example from each category is shown. Several examples at different scales were used to generate the training movies.

8.1 Training movies

We trained the system on 48 categories of line drawing images as shown in figure 19. We had several scaled examples of each category as part of the training set.

Level 1 and level 2 of the system were trained using videos of the training images. These videos consisted of vertical and horizontal translations. Since the examples themselves had several scaled versions of each image, we did not include the scaling motions as part of the videos. Moreover, in the line drawing case, the basic temporal groups at lower levels of the hierarchy for representing scale changes can be learned from translation movies. This is because the nodes at the lowest levels of the hierarchy cannot distinguish (in the line drawing example) between a scaling motion and a translation motion.

The top level was trained using supervision. During this process, we presented the training images at every position and supervised at the top. Every exemplar of a training category was presented at every position.

8.2 Testing images

Testing images are generated using a program which takes a training image and generates several test images by arbitrarily scaling along x and y axes, adding noise, deleting pixels and adding small rotations. Since the scales chosen are arbitrary, it is highly unlikely that an exact testing image would have been presented during the training. A non-negligible number, perhaps 10 percent, are so distorted as to be unrecognizable by a human.

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

8.3 Recognition results

We tested the system on approximately 9000 test images. The system achieved 66 percent recognition accuracy on the test image set. The chance level is 2 percent. The recognition accuracy on training images was 99.73 percent. The recognition results on the test set are impressive because the test images generated by our program were highly distorted versions of the training images. Figure 20 shows several test images that were recognized correctly. These test images are arranged in columns according to their correct category. Figure 21 shows examples of images that were misclassified along with images of the category they were recognized as. Many of these cases show that the network made a reasonably good guess. In many of the incorrectly recognized cases, the second or third choice given by the network corresponded to the correct category.

We observed earlier that the perception of an object should remain stable despite relative movement between the observer and the object as long as the object remains within the field of view. This means that an organism can get additional information about the input by making small eye movements. If the input is ambiguous, the brain can gather further information from the input by making small eye movements. Between these eye movements, the correct hypothesis would remain stable while the competing incorrect hypotheses will vary in a random manner. We tested this idea on our learned network. The recognition accuracy improved considerably with more eye-movements. With nine eye movements we saw roughly ten percent improvement in recognition accuracy.

8.4 How the system generalizes to translations, scaling and distortion

We described how every node in the hierarchy is learning and representing object components and their motion patterns. An object is learned as a particular configuration of the parts represented at lower levels. Since the lower levels have also learned the motion patterns of these parts, once a particular configuration of these parts is learned by exposure to a particular image, the system can generalize to several variations of that image that occur through translation, scaling, rotation etc.

Suppose we take a trained network and try to teach it a new image, an image of the letter 'A', as shown in figure 22. Assume that the network was never exposed to this pattern before. However the network was exposed to the movements of various other patterns from our training set. Now, assume that we expose the network to this new 'A' pattern at one position and supervise at the top of the network. The one position of 'A' to which the network is exposed is shown in figure 22 in black.

Figure 22 (a) also shows three horizontal translations of this 'A' in colors gray, red and green. Notice all the spatial patterns that are formed within various nodes due to translations of the A.

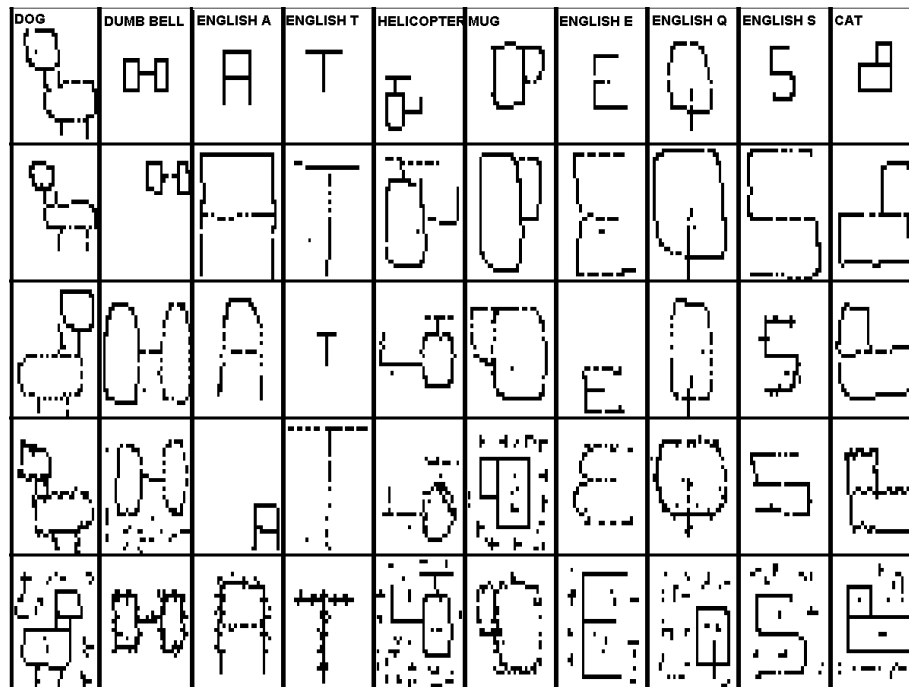


Figure 20: This figure shows examples of test images that were recognized correctly. Each column shows examples of a particular category. Only 10 out of 48 categories are shown here. It can be seen from the test images that the system can recognize these images despite large amounts of translation, scale differences, distortions and noise.

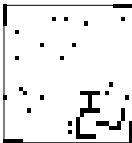
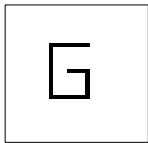
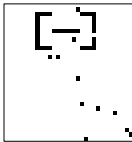
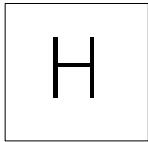
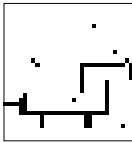
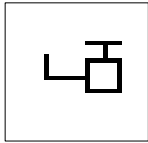
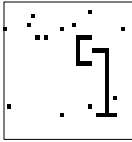
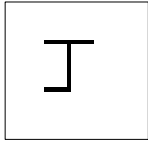
	Distorted Pictures	Recognized as
helicopter		
dumbbell		
dog		
lamp		

Figure 21: Each row of this picture corresponds to different recognition experiments. Consider the first row. The text label on the left identifies the actual category, helicopter, of the distorted picture that is shown next to the label. These distorted pictures were generated automatically using a program. When the distorted helicopter picture was shown to the network for recognition, it incorrectly recognized it as the category ‘G’, shown on the right.

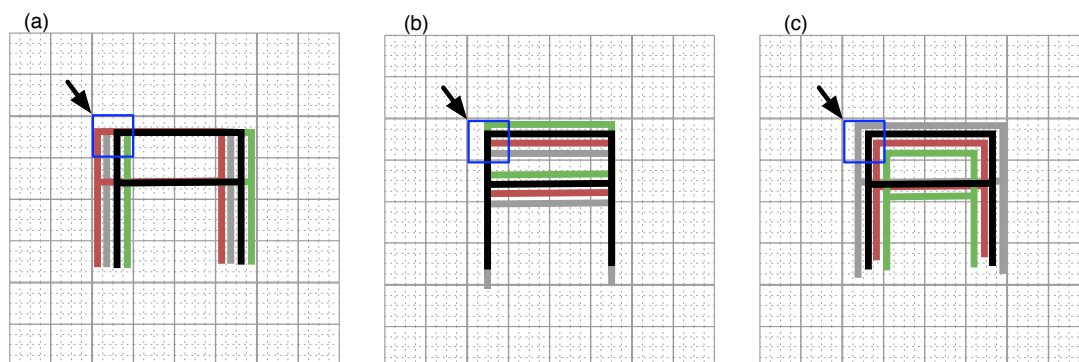


Figure 22: This figure is used to explain the generalization performance of our system. See the text for details. (a) shows several horizontal translations of an “A” (b) shows several vertical translations of an “A” and (c) shows several scaled versions of an “A”

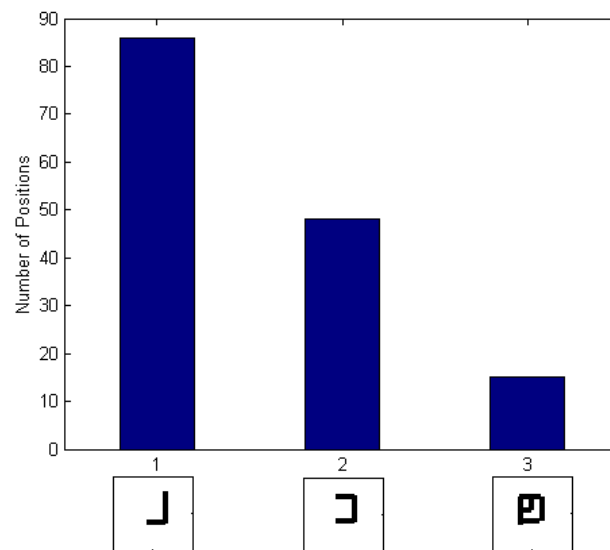


Figure 23: In this experiment we showed the system snapshots of 3 novel images at 100 randomly chosen positions. What is plotted is the number of positions to which the system correctly generalized for each of these novel images. It can be seen that the generalization performance goes down with increasing complexity of the novel pattern.

In the highlighted square patch in figure 22 (a), this corresponds to 4 translations of a corner. Assume that these four translations of the corner belonged to the same temporal group of the node that is attached to this patch. Then, for these translations of 'A', this node would be sending the same group index to the node above it. If this happens in all other nodes, the patterns seen by higher-level nodes will be *invariant* to these local translations of 'A'. The requirement is that all the spatial patterns within a node created by these translations lie within the same temporal group of that node. The temporal groups required for this are shown in the figure. These groups are very similar to the kinds of temporal groups that we formed in section 4.2 by exposing a node to movements of patterns. Figure 22 (b) shows some vertical translations and the associated temporal groups required at level 1 nodes to obtain invariance.

Figure 22 (c) shows the canonical 'A' and several scaled versions of it. It can be seen that the temporal groups required at the lowest level are similar to the ones required for translation invariance. This is because, in a local patch both those transformations could produce the same effect. Therefore, some of the temporal groups required to give us scaling invariance at the lowest level could also be learned just by exposure to translating patterns.

We saw how the temporal groups learned at level 1 gave us invariance to a set of local transformations. Invariance across larger transformations is given by the combined effect of the temporal groups at the lower levels and higher levels. When we learn a new object that is composed of parts of other objects that we have seen, the invariances that we learned for those other objects automatically transfer to these new objects. This occurs only because we learned the objects in a hierarchy. New objects are learned as a reconfiguration of parts of other objects.

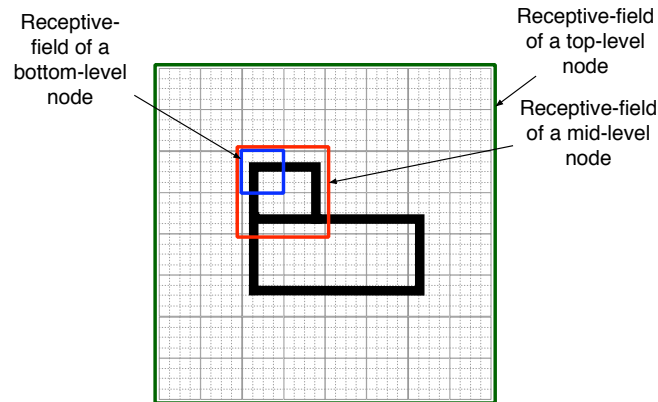


Figure 24: Image of a ‘cat’ as presented to the network. The receptive fields of three different nodes at three different levels are identified in the picture. See section 9 for details.

8.5 Generalization performance

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

In this experiment, we trained the network on 48 categories as described before. Then we exposed the network to three additional categories of images at 100 randomly chosen positions for each category while supervising at the top for the appropriate category. We tested the network for its generalization performance by querying the network for exposures to these images at different locations. The degree of success of the network in recognizing these images at other locations depended on the complexity of the novel image. Simple images generalized to more positions compared to complex images. Figure 23 shows the generalization performance of the network.

9 The Bigger Picture - A World Model

In this section, we will step back a bit to understand the general characteristics of the Pictures world. We will then make connections from the simple Pictures world to problems in the real world.

Consider the picture of a ‘cat’ in figure 24. Assume that this image is presented to our network for recognition. In the figure, we have identified the receptive fields of three nodes at three different levels. The receptive field of a top-level (level-3) node is the whole picture. The receptive field of a mid-level node is an intermediate size patch of size 8x8 pixels and the receptive field of a bottom-level node is a small 4x4 pixel patch. When this image is presented to the network, the top-level node will have a high activation for the ‘cat’ state, the mid-level node will have a high activation for a state that corresponds to ‘cat-head’ and the lowest level node will have a high activation for ‘corner’. To simplify things, let us say that the top-level, mid-level, and bottom-level nodes believe respectively in ‘cat’, ‘cat-head’ and ‘corner’ at this point.

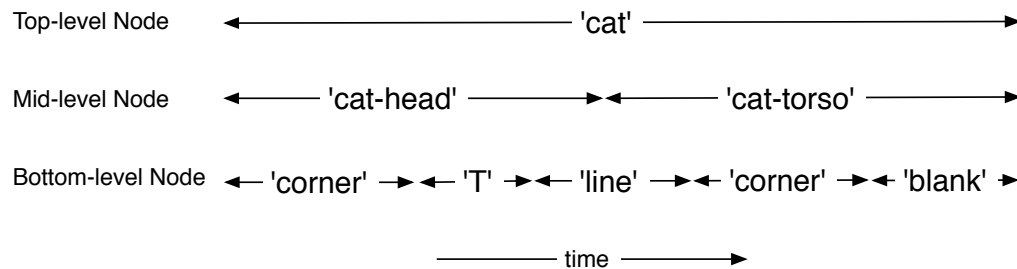


Figure 25: This figure shows the concept of temporal hierarchy in vision. The arrows show durations for which the nodes at different levels believe in a state. Higher levels change their states slowly compared to the lower levels.

Now let us imagine what will happen if the cat starts moving upward. The top-level node will still have the belief that it is looking at the 'cat'. For small displacements, the bottom-level node will believe that it is looking at a 'corner', but soon, it will change its belief to a rotated 'T-junction' and soon after that it will change its belief to a 'vertical-line' and then to a 'blank'. All the while, the top-level node will still believe that it is looking at the 'cat'. Something in-between will happen in the mid-level node. For a duration – longer than the duration for which the bottom-level node believes 'corner', but shorter than the duration for which the top-level node believes 'cat' – its belief state will be 'cat-head'. Later, probably by the time the bottom-level node switches its belief to 'vertical-line', the mid-level node will switch its belief to 'cat-torso'.

What we observed here is that for natural movement patterns, i.e, if the cat is not randomly jumping from one position to another, the nodes at the lower level change their states faster compared to the nodes at the higher level. This can be visualized as shown in figure 25. We also know that the nodes at the higher level encompass more space compared to lower levels. 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 nodes at lower levels. A state that is represented at a higher level is influenced by a larger spatial extent, and is persistent for a longer duration of time compared to the lower level states. It can be hard to picture this inter-connection between the spatial organization and temporal organization. We have made an attempt at that in figure 26.

Instead of thinking of this as a recognition hierarchy, we could 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. So 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.

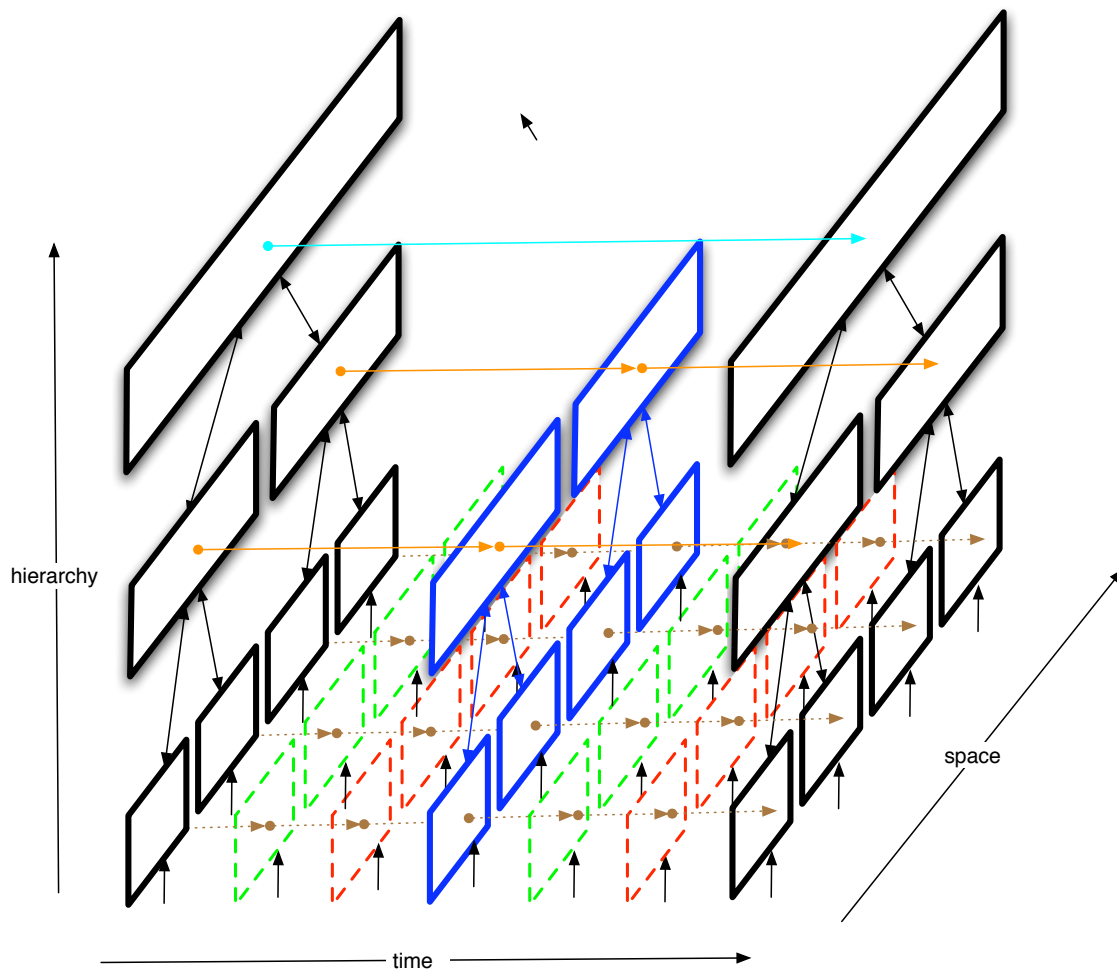


Figure 26: This figure shows the concept of a spatial and temporal abstraction hierarchy. The higher levels abstract larger areas of space and larger durations of time. Thus the higher levels change their states slowly compared to the lower levels.

Although we introduced this model with invariant visual pattern recognition as an example, this kind of multi-scale organization is found to be prevalent in nature. Weather makes a good example. Winter is a high-level state that affects the whole country. During winter, it snows heavily in New York, but it only decreases the number of sunny days in San Francisco. The high level cause, winter, produces different sub-states in different places. Also, the dynamics of snow-storms and sunny days wax and wane in durations measured in days, whereas winter weathers on for several months.

If we assume that the world is organized as an interleaved spatio-temporal hierarchical structure as we saw here, then learning a model for the world is easy if we know the structure of the hierarchy. Thanks to the temporal persistence of higher-level causes compared to lower level causes, this kind of organization allows us to use time as the teacher at each node to form distinct groups. Therefore, by alternating between temporal stability based supervision and coincidence detection, we can construct the world model back. This is one of the fundamental insights behind HTMs.

10 Conclusion

We have described the HTM learning algorithms in the context of a simple visual example. We hope that you have gained insight into the following concepts:

- The challenge of invariant object recognition
- Our key weapons in this challenge:
 - *Time as a supervisor*: Using proximity in time to group diverse spatial patterns
 - *Hierarchical networks*: Developing increasingly complex concepts from simpler, building-block concepts
- How these concepts are translated into node algorithms:
 - *Spatial pooler* reduces the potentially infinite space into a finite set of points
 - *Temporal pooler* applies the time-as-supervisor concept to group these points into temporally-adjacent sets
- How the node algorithms interface with each other in a hierarchy
- The source of generalization in the system

To learn more about HTMs and the Numenta Platform for Intelligent Computing, please visit www.numenta.com.

Appendix

A Belief Propagation in HTM Networks

In this section we describe Belief Propagation (BP) as implemented in HTMs. HTM networks can be thought of as encoding relationship between random variables.⁷ In our vision example, these random variables correspond to the invariant representations learned at multiple levels of the network. The whole network is a model of the visual world.

An input to the network is typically termed *evidence*. For example, if we present a new image to the network for recognition, this image 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 can be thought of as the degree of belief each node assigns to its coincidences and groups.

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 on the groups and coincidences in the nodes of the network reflect this evidence. When a new input image is presented, this information is then passed up the hierarchy using Belief Propagation and the belief states of the nodes in the network change to match the new evidence.

BP can be thought of as a mechanism for calculating the effect of an evidence on the belief states of the network using only local computations. 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. The sum-prop algorithm is usually what people mean when they talk about Belief Propagation algorithm.

Another variant of the BP 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. In Pictures, an assignment of states to the nodes of the network will involve, for example, assigning the top node to the category “dog” while simultaneously assigning a bottom-level node to the group corresponding to “dog leg”. The answer that Belief Revision computes is known as the *Most Probable Explanation* for the current evidence.

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. Most of this notation is standard and will be familiar to anybody with basic training in probability theory.

⁷Bayesian Belief Propagation was pioneered by Judea Pearl. More details on this can be obtained from his 1988 book *Probabilistic Reasoning in Intelligent Systems*

A.1 Notation

\mathcal{C}^k	The set of coincidences of the k^{th} node. When it is clear from context, we will drop the index for the node and use \mathcal{C} to represent the set of coincidences of the node under consideration.
C^k	Random variable representing the coincidences of the k^{th} node.
c_i^k	The i^{th} coincidence in the k^{th} node.
\mathcal{G}^k	The set of groups of the k^{th} node.
G^k	The random variable representing the temporal groups of the k^{th} node.
g_i^k	The i^{th} group in the k^{th} node.
e^-	The evidence from below, from a node's standpoint. That is, it is all the evidence observed by all the descendents of that node. In general BP, evidence can be propagated in both directions. In our current release we have only upward propagation. This means that all the nodes will have only e^- and no e^+
$P(e^- G^k)$	Probability of evidence from below given the 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 coincidences in node k . This is a vector of length equal to the number of coincidences in node k .
$P(e^- c_i^k)$	The probability of evidence from below given the i^{th} coincidence in node k . This is a scalar.
$P(\mathcal{C} \mathcal{G})$	The rows of this matrix correspond to the 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 group g_i .
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 $P(e^- g_r^k)$
y^k	The message that the spatial pooler of the k^{th} node sends to the temporal pooler of that node. This is a vector of length equal to the number of coincidences and is proportional to $P(e^- C^k)$

A.2 Learned memory of a node

Coincidences and groups form the basic memory of the node on which belief propagation operates.

The set of coincidences \mathcal{C} is stored within the spatial pooler of the node. A particular coincidence c_i defines the co-occurrence of temporal groups of its child node. It can be thought of as a vector $[r_{m_1}, \dots, r_{m_M}]$, where the r 's correspond to the indices of the groups of its child nodes. For

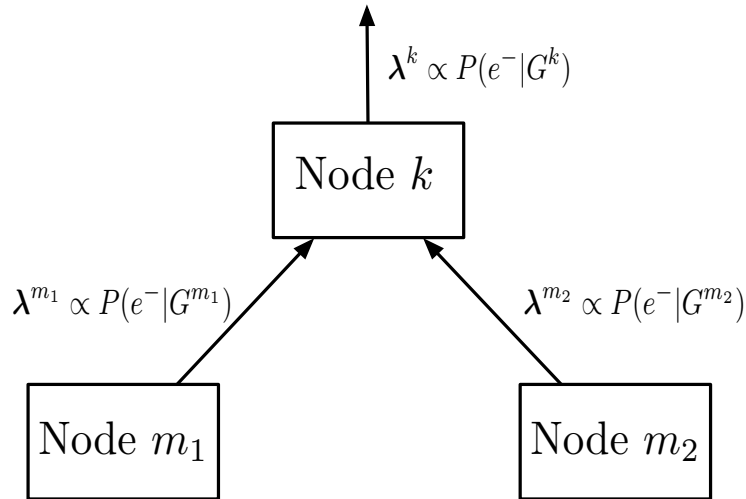


Figure 27: A segment of an HTM network.

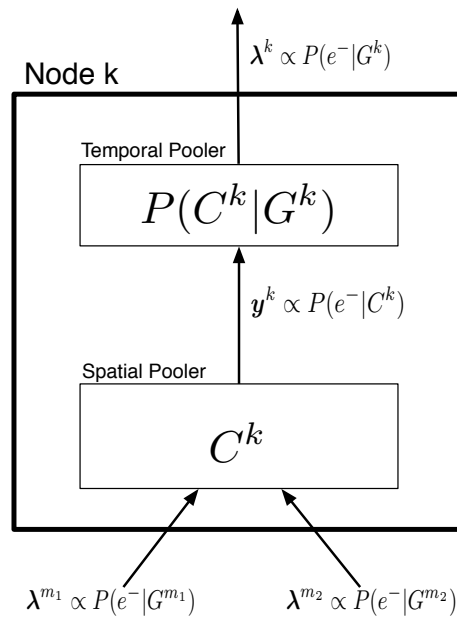


Figure 28: Details of the belief propagation computations within a node.

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 $c_4 = [2, 5]$. C is the collection of all c_i 's.

In section 4.2 we saw how the temporal groups within a node are learned. For Belief Propagation purposes, we need a conditional probability matrix that can be derived from the 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} coincidences.

This matrix is formed by combining the temporal groups information and the frequency of occurrence of each coincidence, both obtained through learning. 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 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.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, \dots, 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,

$$\lambda^{m_i} \propto P(e^-|G^{m_i}). \quad (4)$$

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 .

The spatial pooler calculates its output y based on these inputs. This output is proportional to $P(e^-|C)$. This output is a vector of length N_c where N_c is the number of coincidences in the spatial pooler. The i^{th} component of this vector corresponds to the coincidence c_i . In general, this coincidence can be thought of as represented as a vector of M numbers $[r_{m_1}, r_{m_2}, \dots, r_{m_M}]$ 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

$$y(i) = \alpha_1 \prod_{j=1}^M \lambda^{m_j}(r_{m_j}) \quad (5)$$

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.

Since, $P(e^-|c_i) = \prod_{j=1}^M P(e^-|g_{r_{m_j}}^{m_j})$ and since λ^{m_i} are proportional to $P(e^-|G^{m_i})$ for all i , this ensures that y is proportional to $P(e^-|C)$

The temporal pooler calculates its output based on the input from the spatial pooler. 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. This is calculated as follows. The i^{th} component of this vector is calculated as,

$$\lambda(i) = \sum_{j=1}^{N_c} P(c_j|g_i)y(j) \quad (6)$$

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 is proportional to $P(e^-|G)$

The output of the temporal pooler is the output of the node. The computations within node k are shown in figure A.1.

A.4 Belief Revision Computations

The Belief revision computation is almost identical except for the replacement of the sum in equation 6 with max.

$$y^*(i) = \prod_{j=1}^M \lambda^{*m_i}(r_{m_i}) \quad (7)$$

$$\lambda^*(i) = \max_j P(c_j|g_i)y^*(j) \quad (8)$$