Department of Electrical and Computer Engineering
Faculty of Engineering, School of Graduate Studies
Addis Ababa University

A Proposal and Implementation of a Neural Network Based Hierarchical Temporal Memory to Realize Cognitive Functions

A Thesis Work Proposed, Researched, Documented and Presented to the Department of Electrical and Computer Engineering as a Partial Fulfillment of a Graduate Level Study in Computer Engineering.

Researcher:
Daniel Admassu Teffera _____

Advisor:
Dr. Kumudha Raimond _____

Faculty :
Technology

Department:
Electrical & Computer Engineering

Major:
Computer Engineering

Title of Thesis:
A Proposal and Implementation of a Neural Network Based Hierarchical Temporal Memory to Realize Cognitive Functions

Acknowledgement

This research involved mainly new theories in the current study of cognition, which has made it fairly difficult in both concept and implementation. Several factors contributed to the successful completion of the work. I would like to mention a few of the prominent ones.

Numenta, Inc., original inventor of the HTM model, has made the algorithms and their development platform publicly available online. These resources have been a vital aid to this research.

My advisor, **Dr. Kumudha Raimond**, dedicated a considerable amount of her effort and academic time in addressing several key technical issues of the work as well as in editing the final report.

In addition **Mr. Deleep George** (Co-founder of Numenta), **Mr. Greg Kochniac** and **Mr. Alan Saulius** (participants of the Numenta online forum) provided useful advices in the development of my basic HTM code at the early stages of the work.

Table of Contents

Figure **Page**

Table **Page**

Abstract

Hierarchical Temporal Memory (HTM) is a recent innovation in cognition science. Developed in 2005 by Numenta Inc., an artificial intelligence research firm in the US, HTMs attempt to capture the way the human brain learns and infers its environment. One of the most notable characteristics of this model is the consideration of the hierarchical organization of objects in the world. Data in the world is made up of elementary features that aggregate in successive layers to form perceivable objects. This data can be visual, auditory or from other abstract spaces such as stock markets and scientific studies. The amount of raw data that the brain is exposed to throughout its lifetime is beyond imagination. However the brain is known to use a very noble and systematic approach to handle the perception, storage, and inference of this data. Several studies in neuroscience and psychology indicate that the brain makes use of the hierarchies that features in the world exhibit in their organization to form objects. Hence, for instance, 'corners' and 'lines' can aggregate to form a 'table' object in the visual world. These elementary features, however, can use a different aggregation to form a 'chair' object. The same is true for data in other types of worlds such as audio. HTMs directly apply a similar handling of world data for their cognition. Furthermore, the structure of HTMs, made up of data processing nodes arranged in a hierarchical tree, mimic the physical arrangement of cortical layers in the brain.

The various data analysis algorithms in the nodes of HTMs were, however, found by the researcher to be limiting in several aspects, one of which is handling of unforeseen (untrained) data. For this purpose neural network data structures were used to replace some of the operations in these nodes for their ability in approximating untrained data to nearest matches.

This research work discusses the proposed model, implementation constraints, the operational characteristics, and performance enhancements observed in this modified model with a selected test application. A substantial improvement in cognitive functionality has been observed with the newly proposed model.

Chapter – 1

Introduction

1.1. What is artificial intelligence?

"The study and design of intelligent agents" is the modern definition of artificial intelligence (AI) where an 'intelligent agent' is a system that perceives its environment and takes actions which maximizes its chances of success [28]. The philosophical roots of AI date back as early as ancient Greece. The development of AI has been strongly linked to the study of logic, mathematics, statistics, neuroscience, as well as the improvement of computer hardware and programming languages.

Within the current framework of study various streams of study exist. One such study, symbolic AI incorporates cognitive simulation, logical AI and knowledge based AI. Sub symbolic AI, with pattern recognition and the concept of learning, covers a range of 'computational intelligence' topics. Other more recent approaches include the intelligent agent paradigm and intelligent agents. Other schools of philosophy tend to categorize AI into 'strong AI' that addresses systems capable of general level intelligence and 'weak AI' with systems that exhibit intelligence in a predetermined area of purpose [28].

In recent times AI development has steadily gained attention as the number of possible application scenarios and the power of computational platforms grow. Large research projects dedicated to studying the subject for its own sake rather than for practical purposes are very common. Apart from the analytical and theoretical research there has also been a huge flow of resources into developing powerful platforms. Most notable is the IBM blue gene project which is used to simulate brain functions [26][27]. It seems indispensable to acquire a competent platform to run these systems if a truly intelligent machine needs to be implemented.

1.2. The science of cognition

'Cognition' is a scientific term that draws significance from the natural processes of perception, memory, recognition, planning, motor behavior, and is taken as a manifestation of intelligence in natural entities [25]. However the term is nowadays being widely applied in the science of AI as well.

Several schools of thought and modeling paradigms exist within the study of cognition as applied to AI. Among these the emergent models are approaches embracing connectionist systems, dynamical systems, and enactive systems, all based to a lesser or greater extent on principles of self-organization. In particular connectionist systems involve an interactive set of dynamically adapting units and include those based on neural networks, which are a subject of this work.

Connectionist systems rely on parallel processing of non-symbolic distributed activation patterns using statistical properties, rather than logical rules, to process information and achieve effective behavior. Biological neural networks (BNNs) are considered to be the main inspirations behind artificial neural networks (ANNs), which are at the forefront of the connectionist model.

Neural networks have emerged in different structures and working principles. Self organizing maps (SOM), radial basis functions (RBF), the Boltzmann machine and the perceptron model are among the popular ones [23][24]. The most widely used model of neural network, however, is the multi-layer perceptron (MLP). Even though various irregular structures and operating modes exist for the MLP, the regular fully connected structure with a feed forward operation is the default configuration for different applications. Given the proper topology and operational parameters, the MLP has the ability to cope with extended input set size and high precision in approximating untrained input patterns to nearest outputs. This has made it the subject of research in various recognition applications.

1.3. Literature survey of cognitive models

Due to the relatively recent nature of the study and the lack of a general consensus on many of its philosophical and operational theories, AI is one of few areas of science that is documented with mainly hypothetical and contradicting publications [2].

The research in cognitive models is no exception. In order to assess the relative advantages of each model and the feasibility of merging two of the models, namely neural networks from the connectionist pact [12][23] and HTMs from the hierarchical ones, a fairly wide review of many of the models was conducted.

Hierarchical modeling of cognitive structures had its own emergence with first links to non cognitive computing applications [4]. Such works involved the modeling of a dynamic memory environment to implement in machines of fast access cache architecture in specific applications. Although the research is limited to computational platforms it demonstrated some of the shortcomings of monotonous storage architectures. As an alternative to artificial cognition models, the theory of hierarchy did not emerge until recent years. In the paper, brain-inspired conscious computing architecture [10], the author attempts to model the brain with a very large topology neural network model with no sub-structuring of hierarchical type or otherwise, while others [16][17] try to demonstrate the concepts of feedback in neural networks when applied to visual recognition problems with a simplified circuit as a building block of a larger monotonous architecture.

Neural networks themselves, even though having an older history, have passed several application attempts in cognitive models. A description of one model [9] addresses pre-frontal visual cortex of the brain as a model for a visual cognitron by a direct mapping of the working memory structures of the neurons. Another similar research [7][8] also attempts to directly map the physical structures of neuronal circuits in the brain to an artificial memory system. These researches base their motivation on the notion of the natural brain as the most efficient cognitive memory architecture currently available.

The mapping of the brain in functional behavior rather than structure did not get attention until the recent work of Dileep George and Jeff Hawkins, invariant pattern recognition using Bayesian inference on hierarchical sequences (2004) [13]. The theory is a Bayesian inference model of cognition employing a hierarchical chain of decisions. It did not develop into the more robust theory of hierarchical temporal memory until late 2005 [18][20].

Other works involved employing neural network structures in non-monotonous organization, albeit not truly hierarchical. Mostefa et al. [1], Lin et al. [11], Miikkulainen et al [5] try to model forms of cognition with different architectures that involve the use of neural networks as an inference engine.

It is notable also that an evolution based neural network model has been explored by Garis et al. [3] as a research into a task survival based evolution of a potential artificial brain.

As a most recent development, hierarchical temporal memory [19] expand the theory of hierarchy in terms of functional modularity as well as storage. This model provides a direct match to the natural process of cognition in an intuitive manner. The concept of hierarchical storage of world data features and their aggregations to attain an efficient storage and retrieval system has gained strong acclaim from researchers in the field. Furthermore a functional analogy of the cortical layers of the brain and this model has been observed [15].

Hence the researcher has concluded that HTMs currently offer the best organizational structure in the area of modeling neural network based cognition.

1.4. What is Hierarchical temporal memory?

Formatted: Bullets and Numbering

Hierarchical temporal memory (HTM) is a machine learning (artificial cognition) model developed by Jeff Hawkins and Dileep George of Numenta, inc. That models some of the structural and algorithmic properties of the neocortex as Bayesian networks [15]. It is widely considered a novel approach for proposing functions of the cortical layers. It is considered by Numenta as a new computational paradigm based on cortical theory.

1.4.1. Background concepts

Formatted: Bullets and Numbering

It is a long discovered fact that the neocortex, the part of the brain responsible for cognitive functionalities, works on a common algorithm for tasks that vary from vision to language [8]. A single algorithm is used in the classification of sensory input data form different types of biological sensors. Thus the various trait of behavior and perception are processed in the neocortex as a single type of input data and only maintain their unique processing in the perception or motor areas of the brain allocated to the specific sensor. HTMs share this important characteristic of the biological system. The very concept of data classification in HTMs depend on 'cause discovery'. This is to mean that every separate input to the HTM network is considered to have its own cause. The input, thus, is taken as part of a larger input that has not been fully sensed yet by the particular unit that senses this input. This dictates the absolute necessity of time in learning causes and comprises the 'temporal' nature of this model. Data inputs that consistently occur closer together in time (i.e. One after the other) are taken as being part of a larger cause. This corresponds directly to reality where parts of a 'cause' occur sequentially in time to the receptor. Input patterns that occur in spatially close formation (e.g. One besides the other) are also considered to have a special relationship with each other. This other concept comprises the 'spatial' nature of the model. Hence the model synchronizes the spatio-temporal nature of the real world in a single cognition framework.

Another key concept of the model is the concept of 'hierarchy'. In the real world objects (visual, auditory, or any other) can be observed to have a hierarchy of sub-objects that are shared among higher level objects. For instance a simple diagram of a car and a house may share lower level objects such as horizontal or vertical lines. Hence the upper level objects share these lower level objects during learning or inferencing by the HTM unit. In the actual world, however, this hierarchical composition of objects may be much more complex. In addition other input data type such as audio may not have an easily readable cause-object relationship. This 'cause sharing' of higher level objects from lower level ones results in the storage efficiency of HTMs. It is worthy to note also that the biological brain has also been a center of research in the area of storage as the amount of data it emulates as storing and its sheer size and retrieval speeds do not appear to match.

Mapping the exact 'parent-child' looking relationship of real world objects, HTMs are modeled in a tree shaped hierarchy of nodes, with each node having its own spatial and temporal classification functionalities and operate with the same algorithm as the others. Each node has input and output vectors. The input vector is fed to the node for spatio-temporal analysis and the result is written on the output vector. Nodes at the lower level of the hierarchy, i.e. Leaf nodes, are the first stages of reception for world data. In a visual task, for instance, they can be fed row pixel data of the sensed object. The nodes in the level immediately above the receptors are fed the outputs of two or more child (receptor) nodes depending on the tree architecture. These nodes in turn perform spatio-temporal analysis on their input using the same generic algorithm and output the result for higher level nodes. Hence as one goes up the hierarchy nodes classify higher and higher types of objects (e.g. From pixels, to lines, to corners and to shapes in visual case). The most important thing in HTMs is that the causes discovered by each node is not pre-determined or programmed by the designer of the application.

1.4.2. Current implementation

Formatted: Bullets and Numbering

The current developer of HTMs, Numenta Corporation, has a basic implementation of the concept in each of the identical nodes of the HTM tree [18].

Each separate computing unit of HTMs, called node, has two distinct parts and two main operations corresponding to these parts. The first section, the spatial pooler, is responsible for accepting input data and making spatial classifications based on the pattern. The term 'spatial' should not be misleading in that the pooler is useful for visual input only; rather it means the position of the input pattern's elements with respect to each other. The spatial pooler uses a distance metric computation to classify distinct spatial patterns. Hence a greater distance metric value tends to gather more patterns into a single group, whereas a smaller one assigns each different pattern its own classification. This technique is used to deal with noisy inputs. After classifying input patterns, the pooler begins generating outputs for each incoming input based on the classified categories. The output of the spatial pooler is a vector set of beliefs as to where the current input is classified. Hence a normalized belief vector is written on the output of the spatial pooler which in turn in is the input of the next stage of the node.

The next stage, the temporal pooler, undertakes a more complex operation. The basic job of this pooler is to classify the categories generated by the spatial pooler into a set of temporally adjacent clusters and make temporal inference based on these clusters. Classification decisions are made based on the consistence of a pattern in appearing close in time to another pattern. For this purpose various data structures are employed, the main one being the temporal matrix. This matrix maintains a record of all the patterns the pooler came across and the number of appearance of each pattern after the pattern that appeared previously. At each arrival of a pattern the specific entry in the matrix that corresponds to the arriving pattern row which is under the previously appearing pattern column is incremented with a chosen value. For better performance a non linear priority function is used to increment values of these entries in precedence of late appearance.

After the construction of the matrix is complete the pooler runs a clustering algorithm on the matrix to classify the categories into groups. This classification operation results in groups of patterns that appear closer in time to each other more than the rest. These groups will serve as easy references upon inference operation of the node in determining which feature of an input space is currently being observed. In a visual perception application, or instance, the patterns are classified into clusters of 'vertical line', 'horizontal line', 'lower left corner' and so on. This is possible because the patterns in each of these clusters are more likely to appear closer in time than the patterns in the rest of the clusters. The algorithm for this clustering is the most important one in determining the performance of the node since these clusters provide the means for position/focus/orientation (or all, depending on the desired purpose) invariance to the categorization. Hence a' line' will be classified as a 'line' no matter what position or which orientation the perceived line has.

After training of the node is over, it is said to be ready for inference (recognition). The spatial pooler has to be trained first and put into its own inference mode before the temporal pooler is trained. In the HTM hierarchy lower levels are trained before upper (parent levels). When a level has finished training it is put into inference mode and serves as an input generator for its parent level, which in turn is put into training mode. A number of optimization techniques (or shortcuts) can be employed for faster performance of the network training. Node cloning is used to clone a trained node to all other nodes in the level instead of training each one in turn.

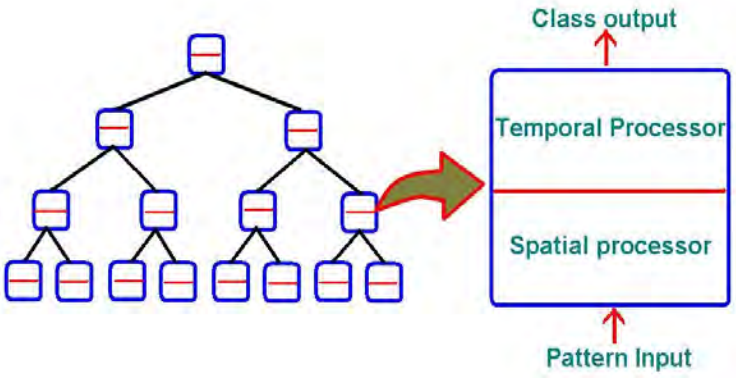


Fig. 1.1 – Implementation of the HTM network

Formatted: Bullets and Numbering

1.4.3. Limitations

As is discussed in the structure and operations of the current HTM implementation, the model is made up of several processing and storage areas to analyze, classify, store and infer patterns. All of these operations are computation based routines that search and match patterns according to explicit criterions and appearance history.

The HTM model is noble in providing autonomy to a cognitive engine in that no pattern cluster organization or type of members in a cluster are predefined or guided by the designer. In contrast the natural adjacency and frequency of appearance of patterns (and hence features) in the real world are guides to the formation of these clusters. This leads not only to the discovery of distinctive features of objects but also the hierarchical relationship between them, resulting in a tremendous amount of conserved memory and efficient operation.

However this autonomous characteristic remains limited in the outermost topology of the network. The internal structure of the nodes themselves is a pure computation based implementation. Patterns are analyzed for elements, discriminated for noise with well known vector dissimilarity relations, stored in tables, and searched for matches.

This approach has a few serious limitations.

a. Problem in dealing with untrained patterns:

Computational algorithms in general rely on explicit associations of data to make decisions. Upon encountering unforeseen data, they tend to introduce error or default to a preset output [22]. The current HTM nodes make use of the spatial noise discriminator functionality, which is designed to discriminate against random noise elements limited in extent, to deal with unforeseen data as well. A Bayesian belief output is computed for the top categories. When the encountered data deviates beyond the threshold of the discrimination function for the beset match, an error signal is generated and the object is tagged as ‘unknown’ in the output. One may argue that increasing the threshold value may solve the problem. However the purpose of the discrimination function is to filter out random noise and increasing the threshold, in anticipation of unforeseen variants of patterns, results in the clustering together of unrelated patterns. Thus at best, the implementation is a tradeoff between noise intolerance and feature approximation.

b. Inconvenience of node algorithms for parallel running machines

Apart from the requirements of explicit data association in the algorithms and their problem in dealing with unpredictable patterns, the search-and-match routines make the current node implementation a bottleneck in exploiting parallel running systems that are expected to be the main platforms in future cognition modeling. Present research in such machines reveals that the design tendency is towards systems made up of a large number of processing nodes running in parallel and possibly in hierarchical organization [27].

Although single nodes of the HTM can be assigned to such processing nodes of these machines and exploit their powers, the true advantages may not be harnessed until single processes in each HTM node can run in data independence manner with other processes and assigned its own processing resources.

Hence the researcher argues that the current HTM model is limited in its ability to be a future model for implementing in such dedicated systems and a further re-organization of the node structures is required to meet this goal.

1.5 Statement of the problem

Formatted: Bullets and Numbering

The main task of this research, as can be deducted from the discussions concerning the limitations of the current model, is to enhance the performance of the model by modifying the internal structures of the HTM nodes. The motivations behind such an enhancement are clear. Better level of dealing with unforeseen (untrained) data and suitability for parallel running are the top characteristics expected from a cognition system that attempts to attain performance that approaches that of natural systems.

Thus the researcher believes the task is to explore various approaches of cognition modeling, some of which naively trying to mimic structures of the brain, and provide a possible solution in a form of a hybrid model from paradigms each of which best represent one of the intended characteristics such as better mapping of the hierarchical organization of objects in the world and good classification approximation in handling unforeseen data.

Formatted: Bullets and Numbering

1.6. Objectives

In implementing the discussed enhancements and analyzing the characteristics of the new model, a number of achievements are expected from this work.

1) General goals

a. Explore theoretical possibilities of merging the natural cognitive functionality and architecture with artificial implementation. This will serve as an insight into the feasibility and compatibility of the connectionist systems, namely neural networks, with mainstream theoretical cognition modeling.

b. Propose a candidate model for artificial cognition based on combination of hierarchical and neural network connectionist classes. This new model will be proposed as a competitive new model for applications and future enhancements.

2) Specific goals

a. Implement the proposed model that includes the neural network enhanced components in software emulation and evaluate its resource and performance constraints.

b. Develop a specific test case application scenario and assess the performance of the model against both generic and comparative measures. The comparative measurements will help to assess the performance in relation to the original model while generic tests, designed by the researcher, will evaluate the proposed model with scenarios likely to be encountered by a cognition engine of this scope. A visual image recognition problem is selected for which a test application will be developed that makes use of the new cognition engine.

1.7. Proposed solution

As indicated, the proposed solution for this problem in this work is a hybrid model of the HTM and the well known connectionist model of neural networks. Thus, the original HTM model itself was found to be the best match to the hierarchical organization of objects in the modeled world while neural networks are known to have the advantages of parallel architectures and better output approximations.

Pattern classification is done primarily in the spatial processor of the HTM nodes. The current implementation of the node structures is illustrated below in fig1.2.

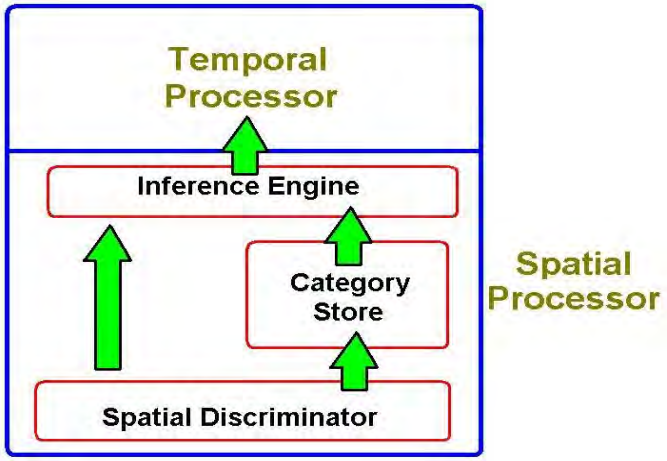


Fig. 1.2 – Functional structures of the spatial processor in the current HTM node.

The inference engine is the main component of the processor in the task of pattern inference. It compared and matches patterns that arrive with table entries in the category store and forward an inference decision to the temporal processor. This makes it the responsible component in dealing with uncategorized data that comes from the discriminator at inference time. As such it has been the main focus of enhancement in the form of neural network data structures. Neural networks by nature are not only classifier engines but also store the learned data in the form of weight values between the neurons. This eliminates the need for the category store as well. The proposed model is shown below in fig. 1.3.

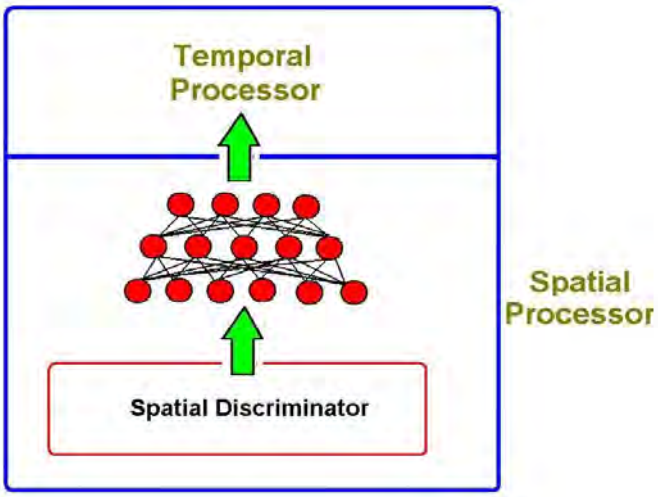


Fig. 1.3 – proposed modifications to the spatial processor of the HTM node.

The figure 1.3 illustrates the replacement of two of the components of the spatial processor in the node. The neural network, when the node is in training mode, is trained for the input patterns and their associated representative category outputs. Thus the various spatial pattern both with and without noise are stored as weight values in the network. When the node is in inference mode, the network is provided with inputs and its output is directly fed to the temporal processor part of the node. In this mode, unforeseen input patterns such as patterns with noise are approximated to the nearest matching output categories.

The output approximation of neural networks is expected to offer superior results in comparison to any other analytic computation approaches such as the Bayesian belief output comparison used in the original model by Numenta.

1.8 Thesis organization

This thesis work is organized in flow that attempts to provide an intuitive insight into the theoretical realms behind some of the concepts involved and the actual design, implementation and testing activities that have taken place.

The introduction part is organized in chapters that deal with concepts of common AI studies and paradigms in cognition. Some of the historical as well as current models are explored in this section. Special attention is given to one of the paradigms, namely connectionism, in order to give insight on the chosen neural network model and its behavior. As the performance of neural networks greatly depends upon the choice of parameters that affect its learning behavior and topology, a special design approach in the form of evolutionary theory is utilized. A background on this theory and how it will be applied for evolving neural networks is discussed.

The second part, analysis and design, is dedicated to analyzing the conditions that make up the problem to be addressed both theoretically and with respect to resource availability. Thus by setting the appropriate limitations and performance expectations a design phase follows in order to reach at an optimal model configuration. A chapter is dedicated to a special design process that deals with optimizing neural networks in evolutionary methodology.

The third part of the paper is a discussion of the practical implementation conditions and the testing methodology adapted. In these chapters the architecture of the developed software for both the cognition engine and the test application are discussed. Various limitations in both hardware resources and coding standards are presented. Finally the testing mechanisms selected and the formats of the generated data are addressed.

A section of this part that deals with the results and the corresponding discussions on them tries to analyze the achievements of the model in contrast to generic expectations as well as comparative tests against the original model. The operational characteristics of the new model are analyzed as well for conformance of the neural network implementation to the basic theory of HTMs.

A conclusion section at the end will wrap the achievements and deductions from this work as well as suggest a few but the most straightforward recommendations.

Chapter – 2

HTM Fundamentals

2.1. Operations of the HTM

Formatted: Bullets and Numbering

2.1.1. Topology analysis of HTM networks

Formatted: Bullets and Numbering

HTMs are tree shaped hierarchical collection of nodes each of which are capable of discovering and inferring 'novel causes' or features from real world objects. Their design explicitly assumes a hierarchical relationship among features of objects in the modeled world. Hence each level of the HTM tree corresponds directly to a level of abstraction in the collection of features or causes in the object.

Numenta corp. Invented and introduced HTMs [16]. Various aspects of the technology are publicly available from their website. With respect to tree topology there is a theoretical set of possible configurations applicable as per the requirements of the modeled problem. Some of which are shown in figure 2.1 below:

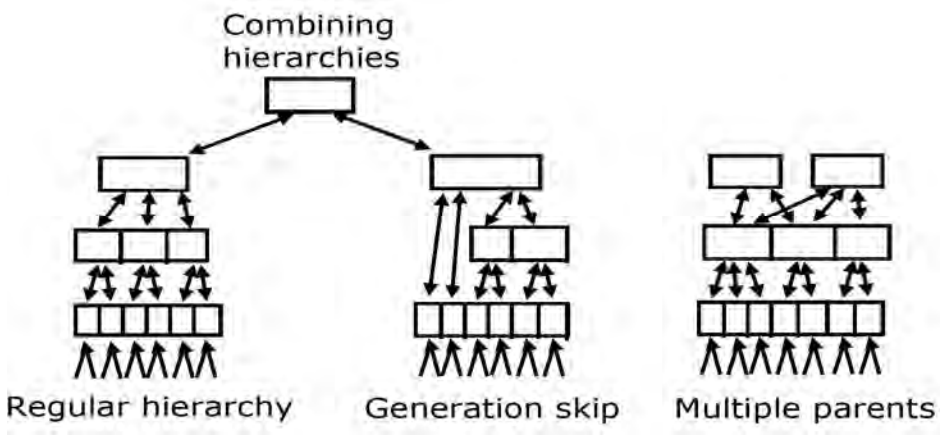


Fig. 2.1 - HTM topologies

The topology design of a given tree depends directly on the nature of the perceived world. There were a number of guiding factors [20] while deciding on the topology structure for the model in this work. The major ones are discussed below.

I. Dimension of the input data

This describes the inter-data relationship between perceived portions of the instantaneous input space. For instance a vision domain corresponds to a 2-dimensional input space and a text stream domain will be modeled in a single dimension. This directly dictates the arrangement of child nodes under any given parent node.

II. Complexity of the modeled object

Numerous levels of complexity exist in the feature set of any given object. For example a 'sentence' object in a text analysis application may be composed of aggregations of 'word', 'sub-word', 'phoneme', and 'letter' features in descending order. In comparison a visual object can be decomposed into 'section', 'corner', 'lines', and 'points'. These levels of aggregation directly correspond to the number of levels in the hierarchy. Clearly the level of complexity in real world objects can be overwhelming to simple HTM trees.

III. Size of perceived objects

Even though the 'size' of an object to be modeled is easy to imagine in a sense of a visual example (such as the breadth and width of a geometrical shape) it applies equally to other domains of input as well. This size directly dictates how much of tree coverage is required in order to fully perceive a given object and hence directly corresponds to the number of leaf nodes in the tree.

The impacts of each of these factors on the actual design are taken in chapter 4.

2.1.2. Node design

Nodes in HTMs have a consistent structure in all levels. This also holds true for the employed nodes in this proposed model. This may not be necessarily true in the original Numenta HTM implementation which has a variety of nodes for a variety of purposes. In the simplest of Numenta HTM tree the top most node differs from other nodes in that it uses a different temporal association algorithm and performs prediction as well. This however is out of the limited scope of this discussion and not a feature of the proposed model.

I. structure of the nodes

As any generic HTM node (see fig. 2.2) the nodes in this model have two distinct parts and two separate operations that correspond to these parts. The first stage is the spatial processing section and the second is the temporal processing stage.

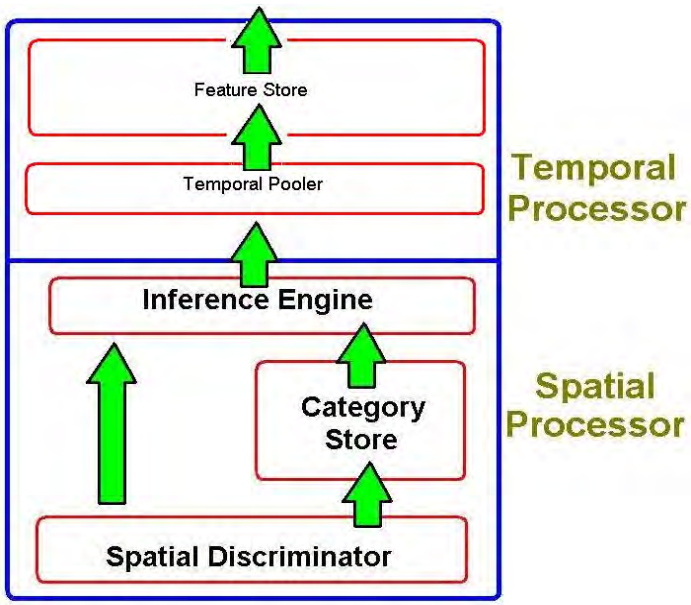


Fig. 2.2 - HTM node structure

Spatial processor: this specific part of the node processes instantaneous input data for spatial information against a set of statistical record. The job of this processor is to discriminate input patterns based on their spatial arrangement of elements and group them accordingly. Hence in the sense of the specific test application chosen input patterns are pixel data gathered from the input image for lower level nodes and concatenated outputs of child nodes for higher level nodes.

The discrimination of input patterns into groups has to consider two main factors upon deciding on the discrimination criterion, called the spatial similarity criterion [16]. These are;

1. Complexity of objects
2. Expected noise

Complexity of objects refers to the observed difference and similarity of spatial features in the object. The more complex the object is the more 'similar looking' the spatial input patterns will be. Hence for complicated class of objects a lower value of the similarity criterion should be used while for less complicated, simpler objects even a higher value would give a decent result.

Expected noise in the input space is the amount of non-causal data in the perceived object that may affect the categorization decision. This noise can be a random variation of pixel data in that is not caused by any of the features. Filtering out noise demands a higher value of the spatial criterion variable.

Thus there is a tradeoff between dealing with complexity and noise filtering. A problematic situation will be an image input set where the objects are highly complex and there is also a high level of noise in the data. However this is not much different in the natural process of cognition where, for instance, complex script characters are very difficult to decipher when coupled with high level of obstruction noise or poor print quality whereas simpler characters would be recognizable in even high amount of noise.

The spatial processor may use a number of optional techniques to compute the spatial difference value between two given vectors. There are choices of spatial dissimilarity test rules in literatures which include Euclidian and variants of hamming distance measures. A few are listed in table 2.1 below [14].

Given two binary vectors x & y, let $s_{i,j}$, $i,j \in (0,1)$ be the number of occurrences of matches with i in x and j in y at the corresponding places.

Measure	$S(X,Y)$	$D(X,Y)$
Jaccard-Needham	$\frac{S_{11}}{S_{11}+S_{10}+S_{01}}$	$\frac{S_{10}+S_{01}}{S_{11}+S_{10}+S_{01}}$
Dice	$\frac{S_{11}}{2S_{11}+S_{10}+S_{01}}$	$\frac{S_{10}+S_{01}}{2S_{11}+S_{10}+S_{01}}$
Correlation	$\frac{S_{11}S_{00}-S_{10}S_{01}}{\sigma}$	$\frac{1}{2}-\frac{S_{11}S_{00}-S_{10}S_{01}}{2\sigma}$
Yule	$\frac{S_{11}S_{00}-S_{10}S_{01}}{S_{11}S_{00}+S_{10}S_{01}}$	$\frac{S_{10}S_{01}}{S_{11}S_{00}+S_{10}S_{01}}$
Russell-Rao	$\frac{S_{11}}{N}$	$\frac{n-S_{11}}{N}$
Sokal-Michener	$\frac{S_{11}+S_{00}}{N}$	$\frac{2S_{10}+2S_{01}}{S_{11}+S_{00}+2S_{10}+2S_{01}}$
Rogers-Tanmoto	$\frac{S_{11}+S_{00}}{S_{11}+S_{00}+2S_{10}+2S_{01}}$	$\frac{2S_{10}+2S_{01}}{S_{11}+S_{00}+2S_{10}+2S_{01}}$
Rogers-Tanmoto-a	$\frac{S_{11}+S_{00}}{S_{11}+S_{00}+2S_{10}+2S_{01}}$	$\frac{2(N-S_{11}-S_{00})}{2N-S_{11}-S_{00}}$
Kulzinsky	$\frac{S_{11}}{(S_{10}+S_{01})}$	$\frac{S_{10}+S_{01}-S_{11}+N}{(S_{10}+S_{01}+N)}$

Where $\sigma = ((S_{10}+S_{11})(S_{01}+S_{00})(S_{11}+S_{01})(S_{00}+S_{10}))^{1/2}$

Where:

The s(x,y) field lists similarity test relations while the d(x,y) field lists dissimilarity.

Table. 2.1 - Popular vector dissimilarity criterion relations

Among the listed approaches the 'Sokal-Michener' similarity test measures were chosen for the model due to their relatively better performance during prototype stage of the model. The Numenta HTM spatial pooler makes use of a simple euclidian dissimilarity measure denoted by

$$d^2(x,w) = \sum_{j=1}^{N_{dims}} (x_j - w_j)^2$$

Where x and w are two binary vectors compared iteratively at every element position j.

Eqn. 2.1 – The Euclidian vector dissimilarity relation

The Sokal-Michener test, although a little more complicated, gave better discrimination results when compared to the Euclidian test in the original HTM model. The spatial pooler maintains an array of encountered spatial patterns and groups similar patterns based on the similarity test. It also keeps the number of times a group has appeared for later use by the temporal processor.
If the input image is believed to be noise free the spatial similarity criterion can be set to 0, given the input objects are clear line drawings of objects.
When the node has learned all the signals and is put in inference mode the spatial pooler uses the similarity relation and criterion to compute a similarity test for the current input vector against each of the grouped distinct categories. The learning and inference stages use a set of algorithms to discriminate and cluster patterns which will be discussed in the algorithms section in detail.

Temporal processor: the next stage after the spatial processor is the temporal processor. This stage is responsible for clustering together temporally coherent sequences of patterns. These clusters correspond to 'features' of the object for that level of the hierarchical tree the node belongs to.

The temporal processor operates through several stages of operation:

- During learning mode, the temporal processor builds the time-adjacency matrix, which keeps track of transitions between coincidences.
- Before the node is switched from learning to inference, the temporal processor builds a list of groups from the time-adjacency matrix.
- During inference, the temporal processor uses its list of groups to convert incoming belief vectors to distributions over groups. The time-adjacency matrix is not used, although it is kept around for debugging purposes.

The input to the temporal processor is a belief distribution vector from the output of the spatial processor. During learning mode the temporal processor builds a time adjacency matrix that

keeps track of the temporal occurrence information of any one of the categories in relation to the others. A sample temporal adjacency matrix may look like the example table below;

		Categories				
		A	**B**	**C**	**D**	**E**
Categories	**A**	4	1	0	1	1
	B	1	5	3	0	0
	C	1	0	4	3	4
	D	0	4	2	5	2
	E	2	1	0	0	3

Table 2.2 – An example of the temporal adjacency matrix at one time instance

The matrix holds a 'happened-after' information for the categories in each row for the previously occurring categories in each column. Note that the matrix is initialized to all 0s. At each occurrence of a category the previous category is updated with an increment amount of any user defined value. This way the matrix keeps track of which category happened how often after each of the other categories. There are a number of key concepts in temporal matrix construction;

Symmetric learning: this refers to updating for a reverse ordered occurrence of categories upon the occurrence of a sequence. For instance, if category c occurred after category a in a specific sequence instance, after updating the entry under row c and column a, the entry under row A and column C is also updated with the same increment. This is to signify the spatial adjacency of features in that if a feature happens to be near to another feature, then that other feature is also likely to happen after the current feature. This approach is entirely optional but reduces learning time significantly.

Temporal history: instead of only updating the matrix column entry for the category that immediately precedes the current category, a history of 'previously-occurring' list of categories may be maintained so that all categories may be updated. Higher priority will be given to the immediate category and incremented with a higher value; whereas for the successive categories, the incrementing value will be reduced proportional to the distance from the current category. This approach is clearly advantageous in capturing the significance of nearby features other than the one in the immediate past and also reduces learning time. A detailed description of the operation of the temporal processor is given below in the algorithms section.

II. Node algorithms

Spatial processor learning: with the exception of using a new discrimination function rather than the Euclidian criterion, the learning stage of the spatial learning uses the same 'Gaussian' learning approach towards clustering novel input vectors. The algorithm can be summarized as follows;

1. Receive input vector
2. If input is the first vector assign it to a new category and make it the representative vector of that category and mark the occurrence value of the category to 1.
3. If not, compute the vector dissimilarity value between the input vector and the representative vectors of each of the existing categories in the spatial discriminator with the Sokal-Michener relation as:

$$D = \frac{2S_{1,0} + 2S_{0,1}}{S_{1,1} + S_{0,0} + 2S_{1,0} + 2S_{0,1}}$$

Eqn. 2.2 - The Sokal-Michener dissimilarity relation

   a) If the value, d is less than the set criterion, assign it to the current category and increment the occurrence value of that category.
   b) If no category has a qualifying dissimilarity measure, assign the input a new category and make it the representative vector of that category and mark the occurrence value to 1.

In this way the spatial processor clusters input vectors into spatially dissimilar groups. For a dissimilarity criterion of 0 as many groups as the number of distinct input vectors are formed with each group containing only one pattern. During this process the occurrence value for each category is maintained but not used. It is to be delivered to the temporal processor which is the next stage of the node.

Spatial processor inference: the inference stage is similar to the learning stage but at this stage no new categories are created. Instead incoming input patterns are compared to each of the clustered categories and a Gaussian probability belief vector is computed in the inference engine.

The process follows steps as described below;

1. Receive an input vector
2. Compute the vector dissimilarity value of the input vector with representative vectors in the spatial discriminator of each of the available categories.
3. Convert these values to belief values using a Gaussian approach in the inference engine.

The algorithm models the input vector as a random sample drawn from one of the categorized Gaussian probability distributions. Each of these distributions is centered on a different coincidence vector in the grouped categories; the categorized groups of vectors represent the mean vectors of the Gaussian distributions. The distributions all have an identical variance, which is uniform across all dimensions of the input space. This variance is the user-specified parameter sigma, also denoted as σ, which is defined as the square root of this uniform variance.

The Gaussian processor computes the belief that the input vector was generated from the cause represented by the j^{th} category as the local probability density of the j^{th} Gaussian distribution, as

$$y_j = e^{-(x-W_j)^2/2\sigma^2}$$

Eqn. 2.3 – A Gaussian distribution relation for computing belief outputs.

Where x is the input vector and w is the representative vector of the compared category. This model provides an exponential decay of the belief in the j^{th} cause with increasing distance between x and w_j.

The representative pattern corresponding to the highest belief output will be given to the temporal processor as an input. Figure 2.3 shows the flow of operations in the spatial processor for both the training and inference patterns in a flowchart.

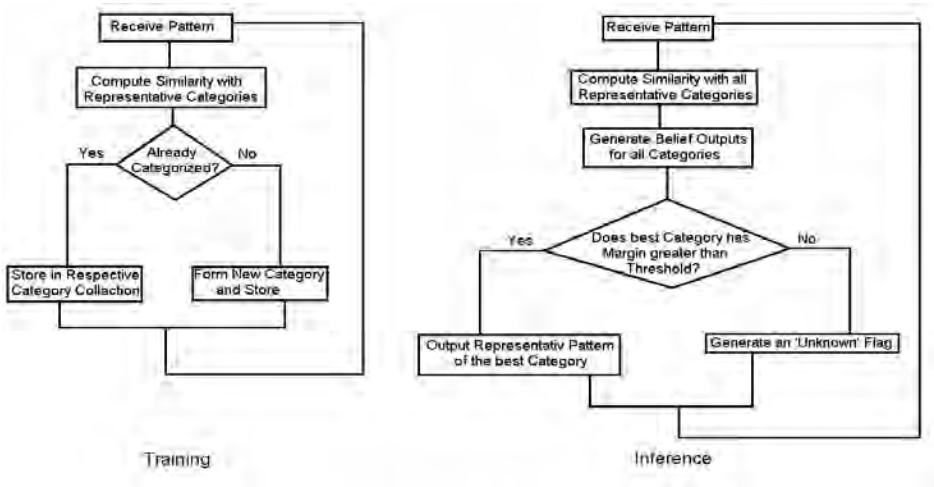


Fig. 2.3 – Spatial processor operations flowcharts

Temporal processor learning: the two tasks of the temporal processor are maintenance of temporal adjacency matrix and formation of temporal clusters.

In the creation of the temporal adjacency matrix the algorithm follows these steps;

1. Receive all the distinct spatial categories from the spatial processor
2. Receive the occurrence values of each of the categories from the spatial processor
3. Upon the occurrence of a category;
   a) If the category is the first to be received put it as the only entry in the temporal history vector
   b) If not, increment the value in the temporal adjacency matrix entry corresponding to the row denoted by the current category and the column denoted by the categories in the temporal history vector (previous time instant categories) with values in decreasing order proportional to the index of the categories in the history vector as shown in the example figure 3.3 below. In the figure c_t denotes the category at the current time instance and c_{t-i} denote

categories time instances in the past with the corresponding categories appearing to the node. Hence a higher increment value, 4, is given to the category in the immediate past (c_{t-1}) and lesser values in progressively decreasing value for the ones before it.

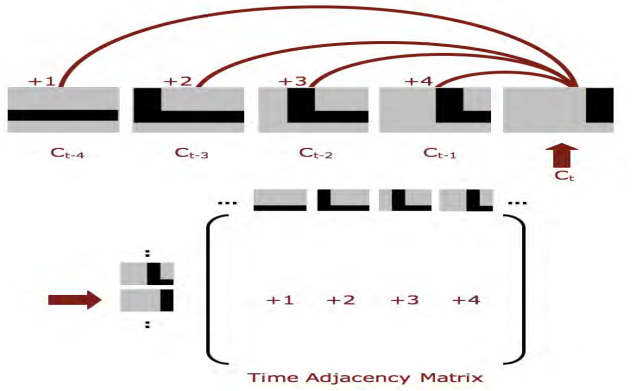


Fig. 2.4 - Temporal history reference

c) Also increment for reverse occurrence sequences with each of the categories in the temporal history.

After construction of the matrix, categories should be clustered into features/groups which is a little more complicated operation which involves;

1. Finds the most frequent category that is not yet part of a group.
   - The most frequent category is the one with the highest corresponding value in the occurrence vector (maintained by the spatial processor).
2. Pick the categories that are the most-connected to this coincidence.
   - The algorithm finds the most-connected categories by finding the highest values in the row of the time-adjacency matrix that corresponds to the current category.
   - The number of categories to pick is determined by a neighborhood constant. The algorithm adds these categories to the current group. Only categories that are not already part of any group are added.
3. Repeats step 2 for each of the neighboring categories.
   - Recursively compute step 2 on their neighbors, and so on, until no new categories are added.

- This process always terminates. Usually, it runs out of new categories to add; as more categories are assimilated, it is more likely that their top neighbors already belong to the current group. But if the group grows to size determined by a maximum size constant, the process is terminated and no new categories are added.

4. All these categories are added to the group lists as a new group and steps 1-4 is repeated.

Temporal processor inference: Inference in the temporal processor simply involves searching of a matching category in each of the groups for the category coming from the spatial processor and, upon a match in a group, give an output of the representative category of that group. This output of the temporal processor then serves as part of the input of the node in the parent level.

Figure 2.5 is an illustration of the processes that take place in the temporal processor for both stages.

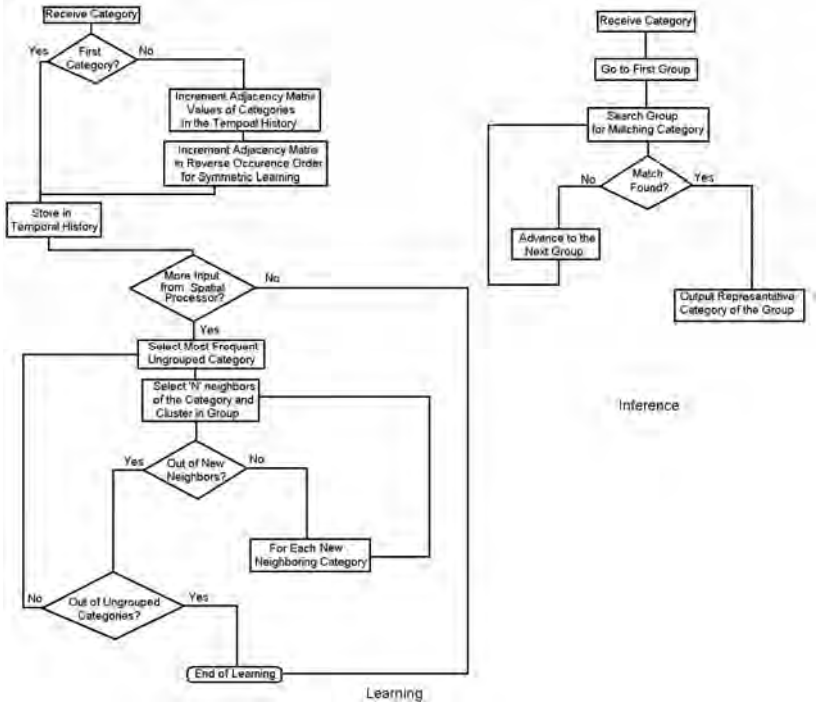


Fig. 2.5 – temporal processor operations flowcharts

Chapter – 3

Neural Networks

3.1 Biological neural networks

What are neurons?

Neurons are special biological cells in the nervous system of higher animals with developed brains. However they are significantly different from other nerve cells in that they are specialized for information storage instead of signal relay only.

What are neural networks?

The important aspect of neurons is that they work as a group in the brain. Each neuron is connected to as many as 10 to 100 thousand others. They form clusters called neural networks for the purpose of handling specific tasks in the perception, storage, or motor behavior of the animal. In higher level animals such as the human being, these clusters are further subdivided in regions and layers for performing advanced operations. A particular region of the human brain usually associated with intelligent behavior is located at the frontal lobe area and is called the neocortex. These regions are made up of layers of neural clusters and exhibit a hierarchical type connection among clusters in successive layers. Figure 3.1 illustrates a simplified neuron and an electron micrograph of a cortical column.

Overall the average human brain contains 100 billion neurons. Together with the large number of connections every neuron maintains, the order of ‘connectedness’ in the brain is astronomical. This seems to be a large number at first glance but studies in the information storage capability of the brain indicate that the amount of data that the brain is exposed to throughout its lifetime far exceeds the storage space formed by these neurons with our current conception of memory. Thus researchers are on an ongoing study to solve this mystery.

The study of the brain has mostly been based on hypothetical theories and models due to the complex nature of its operations. One aim of these studies has been porting the intelligent functional properties of the brain to artificial platforms and, in the process, realize what came to be known as 'artificial intelligence'.

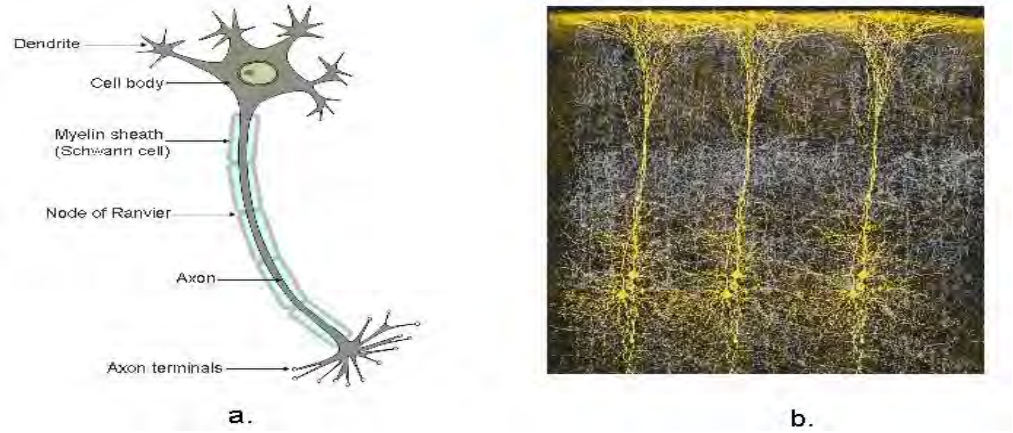


Fig. 3.1 - Biological Neurons a. A single neuron b. Columns of neurons within cortical layers

Some models of the artificial brain concentrate on the structural aspect of the brain while others try to mimic its functional nature. The functional models address the issues of perception, memory and motor actions in functional blocks which are self contained (modular) with limited interactions among each other. These models base their studies on behavioral studies of humans such as motion, reasoning, planning and language.

Structural models directly follow the physical arrangement of information processing blocks in the brain and try to exploit the advantages of such an arrangement. These models usually result in structures that are made up of units of processing connected with each other for information exchange. They are generally grouped as connectionist models.

One such connectionist model is the artificial neural network model. The building units of these models are an artificial implementation of the neurons themselves. However there have historically been several models of ANNs with respect to operational principles and connection topologies.

3.2 Artificial neural networks

Artificial neurons are a crude approximation of their biological counterparts. They consist of an information processing module with weighted connections to other neurons. The weighted connections have a value to multiply incoming signals with. This can correspond to the degree of conductivity of dendrite links for electrochemical signals in the biological neurons. It is this adjustable weight that represents the stored information in the network. Figure 1.2 below gives a view of the mathematical model of an artificial neuron.

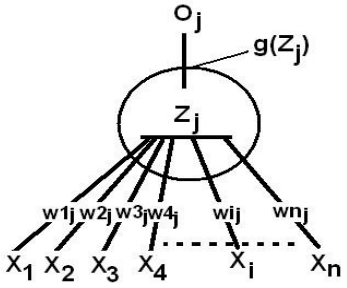


Fig. 3.2 - A mathematical model of the artificial neuron

The processing module performs summation and comparison operations on the incoming signals. The summation is simply represented as the sum of all incoming signals multiplied by their corresponding link weights, represented as Z in the equations below. However the comparison function can take the form of either a step function or or a sigmoid activation function. Each is represented below:

$$Oj=g(Z_j) = 1 \text{ if } Z_j \geq 0, \text{ and } O_j= g(Z_j) = 0 \text{ if } Z_j<0$$

Eqn. 3.1 - The step activation function

$$Oj=g(Z_j)= 1/(1 + e^{-Z_j})$$

Eqn. 3.2 - The sigmoid activation function

Numerous models of ANNs have been developed in the last decades with structural inspiration from arrangements in the biological neurons. The perceptron model (eg. the Multi Layer Perceptron (MLP)), Adaptive Resonance Theory (ART), Radial Basis Function (RBF) and the Reduced Coulomb Energy (RCE) are a few of the widely known models. Functionally, however, these models differed from each other. With respect to sturucture, they can be divided into two main camps, namely Feedforward and Feedback or Recurrent networks. Feedback type networks have connections from neurons in a layer to neurons in a previous layer and hence their operations have more dynamic characteristics. Recurrent of feedback type neural networks have complex behavior and are important to researchers. However, it is the feedforward networks that have so far proved useful in applications. Feedforward networks have input signals relayed from input layer neurons to output layer neurons through a number of hidden layer neurons without being directed back to the previous layers.

The training scheme of neural networks by itself has two principal approaches, supervised and unsupervised. Supervised learning involves presenting the network with example training set which is a typical representative of the actual domain the network is supposed to work on. The aim of this process is to adjust the weight of the links between the neurons after each example so as to minimize the error in the final output. Unsupervised learning in contrast does not involve examples of the actual set and the network is supposed to make classifications of the input data using relative similarity between the data. Unsupervised learning is also a very interesting scheme for researchers as it will be essential in explaining the characteristics of biological neural networks. However, for practical application purposes the most widely implemented scheme is supervised learning.

3.2.1 Multi Layer Perceptron

The multi-layer perceptron (MLP) is a type of feedforward neural network that employs supervised learning. Its structure is composed of layers of perceptron type neurons, usually with fully connected topology, meaning every neuron is connected to every other neuron in the next layer.

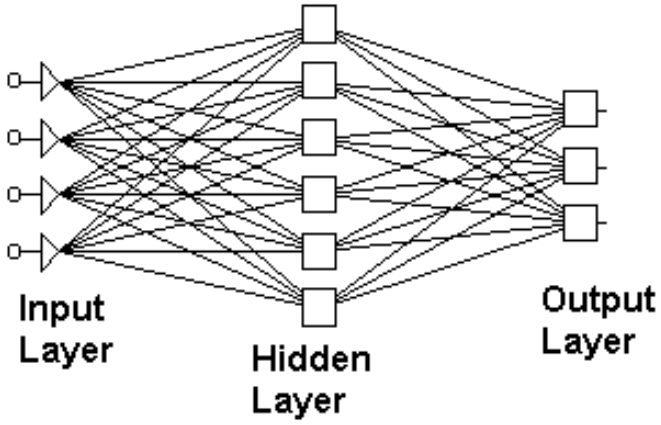


Fig. 3.3 – Structure of the MLP

The delta learning rule is commonly applied in this network, where adjusting the link weights is achieved by propagating the error in the output layer back to the previous layers and computing new weight values for all the neurons in each layer. Donald hebb was the first to propose such a learning law. the sigmoid function is the most widely used activation function in the perceptron type neuron.

The connection weight $w_{ij}(t)$, after observing the t^{th} training example, is given by equation 1.3 as follows.

$$w_{ij}(t) = w_{ij}(t-1) + \triangle w_{ij}(t)$$

where: $\triangle w_{ij}(t)$ is the weight adjustment after the t^{th} example

Eq. 3.3 – Weight adjustment of links

The weight adjustment using the back propagation learning law is given by equation 3.4 as follows:

$$\Delta w_{ij} = \alpha \delta_j x_i$$

Where:

$\delta_j = (D_j - O_j) O_j (1 - O_j)$ (For output layer neurons)

$\delta_j = \sum_k (\delta_k w_{kj}) O_j (1 - O_j)$ (For Hidden layer neurons)

k indexes the next layer.

$D_j \rightarrow$ Desired output

$O_j \rightarrow$ Actual output

$\Delta w_{ij} \rightarrow$ The weight adjustment to be added to w_{ij}

$\alpha \rightarrow$ Learning rate

$x_i \rightarrow$ the i^{th} input to the neuron

Sigmoid activation function of $O_j = \frac{1}{\left(1 + e^{-Z_j}\right)}$

Eq.3.4 - Error back propagation to adjust weights

The MLP employing the delta learning rule is known for its advantages in its superior approximation of untrained data and considerable amount of data it can store and deal with. This has led to its use in numerous industrial and academic applications. Common applications for it have been character and speech recognition, image analysis, banking, medical diagnosis and traffic analysis. These advantages and proven applicability have also led for the model being chosen for this work. The detailed equations of the delta learning rule can be referred from Appendix E.

3.3 Optimizing MLP networks with genetic algorithms

MLP networks are structures with topological as well as operational parameters. In regard to topology, the input vector length, the output vector length, the number of hidden layers and the number of neurons in each of these hidden layers should be defined. Operational parameters commonly refer to the learning rate and the sigmoid slope.

These parameters are known to greatly influence the training convergence of the network. The optimal values of these parameters depend on the application environment the network is meant to work in. Some sets of these values lead to a lengthy convergence cycle while others might lead to local minima, where the final set of weights found do not offer a universal minimization of error, but rather a minimal error relative to the next few nearby iterations.

With lack of a widely accepted methodology to optimize these parameters for each particular application scenario, trial and error based approach is usually employed. A random set of initial values are set for these parameters and after observing the results of training and testing, values of these parameters are altered, usually one parameter at a time. If the results are worse, the parameter is altered in the opposite direction and tried again. Thus the alteration goes in the direction of improvement. Considering the possible number of combination of these parameters this approach is clearly tedious and even not feasible for applications that demand large topology networks.

One possibility of automating this process is using evolutionary mechanism. Artificial evolution or genetic algorithms are inspired by the natural process of evolution where the genetic makeup of members of a population are altered randomly and this alteration may lead some not to survive while resulting in some to be better competitors. These better members are also in the position to breed more widely than the rest. In such a way after consecutive generations a better fit population is achieved.

These same principles are applied artificially in optimizing mathematical models. A set of parameters of the model that are known to be critical for its performance are chosen and considered as the ‘genes’ of the model. A set of instances of the model are initialized with random values for these genes to make up the first ‘generation’. After observing the performance of the members, if the results do not meet requirements, the members are

subjected to the different evolutionary operators such as cloning, breeding (crossover) and mutation (random alteration).

Crossover: the parameters of two members are exchanged with each other. It corresponds to a form of breeding between chromosomes of two individuals in biological terms. This transformation helps in the discovery of combinations of parameters that optimizes performance.

Cloning: this is direct copying of a member to the next generation. Usually it is for the best performer of the last generation. It serves as a control member so as to insure that the performance of the next generation does not get worse.

Mutation: this is an adjustment of a parameter by a random (but bound) value or percentage. This is done for the purpose of optimizing the parameter with the discovery of new values that provide better performance. Even though some members exhibit worsened performances due to mutated parameters, there is also a probable chance of getting better results from other members.

The genetic parameters considered when evolving the MLP neural network thus are:

1. Number of layers of neurons
2. Number of neurons in each layer including the input, output and the hidden layers
3. Learning rate
4. Sigmoid slope value

Formatted: Bullets and Numbering

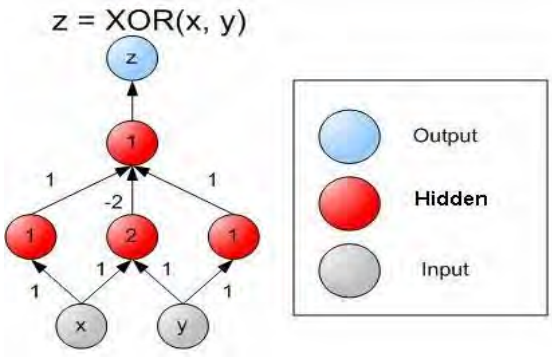


Fig. 3.4 – a simplified MLP neural network

Among these variables, the numbers of neurons in input and output layers are usually determined by the requirements of the application. The rest of the parameters can be subjected to any optimizing operation to reach at desired performance of the network.

Evolving the MLP neural network follows the same procedure as any other evolution. A number of neural network members are initialized for the first generation and a fitness function in terms of accuracy rate is set. As the purpose of neural networks is principally to learn and store patterns, correctness in recognition during the testing phase is the applicable fitness function. For this purpose a set of input/output pairs can be prepared. For neural networks the most important aspects of the data world to be modeled are the size of the input set (the number of input-output pattern pairs) and the length of the input output vectors. Both these factors require adjustment of the topological as well as the learning properties of the network as they directly correspond to the amount of data that are to be stored as weight values.

With the given sizes for the input and output pattern vectors, the number of neurons in the input and output layers of the network is fixed. Each member of the first population is assigned random values of the parameters and trained with the prepared input/output set. The fitness function evaluation comes in the testing phase as the members with the most correct scores of output for the sets of inputs are deemed best fit. The parameters of these 'best' networks are then transferred to the next generation with any combination of the evolutionary transformations mentioned. The process can be terminated after a predefined number of generations or can go on until a desired performance rate is achieved.

Even if recognition performance is the primary fitness function as correctness is the most desired behavior from the network, training time is also crucial in the performance of the network. A network should be able to be trained input/output pairs with as little cycles as possible. Thus when the recognition rates of certain members of a generation is below a desired threshold, training cycle (or epoch) can serve as a secondary fitness function.

Given the functional requirements of this research work in terms of extent of network topology, an automated means of evolving the MLP network was desired. Thus a neural network evolver tool was developed in order to assist in the discovery of the optimal network. The description of the tool and the optimization runs that were performed are discussed in the design part of the paper.

Chapter - 4

Conceptual analysis

4.1 Scope of the model

Platform and resources: HTM is inherently a parallel running algorithm. Each path along the hierarchical tree is an independently operating line of cognition which is best realizable by a separate process running on each node. This suitability for parallel operation emerges from its close depiction of the biological system. At the bottom of the hierarchy are leaf nodes that operate by acquiring their own input and processing consecutive sequences of inputs for spatial and temporal analysis to produce a belief vector output. Parent nodes in turn process concatenated inputs from a given set of child nodes and run a similar algorithm to feed nodes at even higher levels.

Given the unavailability of a parallel operation machine architecture at hand, the model was implemented on a purely software based emulation basis. Hence the algorithms rely on multithreading whenever possible. This use of multithreading imposes a limit on the amount of computation load the model generates and thus the extent of the model configuration has to be limited for analysis purposes in a feasible amount of time and resources. For the implementation purposes, the available system‘s configuration is as shown in table 3.1.

Resource	Value
Processor	Intel ® Pentium m ™, single core
Cache (l2) [KB]	2048
Speed [MHz]	2130
Physical memory [MB]	1024
Bus speed (FSB) [MHz]	533
Operating system	Microsoft ® windows ® XP ™, service pack 2
Framework	Microsoft .net 1.1

Table. 4.1 - Hardware and software platforms used

Data structures and optimization: various types of data structures, some of which with a considerable resource requirement, are required to let the HTM as well as neural network algorithms work properly and efficiently. HTM nodes require a set of large vectors to store pattern and belief data as well as a matrix for storage and maintenance of temporal data. Neural network data structures, embedded in each node, require their own sets of vectors and other storage variables. For the sake of speed and efficiency, system managed classes of variables were used for most of these data structures.

Test domain & expected efficiency: due to the limited topological configuration of the model the domain and complexity of the test data has also been limited. Hence the system will be able to function in an acceptable manner of efficiency while a good level of accuracy is expected from it due to the simplicity of the input domain. While these simple domains might not represent real world data they will serve as fair analysis approaches to the model.

4.2 Test application considerations

Formatted: Bullets and Numbering

Application selection: selecting an application scenario for testing the model required consideration of several factors. The selected test application is an image recognition engine with a simplified input domain of line drawing images (included in appendix a). Some of the factors for selecting the application scenario are discussed below.

4.2.1 Extent of the model's configuration

Formatted: Bullets and Numbering

The extent that the model can be configured to is limited as discussed previously in section 2.1.1. This limitation also restricts the type and amount of data set the model uses to produce output in a feasible amount of time and resource usage.

HTMs, in theory, are capable of dealing with any type of input data provided it is represented in any suitable format. However, to process complex data domain such as an audio stream, the required configuration of the model will easily exceed resources afforded by an emulation platform. Other real world data such as video frames, face recognition and data traffic management also lie in the same level of complexity. While providing an adequate analysis of

the model with a simple test domain of a line drawing visual recognition application, the hierarchical nature of data in these complex scenarios guarantees that the model, with a scaled up configuration, is valid for them.

4.2.2 Available comparison models

Formatted: Bullets and Numbering

Evaluating a new model needs not only generic performance but also comparison analysis against a previous version or a similar model. This will help in assessing the advantages gained by the enhancements or the extent of performance degradation due to the modifications introduced.

The proposed model introduces neural network data structures in HTM nodes for various reasons mentioned in section 1.3.3, one of which is better migration capability to future hardware based platforms. The full advantages of such modifications are, however, not expected to be realized in an emulated implementation due to the additional training time requirements for the neural network in learning phase.

Though the performance comparisons with respect to computational delay and resource usage may be ignored, the model however hast to be cross-compared with models prior-to modification with respect to recognition rate. The current implementation of HTMs (non neural network) is being tested by Numenta on a line drawing image recognition application. Hence testing the model on a similar or even the same data set (available for download at the company's website) provides an intuitive picture of its advantages.

4.2.3 Usefulness as a future research foundation

Formatted: Bullets and Numbering

An important factor in basing the analysis of the model on a certain application area is the trend and amount of research on that area. Current artificial cognition research is mostly based on visual applications as vision is believed to be the most complex and most valuable sense of natural cognition. This research includes work on face recognition in the area of security, number plate recognition, and robotics. Hence testing a promising new model on a popular application area will provided a useful foundation for future research endeavors.

Chapter - 5

Model design

5.1 HTM network design

The HTM cognition model, as has been explained in section 1.5, is a hierarchical collection of nodes with parent-child relationships. As an n-order tree, various aspects of its topology are considered when designing the model for specific applications or when addressing resource constraints as explained in section 4.1. The important factors that were considered for the design of the network topology are discussed in the sections to follow.

5.1.1 Dimension of the input data

The selected application area deals with a vision recognition problem. This implies that the perception level (lower most) nodes perceive input data sets that are 2-dimensionally related. This relationship is perhaps best depicted with the figure below:

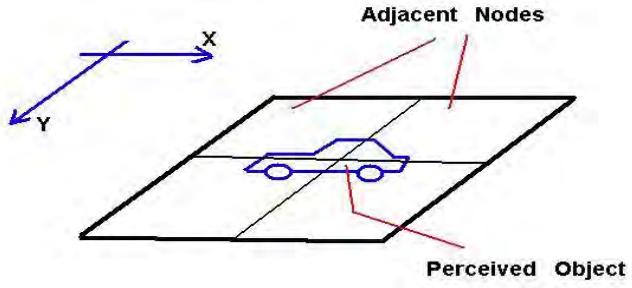


Fig. 5.1 - Input dimensionality for vision problems.

Each of the nodes perceives a portion of the object that is adjacent to other portions in both the x and y directions. This spatial 'adjacency' immediately entitles the portion of data to a spatial relationship with its both neighbors, best expressed by a two dimensional arrangement of the receptor nodes.

This arrangement is a reflection of the inherent hierarchy type of the input space rather than just a convenient or efficient way of representing the input. This is to mean that object dimensionality is observed in all levels of the hierarchy rather than the receptor level only. In an intuitive case instance, a 'line' object will be a 2-dimensionally arranged set of 'point' objects, while 2-dimensionally arranged 'line' objects will form 'corner' objects. Likewise a set of 'corner' objects arranged on a two dimensional plane view may form a 'rectangle' object.

Hence this dimensionality decision in the design of the model is reflected among all levels of the hierarchy. This results in a parent-child relationship scheme of 1 to 4 parent-child relationship as shown in the sub tree diagram 5.2 below:

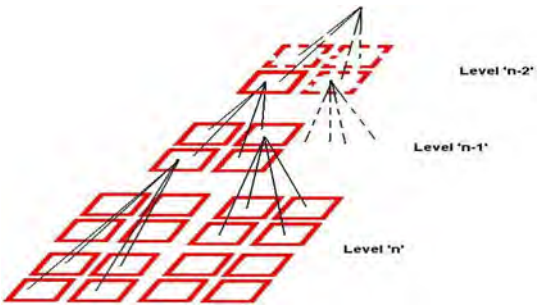


Fig. 5.2 - 2-dimensional sub-tree view.

5.1.2 Complexity of the modeled object.

Formatted: Bullets and Numbering

Modeled objects can vary in complexity from simple aggregation of a few features to extremely compounded ones. Complexity is measured in both the sheer number of individual features as well as the number of existing consistent temporal relationship between two or more of these features.

An object that has features which have equal levels temporal relationship with every other feature does not exhibit a hierarchical aggregation of causes. Such an object, however, exists in the theoretical realm only. Real world objects exhibit a distinct temporal relationship among their features. An example from the selected test application, as depicted in fig. 5.3, will prove useful. In the figure lower level features such as corners and straight lines, as perceived by leaf nodes, are aggregated in a number of orders to form middle layer features. For the selected set of simple line drawing images two levels of middle layer have been observed.

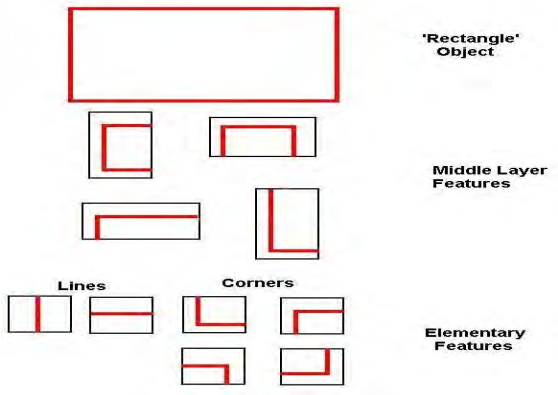


Fig. 5.3 - complexity analysis for the research scope.

In a statistical analysis of the temporal relationship of the features, it can be revealed that some of the features occur more frequently together than others. In the above example, a lower-right corner is not likely to happen temporally adjacent to an upper-left corner. The same will be true for upper-right and lower-left corners. This provides a usable distinct clustering of features.

In more complicated objects, features may not be clustered as distinctly. However, there still will be a difference in the degree of relationship strengths among them and stronger relationships take precedence when clustering. Moreover, has been discussed in section 2.1.2-ii (node algorithms), the temporal clustering algorithm runs a recursive set of steps to group these features which results in a logical collection of features per cluster.

Even though not as simple as the above example, the selected test application also deals with simple shape objects of similar feature sets. These sets of features run in limited levels of aggregation namely:

- Points
- Lines & corners
- Sections (aggregations of lines and corners)
- Object sides (aggregation of sections)
- Object shapes

Point objects are elementary data types for this cognition activity and are not considered a feature by themselves. Thus, corresponding directly to this aggregation, a 4-level tree configuration was selected for the design.

5.1.3 Size of perceived objects

Formatted: Bullets and Numbering

A given object has to lie within the perception range of an HTM tree in order to be recognized as a single distinct object. The reason that lower level nodes are able to filter out features of their level is that the sub-tree below them covers only a portion of the object perceived. Likewise, the upper most nodes are expected to recognize the whole object among a collection of such objects and thus needs to have the full spatial extent of the object within its reach.

The utilized collections of objects in this visual test scenario are 32x32 pixels wide. This is dictated by two primary factors namely:

1. Efficient testing performance
2. Comparison models

The smaller the given image size, the more speedy and resource efficient will be the test run. Also, test input data collection should be similar to the one used by comparison models must be used. In this case, the selected comparison model, the Numenta HTM platform, also used 32x32 sized image sets. Thus, the comparison results are believed to be based on level grounds.

The previous analysis (5.1.2) of input domain complexity yielded a tree structure of 4 levels while the preceding analysis (5.1.1) on input space dimensionality has resulted in a parent-child relationship of 1 to 4. With regular quadruple tree architecture at hand it is easy to calculate the number of leaf nodes of the tree as shown in Equation 5.1;

$$L = C^{(N-1)}$$

where: **l → number of leaf nodes**
c → number of child nodes per parent node
n → total number of levels of the tree

Eqn. 5.1 - determination of leaf nodes.

This gives: $L = 4^{(4-1)} = 64$ which corresponds to an 8x8 matrix of receptor nodes.

Each node covers a 4x4 pixel area of the given image. Thus the overall lower level receptor field of the tree covers 4x8=32 pixels wide and a 4x8=32 pixels long area of the image which is adequate for the intended purpose.

The final tree design configuration looks like the structure in the fig 5.4 below;

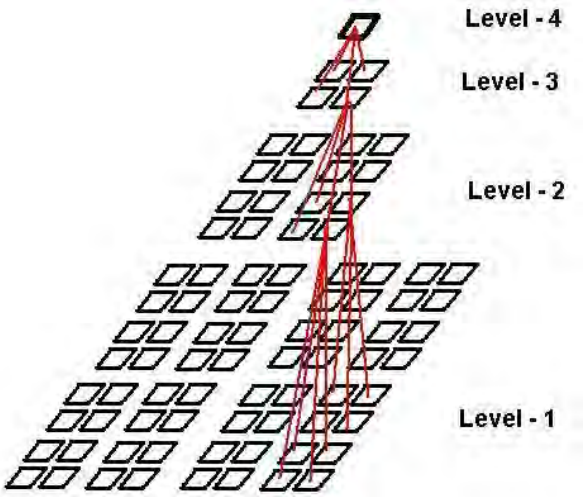


Fig. 5.4 - The final HTM network.

The values of the operational parameters for the HTM network in the final model are as follows:

Parameter	Value
Tree Height	4
Children-per-Node	4
Total number of Leaf Nodes	64
Node input Length	16
Node Output Length	4
Spatial Discrimination Criterion	0.01
Maximum Temporal Cluster Size	24
Temporal Neighborhood size	3
Temporal History Size	4

Table 5.1 – HTM network parameters of the model.

5.2 Neural network data structure

5.2.1 ANN model represented operations

Several structural parts of the HTM node manifest a logical suitability for utilization of artificial neural networks. Note that virtually any of the numerous operations in the node can (and as the researcher believes, should) be replaced with ANN data structures. In the limited scope of this work however only a single major operation was chosen to be implemented in ANNs. There were two principal reasons for this design decision:

1. **Limiting the scope of the research**: The work introduces the use of artificial neural network systems in a plausible model of the neocortex for the first time (as of date of compilation of this report). This will help for the performance study of neural network data structures in cognition operations as well as the study of the dynamics of the whole model as based on ANNs.
2. **Resource limitations**: software based emulation of neural network data structures have been known to require a vast amount of memory space and processor time. This arises from the parallel computing characteristic inherent in their structure. Each node of the network hosts a multitude of links to other nodes along with several variables. The more complex the modeled functionality is the more number of layers and neurons per layer are required per network. Moreover, the training operation in most networks requires iterative process over the layers that increase in number as the size of the input domain increases. Hence, within the limitation range of resources available it was mandatory to restrain the usage of these data structures to a certain operation.

The belief inference engine in the spatial processor performs a critical operation in the functionality of the node. It is responsible for generating a Bayesian belief vector to feed the temporal processor. Its position in the chain of operations is best demonstrated in figure 5.5 below:

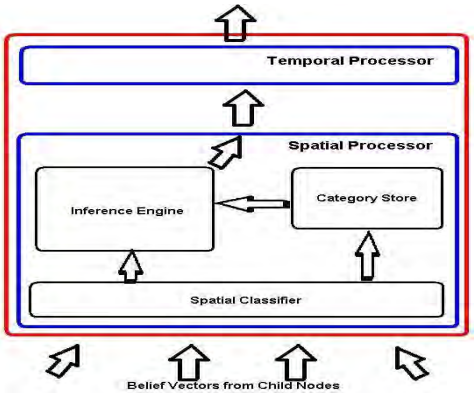


Fig. 5.5 - spatial processor in the HTM node

The inference engine takes the current input from the spatial classifier and compares it against all the stored categories to generate belief outputs corresponding to each. These belief outputs are then analyzed to decide if a reasonably accurate inference can be made on the perception. If the best belief output does not stand out from the rest and there is an almost equal chance of the object likely being more than one, the engine categorizes it as 'unknown'. This calls for a neural network structure as discussed in section 1.8. The neural network model selected for this purpose was the MLP model with the delta learning rule. The reasons for this selection was

1. ability to be configured to take input vectors of extended length
2. storage ability for large number of associations
3. Handle unforeseen data to a good extent

The figure (fig 5.6) below illustrates the modifications to the original HTM node using neural networks as implemented in this research work.

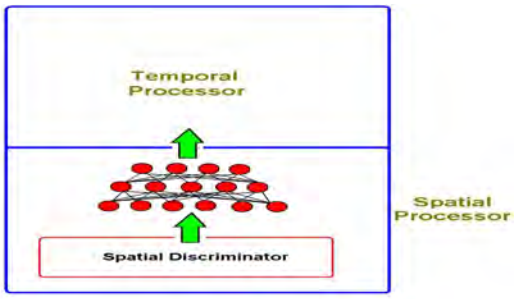


Fig. 5.6 - Neural network modification of nodes.

5.2.2 Tuning ANN parameters

Formatted: Bullets and Numbering

The MLP model depends greatly on the configured topology and network parameters for operation. As discussed in section 3.2 the genetic algorithm was used to reach at optimal values for these parameters. Topology parameters considered as 'genes' of the MLP are:

• Number of layers: the neurons in the network are arranged in layers that correspond to the various abstraction levels of the modeled function. The first and last layers, dubbed input and output layers respectively, are mandatory whereas a number of optional midway or hidden layers can exist between them. It is generally believed that the number of layers matches the complexity of the modeled function but is, in practice, set through experimental trials.

• Neurons per each layer: each layer of the network is composed of neurons that can vary in number. The input and output layers inherit their neuron sizes from the required size of the input space and desired output respectively. However, hidden layers can have any number of neurons as per the performance requirements. Though generally discovered through experimental means, the number of hidden layer neurons is believed to correspond to the amount of associations to store in the network.

• Connection type: connection type refers to the type of interconnection between neurons of the successive layers. These connections are the weighted links between neurons that are used to store 'information' in the network. Though various types of connection arrangements are possible a fully connected configuration, where every neuron in a layer has a link to every neuron in the adjacent layer, is widely utilized.

To add to the difficulty, most of these parameters do not have a clear metric in relation to the modeled function or to the input/output data set size. The most frequently used approach to reach at optimal values for these parameters is trial and error mechanism. A network is initialized at a random parameter scenario and the effects of increasing or decreasing any one of the values is examined and adjusted as necessary until the optimal parameter set is found. Usually, there are some guiding rules as to the range of values when initializing network

topology; the more complex the function, the more number of hidden layers and the more the size of the input domain, the more the number of neurons in the hidden layer will be required. However, apart from these subjective guidelines, there is no clear-cut relation in calculating the exact values. Moreover, for other non-topology related parameters, such as learning rate and sigmoid slope, there is no universal guideline at all.

The research work on this model requires a fairly performing MLP network with huge topology and large input/output data set. Clearly, a random trial and error approach will not be feasible. The sizes of the input and output vectors (vector length) were directly referred from the non-neural network implementation by Numenta and were observed to be at maximum values of

input vector length = 512

output vector length = 128

input set size = 128

The researcher has developed a neural network optimizer tool based on evolutionary mechanisms for the purpose of selecting the optimal parameters for the MLP model rather than by trial and error approach.

In this work, the number of input and output nodes are fixed to 512 and 128 respectively. The number of hidden layers is also fixed to 1. while it is conceptually possible to vary the number of hidden layers and reach at an optimal value, in practice having more than 1 hidden layer in such a large topology has been observed to exponentially increase the computational cost.

Thus the parameters that are tried to optimize in this work are:

1. the number of hidden layer nodes
2. the learning rate
3. the sigmoid slope

The tool is provided with the input/output data vector length and the input domain size. A random pair of 128 input / desired output pair are used.

The tool used a randomly initialized population of size 10 with 20 epoch MLP iteration on each network in every generation with a maximum of 30 generations. It will initialize the first population with a random set of network parameter values (number of hidden layer nodes, sigmoid slope and learning rate) and use the mentioned evolutionary techniques (cloning, mutation and crossover) in various combinations. The fitness test is the average error of each network. A view of the optimizer tool is provided in the figure 5.7 below;

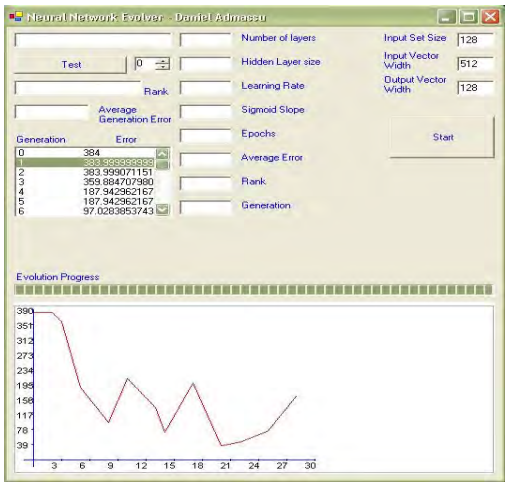


Fig. 5.7 - the neural network optimizer tool.

The evolutionary flow that the tool follows is represented as figure 5.8 below.

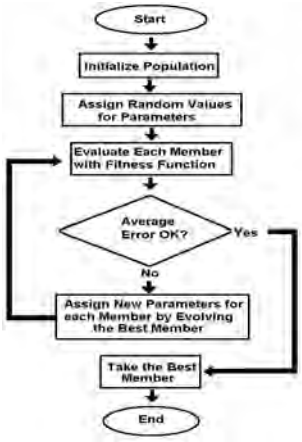


Fig. 5.8 – Evolutionary steps employed in the tool.

The evolution process for the mentioned values of input/output sizes and data set size took a period of several hours per each run. A few sample runs with varied values of mutation percentage are shown in the figures 5.9 below;

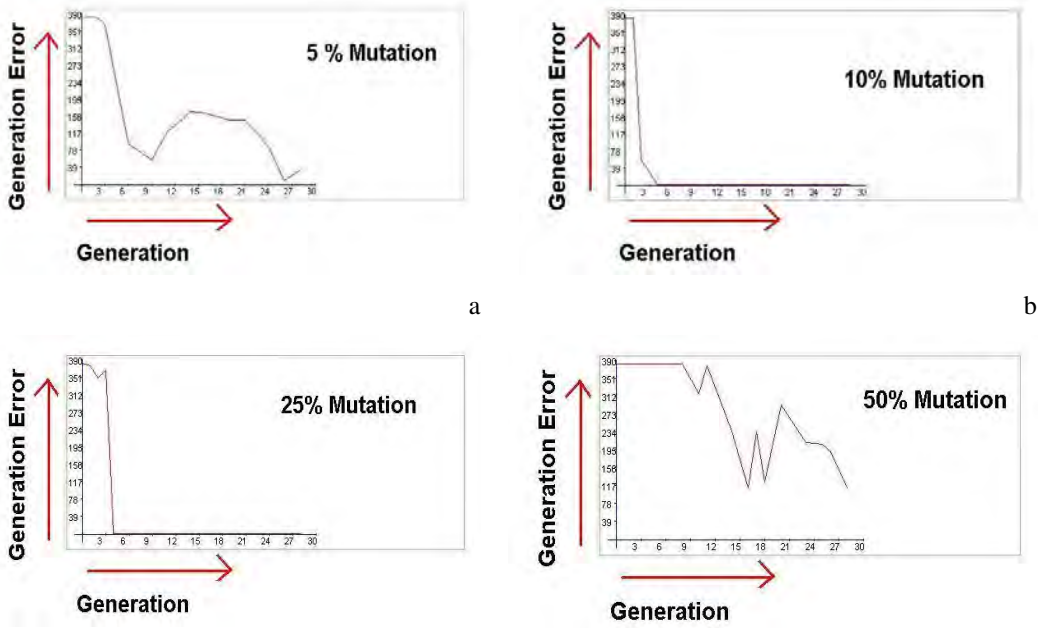


Fig. 5.9 (a-d) - evolution results with different mutations.

The above charts clearly show the advantages of the technique when it comes to a quick convergence to an optimal value set. Both the number of generations and population size have effects in the convergence process. The stopping criterion is set to an error of 0.0001.

The figures also show that optimal value of mutation percentage should be employed to reach at a fine tuned network at the last generation. Too low and too high values of mutation tend to introduce instability and non-optimal network values (fig 5.9 a and fig 5.9 d). The mutation rate values between 10% and 25% were observed to suffice for the process and 10% was used for this work. The other operations of evolution, namely crossover and cloning were used as discussed in section 3.2 and were not varied through generations. 10% of the next generation had been a clone of the best selected individual and 10% were crossovers of the best and the next best in the previous generation. The rest were clones of the best and crossovers of the best and the second best in various combinations of genes with mutation applied to them afterwards. Through experimental analysis, it has been observed that only mutation has a

significant impact in the process. The evolutionary operations performed upon creating a new generation of networks are as follows.

Step-1. 10 instances of MLP networks are created and initialized with random values to the parameters mentioned. Also a random set of 128 input output pairs (with input length 512 and output length 128) was initialized as a training set.

Step-2. Each network is trained for a maximum of 20 epochs on the training set with a stopping criterion of MSE=0.0001.

Step-3. Each network is evaluated for error using the training set. Then the networks are ordered according to their performance error. (Starting with the best performing network)

Step-4. The average error of the generation is evaluated and compared to a preset value of 0.0001. If the error is less than this value the evolution is stopped and the best member is chosen.

Step-5. If the generation does not meet the error requirements, then evolutionary operators are used on the members to create the next generation as shown in figure 5.10 below.

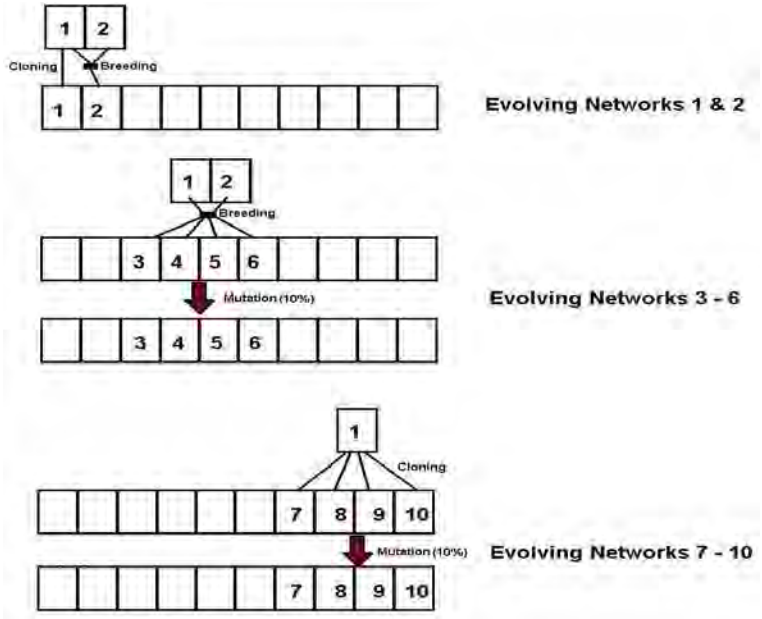


Fig. 5.10 – Evolution of the next generation

Step-6. Return to step 2 and continue until the conditions in step-4 are met or the 30th generation is reached.

From these several evolutionary runs an optimal network topology and other network parameter values were selected for the purpose of the network in the model. In this particular case the generations came to an optimal parameter set at the 5th generation. Hence, the final model is depicted as shown in figure 5.11 below;

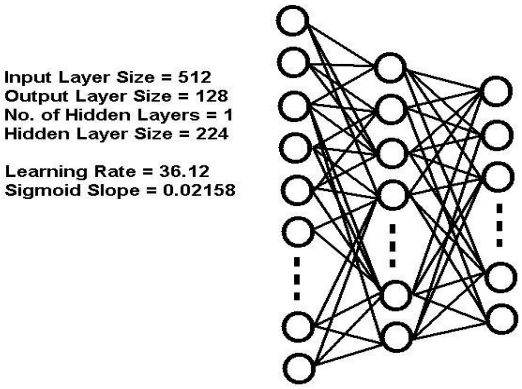


Fig. 5.11 - final optimized neural network structure.

5.3 Input processing and representation

An aspect in the design of the cognition model is input pre-processing and representation in the form suitable for the network from the raw data that is generated by and is specific to the addressed problem or application type.

A direct analogy to the natural system is the concept of 'senses' in the nervous system. Even though different parts of the neocortex deal with different functionalities of the host, their structure and elementary composition remains essentially similar. Hence neurobiologists have been able to conclude that the neocortex is a highly adaptable, all-purpose system that is able to fit into any task scenario presented to it. This has also been indirectly confirmed by the fact that parts of cortical columns quickly transform their organizations in order to take over tasks

of adjacent columns that had been damaged or infected. Hence the neocortex resembles a multi-purpose processing core that operates on uniform principles for every task. Thus it is clear that the responsibility of dealing with the various types of data (eg. Visual, auditory, taste) rests upon the sensual organs and the associated pre-processing parts of the brain.
Similar enough, an artificial cognitron system should have a modular organization that separates the input pre-processing logic from the main cognition engine. This has dual advantages:

1. The cognition engine will be reusable for other types and formats of data other than the original application either alternatively or simultaneously running the various data processing functions within their own data domains. This also provides the opportunity to optimize the network for different scenarios and performance demands.
2. The engine will also be suitable for implementing on specialized, high performance, hardware leaving data generation and problem deciphering to low-end, everyday platforms which is more and more the trend in super-computing architectures.

The chosen test application data domain happens to be relatively simple in that it is intentionally organized and contains limited representation depth. This is to mean that the visual objects are in the form of geometrical diagrams of pictures and alphabets whose unit data is binary pixel states. In real world applications these states need to be represented by a deeper bit length to deal with different colors and grayscales.
As discussed earlier, the network dimensionality is 2, which also applies to the perception (lower-most) layer. Hence object data need to be processed and supplied to the right adjacent node in the cluster of child nodes under a parent. Like wise parents need to receive data that is in the proper order in relation to their peer nodes. This ensures the desired clustering of features and objects in the current dimensional organization of the application data. The figure below demonstrates the scheme applied to the chosen vision problem.

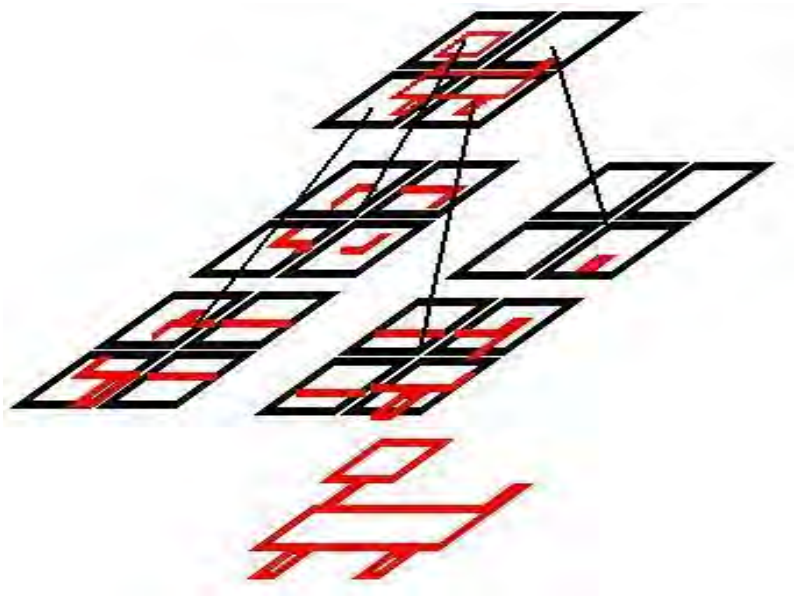


Fig. 5.12 - Input data processing scheme by adjacent nodes.

5.4 Scenario formulation

The neural network based HTM model learns and infers causes in a similar fashion to the non ANN based original model. As with the previous model, the data acquisition is through an exhaustive sweep (orthogonal scan) of the entire visual field of the given image. This is in contrast to a random direction scan option which a natural learning operation is likely to follow.

An orthogonal scan works by advancing from one pixel to the next in horizontal or vertical direction until the edge of the image is reached. Upon the encounter of an edge, the scan increments one step in the other dimension and returns to the opposite edge in the same manner. This is repeated until the whole image area is swept. Such an algorithm results in an exhaustive acquisition of data which does not need further scans.

In contrast the biological cognitrons work in random fashion especially upon first time encounter of an object is made and no prior supervised guidance is available as to the direction of scan. This results in a non optimal learning convergence speed as redundant scans may be

made and some areas of the visual field may be missed from the scan. Usually such congnitrons are known to make up for this by supervised behavior which guides the scan direction and limits in later encounter of the same or similar object, resulting in a better and even more efficient perception process.

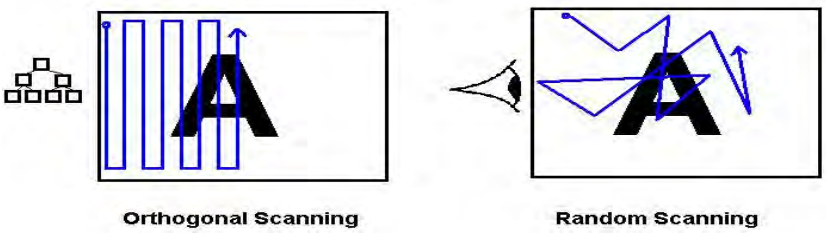


Fig. 5.13 - Orthogonal vs. Random (natural) scanning.

The default test scenario by Numenta inc. is a set of 452 pictures organized in 48 folders representing 48 different simple real life objects. However these different types of objects are not categorically defined to the model as belonging to these categories. The result is that each distinct image is learned as a distinct object even though the object represented is the same. This has led the researcher to device two testing scenarios:-

Scenario A: in this scenario the same image set with the same organization as that of the original Numenta model is used. Hence a subset of the images, totaling 100, is tested on the models.[appendix A-1]. This test helps to determine the overall storage capacity of the models. This is also a mandatory test since it is the only test run Numenta has at the moment and helps in the comparison.

Scenario B: this scenario is a selected set of images from the 48 different object folders [appendix A-2]. The categories are organized in clean, position varied, shape varied, and noisy images of various levels, each having 48 images. This test, as the researcher believes, can be a true measure of a model's cognition performance since an expected advantage of cognitrons is an invariance of judgment in the face of image distortions, noise patterns and relocations. Note that the Numenta implementation does not have an automated routine to test such sets and each image had to be tested individually.

5.5 Data generation

The learning and inference data generation is not a factor in the performance of the model and can take any mode that the designer sees fit. Various types of automated data are generated from the learning and test runs.

Scenario a type tests help determine performance in terms of overall model capacity and are run on non-categorized subsets of the supplied images. This test also, in the new implementation, is used to generate operational characteristics data of the nodes in each level of the HTM tree. This data helps visualize the clustering performance of each node as well as the time requirements of each level both in learning and inference. Theses tests are used to prove the validity of the HTM theory as well as that the resource requirements of an ANN based model lies within feasible range.
An output test is also performed in this scenario in which recognition rate is the main measure of performance. This test helps in determining the storage capacity of the model and is also the default comparison test with that of the original. It is run on varying sizes of subsets of the supplied images to investigate the tolerance of each model on increased number of input images.

Scenario b type tests generate pure output based test data that measures performance by recognition error rates. These tests, however, are done on different modifications of the original model as formulated by the researcher, namely:-

- Clean image set [original]
- Position varied set
- Shape varied set
- Noised set of noise level 1
- Noised set of noise level 2
- Noised set of noise level 3
- Noised set of noise level 4
- Noised set of noise level 5

All the 48 categories of images are prepared in the forms of the above modifications and supplied as test images after training on the original model. Again testing them on the original clean image set will be similar to scenario a tests in that it can only prove the storage capacity of the network, albeit on a smaller set. The other modified test sets will determine the true performance of the model in cognition capabilities.

Chapter - 6

Implementation

6.1 Platform and language features

Formatted: Bullets and Numbering

A given platform for a software implementation of any logic or functionality refers to the hardware and operating system components that assist in carrying out the intended operation of the software. Apart from time imposition and real time performance variations, variations in the platform have little or no effect on the logical performance or yield of the operation. However, a given platform and the set of features it offers may affect the easiness of the implementation and the overall development time.

The selected platform for the implementation is a windows ® based x86 architecture machine with an intel ® mobile family processor. The specific configuration selected is based mainly on the basis of availability rather than on any other performance options. Table 4.1 shows the system configuration used for this work.

Of particular importance is the size of the available physical memory (RAM). HTM data structures are composed of layers of identical functional units that require heavy storage space. Incorporating neural network data structures in them will only worsens the situation which quickly gets the storage requirements to the limit of the system.

The language component of the platform proves to be of more relevance to the design, implementation, and even performance of the model. The selected language environment usually in interrelated with a particular industry standard framework that may affect its usage.
C++ was used as the language of choice. The reasons for this are no different from the reasons for choosing this language set for any other software development. The primary factor is speed of execution. C++ is a collection of instructions and constructs that compile into native code more closely than any other language especially for x86 machines. A second and equally prevalent factor is its object oriented structuring feature. The implementation work needed a

hierarchical 'class-object' type organization of data structures for the network, node relationships, and even neural networks within nodes. Hence the language provided the ideal environment for implementing the model.

The development environment used is the Microsoft ® implementation of visual studio ™ 2005 tool with the .net runtime library framework. The tool offers the choice of 'managed' code compilation which simplifies the creation of software components but also happens to generate heavy object code that results in slower execution. For this reason an 'unmanaged' console implementation of the code was employed.

6.2 software architecture and organization

Formatted: Bullets and Numbering

6.2.1 Functionality

Formatted: Bullets and Numbering

As mentioned in the design sections, the implementation has two major and modular parts, the cognition engine, and the input pre-processor. The cognition engine implements the HTM network and their interaction while the pre-processor handles the test application implementation and input representation.

The cognition network of the neural network based HTM is organized in various levels of functionality. A prominent part addresses the nodes of the tree while another addresses the conceptual network and functional interactions between the nodes. Other levels within the node implementations deal with various important data structures and efficient operations on them.

Another level and an important addition to the cognition engine by the researcher is a network of neurons to implement an MLP neural network component. This component is organized in its own modular code base and instantiated in the spatial processing section of every node. The target of this implementation is to replace the computational version of the spatial inference engine in the HTM node.

The input pre-processor part is also organize into layers of functionality which include the image sensing and representation modules that correspond to 'senses' of the model, and vision test modules that take the role of the test application to 'run' the environment and supply the senses with input data.

6.2.2 Class organization

Formatted: Bullets and Numbering

The diagram (fig. 6.1) below illustrates the class organization of the implementation. Refer to appendix d for implemented code.

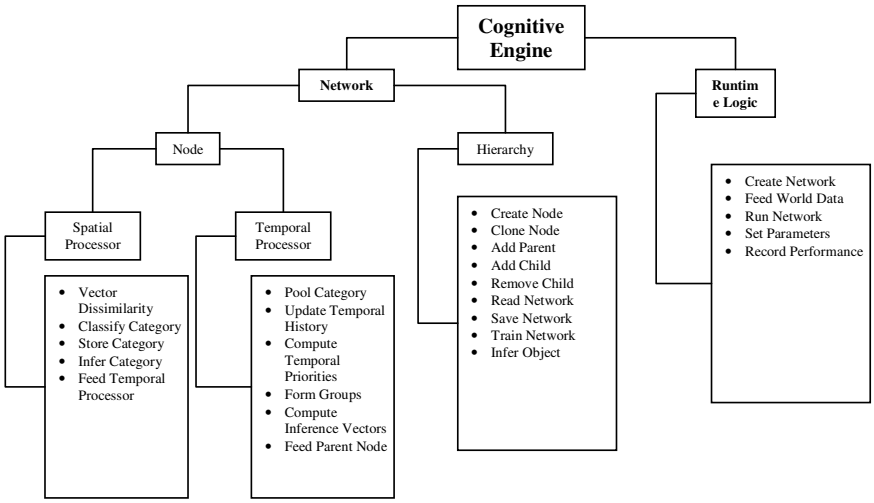


Fig. 6.1 - Cognitive engine software architecture

[class]**htmnetwork**: implements a backbone network class. This class is responsible for providing base code for various network hierarchy functionality as well as instances of node class objects. Among others it creates and removes nodes, clones trained nodes to other node positions in a level, adds a set of child nodes to another parent node or removes them, trains the network one level at a time and performs inference of the trained network. It also attends to housekeeping tasks such as reading a network file and saving a trained network to a file.

[class]**htmnode**: this class implements the basic node. These nodes are essentially identical throughout the network and possess similar data structures. Once implemented they only need to be instantiated as many times as required.

The various functionalities of the node class can be clustered into two categories, spatial processing functions, and temporal processing functions. These correspond to functionalities of the spatial processor and the temporal processor sections in each node. The spatial processing set of functions perform tasks such as discriminating vectors, classify vectors into categories, store these categories, infer incoming vectors for classifications and present output to the temporal processing set. The temporal processing set perform tasks such as pooling categories into the temporal matrix, updating temporal history and assigning priorities, clustering categories into groups, infer incoming categories for groups and output inference to parent nodes.

Other non-classed functionalities perform runtime tasks such as creating an instance of the application, initializing various parameters, presenting pre-processed data to the network, running the network for training or inference, and recording and saving inference and operational data. See fig 6.2 below for details.

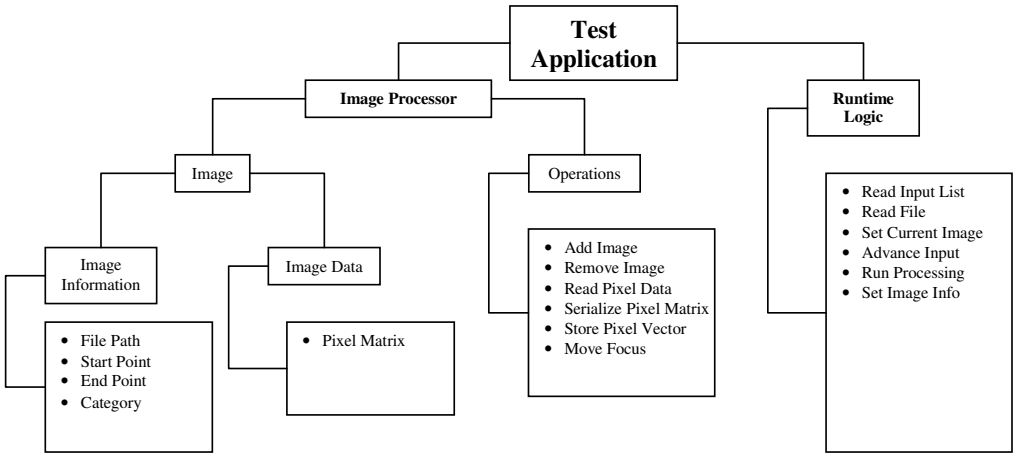


Fig. 6.2 - test application software architecture

The test application hosts a single **imagesensor** class that provides functionalities for both data management and runtime. The image object stores data and metadata for the input set while a set of operations perform processing tasks such as adding and removing images, reading pixel data, serializing them into vectors and storing them and shifting focus of perception. The runtime set of operations manage the application flow of control by reading and providing image file data, starting the image processing routines and maintaining image information metadata.

6.2.3 Coding and system limitations

Formatted: Bullets and Numbering

The development environment utilized, the Microsoft ® visual studio ™ development tool, offers a variety of language selections in a single integrated framework. The specific language selected, as is discussed in the platform selection sections, is c++.

While having clear advantages in integrated debugging, help, error tracing, and intelligent code completion capabilities, the default .net framework based option has its limitation in performance. The code generated within this framework is compiled to the corresponding .net framework runtime, much like a java code compiles to a JRE runtime. This allows the development of highly portable windows API based components like forms and controls. These components provide a highly flexible and easy to manage interface and simpler development process. In a way the compiler manages the generation and management of the code necessary to the creation and behavior of these components. Thus this default option is named ‘managed’ development.

Managed coding results in cumbersome code and resources, as much data that is not required is generated by the compiler. This in turn results in slower compilation and runtime speeds. Moreover the creation, management, and destruction of threads are not flexible enough and force the developer to settle for a non performance optimized implementation.

The project used the optional, ‘unmanaged’, environment to overcome these limitations which are considered highly important for the performance of the model. Thus an improved compilation and runtime speed performance was attained together with a lighter code size to free up more space in main memory for the model data structures. However this option as mentioned above has some inherent limitations that are listed below.

Visual design mode is not supported: the visual designing environment that is the main feature of the tool is not available in this mode. Thus, creation of window forms, dialogues, various interaction controls and input output routines need to be implemented by the developer using the necessary API calls. This is a lengthy process that is time consuming and prone to error. Thus the model was limited to a console based interaction with input output performed mainly through file read/writes.

Automated error handling is not available: generated exceptions are directly thrown to the main thread which causes the program to terminate upon such errors. This has resulted in a considerably longer development time since errors need to be traced manually throughout the code for their source. The coding does not enforce an error catching and reporting mechanism as do other languages such as java in its 'try-catch' structure requirements.

Chapter - 7

Results and analysis

There is generally a large set of possible testing routes and data generation options through varying the network parameters involved and with different modified input data sets. However, the research focused on primarily two areas of analysis to assess the characteristics and performance of the model.

The first one is analyzing the proposed HTM model to get an insight into the operational characteristics of the network as compared to the predicted HTM theoretical behavior with respect to the cognitive activities such as category clustering and feature convergence as data travels across the layers. It also examines the optimal values of some HTM parameters for the particular implementation.

The second analysis focuses mainly on yield of the model as compared to the original model and as a useful cognition alternative.

7.1 Analysis of the operational characteristics

A very useful measure of the internal cognitive behavior of the HTM model is the number of categories and features discovered and classified in the nodes of the network at successive layers. The measure is very important due to the direct mapping of the hierarchical organization of objects in the world with that of the model's operation that the HTM theory claims. The basic theory in HTM, as have been discussed in the theoretical background sections, is that as the number of distinct objects in the world is limited (the research work addresses the visual world whereas the real world is a mixture of many types) and so are the features that constitute them. Moreover these underlying features are shared among objects and only their relative arrangement in space and time differentiates one object from another.

The proof to this assumption is nothing but an examination of the natural congnitron (the brain) itself. The brain discriminates among objects by analyzing their features and their arrangement in relation to each other. Figure 7.1 below demonstrates the concept.

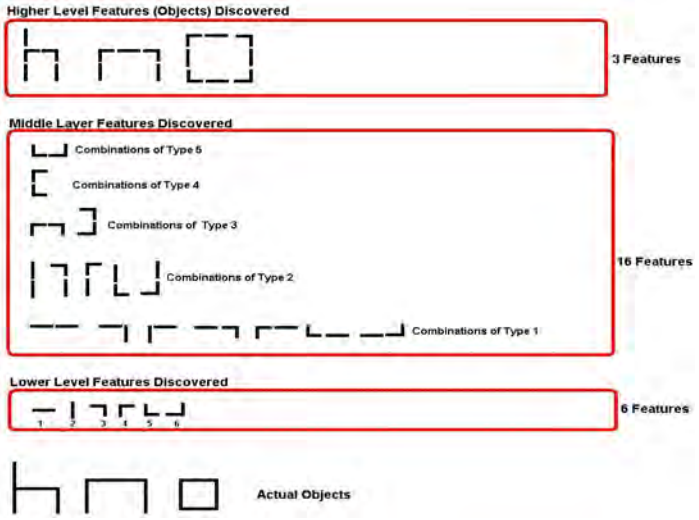


Fig. 7.1 - Variation in number of features of consecutive layers.

It is worthy to note that, though both the demonstrated example and the research samples are more or less simplistic objects, it is intuitively straightforward to extend the concept to more complicated and diversified world data types. This structural modeling of the natural cognition process is what gives HTMs advantage among the few promising models of recent times

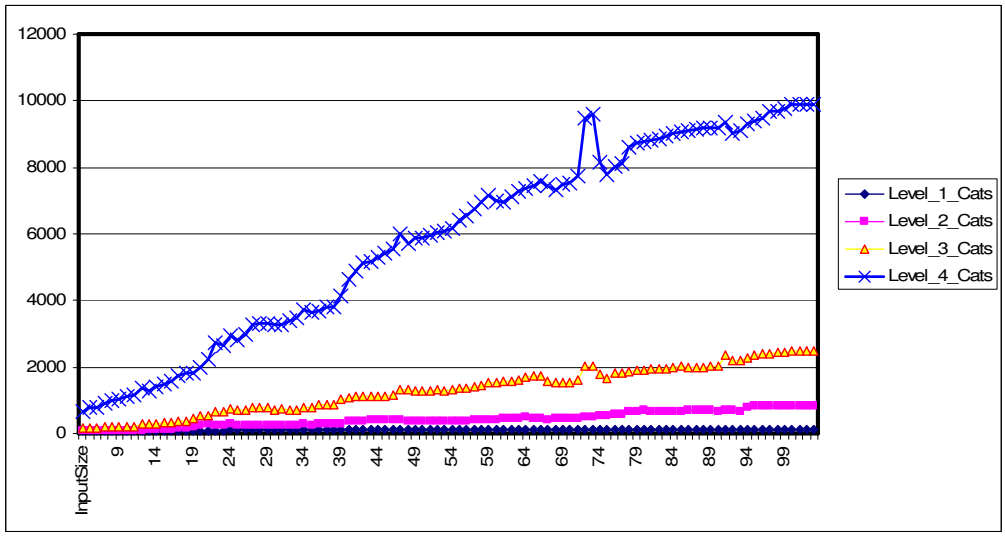
The important observation in the example is that while lower and upper most features remain limited in number, middle layer features, which are combinations of lower level ones in various orders and spatial arrangement, can be large in number. The converging number of features as one goes from these middle layer features to higher ones is a direct consequence of the limited number of distinct objects in the actual world and is yet another proof of direct mapping between the HTM model and real world data.

Thus, any other enhancement to the model, neural networks or otherwise, should follow (if not improve on) this operational characteristics. Hence, the new model was analyzed for its performance with respect to this behavior and encouraging results were obtained. The analyzed results emerging from the scenario A type test cases are shown below in plots 7.1 to 7.5.

7.1.1 Category analysis

Formatted: Bullets and Numbering

Categories are non-temporally clustered patterns in each node. Since they are not yet analyzed for temporal adjacency and not grouped accordingly their number tends to increase especially for higher level nodes as one exposes the model to more number of images. Since, as discussed in section 1.3.2, for the purpose of efficiency only one node in each layer is trained and then cloned to the others, the shown analysis is representative of all nodes in their respective layers. Plot 7.1 analyzes the variation in number of categories in the spatial processor of the nodes at each layer of the HTM tree as the number of input images is increased. The complete generated data can be referred from appendix B-1.



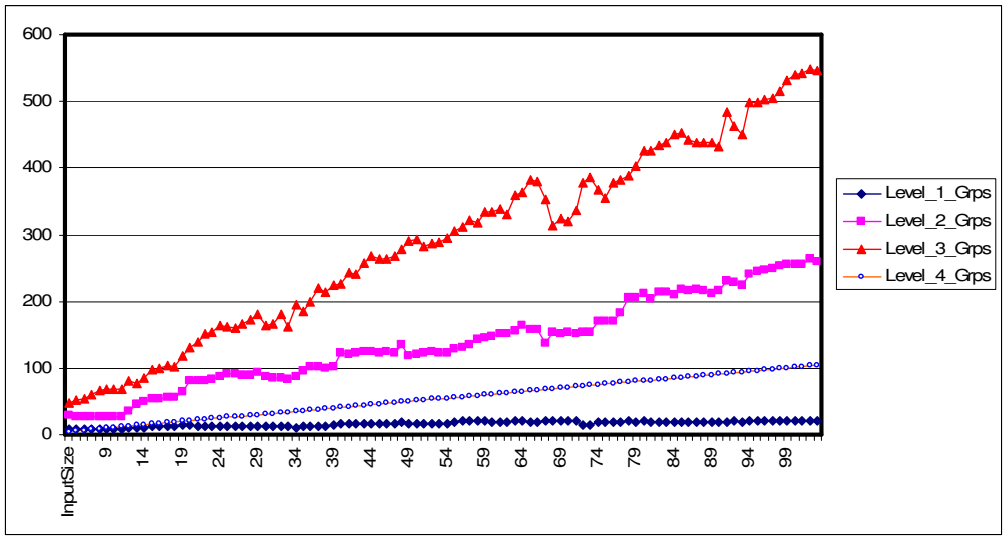
Plot 7.1 - Number of categories in each level as input size increases.

7.1.2 Group (feature) analysis

Formatted: Bullets and Numbering

Groups are simply features discovered by temporally clustering categories within the temporal processor. According to the HTM theory, lower and upper level nodes should report limited number of features and middle layer nodes are expected to discover a larger set due to the large number of possibilities in combining lower level features. The theoretical number of features at the middle layers is of a factorial order though the actual amount is usually less due to invalid types of combinations.

The graph below (plot 7.2) shows the progressive increase of features in all 4 levels of the model as more and more images are analyzed. The number of features in the first and lower levels can be seen to remain fairly constant while the two middle layer levels report an increase in their number of discovered features. The generated test data is archived in appendix B-1.



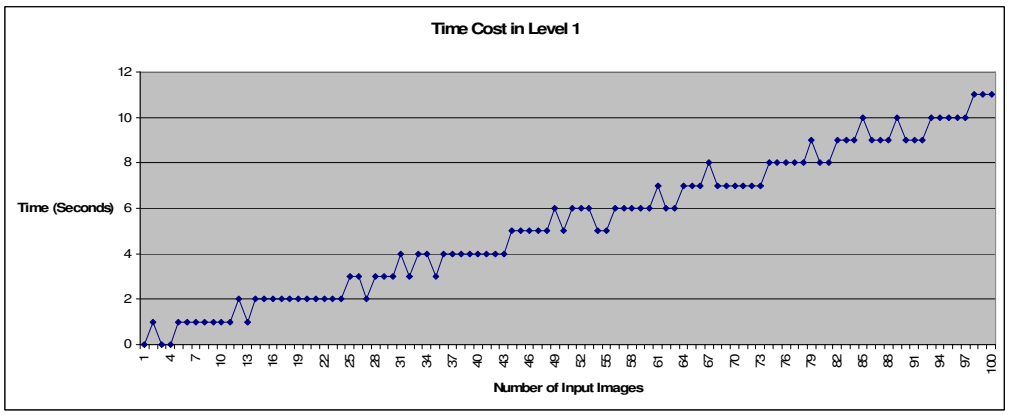
Plot 7.2 - Number of features (groups) in each level as input size increases.

7.1.3 Time cost analysis

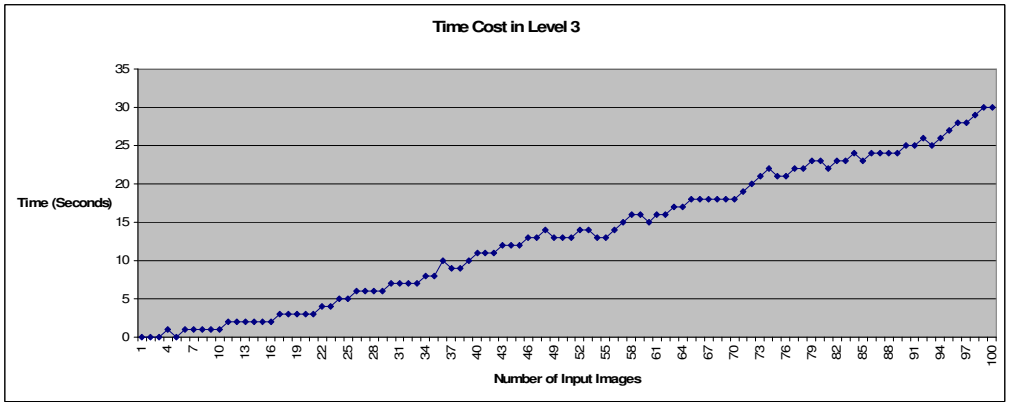
Formatted: Bullets and Numbering

Another dimension in analyzing the operational characteristics of the new model is time cost. The purpose of this cost is to further prove convergence in learning as new features appear infrequently in the lower and upper levels and increase almost linearly in time in middle layers. Hence, abrupt rise in learning time in the first and fourth levels shown below matches this expectation. Plots 7.3 to 7.5 summarize this observation, while the full test output data can be observed in appendix B-3.

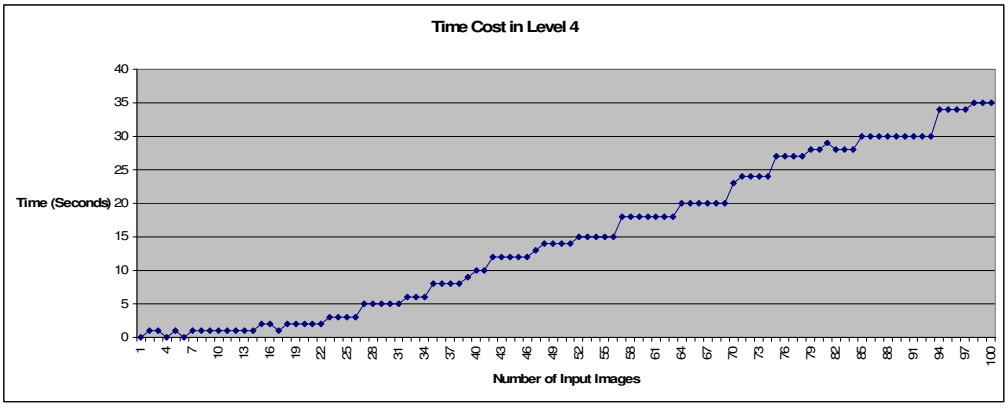
Another reason for this analysis is to confirm the feasibility of the model with respect to the time resource usage.



Plot 7.3 - Time taken during training phase at level 1



Plot 7.4 - Time taken during training phase at level 3 (representative of the middle layers)



Plot 7.5 - Time taken during training phase at level 4

7.2 Output performance analysis

Formatted: Bullets and Numbering

The most important measure of any model is yield or output of its intended purpose. For cognition models, it is undoubtedly the recognition rate or error. Although this enhanced model can be exposed to a number of variations of input images, a few set were selected. The reasons behind the selection are purposes of comparison analysis with the original model as well as standalone performance analysis as discussed in section 6.4.

The tests are:-

1. Standard capacity test (scenario A)

Theses tests are performed on the standard set of images that Numenta supplies with its model. These are non-categorized collections of images and are large in number. Thus they are useful test sets for examining the storage capacity of both networks as they are exposed to an increase in number of images.

2. Customized cognition tests (scenario B)

These are tests performed on a selected set of images. The images are selected in such a way that a representative of a collection of images is chosen and assigned to a category folder. Overall, there are 48 categories of images in the Numenta supplied images. Hence the model is not exposed to a redundant set of images at training time but rather is trained on distinct objects. In testing phase, the model is then exposed to a modification of these images in position, shape, and noise and the results are examined. The different test sets in these modified test sets are:-

a. Original [training] set
b. Position varied set
c. Shape varied set
d. Set with level 1 noise
e. Set with level 2 noise
f. Set with level 3 noise
g. Set with level 4 noise
h. Set with level 5 noise

7.2.1 Standard capacity test

Formatted: Bullets and Numbering

Plot 7.6 shows the result of the standard test as compared to that of Numenta for the same input set. Appendix A-1 has the full set of input images as supplied by Numenta.

Plot 7.6 - Storage capacity comparison with the original (Numenta) model

The non-categorized storage capacity test demonstrates that the performance of the Numenta model over a span of 100 images to be slightly better than the neural network model. The degradation of storage capacity in the new model can be attributed to the fact that the redundant set of images in the list introduces confusion in neural network data structures. The patterns that are used for training are not much different from each other. However, the network is forced to learn those patterns as belonging to different categories. Thus upon testing when the network is presented with one of the images it basically has to incline to one of the categories and make decision, which is prone to error. Along this phenomenon also comes the issue where weight vectors are overwritten while learning each pattern, there is a possibility that the earliest patterns are overwritten with the new ones and the network is not able to recognize correctly the old patterns. This issue is addressed with the pseudo-training technique [6]. Computation based systems usually do not suffer from this symptom because

their search and match algorithms can handle such slight variations with better discrimination. However, the gap in capacity performance seems to be limited and suggest that the use of a more robust neural network topology or further fine-tuning its parameters is likely to solve it. Also, the stopping criteria in training neural networks can be further optimized to suit the specific needs of the task at hand.

7.2.2 Customized cognition tests

Formatted: Bullets and Numbering

a. Test on original (training) set

This test is done on the 48 categories of images that were used to train the network without any modifications. This is similar to the capacity test, but with fewer images. It is primarily used as a control test to serve as a reference of comparison. The results for both the original and the new model are presented below in table 7.1.

Training set test for recognition error	
Model	**% error**
Original	0
Proposed	0

Table 7.1 - Error comparison in training set test.

Both models performed as expected on the non-modified test sets with 100% recognition accuracy rates.

b. Test on position varied set

This set is a variation of the original training set in that the position of the objects in the images is altered. This is meant to test if the model fails to recognize an object just because its location is changed. This can be thought as a test of the basic theory behind the HTM model in that the model does not rely on specific associations of patterns with specific locations rather only their availability and spatial relationships with adjacent patterns. The results are also presented below in table 7.2.

Position invariance test for recognition error	
Model	**% error**
Original	0
Proposed	0

Table 7.2 - Error comparison in position varied set test.

The results again, as expected, show a 100% recognition rate for both models. Failure in this test would have meant that a model still did not attain basic HTM principle operation algorithms.

c. Test on shape varied set

Shape variation is also an interesting testing scenario for the principles which HTM relies on. Recognition based on the presence and spatial adjacency of features means that the model should have tolerance to shape variations at least to a limited extent. Thus a rectangle remains a rectangle whether the sides are elongated or shortened. Similarly other shape variations in common objects observed in the world should be tolerated as is typical to the natural cognitron. The offline recognition viewer developed for this purpose is shown below.

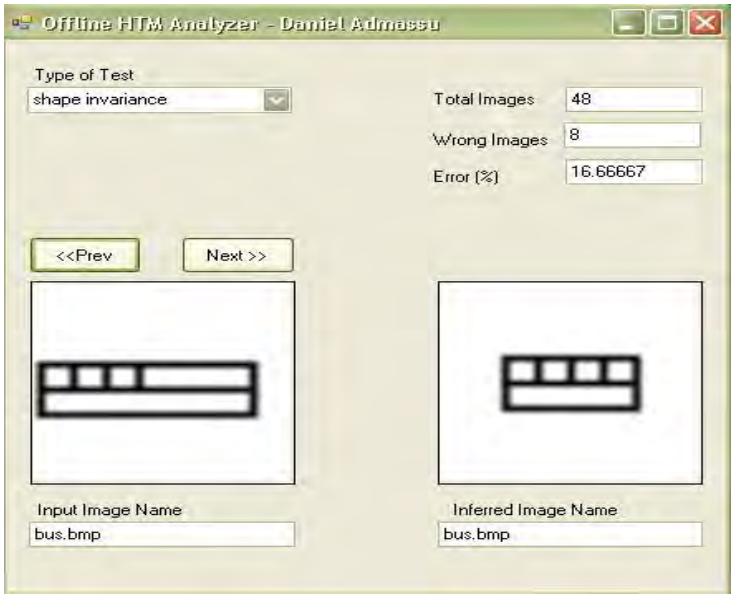


Fig. 7.2 - The offline recognition viewer with an example shape varied image

The analysis results for this test are presented below:-

Shape invariance test for recognition error	
Model	**% error**
Original	20.833
Proposed	16.667

Table 7.3 - Error comparison in shape varied set test.

The results demonstrate that the neural network based model performed slightly better than the original model on altered shapes. It is to be noted that the results are representative of a series of tests done on shape variation and with different sets of variations different results were obtained, all of which in favor of the new model.

d. noise tolerance test

Noise is among the most common variances observed in visual perception. Any model should demonstrate a substantial level of tolerance to noise if it is to be considered a viable alternative. A set of 5 noise levels were prepared for this purpose and pasted on all of the 48 categories of images to create 5, incrementally noised sets for testing. The snapshots of the recognition viewer below illustrate a selected set of views.

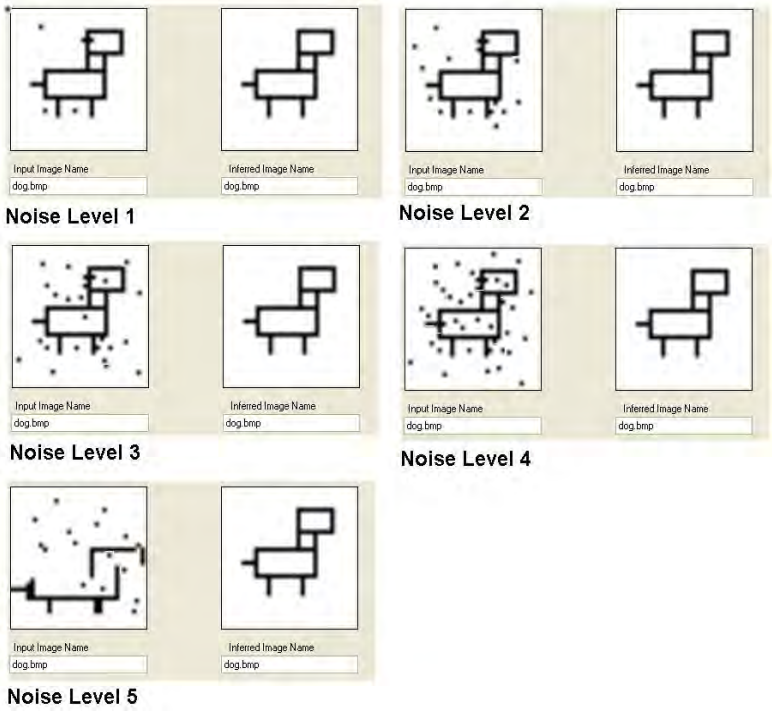


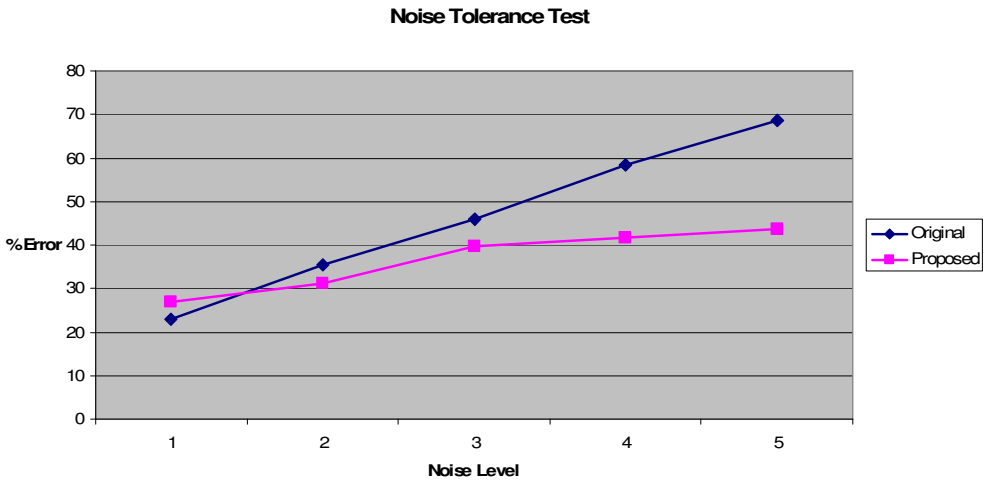
Fig. 7.3 - An example illustration of incremental noise level

It should be noted that no automation feature was supplied in the Numenta implementation of the model in linux and the altered images for all the 48 categories in each of the 5 noise level sets were tested one at a time in command line mode.

The analyzed result for these tests is shown below:-

Noise tolerance test		
Noise level	**Original**	**Proposed**
1	22.91	27.08
2	35.41	31.25
3	45.83	39.58
4	58.3	41.66
5	68.75	43.75

Table 7.4 - Noise tolerance test results



Plot 7.7 - Recognition error comparison with the original model for 5 levels of noise**.**

The plot above illustrates the performance of both models with respect to increasing amount of noise in the test images. The clear advantage of neural networks can be observed in that the 'nearest category' based operation of neural networks outperforms the non-neural network based model in noise tolerance. The error level for increasing level noise tends to stabilize for the new model while increase almost linearly for the original.

Chapter - 8

Conclusion and recommendations

8.1 Conclusions

From the results that emerged from the analysis of both the operational characteristics and output performance of the proposed model, it is evident that encouraging improvements were achieved by merging neural networks with a structured model of cognition. Although some more optimization remains to be done with regard to the stability and storage capacity of the MLP neural network, the typical cognition tests showed the following gains in performance.

- ~ 4.2% recognition rate improvement in shape variation test
- ~ 9.6% average recognition rate improvement in noise tolerance tests

Shape variation and noise discrimination are considered among the few tests that are known to require higher cognitive functionalities for which the brain is the typical reference. These results, in addition to the suitability of the proposed model for more robust implementation on parallel architecture machines, stand in favor of the new model.

Thus:

- Based on the study and analysis of the operational characteristics in the neural network based model, results typical to any theoretical model based on the HTM principles were found. This illustrates that there is indeed a solid possibility that such a hierarchical organization can be applied to neural networks and hence a truly brain-like processing power can be obtained provided the necessary platform and resources.

- Based on the output comparisons of the original and the new model under several conditions, it was found out that neural network based systems can attain improved levels of recognition accuracy in conditions that represent real world variances in input.

8.2 Recommendations

Formatted: Bullets and Numbering

The work is but a specific study of the model and enhancement with a limited scope. Future research and study should be carried both in fine-tuning of the performance and exploring similar implementations in a wide range of applications. However, a promising scenario would be to implement it in specialized hardware systems, the likes of which are currently in supercomputing platforms in research labs. These systems implement an extended architecture of computing nodes fitting the requirements of the model. Currently, there are no such systems, known to the researcher at the time of publication, that run the classic or neural network based HTM system. It is the strong belief of the researcher that such an implementation will open the door for a vast number of possibilities in artificial cognition and AI in general.

Some of the important areas that can be investigated for the purpose of optimizing the model further are:

- Improved temporal clustering algorithms: the nodes in the enhanced as well as the original Numenta HTM model used a recursive clustering algorithm. In the prototype implementation of this model, the researcher has experimented with several other temporal clustering algorithms none of which matched the performance of that of Numenta's. A deeper study in the mater will be worth the effort as the algorithm is a crucial part of the model's performance.
- The researcher has used a fixed configuration of cloning and crossover percentages in optimizing the neural networks. This has been due to the limited effect of variations in these factors observed during the design phase. A further investigation may result in configurations with better yield.
- HTM topology design has been limited to regular tree configuration in both the original and proposed models. The general theory, however, allows for more irregular arrangements such as feedback connections and level skipping. Such tailored designs may give optimal performance for specific applications.

Bibliography

[1] Mostefa Golea, Mario Marchan. . **A Growth Algorithm for Neural Network Decision Trees.** Department of Physics, University of Ottawa. 1990. www.iop.org/EJ/abstract/0295-5075/12/3/003. [Last Accessed June 2007]

[2] Eleine Rich, Kevin Knight. **Artificial Intelligence**, 2nd Ed., pp. 3-27. Tata McGraw-Hill, 1991.

[3] [1]hugo De Garis, [2]sung-Bae Cho, [3]michael Korkin, [4]arvin Agah. **Designing An Artificial Brain With 10,000 Evolved Neural Net Modules.** [1]Brain Builder Group. [2]Dept of Computer Science, Yonsei University. [3]Genobyte Inc. [4]Bio-Robotics Division, Mechanical Engineering Lab (MEL), Tsukuba Science City. 1998. http://citeseer.ist.psu.edu/49849.html. [Last Accessed June 2007]

[4] [1]J. Djordjevic, [2]A. Milenkovic, [2]S.Prodanovic. **A Hierarchical Memory System Environment.** [1]Faculty of Electrical Engineering, University of Belgrade. [2]Informatika Jevrejska. 1999. http://www.ncsu.edu/wcae/ISCA1998/milenkovic.pdf. [Last Accessed July 2007]

[5] Risto Miikkulainen, James A. Bednar, Yoonsuck Choe, and Joseph Sirosh. **A Self-Organizing Neural Network Model Of The Primary Visual Cortex.** Department of Computer Sciences, The University of.2000. http://citeseer.ist.psu.edu/137449.html. [Last Accessed June 2007]

[6] Asim Roy. **Artificial Neural Networks - A Science in Trouble.** Arizona State University, School of Information Management. 2000. http://portal.acm.org/ft_gateway.cfm?id=846192&type=pdf&dl=portal&dl=ACM. [Last Accessed April 2007]

[7] Dean V. Buonomano, Michael Merzenich. **A Neural Network Model of Temporal Code Generation and Position-Invariant Pattern Recognition.** Keck Center for Integrative Neuroscience, University of California. 2001. http://neco.mitpress.org/cgi/reprint/11/1/103.pdf. [Last Accessed July 2007]

[8] David Voge. **A Neural network Model Of Memory And Higher Cognitive Functions In The Cerebrum**. Ross University. 2002. http://www.bbsonline.org/documents/a/00/00/11/77/bbs00001177-01/Vogel_HigerFnct.pdf. [Last Accessed April 2007]

[9] [2]Gwendid T. [1]van der Voort, [1]van der Kleij, [2]Marc de Kamps, [1]Frank van der Velde. **A Neural Model of Binding and Capacity in Visual Working Memory.** [1]Cognitive Psychology Unit, University of Leiden. [2]Robotics and Embedded Systems, Department of Informatics, Technische Universit¨. 2003. http://www.springerlink.com/index/kc9kxcru7q3101ym.pdf. [Last Accessed April 2007]

[10] Włodzisław Duch. **Brain-Inspired Conscious Computing Architecture.** School of Computer Engineering, Nanyang University of Technology. Department of Informatics, Nicholas Copernicus University. 2003. http://cogprints.org/3319/1/03-Brainins.pdf. [Last Accessed July 2007]

[11] LIN Jie, JIN Xiao-gang, YANG Jian-gang. **A hybrid neural network model for consciousness.** Institute of Artificial Intelligence, Zhejiang University. 2004. http://www.zju.edu.cn/jzus/2004/0411/041119.pdf. [Last Accessed July 2007]

[12] Gert Westermann, Denis Mareschal . **Connectionist modeling.** 2004. http://www.cnbc.cmu.edu/~plaut/papers/pdf/Plaut00chap.conn.pdf. [Last Accessed July 2007]

[13] 1Dileep George, [2]Jeff Hawkins. **Invariant Pattern Recognition using Bayesian Inference on Hierarchical Sequences**. [1]Department of Electrical Engineering, Stanford University and Redwood Neuroscience Institute. [2]Redwood Neuroscience Institute. 2004. http://www.stanford.edu/~dil/RNI/DilJeffTechReport.pdf. [Last Accessed July 2007]

[14] Bin Zhang, Sargur N. Srihari. **Properties of Binary Vector Dissimilarity Measures.** CEDAR, Computer Science and Engineering Department, State University of New York. 2005. http://www.cedar.buffalo.edu/papers/articles/CVPRIP03_propbina.pdf. [Last Accessed June 2007]

[15] [1]Dileep George, [2]Jeff Hawkins. **A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex.** [1]Department of Electrical Engineering Stanford University and Redwood Neuroscience Institute. [2]Redwood Neuroscience Institute. 2005. http://www.stanford.edu/~dil/invariance/Download/GeorgeHawkinsIJCNN05.pdf. [Last Accessed July 2007]

[16] Xiao-Jing Wang. **A Microcircuit Model of Prefrontal Functions: Ying and Yang of Reverberatory Neurodynamics in Cognition.** Volen Center for Complex Systems, Brandeis University. 2005. http://assets.cambridge.org/97805216/72252/frontmatter/9780521672252_frontmatter.pdf . [Last Accessed August 2007]

[17] [1]M. W. Spratling, [2]M. H. Johnson. **A Feedback Model of Perceptual Learning and Categorization.** [1]Division of Engineering, King's College. [2]Centre for Brain and Cognitive Development, Birkbeck College. 2006. http://cogprints.org/4885/1/vis_cog06.pdf. [Last Accessed December 2007]

[18] Jeff Hawkins, Dileep George. **Hierarchical Temporal Memory - Concepts, Theory, and Terminology,** Numenta Inc. 2006. http://www.numenta.com/Numenta_HTM_Concepts.pdf. [Last Accessed July 2007]

[19] Numenta Inc. **Hierarchical Temporal Memory, Comparison with Existing Models.** 2006. http://www.numenta.com/for-developers/education/HTM_Comparison.pdf. [Last Accessed December 2007]

[20] Dileep George, and Bobby Jaros. **The HTM Learning Algorithms,** Numenta Inc. 2006. http://www.numenta.com/for-developers/education/Numenta_HTM_Learning_Algos.pdf. [Last Accessed July 2007]

[21] Numenta Inc. **Numenta Platform for Intelligent Computing Programmer's Guide Version 1.0.3.** 2007. http://www.numenta.com/for-developers/education.php. [Last Accessed May 2007]

[22] Numenta Inc. **Zeta1 Algorithms Reference, Version 1.2.** 2007. http://www.numenta.com/for-developers/software/pdf/nupic_gettingstarted.pdf. [Last Accessed June 2007]

[23] Wikipedia [Free online Encyclopedia]. **Connectionism.** http://en.wikipedia.org/wiki/Connectionism Last Revision: 16, Nov. 2007

[24] Wikipedia [Free online Encyclopedia]. **Neural Networks.** http://en.wikipedia.org/wiki/Neural_networks. [Last Accessed June 2007] Last Revision: 11, Dec. 2007

[25] Wikipedia [Free online Encyclopedia]. **Cognition.** http://en.wikipedia.org/wiki/Cognition Last Revision: 18, Dec. 2007

[26] Wikipedia [Free online Encyclopedia]. **Blue Gene.** http://en.wikipedia.org/wiki/Blue_gene Last Revision: 2, Dec. 2007

[27] Wikipedia [Free online Encyclopedia]. **Supercomputers** http://en.wikipedia.org/wiki/Supercomputers Last Revision: 14, Jul. 2007

[28] Wikipedia [Free online Encyclopedia]. **Artificial Intelligence.** http://en.wikipedia.org/wiki/Artificial_intelligence Last Revision: 15, Jan. 2008