

AMERICAN UNIVERSITY OF BEIRUT

HIERARCHICAL TEMPORAL MEMORY:
AN INVESTIGATIVE LOOK INTO A NEW CORTICAL
ALGORITHM

by
NICHOLAS GABI MITRI

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
April 2015

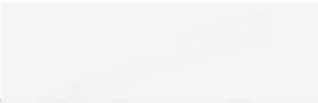
AMERICAN UNIVERSITY OF BEIRUT

HIERARCHICAL TEMPORAL MEMORY:
AN INVESTIGATIVE LOOK INTO A NEW CORTICAL
ALGORITHM

by
NICHOLAS GABI MITRI

Approved by:

Dr. Mariette Awad, Assistant Professor
Electrical and Computer Engineering



Advisor

Dr. Mohamad Adnan Al-Alaoui, Professor
Electrical and Computer Engineering



Member of Committee

Dr. Robert Habib, Associate Professor
Department of Internal Medicine



Member of Committee

Date of thesis/dissertation defense: April 27th, 2015

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: Mitri Nicholas Gabi
 Last First Middle

Master's Thesis Master's Project Doctoral Dissertation

I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

I authorize the American University of Beirut, **three years after the date of submitting my thesis, dissertation, or project**, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.


Signature

May 7th, 2015

Date

ACKNOWLEDGMENTS

Special thanks to my advisor Prof. Mariette Awad for her continual support and guidance.

My gratitude also goes to Dr. Robert Habib for sharing his work and time, as well as the Numenta team for their assistance and advice.

AN ABSTRACT OF THE THESIS OF

Nicholas Gabi Mitri for Master of Engineering
Major: Electrical and Computer Engineering

Title: Hierarchical Temporal Memory: An Investigative Look into a New Cortical Algorithm

The connectionist approach to artificial intelligence has made many attempts to capture the essence of intelligence by adopting some of the principles underlying the operation of the human brain. Since its emergence in the 1950s, this field has been evolving with novel algorithms finding their way to the forefront. One such algorithm is Hierarchical Temporal Memory (HTM).

While lacking in maturity, HTM's approach to prediction and online learning has garnered the interest of machine learning practitioners. Major players in the tech industry like IBM have also taken notice of HTM as of April, 2015 and have dedicated their resources to investigating its merit.

The work presented in this thesis reflects that interest and seeks to establish where HTM stands in the grander machine learning scheme. To that end, we take advantage of the modular design of HTM and propose a number of tests aiming to evaluate its individual modules as well as HTM as a whole. In modular testing, HTM is put through a series of basic tests for clustering and sequence learning tasks. These experiments reveal some of the spatial and temporal pooling promise of HTM but ultimately expose some of its inherent flaws. More involved experiments utilizing the entire HTM stack go on to demonstrate that further with HTM struggling when adapted to multi-class classification tasks and anomaly detection on streaming data.

CONTENTS

ACKNOWLEDGEMENTS	v
ABSTRACT	vi
LIST OF ILLUSTRATIONS	x
LIST OF TABLES	xiv

Chapter

I. INTRODUCTION	1
II. THE NEUROSCIENCE BEHIND HTM	7
III. RELATED WORK	15
IV. OVERVIEW OF HTM & CLA	18
A. HTM Structure	18
B. Cortical Learning Algorithms	21
1. Sparse Distributed Representation	22
a. Sparseness	23
b. Similarity and Comparison	23
c. Storage Efficiency	24
d. Subsampling	24
e. Fault Tolerance	24
f. Union	24
2. Encoders	25
a. Scalar Types	26
b. Category	28
3. Spatial Pooler	29
4. Temporal Pooler	32
5. CLA Classifier	36

a.	Input to CLA Classifier.....	37
b.	Operation.....	38
i.	Learning.....	38
ii.	Inference.....	40
6.	Anomaly Detector.....	41
V.	SPATIAL POOLER FOR BASIC CLUSTERING.....	43
A.	Impact of Noisy Input	43
1.	Clustering without Bias	43
2.	Clustering with Bias	46
B.	SP vs. Clustering Algorithms.....	49
C.	Overlap as Similarity Measure.....	58
VI.	TEMPORAL POOLER FOR SEQUENCE PREDICTION	64
A.	HTM Network Configurations.....	66
B.	First-order Deterministic Sequences	68
C.	High-order Deterministic Sequences	69
D.	Stochastic Sequences	76
VII.	CASE STUDY: HTM IN CLASSIFICATION AND ANOMALY DETECTION	79
A.	Case Study I: Smartphone Based Gesture Recognition Using HTM.....	79
1.	Experimental Setup.....	80
a.	Learning Scenarios.....	80
b.	Network Structure	81
2.	Results	83
a.	User-Dependent	84
b.	User-Independent.....	85
c.	Mixed	86
d.	Discussion	87
B.	Case Study II: Irregular Breathing Detection in CPAP Assisted Patients Using HTM	89

1. Periodicity Detector	91
2. The Numenta Anomaly Benchmark	93
a. What is NAB?	93
b. NAB Scoring.....	95
3. Results	98
VIII. CONCLUSION.....	106

ILLUSTRATIONS

Figure		Page
2.1.	Diagram of the brain with cerebral lobes and examples of cortical sulci and gyri labeled.....	6
2.2.	Illustration of a cross section of the cerebral cortex using different staining techniques.....	11
2.3.	Histological Structure of the Cerebral Cortex.....	12
2.4.	A cortical column spanning all six layers of the cerebral cortex.....	13
4.1.	An HTM cell/neuron model.....	18
4.2.	HTM columns with cells demonstrating the 3 possible states: active (dark grey), predictive (light grey), and inactive (white).....	20
4.3.	Structure of an HTM model.....	21
4.4.	Component level diagram of HTM/CLA.....	22
4.5.	An HTM region with FF activations (red) and resulting multiple predictions occurring simultaneously (yellow).....	25
4.6.	Spatial Pooler flow chart.....	32
4.7.	Temporal pooler Phase I.....	35
4.8.	Temporal Pooler Phase II (left) and Phase III (right).....	36
5.1.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for an unbiased 64-column HTM region.....	45
5.2.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for an unbiased 1024-column HTM region.....	45
5.3.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 64-column HTM region biased using 20 passes	

	of cluster centers.....	47
5.4.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 64-column HTM region biased using 50 passes of cluster centers.....	47
5.5.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 1024-column HTM region biased using 20 passes of cluster centers.....	48
5.6.	Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 1024 column HTM region biased using 50 passes of cluster centers.....	48
5.7.	Data points generated for 2D clustering test.....	50
5.8.	K-means clustering outcome for 4 different runs under the same parameter settings.....	51
5.9.	Fuzzy C-means clustering outcome for 4 different runs under the same parameter settings.....	52
5.10.	Agglomerative Hierarchical clustering dendrogram (left) and outcome (right).....	53
5.11.	SP clustering outcome for 4 different runs with $w = 45$ and no bias passes.....	54
5.12.	SP clustering outcome for 4 different runs with $w = 29$ and 50 bias passes.....	55
5.13.	Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.....	60
5.14.	Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.....	60
5.15.	Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.....	62
5.16.	Overlap Contours for 2D inputs with reference to origin.....	62

5.17.	Overlap Contours for 2D inputs with reference to origin.....	62
5.18.	Overlap Contours for 2D inputs with reference to origin.....	63
5.19.	Similarity Contours for 2D inputs with reference to origin using Euclidean (left) and Manhattan (right) distance.....	63
6.1.	Revised HTM Model.....	68
6.2.	Second-order Deterministic Sequence Set.....	71
6.3.	Prediction performance with varying number of cells per column.....	72
6.4.	Performance for 3 (left) and 4 (right) cells per columns for different numbers of training passes.....	72
6.5.	Prediction performance for a 2048 column HTM region with 2% sparsity as a function of cells per column.....	76
7.1.	Samples of gestures traced by a single user.....	80
7.2.	Flowchart displaying the experimental setup for HTM based classification.....	83
7.3.	Breathing patterns recorded for patient 4 showing a high ratio of anomalous to regular patterns.....	91
7.4.	Breathing patterns recorded for patient 14 showing a proper ratio of anomalous to regular patterns.....	91
7.5.	Flowchart of custom detector.....	93
7.6.	Flowchart of NAB's workflow.....	95
7.7.	Scaled sigmoid scoring function.....	96
7.8.	Selection of ground truth apnea anomalies.....	99
7.9.	Selection of ground truth sigh anomalies.....	99
7.10.	Selection of ground truth general anomalies.....	100
7.11.	Anomaly detection using the HTM-based detector.....	101

7.12. Anomaly detection using the custom periodicity detector.....	102
--	-----

TABLES

Table		Page
6.1.	Number of training passes needed for the TP and CLA classifier to make a perfect prediction at different configurations.....	69
6.2.	Number of unique contexts in which every character occurs.....	74
6.3.	Error between HMM estimated transition matrix and CLA classifier transition matrix.....	78
7.1.	Classification accuracy (%) for 5 users under the user-dependent scenario.....	85
7.2.	Classification accuracy (%) for 5 users under the user-dependent scenario.....	86
7.3.	Classification accuracy (%) for 5 users under the user-dependent scenario.....	87
7.4.	Raw scores and normalized total scores after NAB testing using a scaled sigmoid scoring function.....	103
7.5.	Normalized total scores after NAB testing using a square scoring function.....	103
7.6.	Results of a second run of NAB testing on the HTM-based detector and our custom detector.....	105

CHAPTER I

INTRODUCTION

The human brain has been at the core of Artificial Intelligence (AI) research since the inception of the field. With its web of over 10^{11} interconnected neurons, its unique capacity for learning and adapting has secured its position as the Holy Grail of AI and inspired countless theories on both general intelligence and task-oriented learning. A considerable part of the work in AI aims for a higher level abstraction of the functionality of the brain and employs symbolic structures to encode knowledge and conceive intelligent architectures to emulate it. This has become known as the symbolic approach to AI. Alternatively, others pursue a bottom-up methodology where low-level structural emulation of the physiology of the brain, both simplified and scaled down, is the goal. They employ networks of simple, massively connected processors and encode knowledge as a pattern of numerical strengths of the connections between these processors.²⁹ The latter is the connectionist trend in AI and will be the umbrella under which all the ideas presented in this thesis fall.

At the cornerstone of the connectionist vision are Artificial Neural Networks (ANNs). ANNs are collections of basic computational units modeled after the biological neuron. Organized in networks of varying topologies, these artificial neurons were designed to approximate the information processing capabilities of the brain, and its intelligence by proxy, through instance, batch, or incremental learning. ANNs, as a means to neuro-computing, marked their beginnings with the McCulloch-Pitts neurons proposed in 1943.¹⁹ These were basic summation units with uniformly weighted excitatory and inhibitory connections and binary activation functions. They were the

first model to demonstrate how artificial neurons can compute arithmetic or logical functions. Later models include Frank Rosenblatt's perceptron²⁶ and Bernard Widrow's ADALINE,³² both of which allowed for continuous activation functions but relied on slightly different interpretations of Hebb's synaptic learning rule.¹³ ANNs leveraging these models (and others) were still being used in single layer configuration throughout the 1960s, and their inability to solve simple tasks like the XOR problem was becoming evident. In 1969, Marvin Minsky and Seymour Papert's book *Perceptron*²⁰ demonstrated that these single-layered ANNs were only capable of learning linearly separable patterns. With that and after premature claims that these ANNs were the answer to AI and true machine intelligence, ANN research suffered a decline in both interest and funding. In 1973, Stephen Grossberg proposed in a series of papers multi-layered perceptron networks as a solution to ANN's ailments.⁷ Alas, the field had stagnated and would not gain renewed vigor until the 1980s which saw a wider adoption of back-propagation and other involved learning algorithms as well as the introduction of more complex networks like self-recurrent Hopfield networks,¹⁵ self-organizing maps (SOM),¹⁶ adaptive resonance theory (ART) networks,⁸ time delay neural networks,³¹ and others. Since that resurgence, ANNs have been a mainstay of AI and have been heavily utilized in many applications including vision, speech, and natural language.¹⁸

Despite their wide adoption and many successes, ANNs demonstratively failed to live up to early promises of human-level intelligence. A major contributing factor to this failure has been the reliance on shallow architectures i.e. networks with few layers and a limited number of computing units within each layer. Until recently, this trend was an unfortunate necessity brought on by the lack of adequate computational power and the sub-par scalability of traditional learning algorithms (e.g. back-propagation) due

to technical limitations like vanishing gradient. Additionally, the curse of dimensionality remained a sticking point for any endeavor into deeper networks. It wasn't until recently that the explosion of computing power and introduction of more suitable learning algorithms like Hinton's greedy algorithm¹⁴ that deep learning started gaining traction and set itself as a top trend in the research community. Currently, it stands as the go-to technique for top tier tech companies like Google and Facebook and leads the charge in the ever-growing NN field.

Whether deep learning will prove to be the key to unlocking true machine intelligence is still unclear. With the complexity of the brain and the intricacies of its internal dynamics still not fully understood, there still exists room for innovating models that borrow more of its fundamental structural elements and operational principles. Current literature continues to offer such models. One recent addition is Hierarchical Temporal Memory (HTM), a memory model that merges neuro-computing and neuroscience in a novel way.

HTM capitalizes on the current trend of exploiting deep architectures to extend the learning abilities of intelligent systems. It was proposed by Jeff Hawkins and Georges Dileep as a biomimetic computational model aimed at replicating the functional and structural properties of the neocortex. It incorporates a number of insights from neuroscience as detailed in Hawkins' 2004 book "On Intelligence".¹² Its foundation is deeply rooted in the Memory Prediction Framework (MPF) proposed by Hawkins and Dileep,⁶ which postulates that the key to intelligence is the ability to predict. Consequently, HTM incorporates temporal analysis into its computational model, which sets it apart from the majority of available neuromorphic architectures.

The hierarchical structure of HTM is inspired by the brain. The brain's structure, specifically that of the neocortex, evolved in such a way as to gain the ability to model the structure of the world it senses. At its simplest abstraction, the brain can be viewed as a biological data processing black box that discovers causes in an environment that imposes massive amounts of data on its inputs (senses). The external causes to this continuous stream of information are by nature hierarchical, both in space and time. They are a collection of smaller causes or building blocks that combine to form a larger picture of the world. For instance, speech can be broken down to sentences, sentences to word utterances, word utterances to phonemes, etc. With digital imagery, pixels combine into edges, edges into contours, contours into shapes and finally objects. Every sensed object in the world reveals a similar structure where it can be perceived at varying levels of granularity. It is this hierarchically organized world that the neocortex and therefore HTM by imitation aim to model. This modeling happens in HTM at every level.

In HTM, the lowest level nodes of the network are fed sensory information. This information can be raw or pre-processed depending on the task the network is performing. The nodes learn the most basic features of the data stream by discerning repeatable patterns and sequences in which these patterns occur and storing them (either in local memory structures or via connectivity configurations). These basic patterns and sequences are then used as building blocks at higher levels to form more complex representations of sensory causes. As information travels up the hierarchy, the same learning mechanics are utilized as higher and higher abstractions of the input patterns are formed. Information can also flow down the hierarchy. This allows the network to

act as a generative model where higher levels bias lower levels by communicating their internal states in order to fill in missing input data and/or resolve ambiguity.¹²

HTM has seen two generations during its evolution. The underlying implementation in the 1st generation, known as “zeta 1”, is strongly rooted in Bayesian Belief Propagation (BBP) and borrows much of its computations and rules of convergence from that theory. The 2nd generation algorithms were created to make the framework more biologically plausible. Functionally, much of the concepts of invariant representations and spatial and temporal pooling carry over. In order to achieve that, principles of Sparse Distributed Representations (SDR) and structural changes were employed. Nodes were replaced with analogues of cortical columns with biologically realistic neuron models, and connectivity was altered to allow for strong lateral inhibition.

The new HTM algorithms replace Zeta 1. These algorithms were initially referred to as Fixed-density Distributed Representation (FDR), but are now known simply as HTM Cortical Learning Algorithms (CLA). In lieu of the “barrel” hierarchy with clean-cut receptive fields, each level in the updated framework is a continuous region of cells stacked into columns that act as a simplified model of a cortical column. This 2nd generation of the HTM algorithms will be the focus of the empirical study presented in this work where we aim to make qualitative assessments of the efficacy of HTM and establish its standing in the grander machine learning (ML) scheme. This is accomplished through a comprehensive experimental strategy that adapts HTM to a wide spectrum of ML applications and pits it against popular and well established ML algorithms and will be the main contribution of this work.

The remainder of this thesis is organized as follows. Chapter II introduces to the reader to the neuroscience behind HTM revealing a number of influential insights that motivated its design. Chapter III is a brief listing of HTM related work in the literature. Chapter IV offers an overview of HTM and the algorithms that guide its learning. This chapter will act as a comprehensive introduction to HTM for the interested newcomer. Chapter V marks the beginning of a series of experiments to evaluate HTM starting with clustering tests. Chapter VI continues to examine HTM as a sequence learning while Chapter VII capitalizes on the previous chapters to evaluate HTM using real world data in both classification and anomaly detection tasks. We finally wrap up the thesis with a conclusion in Chapter VIII.

CHAPTER II

THE NEUROSCIENCE BEHIND HTM

The human brain is an organ weighing less than 3 lbs and comprises a network of over 100 billion interconnected nerve cells “that construct our perceptions of the external world, fix our attention, and control the machinery of our action.”²⁵ The part of the brain most pertinent to any discussion of true intelligence is the neocortex. It is the most recently evolved part of the mammalian brain and most developed in man among all mammals. This sheet of neural tissue measures around 1000 cm² in surface area and 2 mm in thickness and is regarded as the seat of intelligent thought, supporting high level vision, hearing, touch, movement, language, and planning. It is the neocortex that allows us to perceive, act, learn, and remember at a capacity far superior to any other species.

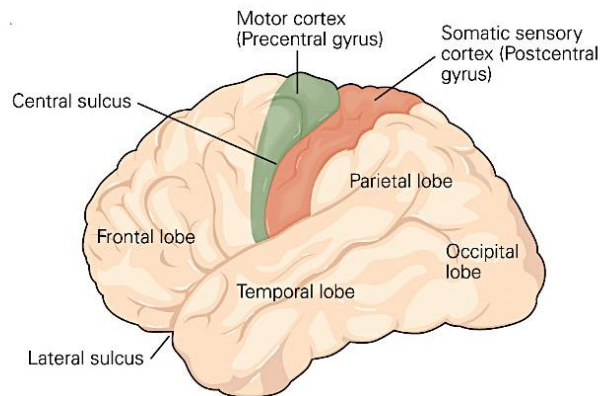


Figure 4.1: Diagram of the brain with cerebral lobes and examples of cortical sulci and gyri labeled²⁵

The study of nervous tissue as a special science did not start until the late 1800s when Camillo Golgi and Santiago Ramon y Cajal produced detailed descriptions of the nerve cells, thus supplying much of the early evidence for the neuron doctrine—the principle that individual neurons are the elementary building blocks and signaling elements in the nervous system. For the next two centuries, every endeavor in the study of the brain and the nervous system capitalized on these initial insights to develop a bigger picture of how the brain functions. The modern day view of nerve cells, the brain, and behavior evolved accordingly and finally emerged from a synthesis of various experimental traditions (anatomy, embryology, pharmacology, etc.) during the 20th century. This happened in the aftermath of two opposing theories on how the brain operates. The first dealt with functional localization and was pioneered by Viennese physician and neuroanatomist Franz Joseph Gall. Gall argued that the cerebral cortex did not function as a single organ but contained within it particular regions that control specific functions. His views on the modularity of the brain's function gave way to the doctrine of phrenology during the 19th century which suggested that complex traits such as combativeness, spirituality, hope, and conscientiousness are controlled by specific areas in the brain which physically expand as the traits develop. The opposing theory was the holistic view of the brain championed by the likes of French physiologist Pierre Flourens who argued that specific brain regions are not responsible for specific behavior, but that all brain regions, participate in every mental operation so that any individual part of the cerebral hemisphere can perform all the hemisphere's functions. This holistic view was seriously challenged in the mid-19th century by the French neurologist Paul Pierre Broca, the German neurologist Carl Wernicke, and the British neurologist Hughlings Jackson who proposed that different motor and sensory functions

could be traced to specific parts of the cerebral cortex. This developed into a view of the brain function called cellular connectionism. According to cellular connectionism, individual neurons are the signaling units of the brain and are arranged in (dozens of) functional groups that connect to one another in a precise fashion to produce cognitive functionality. With that, connectionism and Gall's original idea that discrete regions are specialized for different functions became the cornerstones of modern brain science.

New imaging techniques (like positron emission tomography, PET, and functional magnetic resonance imaging, fMRI) permit anatomical analysis and the visualization of the human brain in action. These techniques have ushered in a level of scrutiny of the brain's structure and its cognitive functions at unprecedented granularity.

At the level of intra-cortical connectivity, we now know that brain functionality is not produced at specific regions but as a result of a neural pathway, an interlinkage of serial and parallel processing in discrete regions characterized by their individual sub-functionalities. This interlinkage establishes a "hierarchical" organization that guides information processing in sensory and motor systems. Each region conveys information to an adjacent, higher order region, a unimodal association area where neurons selectively encode specific features of stimuli to represent information at higher and higher complexity. For example, at very advanced stages of the visual sensory system, individual neurons are responsive to highly integrated information, such as the shape of a face. Another feature of this cortical organization, most evident in sensory systems, is that inputs from peripheral receptive surfaces (retina, cochlea, etc...) are arranged topographically through successive stages of processing. Neurons at each successive relay of a sensory pathway form an orderly neural map of information from the receptive surface. These neural maps reflect not only the spatial arrangement of

receptors but also the variation in receptor density throughout the receptive surface. The density of innervation in an area of skin for example determines the degree of sensitivity of that area to tactile stimuli.

At the structural level, the subcortical regions reveal a nuclear layered organization. This organization is defined by six layers that span the thickness of the neocortex from its outer surface to the white matter. Figure 2.2 shows the result of applying different staining techniques applied to the neocortex. The Golgi stain reveals a subset of neuronal cell bodies, axons, and dendritic tree. The Nissi method shows cell bodies and proximal dendrites. A Weigert stain reveals the pattern of myelinated fibers. Each of the cortical layers is defined primarily by the presence, absence, and packing density of distinctive cell types, but other cytoarchitectonic differences like cell size are similarly considered. The histological structure of the cerebral cortex can be seen in figure 2.3 where different cell types of varying sizes are depicted. We leave the reader with a brief description of the cortical layers:

- Layer I is known as the molecular level. It is occupied by the dendrites of cells located in deeper layers and axons that travel through this layer to make connections in other areas of the cortex.
- Layers II and III contain mainly small pyramidal shaped cells. The axons of pyramidal neurons in layers II and III project to other cortical areas, thereby mediating intra-cortical communication.
- Layer IV is the main recipient of sensory input from the thalamus and is most prominent in primary sensory areas. For example, the region of the occipital cortex that functions as the primary visual cortex has a very

prominent layer IV. In contrast, the pre-central gyrus, the site of the primary motor cortex, has essentially no layer IV.

- In layer V, larger pyramidal neurons give rise to the major output pathways of the cortex projecting to other cortical areas and to subcortical structures.
- Layer VI blends into the white matter that forms the deep limit of the cortex and carries axons to and from areas of cortex.

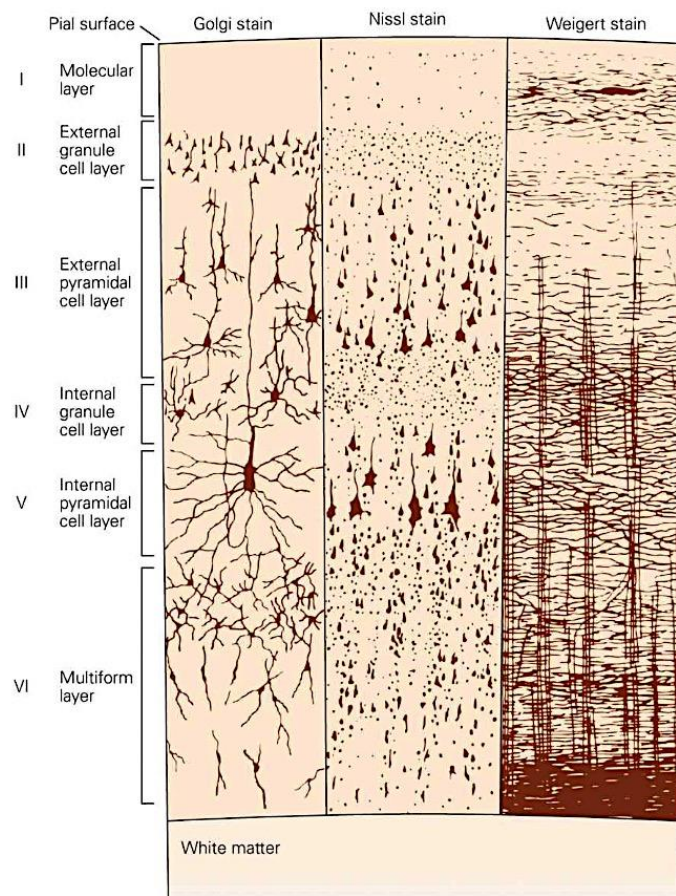


Figure 4.2: Illustration of a cross section of the cerebral cortex using different staining techniques.²⁵

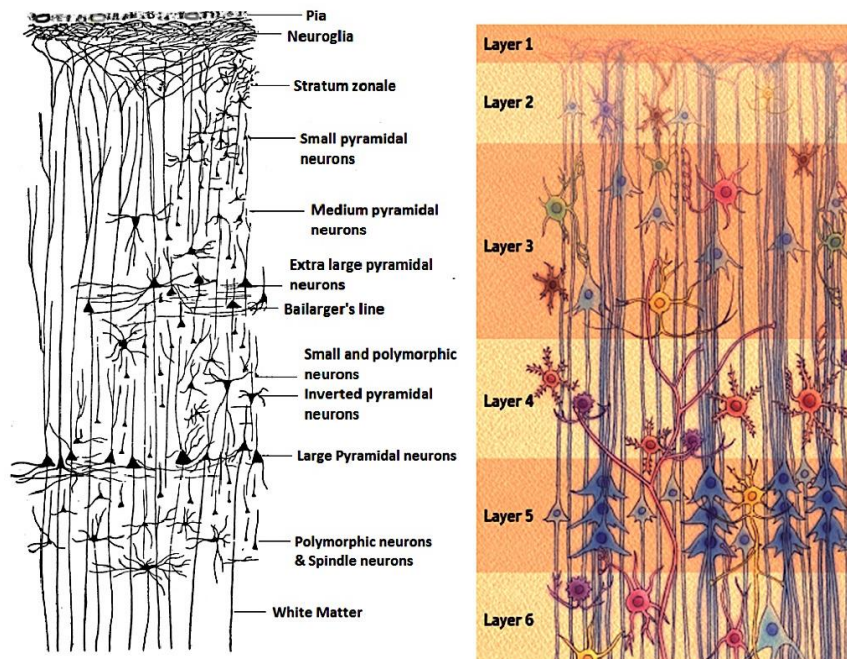


Figure 4.3: Histological Structure of the Cerebral Cortex.³

As for the direction of information flow, feedforward projections originate mainly in layer III and terminate mainly in layer IV of the target cortical area while feedback projections to earlier stages of processing originate from cells in layers V and VI and terminate in layer I, II, and VI.

Along the thickness of the neocortex, spanning the cortical layers, congruent neurons tend to exhibit similar response properties thus forming a local processing network. This structural property was highlighted in a series of pioneering studies by Vernon Mountcastle who discovered that the cortex is organized into vertical columns or slabs.²¹ Each of these columns is 300 to 600 μm wide and spans the six cortical layers from the surface to the white matter. Neurons within each column receive inputs from the same local area of the receptor sheet and respond to the same class of receptors. Horizontal connections within layers II and III link neurons in neighbouring columns, allowing them to share information when activated simultaneously by the same

stimulus. This exchange of information is believed to be part of an inhibitory process that regulates columnar activation. With that, the cortical column has come to be viewed as an anatomical structure that, by organizing inputs that convey related information on location and modality, comprises an elementary functional module of the cortex. More importantly, this information processing at the level of the column occurs in all regions of the neocortex. It is that uniformity in neural circuitry that gives evidence to the existence of a universal and common set of cortical algorithms that enables the brain to perform many different intelligence functions.

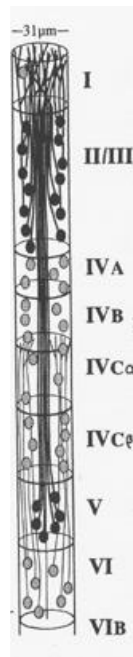


Figure 4.4: A cortical column spanning all six layers of the cerebral cortex.¹¹

At its core, HTM and its founding theory were highly influenced by the insights and discoveries of neuroscience. It provides a theoretical framework for understanding the neocortex and its many capabilities by leveraging these insights. Namely, HTM borrows the concepts of hierarchy, columnar structure, connectivity, and

universality of computation across the entire network. Additionally, it aims to model a subset of the cortical layers with the plan to model those engaged in feedback and thalamic control at later stages of development.

In its current iteration, HTM is limited to a feed-forward model that represents layers III for variable order memory or layer IV for single order memory. This model is populated by cells arranged in columnar fashion with proximal connections propagating information and inhibitory lateral connections creating sparse activation through the model's network.

CHAPTER III

RELATED WORK

In light of the fact that HTM is novel and has a small corpus of published work, this brief chapter aims to give the interested reader a clearer idea of the kinds of applications that the research community has adapted to HTM. For insight into related algorithms, we direct the reader to learning architectures like Deep Belief Networks¹⁴ (DBN), Convolutional Networks,³³ and most relevantly Lipasti's Cortical Algorithm^{9, 10} but exclude any detailed discussions of the aforementioned models. In what follows, applications of HTM as seen in recent literature are showcased.

Zhituo et al. used multiple HTMs in a content-based image retrieval (CBIR) system that leverages categorical semantics of a query image rather than low-level image features for image indexing and retrieval. Using 10 HTMs and training and testing datasets of size 50 each, recall rates higher than 95% were achieved for 4 of the 5 categories involved and higher than 70% for the 5th.³⁴

Bobier et al. recreated a handwritten digit recognition experiment on the USPS dataset which was reported by Numenta to achieve a 95% accuracy rate. The digit images were binarized and fed to the network at varying parameters. The author was able to achieve a maximum rate of 96.26%, which was noted to be not on par with other classifiers like SVM, which delivered higher rates in a fraction of the computational time.¹

Kostavelis et al. presented a biologically inspired object recognition system. Saliency maps were used to emulate visual fixation and reveal only the relevant parts of the image, thus reducing the amount of redundant information presented to the

classifier. The authors chose to substitute the temporal pooler with a correlation-based alternative. Using the ETH-80 and supervised learning at the top node, they were able to outperform other HTM based implementations with both SVM and KNN as top-level supervisors.¹⁷

Sinkevicius et al. explored using HTM for human traffic analysis in public spaces. Two HTM networks are designed, one for human detection, the other for direction of movement detection. An experimental setup involving an overhead mounted camera on a doorway was performed. Evaluation of detection performance was done using multiple scenarios of varying difficulties. The average accuracy achieved was 80.94% for pedestrian detection and 73.44% for directional detection.²⁷

Boone et al. used HTM as an alternative to traditional computer vision techniques for diabetic retinopathy. HTM was used primarily for the detection of the optic nerve on retina images. Images were segmented into fragments the size of an optic nerve and presented with labels (0 or 1) to HTM. Following supervised training, the HTM network was able to correctly classify 77.44% of the presented optic nerves leading Boone et al. to conclude that HTM is not competitive with traditional techniques despite its promise.

Zhuo et al. supplemented state-of-the-art image classification techniques using Locality constrained Linear Coding (LLC) and Spatial Pyramid Matching (SPM) using HTM for feature pooling. Image descriptors were extracted and encoded using LLC. LLC codes were then fed to HTM and multi-scale SPM to form an image vector. The system was evaluated using the Caltech 101 dataset and UIUC-Sport dataset with linear SVM as the classifier. Results showed an increase in accuracy on both datasets (73.5% vs 71.2%, 86.7% vs 84.2% respectively) when compared to the original LLC model.³⁵

Gabrielsson et al. aimed to leverage HTM to create a profitable software agent for trading financial markets. A supervised training scheme was used for HTM with intraday tick data for the E-mini S&P 500 future markets being used as features. The tuned model was used as a predictor of market trends and showed at least comparable results when compared to an ANN.⁵

CHAPTER IV

OVERVIEW OF HTM & CLA

A. HTM Structure

The architecture proposed for the first generation HTM model was strongly influenced by the Bayesian rules it implemented. It benefited from structural characteristics of the neocortex, namely hierarchical organization, but nodes diverged from their biological counterparts. Functional modeling was the emphasis. In the second generation, HTM abandoned those roots and attempted to adhere to neocortical structural guidelines more strictly. The result was a proposed neuron model, known in this context as an HTM cell. Figure 4.1 depicts the model suggested by Hawkins and his research team.¹¹ These cells are noticeably more realistic and biologically faithful than those used in traditional ANN.

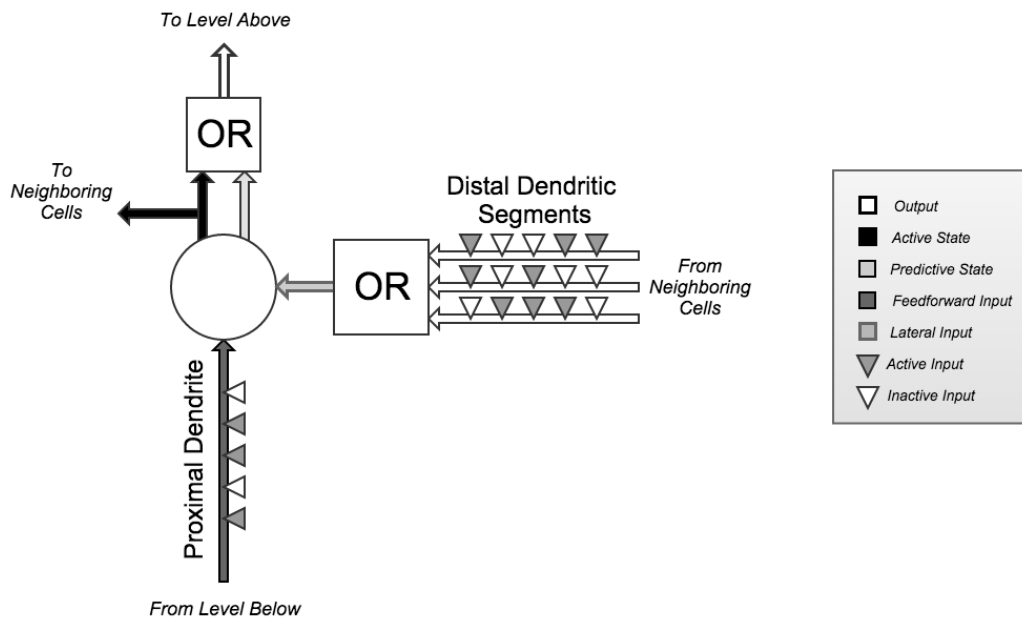


Figure 6.1: An HTM cell/neuron model

HTM cells have two types of incoming connection structures: proximal dendrites and distal dendrites. Dendritic segments of either type are populated by synapses that connect them to other cells. These synaptic connections are binary. One reason for that is the stochastic nature of real neurons. Any algorithm, according to Hawkins,¹¹ that aims to emulate the brain cannot rely on the precision or fidelity of individual neurons. Thus, HTM cells were modeled to have binary non-weighted synapses. They are either connected or not. To account for a real neuron's ability to retract and extend in order to form connection, HTM cell synapses are assigned a parameter called permanence. Permanence is a scalar value between the 0 and 1, which is incremented or decremented based on a synapse's contribution to cell activity. When permanence is above a predefined threshold, a synapse becomes connected. Therefore, all synapses in an HTM model are potential synapses. They are dynamic elements of connectivity.

Proximal dendrites are responsible for feed-forward (FF) connectivity between regions. They are populated by a set of potential synapses that are associated with a subset of the input to an HTM region. A single proximal dendritic segment is shared by all cells of a column (shared FF connectivity). A column's activity is defined by a scaled linear summation of the synaptic states (1 or 0) on this shared segment.

Distal dendrites are responsible for lateral connections across a single region. Several segments are associated with a distal dendrite. These segments act like a set of threshold coincidence detectors i.e. when enough connections are active at any one time, they trigger a response in the receiving cell. It is enough for one segment to be active to trigger a response in the cell (OR gate). Each segment connects an HTM cell to

a different subset of neighboring cells. The activity of those cells is monitored and allows the receiving cell to enter a predictive state. This is essentially the root of prediction in an HTM network. Every cell constantly monitors the activity of surrounding cells to predict its own.

An HTM cell has 2 binary outputs. The first, due to proximal connections, puts the cell into an “active” state if enough activation is present and the cell is chosen as the winning cell of its corresponding column. The second, due to distal connections, forces the cell into a “predictive” state. Finally, the output of the HTM cell is the OR of these 2 outputs. This is what regions higher up in the hierarchy receive as input. Figure 4.2 shows an example of activations of cells in a set of HTM columns.

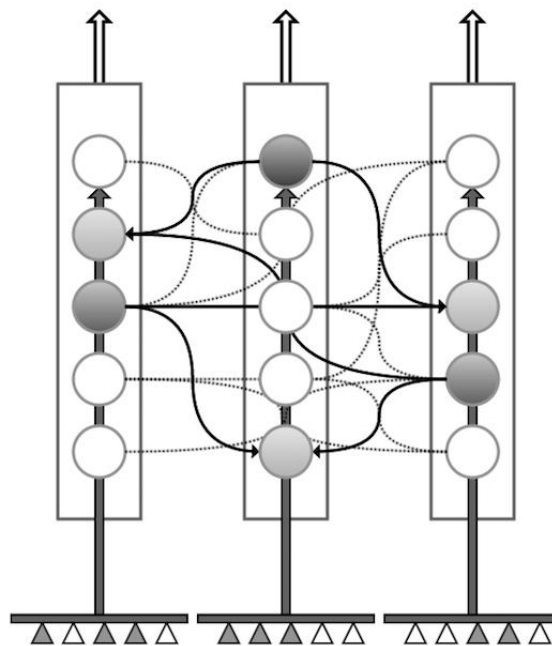


Figure 6.2: HTM columns with cells demonstrating the 3 possible states: active (dark grey), predictive (light grey), and inactive (white).

The HTM network is built using these novel cell models. Cells are grouped vertically in columns. Columns are arranged horizontally to form regions. This

arrangement can extend in dimensionality to form a 3D prism of cells. Finally, the regions are laid out hierarchically to form the network as depicted in figure 4.3.

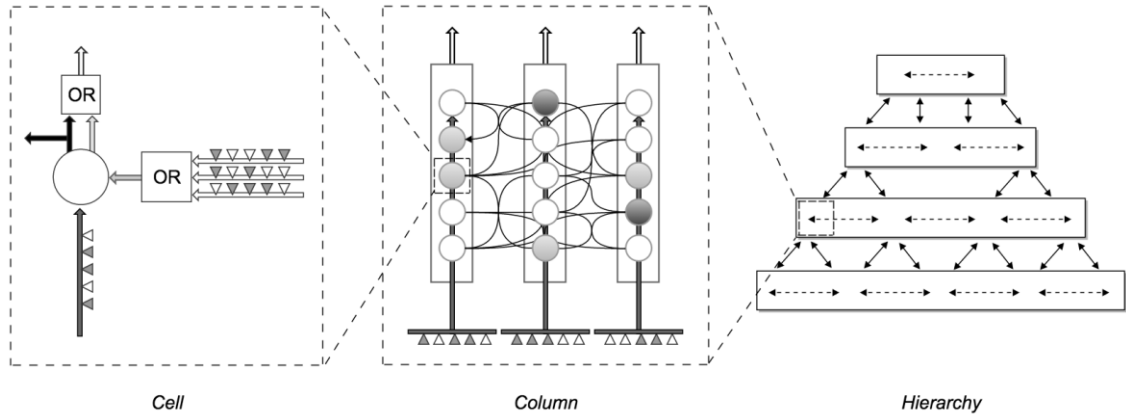


Figure 6.3: Structure of an HTM model.

B. Cortical Learning Algorithms

CLA, as of the date of completion of this thesis, is a work in progress. While some aspects of it, like the algorithms that make up Spatial Pooling, are largely finalized, other features are still moving targets. This is especially true for the Temporal Pooler. The developing team at Numenta used their published white paper¹¹ as a guideline in designing an iterative code base for CLA but have been forward about the many discrepancies between the theory and principles outlined in the paper and the current implementation. In this section, we aim to provide the reader with insight into both the founding principles and the current standing of the algorithms while offering clues to future direction as announced by the Numenta team and its developing community. Figure 4.4 shows a component level diagram of HTM/CLA. In what follows, we discuss the function of a number of these components and their current status in the software implementation.

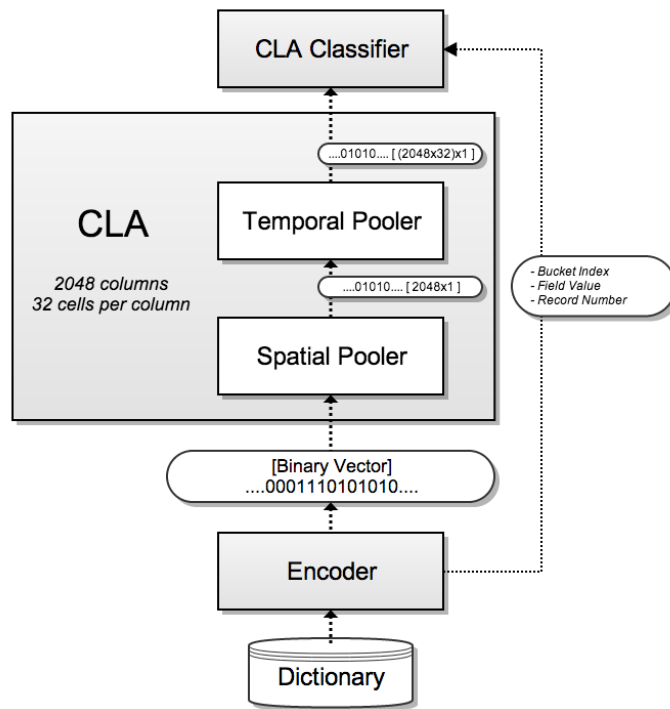


Figure 6.4: Component level diagram of HTM/CLA

1. Sparse Distributed Representation

CLA borrows a lot of its principles of operation from its biological analogue, the neocortex. The neocortex is made up of over 10^{11} neurons that are highly interconnected. Yet, it is still capable of reacting to stimuli with relatively sparse activations levels. This is made possible by the vast amounts of inhibitory connections. Inhibition guarantees that only a small percentage of neurons are active at any one time. CLA implements this sparsification strategy to produce what is termed Sparse Distributed Representations or SDRs. Strongly stimulated columns in CLA inhibit nearby activity, thus reducing the number of active columns and yielding a sparse internal representation of the input pattern or stimuli. This internal representation is distributed across an entire region.

With SDR, individual bits of a binary vector representation gain semantic meaning. This is in contrast to other representations like ASCII code. With dense representations like ASCII code, an individual bit carries no information. SDR, therefore, injects a representational quality into individual activations. Since only a small fraction of a possibly large number of neurons is active, whatever semantic meaning a single neuron gains becomes specific to a very limited number of similar patterns. Consequently, even a subset of the active neurons can be a good indicator of the input pattern that the activation set represents with little theoretical loss of information as claimed by Hawkins.¹¹

SDRs display a set of properties that confirm their viability in the HTM design paradigm:

a. Sparseness

With a small number of active bits, we compromise the number of possible representations for more robust representation. The sparseness ensures that having false matches for two substantially overlapping encodings is highly improbable. Additionally, any false matches (i.e. different inputs are assigned similar encodings) that come up are likely to represent mild semantic errors.

b. Similarity and Comparison

With sparse active bits retaining semantic meaning, two representations that share a number of active bits become semantically similar. With this metric of similarity, two SDRs can be compared by taking the intersection of their active bits. The more active bits they share, the stronger their semantic link.

c. Storage Efficiency

With only a sparse set of active bits, SDRs can be stored as indices of ON bits. This results in significantly lower memory demands considering that current implementations of CLA models typically use 2048-bit SDRs at 2% sparsity i.e. only 40 ON bits.

d. Subsampling

SDRs can be subsampled because the semantics of the data are distributed across the bits. This means that a subset of the active indices will typically be enough for comparison. A novel representation that contains all of the subsampled indices has a high likelihood of being the same value or at the very least is semantically very similar.

e. Fault Tolerance

SDRs are inherently fault-tolerant. Since semantics are distributed across the bits, several active bits can be discarded without a significant loss in representation. This is made possible for the same reason subsampling works.

f. Union

Due to sparseness and the fact that two different inputs are very unlikely to have overlapping representations, multiple sparse representations can be combined into one high dimensional vector. Through this superposition, one can store completely separate patterns in the same vector with minimal chance of false positives. This is

utilized in the Temporal Pooler where multiple predictions based on current input are made simultaneously as depicted in figure 4.5.

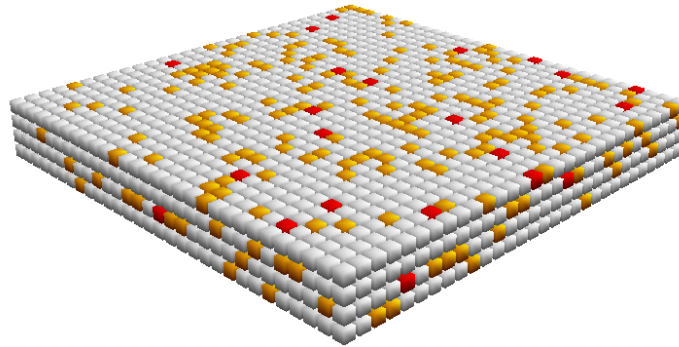


Figure 6.5: An HTM region with FF activations (red) and resulting multiple predictions occurring simultaneously (yellow).

2. Encoders

The first data processing module in the HTM/CLA model is the encoder. Encoders in HTM were designed as an analogue for biological sensory devices like the cochlea and the retina and are responsible for converting streaming input patterns into binary representations that are feed to the HTM network. This conversion process is data type dependent and thus requires the design of multiple encoders. For a number of these encoders, the output is a unique binary encoding of the pattern at the input stream (e.g. category encoder). For others, a quantization step is added that has the effect of reducing noise and binning similar input patterns together. This is similar in essence to the function of Local Sensitive Hashing²³ where the goal is to hash input patterns so that similar inputs are mapped to the same buckets with high probability while patterns deemed sufficiently different are assigned to different buckets. Below is a list of some of the encoders implemented in the CLA code base. We limit detailed discussion to scalar and category encoders thereafter.

- Scalar
- Adaptive Scalar
- Random Distributed Scalar
- Non-uniform Scalar
- Category
- Delta
- Log
- Date
- PassThru (Identity)
- Multi

a. Scalar Types

A scalar encoder linearly encodes a numeric value into a sparse array of bits using a contiguous block of '1's. The location of this contiguous block varies continuously as the input spans its entire range; with similar input patterns characterized by common semantics assigned overlapping representations. Scalar encoders use a number of parameters to determine the encoding for a given input:

- minval: minimum value of an input
- maxval: maximum value of an input
- w: width of a contiguous block of active bits
- n: number of encoder output bits
- resolution = $(\text{maxval} - \text{minval}) / (n - w)$: two inputs separated by more than the resolution are guaranteed to have different representations

- radius = resolution*w: two inputs separated by more than the radius have non-overlapping representations

The output of the encoder will have n total bits with w contiguous on-bits.

Values are placed into one of (n-w)+1 equally sized buckets. The smallest bucket is represented with the first w bits on and the rest off. The next larger bucket is represented by shifting the on bits to the right by one position. In this way, adjacent buckets have the most overlap, which helps to capture the semantics of scalar values. For example, with a range of 1-5, resolution = 0.5, w = 2, and n = 10, we get the following encodings:

1	=>	1100000000
1.5	=>	0110000000
2	=>	0011000000
⋮	⋮	⋮
5	=>	0000000011

There are a number of variations that use the design of the Scalar Encoder as a base. The Adaptive Scalar Encoder is identical to the Scalar Encoder except for the bounds imposed on the minimum and maximum values of the input. The Adaptive Scalar Encoder changes the limit values when it receives input that is outside of the initially set range. While that avoids the necessary clipping that the Scalar Encoder performs for such values, the change in minimum and maximum values means that previously encoded patterns become out of date and have to be relearned.

Another variation is the Non-Uniform Scalar Encoder where buckets are unequally distributed over the input range. This allows for more frequent patterns to be encoded with a higher resolution than those that occur less frequently. The adaptive

behavior of this encoder implies that it suffers the same shortcoming of the Adaptive Scalar Encoder in that previous encodings become obsolete as new patterns are processed and cause a change in bucket sizes.

Finally, we discuss one of the more popular encoders, the Random Distributed Scalar Encoder (RDSE). The RDSE assigns to input patterns encodings whose on-bits are distributed along the output of the encoder instead of presenting them in contiguous blocks. This allows the encoder to represent a larger number of patterns while dynamically changing the min and max range with no negative effect. The representation for a previously in-range scalar is unaffected by a change in range. Additionally, it preserves the important properties of overlapping representations. Similar scalars have high overlap with overlap decreasing smoothly as scalars become less similar.

b. Category

A Category Encoder performs the simple task of converting a list of discrete categories into unique non-overlapping binary representations. Its operation is that of a Scalar Encoder with integer inputs and a window of 1. Each category described by a string is assigned an integer, which is encoded as a block of consecutive on-bits of width w . The next category is encoded as a similar block of on-bits that is shifted by w bits. As an example, an RGB category encoder with $w = 3$ and $n = 9$ would produce the following encodings:

“Green”: 1 => 111000000

“Red”: 2 => 000111000

“Blue”: 3 => 000000111

3. *Spatial Pooler*

Spatial pooling is a task handled at the columnar level of an HTM region. The goal here is to group encoded patterns that are spatially similar and assign them a unique label. The resulting label is an SDR formed using a few strongly active columns.

The function of the SP can be interpreted as both a quantizer and an auto-encoder. On the regional level, similar inputs are pooled or clustered together. Since patterns are represented by a select few strongly active columns, minor changes at the input levels are less likely to cause the activation pattern to shift. Thus, noisy versions of an input are represented similarly at the output level of the SP. In that respect, the SP is very similar to a traditional quantizer in function. With the addition of hierarchy, the SP gains additional auto-encoding functionality. Using Hebbian like learning rules, every column in an HTM region adapts to fire to a certain input sub-pattern. As learning progresses and information is propagated up the hierarchy, features are automatically learned and extracted at the bottom levels from fast varying data only to be combined into more stable abstractions at the higher levels.

Algorithmically, the SP’s operation is defined by an initialization step and 3 functional phases: overlap, inhibition, and learning. During an initialization step, the proximal synapses of each column in a CLA region are cast over a portion of the input bits forming a “potential pool”; typically 50% of input bits. In the whitepaper, each of these synapses is configured with a permanence value chosen from a Gaussian distribution with higher values registered closer to the natural center of a column over the input region. In the current implementation of the SP, permanence allocation is

mainly made randomly with values centered on a threshold for synaptic activation. A parameter $potentialPct$ is used to determine the size of a random set of potential synapses that will have their permanence set above the threshold.

In Phase I, when presented with input data, all columns in an HTM region compute their feed-forward activations, referred to herein as overlap. A column's overlap is a summation of all the input bits in its receptive field multiplied by a boosting factor as shown in (4.1) for a column i with k proximal synapses (input bits).

$$overlap(i) = boost * \sum_k syn(k) \quad (4.1)$$

Columns that register overlaps greater than a predefined stimulus threshold are eligible to become active. Only a small subset of these strongly stimulated columns will be allowed to actually activate. This allows similar inputs to be represented internally by a sparse set of strongly active columns thus rendering the representation tolerant to noise.

The sparsification is established in Phase II via an inhibition process. During inhibition, candidate columns compete either locally or globally. In local inhibition, the number of winning columns in a local area of inhibition (neighborhood of a column) is set to a predefined value, N . A column will be a winner if its overlap score is greater than the score of the N th highest column within its inhibition radius. While more biologically realistic, inhibition on local neighborhoods is highly computationally demanding and restrictive. In global inhibition, reported as 60x less computationally expensive, the top columns with the highest overlap scores across an entire CLA region

are made active. Current implementations of the algorithm favor a 2% level of sparsity with typical counts of 2048 columns per region and global inhibition.

During the 3rd and final phase of learning, updates to the permanence values of all synapses are performed as necessary, as well as to parameters like boost and inhibition radius (for local only). For winning columns, if a synapse is active, its permanence value is incremented; otherwise it is decremented. To give other columns a fair chance of activating and ensure that all regional columns are utilized, two separate boosting mechanisms are implemented periodically utilizing duty cycles. If a column does not win often enough, its overall boost value is increased. Alternatively, if a column's connected synapses do not overlap well with any inputs often enough i.e. consistently low overlap scores, its permanence values are boosted. This phase terminates with updating the inhibition radius if local inhibition is employed. Figure 4.6 shows a flow chart of the 3 phases involved in the SP.

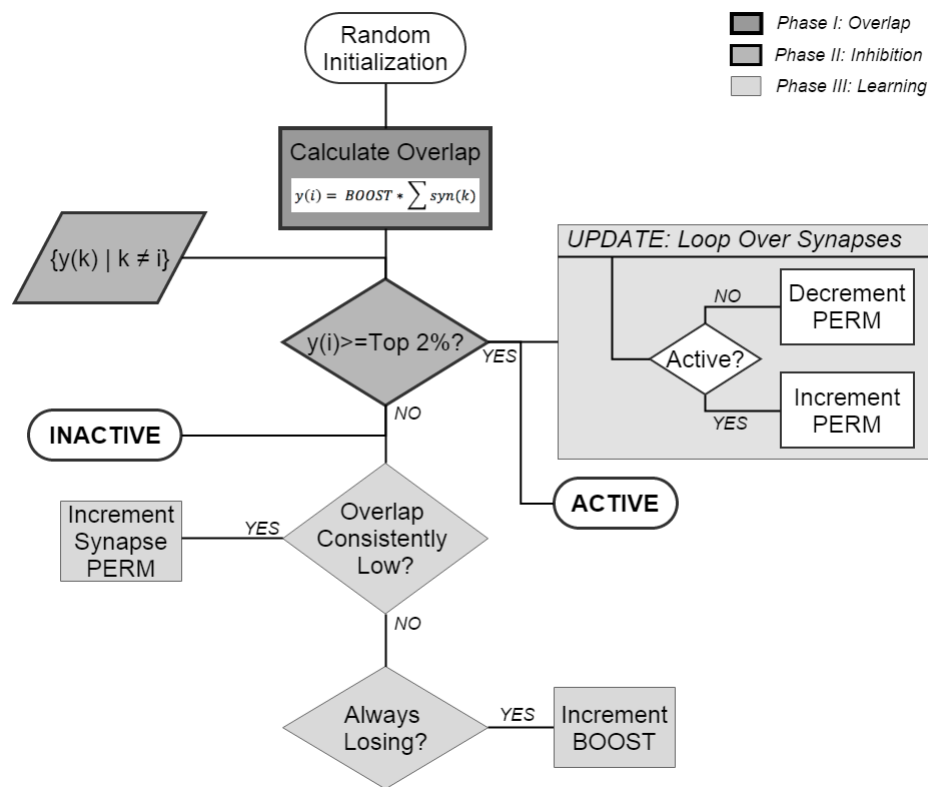


Figure 6.6: Spatial Pooler flow chart

4. Temporal Pooler

While the SP groups patterns that are spatially similar, it is the Temporal Pooler's (TP) task to group those that occur close in time i.e. in sequence. This not only allows HTM to correlate patterns that are very different yet linked to the same cause, but it also enables HTM to make predictions about what it's likely to perceive next. To use the terminology introduced earlier, this means the TP learns transitions between SDRs in time to create a high-order memory where learned transitions are not limited to a single time step. Using that memory, HTM is able to make multiple, context-rich predictions simultaneously.

Before delving into the details of the TP's algorithm, a brief divergence is necessary to clarify some ambiguous terminology. While temporal pooling is defined as

described above, both the pseudo-code in the white paper and the current software implementations fall short of that. The algorithms being currently used coincide mainly with those of a sequence learner, with a now obsolete version of true pooling implemented in code. Consequently, the process carried out by what is termed “Temporal Pooler” is to learn pattern-to-pattern sequence transitions in order to exploit learned sequences to make future predictions. Temporal pooling in the sense of achieving a stable internal state as patterns of a single sequence are seen is currently being reformulated and has not been completely implemented in usable and stable code. With that, we proceed to a discussion of the workings of the TP.

With winning columns calculated by the SP and the SDR established, the HTM network gains insight into what pattern it might be seeing at its input. What it lacks is context. Any one pattern can occur as part of a large number of sequences i.e. multiple contexts. For example, pattern B can be part of sequences ABC or XBY. As mentioned before, essential to HTM theory is the ability to predict through the learning of sequences. In Zeta 1, the Bayesian driven first implementation of HTM, this was achieved by creating Markov chains and coincidence transitional probability matrices. In CLA, sequence learning is achieved with the aid of multiple cells per column. All the cells in a column share feed-forward activation but only a subset (usually a single cell) is allowed to be active. This means that the same pattern represented by the same set of columns can be represented by different cells in each column depending on the context in which it occurs. Going back to the example, B will be represented by the same set of columns regardless of context, but its representation on the cellular level will differ depending on whether it was preceded by A or X. This representation is created through the use of the cells’ distal segments. Each of a cell’s dendritic distal segments has a set

of connections to other cells in the same region, which are used to recognize the state of the network. Cells can predict their own activity by looking at their connections and the current activations. A particular cell might be part of dozens or hundreds of temporal transitions. Therefore, every cell has several distal segments, not just one. Distal connectivity is depicted in figure 4.2.

There are three phases involved with temporal pooling. In the first phase, correctly predicted cells are activated and unpredicted columns are burst. Phase II computes each cell's predictive state. Lastly, synapses are updated in Phase III, the learning phase, by either incrementing or decrementing their permanence values. Below, we discuss the general case where online learning is turned on and occurs in tandem with inference while reminding the reader that learning in CLA can be switched off.

In Phase I, the active state of every cell of a winning column is computed and one cell per column is chosen as a learning cell. If a cell is in a predictive state due to activity on one of its sequence segments in the previous time step, it is set to active. If a learning cell contributed to this lateral activation, the cell is chosen as a learning cell too. On the other hand, if no cell in the column is in a predictive state, all the cells are activated to indicate that context is not clear or novel in a process known as 'bursting'. Finally, if no learning cell has been chosen, the best matching cell is assigned as the learning cell and a new segment is added to its distal segments. Figure 4.7 shows the flowchart associated with Phase I.

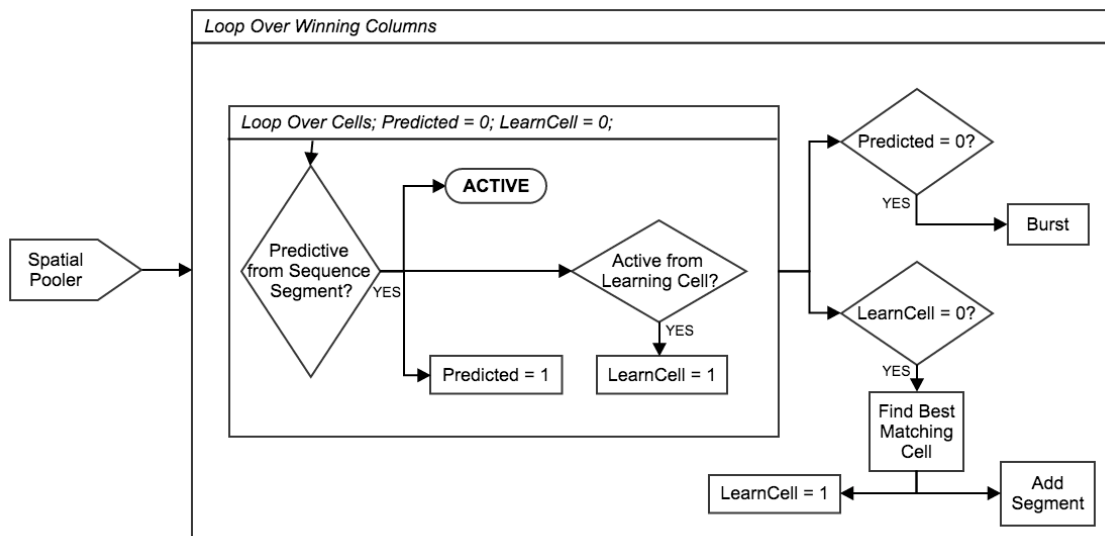


Figure 6.7: Temporal pooler Phase I

Now that all cells in the winning columns are updated, their states are used in Phase II to compute the predictive states for all other cells. For every cell in the region, distal segments tally their active connections. If any of the segments are strongly stimulated due to enough feed-forward activation in other cells, the cell transitions into a predictive state and queues up the following changes:

- Reinforcement of the currently active segment by incrementing permanence values for ‘active’ synapses and decrementing those are inactive, and
- Reinforcement of a segment that could have predicted this activation, i.e. a segment that has a (potentially weak) match to activity during the previous time step.

Phase III is where learning occurs by deciding which of the queued up updated are to be committed. On the next time step, once new feed-forward input is processed, only cells that are chosen as learning cells will have their temporary segments updates carried out. Thus, we reward segments with reinforcement only if they correctly predict

the feed-forward activation of their cell. Otherwise, if the cell ever stops predicting, we negatively reinforce the segments that were queued up for updates. Figure 4.8 shows the flowchart associated with Phase II and Phase III.

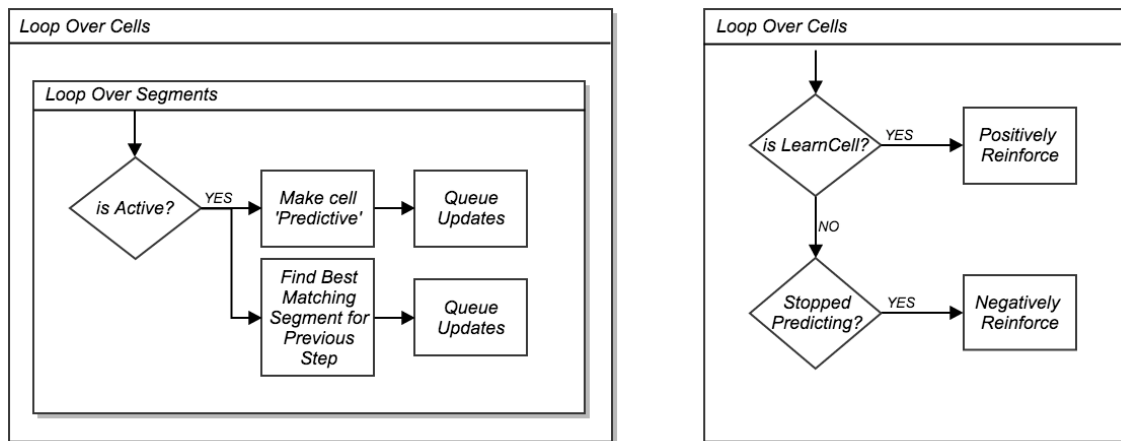


Figure 6.8: Temporal Pooler Phase II (left) and Phase III (right).

Many of the high level concepts introduced here carry over to the code implementation, but many non-biological additions have been incorporated to speed up learning. One major addition we will briefly mention here is backtracking. Backtracking is a process used as an alternative to bursting when too many unpredicted column activations take place. It occurs at two levels in the TP pipeline: After inference where predictive cells are sought out and activated (Phase I), and after prediction where the cells' lateral activity is monitored and their predictive states are computed (Phase II). Its main function is to avoid excessive bursting that can impede the network from locking in to a sequence or context by going back a few steps and attempting to detect the start of a new sequence from which it can derive context.

5. CLA Classifier

The CLA classifier is a non-biological module added to the HTM network on top of the pooling mechanisms to enable multistep prediction, whereby a set of predictions is produced one or more time steps ahead. It is not a classifier in the machine learning sense of the word and should not be confused as one. Rather, the CLA classifier's operation is more associative in nature than discriminative, in that it attempts to associate the HTM network state in the past (e.g. N-steps back) with the current value of a predicted field so that these learned associations can be used to produce predictions (N-steps ahead) by leveraging the current network state.

a. Input to CLA Classifier

The CLA classifier makes use of 4 inputs: network state, predicted field bucket index, predicted field actual value, and record number.

The network state is a binary vector produced at the output of the Spatial Pooler, Temporal Pooler, or Encoder, which makes the classifier agnostic to the source module. Typically, the CLA classifier is linked to the output of the Temporal Pooler. In this common case, the network state is a binary set of the active states at the level of TP output. Predictive states are not propagated to the classifier since active bits have the context (history of a sequence) encoded within as part of the TP operation. This allows the HTM network to produce predictions solely based on active states after utilizing predictive states as a tool for contextual encoding. In short, the active states - by design - hold all the necessary information to allow the CLA classifier to produce meaningful predictions.

The predicted field bucket index is an integer identifying which encoder bucket the predicted field at the input of the HTM network falls into. The predicted field actual

value is the raw input to the predicted field encoder. For instance, the string “Green” input in the example demonstrated earlier is the predicted field value for a Category Encoder while ‘1’ is the bucket index. Similarly, in the example for the Scalar Encoder, ‘1.5’ is the actual value with bucket index ‘2’.

Finally, the record number is a long integer computed by dividing the time stamp of the input record by the aggregation interval. It allows the CLA classifier to deal with missing records.

b. Operation

Like most other modules in the HTM network, the CLA classifier operates in a learning mode and an inference mode. These modes of operation work in tandem to create associations between network state and actual values of predicted fields that can be later used to produce multi-step predictions. Histograms are maintained and updated by every cell to calculate the likelihoods of buckets. Additionally, lookup tables mapping bucket indexes to actual values are stored by the classifier and queried to output the most likely predictions. In what follows, we discuss the workflow of these two modes.

i. Learning

During learning, every cell maintains and updates a histogram over all bucket indexes. For N-step prediction, this histogram is populated by a coincidence factor relating the activity of the cell N steps back to the activity of each bucket at the current time step. During 1-step prediction for example, a bucket that is currently active will update its coincidence factor with every cell that was active during the previous time

step, thus establishing a correlation between the current input to the HTM network and the prior network state. The coincidence factor is implemented as a rolling average and referred to as a Rolling Average Duty Cycle (RADC). Whenever a bucket is active, its RADC is updated for every cell that was active N steps prior according to the rule in (4.2). It is worth noting that a requirement for this update protocol is to continually store network status (cell activity) for future use.

$$RADC_b(t) = RADC_b(t - m)(1 - blAlpha)^m + blAlpha \quad (4.2)$$

where b is the bucket index, m is the number of time steps since the bucket was last active, and blAlpha is a parameter that controls the rate of learning/forgetting of the rolling average. Higher choices of blAlpha favour recent duty cycle values.

A second function of the CLA classifier during its learning phase is to create and maintain an association between every bucket index and an actual value to represent it. A simple approach is to assign the center of the range allocated to every bucket as its representative actual value. Better results are obtained none the less when the actual value reflects the distribution of the actual values over the bucket's range. A rolling average is used here again. Whenever a bucket b is active, the bucket actual value is updated according to (4.3).

$$\begin{aligned} bucketActVal_b(t) = & bucketActVal_b(t - 1)(1 - avAlpha) \\ & + actVal * avAlpha \end{aligned} \quad (4.3)$$

where actVal is the current actual value of the predicted field, and avAlpha is a parameter that controls the influence of a current input on the bucket's representative actual value.

ii. Inference

The inference mode of the CLA classifier computes the likelihood of each bucket using the learned RADC histograms and maps the top buckets to their predicted actual values. First, the histograms of all the active cells at the current time step are normalized to produce the conditional probability of the buckets. Therefore, with cell c belonging to the set active cells AC and b belonging to the set of available buckets B ,

$$P(b|c) = \frac{RADC_b(t)}{\sum_{i \in B} RADC_i(t)} \quad (4.4)$$

Next, the sum of the likelihoods (LS) for every bucket is calculated then normalized as seen in (4.5) and (4.6) respectively. This produces the final likelihood (L) for every bucket.

$$LS_b = \sum_{c \in AC} P(b|c) \quad (4.5)$$

$$L_b = \frac{LS_b}{\sum_{i \in B} LS_i} \quad (4.6)$$

Finally, the top most likely buckets are selected and then mapped to their actual values. These values are then coupled with the calculated likelihoods of their associated buckets and outputted as a set of multiple predictions.

6. Anomaly Detector

With streaming data and the number of sensors not only being ubiquitous but also constantly increasing, efficient tactics to analyze the large body of information and discern what is different or unusual are needed. These tactics are encompassed by what is termed anomaly detection, the practice of finding patterns in data that do not conform to expected behavior. These anomalous patterns can be point, contextual, or collective anomalies.²

Anomaly detection is fundamental to the HTM theory. In the CLA, the anomaly score provides a metric representing the degree to which each pattern is predictable. It attempts to detect both novel input patterns—point anomalies that have never been observed before—as well as familiar patterns that occur in a novel context—both contextual (singular) and collective (multiple) anomalies. This anomaly score is temporal in nature and is computed as a percentage of active spatial pooler columns that were incorrectly predicted by the temporal pooler.

$$anomaly\ score = \frac{|A_t - (P_{t-1} \cap A_t)|}{|A_t|} \quad (4.7)$$

where P_t is the set predicted columns at time t and A_t is the set of active columns at time t . A value of 0 designates a perfectly predicted input while a value of 1 indicates the input was totally unpredicted.

In addition to the anomaly score, the CLA are capable of producing anomaly likelihoods. Thresholding a single anomaly score is not a recommended practice especially with noisy data. Instead, HTM looks at past anomaly scores and compares their recent distribution against the long-term historical distribution. The anomaly

likelihood correlates with the probability that the recent anomaly scores come from the estimated distribution.

CHAPTER V

SPATIAL POOLER FOR BASIC CLUSTERING

The principle function of the SP as reviewed in chapter IV is to group input patterns in space. This fulfills two purposes. The first is to provide the HTM network with robustness against noise. The second, and more important, is to create groupings of patterns that are similar. The SP therefore lends itself naturally to clustering tasks. In this chapter, we investigate how well HTM performs in this propensity.

A. Impact of Noisy Input

The first set of experiments will focus on HTM's ability to tolerate noise at its input fields by assigning the same SDR to similar but noisy patterns. For that purpose, we created two binary data sets. The first consists solely of clustering centers. These are binary vectors of predefined lengths with random on bits spread out. The number of on bits is controlled by a sparsity parameter. The second dataset is a collection of noisy inputs created from the generated clusters. The noise is created by choosing bits at random and flipping them.

With the use of these two simple datasets, we tested HTM's tolerance to noise at different levels and measured its rate of convergence at different configurations.

1. Clustering without Bias

For the first clustering test, a newly initialized SP model is trained using the dataset of noisy inputs. The dataset is created using 5 clusters of length 256 bits and 5

generated noisy copies per center for a total of 30 inputs. The dimensionality of the input is fixed at 256. The number of columns in the SP and therefore the ratio of column count to input size is varied to evaluate its effect on convergence and clustering performance. Other free variables are sparsity level and noise level.

During training, the inputs generated are fed sequentially into the SP for a number of repeated passes. The output of the SP is a vector of the indices of the top winning columns. The size of this output is determined by the inhibition radius. For this set of test, inhibition is performed globally by retaining a maximum of 20 winning columns (2% sparsity). For the sake of readability, the output vectors are mapped into hash codes and stored for comparison.

The aim of this test is to figure out how robust the SP is when exposed to noisy input at different levels of input sparsity. The goal therefore is to check whether the SP will produce the expected clustering pattern and assign the noisy inputs to the corresponding clusters and to quantify how quick it is to do so. The stability criterion used to quantify convergence speed to the desired pattern is 10-step stability where the output doesn't vary for 10 consecutive steps. Figures 5.1 and 5.2 use colored patches to indicate the number of steps (i.e. training passes) required for SP to achieve the desired clustering at different region sizes. The number of steps shown in the plots refers to the first of the stable steps. A number of passes of 0 (darkest blue) is used to designate non-convergence or different yet stable clustering outcomes after the maximum allowed steps are exceeded.

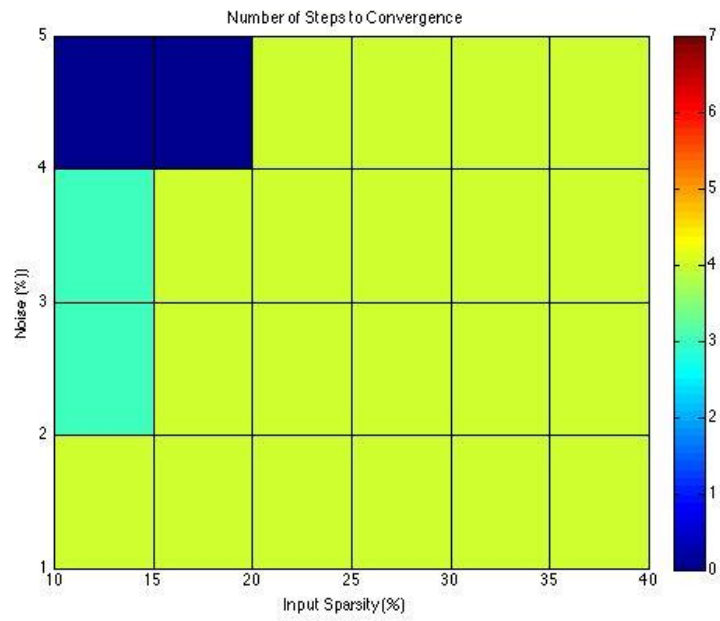


Figure 7.1: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for an unbiased 64-column HTM region.

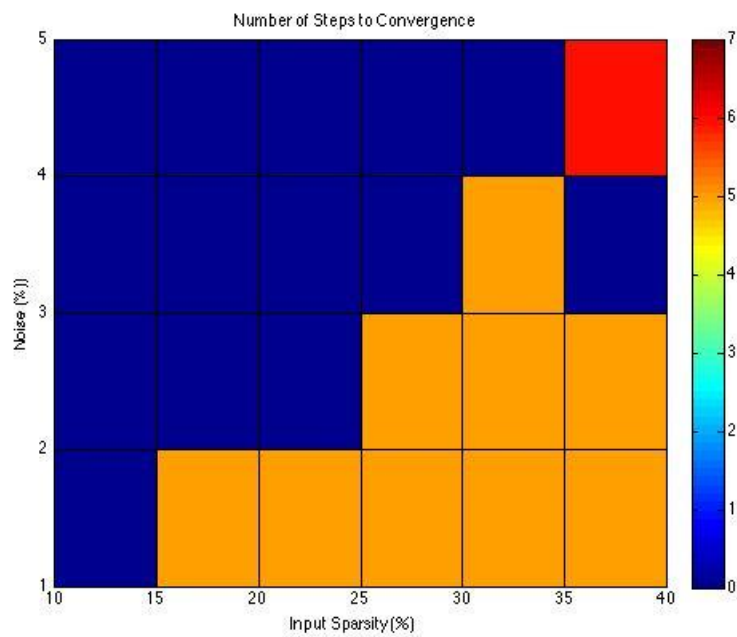


Figure 7.2: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for an unbiased 1024-column HTM region.

Looking at figures 5.1 and 5.2, it is clear that the smaller HTM region outperformed the other. Not only was the former able to converge to the expected cluster pattern at higher levels of noise but it also converged faster than the 1024 column region. This outcome seemingly contradicts the common practice of using HTM with column counts as high as 2048. Higher region sizes are claimed to be more suitable to achieve the pooling effect Sparse. This is partly true due to the fact that CLA lend themselves by design to problems with high volumes of streaming data. In this example, the size of the dataset and, more importantly, the limited number of unique clusters demands a different approach. With 64 columns and a forced output sparsity of 2%, only a single column is allowed to represent an input; as opposed to 20 columns with the 1024 column region. Consequently, the probability of having a subset of the winning columns vary and therefore the probability of obtaining varied representations are drastically diminished for the smaller network.

2. Clustering with Bias

At its core, HTM is a memory system. Every incoming input pattern is processed in the context of past learning. It is therefore interesting to investigate its pooling behavior under bias inflicted by prior training. To that end, we modify the previous clustering test by having the SP undergo multiple learning passes over the clusters alone. This will ensure that the internal representations of these clusters are stable and robust since reinforcement learning will have substantially increased the permanence values of their corresponding input connections. Figures 5.3 and 5.4 show the convergence plots for a 64-column region under 20-pass and 50-pass bias respectively. Figures 5.5 and 5.6 depict the same for a 1024-column region.

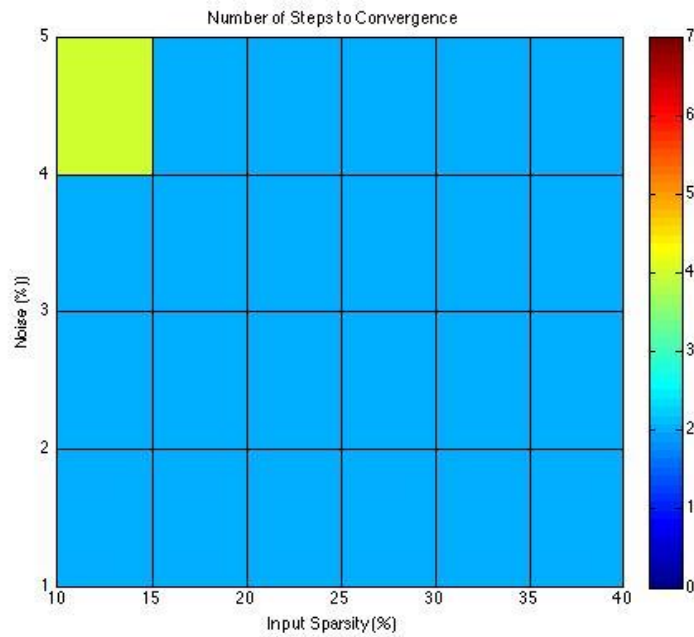


Figure 7.3: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 64-column HTM region biased using 20 passes of cluster centers.

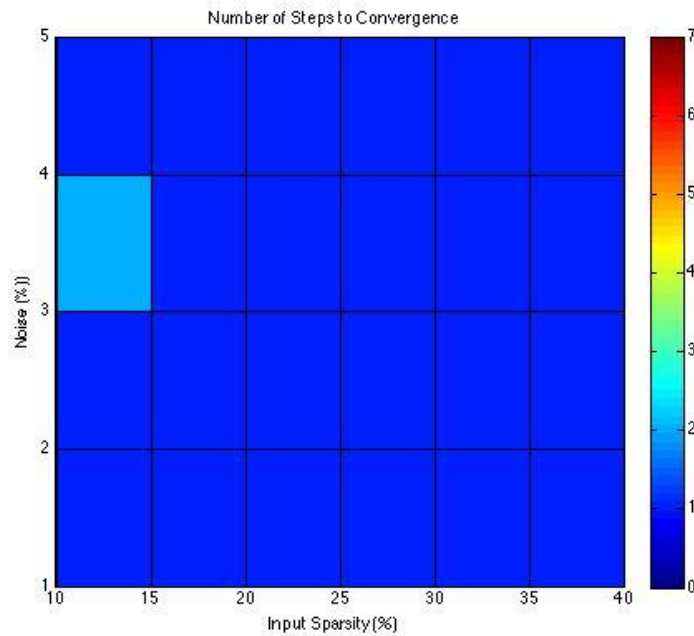


Figure 7.4: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 64-column HTM region biased using 50 passes of cluster centers.

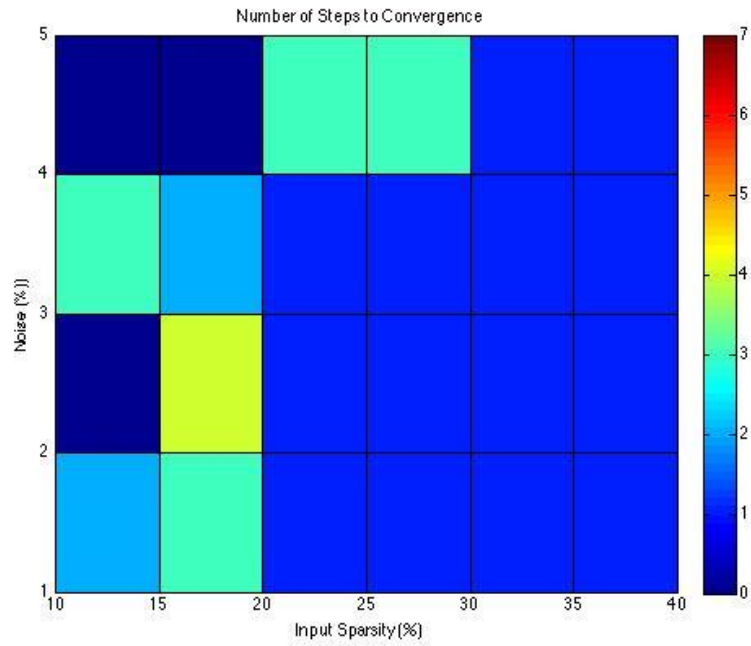


Figure 7.5: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 1024-column HTM region biased using 20 passes of cluster centers.

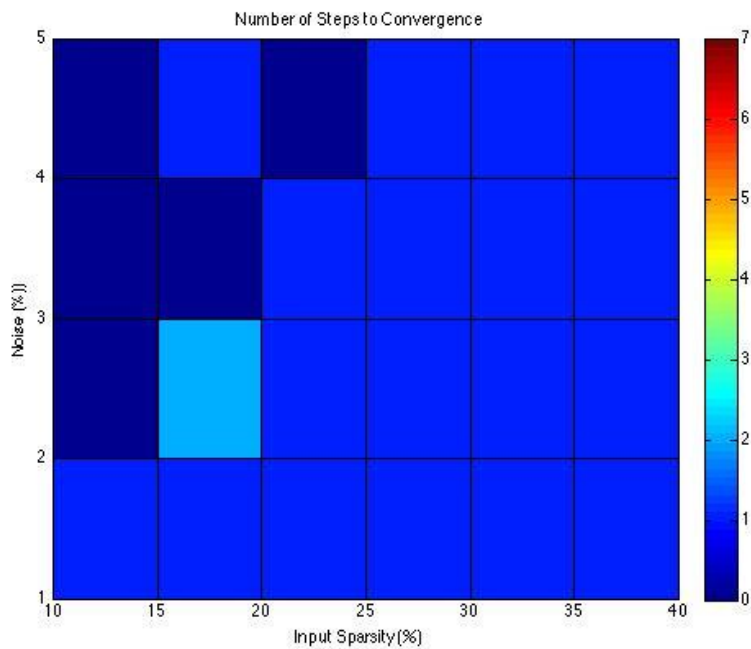


Figure 7.6: Steps to a desired stable clustering outcome at varying input sparsity and noise levels for a 1024 column HTM region biased using 50 passes of cluster centers.

As expected, prior learning had a positive effect on the clustering outcome. The biased SPs for both region sizes achieved faster convergence to the desired cluster pattern. Additionally, the SP became more noise tolerant by achieving the desired clustering at higher levels of input noise while requiring less steps to stabilize.

Finally, two interesting points deserve mention. First, the number of training passes used to bias the SP produce no added value beyond 50 passes. After the SP has learned the patterns, all representations stabilize and little to nothing is learned by further training. Second, the results of running the experiment with the same configurations can yield different results. This is to be expected considering the random connectivity and initialized permanence values of the network's synapses.

B. SP vs. Clustering Algorithms

The previous experiments show that the spatial pooling mechanisms of CLA perform well in terms of clustering highly similar inputs. This is achieved through the compound effect of inhibition and Hebbian learning rules. The first ensures that internal representation is controlled by a sparse set of strongly active columns while the second endows these columns with feature extraction capabilities by tailoring their activations to certain repeatable input patterns.

The next step in evaluating the SP algorithms is comparison with other common clustering techniques. This will allow us to test the SP's performance on more realistic scalar data and offer us the chance to utilize CLA's, which have their own pooling capabilities. For this purpose, we generate 2D clustering data by choosing 5 random cluster centers in a fixed range and spawning 20 points from each using a

multivariate normal distribution function with the centers as mean and a fixed covariance matrix. Figure 5.7 shows the 100 generated data points.

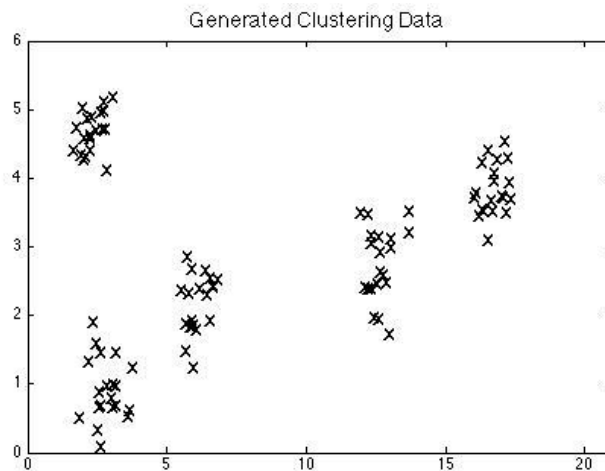


Figure 7.7: Data points generated for 2D clustering test

The first of the algorithms is k-means clustering. K-means is a vector quantization technique that partitions a set of vectors into K clusters by minimizing the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. It begins with a random selection of cluster centroids and assigns every point in a set to the closest centroid as defined by a selected distance metric (Euclidean is used here). Centroids are then recalculated as the mean of their clusters and the process is repeated until convergence. Figure 5.8 shows the result of running k-means with 5 clusters on the generated data and the effect of random initial selection on the outcome.

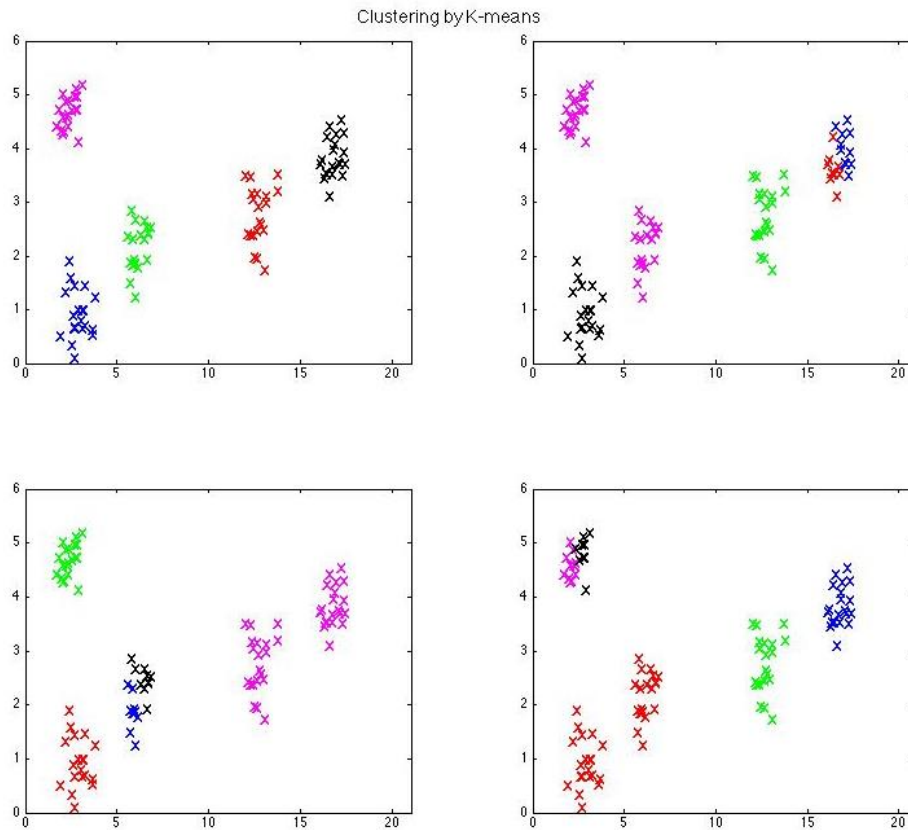


Figure 7.8: K-means clustering outcome for 4 different runs under the same parameter settings

A close variant of K-means is Fuzzy C-means (FCM) clustering. The FCM algorithm minimizes intra-cluster variance and assigns to each point a degree of membership to each centroid. Much like K-means, its iterative process produces a solution that is associated with a local minimum and that depends on the initial choice of weights. Figure 5.9 shows the result of running FCM with 5 clusters on the generated data for multiple times. Overall, the outcome is more consistent than that achieved by K-means for this sample data.

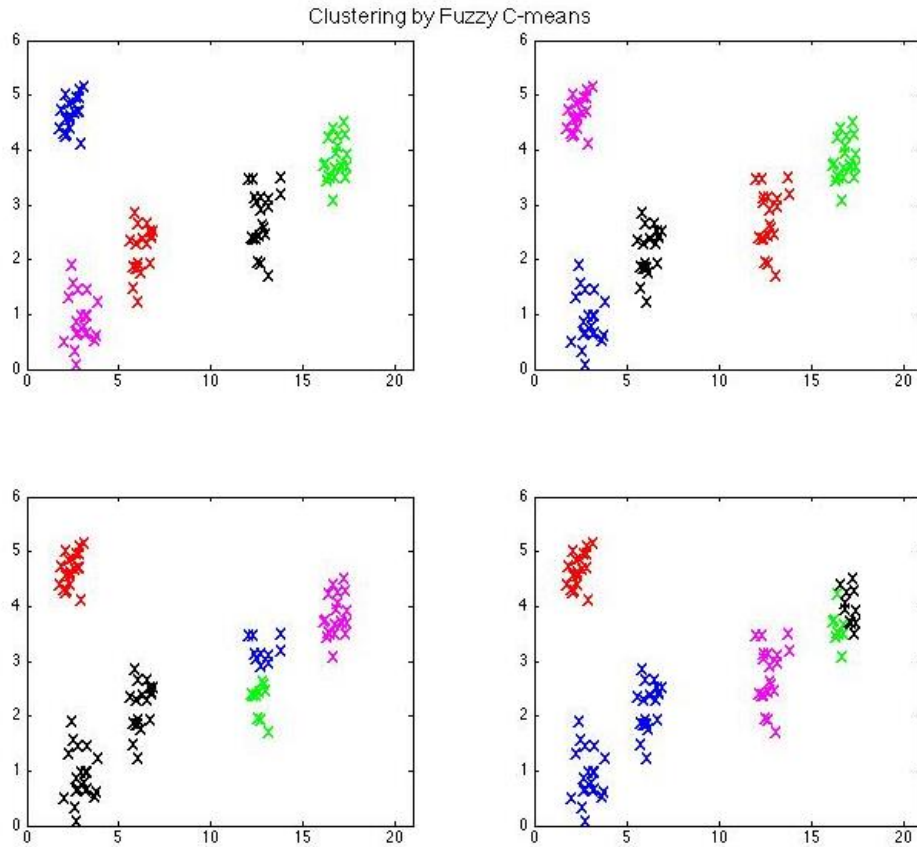


Figure 7.9: Fuzzy C-means clustering outcome for 4 different runs under the same parameter settings

Lastly, we use agglomerative hierarchical clustering (AHC) to construct a maximum of 5 clusters using distance as a criterion. Each node's height in the hierarchical cluster tree represents the distance between the two sub-nodes merged at that node. Figure 5.10 shows the dendrogram (left) produced by the clustering algorithm and the clustering outcome (right). A horizontal cut is used to find the smallest height at which the tree will have 5 or fewer clusters. The outcome of AHC is consistent and produces the expected clustering pattern.

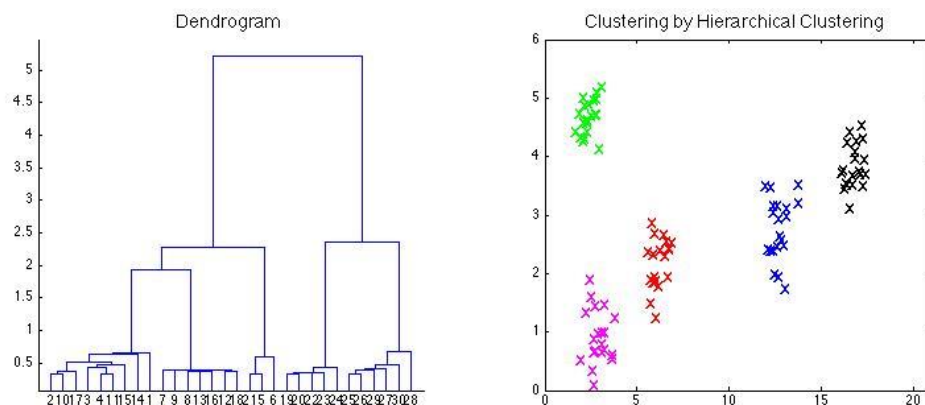


Figure 7.10: Agglomerative Hierarchical clustering dendrogram (left) and outcome (right)

Having tested some of the more popular clustering algorithms, the next step is to evaluate CLA's performance on the generated sample data. Since the data is scalar and 2-dimensional, we set up a pair scalar encoders of size 128. Scalar encoders map scalar values into binary representations by collecting similar values into bins. These bins are then represented by blocks of active bits ('1's) that shift when the difference between two scalars exceeds the resolution. The size of the blocks or windows is defined by a parameter w . With n being the length of the encoded input, the resolution associated with the scalar encoder is $n/(n-w+1)$. It is therefore evident that the clustering outcome is highly dependent on the value of w chosen. Figures 5.11 and 5.12 show the clustering outcome when using SP without and with bias. Values of w were increased until the number of produced clusters averaged at 5. Higher values produce fewer clusters by decreasing the resolution while smaller values do the opposite. The SP model used for both cases has an input grid of 16-by-16 and 64 columns in total with 2% output sparsity.

The positive impact of bias is evident by comparing figures 5.11 and 5.12. With 50 bias passes, the SP was able to produce a clustering pattern in line with our

expectation. It did so more consistently as well. Nevertheless, even with no bias, the SP still managed to perform well and produced results comparable if not superior to those of K-means and C-means on this particular data set. While this does not necessarily prove the CLA's merit as a clustering algorithm, it goes to show that SDRs do indeed possess pooling capabilities that warrant its use as a major design pillar for CLA in specific and cortically inspired learning algorithms in general.

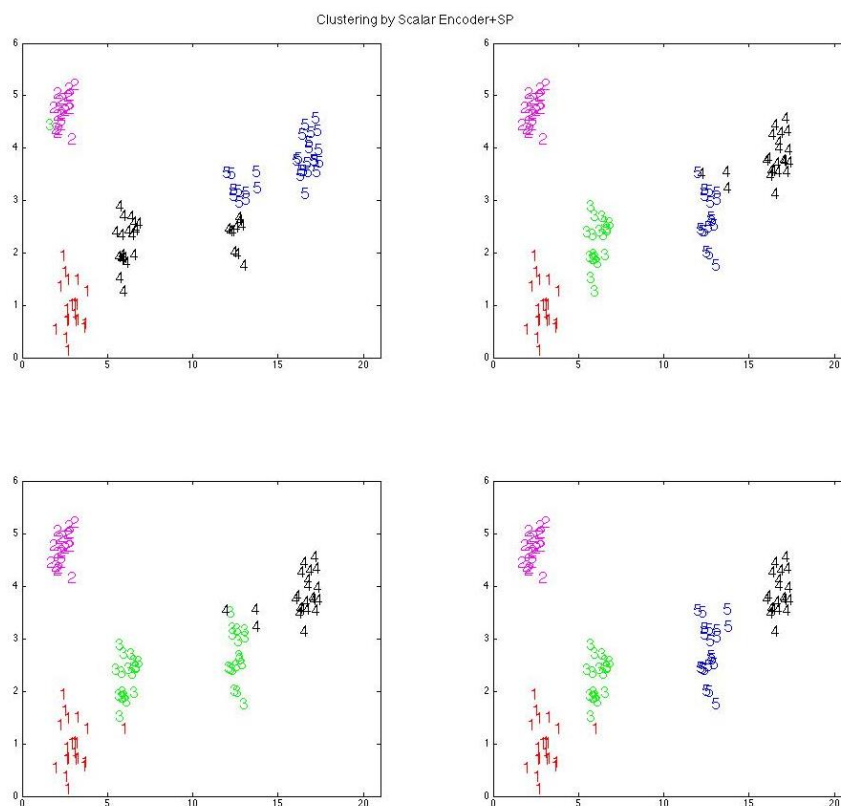


Figure 7.11: SP clustering outcome for 4 different runs with $w = 45$ and no bias passes

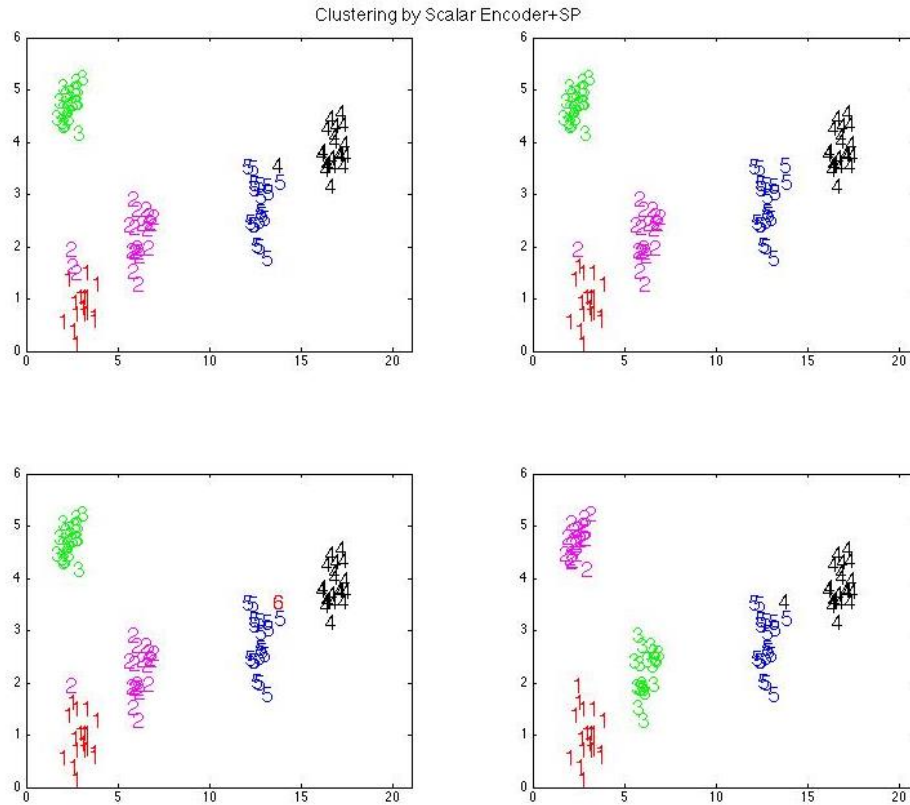


Figure 7.12: SP clustering outcome for 4 different runs with $w = 29$ and 50 bias passes

Having tested CLA's clustering abilities using a small synthetic dataset, we expand the tests to incorporate both real and larger synthetic datasets. We then employ a set of internal and external clustering validation techniques to compare HTM's performance with that of the above algorithms. We chose 2 real datasets from the UCI machine learning repository. The "Seed" dataset has 3 clusters with 70 elements each for a total of 210. "Wine" has 178 elements divided between 3 clusters with a 59-71-48 distribution. For the synthetic 2-D datasets, "SYN1" has 5000 elements divided between 15 clusters with a Gaussian distribution whereas "SYN2" has 3000 elements and 20 Gaussian clusters.

To assess the clustering performance of the algorithms discussed previously, we rely on three internal and a single external evaluation measure. Internal measures are

produced using the clustering outcome alone. The Calinski-Harabasz (CH), Davies-Bouldin (DB), and Dunn (DN) internal indices were used here and calculated using (5.1), (5.2), and (5.3) respectively.

$$CH = \frac{N-K}{K-1} \frac{BGSS}{WGSS} \quad (5.1)$$

where N is the size of the dataset, K the number of clusters, BGSS and WGSS the between and within group sum of squares respectively.

$$DB = \frac{1}{K} \sum_{k=1}^K \max_{k' \neq k} \frac{\delta_{k'} + \delta_k}{\Delta_{kk'}} \quad (5.2)$$

where δ_k is the average of the distances of the elements of a cluster k to its centroid and $\Delta_{kk'}$ the distance between the two clusters k and k' (usually defined as the distance between corresponding centroids).

$$DN = \frac{\min_{k \neq k'} \Delta_{kk'}}{\max_k D_k} \quad (5.3)$$

where D_k is the largest distance between 2 elements of the same cluster k.

Finally, we chose the Rand index (RI) as an external evaluation measure that utilizes “ground truth” labels to evaluate clustering performance. The Rand index or RI measures how similar the clusters are to the benchmark classifications. Essentially, it takes on the role of classification accuracy for supervised learning tasks.

For this set of experiments, the HTM model was configured with a number of columns equal to that of the desired number of clusters to put it on equal grounds with the other algorithms. Additionally, the columns were allowed to span the entire input. As for the input, all scalar features were normalized to a [0-1] range and processed using a scalar encoder. The size of the encoder and its window parameters were treated as free variables to optimize for. A grid search over both was subsequently conducted to find max RI for the 2 real sets. For the synthetic datasets, the size and consequent time complexity were too restrictive to perform a grid search. Parameters for these sets were therefore configured empirically. Also, since the synthetic data lacked any labels, RI calculations were excluded. Table 5.1 presents the evaluation measures for the 4 datasets using the 4 different clustering algorithms.

For the real datasets, the HTM model performed surprisingly well. With “Seed”, it outperformed all the other algorithms on the DB measure. For all other measures on both “Seed” and “Wine”, it achieved results very comparable with the rest of the clustering algorithms. Most notable was its ability to score 88% and 81.6% accuracy on the RI. Synthetic datasets, oddly, presented a different case. On both, the HTM model performed very poorly, coming in last for most measures. Partial accountability can be assigned to the lack of a proper grid search to optimize the encoder parameters. Never the less, its inability to properly perform on a synthetic data set with cleanly separated Gaussian clusters is highly problematic. More so, the variance of the results of grid search was highly suspicious. At its worse, results on the RI index for fixed encoder parameters on the same dataset varied between 56% and 91%. This seems to be due to the initialization stage and the randomness in assigning both

connectivity and permanence values. The end result is a lack of consistency in performance.

Table 5.1 Clustering evaluation with 3 internal and 1 external measure (Higher is better for all measures except DB)

		<i>BASE</i>	K-MEANS	C-MEANS	AHC	HTM
SEED	CH	<i>0.269</i>	0.315	0.314	0.279	0.307
	DB	<i>2.318</i>	2.076	2.090	2.404	2.066
	DN	<i>0.075</i>	0.080	0.080	0.105	0.091
	RI	<i>1</i>	0.890	0.900	0.919	0.880
WINE	CH	<i>0.079</i>	0.083	0.083	0.001	0.065
	DB	<i>5.4311</i>	6.0054	5.675	10.974	6.354
	DN	<i>0.189</i>	0.142	0.142	0.239	0.181
	RI	<i>1</i>	0.949	0.949	0.3427	0.816
SYN1	CH	-	10.920	22.675	22.445	7.690
	DB	-	18.575	4.123	4.196	28.615
	DN	-	0.005	0.036	0.065	0.0007
	RI	-	-	-	-	-
SYN2	CH	-	6.458	7.059	7.330	2.518
	DB	-	84.240	24.180	7.567	287.872
	DN	-	0.009	0.007	0.039	0.0012
	RI	-	-	-	-	-

C. Overlap as Similarity Measure

In this last section of evaluating HTM's Spatial Pooler, we investigate the claim that overlapping SDRs created at the SP level exhibit a strong semantic link i.e. the overlap in bits between two SDRs can be used as an indication of the similarity between their corresponding inputs patterns.

For this purpose, we employed a 1024-column SP with 41-bit long input field populated by a Scalar Encoder with $n=41$ and $w=21$. Inputs in the range [0-20] were encoded with a resolution of 1 meaning that consecutive integers in the range have

encodings that differ by a single bit. The encoded inputs were fed to the network for 10 training passes with learning turned on. This allows the SP to generate stable SDRs for each input. Finally, for each input, we calculated the overlap between its encoding and the encoding of input 0 as well as the overlap between its SDR and that of input 0 (as produced by the SP). Figures 5.13, 5.14, and 5.15 below show the recorded overlaps for 3 different runs under the same settings.

Looking at these plots, the overlap at the encoder level is a straight line as expected since the encoder resolution is set to 1, so that every time the input is incremented by 1, the resultant encoding shifts by a single bit. The SDR overlaps shown in green show a more interesting pattern. For the same configuration of the network, the SP responds quite differently with a general non-increasing trend observed in every case. Most interesting is what happens in figure 5.15 at 13, which shows more overlap with 0 than 12 does. This is problematic to the claim being studied. Figure 5.13 and 5.14 don't show strictly decreasing plots as one would expect when monitoring a similarity metric as distance from a reference point (0 in this case) is increased. While not ideal, this is forgivable considering the heuristic nature of the similarity metric being studied here. The up-tick in figure 5.15, on the other hand, proves that there are cases when SDR overlap is not indicative of similarity or distance.

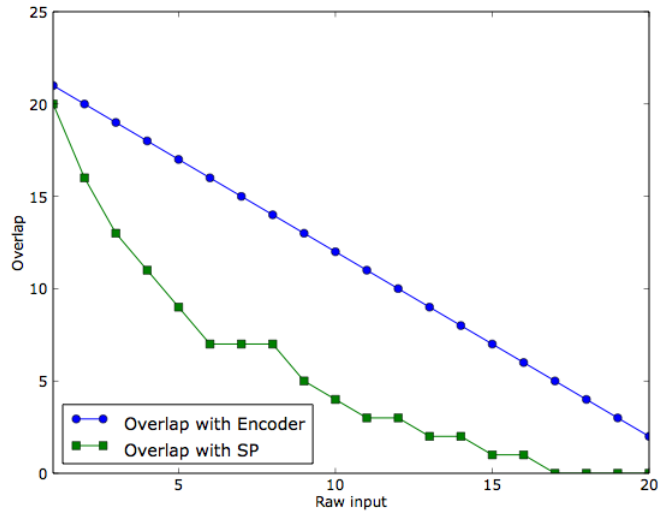


Figure 7.13: Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.

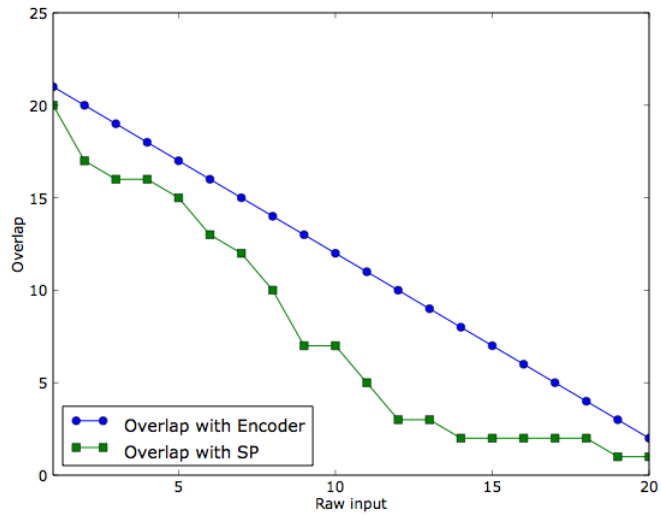


Figure 7.14: Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.

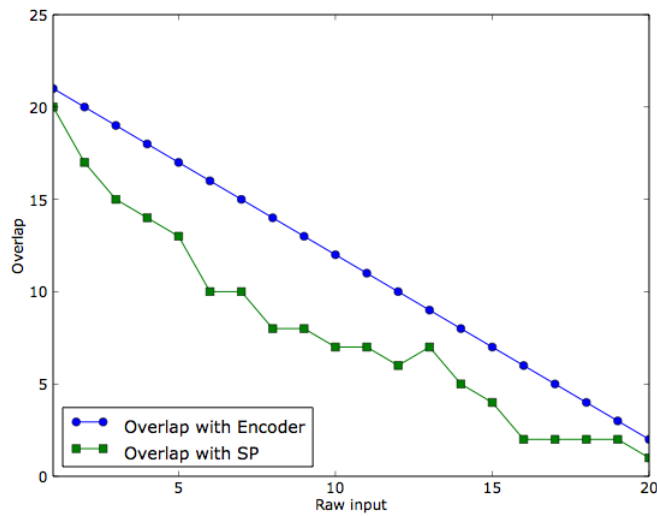


Figure 7.15: Plot comparing the change in overlap at the level of the encoder with that at the level of SP as the input is moved away from the initial value of 0.

The silver lining here is that this case seems to be a rarity. Several runs were required before the problematic up-tick was observed which makes the case for the probability of its reoccurrence being low. Nevertheless, the negative impact this has on HTM's clustering performance can't be ignored. If the overlap is not a reliable measure of similarity, then it is not viable for use in clustering or pooling patterns.

To drive the point home, figures 5.16, 5.17, and 5.18 show contour plots formed by calculating overlap at points of 2D grid with reference to the point (0, 0). These plots show the inconsistency in SP response to the same stream of data as well as the pools of inputs where SDR overlap no longer correlates with similarity or distance to the reference point. The latter is depicted by the lighter regions of the contours surrounded by darker regions. For comparison, figure 5.19 shows the similarity contours obtained using Euclidean and Manhattan distance as similarity metrics. The trend of gradual decrease in similarity as the input moves away from the reference is clear.

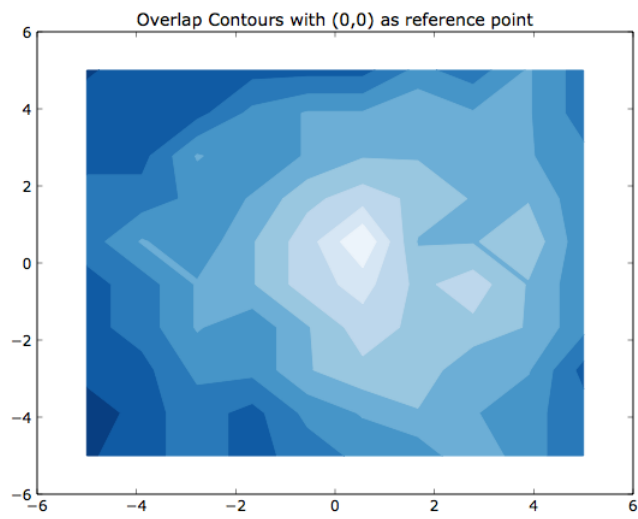


Figure 7.16: Overlap Contours for 2D inputs with reference to origin

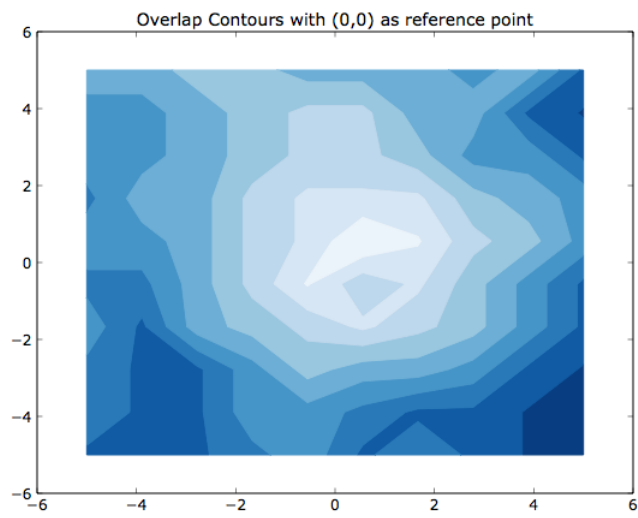


Figure 7.17: Overlap Contours for 2D inputs with reference to origin

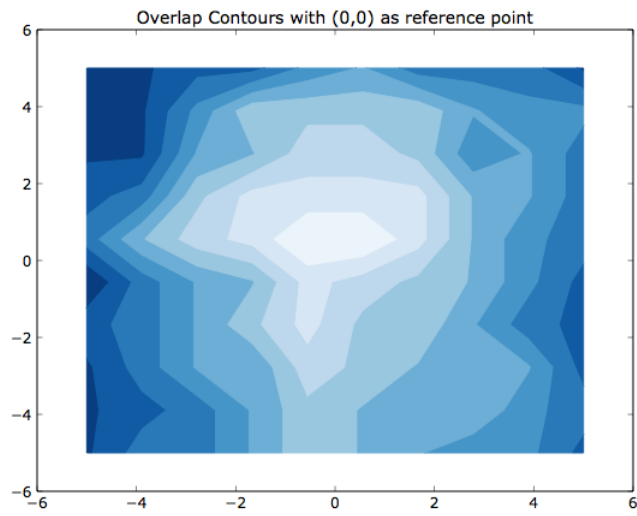


Figure 7.18: Overlap Contours for 2D inputs with reference to origin

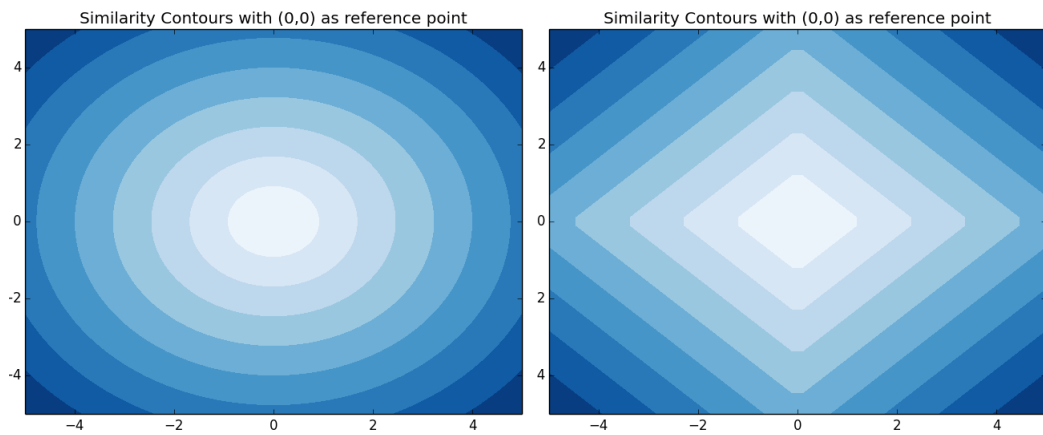


Figure 7.19: Similarity Contours for 2D inputs with reference to origin using Euclidean (left) and Manhattan (right) distance

CHAPTER VI

TEMPORAL POOLER FOR SEQUENCE PREDICTION

A crucial component of an intelligent system, be it biological or artificial, is the ability to learn sequences. Whether such a system learns a sequence of images, notes, events, or actions, the underlying learning principles are highly similar and span four categories: sequence prediction, sequence generation, sequence recognition, and sequence decision-making.³⁰ Most relevant of these categories are sequence prediction and recognition.

Sequence prediction is the process of predicting elements of a sequence on the basis of the preceding elements. More formally, given the sequence elements s_i, s_{i+1}, \dots, s_j , we want to predict s_{j+1} . Sequence recognition, on the other hand, is the process of deciding whether a sequence of elements constitutes a legitimate learned sequence after observing the consecutive occurrence of a number of its elements. This is intimately related to sequence labelling where each learned sequence is assigned a unique “label” or some form of internal representation (e.g. the activation pattern in HTM). At its current state, HTM is more evolved in its ability to do the former while the latter is the end goal of current efforts to achieve true Temporal Pooling by maintaining stable network activation for the length of a learned sequence. For our purposes, we will limit our experimental evaluations to sequence learning. Before jumping into the experimental setups, we present in what follows a high-level summary of how the CLA perform sequence prediction.

In HTM, spatial pooling takes place at the columnar level where feed-forward stimulation causes columnar activation patterns that represent a bundle of similar input patterns. The temporal pooler leverages these activations and operates at the cellular level to learn transitions between different bundles. This is made possible due to the lateral connectivity of the HTM cell. Each cell casts a net of lateral connections over its neighbourhood; sometimes the entire region. This allows the cell to monitor the activity of the neighbourhood by sampling the cell-level outputs and thus gain insight into what patterns are currently observed and the context in which they occur. Groups of cells will then attempt to correlate their collective activity with that specific context using the same reinforcement learning principles that columns use to associate with specific patterns. This happens in two ways. First, the active state of a group of cells belonging to the firing columns encodes the history or context of the sequence seen thus far. So, the cell level representation of a pattern varies based on the preceding patterns while the column level representation of the pattern remains largely stable. Second, the context rich active pattern of those cells triggers the predictive state of regional cells. With repeated exposure to specific sequences, these predictive activation patterns align with the next pattern of a sequence. Every time they align correctly, their active states are turned on, and their connections are strengthened thus reinforcing the correlation between their activity and the context they encode. The sum of these individual processes and the addition of the CLA classifier enable sequence prediction in HTM.

To evaluate HTM's efficacy in sequence prediction, we will bypass the SP and feed encoded sequences directly to the TP. We differentiate here between different types of sequences in terms of two properties. In terms of order, first order sequences are sequences where knowledge of the previous element is enough to make a prediction

about the next element i.e. $P(s_{j+1}|s_j, s_{j-1}, \dots, s_1) = P(s_{j+1}|s_j)$. The next element depends on the previous element only and is independent of any history that extends beyond the previous time step. High order sequences, it follows, are those where the next element is dependent on elements further back in time i.e. $P(s_{j+1}|s_j, s_{j-1}, \dots, s_1) = P(s_{j+1}|s_j, s_{j-1}, \dots, s_{j-i})$ for $0 < i < j$. This distinction is important to study how the multiplicity of cells per column affects HTM's ability to learn the proper context and capture enough history to make a correct prediction. In terms of certainty, we differentiate between deterministic sequences and stochastic sequences. Deterministic sequences are sequences where elements can be predicted with complete certainty i.e. there exists a 1-to-1 mapping between a subsequence of prior elements (context) and the next elements to be observed. In stochastic sequences, on the other hand, elements are generated by a stochastic process. Future elements are sampled from a probability distribution dependent on one or more previous elements. Markov chains are a prime example of this type of sequences. Finally, we will limit the experiments to finite length sequences with a discrete pool of elements. Sequence prediction of a continuous data stream will be the focus of later chapters.

A. HTM Network Configurations

For the following set of experiments, we rely on a single region network structure that excludes the spatial pooler from the module stack. To keep focus on the TP and its operation, we achieve this exclusion by tying the output of the encoder to the TP's input directly. The encoded binary input used has the advantage of being similar to the SP's output while providing control over the binary vector through encoder settings that the SP doesn't afford with its scattered and sparse activations.

As for the encoder, we implement a category encoder with shifting and non-overlapping blocks of on-bits as well as an RDSE. The configurations of both encoders are set up to provide unique encodings for the sequence elements being used. The choice of encoder will only be discussed if it directly affects performance.

For predictions, we add the CLA classifier to the top of the module stack to obtain the most likely predictions. We simultaneously leverage the TP's raw output to monitor the multiple predictions made at each step to assess the level of ambiguity of the context and gain insight into how different network settings (especially number of cells per column) affect performance.

Finally, we supplement the module stack with a sequence generator. Where needed, the generator will provide sets of sequences with different properties. The generator is designed to allow for both random and reproducible sequence sets. Additionally, it provides control over the sequence length, the number of sequences, and whether sequence elements are sampled with or without replacement. Although the sequences used in this chapter will be alphabetical, the generator can produce sequences out of any character set, alphanumerical or symbolic.

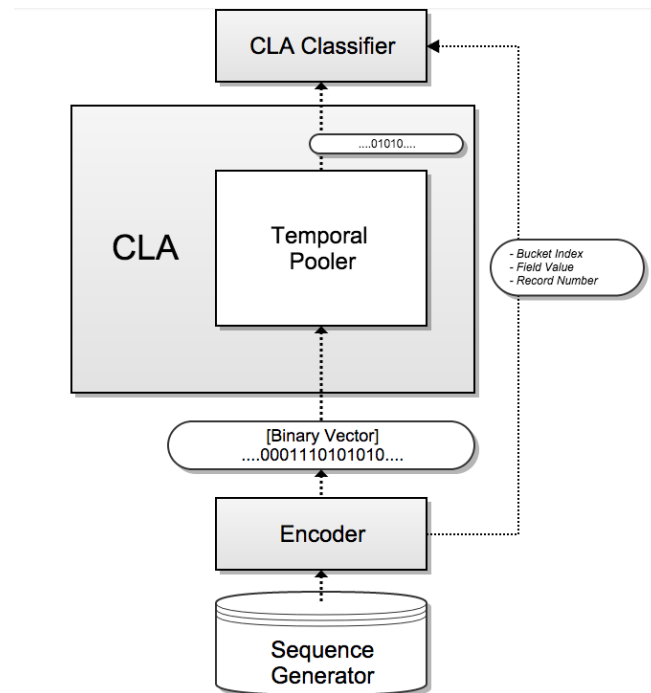


Figure 8.1: Revised HTM Model

B. First-order Deterministic Sequences

We begin the evaluation process with the simplest scenario to consider, namely that where sequences encountered are first-order deterministic sequences where knowing the current element of the sequence is enough to make a perfect prediction of the next. HTM should be able to capture sequences of this type with no errors barring any memory capacity limitations. In such cases where the network size doesn't provide enough resources to uniquely encode a large set of elements, HTM can't be expected to learn properly since different patterns will inevitably be represented identically within the network. HTM's effective memory capacity, as broadly defined here, depends on the encoder type and configurations, number of columns, and the fixed level of sparsity forced throughout its network.

To test how well HTM predicts these types of sequences, we feed it a sequence of 26 characters. The characters are first encoded in order (A then B then C...) before a

26 long sequence is formed by shuffling the alphabet. The number of cells per column is kept at 1 since that's all that is required for first-order transition learning. Table 6.1 shows the number of training passes the CLA classifier and the TP required to make a perfect prediction under varying parameter settings. Different encoders as well as different learning rates as defined by the ratio of permanence increment to decrement values and initial permanence values are used. As seen in the table, the CLA classifier which is unaffected by permanence configurations requires only a single training pass to perfectly predict the sequence. The TP, on the other hand, requires up to 5 training passes at slow learning settings while matching the CLA classifier at fast learning settings.

Table 8.1: Number of training passes needed for the TP and CLA classifier to make a perfect prediction at different configurations

Cells/col	Col	Encoder	w	n	Connected Perm	Initial Perm	PermInc	PermDec	TP Passes	CLA Passes
1	130	Category	5	130	0.5	0.5	0.2	0.1	1	1
1	130	Category	5	130	0.5	0.1	0.1	0.2	5	1
1	130	Random	5	130	0.5	0.5	0.2	0.1	1	1
1	130	Random	5	130	0.5	0.1	0.1	0.2	5	1

C. High-order Deterministic Sequences

A more interesting case to consider in the deterministic scope is high-order sequences. These are sequence where the next element can be predicted with complete confidence when knowledge of an extended history i.e. 2 or more previous elements is available. These sequences are of special interest when evaluating HTM since their prediction relates directly to the number of cells per column. In the previous first-order scenario, only one cell per column was needed to capture the single context possible for

every element of the sequence. Here, the number of possible contexts depends on the order of the sequence set. Second order sequences for example have a maximum of k^2 possible unique contexts where k is the size of the element pool from which the sequences is drawn.

In HTM, a pattern's representation is dependent on the context in which it occurs. With c cells per column and a fixed sparseness leading to N active columns, HTM is theoretically capable of learning c^N possible contexts. For example, for a typical 2048 column HTM region with 4 cells per column and 2% sparsity, 4^{40} ($1.2e^{24}$) contexts can be learned. What's interesting about HTM's context memory, therefore, is that it's not directly dependent on the element pool, and, more importantly, it doesn't force a fixed order on the sequences it admits. Ultimately, it's not the length of the context but the number of unique contexts that is most decisive when choosing HTM's cell count. In practice, there are more factors that come into play. We will attempt to highlight these factors in the experiments carried out in this section.

To test HTM's ability to capture high order deterministic sequences, we use a second-order set of sequences. A 25-state Markov model with 5 character emissions, 'ABCDE', is used to generate the set. 25 (5^2) states were chosen to represent the set of all possible 2-character histories/contexts, and emission probabilities were chosen to be 1 for one character and zero for all others. To generate second-order sequences exclusively, states were made to transition to only one other state as determined by the character it emits. For example, if state 'AB' emitted 'C', it should naturally transition to state 'BC'. With a transition matrix designed accordingly, we generated the 7 six-character long sequence set in figure 6.2.

```

['D', 'B', 'A', 'D', 'E', 'D']
['A', 'B', 'A', 'D', 'E', 'D']
['E', 'D', 'B', 'A', 'D', 'E']
['A', 'D', 'E', 'D', 'B', 'A']
['B', 'A', 'D', 'E', 'D', 'B']
['D', 'E', 'D', 'B', 'A', 'D']
['C', 'B', 'A', 'D', 'E', 'D']

```

Figure 8.2: Second-order Deterministic Sequence Set

The character set is encoded and fed directly to the temporal pooler. For this setup, a 5-column region is used with a sparsity of 20% (1 active column). The sequences above are fed to the TP with learning enabled for 10 training passes. A reset to the TP is performed after every complete sequence. This resets all internal states to avoid any learning between different sequences. A classifier is additionally used to produce best predictions and is configured with a very low *alpha* value to effectively disable forgetting.

For evaluation, two performance metrics are employed. The first is devised to track the performance of the TP's multiple simultaneous predictions. Here, a correct prediction is rewarded partially depending on how many other false but simultaneous predictions are made. For example, if 'A' is seen and the TP predicts 'ADE' from the previous time step, the prediction score is incremented by 1/3. This allows us to quantify the uncertainty of the network's predictions under different configurations. The second metric is more straightforward. It is a prediction score that is tallied using the output of the CLA classifier. Correct predictions increment this score by 1. With both metrics, scores are only calculated after enough elements have been observed. For second-order sequences for example, these scores are calculated for the third sequence element and above. This is to ensure that the network isn't penalized for incorrect predictions made when not enough information is available. Figure 6.3 shows the two

metrics scaled to 100 as the number of cells per column is varied in a network that is fed the sequence set. Figure 6.4 shows the performance for 3 (left) and 4 (right) cells per columns as the number of training passes is changed.

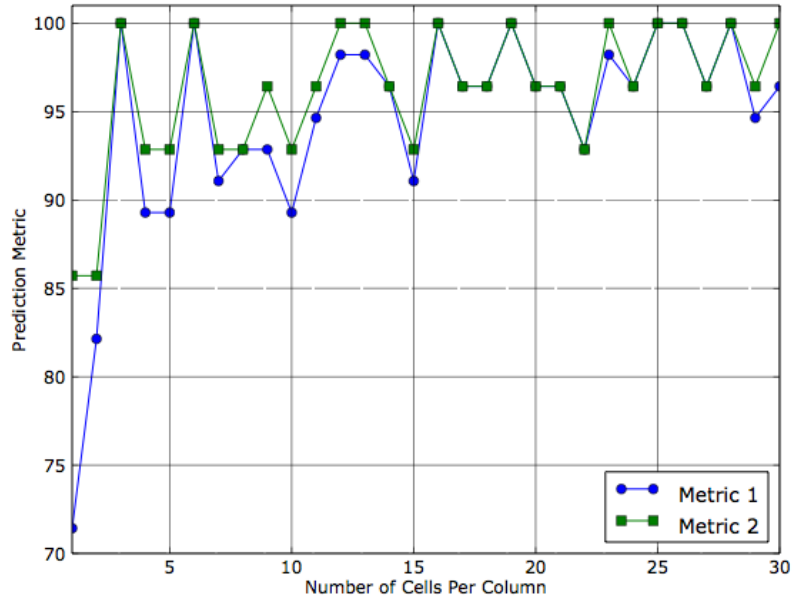


Figure 8.3: Prediction performance with varying number of cells per column

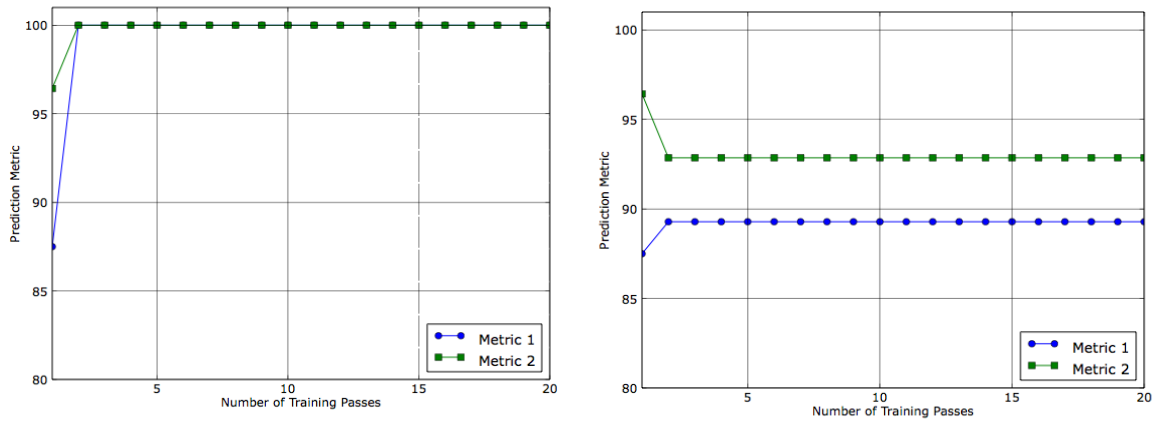


Figure 8.4: performance for 3 (left) and 4 (right) cells per columns for different numbers of training passes

As seen in figure 6.3, the network was able to perfectly predict the second-order sequences after 10 training passes when columns were created with 3 cells. To validate the consistency of that performance, we fix the number of cells per column to 3 while varying the number of training passes. The result can be seen in the left panel of figure 6.4. It takes only 2 training passes to achieve the noted performance. Above that, the number of training passes doesn't cause any improvement or degradation in the performance. Unfortunately, the same can't be said of the number of cells. While a configuration of 3 cells performs perfectly, increasing the cell count results in unexpectedly inconsistent results. This is further demonstrated in the right panel of figure 6.4 where increasing the training passes for 4 cells per column did not alleviate the performance drop.

The results are cause for concern in a number of ways. First, if the number of cells per column is to be interpreted as the HTM network's memory capacity, then it is highly problematic that small increments in that capacity cause its prediction performance to significantly fluctuate. The practical consequence of this behaviour is that any HTM network needs to be exactly tailored to the data it's being applied to. Parameter searches to accommodate this sort of inflexibility in the design are likely to be computationally expensive and impractical. The second and more important observation is that the network achieves perfect prediction at a configuration that is lower than expected while suffering performance loss at higher settings. To demonstrate that, we tabulate the number of unique contexts in which every character in the set occurs in table 6.2. With only 3 cells per column, characters A, B, and D cannot be assigned unique representations for every context. In fact, by monitoring the activation patterns during the single testing pass, it is clear that these characters are achieving

similar activations to different contexts. To understand why it is still possible for HTM to perfectly predict the sequences, we need to consider how contexts are being learned on a lower level. As discussed earlier, every HTM cell has a collection of distal segments that monitor activity in the cell's neighbourhood. It is at the level of these segments that learning happens. This means that, as learning progresses, it is the segments themselves that align with specific activation patterns in the HTM region. Ultimately, context learning happens at the level of these segments. At the cellular level, the activity of segments is OR-ed. As a result, a cell will fire predictively to multiple contexts. So why did the network still perform so well? To answer that, we look at the table 6.2 again for the number of possible future characters each of the character can see in the set we used. While the number of possible contexts sets a maximum on the number of unique representations we expect to the network to assign to a sequence element, the number of elements that will possibly follow sets a minimum. For example, if A is always followed by C or E, then it is enough for the network to represent A in only 2 unique activation patterns. This explains why 3 cells per column are able to perform well. As demonstrated in the right most column of table 6.2, the set is very close to being a first order set. While 3 cells aren't enough to assign a unique representation for every context, they are adequate to confidently predict the next element.

Table 8.2: Number of unique contexts in which every character occurs

Character	# of unique contexts	# of possible futures
A	5	2
B	5	1
C	1	1
D	6	2
E	3	1

The problem remains that even at 29 cells, the network still made prediction errors, which brings HTM's ability to encode context and learn transitions into question. The main contributor to this seems to be cases where intersecting contexts that lead to different futures are represented similarly when there's ample capacity for separate representations. This falls down to the lateral connectivity and the lack of a process to punish behaviour that leads contexts with diverging futures to be lumped together by the same cell, thus losing context resolution. Put more simply, the reader can view the cell as a context pooler. When allowed to pool--under the same representation--contexts that overlap or intersect at one point before diverging, this can cause loss of context and inevitably lead to incorrect predictions.

This issue is resolved at higher column counts. Larger HTM regions at the 2048-column and 2% sparsity seemingly reduce the negative effect of the erroneous "context pooling" as depicted in figure 6.5 where perfect prediction is achieved and maintained for any cell count above 3. We speculate that this is a byproduct of sparsity at high columnar counts where the probability of assigning similar representations to different contexts is lowered. How effective this approach is at achieving higher contextual resolution as defined here is an open question. A point of contention that remains is the impracticality of using HTM networks of such size for simple sequence learning as conducted in this experiment. Considering the high computational demands of the CLA, the compromise between performance and efficiency works against HTM.

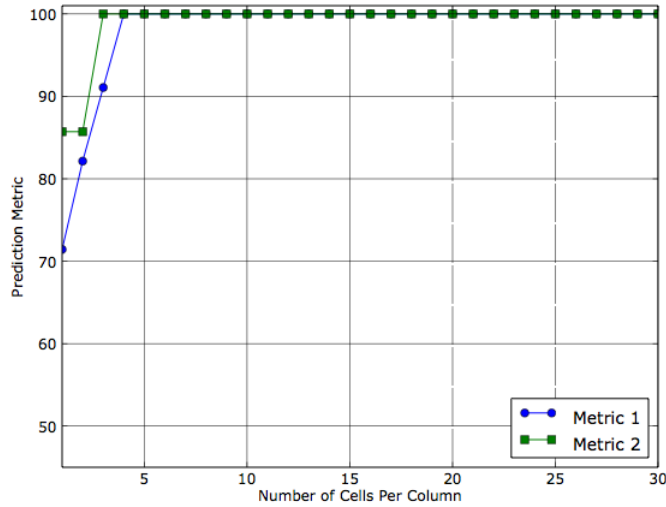


Figure 8.5: Prediction performance for a 2048 column HTM region with 2% sparsity as a function of cells per column

D. Stochastic Sequences

For the last scenario, we consider stochastic sequences where elements are generated from a probability distribution. Here, contexts or sub-sequences can lead to multiple successor elements. Therefore, measuring the prediction accuracy of HTM when observing such sequences is not helpful. Instead, we draw inspiration from Markov models that produce similar sequences. Specifically, we want to test whether the CLA classifier under the right configurations will be able to capture the distribution of the transition probabilities of the sequence set. For this purpose, we employ a 6-state Hidden Markov Model (HMM), 1 starting state and 5 states representing the set. The model is configured with single emissions for every state and an arbitrary transition matrix. We then use the model to generate 6 character long sequences. The result is 145 unique sequences. The sequence set is then fed back to the HMM to estimate the transition matrix based on this smaller set. Equations (6.1) and (6.2) show the original and estimated transition matrices for character states respectively.

$$T_{orig} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0.2 & 0 & 0.8 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0.2 & 0.1 & 0.7 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (6.1)$$

$$T_{est} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0.3577 & 0 & 0.64230 & 0 & 0 \\ 0 & 0.2 & 0 & 0.6094 & 0 \\ 0 & 0 & 0.3740 & 0.4134 & 0.2126 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad (6.2)$$

Since the HMM model generates first-order chains, we feed the 145 sequences into a 5 column HTM network with a single cell per column; higher column counts didn't significantly affect the results we sought. The alpha parameter of the CLA classifier, which controls the rate at which the classifier forgets past entries, is varied while the likelihoods of predictions is recorded and assembled into a transition matrix. The L2-distance between the resulting matrix and T_{est} is then computed as an error metric and tabulated in in Table 6.3 for the different alpha values. The results show that, as the CLA classifier is made to forget less, it perfectly captures the transition information of the HMM model. This is expected since HTM employs a very similar algorithm to construct its DCMA matrix especially with small networks and non-overlapping representations. With the use of larger HTM regions and overlapping representations, the matrices start to naturally diverge. At 2048 columns with 2% sparseness and overlapping RDSE encoding, the error increases to 1.065 and the resulting matrix is noticeably different.

Table 8.3: Error between HMM estimated transition matrix and CLA classifier transition matrix

Alpha	Error
0.1	0.804
0.01	0.183
0.001	0.019
0.0001	0.002
0.00001	~0.00

CHAPTER VII

CASE STUDY: HTM IN CLASSIFICATION AND ANOMALY DETECTION

Having put the individual components of HTM through their paces, we turn towards the bigger picture with a focus on evaluating the entire framework. We will do so in the context of different machine learning tasks, namely classification and anomaly detection. Where HTM doesn't lend itself naturally to the task, we will adapt it to said task by leveraging the metrics it outputs. Classification is a prime example of that scenario. Lastly, while simple datasets lent themselves to the toy evaluation experiments in previous chapters, we introduce here real world datasets that offer more complexity and allow us to formulate a better understanding of how the current version of HTM and its CLA perform with harder tasks.

A. Case Study I: Smartphone Based Gesture Recognition Using HTM

To evaluate HTM as a classifier, a custom built dataset for gesture recognition was used. Instances of the dataset were generated using the Skywriter framework, a gesture tracing and recognition framework for smartphones. Skywriter uses on-board inertial sensors with a combination of kinematic models of the human arm and Bezier curve fitting to estimate the trajectory of a user's hand in motion. The result is a sequence of 2D coordinates (Bezier control points) representing the trace of a user-made gesture using a smartphone.

Sixteen gesture classes were used for creating the Skywriter dataset consisting of 10 digits and 6 basic shapes. The dataset was created with the help of 15 volunteers. Each participant provided 5 instances of each gesture class for a total of 80 instances per user and 1200 gesture instances in total. Figure 7.1 shows samples of the created gestures from one of the users.

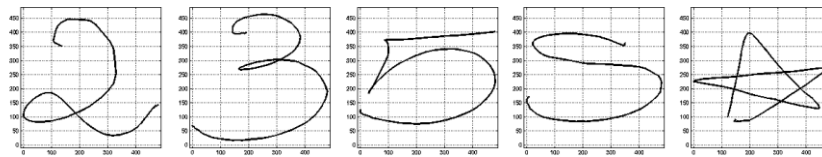


Figure 9.1: Samples of gestures traced by a single user

1. *Experimental Setup*

a. Learning Scenarios

The Skywriter gesture recognition framework was evaluated under 3 distinct learning scenarios: user-dependent, user-independent, and mixed. Each scenario demands a specific partitioning of the dataset into training and testing subsets.

In user-dependent learning, only data from a single user is used. This simulates the real life scenario where a recognition system runs a user's personal device to which only the user has access (e.g. transparent authentication for mobile platforms). In this case, both training and testing data points are extracted from the user's contribution to the dataset. This has the advantage of providing the recognition system with samples that are less varied and therefore easier to learn. On the other hand, it often suffers from having a very limited amount of data to work with due to the inconvenience of tasking the user with providing abundant data samples.

In user-independent learning, the recognition system learns from data provided by other users. This data is referred to as “community” data. Testing is still conducted using the user’s own samples. This approach no contribution from the user for initial setup of the system which is preferable from a user experience point of view. On the other hand, the data used for learning typically exhibits large variances that reflect the specific signature or usage patterns of different user, which can render learning a more difficult affair.

The third and final scenario is a mixed learning scenario. This is a hybrid approach that leverages both personal and community data to train the system. Most commonly, the community data is used to setup the system so that it can work out of the box—in a plug-n-play manner—while personal data is used for fine tuning for a user tailored experience.

b. Network Structure

CLA does not provide support for multi-class learning. We therefore adopt a learning scheme similar to HMM training for multi-class tasks by assigning each of 16 separate HTM regions to a single class. Each region will learn from data points corresponding to its own class while testing all regions is performed on the same set of data.

Numenta recommends HTM regions with a 2048 columns, 32 cells per column, 2% sparsity configuration. This was used as an initial configuration with a number of model parameters chosen using Numenta’s Swarming tool, which adapts a particle swarm optimization to search for optimal parameter values. Unfortunately, the tool was created with large continuous data streams in mind and only searches for a

small subset of parameters including encoder length and block size. Considering how extensive the list of parameters of an HTM region is and how they can all interact to achieve different run results, this scenario is far from ideal and forced us to resort to manual parameter selection in most experiments. We elaborate on this in the next section when results are presented.

All components of the HTM stack were used including the classifier, with a scalar encoder chosen to encode the integer sequences of the Skywriter dataset. During training each region is presented with an element of the training sample with learning switched on. The element is encoded and fed into the SP, TP, and finally the classifier. At the end of the sequence, a reset is initiated to avoid learning inter-sequence transitions. During testing, regions are exposed to the same test sequence. With each presented element, an anomaly score and a prediction error are computed. These metrics are accumulated along the sequence and averaged at its end. They are used as performance metrics to gauge each regions response to a presented test sequence and are used to assign the sequence to the region that outputs the minimum value. Learning is turned off during this testing phase for all modules of the HTM stack to avoid sequences from other classes being learned by the class specific regions. Figure 7.2 depicts a flowchart for the experimental setup used.

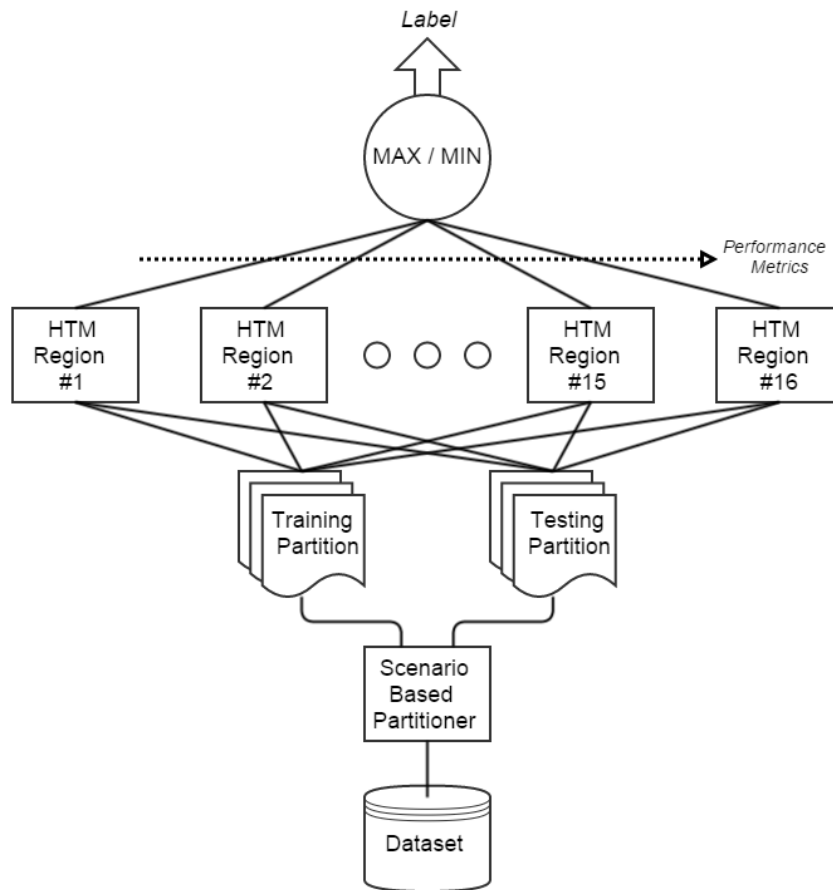


Figure 9.2: Flowchart displaying the experimental setup for HTM based classification

2. Results

Based on the experimental strategy discussed above, we proceed to testing the HTM-based classifier using the Skywriter dataset. This is performed under 3 learning scenarios. For comparison, we include results achieved through the use of a set of common and popular learning algorithms: Support Vector Machines (SVM), Template Matching, K-nearest Neighbor (KNN), and Hidden Markov Models (HMM).

For SVM, we opt for a multiclass kernel based implementation using a radial basis function (RBF). The sigma of the RBF kernel and the soft margin parameter are selected using a grid search.

The HMM setup is similar to one depicted in figure 7.2. The number of states of models is left as a free variable. The number that achieves the best performance for each user is the one selected. Other parameters like the number of iterations used in the Baum-Welch algorithm and the degree of oversampling are dependent on the learning scenario.

Template matching is conducted by averaging training samples then comparing the test samples to that mean vector. Euclidean distance is used as a distance metric, which proved to be comparable or superior to the alternatives.

Finally, the Nearest Neighbour algorithm is implemented with 3 values of K which dictates how many neighbours are used in classifying the test sample.

While the dataset was created using samples from 15 users, we only present the test results achieved for 5 users due to the time cost associated with running HTM and the other algorithms presented. Community training sets were still created from all 15 users.

a. User-Dependent

For this scenario, each user's data is partitioned using 5 fold cross validation into training and testing partitions. Specifically, 64 training samples and 16 testing samples. Table 7.1 shows the results for all considered learning algorithms with HTM to the far right. Most algorithms performed well with average accuracies above 90% with SVM being a clear winner. HMM showed unexpectedly poor performance which we speculate is related to the very low number of training and testing samples.

Oversampling the training set and varying Expectation Maximization (EM) iterations did not offer significant improvement.

HTM, on the other hand, struggled for most users with classification based on cumulative prediction error performing very poorly with 52.75% average accuracy rate. This was expected since prediction error here is calculated as the sum of squared errors between the original trace and the predicted trace, a basic type of template matching methodology that doesn't lend itself to character recognition. Based on cumulative anomaly score, it fared better with user 3 achieving rates as high as 93.75% beating out most of the other algorithms. Unfortunately, the same didn't hold true for the rest of the users where inconsistent performance was observed, some as low as 72.5%. Overall, at an average rate of 81%, HTM performed rather poorly.

Table 9.1 : Classification accuracy (%) for 5 users under the user-dependent scenario

USER	SVM	TEMPLATE	1-NN	2-NN	3-NN	HMM	HTM*	HTM**
1	96.25	92.5	91.25	91.25	95	68.75	72.5	43.75
2	97.5	92.5	96.25	96.25	95	77.5	88.75	63.75
3	97.5	91.25	92.5	92.5	93.75	72.5	93.75	56.25
4	95	95	92.5	92.5	91.25	70	73.75	52.5
5	92.5	88.75	87.5	87.5	90	68.75	76.25	47.5
AVE	95.75	92	92	92	93	71.5	81	52.75
* Accuracy based on minimum cumulative anomaly score								
** Accuracy based on minimum cumulative prediction error								

b. User-Independent

Here, no data from the user is used for training. Instead, the samples collected from all other users are aggregated into a training set. We refer to this set as the community set. Testing is carried out on all 80 of the target user's samples.

Again, the learning algorithms chosen performed very favourably with SVM coming out on top as seen in table 7.2. A majority of them benefited from having access

to community data as well. That's especially true for HMM, which made a significant jump in accuracy to become comparable to the other methods with its 91.75% average classification accuracy.

HTM fared even worse in the user-independent case revealing a trend contrary to what was observed by the other algorithms. While almost all other learning algorithms experienced an increase in accuracy rates aided by the larger number of diverse training samples, HTM suffered for it which brings its generalization abilities into question. More on that in the discussion section below.

Table 9.2 : Classification accuracy (%) for 5 users under the user-dependent scenario

USER	SVM	TEMPLATE	1-NN	2-NN	3-NN	HMM	HTM*	HTM**
1	97.5	92.5	96.25	95	95	88.75	75	42
2	100	97.5	98.75	97.5	98.75	92.5	81.25	51.25
3	93.75	82.5	91.25	91.25	91.25	88.75	75	52.5
4	98.75	88.75	98.75	97.5	97.5	97.5	77.5	46.25
5	96.25	91.25	93.75	95	92.5	91.25	67.5	41.25
AVE	97.25	90.5	95.75	95.25	95	91.75	75.25	46.65
* Accuracy based on minimum cumulative anomaly score								
** Accuracy based on minimum cumulative prediction error								

c. Mixed

In this final scenario, we merge the previous two approaches by partitioning the user's data into training and testing sets, then supplementing the training set with the community set. This grants the system the advantage of having a higher number of sufficiently diverse training samples and having access to the user's own unique samples.

Table 7.3 shows the classification accuracies for this scenario. SVM achieves a remarkable higher accuracy of 99% thus outperforming itself in the previous scenarios

making the case for the merit of the mixed approach. The other algorithms show a similar trend but to lesser effect with HMM remaining comparable despite lagging drastically in the first scenario.

In this scenario, HTM's poor performance persists. Prediction error based accuracy improves compared to the user-independent case, but at 49.5% it becomes clear that this metric is not reliable for this sort of classification task. Anomaly based classification persists at 70+% average classification rate which is not competitive with other algorithms tested.

Table 9.3 : Classification accuracy (%) for 5 users under the user-dependent scenario

USER	SVM	TEMPLATE	1-NN	2-NN	3-NN	HMM	HTM*	HTM**
1	97.5	91.25	96.25	95	95	92.5	68.75	41.25
2	100	98.75	98.75	97.5	98.75	100	73.75	57.5
3	98.75	85	97.5	97.5	95	92.5	82.5	60
4	100	90	98.75	98.75	97.5	100	77.5	42.5
5	98.75	92.5	92.5	95	92.5	93.73	72.5	46.25
AVE	99	91.5	96.75	96.75	95.75	95.75	75	49.5

* Accuracy based on minimum cumulative anomaly score
 ** Accuracy based on minimum cumulative prediction error

d. Discussion

Despite a promising start in the user-dependent scenario, HTM's overall performance was subpar given the fact that the dataset itself was a relatively easy one as evident by the performance of the more basic classifiers tested. Model optimization is complicit in this result. With a majority of learning algorithms, model parameter selection is essential and is usually performed via brute force methods like grid searches or more guided techniques like particle swarm optimization (PSO). We saw an example of that with SVM testing where we optimized for two model parameters to achieve the

impressive rates detailed in the tables above. While HTM's code base is accompanied by a PSO, it is not tailored to classification tasks like the ones carried out here since classification itself is not an inherent capability of HTM. More importantly, the PSO implementation fixes certain parameters like column number and cell number to values that the Numenta team has had the most success with. Unfortunately, configuring the HTM regions with the recommended values yielded very poor results. Instead, we had to resort to manually changing the parameters until good enough performance was achieved. The results above were achieved with HTM regions with 100 columns, 32 cells per column, and 40% sparsity; far from the recommended settings. This was a very laborious task and is easily one of the more limiting factors of deploying HTM. This becomes a more evident concern when considering the number of parameters that directly affect performance. A minimum of ten parameters was identified. Considering the amount of computations executed in every region throughout the HTM stack, model parameter selection presents itself as a significant undertaking with even small regions requiring days to optimize.

Beyond difficulty in model configuration, HTM seems to exhibit undesirable behaviour at the algorithmic levels. While the other algorithms show a general trend of improving from one scenario to the next. HTM does the opposite. This pattern becomes more interesting when we note that an ability to generalize from learned data is required for performance to improve from the user-dependent scenario to user-independent and mixed scenarios. The other algorithms made great use of community samples to create more general models of the learned data, which allowed them to handle novel test data quite well. HTM, on the other hand, seems to degrade in performance when tasked with making inferences about data beyond what it was explicitly taught. In fact, when tested

for learning accuracy (i.e. identical training and testing sets), all but one user achieved a 100% rate with the other user achieving 98.75%. Therefore, it can be concluded that over-fitting is at play. Whether the over-fitting is a result of the non-optimal models selected or an inherent flaw in HTM is up for question.

B. Case Study II: Irregular Breathing Detection in CPAP Assisted Patients Using HTM

The second case study we investigate here focuses on anomaly detection, the core feature of HTM. HTM performs anomaly detection online—as data samples are fed into its model—as opposed to more conventional batch or post processing methods. To achieve that, HTM leverages its memory model to learn the “normal” patterns in the data stream so that when anomalous behavior occurs, it is able to capture it in or ahead of time. This online anomaly detection mechanism works best with data that reveals clear normal patterns for an extended period of time before an irregularity causes it to diverge clearly from its established patterns. Failures in mechanical systems or DoS attacks on servers are prime examples. Data of this nature has been used by Numenta to showcase HTM’s anomaly detection performance. In this case study, we subject HTM to data that exhibits more complex anomaly patterns. To that end, we procured from the Department of Internal Medicine at The American University Hospital data on premature infant respiratory complexity.

Preterm or premature infants often suffer from respiratory complications.⁴ Due to their lack of development both physically and neurologically, these infants struggle with breathing on their own and maintaining a healthy proper gas exchange. The reflexive behavior that would allow them to breathe normally is not fully developed at

that stage when they would typically be dependent on the mother for oxygenation. Therefore, respiratory support becomes necessary. A non-invasive method for such support is Nasal Continuous Positive Airway Pressure, NCPAP. NCPAP relies on nasal tubing and creating positive pressure in the nostrils in order to compensate for the weak negative pressure surrounding a premature infant's lungs. It can be set to multiple levels of pressure depending on the severity of the case. The end goal is to regulate breathing and counteract episodes of apnea (interruption of breathing) and hypopnea (shallow breathing).

Despite having access to monitored breathing patterns of several patients, we had to be selective in choosing which to use for the anomaly experiment. Almost all but one of the patients exhibited patterns that were so irregular that attempting to use HTM to detect their anomalies would have put it at an unfair advantage. Figure 7.3 shows an example of that with the voltage output of the CPAP plotted. We therefore opt out of performing a multi-user experiment that would customarily be needed for validation and instead focus the case study on one patient whose data is highly regular except for instances that a human, expert or not, would be able to reliably discern and categorize like the one shown in figure 7.4 below. In what follow, we introduce a window-based anomaly detector, which was designed to compare with HTM and assess how it performs in relation to methods that incorporate domain knowledge. We then introduce the Numenta Anomaly Benchmark (NAB) repository that was used to carry out the experiments and finally present the results of HTM and a select number of online anomaly detectors.

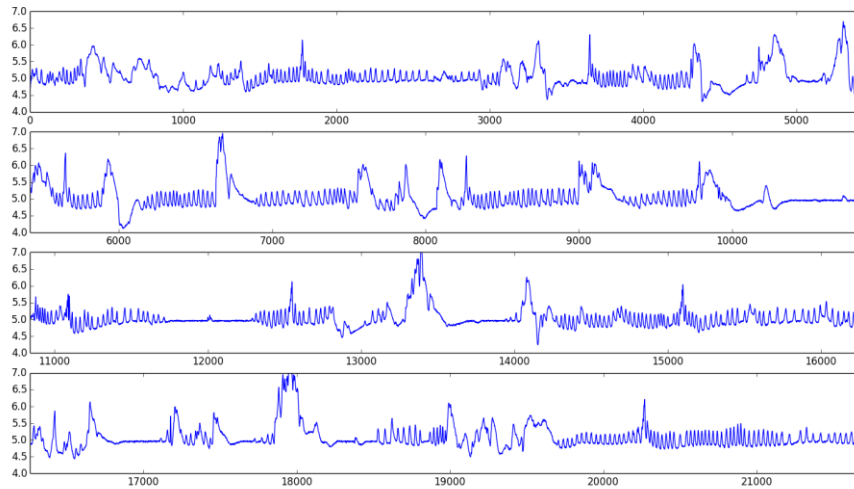


Figure 9.3: Breathing patterns recorded for patient 4 showing a high ratio of anomalous to regular patterns.

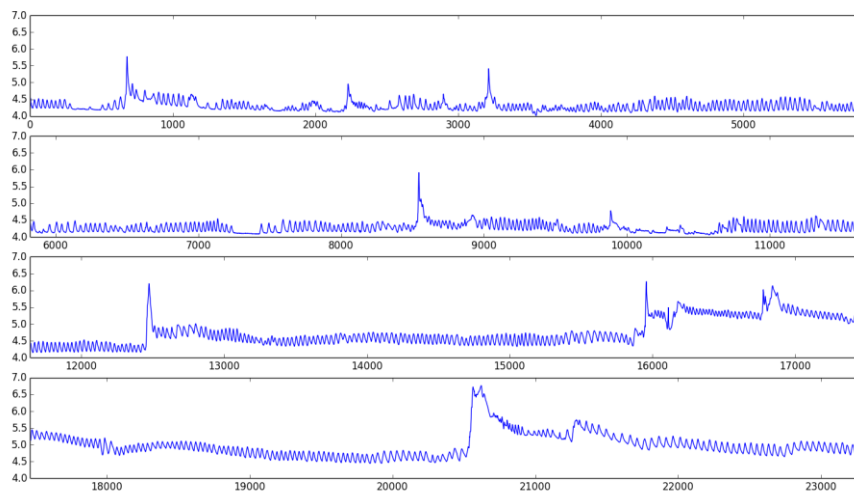


Figure 9.4: Breathing patterns recorded for patient 14 showing a proper ratio of anomalous to regular patterns.

1. Periodicity Detector

Anomaly detectors investigated in research or deployed in various industries are typically driven by domain knowledge from experts. HTM, as well as other online

anomaly detectors, attempt to provide one-size-fits-all models that are independent of such knowledge. While this makes for easier and more application agnostic deployment, the models stand to suffer due to working in a vacuum, with no assumptions about the data and its typical patterns. They instead resolve to extract—on the fly—information from data streams in order to form a model of what is normal and what is not. To provide the reader with insight into how these two approaches compare for a given application, we implemented our own heuristics-based domain-driven detector. This detector operates on a frame-by-frame basis where a frame of a predefined size is processed and labeled as normal or anomalous accordingly. While not online, this method of detection falls more in line with traditional anomaly detectors and will serve as a good competing technique to compare HTM with since it is not biased toward any specific type of anomaly. Detectors of the latter nature like apnea detectors are beyond the scope of this brief case study.

The custom detector presented here performs its processing with the assumption that regular and healthy breathing patterns are marked by periodic patterns. These patterns can change their baseline or offset but their repeatability is maintained throughout. Additionally, the frequency of the periodic patterns doesn't show drastic changes over time since no activity is being performed by the patient being monitored. With that knowledge, we designed the detector to act as a periodicity detector. Every frame of data is de-trended first to eliminate offset and lower frequency trends and is then correlated with itself to produce an autocorrelation waveform. Periodic signals produce autocorrelation waveforms that are similarly periodic. Therefore, we test for periodicity of the frame by checking for that of its autocorrelation waveform. To that end, we used a series of heuristics that involve peak detection and computing peak-to-

peak horizontal separations. If the separations are close in value, the detector interprets that as indication of periodicity within the frame and labels the frame as normal. This is done by calculating the mean and standard deviation of the separations and checking if the deviation is less than 15% of the mean separation. The value of 15% was empirically selected as it provided the best performance for a predefined frame size. A frame that fails this test is labeled as anomalous. Figure 7.5 shows a flowchart of the processing performed by the detector.

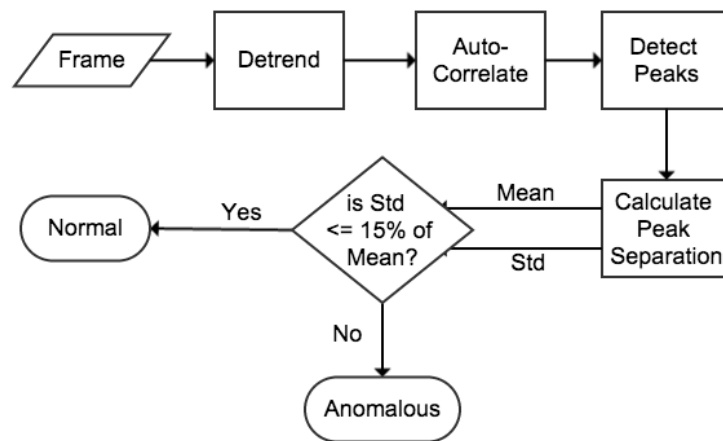


Figure 9.5: Flowchart of custom detector

2. The Numenta Anomaly Benchmark

a. What is NAB?

The Numenta Anomaly Benchmark is an open source repository created by Numenta, aimed at providing a framework to benchmark and compare online anomaly detectors. NAB includes in its current version a corpus consisting of synthetic and real world data including server metrics, machine sensor readings, and temperature readings.

A total of 32 time-series make up the corpus and are either labeled for ground truth anomalies by volunteers or accompanied by a list of known anomalies.²²

NAB includes three detectors: HTM-based detector, Etsy/Skyline detector, and a random detector. The Skyline detector is an open source, real time anomaly detection system, designed for use wherever there is a large quantity of high resolution time-series data which requires constant monitoring.²⁸ The random detector is provided as a trivial baseline for comparison. Its anomaly scores are chosen at random from a uniform distribution. The three inbuilt detectors as well as any custom detectors a NAB user can add are referred to as Detector Under Test (DUT).

DUT testing is performed in three stages: detector phase, threshold optimization phase, and scoring phase.

In the detector phase, every DUT processes the time-series data files in the corpus and produces raw anomaly scores for each data point read. The scores are stored in CSV format.

During the threshold optimization phase, the raw anomaly scores are discretized to values indicating the presence or absence of anomaly. This is done using an optimized threshold. NAB includes a simple hill-climbing routine for finding the optimal threshold given the scoring rules. The threshold used is the one that produces the best score.

Finally, in the scoring phase, the DUT detections are scored with respect to the ground truth timestamps producing a final score for each data file in the corpus. A sigmoid function is used as a scoring function and is weighted according to the application profile chosen. The application profiles enable us to vary the relative cost of the scoring metrics. We elaborate on the scoring routine in the following section.

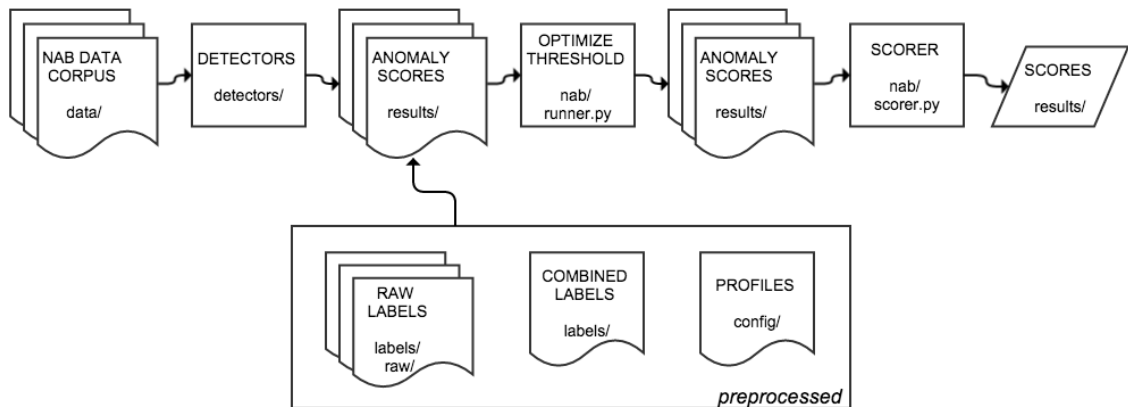


Figure 9.6: Flowchart of NAB's workflow

b. NAB Scoring

For evaluation of DUTs, NAB implements a custom scoring methodology, which qualifies the degree to which the DUT meets the standards of an ideal detector. The ideal detector, as defined by Numenta, is one which:

- Detects all anomalies present in the streaming data
- Detects anomalies as soon as possible, ideally before the anomaly becomes visible to a human
- Triggers no false alarms
- Works with real world data

With raw anomaly scores of the DUT thresholded, NAB proceeds to compare the predicted anomalies with the ground truth anomalies. More specifically, this comparison is performed on anomaly windows centered on the ground truth anomalies. The rationale behind the use of windows is two-folds: 1- anomalous data often occurs over time, rather than a single point, and thus defining the anomaly windows improves

the NAB scoring efficacy, and 2- anomaly windows allow the DUT to not be penalized if its detections are slightly before or after the ground truth.

The scoring mechanics are centered on the scaled sigmoid $S(x)$ in (7.1), with x being the relative position within the anomaly window. Figure 7.7 shows the scoring function as well as the boundaries of the anomaly window where it is applied. The scoring function was designed in such a way as to reward detections that occur at or slightly ahead of the ground truth anomaly the highest, while scoring later detections gradually less as they lag behind the ground truth.

$$S(x) = \frac{2}{1+e^{-5x}} - 1 \quad (7.1)$$

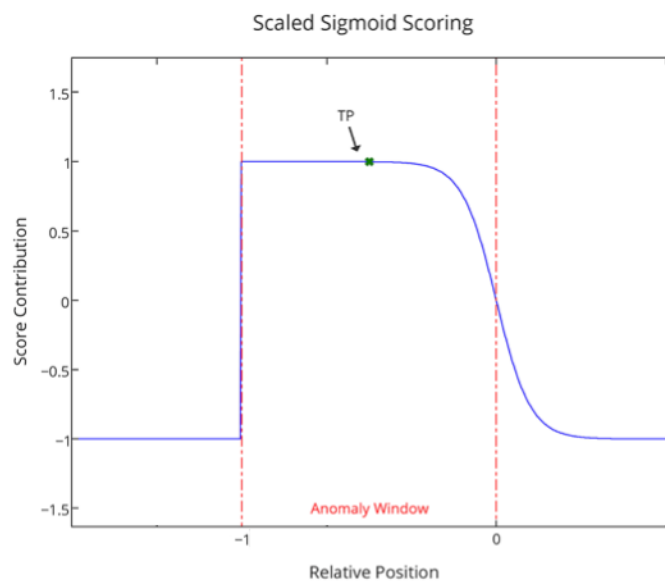


Figure 9.7: Scaled sigmoid scoring function

For anomalies correctly predicted, only the first data point labeled positive within the anomaly window is used to calculate the change in score. Further data points within the anomaly window labeled positive are ignored and do not contribute to the

score. The score is additionally weighted by a parameter that controls the weight associated with true positives (TP).

$$Score = Score + S(x) * TP_{weight} \quad (7.2)$$

A data point labelled as positive outside of the ground truth anomaly window gets a negative valued scoring function and decrements the overall score according to (7.3). Again, the score is weighted by a parameter associated in this scenario with the weight of a false positive (FP). It is worth mentioning that the NAB documentation doesn't detail any special handling of other FPs within the neighbourhood of the first like it does for TPs. This suggests that NAB rewards a single TP per anomaly but penalizes the DUT for all the FP around it. This is evidence that the scoring routine favours DUTs that produce very few anomaly detections, a feature that is problematic if the repository is to include DUTs that produce consistent anomalies for the length of an anomalous pattern.

$$Score = Score + S(x) * FP_{weight} \quad (7.3)$$

Finally, if no anomaly is detected within a ground truth anomaly window, the overall score is reduced with $S(x)$ being assigned a value of -1. A weighting parameter is used here as well, which represents the weight associated with false negative (FN).

$$Score = Score - FN_{weight} \quad (7.4)$$

There is no minimum NAB score. The maximum on the other hand is the number of ground truth anomalies multiplied by the TP weight. Scoring weight as seen in (7.2), (7.3), and (7.4) are configured as part of different application profiles, which either reward low FN, reward low FP, or weight all detections uniformly.

3. Results

The first step to running a set of detectors through NAB is to provide ground truth anomalies for the time-series being processed. We perform manual anomaly selection on the CPAP voltage output of patient 14 using three anomaly types: apneas/hypopneas, sighs, and general irregularities. This produces three separate ground truth files for NAB to process. Figure 7.8 shows the selection of apneas. For each apnea, we select the starting point where breathing flatlines. The size of the resulting anomaly windows is dictated by a heuristic that NAB enforces and is equal to 10% the length of the time series divided by the number of anomalies. Due to the small number of apneas in this file, the result is a window that is 388 samples wide which extends further into the regular patterns than we would like.

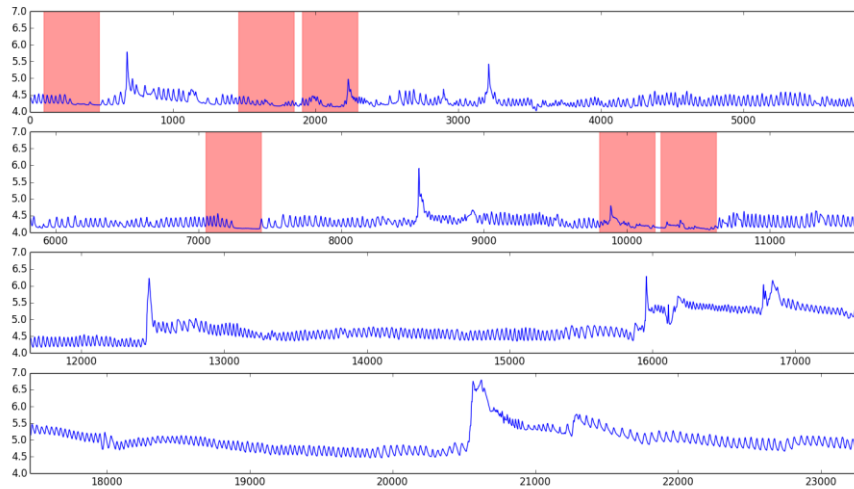


Figure 9.8: Selection of ground truth apnea anomalies

Figure 7.9 reveals the manual selection of the ground truth anomalies for sighs. These anomalies were chosen by selecting the peak of each sigh. The resulting window size here is 258 and does a decent job of containing both the peak and slopes of the sigh patterns.

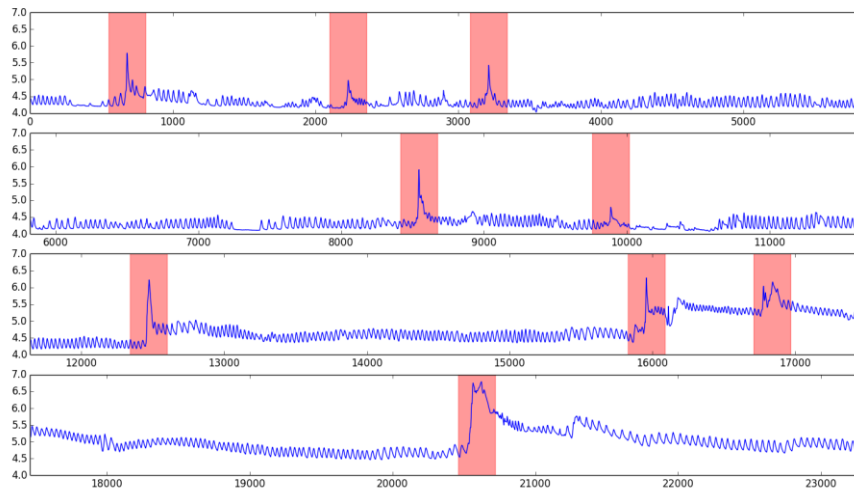


Figure 9.9: Selection of ground truth sigh anomalies

Finally, we manually select all irregular patterns. These include apneas, hypopneas, sighs, and any sort of miscellaneous irregularity in the data. Since none of the detectors involved can be biased toward one type of anomaly, this is the most important ground truth file to consider. Unfortunately, due to the large number of anomalies, window size is limited to 88 samples. This results in anomalous patterns between neighbouring anomalies to fall outside the anomaly windows. This can be seen in the second subplot of figure 7.10.

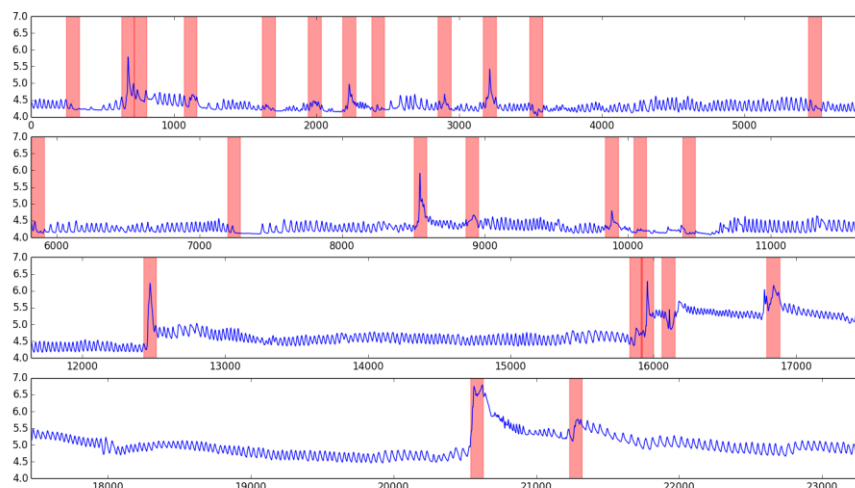


Figure 9.10: Selection of ground truth general anomalies

Having established the ground truth anomalies, we finally run the data through the detectors. Figure 7.11 shows the result of feeding the patient's breathing patterns into HTM. The model parameters included in the NAB repository were used in order to assess how HTM performs with no assumptions on the nature of the data other than min and max allowable values. This approach is applied to all detectors involved. HTM seems to align well with sighs but it misses the last one despite being really pronounced

and preceded by a long lasting normal pattern. Additionally, HTM does not appear to respond well to apneas and throws a number of false positives.

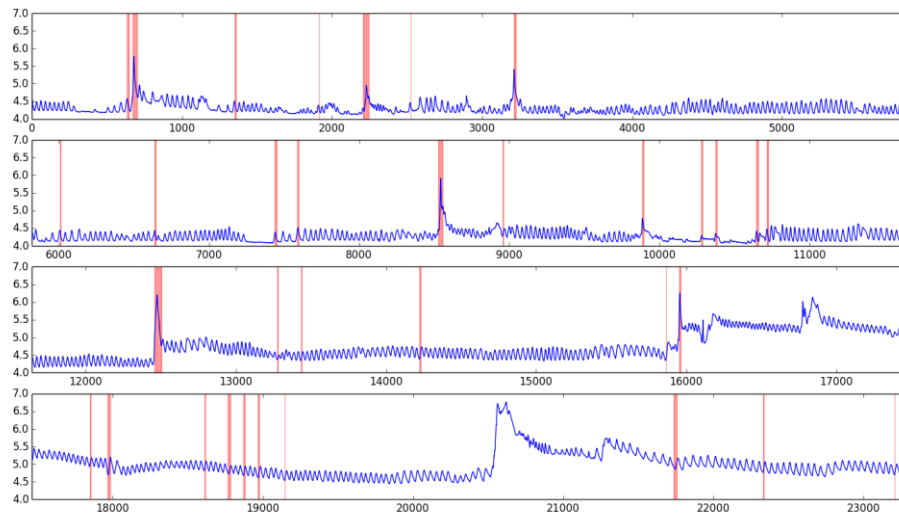


Figure 9.11: Anomaly detection using the HTM-based detector

With a frame size of 150 samples, the result of running our custom detector on patient 14's data can be seen in figure 7.12. While some false positives can be seen, the detector does an admirable job of picking up on every major anomaly. All frames containing apneas, hypopneas, sighs (peaks), or irregular breathing patterns are labeled as anomalous.

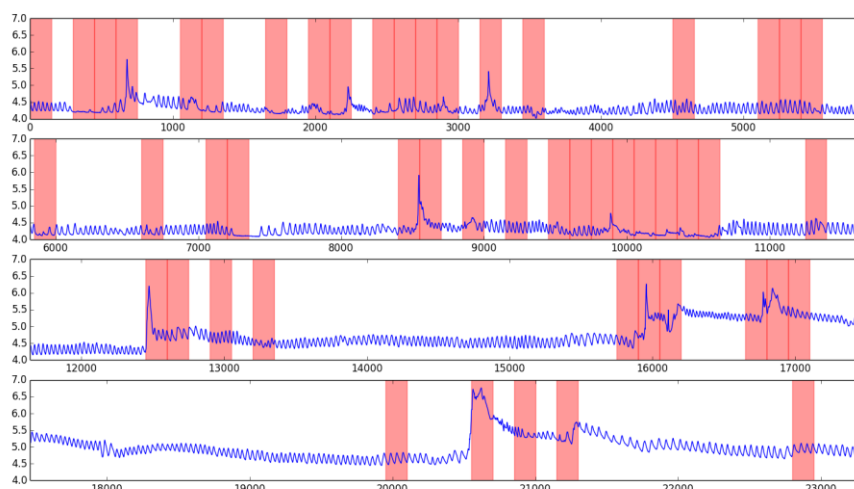


Figure 9.12: Anomaly detection using the custom periodicity detector

NAB scoring is the last step. The data files were run through NAB to produce scores for each of the anomaly files. The scoring was performed under three different application profiles: 1- Reward Low FN with weights $TP=1$, $FP=0.5$, and $FN=2$ which rewards detectors that don't miss any true anomalies; 2- Reward Low FP with weights $TP=1$, $FP=2$, and $FN=0.5$ which rewards detectors with few false alarms; and 3- Standard with weights $TP=1$, $FP=1$, and $FN=1$. Table 7.4 shows the raw scores as well as the normalized totals across all datasets. The best raw score for every dataset is highlighted in yellow while the best normalized total score is in dark grey. The custom detector proposed performs the best in 8 out of the 9 scored tests with the Numenta/HTM detector redeeming itself with a single win against it while outperforming the other detectors. This is isn't surprising given the fact that the custom detector, though simple in design, incorporates domain knowledge and benefits greatly from it especially when it comes to detecting general non-specific anomalies. Still, we suspect that our detector, despite capturing every apnea, suffered for yielding anomalies

that were more delayed and therefore scored less via the scaled sigmoid scoring function. To account for the penalization of having delayed detections, we replace the scaled sigmoid scoring function by a square pattern having a value of 1 within the anomaly window and -1 outside its bounds. Table 7.5 shows that normalized total scores naturally improved but the overall result is the same.

Table 9.4: Raw scores and normalized total scores after NAB testing using a scaled sigmoid scoring function

Application Profile	Random			Skyline			Numenta			Custom		
	Low FN	Low FP	Standard	Low FN	Low FP	Standard	Low FN	Low FP	Standard	Low FN	Low FP	Standard
Sighs	-13.6	-7.2	-6.6	-6.4	-9.0	-9.0	-7.1	-5.9	-3.2	-5.1	-5.1	-2.1
Apneas	-12.7	-7.5	-6.7	-14.3	-6.0	-6.0	-6.7	-2.4	-4.2	-6.3	-6.5	-3.3
Irregular	-50.0	-25.7	-25.0	-43.7	-26.0	-26.0	-41.2	-22.9	-20.0	-35.4	-19.4	-16.4
Normalized Total	4.4	0.6	2.9	13.0	0.0	0.0	21.3	11.4	15.9	27.8	11.5	22.42

Table 9.5: Normalized total scores after NAB testing using a square scoring function

Application Profile	Random			Skyline			Numenta			Custom		
	Low FN	Low FP	Standard	Low FN	Low FP	Standard	Low FN	Low FP	Standard	Low FN	Low FP	Standard
Normalized Total	5.6	2.4	24.8	14.5	0.0	0.0	22.3	11.8	16.7	30.4	14.6	26.2

Finally, we present some additional testing stats for analysis for the HTM-based detector and the custom detector after a second run of the NAB testing in table 7.6. We make the following observations:

- The HTM-based detector doesn't yield the same result for every run. As emphasized in earlier chapters, this is due largely to the random initialization of the HTM model. The result is non-repeatable performance. Here, HTM performs better than the earlier run as seen in the scores.

- The custom detector has bad precision. This is mostly linked to its rigidity in detecting periodic patterns. As evidenced in figure 7.12, the detector is a lot less forgiving towards slight irregularities. This results in a high number of FP, which in turn yields a low precision. The HTM-based detector is aided by having a significantly lower number of detections.
- Recall numbers are extremely low due to the FN counts. As alluded to earlier, NAB's scoring routine rewards the first true positive but penalizes all FNs. This is a consequence of using anomaly windows that force all data samples within to have a ground truth value of 1. When compared against sparse anomalies like the ones produced by the detectors investigated, this naturally and problematically results in artificially high FNs, which reduce the recall rate.
- The scoring routine used in NAB has little correlation with standard stats commonly used to evaluate learning algorithms like precision, recall, and F1 score. This is evident in the results shown where these metrics are rendered meaningless and offer little to no insight into how the DUTs are performing.

Table 9.6: Results of a second run of NAB testing on the HTM-based detector and our custom detector for the standard profile

		TP	TN	FP	FN	Score	Precision	Recall	F1
Custom	Sighs	8	18200	29	1552	-2.1	21.6%	0.51%	0.99
	Apneas	6	18588	31	1164	-3.3	16.2%	0.51%	0.98
	Irregular	7	18328	30	1424	-16.4	18.9%	0.48%	0.93
Numenta	Sighs	10	18225	4	1550	0.07	71.4%	0.64%	1.26
	Apneas	0	18605	14	1170	-7.5	0.0%	0.0%	0.0
	Irregular	7	18351	7	1424	-17.4	50%	0.48%	0.95

CHAPTER VIII

CONCLUSION

HTM is a novel, cortically inspired learning algorithm. It stems from a long line of connectionist learning models but sets itself apart by its focus on biological plausibility and its incorporation of time as a central cog in its workings. Having had very little time to mature, HTM hasn't had the opportunity to position itself as a viable learning algorithm in machine learning. Subsequently, the work presented in this thesis did not attempt to build on it but rather evaluate its foundations. This was done by targeting all modules of the HTM stack and running them through a set of experiments that showcased their functionality as well as positioned them in relation to competing techniques.

We explored HTM's clustering potential by leverage the quantization capability of both its encoders and spatial pooler with both synthetic and real data. A number of performance metrics were used to compare HTM against common clustering methods, which revealed that HTM could be competitive in select datasets but generally demonstrated inconsistency in results. Additionally, overlap in SDRs at the spatial pooler level was shown to be a non-robust metric for similarity, which further diminished HTM's merit as a standalone clustering algorithm.

HTM's performance as a sequence learner was investigated as well using small examples on different types of sequences. Sequence learning tests showed HTM's capacity for fast and precise learning for discrete sequences. Investigations of the number of cells and columns used and their effect of sequence learning performance revealed some concerns again with HTM's consistency. More importantly, the manner

by which lateral connectivity is established brought on doubts about how contexts are learned and pooled together. Nevertheless, overall performance on sequence learning within the bounds of the experiments presented was admirable.

It was in the case studies where HTM's standing suffered most. Two case studies were conducted with real work data. The first ventured to evaluate HTM in classification, a task that is not inherent to HTM. Using a dataset acquired from a gesture recognition and tracing framework created by the author, HTM and a set of classifiers were evaluated in multiple learning scenarios. HTM's performance in all the experiments lagged significantly behind the other classifiers which managed 90%+ accuracy rate while HTM peaked at 93.75% for a single user but coasted in the 70%-80% range for the rest. Additionally, two problems were identified with the use HTM. The first was the difficulty in model selection. With at least ten parameters directly affecting HTM's performance, optimal parameter selection was not practical. Numenta offers a PSO-based swarming tool but it only optimizes for a few parameters and is more tailored for dense data stream prediction. The result was the unavoidable reliance on manual searches and the inevitable suboptimal performance observed. A second concern was the scalability of HTM. Runtimes, while not presented here, were significantly higher than those of the competing algorithms. Models had to be configured in single region fashion and attempts at hierarchical structures were very prohibitive in testing. We refer the interested reader to a thesis titled "Hierarchical Temporal Memory Cortical Learning Algorithm for Pattern Recognition on Multi-Core Architectures"²⁴ which presents a study of HTM's scalability and the benefit of a multi-core implementation.

The second case study was more in line with HTM's strong suit, anomaly detection. The Numenta Anomaly Benchmarking tool was used to test multiple detectors on breathing patterns from an infant respiratory complexity clinical study. Only one patient whose data revealed abundant normal patterns with distinct anomalies was chosen for testing. In addition to HTM and the set of detectors provided by NAB, we proposed and designed a simple heuristically driven batch detector for comparison. After NAB testing, HTM and the custom detector were shown to perform best with the latter coming in first place. Analysis of the scoring scheme carried on by NAB reveals a few concerns that we attribute to the beta status of the tool.

Although HTM shows hints of promise as a learning algorithm, it is still held back by its immaturity. While it continues to improve and find its footing with the backing of the team at Numenta, future work should focus on providing a proof of convergence as a first step. We deem that essential for HTM to gain any traction with the machine learning research community. Additionally, the framework needs to be reworked to include more of the operational principles of the brain. This is already in motion with Numenta's work on sensory-motor integration but many aspects including feedback, input reconstruction, and true temporal pooling are necessary to make HTM a solid contender.

REFERENCES

- 1 Bruce Bobier, 'Hand-Written Digit Recognition Using Hierarchical Temporal Memory', *University of Guelph* (2007).
- 2 Varun Chandola, Arindam Banerjee, and Vipin Kumar, 'Anomaly Detection: A Survey', *ACM Computing Surveys (CSUR)*, 41 (2009), 15.
- 3 Chronopause, <<http://chronopause.com/chronopause.com/index.php/2012/02/21/the-effects-of-cryopreservation-on-the-cat-part-3/index.html>>.
- 4 Milo Engoren, Sherry E Courtney, and Robert H Habib, 'Effect of Weight and Age on Respiratory Complexity in Premature Neonates', *Journal of Applied Physiology*, 106 (2009), 766-73.
- 5 Patrick Gabriëlsson, R Konig, and Ulf Johansson, 'Hierarchical Temporal Memory-Based Algorithmic Trading of Financial Markets', in *Computational Intelligence for Financial Engineering & Economics (CIFER), 2012 IEEE Conference on* (IEEE, 2012), pp. 1-8.
- 6 Dileep George, and Jeff Hawkins, 'A Hierarchical Bayesian Model of Invariant Pattern Recognition in the Visual Cortex', in *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on* (IEEE, 2005), pp. 1812-17.
- 7 Stephen Grossberg, 'Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks', in *Studies of Mind and Brain* (Springer, 1982), pp. 332-78.
- 8 Stephen Grossberg, 'How Does a Brain Build a Cognitive Code?', in *Studies of Mind and Brain* (Springer, 1982), pp. 1-52.
- 9 Nadine Hajj, and Mariette Awad, 'Weighted Entropy Cortical Algorithms for Isolated Arabic Speech Recognition', in *Neural Networks (IJCNN), The 2013 International Joint Conference on* (IEEE, 2013), pp. 1-7.
- 10 Atif Hashmi, and Mikko H Lipasti, 'Discovering Cortical Algorithms', in *IJCCI (ICFC-ICNC)* (2010), pp. 196-204.
- 11 Jeff Hawkins, S Ahmad, and D Dubinsky, 'Hierarchical Temporal Memory Including Htm Cortical Learning Algorithms', *Technical report, Numenta, Inc, Palto Alto* http://www.numenta.com/htmoverview/education/HTM_CorticalLearningAlgorithms.pdf (2010).
- 12 Jeff Hawkins, and Sandra Blakeslee, *On Intelligence* (Macmillan, 2007).
- 13 Donald Olding Hebb, *The Organization of Behavior: A Neuropsychological Approach* (John Wiley & Sons, 1949).
- 14 Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh, 'A Fast Learning Algorithm for Deep Belief Nets', *Neural computation*, 18 (2006), 1527-54.
- 15 John J Hopfield, 'Neural Networks and Physical Systems with Emergent Collective Computational Abilities', *Proceedings of the national academy of sciences*, 79 (1982), 2554-58.
- 16 Teuvo Kohonen, 'Self-Organized Formation of Topologically Correct Feature Maps', *Biological cybernetics*, 43 (1982), 59-69.
- 17 Ioannis Kostavelis, Lazaros Nalpantidis, and Antonios Gasteratos, 'Object Recognition Using Saliency Maps and Htm Learning', in *Imaging Systems and*

- Techniques (IST)*, 2012 IEEE International Conference on (IEEE, 2012), pp. 528-32.
- 18 Robert Linggard, DJ Myers, and C Nightingale, *Neural Networks for Vision, Speech, and Natural Language* (Chapman & Hall, Ltd., 1992).
 - 19 Warren S McCulloch, and Walter Pitts, 'A Logical Calculus of the Ideas Immanent in Nervous Activity', *The bulletin of mathematical biophysics*, 5 (1943), 115-33.
 - 20 Marvin Minsky, and Papert Seymour, 'Perceptrons', (1969).
 - 21 Vernon Mountcastle, 'An Organizing Principle for Cerebral Function: The Unit Model and the Distributed System', (1978).
 - 22 NAB, <<https://github.com/numenta/NAB>>.
 - 23 Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg, 'Locality Sensitive Hashing: A Comparison of Hash Function Types and Querying Mechanisms', *Pattern Recognition Letters*, 31 (2010), 1348-58.
 - 24 Ryan William Price, 'Hierarchical Temporal Memory Cortical Learning Algorithm for Pattern Recognition on Multi-Core Architectures', (2011).
 - 25 Kandel Eric R, Schwartz James H, and Jessell Thomas M, 'Principles of Neural Science', (McGraw Hill, 2000).
 - 26 Frank Rosenblatt, 'The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain', *Psychological review*, 65 (1958), 386.
 - 27 S Sinkevicius, R Simutis, and V Raudonis, 'Monitoring of Humans Traffic Using Hierarchical Temporal Memory Algorithms', *Elektronika ir Elektrotechnika*, 115 (2011), 91-96.
 - 28 Skyline, <<https://github.com/etsy/skyline>>.
 - 29 Paul Smolensky, 'Connectionist Ai, Symbolic Ai, and the Brain', *Artificial Intelligence Review*, 1 (1987), 95-109.
 - 30 Ron Sun, and C Lee Giles, 'Sequence Learning: From Recognition and Prediction to Sequential Decision Making', *IEEE Intelligent Systems*, 16 (2001), 67-70.
 - 31 Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang, 'Phoneme Recognition Using Time-Delay Neural Networks', *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 37 (1989), 328-39.
 - 32 Bernard Widrow, *Adaptive "Adaline" Neuron Using Chemical "Memistors."* (1960).
 - 33 Matthew D Zeiler, and Rob Fergus, 'Visualizing and Understanding Convolutional Networks', in *Computer Vision—Eccv 2014* (Springer, 2014), pp. 818-33.
 - 34 Xia Zhituo, Ruan Hao, and Wang Hao, 'A Content-Based Image Retrieval System Using Multiple Hierarchical Temporal Memory Classifiers', in *Computational Intelligence and Design (ISCID), 2012 Fifth International Symposium on* (IEEE, 2012), pp. 438-41.
 - 35 Wen Zhuo, Zhiguo Cao, Yueming Qin, Zhenghong Yu, and Yang Xiao, 'Image Classification Using Htm Cortical Learning Algorithms', in *Pattern Recognition (ICPR), 2012 21st International Conference on* (IEEE, 2012), pp. 2452-55.