Using Hierarchical Temporal Memory for Detecting Anomalous Network Activity

THESIS

Gerod M. Bonhoff, 1st Lt, USAF

AFIT/GCS/ENG/08-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCS/ENG/08-04

Using Hierarchical Temporal Memory for Detecting Anomalous Network Activity

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Degree of Master of Science (Computer Science)

Gerod M. Bonhoff, B.S.C.S.

1st Lt, USAF

March 2008

AFIT/GCS/ENG/08-04

Using Hierarchical Temporal Memory for Detecting Anomalous Network Activity

Gerod M. Bonhoff, B.S.C.S.
1st Lt, USAF

Approved:

/signed/	27 Mar 2008
Lt Col (Ret) Robert F. Mills, PhD (Chairman)	date
/signed/	27 Mar 2008
Maj (Ret) Richard A. Raines, PhD (Member)	date
/signed/	27 Mar 2008
Dr. Gilbert L. Peterson (Member)	date

Abstract

This thesis explores the nature of cyberspace and forms an argument for it as an intangible world. This research is motivated by the notion of creating intelligently autonomous cybercraft to reside in that environment and maintain domain superiority. Specifically, this paper offers 7 challenges associated with development of intelligent, autonomous cybercraft.

The primary focus is an analysis of the claims of a machine learning language called Hierarchical Temporal Memory (HTM). In particular, HTM theory claims to facilitate intelligence in machines via accurate predictions. It further claims to be able to make accurate predictions of unusual worlds, like cyberspace. The research thrust of this thesis is then two fold. The primary objective is to provide supporting evidence for the conjecture that HTM implementations facilitate accurate predictions of unusual worlds. The second objective is to then lend evidence that prediction is a good indication of intelligence.

A commercial implementation of HTM theory is tested as an anomaly detection system and its ability to characterize network traffic (a major component of cyberspace) as benign or malicious is evaluated. Through the course of testing the poor performance of this implementation is revealed and an independent algorithm is developed from a variant understanding of HTM theory. This alternate algorithm is independent of the realm of cyberspace and developed solely (but also in a contrived abstract world) to lend credibility to concept of using prediction as a method of testing intelligence.

Acknowledgements

My AFIT experience has provided me with some of the most interesting times I've had in the Air Force. I own every thrilling "ah-ha!" moment to AFIT's dedicated faculty and staff.

Of course, no thesis acknowledgment is complete without paying due respect to the thesis adviser. In my case, however, no simple nod will due for the support and encouragement provided by Dr. Robert Mills. Under the guise a laissez-faire research policy I was invisibly guided along an academic path more interesting than any in my past. Although much of what I learned won't make it into this thesis, your encouragement of outside the box thinking was refreshing. I appreciate you considering my novel ideas with sincerity and for extending that helping hand even when I didn't think I needed it. Indeed, at the very end you were there for me when I needed you the most. I want to thank you for being on the side of exploration and discovery.

Last but not least, I need to thank my support system both in and out of the classroom. To the *Old Man*, the *Sabres' Fan*, my *ASBC Pal* and *Racing Gal*, thank you for being both true friends and sounding boards. To the *Bagel Lady*, thank you for being a friendly face I could always trust for concern and support. To my parents, who have supported me through each trial and tribulation of my military career, thank you for ensuring each is a triumph. Most importantly I want to thank my wife. For over 18 months you've unselfishly made long drives, late night flights, daily phone calls, and countless care packages. Thank you for ensuring constant company along the way and for providing a warm home at the bottom of the hill.

Gerod M. Bonhoff

Table of Contents

List of Figures

List of Abbreviations

Using Hierarchical Temporal Memory for Detecting Anomalous Network Activity

I. Introduction

Questions defining the nature of cyberspace have become potent in recent months as the United States government declares the environment of cyberspace as vital to the national interests as those of land, sea, air, and space. The US Air Force (USAF) has taken the first steps toward supporting those interests by creating the nation's first Cyberspace Command which will have equal footing with their current charges of air and space. Cyberspace doctrine, attempting to relate the similarities of military strategies and protocols from historical fronts of war, is under development.

A conceptual cyber-entity known as a "Cybercraft" has been proposed by researchers. The idea is to create a cyber-platform to ease the leap from cyberspace strategy to tactical, defensive information operations in the cyberspace domain. The concept is currently being pushed from the theoretical to the engineer's drawing board by USAF leadership.

To accomplish operations in cyberspace, cybercraft must be given the ability and knowledge to perform complex behaviors resulting in mission success. Indeed, one of the core components a cybercraft design will need to incorporate is "autonomy" [32]. Moreover, the cybercraft concept [28] implicitly requires a cybercraft employ "intelligent autonomy" [30]. Although a simple cockroach can be said to be autonomous, the challenge to cybercraft will be developing intelligent autonomy which is an attribute currently possessed by the only known intelligent entities: humans.

Developing an intelligently autonomous cybercraft is a tricky endeavor, however, as humans have difficulty comprehending just exactly what the cyberspace environment *is* [26]. Teaching any intelligent entity (artificial or otherwise) is difficult if the teacher dosn't know what it is that must be taught. Because humans have evolved in

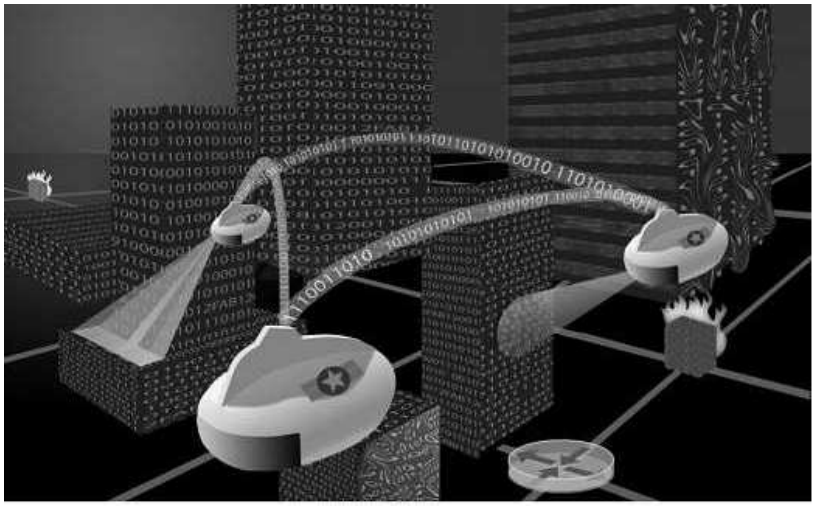

Figure 1.1: Artistic Rendition of a CyberCraft

the physical world, they have no internal model of other, more unusual worlds [11], like cyberspace. Examples of such intangible worlds include "weather worlds" (with their complex, and as of yet, not fully understood, "butterfly-effect" weather patterns) and the two-dimensional "interstate world" a smart-car might traverse [11].

So, just as human beings operate in the physical realms of Earth, cybercraft must operate intelligently and autonomously in cyberspace — a seemingly intangible world. Endowing such abilities to machines, like cybercraft, is the primary goal within the field of AI. Although the list of the AI community's accomplishments is long and distinguished, there does not currently exist any known method of creating intelligence comparable to that of a human. Certainly, training a machine to be intelligent in a world humans have difficulty comprehending presents a more in-depth problem.

The founders of the cybercraft initiative call for reasoning abilities that seem to go beyond the algorithms currently entrusted to play chess or even fly aircraft. Such sapient abilities are required to recognize new problems and solve them for the success of a higher mission. Yet, in a mere coincidence, at about the same time the cybercraft was proposed, a machine learning language was offered that claims to hold the key for solving such problems [11]. Like most AI concepts, Hierarchical Temporal Memory (HTM) is based on a conceptual theory of how the human brain might work [11]. The

primary difference with HTM theory and other proposals is that HTM claims to be able to perform a measure of unsupervised learning to make predictions in unusual worlds (i.e. cyberspace) where humans do not exist.

Motivated by intelligent autonomy requirements of cybercraft, this research investigates the claims of the machine learning method of HTM and the cortical theory on which it is founded. The thesis begins with investigations into what it means to *be intelligent* and the underpinning connections to HTM theory. This paper concludes with experiments implementing an HTM network algorithm as an anomaly detection system. The ability of an HTM network to understand the difference between nominal computer network traffic and malicious computer network activity would support HTM theory's claims and potentially lead to it's adaptation for various cyberspace activates, including those of cybercraft.

1.1 Research Focus

This thesis researches many ideas, concepts, and theories with regard to cyberspace and intelligence. The primary focus is an analysis of the claims of HTM theory, its software implementations, and its viability to evolve the concept of AI into cyber-worlds.

This thesis studies the cortical theory which claims intelligence is built on the basic ability to predict outcomes. This research will focus on developing support for HTM theory, specifically in its ability to provide accurate predictions in unusual worlds. In the course of this exploration a commercial software implementation of HTM theory is presented and tested for viability in a cyberspace environment. Additionally, a algorithm is built from scratch to study the introduction of feedback, a key concept left out of the commercial implementation, within an HTM network. Both experiments are designed to employ a measure of unsupervised learning within the confines of an unusual world.

A final analysis of HTM theory, its implementations, and viability for cyberspace applications (such as anomaly detection) is offered.

1.2 Research Impact

The impact of this research is two fold. First, this research intends to investigate the claims that prediction can be used as a primary measure of intelligence. Secondly, evidence (both informational and experimental) is gathered to support the applicability of HTM theory as a feasible option for generating accurate predictions in unusual worlds.

1.3 Thesis Organization

The thesis is organized into five main sections. The next section reviews pertinent information and previous research applicable to the thesis. This background information includes a look at cyberspace, cyber-operations, the need for cybercraft, the relation of intelligence to the program, theories of intelligence, and AI approaches. Of primary focus is an in-depth look at HTM theory.

The section that follows reviews the problem domain [21] and the HTM applicable solution. It maps HTM theory to algorithm and the implementation. The HTM application is introduced and discussed. Here, the anomaly detection experiment is proposed to test the HTM implementation along with a framework for evaluation.

The next section is an detailed look at the proposed experiment. Specifications on experiment design, HTM incorporation, and procedures are provided. Additionally, goals and expectations are discussed.

The subsequent section reviews the results of the experiment. Data is presented and discussed. Modifications and refinements to experiment along with explanations for their use are presented. Finally, analysis of all results is performed.

Before conclusions are drawn, a section is provided where an independent implementation of HTM Theory (designed from the author's understanding of key HTM concepts) is demonstrated. Here, results are analyzed independent of the anomaly detection experiment. The goal of this section is to provide a mere glimpse into aspects

of HTM Theory that are not currently implemented in the commercial system and thus not covered by the primary experiment.

In the last section, conclusions are drawn from the experimental results with respect to background research and initial expectations. Interpretations of the findings are provided along with any perceived significance. In conclusion, recommendations for further research and additional experiments are proposed.

II. Literature Review

Before this thesis can move towards the realization of its title, a firm foundation must be established. As such, this section has two major goals. The first goal is to solidify a general understanding of ideas critical to the theories presented later in this paper. To this end, background information on understood concepts, such as "Information Warfare" and "Artificial Intelligence," will be summarized. Additionally, abstract terms such as "cyberspace," "cybercraft," and "intelligence" will be clarified. The second goal is to provide a comprehensive review of the theory of Hierarchical Temporal Memory.

2.1 Information Warfare

In 1996 the Institute for the Advanced Study of Information Warfare defined Information Warfare (IW) as "the offensive and defensive use of information and information systems to exploit, corrupt, or destroy an adversary's information and information systems, while protecting one's own. Such actions are designed to achieve advantages over military or business adversaries" [26]. Information is a resource, a *commodity*, like rice or oil. Although offensive and defensive operations to exploit those resources may more accurately qualify as non-combative [26], conducting IW (or Information Operations (IO) as it is referred to in Joint Publication 3-13 [26]) can have just as severe an impact on resources, informational or physical.

IO is currently defined as the "integrated employment of the core capabilities of electronic warfare, computer network operations, psychological operations, military deception, and operations security, in concert with specified supporting and related capabilities, to influence, disrupt, corrupt or usurp adversarial human and automated decision making while protecting our own" [26]. The last part of this definition is important because it begins to touch upon how IW/IO doctrine is to be created for cyberspace. As with other military operations, the heart of activities revolves around a notion, proposed by military strategist John Boyd, known as the OODA Loop. The

concept refers to the cyclic flow of command and control functions through "Observe, Orient, Decide, Act" phases.

2.1.1 IW/IO OODA Loops. Problems arise when the various IW/IO concepts and disciplines (Information Dominance, Net-Centric Warfare, Defensive Information Operations, Information Security, Information Assurance, Information Survivability, etc.) are attempted within an OODA Loop framework for cyberspace. Each phase is hindered by unique difficulties associated with various aspects of IW/IO. The OODA Loop concept suffers due to the unique environmental factors of cyberspace. Causes and problems associated with each phase are proposed and illustrated below:

2.1.1.1 Observe. Observation is "the collection of data by means of the senses " [17]. Observing the environments of the real-world (along with the associated operations) comes easily to humans. However, performing such surveillance or reconnaissance missions in cyberspace is more difficult. Two generic reasons are proposed:

- First, observation of certain insider threat (IT) activity is difficult on a network. This is because the covert, pre-authorized, and traitorous nature of such operations are difficult for logic based security systems to comprehend [2]. While a human might find distinguishing between the treacherous activity of an insider and normal actions of trusted users less difficult, such abilities are not currently possessed by computers.

  Consider a social engineering IT attack: If a human were fully aware of (and strictly compliant to) operational security regulations they could easily listen-in on and foil an insider's attempt to gain protected information. However, the only system capable of the functional requirements of such flawless, unemotional monitoring is a computer system, not a human. Unfortunately, computers currently can not understand such a conversation. In short, some observations can

be conducted better with human sensing capabilities but would be improved by computer abilities.

- The second reason is the inverse of the first. Observation of network activity is something very difficult for a human to understand, not for a lack of comprehension, but for the lack of an appropriate set of sensors appropriate to the network environment. Eyes and ears are probably not the ideal sensors for presenting the world of cyberspace to humans. Computers, on the other hand, can be given sensors unlike that of any human. Such sensors would enable machines the unique capability to observe the network environment better than humans. Unfortunately, computers do not have the intelligence to efficiently utilize those observations.

  Consider detecting suspicious activity: A human would see someone riffling though a file cabinet at 0230 in the morning suspicious. Similarly, if they saw someone looking at secret network files they would also take note. The problem is that detecting someone sifting though databanks is not as simple as seeing a light on in the office during the wrong time of day. Yet, a computer could sense network activity with ease. However, determining the malicious nature of the network activity is an ability currently beyond that of machines. In short, some observations can be conducted better with computer sensing capabilities but would be improved by human abilities of perception or understanding.

2.1.1.2 Orient. Human orientation of any observations in cyberspace also has two areas of difficulty. Because orientation means "analysis and synthesis of data to form one's current mental perspective," [17] two areas of concern immediately become apparent given the issues with observing cyberspace:

- First, as mentioned above, humans often have need for computers to analyze data due to limited human capabilities when dealing with large amounts of complex data. While human understanding is required, the ability to access and

synthesize information from copious data sources would improve with computer-like abilities. It would be a painfully slow process for a human to check months network traffic logs to attain the appropriate mental, network status "picture".

- Secondly, due a lack of the appropriate senses for cyberspace, humans lack an accurate model of the "cyberspace world." As such, attaining an accurate "mental picture" becomes difficult and the perspective is prone to error. Just as if a baby were exposed to holographic fire their entire life, as an adult, he might make a deadly decision about evacuating from a house fire.

2.1.1.3 Decide. Determining a course of action based on a current mental perspective [17] is the essence of the decide phase. Besides the aforementioned difficulties making appropriate decisions with imperfect observations and an incorrect orientation, there is a larger difficulty faced at this stage. Poor performance of the decision phase in the cyberspace environment has two contributing factors:

- Due to the revolutionary development and rapid growth of cyberspace, understanding observations or the orientation can be limited if decision makers are unfamiliar with the environment of cyberspace. Misinformation, poorly communicated issues, different understandings of the environment itself, or even a lack of basic technical skill and knowledge all affect decision making. If a briefer is explaining to a commander (a decision maker) a jamming attack on satellite communications but the commander only understands cyberspace as the Internet, the commander may inaccurately order a rerouting of information be taken. Such a re-routing of information over the Internet is simple and automatic. However, re-routing information flow over limited satellite resources could have a far-reaching impact.
- The more obvious issues with decision making in a cyberspace environment is time. In an environment in which operations take place at the speed of light, slowing down the decision making process to any slower speed could have

disastrous consequences. Assume a commuter virus has just attacked an organization's e-mail servers. In the time it takes to compile the data form the orientation into a slide presentation, and brief the aforementioned commander, the viruses could have already taken down the entire system.

2.1.1.4 Act. This is the most critical aspect of the OODA Loop and is where, perhaps, the biggest issues with the application of the theory to cyberspace lie. Despite all the issues presented above, it is an assumption of this research that humans cannot truly act in cyberspace until it is better understood and controllable. Before cyberspace can transition from an environment to a domain, in which an OODA Loop (or any doctrinal practices) could take place, the concept of cyberspace must be explored for answers to the questions: "What is cyberspace?" and "How can it be controlled?"

2.2 Cyberspace

The term "cyberspace" was coined by science fiction writer William Gibson in his 1984 novel *Neuromancer* where he described it as a vast "dataspace" or a "world in wires" [20]. Although Gibson's cyberspace has more in common with the popular movie *The Matrix* than modern networks, parallels can be drawn between such fictional imaginings and reality. In fact the connectivity of Gibson's "world in wires" could be thought of as tantamount to the modern Internet which connects networks of computers to other networks [20]. The Internet currently spans the globe and has become a common tool for facilitating information transfer at the speed of light.

However, although the terms cyberspace and Internet are often used synonymously, they are distinctly different concepts. In 2007, Lt. Col. Forrest B. Hare, working for the US Air Force Cyberspace Task Force, authored a whitepaper entitled *Five Myths of Cyberspace and Cyberpower* [8]. In his second myth, Hare cautions that considering cyberspace to be only the Internet would be "catastrophic" for the United

States [8]. He insists that cyberspace should not be viewed as a "cognitive concept" and states that we must "quickly convey the understanding that the [cyberspace] domain goes well beyond the Internet and is anything but *virtual*... [and] appreciate that the domain is a physically manifested space with closed/wired segments as well as free space segments." [8].

So then, if cyberspace is more than a concept but encompasses more than just the tangible, physical Internet, what *is* cyberspace?

2.2.1 Defining the Undefined. According to the National Military Strategy (NMS) "cyberspace is characterized by the use of electronics and the electromagnetic spectrum (EMS) to store, modify, and exchange data via networked systems and associated physical infrastructures" [6]. Today, communication infrastructures are being assimilated into a vast information medium. It is this new medium, born out of the integration of electronics and computers with analog phone lines, digital data cables, and wireless radio waves which fully represents cyberspace.

Hare goes further to explain how this definition relates cyberspace to other environments. "In the cyberspace domain, the electromagnetic spectrum (EMS) is the maneuver space also governed by laws of physics" [8]. Hare draws an analogy between the electromagnetism of cyberspace and the fluid dynamics of the maritime environments. He further stresses that "just as the boundaries between air and space can be blurred, cyberspace can occur within the other physical domains. If we do not recognize cyberspace as a physical domain, occurring any place where we are interlinking the EMS and electronic systems, we allow for seams and access points for our adversary to hold us at risk" [8].

2.2.2 Domain vs. Environment. To be clear, while the terms environment and domain may seem to be interchangeable, they are, in fact, completely different concepts. Webster's Dictionary defines *domain* as "a territory over which dominion is exercised;" "a sphere of knowledge, influence, or activity" [25]. An *environment* has

Environment	Physical	Physics	Domain
Land	land	dynamics of friction	Land
Sea	water	fluid dynamics	Sea
Air	air	aerodynamics	Air
Space	space	astrodynamics	Space
EMS	EMS	E-M/wave-particle	"Fifth"

Figure 2.1: Cyberspace: Environment & Domain [26]

unique physical properties but, just as land is not naturally color coded by country, environments lack the qualities of a domain until regional influence and control is exercised within. Table 2.1 is provided to help define the environment and domain of cyberspace [26]. Cyberspace is a new *environment* where all imaginable information, whether presented as text, voice, image, sound, or video, can coexist.

2.2.3 Fly & Fight in Cyberspace. In December 2005, Chief of Staff of the Air Force (CSAF) T. Michael Moseley modified the mission of the US Air Force (USAF) "The mission of the United States Air Force is to deliver sovereign options for the defense of the United States of America and its global interests - to fly and fight in Air, Space, and Cyberspace" [35].

To meet this new mission, the USAF must address a very important question: 'can humans wage war in this environment?' Can humans control cyberspace asserting dominion over the environment? Is cyberspace a *domain*? Certainty military related operations can be, and are, executed *using* cyberspace to benefit real-world operations - but is this enough for true control?. A crucial assumption of this research is that it is not enough.

IW/IO are currently undertaken *using* the environment of cyberspace but not *in* a domain of cyberspace. Attacks using cyberspace can be said to simply resolve thought set of specific, EMS-related properties (like, protocols or software rules for networks). A cyber attack is only noticed after it has impacted the physical, real world directly. However, there are no battles *in* cyberspace determining that impact

because humans currently cannot attain dominion over the environment. Humans do not reside *in* the environment so they cannot control it directly.

To illustrate this concept an analogy can be drawn. The difference between fighting a war *using* cyberspace instead of *in* cyberspace can be seen as the difference between *using* SCUD missiles to attack ground forces verse performing dogfights *in* an air battle. Currently, "cyber SCUD missiles" are launched from one computer impacting another computer. The only defense is pre-programmed, "cyber patriot missiles." Both *use* the environment of air but do not actively fight in it. Yes, a resourceful enemy can redesign their "cyber SCUDs" to fly higher, faster, or in a different flight path to avoid such defensive measures. The "cyber patriot missiles" can also be modified and thus the game of "spy vs. spy" continues indefinitely. This analogy is directly mapped to such current IW practices such as computer viruses vs. antivirus software and remote computer control vs. intrusion detection.

Just as with the "cyber SCUD" scenario, the parallel strategy is also assumed to be true. To wage war *in* cyberspace humans must gain cyberspace superiority in the same manner in which air superiority is currently attained. A strict interpretation of this scenario indicates that conscious beings must actively fight each other directly *in* the environment for the ability to impact the enemy through that domain. Thus, cyberspace superiority requires, just as with air superiority, that real-world impact should only resolve after conscious battles are waged *in* that environment. A simple examination of the facts reveals that there are currently no such cyber dogfights for cyberspace superiority because there are no conscious or intelligent beings *in* cyberspace providing human influence over that environment. It seems that, upon review of the available definitions, a strict interpretation of the concepts concerning domain and environment yields the information that cyberspace is not *technically* a domain — yet.

Humans must implement some way to understand the environment of cyberspace, this intangible world, to efficiently and effectively practice dominion and claim cy-

berspace superiority. So then, the new question becomes "how can humans begin to truly control cyberspace?" If cyberspace superiority is the answer [32], how can humans enter into, and do combat within, cyberspace. Tanks, ships, and aircraft currently carry human consciousness into the respective environments (creating a domain thereof). However, there does not currently exist any technology, any true "cybercraft," that can carry a human *into* cyberspace. Any human actively operating a computer is still allowing cyber attacks to resolve in the real-world (on that computer) before the human even knows to react. The cyber-OODA loop needs to run faster, and that means limiting reliance on the human equation [26]. But how can you remove the only intelligent being from the decision making process?

The problem is thus distilled: Control of cyberspace from *within* the environment is required for the establishment of a cyberspace domain with appropriate doctrine. Only then can IW/IO be performed for true cyberspace superiority. America needs autonomous, intelligent cybercraft to dominate cyberspace.

2.3 Cybercraft

The concept of the cybercraft, as imagined in the previous section, was first taken from the realm of science fiction to applicable science theory by Dr. Paul W. Phister, Jr. and his team at the Air Force's Research Laboratory (AFRL) in Rome, New York [28]. In 2004, Phister published his paper *CyberCraft: Concept Linking NCW Principles with the Cyber Domain in an Urban Operational Environment.* In this paper Phister defines a vision for a working cybercraft and outlines the research required for such an endeavor.

2.3.1 Vision. Ultimately, cybercraft essentially will enable the transition of cyberspace from an environment to a domain, facilitating cyberspace superiority [32]. Phister envisions that cybercraft will use "the cyber domain to conduct military operations within a military environment...[They will have] significant potential to create the desired effects with either little or minimal collateral damage" [28].

In Phister's vision, cybercraft are essentially command, control, and communications (C3) platforms. They provide a view of cyberspace and autonomously achieve mission objectives through a pre-loaded set of payloads.

Phister states the characteristics of cybercraft will include "the ability to be launched from a network platform, the ability to embed control instructions within the craft, the ability to positively control the cybercraft from a remote network location, the capability for the craft to self-destruct upon being recognized, the capability for the craft to operate with minimal or no signature/footprint, and the ability for the cybercraft to rendezvous and cooperate with other friendly cybercraft" [28].

Having an achievable concept is one thing, planning to build it is a far more difficult task.

2.3.2 Specification: Six Focus Areas. With the concept of a cybercraft preliminarily defined, the process of construction could begin. To this end, Phister posed 6 crucial questions of development:

1. How can we "trust" the "cybercraft" to "do the right thing"?
2. How do you control the "cybercraft"?
3. How can a "cybercraft" determine the "landscape" or "terrain" of an adversary's network?
4. How do you provide stealthy feedback mechanisms?
5. What would be possible missions of the "cybercraft"?
6. What effect measures would the "cybercraft" have to gather? [28]

To help answer these questions, the cybercraft initiative has re-defined them as six fundamental focus areas summarized below:

- **Map and Mission Context:** This area focuses on creating a strategic and operational picture of cyberspace. This requires "[combining] data to paint

a single multilayered Common Operating Picture (COP) of the [new] cyber-domain" [32]. The idea here gets back to doctrine and strategy in the cyber-domain. A traditional mapping of mission context would be a Civil War era General ordered to defend a local town. The General would undoubtedly use current doctrine and strategy to take certain hills based on the most current maps and reconnaissance. In the same way, a commander in charge of defending cyberspace will need to communicate his mission objectives and intentions based on an understanding of the current cyberspace "maps," "weather," enemy movements," etc.

- **Environment Description:** This area focuses on giving the cybercraft the ability to, ultimately, describe its environment to leadership for strategic and operational planning. This area "is closely tied with mapping, as the system uses the description of the environment to graphically display it to the user" [32]. Developing a way to understand what the cybercraft is "seeing" is important as humans have no way to see cyberspace that way a cybercraft will. As a seeing-eye-dog communicates the state (dangerous or safe) of a crosswalk to a blind man, so too will the cybercraft need to communicate the state of cyberspace to its operators.
- **C3 Protocols and Architecture:** This area requires the development of C3 protocols to facilitate coordinated cybercraft operations based on the mission context [32]. Just as dispersed naval ships or aircraft require the ability to communicate with leadership, so too do cybercraft. In the same way, cybercraft also require rules and regulations from which to operate by in the case of a loss of communications, enemy detection, capture, etc.
- **Formal Model and Policy:** Here the goal is to prove the cybercraft does what it should. A formal model and policy for Cybercraft must be created to assure leadership that cybercraft behavior conforms to the commander's intent. To this end, a "formal model must be built to describe the set of states that the cybercraft can be in [this model] must mathematically prove [all] the state

transitions...so thatthe system is predictable" [32]. Once a policy is in place, the cybercraft would become provably reliable, a far cry from their human operators.

- **Self Protection Guarantee:** Tied to both the Formal Model and Policy area and the C3 Protocols and Architecture area, this area focuses directly on defining cybercraft characteristics required to "conduct assured operations" [32]. This incorporates (but is not necessarily limited to) anti- tamper/software protection research [32]. Additionally, there must specifically be some "mechanism to identify a compromised agent so that a compromised agent does not pollute the data used by [other cybercraft]" [32].
- **Interfaces and Payloads:** This last area, also related to C3 Protocols and Architecture, is tasked with creating standard, extendable, flexible interfaces. Basic interfaces "between the agent and the host OS, the agent and the network, and the payloads" will be required [32]. Interfaces will need to evolve with the rapidly changing cyberspace environment and perpetually changing missions and payloads.

2.3.3 Autonomy Challenges. As cybercraft development breaks into six, interdependent, focus areas, the need for new research in certain areas becomes apparent. Based on an AFRL/IF MURI proposal, Phister outlined 7 areas of research that should be pursued to answer the crucial development questions [28]:

> 1) Simulations of multiple, interdependent infrastructures. Includes research into interdependencies and emergent behaviors of complex adaptive systems;
>
> 2) Basic research that connects decision-making behaviors (desired political-military outcomes at the operational and strategic levels) to specific physical effects (operations and military actions);
>
> 3) Intelligent agent based systems to collaborate, coordinate and solve problems, automatically without human intervention. These agent based systems will have the ability to sense there environment and based on goals and constraints, provided by the user, achieve the objectives assigned;
>
> 4) Real-time updating of simulations. Includes real-time data ingestion and updating, dating mining, data validation, and methods of handling extremely large, dynamic datasets;

5) Self-organized modeling with the basic ability to have the models automatically organize themselves based on present conditions and predict the future battlespace environment;

6) Cyber defense and offense techniques including new ways of detecting attacks and executing attacks, countering adversary attacks, responding, performing forensics and anti-forensics and gaining real-time cyber situational awareness/understanding; and,

7) C2 theories such as control theory, uncertainty management and decision making theory.

[28]

Prominent in areas of research above, and potentially required by all six focus areas, is a call for exploration into *intelligent* autonomy of the cybercraft, not mere autonomy. Dr Stephan Kolitz and Dr. Michael Richard define this concept of intelligent autonomy as "the ability to plan and execute complex activities in a manner that provides rapid, effective response to stochastic and dynamic mission events. Thus, intelligent autonomy enables the high-level reasoning and adaptive behavior for an unmanned vehicle..." [30]. Plucking requirements from Phister's research areas above clarifies the need for cybercraft autonomy and intelligent reasoning.

This thesis breaks Phister's specifications down in to 7 Challenges to Intelligent Cybercraft Autonomy.

Cybercraft will:

1. Have "the ability to sense their environment."
2. Gain "real-time cyber situational awareness/understanding."
3. Be able to "collaborate, coordinate and solve problems."
4. Use known "goals and constraints... [to] achieve the objectives assigned."
5. Be able to apply "decision-making behaviors... [for] specific physical effects."
6. Provide "new ways of detecting attacks, ... countering adversary attacks, responding[to/recovering from attacks], ...and performing [cyber-] forensics."
7. Do all this "automatically without [much] human intervention."

Indeed, this appears that each challenge to intelligent cybercraft autonomy is directly related to the three primary challenges to intelligent autonomy as outlined by Kolitz and Richard, which are:

> 1) Developing and executing plans of activities that meet mission objectives and honor constraints.
> 2) Dealing with uncertainty.
> 3) Providing a capability for dynamically adjusting a vehicle's plan in real time.
> [30]

With regards to these 7 challenges to intelligent cybercraft autonomy, challenge 7 relates directly to the second intelligent autonomy challenge. Challenges 4-6 and 1-3 apply to the first and third intelligent autonomy challenges, respectively. Yet, engineers cannot simply replicate intelligence and "add it"' to autonomous protocols. No, intelligence itself should first be understood before any engineering of artificial intelligence or evaluation of theories can take place.

2.4 Intelligence

Before attempting to create "autonomous, intelligent cybercraft to dominate cyberspace," the term *intelligence* must be discussed. Merriam-Webster's Dictionary defines intelligence as "the ability to learn or understand or to deal with new or trying situations to apply knowledge to manipulate one's environment or to think abstractly as measured by objective criteria; the skilled use of reason" [25]. Yet, a simple dictionary solution is far from sufficient to completely encompass this mammoth concept.

Indeed, in 1986 two dozen prominent theorists were asked to define intelligence. It came as no surprise that they gave an equal number of different definitions of the concept [27]. Although agreement on the nature of intelligence remains eternally shrouded in philosophical and scientific controversy, this thesis attempts to disambiguate the term so that theories of artificial intelligence can be presented.

2.4.1 Intelligent Behavior & The Turing Test. Alan Turing, inventor of the imaginary *Universal Turing Machine*, is regarded to be the first to tackle the question "Can machines think?" [34] Turing first had to identify what he thought it meant to "think" [11]. He proposed that thinking was the inevitable act performed during any question and answer discourse among humans. "The question and answer method," Turing deduced, "seems to be suitable for introducing almost any one of the fields of human endeavor that we wish to include" [34]. Turing's test for intelligence, which he called the "Imitation Game" was thus formed and proceeds as follows:

> It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart front the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A." [34]

Turing implies that a similar game consisting of a machine (A) and any human (B) would force the interrogator (C) to conclude "A is indeed intelligent" if C could not, short of random guessing, honestly determine if X or Y were certainly B. To complete the illusion, Turing proposed that tones of voice must not help the interrogator and that "the answers should be written, or better still, typewritten. The ideal arrangement is to have a teleprinter communicating between the two rooms" [34]. Today, the idea has evolved to something similar to the imitation game being performed via some instant messaging system.

2.4.2 Prediction: The Essence Of Intelligence. Proposals explaining the inability of previous AI attempts to pass Turing's test range from a lack of computational power to an argument that the Turing Test itself, which defined and shaped AI theory, is wrong [11]. This research follows the later notion, that Turing's test might be premature. While Turing suggested that behavior is an indicator of intelligence [11], it has been proposed this alone is not the true essence of intelligence.

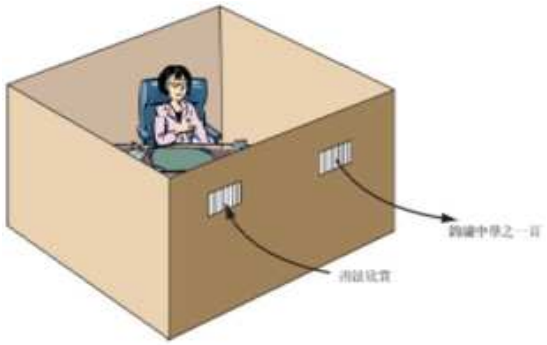

Figure 2.2: Chinese Room Thought Experiment [1]

"Cogito, ergo sum". "I think, therefore I am," was a philosophical phrase, first used by Ren Descartes, which can help sum up why Turing's test for intelligence fails. While the Turing Test focused on behavior, it neglects the fact that the *act of thinking* requires no behavior. Can an entity be conscious or intelligent and exhibit no behavior to indicate it? This is exactly what happens when you lay on your bed in a dark room and think [11]. So what separates a human lying on the bed thinking about astrophysics and a computer calculating pi? The simple answer is *thought*, understanding, sapience.

But if a lack of behavior in intelligent beings illustrates the error in designing intelligent computers to take the Turing Test, does that invalidate the test itself? If a computer could pass the test would it not still be intelligent? John Searle, in his famed article *Minds, Brains, and Programs*, proposes the "Chinese Room" mind experiment to argue that passing the Turing Test does not constitute an intelligence. Only *understanding* (being sapient) can define intelligence. The experiment's presentation is thus summarized:

> You are locked in a room and given a large batch of Chinese characters together with a set of English rules for correlating said characters. No explanation is given, simply instructions like "when you see this set of characters write this set." Suppose furthermore that you know no Chinese, either written or spoken. To you, Chinese writing is just so many meaningless squiggles. The instructions enable you to correlate one set of formal symbols with another set of formal symbols and so on. Now imagine that people provide you sheets of paper with Chinese sentences

> on them through a slot in the room. Unknown to you, this is a story, written in Chinese followed by a set of questions also written in Chinese. You take the paper and transcribe symbols at the bottom as the instruction book indicates and pass the paper back through the slot. You have written answers to the questions which are absolutely indistinguishable from those of native Chinese speakers. Nobody just looking at the answers can tell that you don't speak a word of Chinese. From the external point of view (e.g. from the point of view of someone reading the answers), are these solutions to the Chinese questions are correct. As far as understanding Chinese is concerned, you don't because you have simply behaved like a computer; performed computational operations on formally specified elements. [11, 31]

With this experiment Searle illustrates that any computer passing the Turing Test is not (necessarily) intelligent because it lacks understanding. As shown, understanding is a requirement of sapience and intelligence. Therefore, the Turing Test does not screen for intelligence.

Jeff Hawkins has proposed a new test for intelligence. Instead of looking at what demonstrates *thinking* in an intelligent being, Hawkins tried to determine what would show *understanding* [11]. The conclusion proposed is *prediction*. Hawkins proposes that prediction is the essence of intelligence. Hawkins is not alone in his conviction that prediction is the root of sapience and intelligence. Calvin also forwarded a complimenting theory saying "This idea neatly covers a lot of ground: finding the solution to a problem or the logic of an argument, happening on an appropriate analogy, creating a pleasing harmony or guessing what's likely to happen next" [3]. He observed "we all routinely predict what comes next, even when passively listening to a narrative or a melody. That's why a joke's punch line or a P.D.Q. Bach musical parody brings you up short—you were subconsciously predicting something else and were surprised by the mismatch" [3]. Notable neurobiologist Horace Barlow of the University of Cambridge framed his agreement suggesting that "intelligence is all about making a guess that discovers some new underlying order" [3]. To illustrate Hawkins' "prediction is intelligence" theory he propose the following thought experiment:

> When you come home each day, you usually take a few seconds to go through your front door. You reach out, turn the knob, walk in, and shut it behind you. It's a firmly established habit, something you do all the time and pay little attention to. Suppose while you're out, someone sneaks over to your house and changes something about your door. It could be almost anything. The knob could be moved over an inch, changed from a round knob to a thumb latch, or changed from brass to chrome. The door's weight or color could be changed, hinges could be made squeaky and stiff, or a peephole could be replace by a window. When you come home that evening and attempt to open the door, you would quickly detect that something is wrong. It might take you a few seconds' reflection to realize what exactly had changed. [11]

But the fact is you *have* noticed a change. You noticed a change because you had an expectation, you made a *prediction*, of what you would encounter as you walked through the door. While prediction thus seems to proceed behavior, the question may still exist: How does prediction lead to intelligent understanding with or without behavior? The answer is in the above thought experiment.

Can it not be said that you *understood* your door? Better still, tie prediction into the Chinese Room experiment. Would you, as the transcriber of Chinese characters, not *understand* Chinese if you could make some prediction of what to expect after each character, phrase, sentence, etc.? In English, if you were told the Chinese story: "Jack and Jill went up the" you would predict the word "hill." You probably *did* just predict exactly that word in your head before you saw the word written—assuming you had ever heard the nursery rhyme. You then would probably make a prediction of what Jack and Jill were going to do. Furthermore, you could predict a moral for the story and even answer questions asked of you about the moral of the story. You can do all of this because of your ability to predict the outcome based on a learned experience.

The analogy between prediction, intelligence, behavior, and current AI methods becomes clear. It will take you a fraction of the time it would take a computer to discover what was wrong with the door (it is too heavy), predict a solution to the new

input (don't push as hard), and proceed with your intelligent behavior of entering the house. A computer robot would simply fall down.

It is now clear why current AI approaches have seen limited success. When you walk thought your door you are not cycling though all the endless possibilities of doors to see if the door has changed. No, you make quick, intuitive, accurate *predictions* on what the door will look, sound, and feel like from memory. Prediction is at the heart of Hawkins cortical theory and, indeed, prediction (a strictly mental or expressed in behavior) is the proposed yardstick to determine intelligence — that understanding has occurred .

2.4.3 The Human Brain. But how are computers supposed to be built to emulate the predictive elements that seem to be the basis for human intelligence? The first step is to replicate the how the human brain functions. Using the human brain as a constraint and a guide (not as an antiquated model of an intelligent machine, as traditional AI approaches do) intelligent computers could be created.

About 60% of human brain is composed of a component called the neocortex (or simply "cortex"') [9]. The cortex is the part of the brain responsible for almost all high-level thought and perception in humans [9]. Because such sapient brain functions have been shown to resolve to the *prediction* element of the intelligence equation, modeling a computer after the construction of the cortex is a logical first step in creating intelligent agents.

To create such a model, a working theory of how the brain operates from a functional and algorithmic level must first be deduced. Jeff Hawkins claims to have proposed the world's first comprehensive theory on neocortical function [11]. Because of its uniform structure, "neuroscientists have long suspected that all its parts work on a common algorithm" [9]. Conceptually, this means the brain "hears, sees, understands language, and even plays chess with a single, flexible tool" [9]. Hawkins has gone further and proposed that an auto-associative hierarchy, based on both spatial and temporal patterns, is responsible for all memory storage and cognitive (predic-

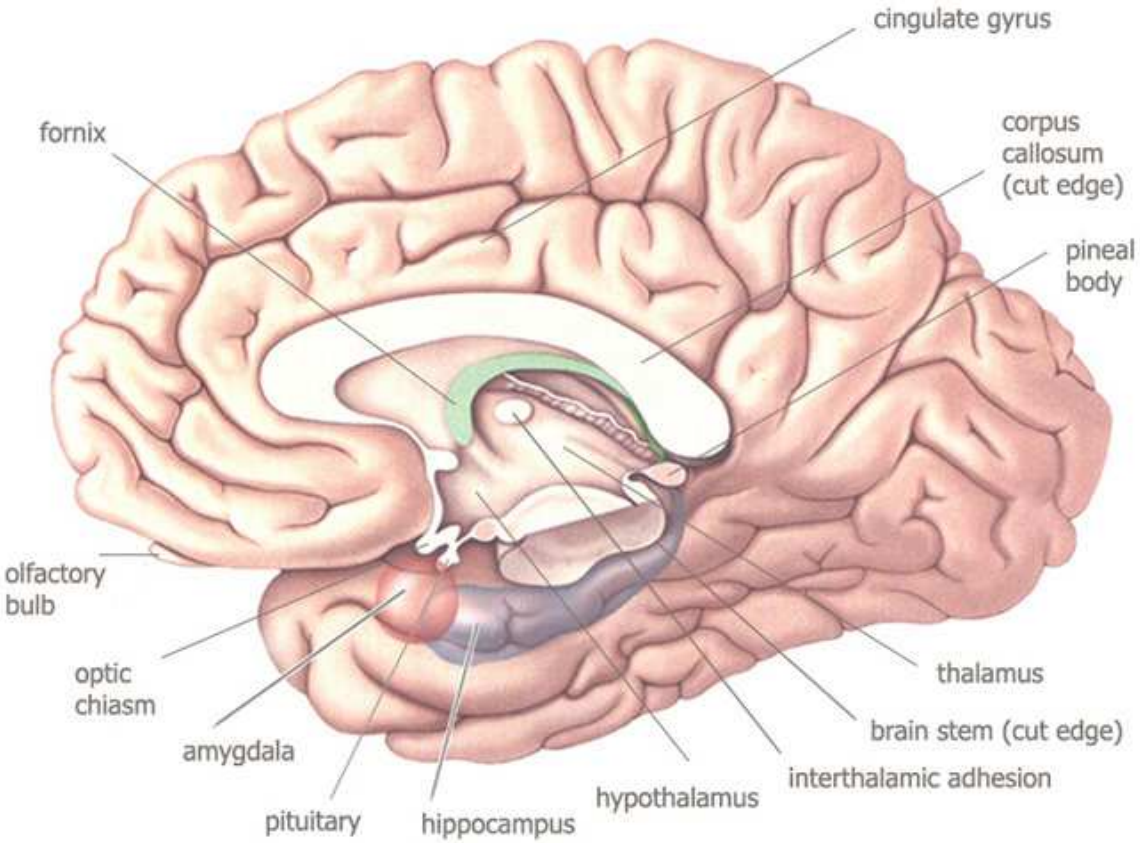

Figure 2.3: Parts of the Human Brain

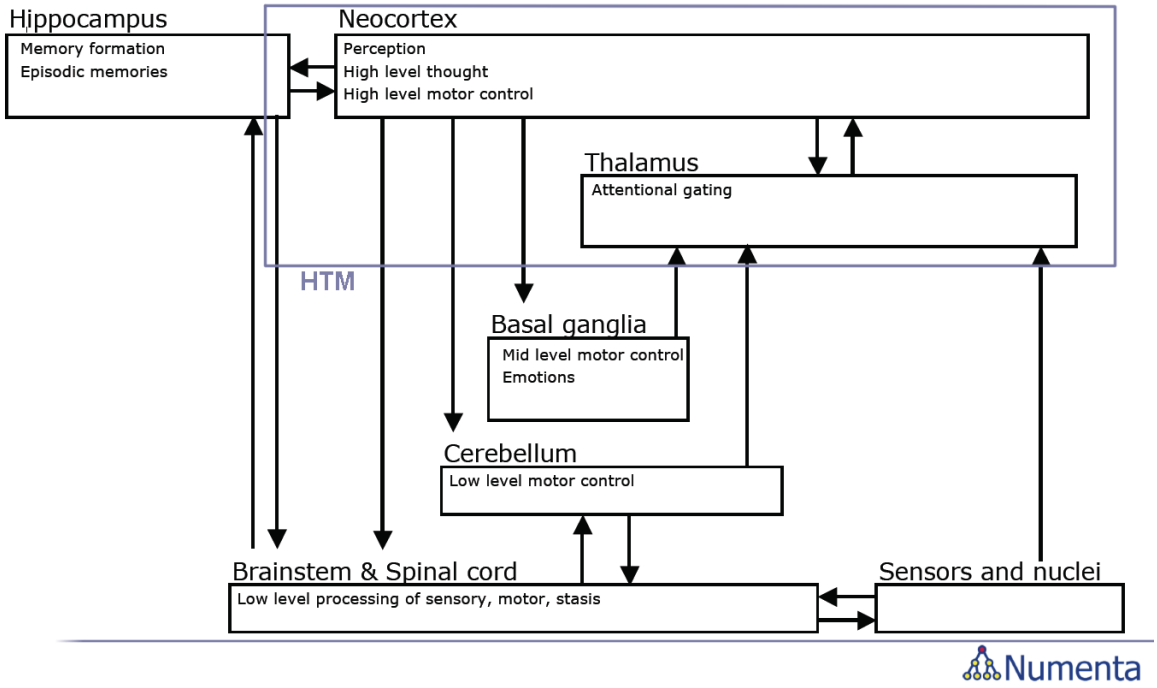

Figure 2.4: Cortical Theory Flow Chart [14]

tive) behavior in humans [11]. To summarize the cortical theory at an elementary level, sensors (eyes, ears, skin, etc.) send a signal to a neuron, a memory element (or node) in the brain. If the sensor is an eye and it sees a German Shepherd, the signal which represents "seeing the dog's tooth" will be sent to a memory node. "Nearby" memory nodes may get signals for another "tooth", "a lip", "gums", etc (See Figure 2.5 - Level 1). These all send what they perceive up to a higher memory node which perceives a "jaw." This node, in conjunction with other nodes, will send similar representations up to another, higher node which may perceive "a dog's head" (See Figure 2.5 - Level 2). The process continues until the perception or understanding of seeing a "German Shepherd" is attained (See Figure 2.5 - Level 3) [9].

Critical to this process is auto-associative feedback based on spatial-temporal patterns or learned experiences. At each level the nodes send information back down to the lower nodes, essentially saying "I have seen this before, it is a 'dog head' and you can expect to see a 'dog tail' if you look at the other end of the dog." This information is passed down in the hierarchy as appropriate sub-representations until

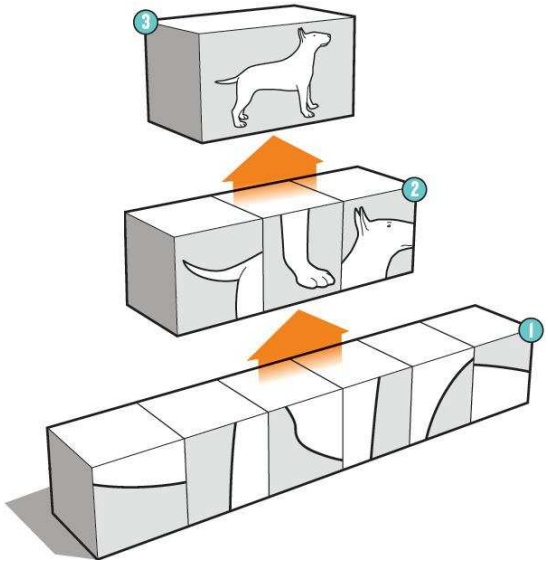

Figure 2.5: Neocortical/HTM Theory - Visual Sensor: Perception & Prediction [9]

an eye sensor gets the command to look at the dog's tail for verification that a dog is, indeed, perceived - it *predicts* that a "dog tail" will be seen at a certain location by the sensor [9, 11].

While this explanation is simplistic, the basic concepts are sound. Because this thesis is not biological in nature, going into the Hawkins' explanation on the physical structures and interactions of neural cells is not required. However, Hawkins' theory is exhaustive with respect to the functional aspects of his brain theory and the direct mapping of those aspects to physical cellular structures in the brain [11]. In particular, evidence for the hierarchical structure of Hawkins' theory, along with the principle of feedback, can be seen in the cortex's primary neural structure, a hierarchical, 6-layered "column" (Figure /redbraincols) [11].

A algorithm built on this memory framework should allow software to perceive, predict, and understand its world. Hawkins' dubbed this memory framework Hierarchical Temporal Memory (HTM). An overview of the key components of HTM follows in Section 2.6

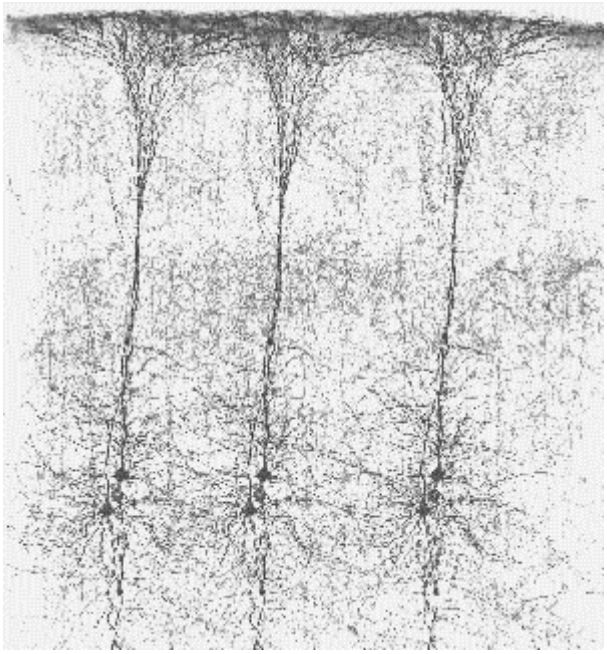

Figure 2.6: Hierarchical Brain Neurons [9]

2.5 Pertinent Artificial Intelligence Approaches In-summary

AI research has seen great attention since the publication of Turing's *Computing Machinery and Intelligence* in 1950. Since the late '50s, algorithms have been designed around theories of human brain function and structure. However, according to Hawkins, because of the behavioral focus of the Turing Test, AI research has traditionally taken an approach of imitating human behavior [11]. Not until recently have purely predictive cortical theories been adapted into computer algorithms [11].

Although many promises of AI have gone unfulfilled to date, many of the discipline's methods and concepts will play an important foundation role in any new research. The most pertinent of AI approaches are summarized here.

2.5.1 Artificial Neural Networks. Artificial Neural Networks (ANNs) are "parallel, distributed information processing structure consisting of processing elements (which can possess a local memory and can carry out localized information processing operations) interconnected together with unidirectional signal channels called connections" [12].

Each processing element outputs a signal to any number of other elements [12]. Incoming signals are only able to be processed with local element data or currently incoming data [12]. In this way networks of elements are connected together to process and feed data forward in the network. In addition to feed forward connections, each element is capable of sending backwards a simple "error feedback" signal during network training [12].

Modifications to ANNs result in a wide assortment of algorithms. Backpropagation ANNs use a cycle of input, output, and error signals to tune and generate appropriate weights. The cycles can be repeated for durations but ultimately results in "inputs to the network 'bubbling up' from the bottom to the top and then the errors 'percolating down' from the top to the bottom" [12]. Careful tuning or "training" of the each element (including all the internal or 'hidden' elements) results in limited categorization of a diverse set of inputs.

Even with backpropagation, ANNs still require humans to be highly involved in the tuning, training (weighting) of the network elements to afford the appropriate outputs. ANNs can have difficulty adapting and filtering new, heterogeneous inputs accurately.

2.5.2 Neuro-Fuzzy Networks. "Fuzzy logic" was introduced in 1965 by Lotfi Zadeh whose fuzzy set theory provided a way of "characterizing non-probabilistic uncertainties" [18]. The categorization of unpredictable inputs fits AI so well that hybrid ANNs, known as Neuro-Fuzzy Networks (NFNs), were soon developed. In their purest form, NFNs are ANNs that use fuzzy logic rules to derive fuzzy sets.

Fuzzy reasoning allows implications on a ranging scale. If, for example, given the true implication fact "if the tomato is red then it is ripe." Fuzzy logic would allow inference from that fact to a true rule of (say) "the tomato is 'kind of' red then it is 'kind of' ripe" [18]. Fuzzy reasoning (or "approximate reasoning") [18] can be used to "derive conclusions from a set of fuzzy if-then rules." ANN backpropagation training

techniques allows the rules of such a fuzzy inference systems (FIS) to be more easily defined.

2.5.3 Bayesian Networks. A Bayesian network is "an annotated directed graph that encodes probabilistic relationships among distinctions of interest in an uncertain-reasoning problem" [13]. In the early 1990s, AI researchers began to use these probabilistic models to develop new learning methods for the networks [13, 19]. Bayesian networks were infused with prior knowledge of a data set to provide probabilistic reasoning of the data. So then a learning Bayesian network amounts to "searching for network-structure hypotheses with high relative posterior probabilities" [13].

Bayesian networks are, at their hearts, decision graphs [19]. If there were a "wine tasting" Bayesian network, wine might be broken down into whites and reds. A certain percentage of "warm fruit," "oak flavor," and "buttery finish" in a blind taste of wine might lead such a Bayesian network to conclude that a particular wine is a white (with 95% certainty), and specifically a chardonnay (with, 80% certainty). If the Bayesian network were pre-trained with enough information it could potentially choose between vineyards or even vintages. As such, the main purpose of a Bayesian networks is to "give estimates of certainties for events" /citebn2.

A dynamic Bayesian network is a Bayesian network that "represents sequences of variables." These sequences commonly include data that is in a time-series, like human speech. The ability to deal with such data makes dynamic Bayesian network useful for solving many time-linked problems, including speech recognition [17].

2.5.4 Hidden Markov Models. The Hidden Markov Model (HMM) is based on a Markov assumption. At the simplest level, this process uses a known set of data to probabilistically determine unknown or "hidden" data based on observable patterns [5, 17]. The hidden Markov model can be considered as the "most simple dynamic Bayesian network" using time-sequenced variables as initial patterns [17].

HMMs are commonly used to model "stochastic processes" and variable sequences in current speech recognition software [5]. Their multidimensional cousins (i.e. 2D HMMs) are frequently used in picture or handwriting identification applications [5, 22]. HMMs have predetermined topologies and estimated parameters. They also need algorithmic solutions to "acuminating" or "canonical" problems; that is they require problem reduction to simplest form without loss of generality. These problems are [5]:

1. *Calculating the likelihood of a sequence*
2. *Finding the most probable state sequence*
3. *Estimating the parameters of a model*

2.5.5 Hierarchical Temporal Memory. The thesis of this paper revolves around recent AI theory and is the subject of in-depth analysis in later sections. In general, Hierarchical Temporal Memory (HTM) theory combines concepts of ANNs, NFNs, Bayesian networks, and HMMs.

Like ANNs, HTM networks have many processing elements or nodes but HTM networks are always organized as a tree-shaped hierarchy. Also like ANNs, backpropagation feedback techniques are also used but play a far more critical role. Unlike some ANNs, each node implements a "common learning and memory function" [10]. Additionally, similar to NFNs, HTM networks have an inference ability that allows them to categorize uncertain data.

HTMs are similar to Bayesian networks in structure and topology. However, HTM networks differ from Bayesian networks in the way that "time, hierarchy, action, and attention are used" [10]. Further, compared to HMMs, the temporal and feedback aspects in an HTM network have a more centralize role. Unlike HMMs, HTM networks do not calculate futures states based on a single past state.

2.6 HTM Theory In-depth

It is common for designers to "seek a good combination of simplicity and power; in other words, elegance...only the field as a whole can determine whether they have succeeded" [4]. HTM theory does not claim to be definitively correct or to be a completely new idea. It is a combination of old AI methods, new theories of intelligence, and innovative ideas on human brain function.

2.6.1 Nuts & Bolts. HTM Theory is so named for the three unique qualities of the theory which combine to create this new idea.

- **Hierarchical:**

  As the name implies, HTM Networks are organized as a tree-shaped hierarchy of sensor and memory nodes similar to Bayesian Networks [9,14]. An HTM Network requires a connected graph where "each node in the graph represents a belief or set of beliefs" [10]. From earlier in the paper these beliefs were known as *invariant representations* [11]. "Lower-level nodes receive large amounts of input and send processed input up to the next level" [14]. In this way, HTM Networks abstract the information and are able to pass that information up the hierarchy as an invariant representation [9] (See Figure 2.7).

  There are two primary types of nodes in an HTM Network hierarchy:

  – *Memory Nodes* are nodes which process sensory data using both spatial and temporal algorithms. If a pattern is recognized its invariant representation is sent up the hierarchy, if the data seems novel, a representation is created and then passed up as a seemingly new representation. [9–11, 14]

  – *Sensor Nodes* are the nodes which send spatial data, based on environmental inputs, up the hierarchy of memory nodes. These are equivalent

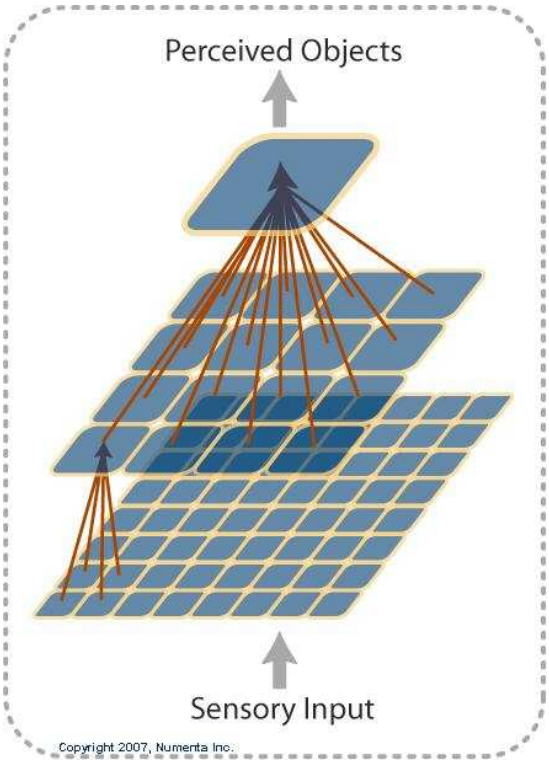

Figure 2.7: HTM Theory - Beliefs & Invariant Representations [1]

to the eyes or ears in humans but, more critically to a cyberspace application, could be programmed to detect any environmental stimulus (i.e. TCP packets, network traffic volume, destinations, etc.). [9–11, 14]

- **Temporal** *(Spatial-Temporal patterns)* **:**

  HTM applications are presented with spatial data as it changes over time. Figure 2.7 is an illustration of an instance of spatial data as it is presented up the hierarchy. The lowest memory node receives sensor data from sensor nodes. Each node essentially "votes" on what it believes to be the correct invariant representation of the spatial data based on prior memory and (most critically) temporal patterns, and passes that up to the next memory node in turn [9–11, 14].

  This temporal element is critical. Both HTM and the theoretical cortical algorithm expect sensory input that changes gradually over time. With respect

to the nursery rhyme example from earlier, perceiving the invariant representation of "Jack" may mean nothing or literally *nothing*. It is the spatial-temporal pattern associated with the word, the invariant representation, that defines it. Learning a nursery rhyme as a sequence of words enables HTM technology to simultaneously predict the following words "and Jill" but create an invariant representation of *nursery rhyme*. [9–11, 14]

- **Memory:**

  HTM applications work in two stages. In the first stage the network's memory is given training (either supervised or unsupervised). The second stage uses that memory to interpret new inputs and potentially continue learning. "During training, the HTM Network learns to recognize patterns in the input it receives. In the fully trained HTM Network, each level in the hierarchy knows - has in memory - all the [invariant object representations] in its world" [14].

  There are two key concepts critical to each stage:

  - *Invariant Representations:*

    In HTM Networks, invariant representations (IRs) are based on the spatial and temporal patterns presented up the hierarchy from sensor nodes. In the example above the invariant representation of "dog head" is stored at a mid-level memory node while the invariant representation of "German Shepherd" is stored at the topmost memory node. How invariant representations are learned and stored depend on the implementation, however, Figure 2.8 is an illustration of the concept of IR belief in HTM networks. [9–11, 14]

  - *Auto-Associative:*

    HTM theory is loosely based on auto-associative neural networks [11]. The concept of auto-associative memory in HTM can be abstracted and summarized to mean feedback which is pushed down to the lower levels

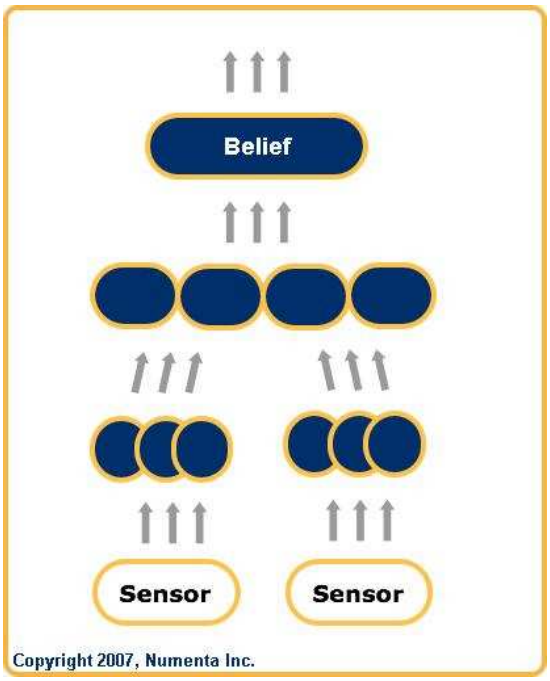

Figure 2.8: HTM Theory - Hierarchical Networks [1]

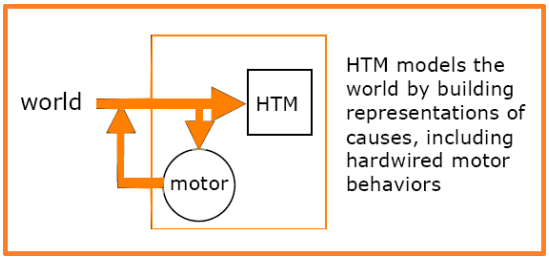

Figure 2.9: HTM Theory - Auto-Associative Memory [10]

at each stage of the hierarchy. This allows for the important "predictive" nature of HTM networks and is critical to its application for creating and predicting invariant representations [11]. Expected invariant representations of objects are fed back down the hierarchy and used to create, perceive, predict, and act upon spatial and temporal patterns (See Figure 2.9).

HTM applications are proposed as software solutions based on Hawkins' cortical theory. This would allow computers to theoretically perceive, predict, and interact with the world. Under the definitions and assumptions in this research, such software

could technically become sapient, possess conscious understanding, and be intelligent. Cyberspace software programmed to use HTM Networks might actually allow for the first truly intelligent computer to understand cyberspace. Such intelligence applied to a cybercraft could meet the aforementioned requirements for cyberspace superiority.

2.6.2 Reality Check. HTM theory is not a "magic key," to unlock the secrets of all previously unsolvable problems. The biggest constraint on HTM networks is the viability of training and learning. As explained in Section 2.4.5, HTM networks are based on the neocortex in the human brain. As such, the problems for which an HTM network solution is appropriate should fall into one of the following two categories [16]:

- **Inferring the Cause of Novel Input** [16]. This category covers those functions which humans perform very well but that computers cannot do reliably (i.e. facial recognition, environmental interaction/reaction, etc.).
- **Discovering Causes in Sensory Data** [16]. This category covers those areas where human-like understanding and inference would enable computers possessing exceptional sensors and/or residing in unique environments the ability to discern inputs and make predictions in unusual worlds.

It seems that aspects of creating intelligent cyberspace software agents fall into not one, but both categories above. However, this is not enough to select HTM technology as a solution for the problem. The feasibility of training the HTM network must also be considered. The two primary factors are provided:

- The *time* allotted for training is critical [16]. Because both humans and HTM Networks require training before they are able to reliably solve problems it is crucial that the time needed to train the HTM network effectively be available. Some problems require a one-time training session of a few moments while others may require hours or days of training or training that, like humans, continues indefinitely.

- The *alternatives* to solving the problem with HTM technology must also be considered. Problems that can be currently solved, to similar degrees of accuracy, with modern computer science applications and techniques may not be an appropriate use of HTM Networks [16]. Although the problem could be solved with HTM technology it doesn't mean it should.

If HTM is a viable option, if there exists enough training time, and if current techniques fall well short of the desired result, then HTM technology should be a good fit to solve the problem. However, the data needed to train the HTM network should meet the following criteria:

- The *quantity* of data available for training is key [16]. Many problems, while perfect for a HTM solution, lack the vast amounts of data required for the network to build correct invariant representations of the world to which it is exposed.
- The *quality* of the data is important as well [16]. Validation to the accuracy of sensory input data should be performed to eliminate the potential of tainted data. If an HTM network were to be given flawed input to sensors it would perceive a different world model than the one which was intended.
- Most importantly the data should represent a *spatial-temporal hierarchy* which corresponds to the world from which it was taken [16]. As defined in Section III, the concepts of spatial and temporal hierarchies are crucial to both human and HTM network learning.

If training an HTM with vast amounts of accurate, spatial-temporal data is feasible and suitable for the problem, HTM technology is a good fit [16]. Provided suitable data can be obtained, training intelligent software to reside/react in cyberspace could be an attainable goal. The next chapter explores the challenges of implementing HTM theory.

III. Methodology - Implementing HTMs

HTM applications are proposed as software solutions based on Hawkins' cortical theory. This would allow computers to theoretically perceive, predict, and interact with unusual worlds. Perhaps software programmed with HTM networks could help addressing the three challenges to intelligent autonomy. Such capabilities are central to cybercraft, the understanding cyberspace, and providing superiority in such a domain.

The ultimate goal of *this* research is not production of such intelligent, autonomous cybercraft. Sapient cybercraft is a vision which Chapter 2 indicates might be attained with by concentrating on the 7 challenges to intelligent cybercraft autonomy. This paper can provide important steps towards the realization of this vision by first refining the problem into a preliminary set of goals and then investigating potential solutions.

3.1 Problem Refinement

If cybercraft are to autonomously defend cyberspace with a reasonable expectation of success, intelligent, reasoning abilities should be acquired. The fundamental research questions of this paper are then: "Can HTM implementations provide an understanding into the abstract world of cyberspace and are the predictive foundations of intelligence supported by such HTM implementations?"

HTM theory claims to combines many positive aspects of proven AI methods. Like many such methods, HTM builds on a comprehensive, biological theory of the only known, functioning, intelligent machine - the human brain. However, unlike many other AI methods, HTM theory is unique in asserting the ability for unsupervised learning of intangible worlds. It further claims the ability to model such worlds with enough accuracy to enable intelligent predictions [11]. Although first proposed in 2004, this particular claim of the theory has gone relatively untested in favor of supervised learning of known worlds (See section 3.4.3) [14].

The problem domain [21] is then formed: Could HTM theory be used as a basis for an intelligent cybercraft? More directly, does HTM theory provide computers or software with predictive abilities, specifically (but not uniquely restricted) to cyberspace? This last question is broken down into 4 specific problem areas that need to be addressed:

Problem 1: Modeling the *unknown environment* of cyberspace could be impossible due to the very ambiguity of the environment.

Problem 2: Making *predictions in an ambiguous model* is mere guesswork if the model of cyberspace is not understood to begin with.

Problem 3: Inability to *act* or *react* based on unknown predictions.

Problem 4: Difficulty *adapting to changes* unanticipated, and thus not represented, in the model without human intervention.

3.2 HTM Theory - Applicability

To answer the problems as proposed, the problems should show applicable mappings [21]to known (supposed) solutions claimed by HTM theory.

Problem 1: HTM theory addressed the generation of a model by allowing the self-generation of invariant representations of the environment via supervised or unsupervised training. Programming a model is not required, observation of a world is all that is needed. HTM theory specifically acknowledges that through multiple sensors but one algorithm, even alien environments like cyberspace can be internalized.

Problem 2: The model itself is generated using the natural spatial-temporal patterns in the observed world. Not only does this generate a known model of the world (at least to the observer) but it directly facilitates predictions based on the presumed reoccurrence of familiar patterns.

Problem 3: In HTM theory, predictions are continuously made and provided as feedback to an HTM network. Actions and reactions to input are simply those predictions. As such, those predictions can change as the input is or is not anticipated.

Problem 4: The ambiguity of inputs can be resolve up the hierarchy of an HTM network thus reducing previously un-modeled inputs to a predictable (if potential and initially incorrect) conclusion [11]. When predictions are shown to be incorrect, continual learning can occur and uncertainty removed from future inputs of that type. Thus, when novel, unanticipated changes occur and ultimately become unpredictable, this new pattern is added to current spatial-temporal patterns. If this novel input occurs enough, it becomes anticipated and predictable. In this way, HTM theory resolves situational adaptation via unsupervised learning without human intervention.

3.3 Hypothesis

Given the above applicability of HTM theory to this problem domain [21], a hypothesis can be stated.

With appropriate spatial-temporal data and sensors, an HTM network should be able to create an internal representation of an environment, make predictions, and take appropriate actions based on said predictions.

The objective is then to test implementations of HTM theory with respect to accuracy of this hypothesis, specifically within a cyberspace environment.

3.4 Mapping Theory to Algorithm

Mapping HTM theory to a working algorithm is the next step [21]. Algorithms can then be implemented and tested with respect to the hypothesis.

3.4.1 Numenta Inc. Since 2003, Hawkins has worked diligently with the Redwood Center for Theoretical Neuroscience at the University of California at Berke-

ley to develop his HTM theory. In, 2005 he co-founded a company called Numenta Inc., dedicated to developing an implementation of his HTM theory [14].

3.4.2 Algorithm Overview. In March of 2007, Numenta released what they claimed was a "research implementation" of HTM theory called Numenta Platform for Intelligent Computing (NuPIC). The algorithm used by NuPIC at this time is called "Zeta1." NuPIC was released as an open source software platform and binary files of the Zeta1 algorithm. Because of licensing, this paper is not allowed to discuss the proprietary implementation aspects of Numenta's Zeta1 algorithm [14]. There are, however, generalized concepts of implementation that can be discussed freely. The two most important of these are how the Zeta 1 algorithm (encapsulated in each memory node of the network hierarchy) implements HTM theory.

To implement any theory in software, an algorithmic design for each aspect of the theory must be addressed [21]. The most important design decision Numenta adopted was to eliminate feedback within the hierarchy and instead choose to simulate this theoretical concept using only data pooling algorithms for weighting [15]. This decision is immediately suspect and violates key concepts of HTM. Feedback, Hawkins' insists, is vital to cortical function and central to his theories. Still, Numenta claims that most HTM applicable problems can be solved using their implementation and proprietary pooling algorithms [14].

Additionally, NuPIC implement these independent temporal and special pooling algorithms at each node [15]. Numenta's white paper entitled *HTM Learning Algorithms* discuses these concepts (See Figure 3.1):

- **Spatial Pooler:** Learns a mapping from a potentially infinite number of input patterns to a finite number of quantization centers. The output of the spatial pooler is in terms of its quantization centers [7].

- **Temporal Pooler:** Learns temporal groups - groups of quantization centers - according to the temporal proximity of occurrence of the

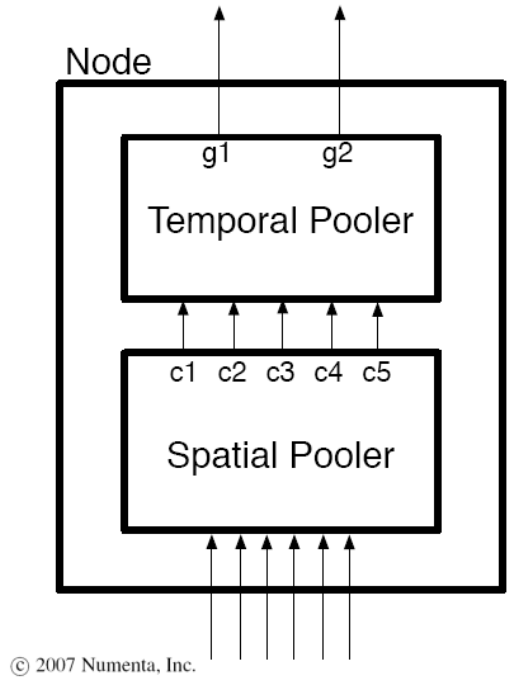

Figure 3.1: Graphical Representation Zeta1 Memory Node [7]

> quantization centers of the spatial pooler. The output of the temporal pooler is in terms of the temporal groups that it has learned [7].

To summarize, NuPIC essentially attempts to implement HTM theory in a feed-forward system. It uses temporal and special pooling algorithms at each node in the hierarchy to build an internal representation of the world. Such abilities could solve Problems 1 & 2 above. However, feed-forward architecture automatically eliminates NuPIC from solving Problem 3 as the solution provided by HTM theory requires feedback. Finally, without feedback, the NuPIC implementation will depend solely on the accuracy of the constructed model to comprehend unanticipated inputs. Until the model is constructed, NuPIC's ability to solve Problem 4 remains unknown.

Numenta's decision to implement only the feed-forward aspects of true HTM theory leaves a high representational gap between implementation and Hawkins' cortical hypotheses. To prevent confusion and to underscore Numenta's design compromise with regards to feedback and the biological foundations of HTM theory, HTM networks created using NuPIC will be referred to (in this paper) as *N-HTM networks*.

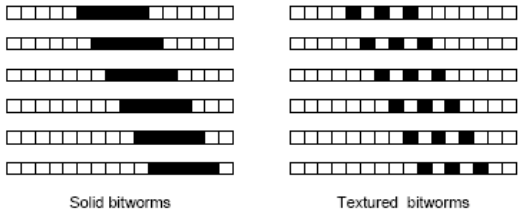

Figure 3.2: Graphical Representation of Bitworms and N-HTM Network Training [15]

3.4.3 Previous Experiments. Despite their variation on HTM theory, Numenta has shown positive performance in many of its initial NuPIC experiments. Prior to NuPIC's release, Numenta had performed several tests which now come bundled with the platform as downloaded from Numenta's website. Two notable experiments are the *Bitworm* and *Pictures* experiments.

- **Toy Experiment - "Bitworm"**

NuPIC's "hello world" experiment is known as the Bitworm example [15]. It is an entirely fictional problem that simply, yet effectively, demonstrates the power of Numenta's HTM implementation and NuPIC. "Bitworms" are 16-bit vectors (arrays) and are of two varieties: solid and textured (See Figure 3.2) [15].

As in Figure 3.2, the bitworm is briefly "scanned" by the N-HTM network sensor node over sequential "time units" collecting the appropriate spatial-temporal data. Each line represents a "modified look" at the same bitworm. During training the N-HTM network builds its own invariant representations of both solid and textured bitworms [15]. When exposed to a set (file) full of 420 different bitworms of various textured or solid patterns (i.e. 0000011111000000 or 000010101000000) the N-HTM network accurately recognizes and correctly classifies the bitworms 97.86% of the time (411 of 420 bitworms correctly identified) [15]. That is to say an N-HTM network trained on a limited number of pre-categorized "specimens" (consisting of both solid and textured baitworms) can accurately categorize a large, randomized list composed

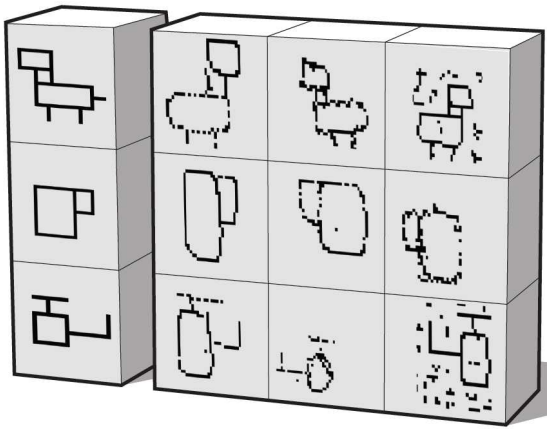

Figure 3.3: Example Line Art Input and N-HTM network Recognition Output [9]

of members of each "species." Such an N-HTM is thus able to categorize a bitworm 0000000011111110 as solid, even if it has never seen a bitworm with seven 1's so far to the end of the any solid bitworm.

- **Line Art Recognition - "Pictures"**

The second experiment, the Pictures example, is far less trivial and truly begins to demonstrate the a robustness for the NuPIC platform. Explanation and analysis of the experiment and N-HTM network is available from Numenta, but Figure 3.3 effectively conveys both problem and impact.

To summarize, NuPIC is fed dozens (three of which are seen in Figure 3.3) of basic line art graphics. Each graphic is know to programmers by names like "dog" or "mug," however NuPIC has no English word for such images. NuPIC fully scans each image during training by "moving it's eyes" (moving the picture) right, left, up, down, in, and out. It is then it is fed an assortment of line art representations of similar, but imprecise images. As seen in Figure 3.3 the network is able to easily identify an image despite noise, distortion, and/or reorientation [14].

Clearly Numenta's implementation, while not proven by these examples, certainly seems viable enough to warrant experimentation in understanding the alien environment of cyberspace.

3.4.4 Proposed Experiment. Because NuPIC has already been shown to work on small toy experiments, in addition to success on more complicated experiments, it can be tested as a solution within the proposed problem domain. Because NuPICs ability to act or react to environmental stimulus is impeded do to the missing feedback mechanisms, any cyber-oriented experiment most solely concentrate on NuPIC's categorization abilities.

- **Cyber-Experiment - Anomaly Detection**

The proposed experiment is an anomaly detection application. Given NuPICs categorization focus, it should be possible to create an N-HTM network which can categorize network traffic as normal (recognized as benign) or anomalous (potentially malicious).

Using the Python script interface to NuPIC, an N-HTM network can be created to sense any environment with custom sensors [15]. If the sensors of said N-HTM network were given packets of network traffic, the N-HTM network should (theoretically) be able to model cyberspace as presented by those packets. The first step towards this goal is to find sufficient, benign data that fits HTM training requirements and corresponding test data which contains packets of a malicious nature.

One "standard" benchmark of readily available network traffic for intrusion detection testing is found on the Massachusetts Institute of Technology's (MIT) Lincoln Laboratory website [33]. The DARPA-led collection of network traffic data (i.e."TCP dumps") for network anomaly detection was performed though the years of 1998, 1999, and 2000 [33]. Each year contains control data and test data captured during network attack exercises [33].

Although the data sets are known to have corrupt data during some periods, instances are documented sufficiently to be avoided [24]. Additionally, criticism of the sterile nature of the data is noted [24] but this is not applicable as the sterility is considered consistent in the testing data — maintaining potential spatial pattern consistency. The data may not be the best, but it's all that is available in enough documented quantity.

This anomaly detection experiment will use the 1999 MIT data sets. This data contains large TCPdump files that contain network packets captured in sequential order. The packets themselves contain many spatial characteristics including source address, ports, packet size, and even time delta (the time elapsed since the last packet captured). The sequential ordering of the packets fulfills the temporal requirements. Control data representing "normal" network activity is used to train an N-HTM network. Using the internal model generated from this training, the N-HTM network is then given the corresponding test data. The N-HTM flags any unrecognized packets as anomalous network activity. If Numenta's implementation of HTM theory is correct, the majority of anomalous packets will correspond to packets of a computer network attack.

3.5 Evaluation Framework

The evaluation framework for the Numenta HTM implementation should be evaluated based on a "reasonable expectation for success." Success will be based on how the implementation solves the problem in the experiment. Success in this experiment will lend important support for HTM theory's applicability for potential predictions of the unusual world of cyberspace.

Reasonable expectations for any anomaly detection system includes a balance between two main elements:

1. *All malicious network activity is flagged as abnormal.* That is, test data should present the N-HTM network with spatial-temporal patterns that do not match

any invariant representation previously internalized during training. No malicious packets should be perceived as having "been seen" before.

2. *Low percentage of benign network traffic flagged as abnormal* (a low rate of "false-alarms"). That is, normal network traffic within the test data should rarely be flagged as anomalous.

With respect to the experiment, success can be summarized as the N-HTM network creating new categories for malicious packets in testing data. These categories are the anomalies detected. Such malicious categories would (theoretically) not have been created from the normal (benign) training data. Consequently, malicious packets should not be categories as recognized from training. Additionally, the formation of new categories for novel, but benign, network activity should be lower than the number of malicious categories perceived (for rate of false-alarm).

3.6 Assumptions

Per the data requirements of HTM implementations, the assumption must be made that spatial-temporal patterns exist in the data representing the environment. For example, although TCPdumps contain spatial-temporal data (Section 3.4.1.3), the assumption that spatial-temporal *patterns* exist must be assumed. Until found, there is no proof that such a pattern exists in cyberspace.

Another assumption is consistency between training and testing data. Both the data used to create an HTM network model of the world and the data used to test that model should be assumed to accurately represent the same world. That is, predictions can only be made from knowledge and experience. The concept is similar to the automobile driver who is given an airplane. Assuming he has been licensed only for cars and never taught anything about flying, the driver should not be expected to know how to fly the plane (maybe only drive it around).

Finally, the idea that data manipulations preserve pattern integrity must be assumed. For experiments with NuPIC, TCPdump data must be greatly altered

before a N-HTM network can sense a packet. Specifically, filtering or parsing the packets of TCPdump data should not destroy inherent (or create artificial) spatial-temporal patterns. This assumption is verified by the company [14, 15].

3.7 Experiment Configuration: Anomaly Detection

The previous chapter introduced the NuPIC application for creating, training, and testing N-HTM networks. Previous experiments were able to successfully predict and categories textual and image data. The Anomaly Detection experiment, as required by this research, extends previous experiments in two specific ways:

1. *Make Predictions from an Unusual World* - Both the Bitworm and Pictures experiments make predictions from well understood environments - attempting to perform a task humans can do well, only more efficiently. These experiments demonstrate NuPIC's capability for solving the first set of problems HTM theory was proposed to solve (Section 2.6.2 - Inferring the Cause of Novel Input). The Anomaly Detection experiment will be the first, NuPIC-based, HTM implementation designed to solve the second set of problems [14]: Discovering Causes in Sensory Data.
2. *Perform Without Supervision* - Both the Bitworm and Pictures experiments utilize an optional, NuPIC-specific concept of a Category Node for supervised learning. These elements of the N-HTM network hierarchy allow programmers to teach each application the category to which a newly learned spatial-temporal pattern belongs (i.e. "Solid" in Bitworm or "Dog" in Pictures). This is similar to teaching a 2-year old that the red, shiny fruit they are eating is called an "Apple." However, HTM theory does not require a "Name" be provided for perceived invariant representations (e.g. not knowing an apple is called "Apple" does not prevent a 2-year old form grasping the concept its existence - a fruit with characteristics that differ from an orange or a mouse). The Anomaly Detection experiment will be one of few NuPIC experiments utilizing unsupervised

learning techniques. A lack of supervision makes intuitive sense with unknown environments (humans do not have a word for, nor would they recognize, the supposed spatial-temporal pattern that provides an invariant representation of the cyberspace environmental anomaly: "HTTP Tunnel").

3.7.1 Network Setup. Constructing the hierarchical network topology is the first step in creating an N-HTM network. Five NuPIC node types were used for constructing the anomaly detection network.

3.7.2 Sensor Nodes - VectorFileSensor. The first node of any HTM network will be the sensor node. These bottom-level nodes take appropriate environmental data and convert the input signal into the language the common language the HTM network understands. This function is analogous to eyes processing visual images and ears processing auditory information. Each environmental signal is encoded and presented to the brain (per HTM theory) as neural impulse patterns.

The sensor node used in this experiment is the VectorFileSensor which reads vectors from a data file. Except for the initial line of a data file, each line is a vector and the element of the vector must be floating point (or integer) values. Each element is separated by a single space. The first line of a data file is a single integer which instructs the VectorFileSensor how many elements to read from (or are contained in) each vector.

Example VectorFile Format (5 vectors of 7 elements):

```
7
1 23 4.5 12 8 3456 0.234
2 24 5.0 13 9 3356 0.123
3 24 5.5 14 10 3256 0.234
4 26 6.0 15 11 3156 0.345
5 27 6.5 16 12 3056 0.456
```

As an advanced option VectorFileSensors can be directed to send the value from any vector to any number of other memory nodes. In this way, having multiple sensor

nodes is only required if data is physically located in another file. It is recommended that data spread across multiple files be combined into one rather than having multiple sensors. NuPIC also provides connectivity for programmers who whish to create specialized sensor nodes for reading incompatible data. Often, pre-formatting incompatible data (if possible) is recommended. Instead of creating a new sensor node to read TCPdump files, performing data to fit the VectorFile format was the option employed in this experiment.

3.7.2.1 Memory Nodes - Zeta1Nodes. The primary nodes in NuPIC are the Zeta1Nodes which implement the Zeta1 learning algorithm. Zeta1Nodes analyze data received from lower nodes, execute the Zeta1 spatial and temporal pooler algorithms, and then send output to any number of higher nodes. All Zeta1Nodes have several different node parameters that are configurable through node instantiation fields and will affect the learning behavior.

Of the parameters which can be set, this experiment specifies 6 important fields of the Zeta1Nodes [15]:

- *maxDistance* is a required part of the spatial pooler algorithm. "The parameter sets the maximum Euclidean distance at which two input vectors are considered the same during learning."
- *topNeighbors* specifies "how many simultaneous coincidences to consider when computing groups" and is a required part of the temporal pooler algorithm. "To form temporal groups, the node walks through the transition matrix. The transition matrix shows how likely it is for each coincidence to precede another coincidence."
- *spatialPoolerAlgorithm* allows the programmer to select the method of "coincidence detection inference." The algorithm for bottom-level nodes (above the sensor) is always *Gaussian*. With this algorithm "each coincidence output is a similarity score based on the Euclidian distance between the input and that

coincidence." Nodes at other levels can be given a *product* or dot algorithm. With the product or dot method each coincidence output is the product or sum, respectively, of the child input values corresponding to that coincidence.

- *symmetricTime* takes a boolean value for which, if true, the algorithm assumes that if data arriving in a certain order is equally as likely to arrive in the reverse order.
- *transitionMemory* "specifies how many true steps of history to keep in the temporal pooler to track the time structure of coincidences."
- *temporalPoolerAlgorithm* allows the programmer to select the method of "computing output probabilities." When *maxProp*, is selected, the node "computes a more peaked score" for temporal groups. If *sumProp* is selected, the node computes a "smoother score" temporal groups.

3.7.2.2 Top Nodes - Zeta1TopNode. The highest level learning node in an N-HTM network is the Zeta1TopNode. This node functions like a Zeta1Node with the additional ability to interact with Category Nodes (used only for supervised learning and unused in this experiment as previously mentioned). The Zeta1TopNode is also connected to Effector Nodes.

3.7.2.3 Effectors Nodes - VectorFileEffector. Nodes in an HTM-based network which can provide a meaningful way of using the predictive and categorization abilities of the networks are called Effector Nodes. Motor reflexes and conscious movement represent biological effectors in humans. In NuPIC the primary Effector Node used is the VectorFileEffector. This node combines sensory input (via PassThrough Nodes) with assigned categories and writes the information to a file. NuPIC also offers the ability to create unique Effector Nodes which could potentially connect to a user interface or other N-HTM networks.

3.7.2.4 Other - PassThroughNodes. NuPIC allows for direct sensory input to effectors. Because N-HTM process input in phases according to hierarchical

levels, PassThroughNodes are available to connect input directly to output with no manipulation or learning. In this way, Effector Nodes can be told what sensory information should be added to the categories received from the N-HTM network.

3.7.3 Topology. N-HTM networks should provide a low representational gap between sensory input and network topology. To this end, the network packets provided to the NuPIC should be appropriately distributed to nodes in the network. Preprocessing of data (See Section 4.2.3.2) has yielded a 31 element vectors to represent each packet. 16 packet header and information fields were initially chosen for NuPIC analysis. Each field is provided below preceded by the vector elements from which they are composed. An ID number for packet reference was also added as the first element in each vector.

— (0)ID — (1)timeDeltaBetweenPackets — (2-6)sourceMAC — (7-11)destinationMAC — (12)type — (13-16)sourceIP — (17-20)destinationIP — (21)totalLength — (22)identification — (23)protocol — (24)flagsIP — (25)sourcePort — (26)destinationPort — (27)sequenceNum — (28)AcknowledgementNum — (29)flagsTCP — (30)windowSize —

Each field is fed to a Level 1 Zeta1Node. The output of these nodes is in turn fed to Level 2 Zeta1Nodes representing the network protocol layer from which the fields were derived. Output from Level 2 nodes is combined with the time delta information (from Level 1) at a the Zeta1TopNode of Level 3. The Zeta1TopNode is connected with a VectorFileEffector which combines the networks output categories with the ID number (first element of each vector) via PassThroughNodes present at each level of the hierarchy. The resulting network is illustrated in Figure 3.4.

3.7.4 Data. As previously mentioned, the 1999 MIT data sets that will be used contain both training and testing data. In 1999, five 5-day weeks of data was collected. Each day, well over a million network packets were captured an saved into a TCPdump file. Weeks 1 and 3 contain attack-free network traffic. Weeks 2, 4, and 5 contain both the normal network traffic of day-to-day operations along with packets corresponding to various network attacks.

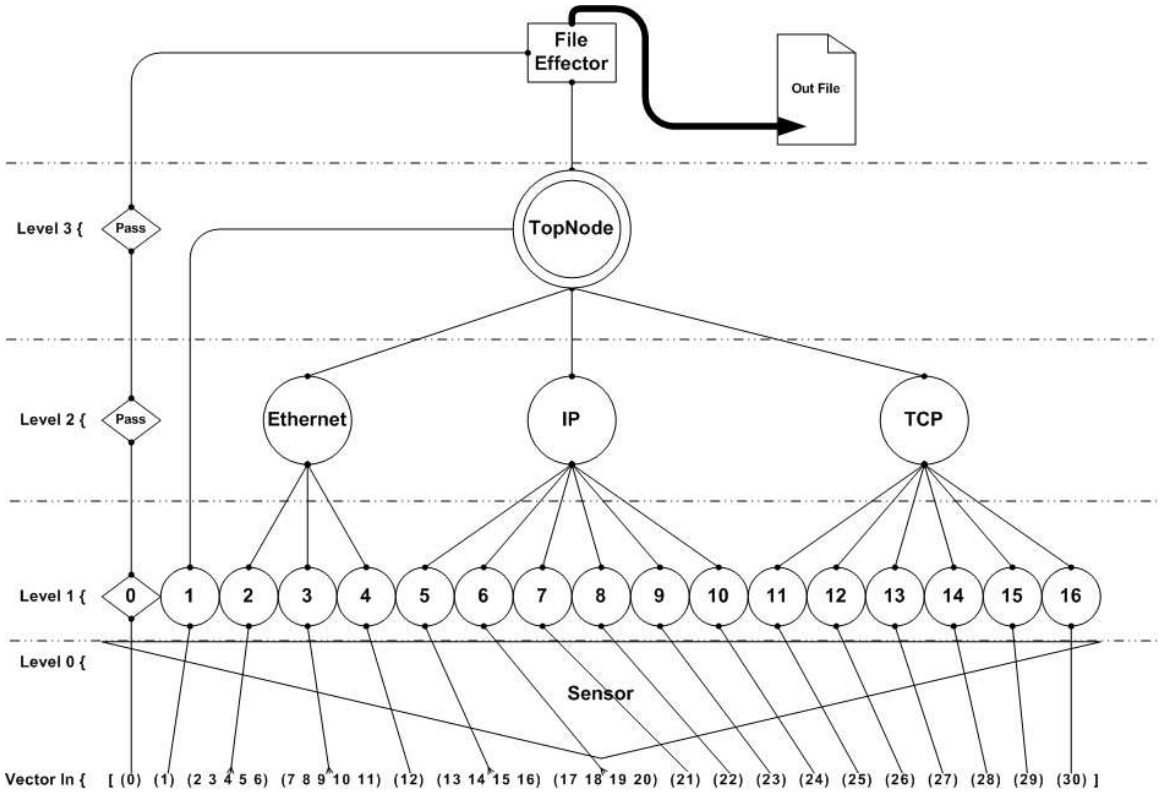

Figure 3.4: Default N-HTM Network Topology

3.7.4.1 Filtering TCPdump Files - Wireshark. All network traffic was captured in the TCPdump files, not just those packets of a particular protocols (i.e. TCP and IP). Because the fields chosen for analysis require a consistent set of layered protocols, packets using protocols disruptive to this consistency were filtered out (i.e. UDP). To do this an open source software program called *Wireshark* (See Figure 3.5) was used to open and filter the TCPdump files.

Packets without an *Ethernet, IP, TCP* protocol structure were eliminated using the filter command seen in Figure 3.6. This filtered file was then saved into *K12* format. The first 3 (filtered) packets of a Monday (Week 3) TCPdump are seen below in (abbreviated) K12 format.

```
+---------+---------------+----------+
13:00:19,888,838   ETHER
|0    |00|10|7b|38|46|32|00|c0|4f|a3|58|23|08|00|45|00|00|2c|00|74|00|00|40|06|4d|1b|c4|25|4b|9e|ac|10|71|69|04|00|00|19|8c|6c|0f|3b|...

+---------+---------------+----------+
13:00:19,893,523   ETHER
|0    |00|c0|4f|a3|58|23|00|10|7b|38|46|32|08|00|45|00|00|2c|01|1f|00|00|3f|06|4d|70|ac|10|71|69|c4|25|4b|9e|00|19|04|00|37|4d|41|c0|...

+---------+---------------+----------+
13:00:19,893,723   ETHER
|0    |00|10|7b|38|46|32|00|c0|4f|a3|58|23|08|00|45|00|00|28|00|75|40|00|40|06|0d|1e|c4|25|4b|9e|ac|10|71|69|04|00|00|19|8c|6c|0f|3c|...
```

3.7.4.2 Converting TCPdump to NuPIC Vector Form. To convert the K12 format to VectorFile format readable by the N-HTM network, a Java-based parser was written. This parser converts the hexadecimal representations into consistent integer and float representations and distributes them into a vector. The first 3 packets of the filtered Monday (Week 3) K12 file are seen below in VectorFile format:

```
31
1 0 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 44 116 6 2 1024 25 2355892027 0 0 512
2 0.004685 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 44 287 6 18 25 1024 927809984 2355892028 0 32736
3 0.0002 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 40 117 6 16 1024 25 2355892028 927809985 0 32120
...
```

3.7.5 Specifications & Parameters. Numenta concedes that there is no way to truly know how the fields of the Zeta1 nodes should be set. Settings based on educated guesses refined by trial and error is the only recommended method. As

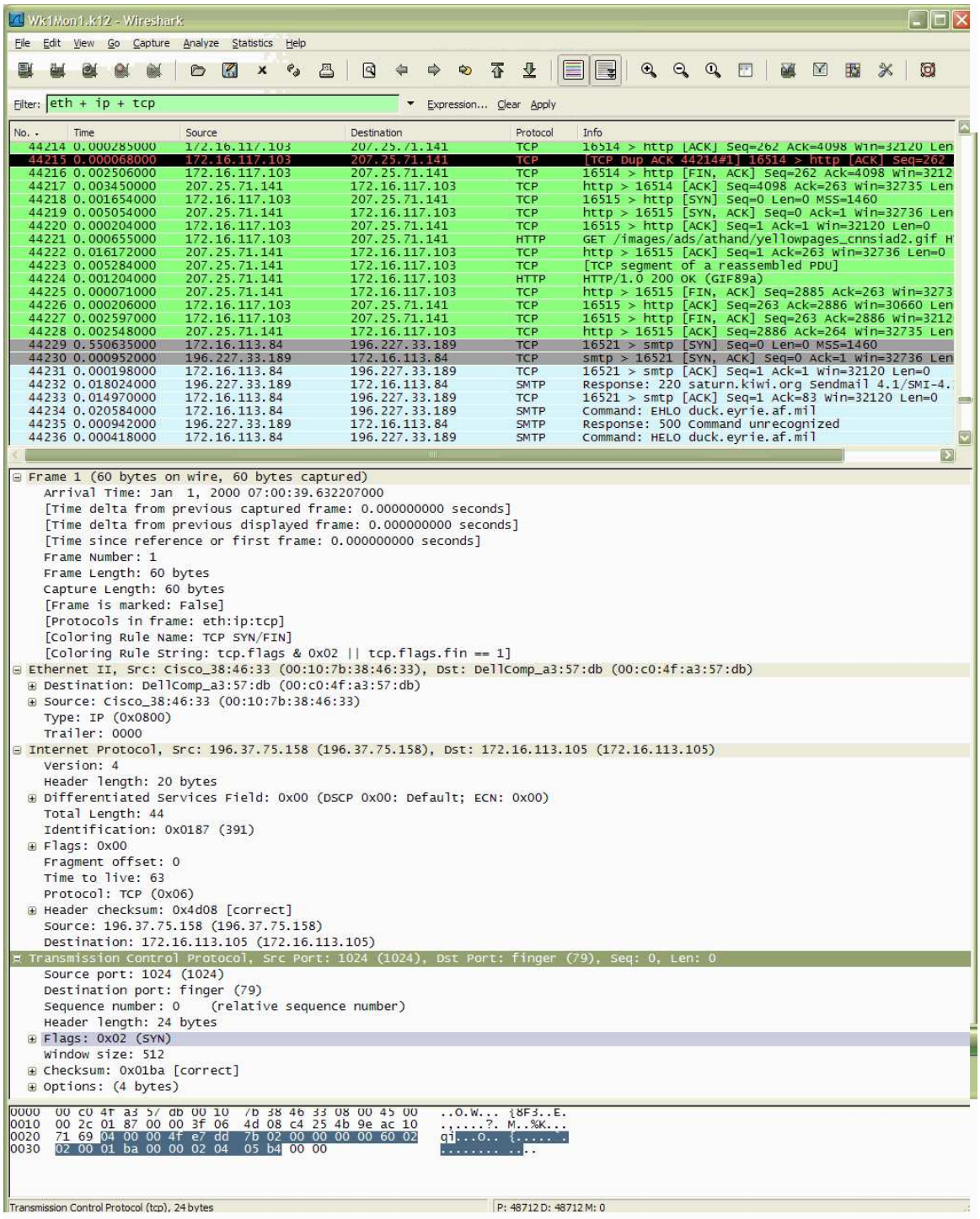

Figure 3.5: Wireshark Analyzing TCPdump File

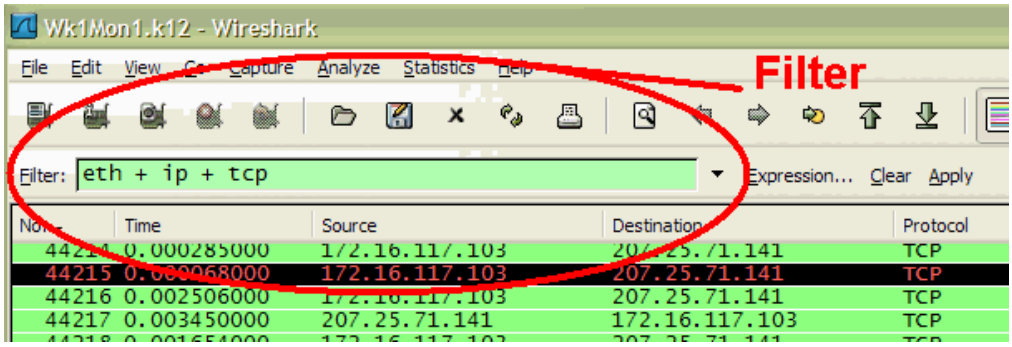

Figure 3.6: Wireshark Filter

such, the following were used as the default experiment parameters for the Zeta1 applicable nodes:

`maxDistance = 0.0`

Justification: No two packets would reasonable be perceived to be identical. Each packet should lend critical information to any model of the cyberspace environment.

`topNeighbors = 2`

Justification: This must be greater than 0. With over 1 million packets captured each day, checking for coincidences at a time should be sufficient.

`symetricTime = False`

Justification: Just because one packet is followed by another there is no reason to assume the reverse ordering is true considering all other aspects which affect the order in which packets are captured over a network.

`transitionMemory = 100`

Justification: It would seem reasonable to assume packets of any "object" in cyberspace might be up to 100 packets long.

The default algorithm settings for nodes at each level are as follows:

Level 1 (Numenta recommended):

`patialPoolerAlgorithm = "gaussian" temporalPoolerAlgorithm = "sumProp"`

Level 2:

`spatialPoolerAlgorithm = "gaussian" temporalPoolerAlgorithm = "sumProp"`

Level 3 (Numenta recommended):

`spatialPoolerAlgorithm = "dot" temporalPoolerAlgorithm = "sumProp"`

3.7.6 Procedures. The procedures taken for the experiment were quite straightforward:

1. *Training*: A VectorFile formatted training data file must be selected from attack-free data sets. Training is unsupervised and the objective is to build a world model for the "normal" network environment of cyberspace. Because training the N-HTM with all, 1 million plus, packets would take weeks, the number of vectors will be limited initially to facilitate the trial and error methodology required for parameter tuning. Additionally, because training is the most time consuming aspect of N-HTM creation, limiting the number of packets results in more time alluded for testing. Based on training vectors used with other NuPIC experiments, several thousand vectors will be required for appropriate, initial training.
2. *Testing*: Once a N-HTM network is trained, it will be used to test an entire day of packets from a week where attacks were conducted (e.g. Week 4) on the network. In this way several days worth of attacks can be observed and, hopefully, categorized in a category not created during training.
3. *Analysis*: The categories assigned to packets in the testing data will be checked for uniqueness against old categories derived from training. New categories assigned to malicious packets will be considered successful. New categories assigned to benign packets or old categories assigned to malicious packets should be considered a failure of the anomaly detection system.

3.7.7 Goals & Expectations. Although a superior anomaly detection method is not expected to emerge from this experiment, a generalized indication that the N-HTM network is capable of categorizing malicious network packets as different from benign packets is anticipated. Although false-alarms will probably be higher in this experiment than with other anomaly detections system, limited success might point towards future success of HTM networks or, more precisely, Numenta's interpretation and implementation of HTM theory.

IV. Results

This chapter reports the results of the Anomaly Detection experiments. To begin, a summarized account of the experiment's administration will precede the corresponding results. In the course of conducting said experiments, unanticipated adjustments are sometimes made to the proposed experimental specifications. Such alterations, as applicable, are documented here along with any pertinent consequences of their implementation. Finally each set of developed results is explained and analyzed.

4.1 Running Anomaly Detection Experiment

For this experiment, several extensive and different variations were run to ensure adequet conclusions could be drawn. In addition to several trial experiments, 32, large-scale (similar to the ones presented in this paper) experiments, as specified in Chapter 4, were performed with NuPIC. Performing 32 trials was not premeditated, instead it's just happened to be the case that after 32 trials of consistent results, analysis could be resolved and conclusions could be drawn.

In accordance with Numenta's recommended "trial and error" method, modifications were constantly made to input data, parameters, and even network topology. The training data (days of packet captures from Weeks 1 and 3) was constantly varied as were the testing data sets (Weeks 2 and 4). Modifications or the initial parameters including fields affecting spatial and temporal poolers, were attempted. In some experiments, network topology was even modified to incorporate a 4th level (See Figure 4.1). Training in a "supervised" manner was attempted at one point (per the NuPIC guide) by adding a line of 31 zeros to separate packets into artificial, temporal frames. Even certain input vectors elements were, at times, ignored to increase variation between packets.

The details of experiments recounted in this section are those implementations which showed the best results.

4.1.1 Training. Numenta insists that the data used for training is critical to success. For this reason, data from both Weeks 1 and 3 were used for training the N-HTM network. Each day is capable of offering TCPdump files in two different ways. Some days provide TCPdump files representing network activity from "inside" the network while others captured packets originating from "outside" the network. Some days may only provide one or the other. Although both types of TCPdump files were used, for anomaly detection the assumption was made that network traffic originating from outside a network would present more potential abnormalities.

Additionally, the first day (Monday) of Week 3 was favored because it seemed to have captured one of the more "average sized days" outside the network. That is, the size of the outside TCPdump file for this day was not the largest or the smallest. The assumption was made that moderate network traffic loads would be most typical.

Perhaps the second most important aspect of training is how much is provided. Of the 32 experiments performed, N-HTM networks were successfully trained using 100, 300, 1,000, 3,000, 10,000, and 60,000 packets from varying days. Training with 120,000 packets was even attempted but failed to produce a N-HTM due to memory constraints.

All results presented in this paper are derived from an N-HTM network trained with the first 60,000 packets of Monday's outside (Week 3) TCPdump file. This network will be referred to as network "MW3-60K."

4.1.2 Testing. Testing was performed by running many packets through the trained N-HTM network for categorization. Weeks 2 and 4 provided 5 days each of TCPdump containing attacks. Both outside and inside variations were uses through training. Testing was performed on each day of Weeks 2 and 4. The number of packets used was often determined by were certain attacks were located within a day's TCPdump file. In an attempt to preserve any spatial-temporal patterns located for an attack, no fewer than 300 packets were provided to the N-HTM at any one time. More often, an entire day's collection of packets was given to the N-HTM for

categorization. The largest successful categorization was performed on all 1,544,574 packets of Thursday from Week 4 with the MW3-60K network.

4.1.3 Data Collection. All training results and testing results were output by the network's VectorFileEffector to files called "training_results.txt" and "test_results.txt" respectfully. Results were in a two-element vector where the first element is the assigned category and the second vector is the packet's ID number. See the three following example outputs (important for later examples):

```
1234.5  1
2345.6  2
1357.9  3
```

A Java "visualization" program was created to aid in the analysis of such files and was customized to each experiment. Customization was required, among other reasons, to reformat scientific notation (used by NuPIC for integers over 1 million) into integer form. Results formatted in this way were saved in "test_results.format.txt" files (training files had do need for this formatting as they were always composed of far fewer than 1 million packets).

This Java program also produced "training_results.report.txt" and "test_results.format.report.txt" files which contained a corresponding list of the categories formed from each phase. Additionally, this program calculated the number of initial categories created by the N-HTM network during training and the number of new categories discovered after testing.

To aid in quick, visual inspection of test result files, the Java program also produced a filtered file called "test_results.format.out.txt." In this file, packets were "zeroed out" if they were assigned a category previously derived from training. These zeros help with analysis and the distinction is important for interpreting results. To aid in future understanding in this thesis, the following example shows the "out.txt"

file corresponding to the previous example of three testing outputs. Here it is shown that categories 1234.5 and 1357.9 where identified from training:

```
0
2345.6  2
0
```

4.1.4 Initial Results. The reason all 32 experiments are not discussed in detail is because no single variation provided a significant advantage over the others. Two specific variations did, however, prove far worse than the majority.

The inclusion of a time delta as a spatial element in the packet was identified as a problem early in initial experiments. The time delta proved problematic as abnormally large time deltas seemed to unduly influenced the N-HTM. These large time deltas were attributed to the filtering out of all packets but those that containing Ethernet, IP, and TCP protocol layers. Although this filtering is critical for automated parsing of a million (plus) packets, an unanticipated side affect occurred. For example, if 500 UDP packets arrived between two TCP packets, the time delta would appear to be very large between the first and second TCP packets. Time delta thus could become a far too powerful spatial element when perceived and weighted by the N-HTM during categorization. Just a few packets filtered out from between two included packets would produce a immensely abnormal time delta resulting in an equally and radically different category.

The solution was to remove the connections between the first memory node and the sensor for the time delta element of each vector. This results in a packet vector type called "timeless" in this paper and the Python code. This is not to mean the temporal aspects were affected in anyway as the time delta is only an element of the spatial pattern.

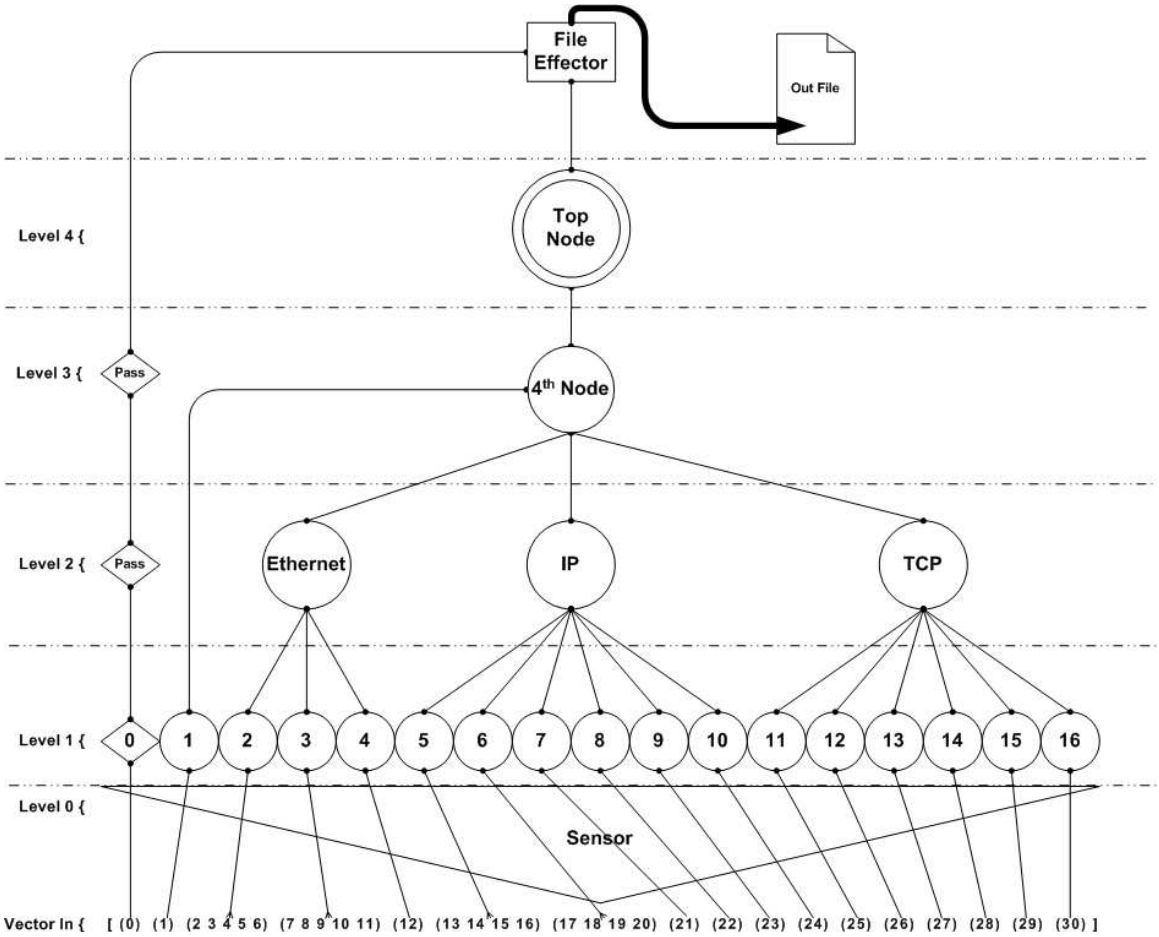

Figure 4.1: 4-Level N-HTM Network Topology

The second variation was to allow for the dynamic allocation of a 4th level to the hierarchy (See Figure 4.1). A Zeta1Node was placed between the three protocol nodes and the top node. The result was far fewer categories. In fact, no (or very few) new categories were able to be created during testing. These poor results limited useful testing of the 4-level N-HTM (in this configuration).

The most comprehensive experiments were those done with a 3-Level, "timeless" MW3-60K network (Lv3T MW3-60K) used to test every packet in each day (Monday-Friday) of the Week 4. Although testing results were obtained from each day of Week 4, only the accuracy of anomaly detection for Monday and Tuesday is presented in this paper.

The categorization statistics (provided by the Java visualization program) for each day of the week are provided in Appendix B. A glance at these statistics indicate what was observed: N-HTM categorization was consistent across each day. So too, the ability of the Lv3T MW3-60K network to detect anomalous activity was the same for each day of the week. Monday and Tuesday results are presented below because they contain a representative cross-section of attacks performed throughout Week 4.

4.1.5 Analysis. The attacks that were performed over Week 4 fall into 5 classes [33]:

1. *Denial of Service (DOS)*: "An attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine" [33] These attacks include "processtable," "mailbomb," and "land."
2. *User to Root Attacks*: "Exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system" [33] These attacks include "ps," "loadmodule." "sqlattack," and "sechole."
3. *Remote to Local Attack*: "Occurs when an attacker who has the ability to send packets to a machine over a network but who does not have an account on that machine exploits some vulnerability to gain local access as a user of that machine" [33]. These attacks include "sshtrojan," "xsnoop," "snmpget," "guest-telnet," "ftpwrite," "httptunnel," and "phf."
4. *Probes*: "Automatically scan a network...to gather information or find known vulnerabilities" [33] The attack included here is "portsweep."
5. *Data*: These attacks "involve someone (user or administrator) performing some action that they may be able to do on a given computer system, but that they are not allowed to do according to site policy. Often, these attacks will involve

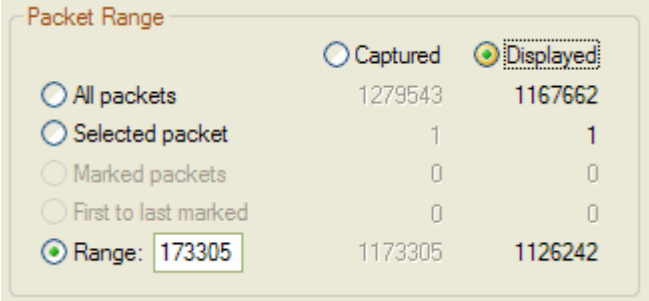

Figure 4.2: Wireshark Filter Packet Calculator

transferring 'secret' data files to or from sources where they don't belong" [33]. The attack given here is denoted "secret."

Of the attacks performed over Monday and Tuesday of Week 4, at least one attack (included above) from each of the 5 attack-classes was present. Each attack's start time is indicated in the MIT documentation. For the purposes of evaluation, the packet at this time is considered the target or start of anomalous behavior. Ideally, anomalous behavior would end when the attack resolves. In accordance with a reasonable expectation of success, this experiment makes distinctions between new categories (anomalies) assigned to the start packet and 10 packets above or below that start packet. This is done to allow for potential variance in spatial-temporal patterns learned from this unusual world. I is reasonable that the N-HTM might predict anomalies (thus notifying users) before they have started or take some time to realize they have begun. Either circumstance is conceivable from HTM theory.

It is important to note that the protocol filtering performed by Wireshark disrupts packet ID numbers. Because time is only visible before conversion to K12 format, Wireshark must be used to analyze and find start packets (Pkt) and infer the filtered start packet (Filtered Pkt). That is, if an attack starts with Pkt=400 but 50 packets have been filtered out prior to this ID number, the Filter Pkt=350 will be the VectorFile ID number. As seen in Figure 4.2, Wireshark provides a calculator for finding the correct Filter Pkt given a start Pkt.

Year: 1999	Week: #4	Location: Outside		Day: Monday
NuPIC Anomaly Detection Results				Key A=Above, B=Below
Attack Type	**Start Time**	**Packet (Pkt)**	**Filtered Pkt**	**Anomaly Detected**	**Near-By Anomalies**
sshtrojan :	9:36:23	22587	19510	Yes	No
xsnoop :	11:22:43	215792	204698	Yes	1A,1B
snmpget :	11:45:12	241125	228418	No	5A
snmpget :	12:10:48	273100	258520	No	No
guesstelnet :	11:47:16	243879	231032	No	2A,2B
portsweep :	12:22:15	288256	273001	No	2A
portsweep :	12:23:55	291518	276192	No	2A
portsweep :	12:25:35	294267	278878	No	No
portsweep :	12:27:15	297285	281820	No	2A
portsweep :	16:27:10	779081	749098	No	No
portsweep :	16:27:16	779156	749170	No	1A
portsweep :	16:28:22	782312	752208	No	5A,7B
portsweep :	16:28:28	782376	752269	No	1A,2B
portsweep :	16:29:34	786582	756411	No	No
portsweep :	16:29:40	786831	756657	No	No
portsweep :	16:30:46	789132	758895	Yes	3A,5B
portsweep :	16:30:52	789218	758951	No	3A
portsweep :	16:31:59	791148	760847	No	No
portsweep :	16:32:05	791223	760918	Yes	1B
ftpwrite :	13:58:16	476579	454471	No	No
ftpwrite :	13:58:20	476772	454663	No	No
ftpwrite :	14:03:29	487405	465049	No	No
secret :	18:24:19	1034566	998367	No	No
secret :	18:27:45	1042280	1005959	No	1A,1B

Figure 4.3: Anomaly Detection Results - Week 4, Monday

When the attacks (and corresponding start times) are used to find the respective Pkt and Filtered Pkt, anomalies can be determined from the N-HTM network output. In this way, the tables in Figures 4.3 and 4.4 are constructed. If a test_result.format.out.txt file contains a Filter Pkt (equal to the ID number of the packet vector and line number of the out.txt file) assigned to a new category, an anomaly is said to have been detected.

Year: 1999	Week: #4	Location: Outside		Day: Tuesday	
NuPIC Anomaly Detection Results					Key A=Above, B=Below
Attack Type	Start Time	Packet (Pkt)	Filtered Pkt	Anomaly Detected	Near-By Anomalies
httptunnel :	9:09:17	3028	2124	No	2A
httptunnel :	9:10:44	3791	2763	No	No
phf :	9:41:12	13923	10232	No	No
loadmodule :	10:41:12	102272	93900	Yes	5A,10B
ps :	11:29:17	189552	178267	Yes	1B
ps :	11:30:50	191442	180068	No	No
ps :	11:47:10	213687	201343	No	No
secret :	12:22:13	289181	274898	No	No
secret :	12:48:08	363588	347972	No	No
secret :	12:53:30	375281	359393	No	1A
sqlattack :	13:54:17	525772	506596	Yes	10B
sechole :	14:32:15	627340	606141	No	No
sechole :	14:32:19	627397	606198	No	No
sechole :	14:32:23	627412	606203	No	No
land :	14:54:10	708238	685936	Yes	10A,7B
mailbomb :	15:51:16	858100	833251	No	2B
mailbomb :	15:51:18	858265	833415	No	2A,2B
mailbomb :	15:51:21	858641	833791	Yes	8A,5B
mailbomb :	15:51:25	858906	834053	No	7A,1B
mailbomb :	...	...	...	...	...
processtable :	17:49:15	1098237	1066624	No	No

Figure 4.4: Anomaly Detection Results - Week 4, Tuesday

Filter Pkt 19510 of Monday shows an anticipated outcome for anomalies. What is not anticipated is that the corresponding category 2827.6 for this attack appears 60 other times in this 1,167,661 packet file. However, the attack for which it is designated (sshtrojan) occurs far less!

The next attack starts at Filter Pkt 204698. While new category 2923.4 is also given to this attack (xsnoop), the data below shows two other anomalies (in this case the same category) within 10 packets above and below. This category is also common in the categorization of packets this day in spite of the xsnoop attack appearing only once.

The expectation would be to find a block of anomalies starting with that first Filter Pkt. An example of this expectation can be seen in the below data from the sqlattack denoted in Tuesday's table by Filter Pkt 506596.

Again, it is observed that thousands of these anomaly blocks exist in the data, even when there is nothing but normal network activity occurring. From the category statistics in Appendix B and the innumerable false-alarms indicated in visual review of the data, it would seem as if anomalies were *nominal*. Indeed, the majority of attacks indicated in both tables go unnoticed by the N-HTM while anomalies or consistently detected elsewhere within the benign packets of the results.

The overall performance statistics also show the common trends. From Monday's result table, a strict interpretation of the anomaly detection column shows that only 4 of 24 anomalies (16%) would have been detected. Expanding this window of detection to the 10-packet "grace period" for detection (both above and below) would yield the notion that 14 anomalies (58% of those available for detection) were, indeed, predicted.

While this number appears promising, the false alarm rate tells the rest of the story. On Monday, 181,849 packets were predicted to be anomalous. Even if all anomalous and near-by anomalous packets (51 packets as indicated in the results) were considered accurate, this still means that 181,798 packets are false alarms. Although these false positives make up less than 16% of all 1,167,661 packets tested, the seemingly randomized distribution of false anomalies shows little indication that they are connected with any of the malicious events (i.e. there is no "clumping" of anomalies during attacks as anticipated). This falls alarm Similar results were found the rest of the week including Tuesday which had 302,470 anomalies detected with only a handful to potentially indicating malicious activity.

Further, in every N-HTM test, the number of new (anomalous) categories created was nearly equal (See Appendix B) to the number learned during testing. If learning had occurred then these new categories would be clustered around attacks as packets never before seen. The randomized nature of new category appearance in the datasets (often in sections of network traffic that are considered "normal") indicates that no true learning had occurred.

Comparing these results to other machine learning anomaly detection systems is not favorable. Matthew Mahoney has constructed several experiments with the MIT datasets from 1999 [23]. One of his more published packet header anomaly detection systems, aptly named "PHAD", produced detection rates far superior to the N-HTM while accurately locating their arrival (i.e. not grace period) [23]. In a telling example, PHAD 87% of portsweep attacks (13/15) while the N-HTM detects only 14% (2/14) accurately or 64% (8/14) if the arbitrary grace period window is used [23]. PHAD also has a superior false alarm rate of "10 per day" [23] compared to a staggering 181,798 for the N-HTM on Monday alone. In the end, although PHAD was trained on an entire week's work of normal network traffic [23], the system still detects more types of network attacks, more accurately and more often than the N-HTM.

V. Re-Examining Feedback and Prediction

Cautious of Numenta's implementation, which omits HTM theory's essential feedback mechanisms, a second implementation is proposed to more adequately test this aspect of the theory and its affect on prediction. The goal of this project is to design an HTM algorithm that closely resembles Hawkins cortical theory as interpreted by this research. An implementation of that algorithm will be used to solve a "toy problem", similar to Numenta's "Bitworm." This thesis will refer to this algorithm and its implementation as *BackTalk*.

5.1 Mapping Theory to Algorithm

To implement a software solution for each of the 4 Problems proposed in Chapter 3, mapping HTM theory to an algorithm is required [21]. In keeping with Hawkins theories, the BackTalk algorithm will be designed around a Bayesian hierarchical framework.

From a structural view, BackTalk's HTM network topology is similar to NuPIC's implementation. However, BackTalk must address three major differences between an N-HTM network and a true HTM network:

1. Instead of passing up temporal groups of quantization centers like an N-HTM network, a true HTM network must only forward invariant representations of a perceived spatial-temporal pattern.

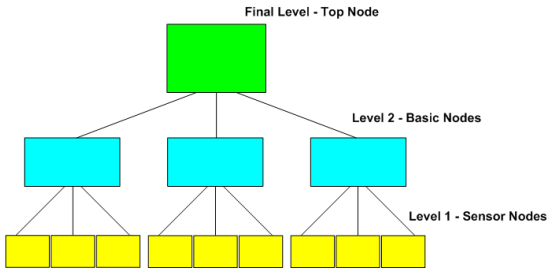

Figure 5.1: Proposed HTM Network Hierarchy

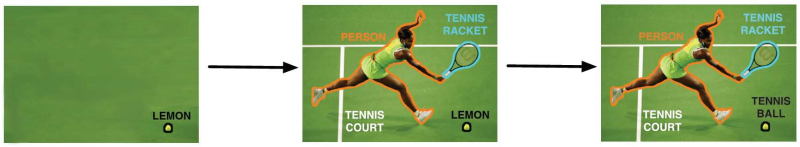

Figure 5.2: Categorization via Objects in Context [29]

2. Feedback mechanisms are absent in an N-HTM network. In accordance with cortical theory, a true HTM network must percolate predictions back down the hierarchy as feedback.

3. In N-HTM networks, memory is static after training. HTM theory requires any implementation to dynamically learn novel inputs during and after training.

Additionally, for any HTM algorithm to be accurate to HTM theory and functional for experimentation, the 4 problems presented in this paper must be addressed.

To solve Problem 1 the algorithm must facilitate self-generation of invariant representations of a world environment via supervised or unsupervised training. In order for an algorithm to do this it must be shown as set of objects and "told" what was just observed. This is known as *supervised learning* and it is exactly the way NuPIC learns the difference between bitworms or pictures. *Unsupervised learning* means learning is done without guidance towards proper categorization. However, to perform unsupervised learning, *something* must be used to determine a category shift. Otherwise everything learned would be put into one category.

This is where the concept of *context* becomes central to the design of the BackTalk algorithm. Semantic context (or simply "context") is the expected probability of an object's relation to "nearby" objects. Categorizing objects within a context has proven important in recent applications [29]. This idea of context is illustrated in Figure 5.2.

Assume an avid tennis player observes the first illustration in Figure 5.2 without seeing (or being allowed to see) the rest of the picture. You can imagine that the initial perception that the yellow dot could be a lemon. However, the following image reveals

more information and thus a lemon on a tennis court does not *make sense* given the context of the nearby tennis racquet and tennis player. When objects are perceived to be "out of context," a more "fitting" explanation of the round object can be offered. In this case, acknowledging that a lemon is out of context with a tennis court leads to the assumption that the lemon is, indeed, really a tennis ball [29].

The proposed BackTalk algorithm will use the context of spatial-temporal patterns to provide a guide for unsupervised learning. For BackTalk, initial context will be provided in a supervised manner. In this way, if BackTalk predicts a known spatial-temporal pattern but perceives a novel pattern the current input is said to be "out of context" and learning of this new, novel pattern can begin. In the same way, an analogy can be drawn between the way humans focus attention from a "dog" to "fur" or even "animal" and the way a BackTalk application uses context to govern perception. This change in focus or perception could be used to facilitate unsupervised learning.

Context also plays a critical role in BackTalk algorithms by allowing for adaptation to change. Once a change in context is detected, BackTalk could not only learn new and unanticipated patterns, it can act and react according to the patterns being learned (or previously learned). This application of objects in context is similar to a human looking away from the dog in the front yard to the car on the street. A change in context has triggered either the learning of a new concept of "street with cars" or, if previously learned, the human can predict the car's movement down the street.

Applying the concept of an object's context could allow a BackTalk algorithm to solve Problems 2, 3 and 4. Detecting context change requires prediction from higher levels of an HTM network hierarchy. This illustrates the significance of feedback within the a proposed HTM network. It would seem the lack of feedback in N-HTM networks inhibits the concept of "context." Accordingly, N-HTM networks are unable to perform true, unsupervised learning, nor can such feed forward systems hope to adapt to changing environments.

5.2 Algorithm

The following is a generalized algorithm of BackTalk functionality as laid out in the initial overview:

STEP 1 - PREDICT
(Top Node):
Initial Case:
No predictions are made by Top Node

Normal:
Case 1 - No sequence can be determined from perceived sequence:
Predict to children last perceived code as a an IR to children.
Case 2 - Currently playing (or other saved) sequence is determined to be correct:
Predict the next code in current sequence as an IR to children.

Note: When currently playing sequence has finished (no prediction can be made) use perceived sequence to make next prediction. If no prediction is found, result is Case 1.

(Other Node):
Initial Case:
No Operation performed (n/a per Top Node).
Normal:
Whenever a parent provides an IR, determine matching code in self then pass that code as an IR to children.
STEP 2 - SENSE
(Sensor Nodes):
Feed Forward:
Convert ASCII character to integer (code)
Check for novelty:
If - Code is new:
Add code to tracking list.
Set novel input flag to true.
Else - Code already exists:
Set novel input flag to false.
Check for context change:
If - Code is not predicted:
Set context changed flag to true.
Else - Code is predicted:
Set context changed flag to false.
Pass code and flags to parent.

Feedback:

If sensors are set to act as effectors, perform task.

Note: HTM theory proposes that feedback facilitates learning through prediction but ALSO acts as commands for responses. In the upcoming experiment, the sensor, which is given the feedback based on the expectations, tells a potential output mechanism to act on the prediction in the hopes that the prediction is correct.

Step 3 - Perceive Input

(Top & Basic Nodes):

When all children have reported:

Convert child integer values to spatial input pattern.

Check for novelty:

If - Pattern is new:

Add to tracking list and create new code to forward.

Set novel input flag to true.

Else - Code already exists:

Use existing code to potentially forward.

Set novel input flag to false.

Check for context change:

Note: Context change in this algorithm can be determined by any number of other algorithms (i.e. probabilities, majority rule, or potentially NuPIC's Spatial & Temporal Pooling). For the Urban Challenge toy problem, majority rule will be sufficient for testing and so this algorithm is written accordingly.

If - context changed (e.g. majority of children perceive that the context has changed):

Set context changed flag to true.

If - new code & can make a best guess:

Note: If the context has changed and the perceived input from some children is new, there is a chance that a guess can be made as to the true novelty of the input at this level. That is, sometimes a minority of children see undesired "noise" in an otherwise understandable pattern. If the majority of children perceive a recognized, but out of context, pattern then the chance exists that the minority who perceived new input are seeing noise. This can be thought of as a human viewing a car obscured by a fire hydrant. Most of the mind recognizes the perception of seeing a car and the fire hydrant is ignored. Again, many algorithms can be used to determine noise, here majority rule is again used. Further, guesses

are made by picking the first code from known patterns where the values determined to be noise are "wild card" or don't-care-states.

Iff - guess pattern found:
Return guess as code to forward.
Set novel flag to false.
Else - predicted code (via parent IR) is perceived:
Set context changed flag to false.
Use expected code as code to forward.
Pass code and flags to parent if not a Top Node.
If Top Node, go to Step 5.

STEP 4 - REPEAT STEP 3 *if not a Top Node*

STEP 5 - MODIFY SEQUENCES

(Top Node)

Add code to perceived sequence.
Trim - sequence to max length (i.e. if max = 4 then sequence has is four codes long).
If novel input:
Call feedback to save parent codes as IR in all childrenrecurse to sensor nodes.
If context is not changed:
Go to - Step 1.
Check for sequence matches:
If - recording a new sequence:
Check for match in this recording.
If - match found stop recording, add recording to saved sequences, set currently playing sequence to this sequence.
Check for matches in all saved sequences.
Else - not recording a new sequence:
Check for match in saved sequences.
If - match found set currently playing sequence to this sequence.
Else - start recording perceived sequence.
Go to Step 1.

5.2.1 Key Concepts Illustrated. Without going down every possible path the BackTalk algorithm could take, there still are some basic concepts that can be better explained with examples. Important ideas like predictive feedback, saving feedback, context change, novel input, and recording are expounded upon in this section. The complete BackTalk source code, implementing the Urban Challenge experiment (Section 4.3), can be found in Appendix C.

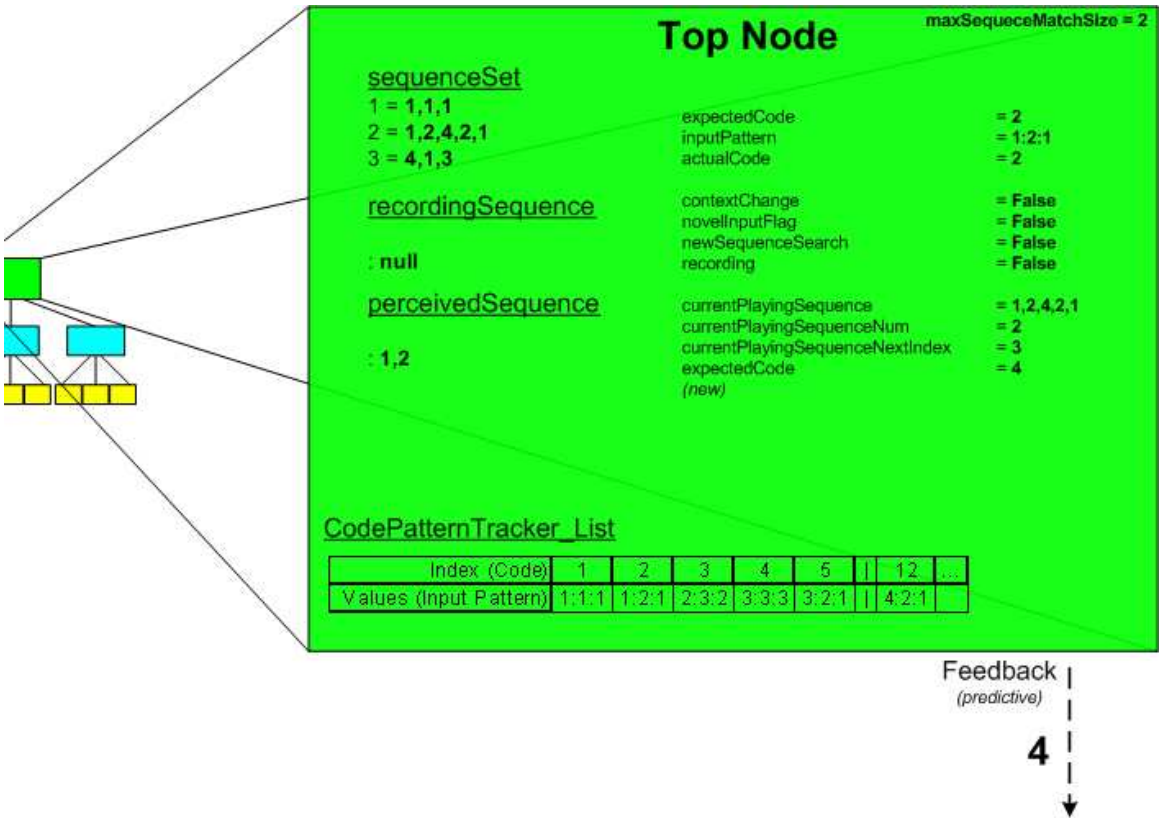

Figure 5.3: Illustration: Normal Top Node Operation - Predictive Feedback

First, it is important to make a distinction. Numbers in the algorithm (see Figures) do not represent their numerical values. Instead, they are a simple (and easily incremented) placeholder for the concept of the invariant representation (IR). IR:4 could just as easily be replaced with a letter like 'Q' or even a word like "boat," "keel", or "rivet." Secondly, this programmer originally used the term "Quale" (or its plural term "Qualia") synonymously with IR in programming variables. Diagrams reflect variable names used in code but do not reflect any proven connection between qualia and the concept of IR.

The first step in the algorithm is prediction feedback from the Top Node. In Figure 5.3, the Top Node has perceived a sequence that is within the context of its currently playing sequence. The Top Node predicts the next invariant representation it expects to perceive (IR:4) and provides it as feedback to child nodes.

Because only invariant representations are communicated in HTM theory, all nodes that receive feedback must learn to understand parent invariant representations and discover node-specific IR evoked by parent invariant representations. Say the Top Node predicts IR:7 (an IR for "Dog") to its children. A child node of the that Top Node must have some way of understanding IR:7. The child node has no understanding of higher-level complex concepts like "Dog," only component concepts like "Head," "Foot," or "Tail." It may mean that when the parent, Top Node predicts "Dog," this child would know to expect to see a "Tail" (or IR:2). In this way the Basic Node in Figure 5.4 knows that a prediction of IR:7 means it can expect to see an IR:2. In this example, a parent IR code of IR:2 will also trigger child IR code IR:2, but these codes do not mean the same thing as they are at different levels. Similarly, an IR:2 code in a sibling of this child will not necessarily mean the same thing as it does to this child and their parent's IR:7 code may trigger IR:53 in the sibling.. This (each) child's IR code (in this child IR:2) is also then fed back down to its child nodes as a parent until each node in the network has an expectation of what should be seen next.

In Figure 5.4, the Basic Node has received sensory inputs from its children post-feedback. The majority of children concur with the prediction and agree that the expectations have been met and the pattern perceived is in context. In this way, the Basic Node forwards the confirmation to its parent node.

When a Top Node's predictions are not met, the context change can force a search for a new sequence from which to make predictions. As illustrated in Figure 5.5, the expectation was to perceive a sequence 1:2 but instead a sequence of 1:1 was perceived. Because this new perceived sequence matches a previously saved sequence, a prediction of IR:1 can be made.

Sometimes a context change at the Top Node can force a search for a known sequence, but no sequence is matched. In Figure 5.6, such a search has just failed and the result is the initiation of a new sequence to record. By default, this implementation

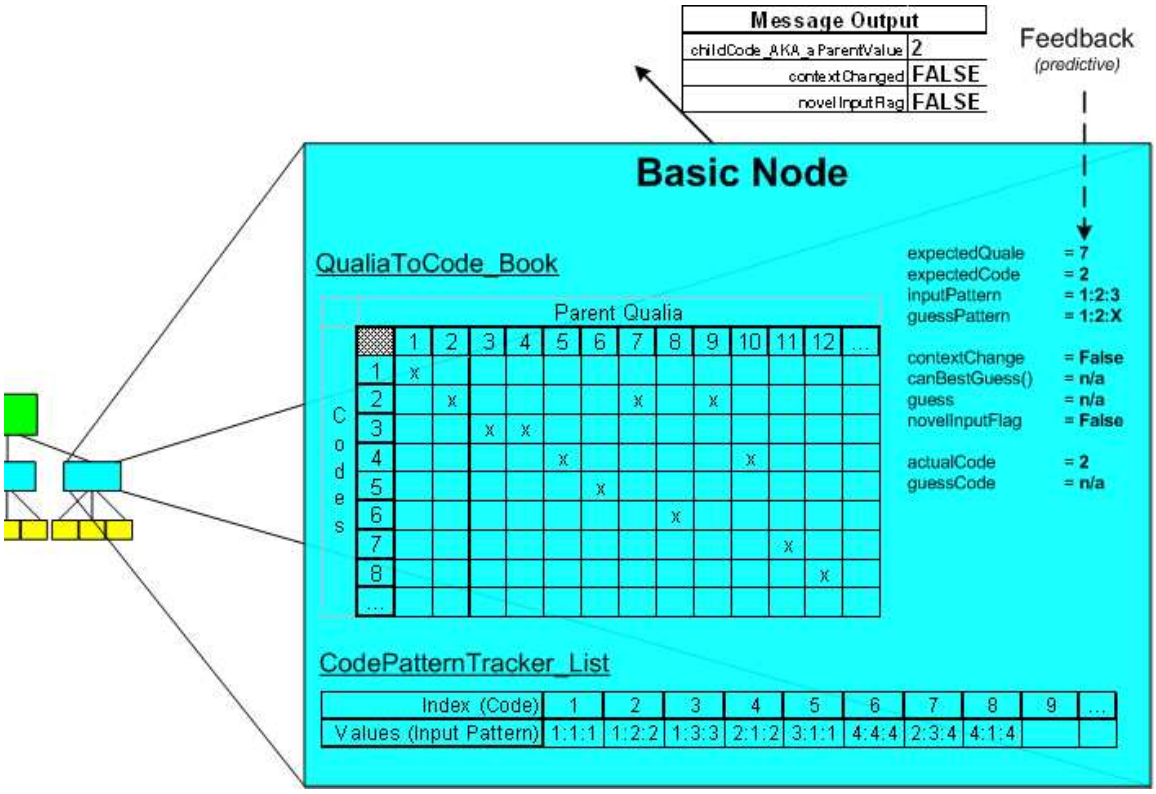

Figure 5.4: Illustration: Normal Basic Node Operation - No Context Change

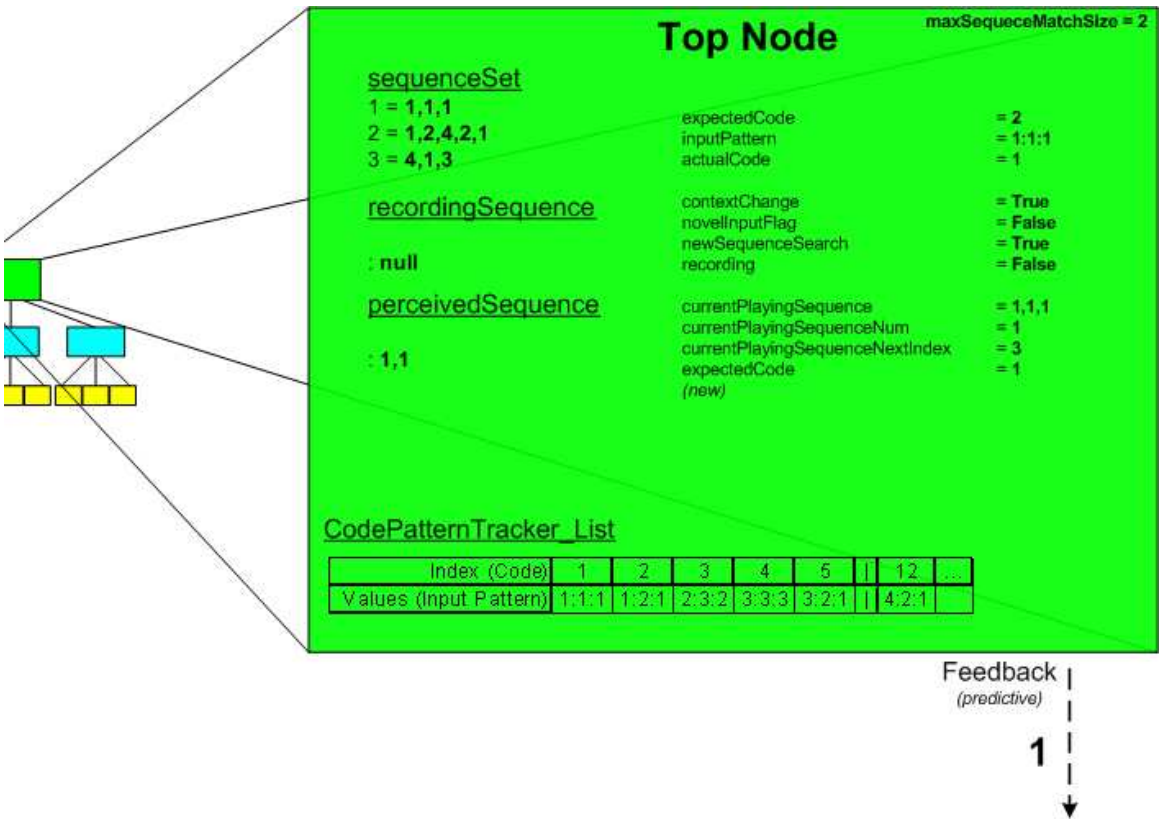

Figure 5.5: Illustration: Basic Node Operations - Context Change

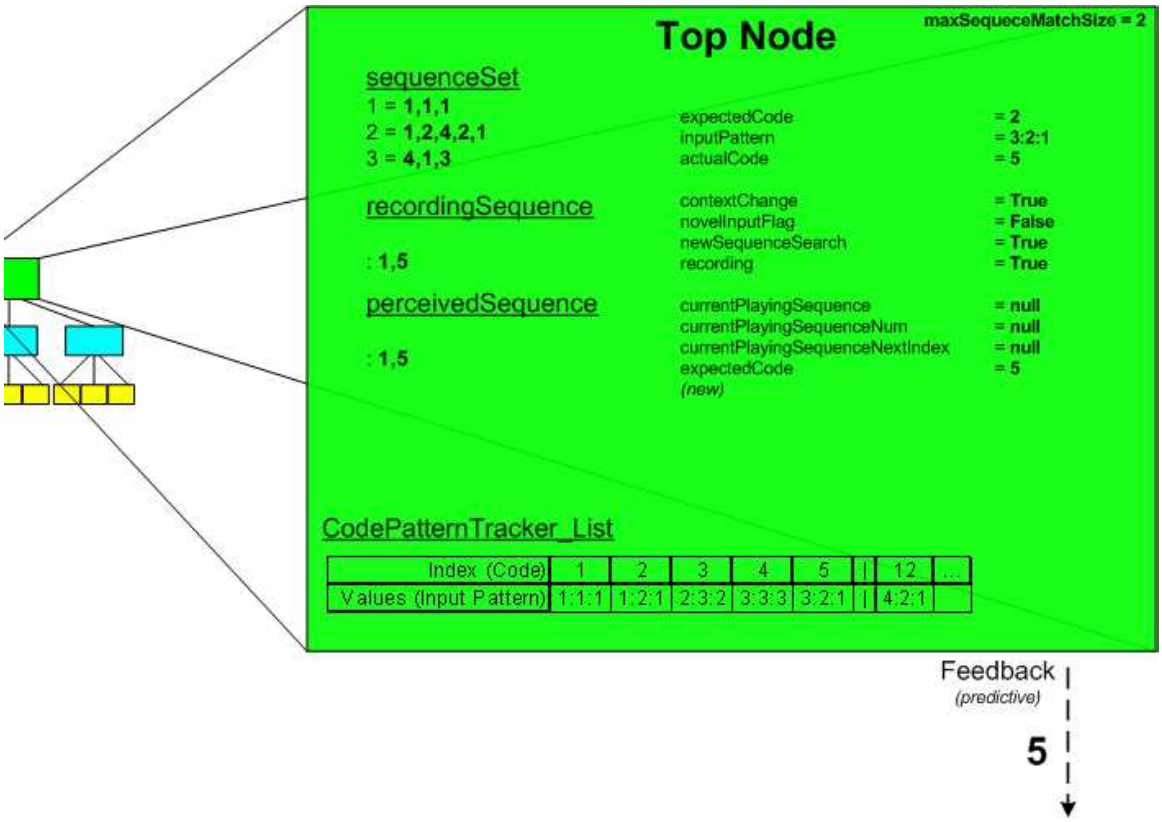

Figure 5.6: Illustration: Top Node Operations - Recording

simply predicts the last perceived invariant representation if no prediction is possible from known sequences.

As explained in the algorithm, sometimes known objects may be perceived to be new objects (or novel input) due to noise. In Figure 5.7, the Basic Node has perceived a novel input pattern from its children due to a new perception forwarded by the third child. Per the guessing algorithm, the Basic Node attempts to discern if the pattern could be recognized if not for the noise of IR:5 in pattern 1:3:5. The child's input value of IR:5 could be akin to a branch partially obscuring the view of a building recognized by pattern 1:3:3. In this way, the Basic Node is able to forward a best guess for what it has perceived. Since the nodes internal code of IR:3 is out of context (the parent had predicted IR:2) the context is said to have changed.

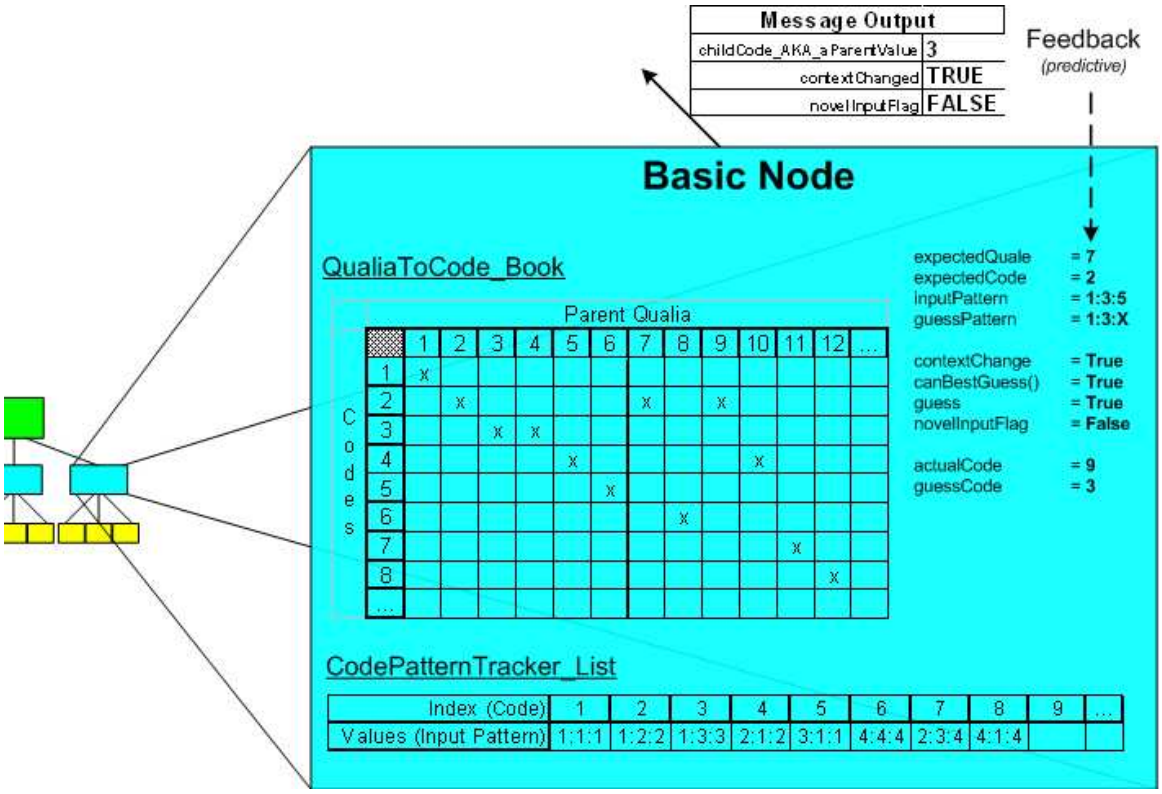

Figure 5.7: Illustration: Basic Node Operations - Ignoring Noise

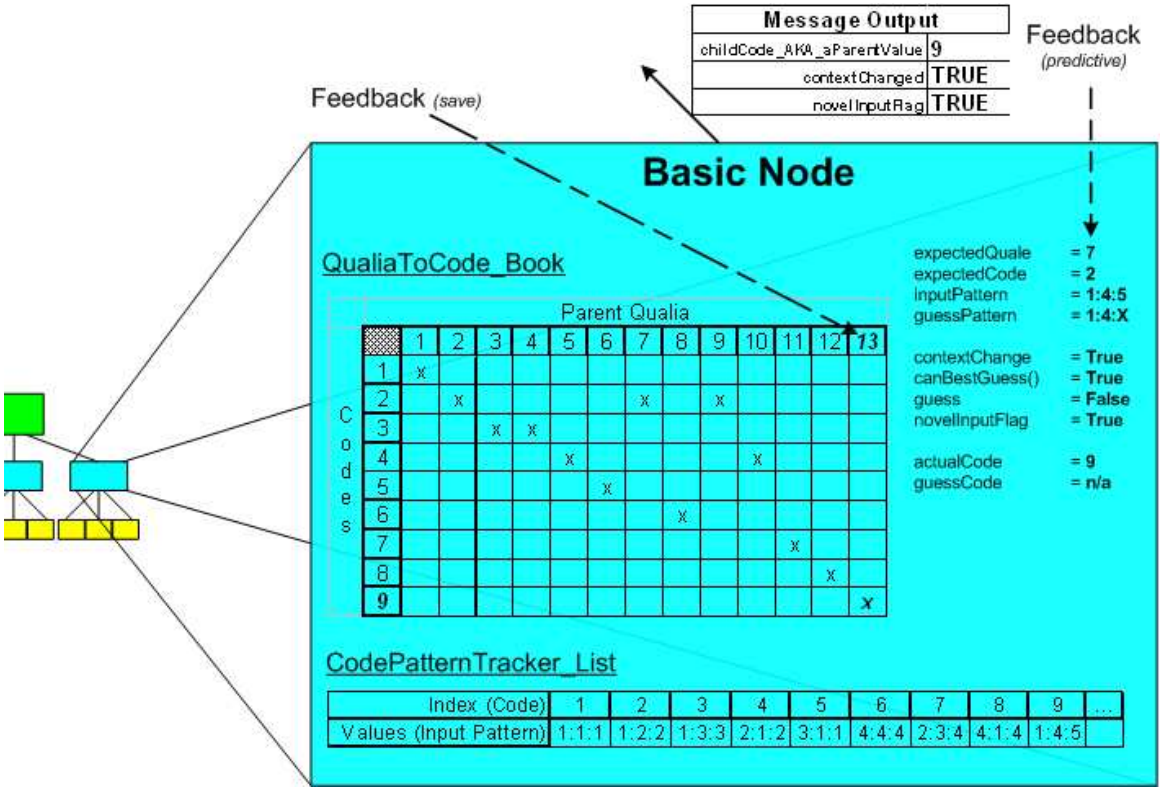

Figure 5.8: Illustration: Basic Node Operations - Saving Novel Input

Often, however, the case exists where input patterns are undoubtedly new. In Figure 5.8, the novel input pattern 1:4:5 is out of context. Although a guess is attempted, no previously known pattern fits and thus 1:4:5 is added to the knowledge pool of the tracker list and forwarded. If a Top Node concurs with the novelty of a pattern, it will instruct all nodes in the hierarchy to assign a new IR to the perceived input. That is, the parent will instruct all its children (including this Basic Node) to save their actual perceptions as registered to a new parent invariant representation. Such feedback for saving novel input is illustrated in Figure 5.9 where the novel input has also triggered the recording of a new sequence.

5.3 Proposed Experiment

The viability of HTM networks to make predictions in unusual worlds cannot be completely dismissed via the success (or failure) of a single experiment, such as the

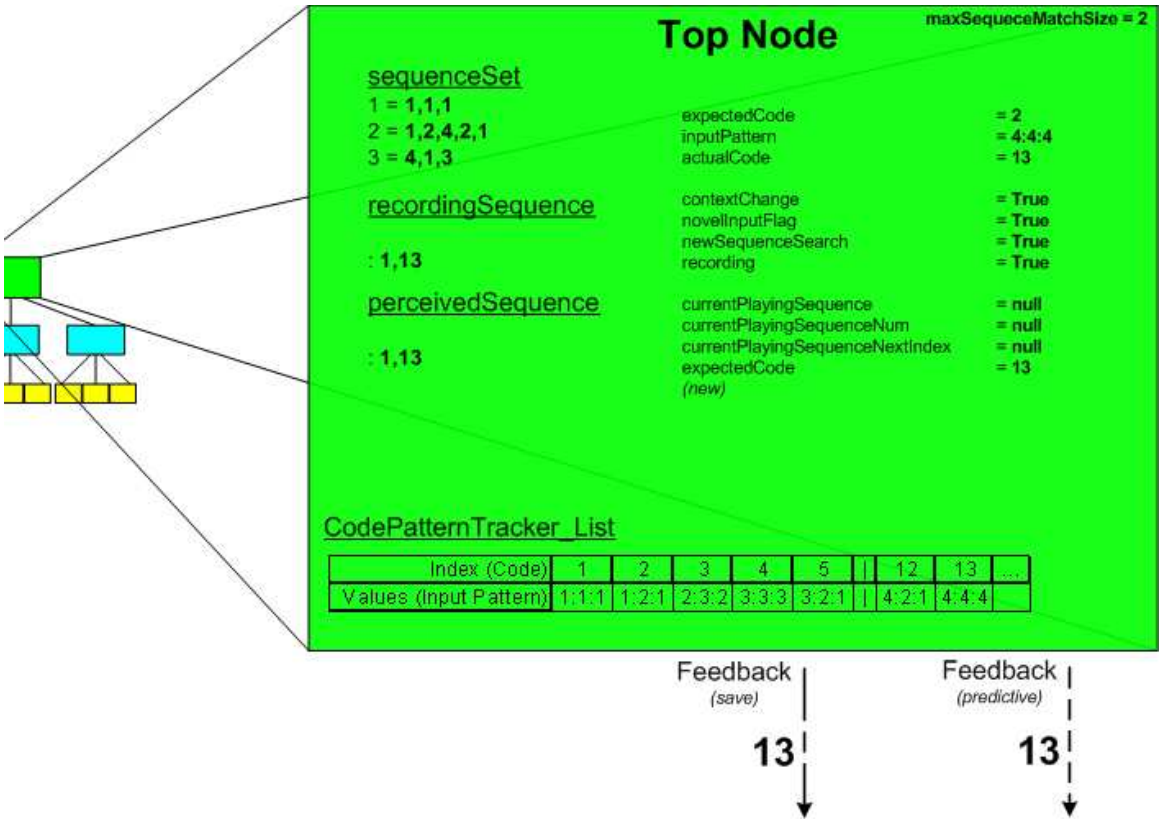

Figure 5.9: Illustration: Top Node Operations - Feedback for IR Persistence (Saving)

anomaly detection system. The BackTalk algorithm has strengths and weaknesses but seems is strongest where Numenta's implementation appears most vulnerable — incorporating feedback into prediction and learning.

A BackTalk implementation will next be tested in addition to the NuPIC implementation. However, where the more developed NuPIC can be tested with the anomaly detection experiment, the never-before implemented BackTalk requires a "hello world" experiment. BackTalk testing will be performed on a concept of "song recognition" as presented by Hawkins [11]. This concept is then extended into the primary experiment known as *Urban Challenge*.

- *Sequence Recognition - The Song Concept*

The idea that music or songs are recognized as spatial-temporal patterns is easy to understand. Humans hear the (spatial) sounds or words of a song (in temporal order) and are able to subconsciously predict the sounds. In this way, humans notice when a temporal pattern of (say) notes are wrong. The Monday Night Football jingle goes "Dun Dun Dun Da... Bum Bum..." not "Bum Dun... Da Bum Dun Dun."

The spatial pattern is more complicated to explain in songs but spatial data of a song can be understood as accompanying characteristics of any instance of sound. In keeping with the Monday Night Football example, the "Dun" of the song has more characteristics that are heard by simple phonetically pronunciation of the onomatopoeia. These spatial qualities may include the pitch, note, key, etc. In this way too, humans are able to pick out when songs are played in a new tempo or if sung "off-key."

Additionally, consider hearing a song (say) on a radio station. Assume the DJ changes the song in mid-play to the middle of some other recognized song. The human brain is able to recognize the difference (perhaps after a momentary bout of confusion) and discern that a second song is now being played.

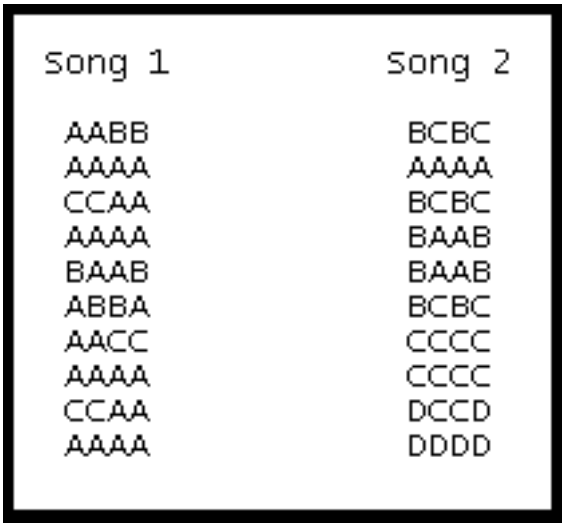

Figure 5.10: Example "Songs" for Song Recognition Problem

BackTalk should be able to act in the same manner. To clarify, a pedagogical problem can thus be stated:

Consider a song "Song 1" to be a temporal sequence composed of spatial elements: notes, keys, pitches, and sounds. Let each spatial element variance be represented by alphabetic symbols (A, B, C, etc.). Let the spatial element in this song be represented at each temporal unit by a 4-element "word" vector (i.e. ABCD, AABB, etc.). Let Song 1 and Song 2 consist of the sequence of 10 words in Figure 5.10.

After supervised learning of each song, BackTalk, given Song 1 or Song 2, should be able to distinguish which song is playing. Additionally if part of Song 1 and Song 2 are merged together, BackTalk should be able to tell when Song 1 playing and when Song 2 is playing (See Figure 5.11.

- **Initial Experiment - "Urban Challenge"**

From the song recognition problem, the following experiment is proposed.

Let each song represent a text-based segment (See Figure 5.12) of road where:

Input	Perceived As Song	Input	Perceived As Song
BCBC	Song 2	AABB	Song 1
AAAA	Song 2	AAAA	Song 1
BCBC	Song 2	CCAA	Song 1
BAAB	Song 2	BAAB	Song 2
BAAB	Song 2	BAAB	Song 2
AABB	Song 1	BCBC	Song 2
AAAA	Song 1	CCCC	Song 2
CCAA	Song 1	AAAA	Song 1
AAAA	Song 1	CCAA	Song 1
BAAB	Song 1	AAAA	Song 1

Figure 5.11: Example of correct perceptions of "Songs" for Song Recognition Problem

Figure 5.12: Proposed Urban Challenge Experiment

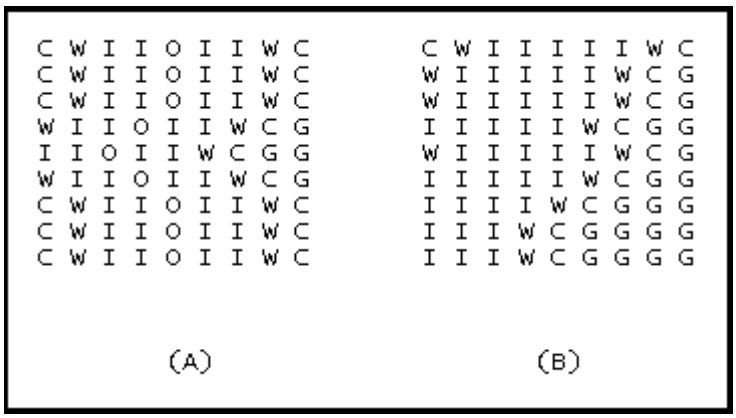

Figure 5.13: Example of Urban Challenge Experiment Training (A) and Testing (B) Data

I = Interstate
W = White Line
C = Curb
G = Grass
O = Car Location

These segments can be read by BackTalk during training. A stretch of road containing 'O' would be an example of how to teach BackTalk to keep the car centered during a left turn in the road (See Figure 5.13.A).

Notice that the correct location of a car is provided at each step of the road. After a short training period, the BackTalk should be "set free to drive itself"' on a given stretch of random road without car placement. Using the algorithm's predictive, learning, and context distinguishing abilities, the BackTalk should be able to predict the placement of the car. A sample of road used for testing can be seen in Figure 5.13.B.

Notice that BackTalk has never seen an elongated, gradual turn in a stretch of road. Nor does this stretch of road move all the way over to the end of the text vector as in training. It will be BackTalk's job to place the car (O-symbol) on the road without running over the curb. If HTM theory is correct (and correctly implemented) the BackTalk algorithm should learn, adapt, react, and predict how to stay off the grass given a random stretch of text-road.

5.4 Evaluation Framework

Reasonable expectation of BackTalk success is, ultimately, as follows: BackTalk must keep the "car" on the "interstate." More precisely, Quiliabear should be able to perform the following:

1. *Ability to learn from initial training in an unsupervised way.* That is, the BackTalk application should be shown a small set off driving maneuvers and it should self-internalize what it perceives as invariant representations of spatial-temporal patterns.
2. *Show intelligent predictive ability while driving.* That is, BackTalk should be able to perceive the current state of the road and provide an accurate prediction of where the car should be driven next. Predictions resulting in car placement either on a "curb" or the "grass" should be evaluated in this experiment to be incorrect placement. Placement on the "interstate" or even a "white line" should be considered correct car placement.
3. *Poses the capability to adapt to and learn from novel input.* That is, after training in how to "drive straight," "turn left," and "turn right," the BackTalk should learn to handle turns of any magnitude or straight driving at any required point based on these initial invariant representations of its model. Random roads should not "throw off" the driving BackTalk.

VI. Conclusions

This chapter attempts to coax significance from knowledge learned during background research and analysis of each HTM experiment. The problems which this paper attempts to solve are recounted before experimental results are examined. Finally, any significant conclusions are provided along with suggestions for future work.

6.1 Problem Summary

Two questions were posed in this research and can be quickly summarized. Can HTM provide an understanding into the abstract world of cyberspace? That is, can an HTM network show that it has learned spatial-temporal patterns in network traffic despite the lack of a human teacher that could recognize those same patterns. Can those learned patterns then be use by the HTM network to make accurate, intelligent predictions and decisions? The second question is, are the predictive foundations of intelligence supported by HTM? Unless the premises is false, it is reasonable to expect an HTM theory-based algorithm to use accurate predictions to solve problems in an intelligent manner. If predictions are made but the problem cannot be solved by those predictions, mere prediction as a foundation of intelligence seems unlikely.

6.2 Interpretation of Results

Because of the vast differences between the two HTM implementations, their respective experiments, and the experimental focus/goals, the corresponding results must be interpreted *completely* separately and independently.

6.2.1 Anomaly Detection Experiment. The overall assessment of NuPIC and N-HTM networks as a viable option for predictive, understanding of the cyberspace environment is not favorable. In fact, according to every set of results attained form over 30 varied experiments, not a single NuPIC created N-HTM network was able to function as a mediocre (let alone good) anomaly detection application.

The complete failure of even the most promising (Lv3T MW3-60K) N-HTM network to detect a difference between malicious, attack-packets and benign, nominal-packets is hard to deny. The only problem presented by the results is the *cause* of the failure. One thing seems certain, Numenta's HTM implementation seems unable to provide users a credible means to interpret intangible worlds as the theory indicates is possible.

But are the disappointing results symptomatic of a fundamental flaw in HTM theory itself? Perhaps the root cause are the obvious inconsistencies between Numenta's N-HTM networks and the quintessential HTM network proposed from Hawkins' cortical theories. On the other hand, poor performance could simply be bugs that cause NuPIC to perform inappropriately. Maybe the N-HTM network topology needs to be reworked from the ground up. Perhaps the Python code used to construct, train, and test the N-HTM is written incorrectly (Source Code can be found in Appendix A) due to oversights or fundamental flaws in the operation of NuPIC.

Because extensive testing was performed and results obtained, the inappropriate use of NuPIC seems improbable. Although (per Appendix B) a nearly equal number of categories were typically perceived in testing data as were created from training, the fact that the testing data was (at times) 25 times larger than that used in training indicates that NuPIC is operating correctly.

Perhaps far more training than 60,000 packets is required to construct an accurate model of "normal" cyberspace. PHAD, after all, used and entire week's worth of data for training [23]. If this is the case, NuPIC must be redesigned to accommodate the gigantic memory requirements of (potentially) several million training vectors. Such a suggestions begs the question, how is using a N-HTM network better than using traditional AI approaches or cyber-defense software where all possible conditions are programmed into memory for comparison.

HTM theory seems, if nothing else, to be an elegant, theoretical solution to the complex problem of intelligence. Backed up by both biological data and philosophical

hypotheses, the cause of the problems do not immediately point to the incorrectness of HTM. A failure of NuPIC to perform predictions in cyberspace is not as devastating to HTM theory as the implementation is loosely tied to HTM theory to begin with (i.e. not feedback).

In fact, Numenta so greatly diverged from vital concepts in HTM theory that the term N-HTM networks was coined in this paper to indicate the disparity. This divergence between theory and implementation point to the most likely culprit of the failure. Indeed, Numenta's basic disregard for consistency between their NuPIC implementation and Hawkins' HTM and cortical theories might be a cause of inept anomaly detection through an N-HTM network.

For this reason, the failures associated with this experiment can be said to indicate that NuPIC and N-HTM networks are not viable candidates for unsupervised learning and prediction in unusual worlds. However, because NuPIC is currently the only accepted HTM implementation readily available, a conclusion of HTM theory's inability to make predictions in unusual worlds must still be considered.

6.2.2 Urban Challenge Experiment. Although other AI methods certainly could solve the Song Recognition or Urban Challenge problems easily, finding the most efficient and effective way to solve such problems was not the goal of BackTalk experimentation. Indeed, a simple "if given this string move the car here" approach would solve the Urban Challenge problem faster, more effectively, and with far less code than used in the BackTalk solution. However, the purpose for the BackTalk experiments is to provide indications of the worth of feedback in HTM networks and the applicability of prediction to the intelligence equation.

In must be emphasized that the following results, especially on a toy problem, would have no bearing on HTM theory's applicability to the understanding cyberspace. The obvious addition to this statement is to point out BackTalks further, direct applicability to the cybercraft program. Additionally, there is no reason to suggest that this programmer's interpretation is a more accurate representation

of an HTM implementation. To that end, in many ways, Numenta's implementation (i.e. weighted learning of invariant representations) may be more accurate. This implementations has three main proposes:

1. Test the importance of feedback to unsupervised learning of spatial-temporal patterns.
2. Test the importance of feedback in an HTM model for accurate predictions.
3. Test the predictive foundations of intelligence.

It is with pleasant and curious surprise, however, that BackTalk exceeds all expectations by never once causing a car to run off the road and into the ditch (overlaying an 'O' on the static 'C' or 'G' characters of a text-road). Although the "driving" of BackTalk can be erratic at times (in some trials - see Appendix D), the fact that the car never crosses the white line is (seemingly) impressive.

Of more interest to the questions regarding the correctness and applicability of feedback and prediction in HTM theory, is the *way* in which the BackTalk learns. The unsupervised, continuous learning adequately follows learning as proposed in HTM theory. Further, the sequences learned by the Top Node often seem nonsensical but sufficiently match the concept of invariant representation. Additionally, the fact that every BackTalk performed flawlessly, despite learning varying numbers and classes of sequences is consistent with the individuality accounted for in cortical and HTM theory.

Perhaps, most the significant interpretation of BackTalk results is the apparent affirmation or the important, dual-nature of prediction and feedback in HTM theory. Predictions (as only invariant representations) fed back down the HTM network appear to adequately facilitate learning and reactionary behavior.

Similarities between BackTalk's HTM implementation and the feedback, topology, and statistical analysis characteristics of other AI methods (i.e. ANNs, Baysian networks, or HMMs) seem to exist. Perhaps BackTalk more adequately falls under

one of those other methods than HTM. Or, perhaps, HTM theory, itself, falls under one of those methods. Either way, the classification of BackTalk's methodology is not important and is in no way meant to detract from proven AI methods. What is important is that BackTalk was built solely from study of HTM theory (mixed with an applicable concept like context). Resemblance to other AI methods is provably coincidental.

So, does BackTalk testing indicate the importance and usefulness of feedback in HTM theory? Because of the essential nature of feedback to the Urban Challenge experiment, and because of BackTalk's development from HTM theory, this conclusion is supported. Additionally, feedback appears important to learning both to HTM theory and the success of this algorithm. There may even be a correlation between the intelligent behavior of BackTalk and the predictive foundations of HTM theory due to the HTM basis of the applied algorithm. From that assertion a conclusion could be drawn that accurate intelligence predictions in HTM theory can benefit from feedback within an HTM network.

6.3 Significance of Research

The research conducted by this thesis has several implications.

1. The predictive foundations of intelligence appear to be supported by this research.
2. The biological foundations which establish HTM theory are reinforced.
3. Feedback appears beneficial to some successful applications of HTM like the implementation known as BackTalk.
4. HTM theory, although proclaimed to be able to interpret intangible worlds, shows no evidence of being able to understand anything as complicated as cyberspace. Even the successful navigation of the alien text world presented to BalkTalk is not complicated enough to validate these claims.

6.4 Recommendations for Future Research

The tantalizing research on the nature of intelligence is combined with the results and analysis from two experiments to encourage potential studies into HTM theory and implementations.

The first proposal is for a similar anomaly detection experiment be attempted with a grander, cleaner BackTalk implementation. Additional uses and experiments for similar, unsupervised BackTalk implementations may include UAV target tracking or robotics navigation.

Another avenue for research is the re-testing of new NuPIC versions with the anomaly detection framework. Numenta is constantly refining and developing their algorithms. Their research website indicates that, in the near future, a Zeta2 memory node will be released that incorporates the missing elements of feedback. Once such feedback is added to a N-HTM network, re-testing could be warranted.

Appendix A. NuPIC Cybercraft Python Code - Anomaly Detection

This appendix provides the pertinent Python code to implement the Anomaly Detection experiment under NuPIC version 1.3 for Microsoft Windows.

A.1 LaunchCybercraft.py

```
"""
## @file
Cybercraft Anomaly Detection Analysis Program.

The goal of this program is to show the ``cybercraft" N-HTM network
TCPdump data using the Numenta Tools and see if it can then detect
anomolus patterns.  To run it, type:

       python LaunchCybercraft.py

from the command line.

The domain is that of Ethernet+IP+TCP packets convered to the float vector format:

SENSOR INPUT EX:
31
1 0 192 79 163 87 219 16 123 56 70 51 2048 172 16 114 148 197 218 177 69 44 238 6 2 1025 21 2850393447 0 0 512

| (ELEMENT) NAME |
| (0)ID |
| (1)timeDeltaBetweenPackets |
| (2-6)sourceMAC | (7-11)destinationMAC | (12)type |
| (13-16)sourceIP | (17-20)destinationIP | (21)totalLength | (22)identification | (23)protocol | (24)flagsIP |
| (25)sourcePort | (26)destinationPort | (27)sequenceNum | (28)AcknowledgementNum | (29)flagsTCP | (30)windowSize |


Textual Spec of Network:


(FILE)EFFECTOR - LV3[NODE] - LV2[NODE] - LV1[NODES] << SENSOR
             /------[P]---------[P]---------[P]     << SENSOR
            /           /-------------------[0]     << SENSOR
           /           /--------[0]---------[1-3]   << SENSOR
(file)---X----------[0]---------[1]---------[4-9]   << SENSOR
                      \--------[2]---------[10-15] << SENSOR


The N-HTM Network will create a statistical model of this world and use this model
to distinguish between patters it learns.  No programming is performed.

The cybercraft program goes through the following steps:

   1. Create a N-HTM Network with default parameters.
   2. Create a trained network using training data file.
   3. Test the network by running inference on the original training set.
   4. Further test the network by running inference on an independent test set.

If set correctly, this program will test a trained cybercraft on many test files.

Please refer to the Numenta Programmer's Guide for further details.
"""
```

```
from CreateCybercraftNetwork import createCybercraftNetwork
from TrainCybercraftNetwork import trainCybercraftNetwork
from RunCybercraftInference import runCybercraftInference
from nupic.analysis import Visualizer
import os
import shutil
##from GenerateReport import generateReport

#===============================================================================
# The main routine
#===============================================================================
def launchCybercraft():
    """
    Run one cycle of N-HTM training and testing.
    """
    pick = raw_input( "Would you like to quick launch the cybercraft? <y/n>>> ")
    if pick == "y" or pick == "Y":
        slow = False
    else:
        slow = True

    print "    Initializing parameters... \n"
    if slow:
        pick = raw_input( "Would you like to reconstruct the cybercraft? <y/n>>> ")

    if pick == "y" or pick == "Y":
        # Create the cybercraft network.
        print "\nBuilding Cybercraft N-HTM Network..."

        createCybercraftNetwork(untrainedNetwork = untrainedNetwork,
                  inputSize        = inputSize,
                  maxDistance      = maxDistance,
                  topNeighbors     = topNeighbors,
                  maxGroups        = maxGroups,
                  transitionMemory = transitionMemory,
                  maxGroupSize     = maxGroupSize,
                  timeless         = timeless,
                  level4           = level4)

        print "    Cybercraft N-HTM Network construction complete.\n"

    if slow:
        pick = raw_input( "Would you like to retrain the cybercraft? <y/n>>> ")
    if pick == "y" or pick == "Y":
        # Train the network
        print "\nTraining Cybercraft N-HTM Network with "+str(numTrainingVectors)+" vectors..."

        trainCybercraftNetwork(untrainedNetwork, trainedNetwork, trainingFile, numTrainingVectors,level4)

        print "    Cybercraft N-HTM Network training complete.\n"

        # Ensure the network learned the training set

        print "\nValidating autonomous cybercraft intuition...."

        runCybercraftInference(trainedNetwork, trainingFile, trainingResults, numTrainingVectors)

        print "    Independent learning confirmed: Cybercraft intuition validated.\n"
```

```
    if slow:
        pick = raw_input( "Would you like to test the cybercraft? <y/n>>> ")
    if pick == "y" or pick == "Y":

        print "\nCommencing packet categorization of test data with "+str(numTestVectors)+" vectors..."

        # Run inference on test set to check generalization

        runCybercraftInference(trainedNetwork, testFile, testResults, numTestVectors)

        # Write out a report of the overall network progress

        print "    All test data has been categorized.\n"

    if slow:
        pick = raw_input( "Would you like to create a HTML Histogram the cybercraft? <y/n>>> ")
    if pick == "y" or pick == "Y":

        print "\nBuilding cybercraft visualization..."

## SLOW METHOD
##  FOR ALL
##        v = Visualizer(trainedNetwork, "bar_graph")
##        v.visualizeNetwork()
##  PICK A PIECE
##        v = Visualizer(trainedNetwork, "bar_graph")
##        v.visualizeNetwork("Level1.*")
##        v.visualizeNetwork("Level2.*")
##        v.visualizeNetwork("Level3.*")
##        if level4:
##            v.visualizeNetwork("Level4.*")

## FAST METHOD
        v = Visualizer(trainedNetwork, "blank")
        v.visualizeNetwork()

        print "    Cybercraft visualization complete.\n"

    print "\nEnding processes...DONE"

    pick = raw_input( "Hit Enter to exit>>> ")
#===============================================================================
# List of parameters used in the example
#===============================================================================

# Learning/Network creation parameters
maxDistance                = 0.0
topNeighbors               = 2
maxGroups                  = 16000
maxGroupSize               = 2000000
transitionMemory           = 100
inputSize                  = 31
timeless                   = True
level4                     = False

#!!Experiment Information!!

experimentDateTime = "241007_1315"

trainingData       = "Wk3Mon_out"
```

```
trainingMax        = False
numTrainingVectors = 60000
testingData        = "Wk4Mon_out"
testingMax         = True
numTestVectors     = 1000000

# Training/Inference parameters

#TRAINING#
if trainingMax:
    if trainingData == "Wk1Mon1":
        numTrainingVectors = 1369134-1
    elif trainingData == "Wk1Tue2":
        numTrainingVectors = 1153247-1
    elif trainingData == "Wk1Wed3":
        numTrainingVectors = 1574274-1
    elif trainingData == "Wk3Thr18":
        numTrainingVectors = 1152296-1
    elif trainingData == "Wk1Fri5":
        numTrainingVectors = 1363645-1
    elif trainingData == "Wk3Mon_out":
        numTrainingVectors = 1454465-1

#TEST#
if testingMax:
    if testingData == "Wk2Mon8":
        numTestVectors = 1364563-1
    elif testingData == "Wk2Tue9":
        numTestVectors = 1482408-1
    elif testingData == "Wk2Wed10":
        numTestVectors = 922164-1
    elif testingData == "Wk2Thr11":
        numTestVectors = 1474671-1
    elif testingData == "Wk2Fri12":
        numTestVectors = 1280621-1
    elif testingData == "Wk4Mon_out":
        numTestVectors = 1167662-1

# Various file names
                                                                #Locatiuon of cybercraft project
homeDir = "D:\\Documents and Settings\\gbonhoff\\My Documents\\nupic-1.3\\share\\projects\\cybercraft\\"
                                                                #Locatiuon of cybercraft training/testing data
dataLocation       = homeDir+"data\\vectorForm\\"

lv = "Lv3"
t = ""
if level4:
    lv = "Lv4"
if timeless:
    t = "T"

dataUsed = lv+t+"_"+trainingData+"_"+str(numTrainingVectors)+"-"+testingData+"_"+str(numTestVectors)

                                                                #Locatiuon of cybercraft files/reports/results
birthingPlace      = homeDir+"data\\cybercraft\\"+experimentDateTime+"\\"
if not os.path.isdir(birthingPlace):
    os.mkdir(birthingPlace)

birthingPlace      = homeDir+"data\\cybercraft\\"+experimentDateTime+"\\"+dataUsed+"\\"
if not os.path.isdir(birthingPlace):
```

```
    os.mkdir(birthingPlace)
shutil.copyfile(homeDir+"LaunchCybercraft.py",birthingPlace+"LaunchCybercraft.py")
shutil.copyfile(homeDir+"CreateCybercraftNetwork.py",birthingPlace+"CreateCybercraftNetwork.py")

untrainedNetwork   = birthingPlace + "untrained_cybercraft.xml"     # Name of the untrained network
trainedNetwork     = birthingPlace + "trained_cybercraft.xml"       # Name of the trained network
trainingFile       = dataLocation  + "training_data_"+trainingData+".txt"             # Location of training data
trainingResults    = birthingPlace + "training_results.txt"         # File containing inference
                                                                      # results for each training pattern
testFile           = dataLocation  + "test_data_"+testingData+".txt"             # Location of test data
testResults        = birthingPlace + "test_results.txt"             # File containing inference

#===============================================================================
# When invoked from command line, create network, train it, and run inference
#===============================================================================
if __name__ == '__main__':
  print "Cybercraft protocol initiated..."

  launchCybercraft()
```

A.2 CreateCybercraftNetwork.py

```
"""
## @file
This class generates the untrained HTM Network for the Cybercraft.
This file is templated on Bitworm which is Copyright (C) 2007 Numenta Inc
"""

from nupic.network import *

"""
This class generates a 3(or 4)-level network. The network
consists of 8-9 nodes: 1 input sensor node, no category sensor nodes,
15 unsupervised level 1 nodes, 3 unsupervised level 2 nodes,
1 (maybe) unsupervised level 3 nodes with a Zeta1TopNode (3rd or 4th level as appl)
and an effector node (4rd or 5th level as appl)at the top.
"""

def createCybercraftNetwork(untrainedNetwork,
                      inputSize=31,
                      topNeighbors=4,
                      maxGroups=2000,
                      maxGroupSize=2000000,
                      maxDistance=0,
                      transitionMemory=10,
                      timeless = False,
                      level4 = False):
  """
  Generates the HTM Network using the parameters.
  Creates each of the nodes types for each level and
  then links them together into regions, finally it links the
  regions and saves the resulting network in a file.

  @param  name    The name of the output network file
  """

  #-------------------------------------------------------
  # Step 1. We create a Net instance and initalize phase = 0
```

```
net = Network()
phase = 0
#-------------------------------------------------------
# Step 2. We create each node type, applicable regions,
# and add them to the network

# Create and add sensor to read in the data file
sensor = CreateNode("VectorFileSensor",
                    phase=phase,
                    dataOut= inputSize)

net.addElement("Sensor", sensor)

L1SPA = "gaussian"
L1TPA = "sumProp"

# Create level 1 unsupervised node region and add it to the net
phase = phase+1
addysPorts = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups,
                        spatialPoolerAlgorithm= L1SPA,
                        maxDistance=maxDistance,
                        maxGroupSize=1,
                         ##sigma=,
                         #TEMPORAL SETTINGS
                        temporalPoolerAlgorithm= L1TPA,
                        topNeighbors=topNeighbors,
                        transitionMemory=transitionMemory,
                        symmetricTime=False,
                         #OTHER SETTINGS
                        detectBlanks=0,
                        phase=phase)

net.addElement("Level"+str(phase), Region (7, addysPorts))

timeNode = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups,
                        spatialPoolerAlgorithm= L1SPA,
                        maxDistance=0.0001,
                        maxGroupSize=maxGroupSize,
                         ##sigma=,
                         #TEMPORAL SETTINGS
                        temporalPoolerAlgorithm= L1TPA,
                        topNeighbors=topNeighbors,
                        transitionMemory=transitionMemory,
                        symmetricTime=False,
                         #OTHER SETTINGS
                        detectBlanks=0,
                        phase=phase)

net.addElement("Level"+str(phase)+"_time",timeNode)

ipSize = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups,
                        spatialPoolerAlgorithm= L1SPA,
                        maxDistance=512,
                        maxGroupSize=maxGroupSize,
```

```
                         ##sigma=,
                         #TEMPORAL SETTINGS
                        temporalPoolerAlgorithm= L1TPA,
                        topNeighbors=topNeighbors,
                        transitionMemory=transitionMemory,
                        symmetricTime=False,
                         #OTHER SETTINGS
                        detectBlanks=0,
                        phase=phase)

net.addElement("Level"+str(phase)+"_sizeIP",ipSize)

tcpSize = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups,
                        spatialPoolerAlgorithm= L1SPA,
                        maxDistance=512,
                        maxGroupSize=maxGroupSize,
                         ##sigma=,
                         #TEMPORAL SETTINGS
                        temporalPoolerAlgorithm= L1TPA,
                        topNeighbors=topNeighbors,
                        transitionMemory=transitionMemory,
                        symmetricTime=False,
                         #OTHER SETTINGS
                        detectBlanks=0,
                        phase=phase)

net.addElement("Level"+str(phase)+"_sizeTCP",tcpSize)
# Create level 2 unsupervised node region and add it to the net

phase = phase+1
nodesLevel2 = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups*((phase-1)*2),
                        spatialPoolerAlgorithm= "gaussian",
                        maxDistance=maxDistance,
                        maxGroupSize=maxGroupSize,
                         ##sigma=,
                         #TEMPORAL SETTINGS
                        temporalPoolerAlgorithm= "sumProp",
                        topNeighbors=topNeighbors,
                        transitionMemory=transitionMemory,
                        symmetricTime=False,
                         #OTHER SETTINGS
                        detectBlanks=0,
                        phase=phase)

net.addElement("Level"+str(phase), Region (3, nodesLevel2))

if level4:
  print "Adding a 4th level..."
  phase = phase+1
  addInLevel = CreateNode("Zeta1Node",
                         #SPATIAL SETTINGS
                        bottomUpOut=1+maxGroups*((phase-1)*2),
                        spatialPoolerAlgorithm= "product",
                        maxDistance=maxDistance,
                        maxGroupSize=maxGroupSize,
                         ##sigma=,
```

```
                          #TEMPORAL SETTINGS
                          temporalPoolerAlgorithm= "sumProp",
                          topNeighbors=topNeighbors,
                          transitionMemory=transitionMemory,
                          symmetricTime=False,
                          #OTHER SETTINGS
                          detectBlanks=0,
                          phase=phase)

    net.addElement("Level"+str(phase), addInLevel)


  # Create Supervised node that can learn mappings for X categories
  phase = phase+1
  topNode = CreateNode("Zeta1TopNode",
                       phase=phase,
                       spatialPoolerAlgorithm= "dot",
                       mapperAlgorithm= "sumProp",
                       categoriesOut=1+maxGroups*((phase-1)*2))

  net.addElement("Level"+str(phase), topNode)

  # Create Effector for sending output to a file
  phase = phase+1
  effector = CreateNode("VectorFileEffector", phase = phase)
  net.addElement("FileOutput",effector)

  #-------------------------------------------------------
  # Step 3. We create each link
  #0=sm, 1=dm, 2=sip, 3=dip, 4=sp, 5=dp, 6=fip, 7=time, 8=ipSize, 9=tcpSize
  # Link sensor to unsupervised node
  net.link("Sensor","Level1_time", SingleLink("dataOut",1,1,"bottomUpIn"))
  net.link("Sensor","Level1[0]", SingleLink("dataOut",2,5,"bottomUpIn"))
  net.link("Sensor","Level1[1]", SingleLink("dataOut",7,5,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1[3]", SingleLink("dataOut",12,1,"bottomUpIn"))
  net.link("Sensor","Level1[2]", SingleLink("dataOut",13,4,"bottomUpIn"))
  net.link("Sensor","Level1[3]", SingleLink("dataOut",17,4,"bottomUpIn"))
  net.link("Sensor","Level1_sizeIP", SingleLink("dataOut",21,1,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1_time", SingleLink("dataOut",22,1,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1_sizeIP", SingleLink("dataOut",23,1,"bottomUpIn"))
  net.link("Sensor","Level1[6]", SingleLink("dataOut",24,1,"bottomUpIn"))
  net.link("Sensor","Level1[4]", SingleLink("dataOut",25,1,"bottomUpIn"))
  net.link("Sensor","Level1[5]", SingleLink("dataOut",26,1,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1[12]", SingleLink("dataOut",27,1,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1[13]", SingleLink("dataOut",28,1,"bottomUpIn"))
#REMOVED#  net.link("Sensor","Level1[14]", SingleLink("dataOut",29,1,"bottomUpIn"))
  net.link("Sensor","Level1_sizeTCP", SingleLink("dataOut",30,1,"bottomUpIn"))

  # Link unsupervised node to unsupervised node
  net.link("Level1[0]","bottomUpOut","Level2[0]", "bottomUpIn")
  net.link("Level1[1]","bottomUpOut","Level2[0]", "bottomUpIn")
#REMOVED#  net.link("Level1[3]","bottomUpOut","Level2[0]", "bottomUpIn")
  net.link("Level1[2]","bottomUpOut","Level2[1]", "bottomUpIn")
  net.link("Level1[3]","bottomUpOut","Level2[1]", "bottomUpIn")
  net.link("Level1_sizeIP","bottomUpOut","Level2[1]", "bottomUpIn")
  net.link("Level1[6]","bottomUpOut","Level2[1]", "bottomUpIn")
#REMOVED#  net.link("Level1_sizeIP","bottomUpOut","Level2[1]", "bottomUpIn")
#REMOVED#  net.link("Level1_sizeTCP","bottomUpOut","Level2[1]", "bottomUpIn")
  net.link("Level1[4]","bottomUpOut","Level2[2]", "bottomUpIn")
  net.link("Level1[5]","bottomUpOut","Level2[2]", "bottomUpIn")
```

```
  net.link("Level1_sizeTCP","bottomUpOut","Level2[2]", "bottomUpIn")
#REMOVED#  net.link("Level1[13]","bottomUpOut","Level2[2]", "bottomUpIn")
#REMOVED#  net.link("Level1[14]","bottomUpOut","Level2[2]", "bottomUpIn")
#REMOVED#  net.link("Level1[15]","bottomUpOut","Level2[2]", "bottomUpIn")

  # Link unsupervised node to unsupervised node
  net.link("Level2","Level3", SimpleFanIn("bottomUpOut", "bottomUpIn"))
  if timeless:
    print "Time delta packet data omitted from hierarchy!\n"
  else:
    print "Configuring for time delta in hierarchy..."
    net.link("Level1_time","Level3", SingleLink("bottomUpOut", "bottomUpIn"))
    print "Done!\n"

  if level4:
    # Link Level4 node's output to the file writing effector
    net.link("Level3","bottomUpOut","Level4", "bottomUpIn")
    net.link("Level4","FileOutput", SingleLink("categoriesOut", 0, 1, "dataIn"))
    print "    Level 4 added sucsessfully!"
  else:
      net.link("Level3","FileOutput", SingleLink("categoriesOut", 0, 1, "dataIn"))

  #-------------------------------------------------------
  # Step 3.5 Link Sensor to FileOutput node
  # For the Sensor connection, we use PassThrough nodes to avoid a
  # level-skipping connection, which would prevent the pipeline scheduler from
  # working propertly (advanced)
  passThrough1 = CreateNode("PassThroughNode",
                            phase=1,
                            realOut=1)
  net.addElement("PassThrough1", passThrough1)
  passThrough2 = CreateNode("PassThroughNode",
                            phase=2,
                            realOut=1)
  net.addElement("PassThrough2", passThrough2)
  passThrough3 = CreateNode("PassThroughNode",
                            phase=3,
                            realOut=1)
  net.addElement("PassThrough3", passThrough3)

  if level4:
      passThrough4 = CreateNode("PassThroughNode",
                            phase=4,
                            realOut=1)
      net.addElement("PassThrough4", passThrough4)

  net.link("Sensor", "PassThrough1", SingleLink("dataOut", 0, 1, "realIn"))
  net.link("PassThrough1", "realOut", "PassThrough2", "realIn")
  net.link("PassThrough2", "realOut", "PassThrough3", "realIn")

  if level4:
    net.link("PassThrough3", "realOut", "PassThrough4", "realIn")
    net.link("PassThrough4", "realOut", "FileOutput", "dataIn")
  else:
    net.link("PassThrough3", "realOut", "FileOutput", "dataIn")
  #-------------------------------------------------------
  # Step 4. We save the network in a file
  net.writeXML(untrainedNetwork)

#============================================================================
```

```
# If invokedfrom the command line, just create network and save it
#============================================================================
if __name__ == '__main__':
    createCybercraftNetwork('untrained_cybercraft.xml')
    print "Saved HTM network to file 'untrained_cybercraft.xml'"
```

A.3 TrainCybercraftNetwork.py

```
"""
## @file
This class trains the network for the bitworm example.
This file is templated on Bitworm which is Copyright (C) 2007 Numenta Inc

Note: there is one extra setInference command after Level2.* training is complete:
    session.execute('Level2.*', ['setInference', '1'])
This turns on inference in the top node for subsequent testing (unlike prior
releases, in NuPIC 1.2, inference is off by default.) This command is missing in
the code fragment quoted in the Programmer's Guide.
"""

from nupic.session import Session
import sys
import os

##def trainNetwork(untrainedNetwork, trainedNetwork, trainingFile,
##                 trainingCategories, numVectors):

def trainCybercraftNetwork(untrainedNetwork, trainedNetwork, trainingFile, numVectors, level4):

  """Take the given training data files and train the network using Sessions.
  After training, the working directory will contain a trained network in
  the filename specified in trainedNetwork.
  """
  #-------------------------------------------------------
  # Setup

  # Prepare session and data files we will use.
  session = Session(os.path.join("sessions", "cybercraft"))
  session.addFiles(trainingFile)
##  session.addFiles(trainingCategories)
  session.start()

  # Load network file and disable all nodes. We will enable select nodes later.
  session.loadNetwork(untrainedNetwork)
  session.disableNodes()

  # Load training data into sensors
  session.execute("Sensor", ("loadFile", trainingFile, "0"))
##  session.execute('CategorySensor', ("loadFile", trainingCategories, "0"))

  #-------------------------------------------------------
  # Train Level 1
  # Only enable Sensor and Level1.* nodes when training Level 1
  print 'Training level 1'

  session.enableNodes("Sensor")
  session.enableNodes("Level1.*")
```

```
session.enableNodes("Level1_time")
session.enableNodes("Level1_sizeIP")
session.enableNodes("Level1_sizeTCP")
session.execute('Level1.*', ['setLearning', '1'])
session.execute('Level1_time', ['setLearning', '1'])
session.execute('Level1_sizeIP', ['setLearning', '1'])
session.execute('Level1_sizeTCP', ['setLearning', '1'])
session.compute(numVectors)
session.execute('Level1.*', ['setLearning', '0'])
session.execute('Level1_time', ['setLearning', '0'])
session.execute('Level1_sizeIP', ['setLearning', '0'])
session.execute('Level1_sizeTCP', ['setLearning', '0'])
session.execute('Level1.*', ['setInference', '1'])
session.execute('Level1_time', ['setInference', '1'])
session.execute('Level1_sizeIP', ['setInference', '1'])
session.execute('Level1_sizeTCP', ['setInference', '1'])

#-------------------------------------------------------
# Train Level 2
# While level 2 is being trained, level 1 needs to be in inference mode
print 'Training level 2'

# Make sure the sensors start at the beginning of the file.
session.execute("Sensor", ("setParameter","position", "0"))
#session.execute('CategorySensor', ("setParameter","position", "0"))

#session.enableNodes("CategorySensor")
session.enableNodes("Level2.*")
session.execute('Level2.*', ['setLearning', '1'])
session.compute(numVectors)
session.execute('Level2.*', ['setLearning', '0'])
session.execute('Level2.*', ['setInference', '1'])
#-------------------------------------------------------
# Train Level 3
# While level 3 is being trained, level 2 needs to be in inference mode
print 'Training level 3'

# Make sure the sensors start at the beginning of the file.
session.execute("Sensor", ("setParameter","position", "0"))
#session.execute('CategorySensor', ("setParameter","position", "0"))

#session.enableNodes("CategorySensor")
session.enableNodes("Level3.*")
session.execute('Level3.*', ['setLearning', '1'])
session.compute(numVectors)
session.execute('Level3.*', ['setLearning', '0'])
session.execute('Level3.*', ['setInference', '1'])
#-------------------------------------------------------
if level4:
  #-------------------------------------------------------
  # Train Level 4
  # While level 4 is being trained, level 2 needs to be in inference mode
  print 'Training level 4'

  # Make sure the sensors start at the beginning of the file.
  session.execute("Sensor", ("setParameter","position", "0"))
  #session.execute('CategorySensor', ("setParameter","position", "0"))

  #session.enableNodes("CategorySensor")
  session.enableNodes("Level4.*")
```

```
    session.execute('Level4.*', ['setLearning', '1'])
    session.compute(numVectors)
    session.execute('Level4.*', ['setLearning', '0'])
    session.execute('Level4.*', ['setInference', '1'])
    #-------------------------------------------------------

  # Save the network and retrieve trained network file from bundle
  print 'Training complete'
  session.saveRemoteNetwork(trainedNetwork)

  session.stop()
  session.getLocalBundleFiles(trainedNetwork)
  session.setCleanupLocal(False)


#===============================================================================
# If invoked from the command line, do nothing
#===============================================================================
```

A.4 RunCybercraftInfrence.py

```
"""
## @file
This class trains the network for the bitworm example.
This file is templated on Bitworm which is Copyright (C) 2007 Numenta Inc
"""

from nupic.session import Session
import sys
import os

def runCybercraftInference(trainedNetwork, testFile, resultsFile, numVectors):
  """Take the given training data and the network we've generated,
  add it to the bundle, and run inference on the network. After inference,
  the results will be in the file 'outputFile'
  """
  #-------------------------------------------------------
  # Setup

  # Prepare session and data files we will use
  session = Session(os.path.join("sessions", "cybercraft"))
  session.addFiles(testFile)
  session.start()

  # Load network file. All nodes will be enabled by default
  session.loadNetwork(trainedNetwork)
  session.execute('Sensor', ('loadFile', testFile, '0'))

  # We would like to keep the effector outputs, so set the filename
  # where they will be stored (inside the bundle)
  session.execute("FileOutput", ('setParameter', 'outputFile', resultsFile))

  #-------------------------------------------------------
  # Disable the CategorySensor, as the categories are not known
  # during inference.
##    session.disableNodes("CategorySensor")
```

```
  #-------------------------------------------------------
  # Run through training patterns
  print 'Running inference'
  session.execute("FileOutput",
      ("echo", "Numbers show Level 3 node output followed by the sensor output"))
  session.compute(numVectors)

  #-------------------------------------------------------
  # Retrieve the results file and cleanup
  session.stop()
  session.getLocalBundleFiles(resultsFile)
  session.setCleanupLocal(True)

  print "Inference complete; results in ", resultsFile

#===============================================================================
# If invoked from the command line, do nothing
#===============================================================================
```

Appendix B. Anomaly Detection - Data

B.1 Training Data Excerpt (Input)

First 50 input packets (vectors) from Monday, Week 3 in VectorFile format:

```
31
1 0 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 44 116 6 2 1024 25 2355892027 0 0 512
2 0.004685 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 44 287 6 18 25 1024 927809984 2355892028 0 32736
3 0.0002 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 40 117 6 16 1024 25 2355892028 927809985 0 32120
4 0.196696 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 126 298 6 24 25 1024 927809985 2355892028 0 32736
5 0.019231 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 40 122 6 16 1024 25 2355892028 927810071 0 32120
6 0.026168 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 65 123 6 24 1024 25 2355892028 927810071 0 32120
7 0.000505 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 66 299 6 24 25 1024 927810071 2355892053 0 32736
8 0.000475 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 65 124 6 24 1024 25 2355892053 927810097 0 32120
9 0.00081 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 87 300 6 24 25 1024 927810097 2355892078 0 32736
10 0.000326 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 81 125 6 24 1024 25 2355892078 927810144 0 32120
11 0.000821 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 88 301 6 24 25 1024 927810144 2355892119 0 32736
12 0.000371 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 80 126 6 24 1024 25 2355892119 927810192 0 32120
13 0.000837 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 79 302 6 24 25 1024 927810192 2355892159 0 32736
14 0.000306 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 46 127 6 24 1024 25 2355892159 927810231 0 32120
15 0.010984 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 40 303 6 16 25 1024 927810231 2355892165 0 32736
16 0.004283 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 90 304 6 24 25 1024 927810231 2355892165 0 32736
17 0.001132 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 1064 128 6 24 1024 25 2355892165 927810281 0 32120
18 0.014558 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 40 305 6 16 25 1024 927810281 2355893189 0 32736
19 0.000229 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 135 129 6 24 1024 25 2355893189 927810281 0 32120
20 0.015773 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 59 306 6 24 25 1024 927810281 2355893284 0 32736
21 0.00024 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 46 130 6 24 1024 25 2355893284 927810300 0 32120
22 0.000753 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 64 307 6 24 25 1024 927810300 2355893290 0 32736
23 0.000851 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 40 308 6 17 25 1024 927810324 2355893290 0 32736
24 0.000176 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 40 131 6 16 1024 25 2355893290 927810325 0 32120
25 0.001069 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 113 105 40 132 6 17 1024 25 2355893290 927810325 0 32120
26 0.000699 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 196 37 75 158 40 309 6 16 25 1024 927810325 2355893291 0 32735
27 31.042633 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 44 143 6 2 1026 25 3803014164 0 0 512
28 0.004815 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 44 665 6 18 25 1026 1349254056 3803014165 0 32736
29 0.000199 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 40 144 6 16 1026 25 3803014165 1349254057 0 32120
30 0.049387 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 125 679 6 24 25 1026 1349254057 3803014165 0 32736
31 0.011425 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 40 149 6 16 1026 25 3803014165 1349254142 0 32120
32 0.035783 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 63 150 6 24 1026 25 3803014165 1349254142 0 32120
33 0.000831 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 66 684 6 24 25 1026 1349254142 3803014188 0 32736
34 0.000483 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 63 151 6 24 1026 25 3803014188 1349254168 0 32120
35 0.00082 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 85 685 6 24 25 1026 1349254168 3803014211 0 32736
36 0.000317 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 75 152 6 24 1026 25 3803014211 1349254213 0 32120
37 0.000808 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 82 686 6 24 25 1026 1349254213 3803014246 0 32736
38 0.000361 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 76 153 6 24 1026 25 3803014246 1349254255 0 32120
39 0.000814 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 75 687 6 24 25 1026 1349254255 3803014282 0 32736
40 0.000305 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 46 154 6 24 1026 25 3803014282 1349254290 0 32120
41 0.002317 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 90 688 6 24 25 1026 1349254290 3803014288 0 32736
42 0.001141 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 1064 155 6 24 1026 25 3803014288 1349254340 0 32120
43 0.01734 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 40 690 6 16 25 1026 1349254340 3803015312 0 32736
44 0.000993 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 1051 156 6 24 1026 25 3803015312 1349254340 0 32120
45 0.003849 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 59 691 6 24 25 1026 1349254340 3803016323 0 32736
46 0.000244 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 46 157 6 24 1026 25 3803016323 1349254359 0 32120
47 0.000746 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 64 692 6 24 25 1026 1349254359 3803016329 0 32736
48 0.001002 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 84 197 182 91 233 40 693 6 17 25 1026 1349254383 3803016329 0 32736
49 0.000203 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 40 158 6 16 1026 25 3803016329 1349254384 0 32120
50 0.007256 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 84 40 164 6 17 1026 25 3803016329 1349254384 0 32120
```

B.2 Testing Data Excerpt (Input)

First 50 input packets (vectors) from Monday, Week 4 in VectorFile format:

```
31
1 0 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 44 310 6 2 1024 25 2758648148 0 0 512
2 0.000347 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 44 364 6 18 25 1024 3449610373 2758648149 0 32736
3 0.002809 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 40 311 6 16 1024 25 2758648149 3449610374 0 32120
4 0.25269 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 126 375 6 24 25 1024 3449610374 2758648149 0 32736
5 0.013744 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 40 316 6 16 1024 25 2758648149 3449610460 0 32120
6 0.029191 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 66 317 6 24 1024 25 2758648149 3449610460 0 32120
7 0.000232 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 66 376 6 24 25 1024 3449610460 2758648175 0 32736
8 0.001032 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 66 318 6 24 1024 25 2758648175 3449610486 0 32120
9 0.000239 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 88 377 6 24 25 1024 3449610486 2758648201 0 32736
10 0.000878 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 82 319 6 24 1024 25 2758648201 3449610534 0 32120
11 0.000247 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 89 378 6 24 25 1024 3449610534 2758648243 0 32736
12 0.00096 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 79 320 6 24 1024 25 2758648243 3449610583 0 32120
13 0.000235 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 78 379 6 24 25 1024 3449610583 2758648282 0 32736
14 0.000818 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 46 321 6 24 1024 25 2758648282 3449610621 0 32120
15 0.00052 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 90 380 6 24 25 1024 3449610621 2758648288 0 32736
16 0.002412 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 1004 322 6 24 1024 25 2758648288 3449610671 0 32120
17 0.000851 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 59 381 6 24 25 1024 3449610671 2758649252 0 32736
18 0.000748 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 46 323 6 24 1024 25 2758649252 3449610690 0 32120
19 0.000259 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 64 382 6 24 25 1024 3449610690 2758649258 0 32736
20 0.000588 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 40 383 6 17 25 1024 3449610714 2758649258 0 32736
21 0.000438 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 40 324 6 16 1024 25 2758649258 3449610715 0 32120
22 0.001677 16 123 56 70 50 192 79 163 88 35 2048 172 16 112 194 196 37 75 158 40 325 6 17 1024 25 2758649258 3449610715 0 32120
23 0.000194 192 79 163 88 35 16 123 56 70 50 2048 196 37 75 158 172 16 112 194 40 384 6 16 25 1024 3449610715 2758649259 0 32735
24 33.588621 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 44 337 6 2 1025 79 3698898158 0 0 512
25 0.000334 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 44 501 6 18 79 1025 543102587 3698898159 0 32736
26 0.002755 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 40 338 6 16 1025 79 3698898159 543102588 0 32120
27 0.0002 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 45 339 6 24 1025 79 3698898159 543102588 0 32120
28 0.017274 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 40 502 6 16 79 1025 543102588 3698898164 0 32731
29 0.000684 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 42 340 6 24 1025 79 3698898164 543102588 0 32120
30 0.019294 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 40 503 6 16 79 1025 543102588 3698898166 0 32729
31 0.191384 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 260 506 6 24 79 1025 543102588 3698898166 0 32736
32 0.000125 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 40 507 6 17 79 1025 543102808 3698898166 0 32736
33 0.000794 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 40 341 6 16 1025 79 3698898166 543102809 0 31899
34 0.002729 16 123 56 70 50 192 79 163 88 35 2048 172 16 113 105 197 182 91 233 40 342 6 17 1025 79 3698898166 543102809 0 32120
35 0.000164 192 79 163 88 35 16 123 56 70 50 2048 197 182 91 233 172 16 113 105 40 508 6 16 79 1025 543102809 3698898167 0 32735
36 24.866747 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 44 374 6 2 1027 25 1945073538 0 0 512
37 0.000315 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 44 516 6 18 25 1027 3748813214 1945073539 0 32736
38 0.002799 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 40 375 6 16 1027 25 1945073539 3748813215 0 32120
39 0.024053 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 123 527 6 24 25 1027 3748813215 1945073539 0 32736
40 0.019396 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 40 380 6 16 1027 25 1945073539 3748813298 0 32120
41 0.025469 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 64 381 6 24 1027 25 1945073539 3748813298 0 32120
42 0.000329 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 66 528 6 24 25 1027 3748813298 1945073563 0 32736
43 0.000971 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 64 382 6 24 1027 25 1945073563 3748813324 0 32120
44 0.000242 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 86 529 6 24 25 1027 3748813324 1945073587 0 32736
45 0.000863 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 79 383 6 24 1027 25 1945073587 3748813370 0 32120
46 0.003624 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 86 530 6 24 25 1027 3748813370 1945073626 0 32736
47 0.000926 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 74 384 6 24 1027 25 1945073626 3748813416 0 32120
48 0.000244 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 73 531 6 24 25 1027 3748813416 1945073660 0 32736
49 0.000814 16 123 56 70 50 192 79 163 88 35 2048 172 16 114 169 135 13 216 191 46 385 6 24 1027 25 1945073660 3748813449 0 32120
50 0.000531 192 79 163 88 35 16 123 56 70 50 2048 135 13 216 191 172 16 114 169 90 532 6 24 25 1027 3748813449 1945073666 0 32736
```

B.3 Training Data Excerpt (Output)

First 50 input packets (vectors) from Monday, Week 3 in VectorFile format:

```
1294.7  1
1423.4  2
1296.13  3
1423.37  4
1296.13  5
1296.24  6
1423.6  7
1296.24  8
1423.52  9
1296.24  10
1423.6  11
1296.24  12
1423.6  13
1296.24  14
1423.55  15
1423.6  16
1296.2  17
1423.55  18
1296.24  19
1423.6  20
1296.24  21
1423.6  22
1423.47  23
1296.13  24
1296.11  25
1421.82  26
1294.85  27
1423.21  28
1296.34  29
1424.47  30
1296.34  31
1297.47  32
1424.48  33
1297.47  34
1424.48  35
1297.47  36
1424.48  37
1297.47  38
1424.48  39
1297.47  40
1424.48  41
1297.23  42
1423.74  43
1297.47  44
1424.48  45
1297.47  46
1424.48  47
1423.38  48
1296.34  49
1296.2  50
```

B.4 Testing Data Excerpt (Output)

First 50 input packets (vectors) from Monday, Week 4 in VectorFile format:

```
1421.01  1
1297.04  2
1422.44  3
1297.07  4
1422.44  5
1422.56  6
1297.32  7
1422.56  8
1297.32  9
1422.56  10
1297.32  11
1422.56  12
1297.32  13
1422.56  14
1297.32  15
1422.56  16
1297.32  17
1422.56  18
1297.32  19
1297.16  20
1422.44  21
1422.39  22
1295.49  23
1419.42  24
1293.16  25
1422.61  26
1424.07  27
1295.25  28
1424.09  29
1295.25  30
1296.31  31
1295.2  32
1422.61  33
1421.52  34
1295.26  35
1431.67  36
1306.07  37
1427.72  38
1306.22  39
1427.72  40
1427.78  41
1306.22  42
1427.78  43
1306.22  44
1427.78  45
1306.22  46
1427.78  47
1306.22  48
1427.78  49
1306.22  50
```

B.5 Lv3T MW3-60K: Week 4 Categorization Statistics

Training Week: Week 3

Day: Monday

N-HTM Network: "Timeless," 3-Level

Packets Captured From: Outside

Number of Training Vectors Used: First 60,000

Number of Categories Created: 1877

Testing Week: Week 4

Day: Monday

File: test_results.format.txt

Packets: 1167661

Categories In File: 2096

New Categories From Testing: 1017

Old Categories Recognized Testing: 1079

TOTAL Unique Categories: 2894

51.47900581359863% Old Categories

48.52099120616913% New Categories

Day: Tuesday

File: test_results.format.txt

Packets: 1218628

Categories In File: 3040

New Categories From Testing: 1509

Old Categories Recognized Testing: 1531

TOTAL Unique Categories: 3386

50.361841917037964% Old Categories

49.638158082962036% New Categories

```
     Day:  Wednesday

File: test_results.format.txt
Packets: 1224543
Categories In File: 3152
New Categories From Testing: 1579
Old Categories Recognized Testing: 1573
TOTAL Unique Categories: 3456
49.904823303222656% Old Categories
50.095176696777344% New Categories

     Day:  Thursday

File: test_results.format.txt
Packets: 1544574
Categories In File: 3374
New Categories From Testing: 1753
Old Categories Recognized Testing: 1621
TOTAL Unique Categories: 3630
48.04386496543884% Old Categories
51.95613503456116% New Categories

     Day:  Friday

File: test_results.format.txt
Packets: 1229260
Categories In File: 2768
New Categories From Testing: 1321
Old Categories Recognized Testing: 1447
TOTAL Unique Categories: 3198
52.27600932121277% Old Categories
47.72398769855499% New Categories
```

Appendix C. BackTalk Code - Urban Challenge

This appendix provides the pertinent Java v1.5 code to implement the Urban Challenge experiment under the BackTalk algorithm (also known in code as Qualiabear) implementation.

C.1 BasicNode.java

```
import java.io.PrintStream;
import java.util.LinkedList;
import java.util.List;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class BasicNode {
// NODE CONNECTION PARMS
// every node has a single parent (null in Top Nodes)
protected BasicNode parent;

// every node has 1 or more children (sensors do not have children)
protected List<BasicNode> children = new LinkedList<BasicNode>();

// DATA STRUCTURES FOR QUALIA & PATTERNS
// each node creates an invarient representation (IR) for each spatial code
// percived
// i.e. input pattern 2,3,2 might be code = 12
protected List<String> code_pattern_trackerList = new LinkedList<String>();

// during feedback, each parent is able to give children a IR, a qualia of
// what is expected
// these qualia (could) have a 1-to-many relationship within a given child
// node
// i.e parent might tell this node to expect a qualia of '3'
// the child knows that qualia '3' = code '12'. It may happen that qualia
// '4' = code '12' too
// **A node's code is 'qualia' to children**
protected Map<Integer, Integer> qualiaToCodeBook = new HashMap<Integer, Integer>();

// Percent of nodes that must agree with a prediction for it to be in
// context
private static double similarityThreshold = 66.66; // Percent

// LOCAL NODE VARIABLES
// ACCESSABLE DIRECTLY BY OTHER NODES

// currenly assigned code of input pattern
protected int actualCode = 999;

// if guessing was requierd, the code is saved here
protected int guessCode = 999;

// Provided in feedback from the parent
protected int expectedQuale = 999;

// Derived from expectedQualia
```

```
protected int expectedCode = 999;

// actual input pattern as percived by children
protected String inputPattern = "";

// if children percive new patterns, a guess pattern will have a X (don't
// care state) for that input element
protected String guessPattern = "";

// NOT USED or TESTED - meant to help with analysis
protected double predictionCertainty = 100.0;

// Denotes if this node percives a context change
protected boolean contextChange = false;

// Denotes if this node percives input as 'never before seen'
protected boolean novelInputFlag = false;

// denotes if a guess is requiered
protected boolean guess = false;

// File I/O
PrintStream write = null;

// What is passed up to parrents/recived from children
protected Message output = new Message();

// creates/stored incoming child values (their codes) which create a pattern
// order is not importaint if is is consistent
protected Set<Message> inputValues = new HashSet<Message>();

// ////////////////////////////////////////////////////////////////
// //////////////////////FEED FORWARD FUNCTIONS////////////////////
// ////////////////////////////////////////////////////////////////

protected void feedforward() {
output = new Message();

if (guess) {
output.setChildCode_AKA_aParentValue(guessCode);
} else {
output.setChildCode_AKA_aParentValue(actualCode);
}
output.setContextChange(contextChange);
output.setNovelInputFlag(novelInputFlag);
output.setPredictionCertainty(predictionCertainty);
parent.receiveChildInput(output);
}

public void receiveChildInput(Message input) {
// Add child's input to parent's knowledge pool
inputValues.add(input);
inputPattern = inputPattern + input.getChildCode_AKA_aParentValue()
+ ":";

if (input.getNovelInputFlag()) {
guessPattern = guessPattern + "X:";
} else {
guessPattern = guessPattern + input.getChildCode_AKA_aParentValue()
+ ":";
```

```
}

// If all "votes are in" then infer from pattern
if (inputValues.size() == children.size()) {
updateTrackerList();
perceiveInputs();
feedforward();
guessPattern = "";
}
}

// ////////////////////////////////////////////////////////////////
// ////////////////////////PERCEPTION FUNCTIONS//////////////////////
// ////////////////////////////////////////////////////////////////

protected boolean canBestGuess() {
if (guess) {
String[] guessElements = guessPattern.split(":");
for (int i = 0; i < code_pattern_trackerList.size() - 1; i++) {
String[] currentElements = code_pattern_trackerList.get(i)
.split(":");
int numberCorrect = 0;
for (int j = 0; j < currentElements.length; j++) {
if (guessElements[j].equals("X")
|| guessElements[j].equals(currentElements[j])) {
numberCorrect++;
}
}
if (numberCorrect == children.size()) {
guessCode = i;
guessPattern = "";
return true;
}
}
}
guessPattern = "";
return false;
}

protected void updateTrackerList() {
// if any child has novel/new input then flag as new
if (!code_pattern_trackerList.contains(inputPattern)) {
novelInputFlag = true;
// tell children that it is new and needs to be saved under new IR
code_pattern_trackerList.add(inputPattern);
actualCode = code_pattern_trackerList.size() - 1;

} else {
novelInputFlag = false;
actualCode = code_pattern_trackerList.indexOf(inputPattern);
}
inputPattern = "";
}

protected void perceiveInputs() {
contextChange = false;

double numberCorrect = 0.0;
double numberNew = 0.0;
double numberTotal = inputValues.size();
```

```
double cumulativeAverage = 0;
// Look to see if the votes support our expectations

for (Message m : inputValues) {

// count the number of children who recognize their inputs
if (!m.getContextChange()) {
numberCorrect++;
}
if (m.getNovelInputFlag()) {
numberNew++;
}
// for calculating the average correctness of the perception
cumulativeAverage = cumulativeAverage + m.getPredictionCertainty();
}
if (novelInputFlag) {
if (numberNew == 1) {
guess = true;
} else {
guess = false;
}
}

// the average correctness of the perception
predictionCertainty = (cumulativeAverage / numberTotal);
// decide if you agree with the qualia or have a suggestion
if (similarityThreshold <= ((numberCorrect / numberTotal) * 100.0)) {
contextChange = false;
} else {
contextChange = true;
}
guess = canBestGuess();
inputValues = new HashSet<Message>();
}

// ////////////////////////////////////////////////////////////////
// ////////////////////////FEEDBACK FUNCTIONS//////////////////////
// ////////////////////////////////////////////////////////////////

protected void saveQualiaContext(int newQuale) {
qualiaToCodeBook.put(newQuale, actualCode);
for (BasicNode c : children) {
c.saveQualiaContext(actualCode);
}
}

protected void removeNovelAsNoise() {
if (novelInputFlag) {
novelInputFlag = false;
code_pattern_trackerList
.remove(code_pattern_trackerList.size() - 1);
}
for (BasicNode c : children) {
c.removeNovelAsNoise();
}
}

protected void feedback(int nextExpectedQuale) {
setExpectedQuale(nextExpectedQuale);
for (BasicNode c : children) {
```

```
c.feedback(expectedCode);
}
}

protected void setExpectedQuale(int quale) {
expectedQuale = quale;
expectedCode = qualiaToCodeBook.get(quale);
}

// ////////////////////////////////////////////////////////////////
// //////////////////////SETUP/MISC FUNCTIONS//////////////////////
// ////////////////////////////////////////////////////////////////

public void setChildren(List<BasicNode> children) {
this.children = children;
for (BasicNode c : children) {
c.setParent(this);
}
}

private void setParent(BasicNode parent) {
this.parent = parent;
}

public void setWrite(PrintStream write) {
this.write = write;
for (BasicNode c : children) {
c.setWrite(write);
}
}

protected void println(String s) {
if (write != null) {
write.println(s);
}
System.out.println(s);
}

protected void print(String s) {
if (write != null) {
write.print(s);
}
System.out.print(s);
}

}
```

C.2 TopNode.java

```
import java.util.LinkedList;
import java.util.List;

public class TopNode extends BasicNode {

// tracks the temporal elements of patterns
private List<List<Integer>> sequenceSet = new LinkedList<List<Integer>>();

// number of IRs to track for a temporal sequence
```

```
private int maxSequeceMatchSize = 2;

// the sequence that is perceived, length <= maxSequenceMatchSize
private List<Integer> perceivedSequence = new LinkedList<Integer>();

// if learning a new sequence, it is saved here until learning is over
// then it is saved to the sequenceSet
private List<Integer> recordingSequence = new LinkedList<Integer>();

// if a sequence is perceived, this points to it
private List<Integer> currentPlayingSequence = null;

// the number of the sequence as numbered in the sequenceSet
private int currentPlayingSequenceNum;

// the predicted next LIST INDEX of a playing sequence
private int currentPlayingSequenceNextIndex;

// 'tolerence' for objects out of context - dynamic based on
// maxSequeceMatchSize
// if inContextCounter is 0, then a change in context HAS occurred
// if inContextCounter is >= maxSequeceMatchSize, then objects out of
// context (if so) are tolerated
private int inContextCounter = 0;

// if learning id taking place, true
private boolean recording = false;

// if a sequence should be chosen from the sequenceSet OR recordingSequence,
// true
private boolean newSequenceSearch = false;

public TopNode() {
super();
}

// modified for printing to files/screen
@Override
public void receiveChildInput(Message input) {
// Add child's input to parent's knowledge pool
inputValues.add(input);
inputPattern = inputPattern + input.getChildCode_AKA_aParentValue()
+ ":";

if (input.getNovelInputFlag()) {
guessPattern = guessPattern + "X:";
} else {
guessPattern = guessPattern + input.getChildCode_AKA_aParentValue()
+ ":";
}

// If all "votes are in" then "make the call"
if (inputValues.size() == children.size()) {
super.updateTrackerList();
super.perceiveInputs();
guessPattern = "";
updateSequences();
checkRecognition();
println("Song=" + currentPlayingSequenceNum + ":"
+ currentPlayingSequence);
```

```
println("Prediction:" + expectedCode);
println("Recording:" + recordingSequence);
for (BasicNode c : children) {
c.feedback(expectedCode);
}
}
}

// update the perceivedSequence with appropriate IR
private void updateSequences() {
// Trim the perceived sequence so that it is = maxSequeceMatchSize
if (perceivedSequence.size() == maxSequeceMatchSize) {
perceivedSequence.remove(0);
}
println("");
print("Top Node Decision: ");
if (!contextChange) {
println("noChange");
perceivedSequence.add(expectedCode);
removeNoise();
} else if (guess) {
println("addGuess");
perceivedSequence.add(guessCode);
expectedCode = guessCode;
removeNoise();
} else if (!novelInputFlag) {
println("codeRecognized");
perceivedSequence.add(actualCode);
expectedCode = actualCode;
} else {
println("newCode");
code_pattern_trackerList
.remove(code_pattern_trackerList.size() - 1);
inputPattern = "";
for (BasicNode c : children) {
inputPattern = inputPattern + c.actualCode + ":";
c.saveQualiaContext(actualCode);
}
code_pattern_trackerList.add(inputPattern);
perceivedSequence.add(actualCode);
expectedCode = actualCode;
inputPattern = "";
}

print("perceived Code Seq: " + perceivedSequence.get(0));
if (perceivedSequence.size() != 1) {
print("," + perceivedSequence.get(1));
}
println("");
if (contextChange) {
if (!(inContextCounter == 0)) {
inContextCounter--;
}
} else {
if (!(inContextCounter == maxSequeceMatchSize)) {
inContextCounter++;
}
}
setNewSequenceSearch();
}
```

```
// checks to see if a new sequence should be tried and flags accordingly
private void setNewSequenceSearch() {
if (!(currentPlayingSequence == null)) {
if (inContextCounter > 1) {
newSequenceSearch = false;
} else {
newSequenceSearch = true;
}
} else {
newSequenceSearch = true;
}
}

// checks to see if a prediction can be made from the perceivedSequence
// If not, then the previouse pridiction is repeated
private void checkRecognition() {
if (newSequenceSearch) {

if (perceivedSequence.size() == maxSequeceMatchSize) {

if (recording && recordingSequence.size() > maxSequeceMatchSize) {

if (inRecording(perceivedSequence)) {
// automatically saved and stopped
expectedCode = currentPlayingSequence
.get(currentPlayingSequenceNextIndex);
return;
}
}

if (!isRecognizedSequence(perceivedSequence)) {
if (recording) {
recordingSequence.add(perceivedSequence
.get(maxSequeceMatchSize - 1));
} else {
startRecodring();
}
currentPlayingSequence = null;
} else {
if (recordingSequence.size() > maxSequeceMatchSize) {
stopRecordingSave(true);
} else {
stopRecordingSave(false);
}
expectedCode = currentPlayingSequence
.get(currentPlayingSequenceNextIndex);
return;
}
}

// NOT A NEW SEQUENCE (PREDICTED)
} else {
currentPlayingSequenceNextIndex++;
if (currentPlayingSequence.size() == currentPlayingSequenceNextIndex) {
currentPlayingSequenceNextIndex = maxSequeceMatchSize;
expectedCode = currentPlayingSequence
.get(currentPlayingSequenceNextIndex);
} else {
expectedCode = currentPlayingSequence
```

```
.get(currentPlayingSequenceNextIndex);
}

}
}

private void startRecodring() {
recordingSequence.addAll(perceivedSequence);
recording = true;

}

private void stopRecordingSave(boolean save) {
if (save) {
sequenceSet.add(recordingSequence);
}
recordingSequence = new LinkedList<Integer>();
recording = false;
}

// checks recordingSequence for a sequence = seq
private boolean inRecording(List seq) {
List l = recordingSequence;
int length = l.size();
int start = 0;
int end = length - maxSequeceMatchSize;
for (int i = start; i < end; i++) {
if (seq.equals((l.subList(i, i + maxSequeceMatchSize)))) {
sequenceSet.add(recordingSequence);
currentPlayingSequenceNum = sequenceSet.size() - 1;
currentPlayingSequence = sequenceSet
.get(currentPlayingSequenceNum);
currentPlayingSequenceNextIndex = i + maxSequeceMatchSize;
stopRecordingSave(false);
return true;
}

}
return false;

}

// checks sequenceSet for an seqence = seq
private boolean isRecognizedSequence(List seq) {
for (List l : sequenceSet) {
int length = l.size();
int start = 0;
int end = length - maxSequeceMatchSize;

for (int i = start; i < end; i++) {
if (seq.equals(l.subList(i, i + maxSequeceMatchSize))) {
currentPlayingSequence = (LinkedList<Integer>) l;
currentPlayingSequenceNum = sequenceSet.indexOf(l);
currentPlayingSequenceNextIndex = i + maxSequeceMatchSize;
return true;
}
}
}
return false;
}
```

```
public final List<List<Integer>> getSequenceSet() {
return sequenceSet;
}

// Called if top node determins that the input is predicted or something
// seen before
// This removes unneeded codes from nodes that were deemed to be "wrong"
// either these nodes saw something as new that was simply noise
// or the top node did not percive a context change
private void removeNoise() {
removeNovelAsNoise();
for (BasicNode c : children) {
c.removeNovelAsNoise();
}

}
}
```

C.3 Sensor.java

```
public class SensorNode extends BasicNode {

Car car;

int name;

public SensorNode(int name, Car car) {
super();
this.name = name;
this.car = car;
}

// performs no perception functions
// forwards a code for the sensed value
// sets novelty of input
// sets inital context of input
public void sense(int i) {

output = new Message();
actualCode = i;
predictionCertainty = 100.0;
if (qualiaToCodeBook.containsValue(actualCode)) {
novelInputFlag = false;
} else {
novelInputFlag = true;
}

if (actualCode == expectedCode) {
contextChange = false;
} else {
contextChange = true;
}
feedforward();
}

@Override
protected void removeNovelAsNoise() {
```

```
}

// If this sensor is told where the 'O' should be, tell the car
@Override
protected void feedback(int nextExpectedQuale) {
setExpectedQuale(nextExpectedQuale);
if (Character.getNumericValue('O') == expectedCode) {
car.setNewLocation(name);
}

}

// Convert integers to ints
public void sense(char c) {
sense(Character.getNumericValue(Character.toUpperCase(c)));
}

}
```

C.4 Message.java

```
public class Message {

// HTM Theory says only IRs are passed up and down the hierarchy
// this is the IR
private String childCode_AKA_aParentValue;

// Biological means of detecting novel input (per hawkins) is simply
// simulated here
// by passing novelty as a boolean field
private Boolean novelInputFlag = false;

// The addition of changing context is not laid out in HTM Theory
// with no basis in HTM, context is simulated here as a boolean field
private Boolean contextChanged = false;

// Not used in unsupervised learning
private String passThroughValue = "";

// NOT USED or TESTED - meant to help with analysis
private double PredictionCertainty = 100.0;

public double getPredictionCertainty() {
return PredictionCertainty;
}

public void setPredictionCertainty(double predictionCertainty) {
PredictionCertainty = predictionCertainty;
}

public Message() {
}

public Boolean getContextChange() {
return contextChanged;
}
```

```
public void setContextChange(Boolean contextChanged) {
this.contextChanged = contextChanged;
}

public Boolean getNovelInputFlag() {
return novelInputFlag;
}

public void setNovelInputFlag(Boolean novelInputFlag) {
this.novelInputFlag = novelInputFlag;
}

public String getPassThroughValue() {
return passThroughValue;
}

public void setPassThroughValue(String passThroughValue) {
this.passThroughValue = passThroughValue;
}

public final String getChildCode_AKA_aParentValue() {
return childCode_AKA_aParentValue;
}

public final void setChildCode_AKA_aParentValue(Integer value) {
this.childCode_AKA_aParentValue = Integer.toString(value);
}

}
```

C.5 CreateNetwork.java

```
import java.io.PrintStream;
import java.util.LinkedList;
import java.util.List;

public class CreateNetwork {

// Setup the levels of the HTM network
// bottom level must be made of sensors
private List<BasicNode> sensorList = new LinkedList<BasicNode>();

// middle levels can be made of basic nodes
private List<BasicNode> lev1List = new LinkedList<BasicNode>();

// top level must be a single top node
private BasicNode topNode = new TopNode();

@SuppressWarnings("unused")
private PrintStream write = null;

// car is created with the network
Car car = new Car();

public CreateNetwork(PrintStream write) {
this.write = write;
```

```
this.initializeNet();
}

public CreateNetwork() {
this.initializeNet();
}

private void initializeNet() {

// these sensors must be able to contact the car
// there is one sensor for each character of the text-road segment
for (int i = 0; i < 9; i++) {
sensorList.add(new SensorNode(i, car));
}

// the middle level has 3 basic nodes

// make each mid-level node, add sensors as children, and add it to the
// level
BasicNode node = new BasicNode();
node.setChildren(sensorList.subList(0, 3));
lev1List.add(node);
// make each mid-level node, add sensors as children, and add it to the
// level
node = new BasicNode();
node.setChildren(sensorList.subList(3, 6));
lev1List.add(node);
// make each mid-level node, add sensors as children, and add it to the
// level
node = new BasicNode();
node.setChildren(sensorList.subList(6, 9));
lev1List.add(node);
// set all mid-level nodes as children of the top node
topNode.setChildren(lev1List);

}

public List<BasicNode> getSensorList() {
return sensorList;
}

public BasicNode getTopNode() {
return topNode;
}

public Car getCar() {
return car;
}

}
```

C.6 Car.java

```
import java.io.PrintStream;

public class Car {

String[] currentRoadIn;
```

```
int carLocation = 4;

String carInRoad = "";

// Cars respond over time, this little function moves a car left or right
// when told to do so by Qualiabear
public void setNewLocation(int name) {
if (carLocation > name) {
carLocation--;
}
if (carLocation < name) {
carLocation++;
}

}

// Reads what the road currently looks like so Qualiabear can tell car
// to place an 'O' over any of the road locations.
public void readRoadIn(String[] splitLine) {
this.currentRoadIn = splitLine;

}

// Place a 'O' where Qualiabear tells the car to drive next
public void writeRoadOut(PrintStream write, boolean autoPilot) {
if (autoPilot) {
carInRoad = "";
currentRoadIn[carLocation] = "O";
for (int i = 0; i < currentRoadIn.length; i++) {
carInRoad = carInRoad + currentRoadIn[i] + " ";
}

write.println(carInRoad);
}

}

}
```

C.7 GenerateRoad.java

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;
import java.util.HashMap;
import java.util.Map;
import java.util.Random;

public class GenerateRoad {
// The only possible road segments
private Map<Integer, String> roadViews = new HashMap<Integer, String>();
```

```
// The length a sequence of road will be
private static int MAX;

@SuppressWarnings("static-access")
public GenerateRoad(String path, int MAX) {

this.MAX = MAX;
setRoadViews();
}

private void setRoadViews() {
roadViews.put(0, "I I I W C G G G G");
roadViews.put(1, "I I I I W C G G G");
roadViews.put(2, "I I I I I W C G G");
roadViews.put(3, "W I I I I I W C G");
roadViews.put(4, "C W I I I I I W C");
roadViews.put(5, "G C W I I I I I W");
roadViews.put(6, "G G C W I I I I I");
roadViews.put(7, "G G G C W I I I I");
roadViews.put(8, "G G G G C W I I I");

}

public void generateRoad(File trainingFileIn, File testingFileOut) {
// Setup the random generator
Random generator = new Random();
int rand = generator.nextInt(3);
int currentLocation = 4;
String line = "";

try {
// File I/O parms
FileReader fr = new FileReader(trainingFileIn);
BufferedReader read = new BufferedReader(fr);
FileOutputStream out = new FileOutputStream(testingFileOut);
DataOutputStream dos = new DataOutputStream(out);
PrintStream write = new PrintStream(dos);

// Add the training file to the begining of the testfile
line = read.readLine();
while (line != null) {
write.println(line);
line = read.readLine();
}

// Produce a random road
int count = 0;
while (count != MAX) {
if (rand == 0 && currentLocation != 0) {
currentLocation--;
}
if (rand == 2 && currentLocation != 8) {
currentLocation++;
}
write.println(roadViews.get(currentLocation));
rand = generator.nextInt(3);
count++;
}
```

```
// close file I/O parms
write.close();
dos.close();
out.close();
read.close();
fr.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}
}
}
```

C.8 RunQualiabear.java

```
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;

public class RunQualiabear {

public static void main(String[] args) {
// Path of input/output files (must contain trainingRoad.txt)
String path = "C:\\Documents and Settings\\WilyPuma\\My Documents\\AFIT\\eclipseWorkBench\\QualiaProjects\\qualiabear\\";

// Input file of training road for the Qualiabear
File trainingFile = new File(path + "trainingRoad.txt");

// Output file for the combined training and testing road
File inFile = new File(path + "genTestingRoad.txt");

// Output file containing car location resulting from Qualiabear driving
// Overlays an 'O' on the infered location where the car should be on
// the test road
File outFile = new File(path + "genTestingRoad.results.txt");

// Output file of TopNode perception at each step
// Last lines are a 'Brain-dump' of the TopNodes sequeces
File dbFile = new File(path + "genTestingRoad.step_results.txt");

try {
// Setup File I/O
FileReader r = new FileReader(inFile);
BufferedReader read = new BufferedReader(r);

FileOutputStream out = new FileOutputStream(outFile);
DataOutputStream dos = new DataOutputStream(out);
PrintStream write = new PrintStream(dos);

FileOutputStream dbOut = new FileOutputStream(dbFile);
DataOutputStream dbDos = new DataOutputStream(dbOut);
PrintStream dbWrite = new PrintStream(dbDos);
```

```
// File I/O variables
String line;
String[] splitLine;

// Create HTM network per CreateNetwork.java file specs
// to write only to he screen use CreateNetwork() constructor
CreateNetwork HTMnet = new CreateNetwork(dbWrite);

// Set topNode pointer to the TopNode of the HTM netowrk created
TopNode topNode = (TopNode) HTMnet.getTopNode();

// Set car pointer of the car created by default with network
Car car = HTMnet.getCar();

// Set the TopNode to output perception to file (as well as screen)
// if commented input is only written to the screen
topNode.setWrite(dbWrite);

// Generate a random stretch of road (length: X text-road segments)
// saved in file: inFile with training
// from file: trainingFile at the start.
// new GenerateRoad(path,X).generateRoad(trainingFile,inFile);
new GenerateRoad(path, 4000).generateRoad(trainingFile, inFile);

// Begin reading the inFile
line = read.readLine();
while (line != null) {
splitLine = line.replaceAll(" ", ":").split(":");
// Assume that each new line of road may requier autopilot
boolean autoPilot = true;
car.readRoadIn(splitLine);
topNode.print("Current Input: ");

// Feed each input character to it's sensor node
for (int i = 0; i < splitLine.length; i++) {
topNode.print(splitLine[i]);
if (splitLine[i].equals("O")) {
// if ever a 'O' is in the input stream, it is
// considered training
// if training, disable autopilot and do not let
// Qualiabear drive
autoPilot = false;
}
((SensorNode) HTMnet.getSensorList().get(i))
.sense(splitLine[i].toCharArray()[0]);
}
// If Qualiabear is driving (autopilot - true) car will place an
// 'O' (move the car) as
// directed by the Qualiabear predictions.
car.writeRoadOut(write, autoPilot);
topNode.println("");
line = read.readLine();
}

// Add a 'Brain-dump' of the TopNodes learned invarrient
// representaion sequences
topNode.println("Number of sequences learned: "
+ topNode.getSequenceSet().size());
for (int i = 0; i < topNode.getSequenceSet().size(); i++) {
```

```
topNode.println("Seq#:" + +i + " = "
+ topNode.getSequenceSet().get(i));
}
// close file i/o parms before exiting
write.close();
dos.close();
out.close();
dbWrite.close();
dbDos.close();
dbOut.close();
read.close();
r.close();
} catch (FileNotFoundException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
}

}

}
```

Appendix D. Urban Challenge - Data

This appendix provides the training data and samples of corresponding testing and result data used in the Urban Challenge experiment. Included also are pre and post testing "Brain-dumps" of the Top Node's invariant representations.

D.1 Training Data (Input)

Below are the standard 31 vectors created to train BackTalk on the basics of driving:

```
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
W I I O I I W C G
I I O I I W C G G
I O I I W C G G G
O I I W C G G G G
I O I I W C G G G
I I O I I W C G G
W I I O I I W C G
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
G C W I I O I I W
G G C W I I O I I
G G G C W I I O I
```

```
G G G G C W I I O
G G G C W I I O I
G G C W I I O I I
G C W I I O I I W
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
C W I I O I I W C
```

D.2 Post Training Brain-dump Data (Output)

Below are the the sequences of invariant representations learned by the Top Node of BackTalk after training:

```
Number of sequences learned: 3
Seq#:0 = [0, 0, 0]
Seq#:1 = [0, 1, 2, 3, 4, 3, 2, 1, 0]
Seq#:2 = [0, 5, 6, 7, 8, 7, 6, 5, 0]
```

D.3 Testing Data (Input)

The first 100 vectors of a randomly generated, 500-vector testing file are as sent to the Sensor Nodes are presented below:

```
G C W I I I I I W
C W I I I I I W C
G C W I I I I I W
G G C W I I I I I
G C W I I I I I W
G C W I I I I I W
C W I I I I I W C
W I I I I I W C G
I I I I I W C G G
W I I I I I W C G
W I I I I I W C G
C W I I I I I W C
G C W I I I I I W
G G C W I I I I I
G G C W I I I I I
G G G C W I I I I
G G G G C W I I I
G G G C W I I I I
G G G C W I I I I
G G G G C W I I I
G G G G C W I I I
G G G C W I I I I
G G G G C W I I I
G G G C W I I I I
G G G G C W I I I
G G G C W I I I I
```

G G C W I I I I I

G G G C W I I I I

G G G C W I I I I

G G C W I I I I I

G G C W I I I I I

G G C W I I I I I

G G C W I I I I I

G G C W I I I I I

G C W I I I I I W

G G C W I I I I I

G C W I I I I I W

G C W I I I I I W

G C W I I I I I W

C W I I I I I W C

W I I I I I W C G

C W I I I I I W C

W I I I I I W C G

I I I I I W C G G

W I I I I I W C G

W I I I I I W C G

W I I I I I W C G

I I I I I W C G G

I I I I I W C G G

I I I I I W C G G

I I I I W C G G G

I I I I I W C G G

W I I I I I W C G

I I I I I W C G G

W I I I I I W C G

I I I I I W C G G

W I I I I I W C G

C W I I I I I W C

C W I I I I I W C

G C W I I I I I W

G C W I I I I I W

C W I I I I I W C

G C W I I I I I W

G G C W I I I I I

G G C W I I I I I

G C W I I I I I W

G C W I I I I I W

G C W I I I I I W

G C W I I I I I W

G C W I I I I I W

G G C W I I I I I

G G C W I I I I I

G C W I I I I I W

G C W I I I I I W

C W I I I I I W C

C W I I I I I W C

G C W I I I I I W

G C W I I I I I W

G G C W I I I I I

G G C W I I I I I

G G G C W I I I I

G G C W I I I I I

G G C W I I I I I

G G C W I I I I I

G C W I I I I I W

C W I I I I I W C

C W I I I I I W C

C W I I I I I W C

W I I I I I W C G

I I I I I W C G G

W I I I I I W C G

I I I I I W C G G

W I I I I I W C G

W I I I I I W C G

C W I I I I I W C

C W I I I I I W C

G C W I I I I I W

C W I I I I I W C

G C W I I I I I W

D.4 Results Data (Output)

The first 100 vectors of the previous testing file are overlaid with the representation 'O' (for BackTalk's driving) as output by the Effector Nodes to the simulated car below:

```
G C W I I O I I W
C W I I O I I W C
G C W I I O I I W
G G C W I I O I I
G C W I I O I I W
G C W I I O I I W
C W I I O I I W C
W I I O I I W C G
I I O I I W C G G
W I I O I I W C G
W I I O I I W C G
C W I I O I I W C
G C W I I O I I W
G G C W I I O I I
G G C W I I I O I
G G G C W I I O I
G G G G C W O I I
G G G C W O I I I
G G G C W I O I I
G G G G C W I O I
G G G G C W O I I
G G G G C O I I I
G G G C W I O I I
G G G G C W I O I
G G G C W I O I I
G G G G C O I I I
```

G G G C W I O I I

G G C W I I I O I

G G G C W I O I I

G G G C W O I I I

G G C W I I O I I

G G C W I I O I I

G G C W I I O I I

G G C W I I O I I

G G C W I I O I I

G C W I I O I I W

G G C W I I O I I

G C W I I O I I W

G C W I O I I I W

G C W I O I I I W

C W I I O I I W C

W I I O I I W C G

C W I I O I I W C

W I I O I I W C G

I I O I I W C G G

W I I O I I W C G

W I I I O I W C G

W I I I O I W C G

I I I O I W C G G

I I O I I W C G G

I O I I I W C G G

I I O I W C G G G

I I I O I W C G G

W I I I O I W C G

I I I O I W C G G

W I I I O I W C G

I I I O I W C G G

W I I I O I W C G

C W I I O I I W C

C W I I O I I W C

G C W I I O I I W

G C W I O I I I W

C W I I O I I W C

G C W I I O I I W

G G C W I I O I I

G G C W I I I O I

G C W I I I O I W

G C W I I O I I W

G C W I O I I I W

G C W I O I I I W

G C W I O I I I W

G G C W I O I I I

G G C W I I O I I

G C W I I O I I W

G C W I O I I I W

C W I I O I I W C

C W I I O I I W C

G C W I I O I I W

G C W I O I I I W

G G C W I O I I I

G G C W I I O I I

G G G C W I I O I

G G C W I I O I I

G G C W I O I I I

G G C W I I O I I

G C W I I O I I W

C W I I O I I W C

C W I I O I I W C

```
C W I I O I I W C
W I I O I I W C G
I I O I I W C G G
W I I O I I W C G
I I O I I W C G G
W I I O I I W C G
W I I I O I W C G
C W I I O I I W C
C W I I O I I W C
G C W I I O I I W
C W I I O I I W C
G C W I I O I I W
```

D.5 Post Testing Brain-dump Data (Output)

The output below represents the sequences of invariant representations learned by the Top Node of Qualibaer after testing:

```
Number of sequences learned: 13
Seq#:0 = [0, 0, 0]
Seq#:1 = [0, 1, 2, 3, 4, 3, 2, 1, 0]
Seq#:2 = [0, 5, 6, 7, 8, 7, 6, 5, 0]
Seq#:3 = [5, 5, 5, 0]
Seq#:4 = [1, 1, 0]
Seq#:5 = [6, 6, 6]
Seq#:6 = [8, 5, 0]
Seq#:7 = [2, 4, 4, 4]
Seq#:8 = [4, 1, 0]
Seq#:9 = [2, 2, 2]
Seq#:10 = [6, 8, 8, 8]
Seq#:11 = [7, 5, 0]
Seq#:12 = [3, 1, 0]
```

Bibliography

1. Ahmad, S. "A Technical Overview of NuPIC", 2007. URL `http://www.numenta.com/for-developers/education/NuPIC_Technical_Overview.pdf`.

2. Bishop, M. and M.A. Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, 2003.

3. Calvin, W.H. "The Emergence of Intelligence". *Scientific America, Inc.*, 1998.

4. Carriero, N. and D. Gelernter. "A computational model of everything". *Communications of the ACM*, 44(11):77–81, 2001.

5. Fine, S., Y. Singer, and N. Tishby. "The Hierarchical Hidden Markov Model: Analysis and Applications". *Machine Learning*, 32(1):41–62, 1998.

6. Franz, Maj. T.P., Maj. M.F. Durkin, Maj. P.D. Williams, Maj. (Ret) R.A. Raines, and Lt Col(Ret) R.F. Mills. "Defining Information Operations Forces: What Do We Need?" *Air & Space Power Journal*, 2007. URL `http://www.airpower.maxwell.af.mil/airchronicles/apj/apj07/sum07/franz.html`. Information extracted is unclassified.

7. George, Dileep and Numeta Inc. Bobby Jaros. "The HTM Learning Algorithms. numenta.[en línea]". *Technical White Paper, Numeta Inc*, 2007. URL `http://www.numenta.com/for-developers/education/Numenta_HTM_Learning_Algos.pdf`.

8. Hare, F. "Five Myths Of Cyberspace And Cyberpower". *SIGNAL-FALLS CHURCH VIRGINIA THEN FAIRFAX–*, 61(10):89, 2007.

9. Hawkins, J. "Why Can't a Computer be more Like a Brain?" *Spectrum, IEEE*, 44(4):21–26, 2007.

10. Hawkins, J., D. George, and N. Inc. "Hierarchical Temporal Memory. Concepts, Theory and Terminology. numenta.[en línea]". *Technical White Paper, Numeta Inc*, 2007. URL `http://www.numenta.com/ Numenta_HTM_Concepts.pdf`.

11. Hawkins, Jeff and Sandra Blakeslee. *On Intelligence*. Times Books, 2004. ISBN 0805074562.

12. Hecht-Nielsen, R., HNC Inc, and CA San Diego. "Theory of the backpropagation neural network". *Neural Networks, 1989. IJCNN., International Joint Conference on*, 593–605, 1989.

13. Heckerman, D., D. Geiger, and D.M. Chickering. "Learning Bayesian networks: The combination of knowledge and statistical data". *Machine Learning*, 20(3):197–243, 1995.

14. Inc, Numenta. "Official Website", 2007. URL `http://www.numenta.com`.

15. Inc., Numeta. "Numenta Platform for Intelligent Computing Programmers Guide". *Technical NuPIC Programming Guide, Numeta Inc*, 2007. URL `http://www.numenta.com/for-developers/education/ nupic_prog_guide.pdf`.

16. Inc., Numeta. "Problems that Fit HTMs. numenta.[en línea]". *Technical White Paper, Numeta Inc*, 2007. URL `http://www.numenta.com/for-developers/education/ProblemsThatFitHTMs.pdf`.

17. Inc., Wikimedia Foundation. "Wikipedia: The Free Encyclopedia". Encyclopedia on-line, 2007. URL `http://en.wikipedia.org`.

18. Jang, J.S.R. and C.T. Sun. "Neuro-fuzzy modeling and control". *Proceedings of the IEEE*, 83(3):378–406, 1995.

19. Jensen, F.V. *Bayesian Networks and Decision Graphs*. Springer, 2001.

20. Kitchin, R.M. "Towards geographies of cyberspace". *Progress in Human Geography*, 22(3):385–406, 1998.

21. Lamont, G.B. "Lecture Notes/Handouts/Example Reports", 2007. CSCE686.

22. Li, J., RM Gray, and RA Olshen. "Multiresolution image classification by hierarchical modeling with two-dimensional hidden Markov models". *Information Theory, IEEE Transactions on*, 46(5):1826–1841, 2000.

23. Mahoney, M. and P.K. Chan. "PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic". *Florida Institute of Technology Technical Report CS-2001-04*, 2001.

24. Mahoney, M.V. and P.K. Chan. "An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection". *RAID 2003*, 220–237, 2003.

25. Merriam-Webster. "Merriam-Webster Online Dictionary". Dictionary on-line, 2007. URL `http://www.merriam-webster.com/`.

26. Mills, R. "CSCE 525 - Intro to Information Warfare Course Slides". CD-ROM, 2007.

27. Neisser, U., G. Boodoo, T. Bouchard, A. Wade Boykin, N. Brody, S. Ceci, D. Halpern, J. Loehlin, R. Perloff, R. Sternberg, et al. "Intelligence: Knowns and Unknowns", 1996. URL `http://faculty.mwsu.edu/psychology/Laura.Spiller/4503_Tests/intelligence_knowns_and_unknowns.pdf`.

28. Phister Jr, P.W., D. Fayette, and E. Krzysiak. "CyberCraft: Concept Linking NCW Principles with the Cyber Domain in an Urban Operational Environment". *MILITARY TECHNOLOGY*, 31(9):123, 2007. URL `http://www.au.af.mil/au/awc/awcgate/afrl/cybercraft.pdf`.

29. Rabinovich, A., A. Vedaldi, C. Galleguillos, E. Wiewiora, and S. Belongie. "Objects in Context". *unknown*, 2007.

30. Ricard, M. and S. Kolitz. "The ADEPT Framework for Intelligent Autonomy". *Research and Technology Organization: Technical Publications*, RTO-EN-022, 2002.

31. Searle, J.R. "Minds, brains, and programs". *Behavioral and Brain Sciences*, 3, 1980.

32. Stevens, Capt. Michael. *Use of Trust Vectors in Support of the CyberCraft Initiative*. Master's thesis, Air Force Institute of Technology, 2007.

33. of Technology: Lincoln Laboratory, Massachusetts Institute. "Official Website", 2001. URL `http://www.ll.mit.edu/IST/ideval/`.

34. Turing, AM. “Computing Machinery and Intelligence”. *Mind*, 59(236):433–460, 1950.

35. US Air Force. *Cyberspace as a Domain In which the Air Force Flies and Fights*, 2006. URL `http://www.af.mil/library/speeches/speech.asp?id=283`.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27–03–2008	Master's Thesis	Sept 2006 — Mar 2008

4. TITLE AND SUBTITLE

Using Hierarchical Temporal Memory
for Detecting Anomalous Network Activity

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Gerod M. Bonhoff, 1Lt, USAF

5d. PROJECT NUMBER

N/A

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management(AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/08-04

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Dr. Steven K. Rogers
USAF AFRL/RY
2241 Avionics Circle
Area B Bldg 620
WPAFB, OH, 45433
DSN 674-9891 (steven.rogers@wpafb.af.mil)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT This research is motivated by the creation of intelligently autonomous cybercraft to reside in the intangible environment of cyberspace and maintain domain superiority. Specifically, this paper offers 7 challenges to the development of such a cybercraft. The focus is analysis of the claims Hierarchical Temporal Memory (HTM). In particular, HTM theory claims to facilitate intelligence in machines via accurate predictions. It further claims to be able to make accurate predictions of unusual worlds, like cyberspace. The primary objective is to provide evidence that HTM facilitates accurate predictions of unusual worlds. The second objective is to lend evidence that prediction is a good indication of intelligence. A commercial implementation of HTM theory is tested as an anomaly detection system and its ability to define network traffic (a major aspect of cyberspace) as benign or malicious is evaluated. Through the course of testing the performance of this implementation is poor. An independent algorithm is developed from a variant understanding of HTM theory. This alternate algorithm is independent of cyberspace and developed solely (but also in a contrived abstract world) to lend credibility to the use of prediction as a method of testing intelligence.

15. SUBJECT TERMS

Anomaly detection, prediction, cyberspace, hierarchical temporal memory, Bayesian network

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Lt Col (Ret) Robert F. Mills, PhD, AFIT/ENG
U	U	U	UU	144	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 4527 (robert.mills@afit.edu)