CAPTURING THE PREDICTIVE POWER OF CORTICAL LEARNING ALGORITHMS

By

Alexander C Michels

Submitted to the Department of Computer Science and Mathematics

Westminster College, New Wilmington, PA

In partial fulfillment of the requirements for Honors Research

advised by

Carolyn Cuff, Ph.D.

C. David Shaffer, Ph.D.

William Procasky, Ph.D.

March 24, 2019

Contents

Acknowledements ii

Abstract iii

List of Figures iv

List of Tables v

1 Introduction 1

2 Time Series 8

3 Cortical Learning Algorithms 11

4 Literature Review 20

5 Experiment Design 21

6 Results and Discussion 25

7 Further Work 26

References 28

Appendices 32

ACKNOWLEDGEMENTS

Type your acknowledgements here

ABSTRACT

Hierarchical Temporal Memory (HTM) is model of intelligence based on the the interactions of pyramidal neurons in the mammalian neocortex which is currently being developed by Numenta. It has stood out due to high noise tolerance and learning capacity which makes it well-suited for anomaly detection, prediction, signal processing, and sensorimotor processes. We seek to find a mathematical analogy to the predictive power of this biologically constrained theory using models from time series analysis. To do this, we statistically analyzed the predictions of HTM networks which were asked to predict outputs of autoregressive moving average (ARMA) models, varying the parameters of both the networks and the ARMA models. We hope to to find a relation between sets of HTM network parameters and ARMA model parameters to better explain cortical learning algorithms and the neocortex.

List of Figures

2.1 Examples of Time Series 8

3.1 Overview of the Human Brain 12

3.2 Column of the Neocortex 13

3.3 Pyramidal Neuron vs. Hierarchical Temporal Memory Cell 16

3.4 Spatial Pooler 17

3.5 Spatial and Temporal Poolers 18

3.6 SDR Classifier 19

5.1 Visualization of the PSO Algorithm 23

List of Tables

5.1 Hardware Specifications 22

CHANGE LOG

- Brought over Chapter 1 from Proposal with edits and added an "Biologically Inspired Computing" section to segue into HTM
- Wrote Chapter 2 - Time Series
- Brought over Sparse Distributed Representations from Proposal (Section 3.2)
- Wrote some of "Hardware and Software Specifications" (Section 5.1)
- Wrote some on Swarming (Section 5.2)

Chapter 1

Introduction

Although we are continually bombarded with sensationalist news stories proclaiming the dawn and dangers of "artificial intelligence," it is important to first define what it means to say a machine is intelligent. This is much the same way Turing began his preponderance of artificial intelligence in his 1950 *Computing Machinery and Intelligence* [31]. Turing came up with an eloquent boundary for determining the point at which we call a machine 'intelligent.' His proposal, which he called "The Imitation Game," but is now referred to as "The Turing Test," is to have an interrogator question a human and an artificial intelligence, randomly labeled X and Y, in the hopes of distinguishing between the two [31]. We reach "intelligence" when an interrogator cannot reliably distinguish between the two [31].

As mathematicians, we could also take our favorite route of defining something: we look at a collection of things that we would agree to be "intelligent" and abstract the shared properties we consider to be desirable until we have a set of properties that must be met for a system to be considered intelligent. From this approach, we could say that a system **S** is intelligent if and only if it is able to "use language,

form abstractions and concepts, solves kinds of problems now reserved for humans, and improve themselves" [24]. This is the definition from the Dartmouth AI Summer Research Project, but such a definition is obviously hard to evaluate and it would be much more difficult to form a consensus on what set of properties define intelligence than it is to get a set of properties to define other abstract mathematical concepts such as an integral domain.

It quickly becomes apparent that we also need more states than just the two Boolean "intelligent"/"not intelligent" ones to talk about intelligence in a constructive manner. Without this everything on the spectrum from a for loop with an if-else to the robots in Isaac Asimov's *I, Robot* are under the same label of "not intelligent" yet we know that the 'intelligence' exhibited in those cases are not at all comparable. We need a spectrum with many states of intelligence to constructively talk about the intelligence of a system.

John Searle's 1980 paper "Minds, Brains, and Programs" introduced the world to The Chinese room argument [30]. It supposes that artificial intelligence research is successful and produces an artificial intelligence that is capable of behaving as if it understands Chinese, then asks does the machine literally understand Chinese or it simulating the ability to understand Chinese [30]? Although this may seem like a pedantic distinction at first glance, the difference is truly important. The hypothesis that we can only ever simulate the ability to think is known as the weak AI hypothesis whereas the hypothesis that we can produce a machine capable of thought is the strong AI hypothesis [24]. These both stand in contrast to Artificial General Intelligence, which is an intelligence capable of performing any intellectual task that a human can [7].

1.1 The History of Artificial Intelligence

One would think that artificial intelligence would have its roots in the last century or two, but mankind has dreamed of and proposed machines with human-like intelligence for thousands years, dating back to at least Homer [7]. From our imagination and literature, artificial intelligence was brought into the realm of the academic by philosophers and mathematicians such as René Descartes' "mechanical man" and Gottfried Wilhelm Leibniz's mechanical reasoning devices [7]. Pascal and Leibniz both designed calculating machines capable of automated arithmetic, but proposing a calculator is far from what we think of as artificial intelligence today [7].

It was the rise of electronics from Turing, IBM, Bell Laboratories, and countless others in the mid-twentieth century started to change the question from a philosophical one to a practical one [7]. Artificial intelligence is a testament to the kind of interdisciplinary problem solving encouraged by the liberal arts, with contributions coming from the fields such as engineering, biology, psychology, game theory, communication theory, logic, philosophy, and linguistics [7]. Eventually, with advancements in computational power, operating systems, and language design, researchers were able to demonstrate impressive computational problem solving such as Arthur Samuel's 1952 checker-playing program written in assembly language and one of the first examples of evolutionary computation [7].

Newell, Shaw, and Simon's "Logic Theorist" program became the first artificial intelligence written for a computer in 1956 [18]. Through the use of heuristic algorithms, Logic Theorist was able to prove theorems and solve complex problems astonishingly well [18]. These early attempts at artificial intelligence were largely doing two things: searching and finding ways to represent and manipulate knowledge. Claude Shannon pointed this out in his 1950 "Programming a Computer for Playing Chess" in which

he produced what is now called the Shannon number, 10^{120}, which is a lower bound of the game tree complexity of Chess [24].

Artificial intelligence became formally recognized as a field of study and got its name from the 1956 Dartmouth Artificial Intelligence Conference [7]. Another product of the conference was a step forward in artificial intelligence's ability to represent and manipulate knowledge with John McCarthy's development of the first AI programming language, LISP [24]. The strides towards strong AI came crashing down with the publication of the 1969 paper "Perceptrons" which showed that single layer perceptrons were not able to properly handle linearly inseparable problems, leading to a steep decline in neural network research and "AI Winter" [24].

Research into artificial intelligence reemerged in the mid to late eighties, but this time with a more practical focus rather than searching for Searle's strong AI [24]. Algorithms developed and used for artificial intelligence found their way into camera auto-focus, anti-lock brakes, search engines, and medical diagnoses [24]. Another marked difference is the plethora of approaches such as agent systems and biologically inspired systems. Today research into artificial intelligence has largely remained in this practical realm, using neural networks, data mining, fuzzy logic and other tools to solve real-world problems while slowly marching towards an Artificial General Intelligence.

1.2 Approaches to Intelligence

Artificial intelligence, because of how broadly the word can be defined and how many fields contribute the its progress, can be hard to wrap one's head around. However, Connell and Livingston have proposed four categories for artificial intelligence

approaches which are useful for understanding the state of artificial intelligence research and the varied potential paths to Artificial General Intelligence [9].

Their first category is labeled "Silver Bullets" and describes approaches in which much of what is needed is already believed to be present, but we are missing a crucial piece that will supposedly resolve our problems and deliver us a system with intelligence [9]. Examples include 'Fancy Logic' (second-order, non-monotonic, epistemic, deontic, modal, etc), Fuzzy Logic, Deep Language, Embodiment, and Quantum Computing [9]. Disciples of this school of thought are chasing their particular "Silver Bullet," working to formalize and perfect what they believe to be the missing link.

They next describe the "Core Values" section which puts emphasis on the central organizational scheme over other computational details, believing that this macro-level structure has greater influence than the exact algorithms used [9]. Situatedness, Emotionality, Self-Awareness, and Hierarchy & Recursion are a few of these ideologies. There are strong arguments for this category, especially Hierarchy & Recursions argument that an intelligence needs to be able to abstract recursively [9].

Connell and Livingston's third category, "Emergence," looks at artificial intelligence approaches which believe they already have the essentials, but we haven't implemented the essentials on a large enough scale to get our intelligence yet [9]. For example, one might hold the position that intelligence is simply the ability to generate and search decision trees and we haven't realized an Artificial General Intelligence yet because our hardware doesn't allow us to do this effectively enough yet. Approaches in this category include Axiomatization, Commonsense, Learning, Evolution, and Integration [9].

Lastly, we visit "Emulation" which is the school of thought that says we are

better off copying intelligence than designing our own [9]. Neural simulation, neural networks, animal models, human development, sociality, and cortical learning algorithms all fit in this category [9]. The danger with this approach is abandoning theory in its sprint towards a functional copy, because if one does not understand the thing they have made, it is hard to see what it can do and where it can be improved. Another excellent point is that it can be very hard to correctly identify what needs to be copied, as Connell and Livingston note, "artificial feathers and flapping turn out not to be needed to create airplanes" [9].

1.3 Biologically Inspired Computing

Emulating natural processes–and even the human brain–is not a new idea to computer scientists. In fact, artificial neural networks that we still use today were originally proposed in a 1943 paper that introduced the first mathematical model of a neuron [12]. The 1950's and 60's saw the rise of evolutionary computing which attempts to simulate evolution usually for finding optimal solutions to problems. Computational neuroscience is the newest field to emerge from this interdisciplinary intersection of biologists and computer scientists.

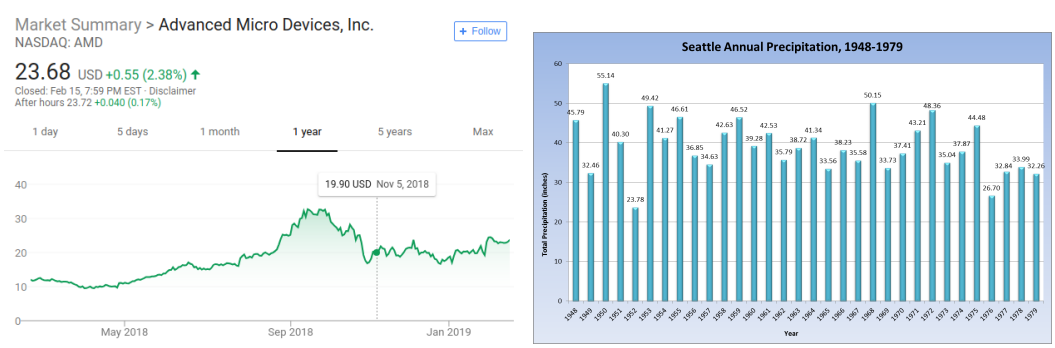
Computational neuroscience can be described as "simulating human cognition using biologically based networks of neuronlike units" [14]. In other words, the field has tasked itself with the incredibly difficult task of trying to understand cognition by breaking the brain down into algorithms and data structures which we can then implement in a computational model. So rather than trying to create a computational model that is as intelligent as the brain, they are taking the more direct approach in attempting to model the brain itself.

One such set of computational neuroscientists are the researchers at Numenta who are attempting to replicate the mammalian neocortex. Their founder, Jeff Hawkins, realized in the early 1980's that we needed to truly understand what made the brain intelligent before we created our own intelligence [23]. Unfortunately, this view was not shared by researchers at Intel and MIT where he unsuccessfully attempted to start a research lab and study computer science respectively. To support his inquiry, Jeff Hawkins decided to supplement his computer science and engineering background with some training in biophysics and set out to develop a theoretical understanding of the neocortex. In 2002, he founded the Redwood Neuroscience Institute and Hawkins is currently developing cortical learning algorithms and Hierarchical Temporal Memory at Numenta which he founded in 2005 [23].

Chapter 2

Time Series

A **time series** is a "sequence of observations taken sequentially in time" and are everywhere in today's world [6]. Anything that can be measured numerically over time can become a time series and Figure 2.1 shows two such examples, stock prices (Figure 2.1a) and levels of precipitation in Seatle, WA from 1948-1979 (Figure 2.1b). Their ubiquity in nearly every field has lead to an intense study to mathematically formalize time series to characterize and predict them.



(a) A chart of AMD stock prices

(b) Seattle precipitation, 1948-1979

Figure 2.1: Examples of Time Series [1]

Understanding time series can be an extremely complex task because there is generally a dependence on prior terms due to the sequential nature and some uncertainty to their movements. One model for representing time series is the autoregressive (or AR) model. In an autoregressive model, the series at some time t is expressed as a finite weighted sum of previous values of the series plus a random component a_t [6]. Let $z_t, z_{t-1}, z_{t-2}, ...$ denote the series at times $t, t-1, t-2, ...$ Further, let μ be the mean or level of the process and let $\tilde{z}_{t-n} = z_{t-n} - \mu$. We can then express an autoregressive process of order p by

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + \phi_2 \tilde{z}_{t-2} + ... + \phi_p \tilde{z}_{t-p} + a_t \tag{2.1}$$

where $\mu, \phi_1, \phi_2, ..., \phi_p, \sigma_a^2$ are the $p+2$ unknown parameters of the model with σ_a^2 being the variance of the white noise process a_t [6]. Using a backshift operator B such that $Bz_t = z_{t-1}$ we can define an autoregressive operator of order p by

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - ... - \phi_p^p \tag{2.2}$$

which allows us to write an autoregressive model succinctly as $\phi(B)\tilde{z}_t = a_t$ [6].

Suppose instead that the true process is regressed on the random shocks rather than the deviations from the level. Starting from our autoregressive model of order p we can recursively substitute the model's definition of $\tilde{z}_{t-n}$ until we end up with an infinite summation of innovation terms [6]. However, the moving average model provides us a much more direct approach by representing the time series as a finite number of shocks to the process. Formally we express a moving average (MA) model of order q by

[1]Image (b) courtesy of Seattle Weather Blog: http://www.seattleweatherblog.com/rain-stats/

$$\tilde{z}_t = a_t - \theta_1 a_{t-1} - \theta_2 a_{t-2} - ... - \theta_q a_{t-q} \tag{2.3}$$

just as with the autoregressive model, we have $q + 2$ unknown paramters and using the backshift operator we can define a moving average operator that allows us to express the model more succinctly yielding $\tilde{z}_t = \theta(B)a_t$ [6].

Real world processes may require a combination of these two approaches for a succinct and accurate model, resulting in a mixed Autoregressive Moving-Average (ARMA) model [6]. Such a model includes terms from both the autoregressive and moving average model resulting in:

$$\tilde{z}_t = \phi_1 \tilde{z}_{t-1} + ... + \phi_p \tilde{z}_{t-p} + a_t - \theta_1 a_{t-1} - ... - \theta_q a_{t-q} \tag{2.4}$$

This model has $p+q+2$ unknown parameters and can also be written more succinctly using our autoregressive operator and moving average operator resulting in $\phi(B)\tilde{z}_t = \theta(B)a_t$ [6].

The ubiquity of time series makes forecasting them an important and many times profitable task such as in the case of financial time series (as seen in Figure 2.1a), making them a common prediction task for machine learning and artificial intelligence systems. It allows us to test an "intelligent" systems ability to discover temporal patterns which may sometimes be fuzzy or complex: a hallmark of human intelligence.

Chapter 3

Cortical Learning Algorithms

Hierarchical Temporal Memory is a framework attempting to produce cortical learning algorithms—algorithms that mimic the processes of the human neocortex. In the human brain, the neocortex is approximately seventy-percent of the brain's volume [28]. It can be described as a "wrinkly sheet of sheet of cells, about two millimeters thick" and is responsible for the characteristics of intelligence that set humanity apart from other species [28].

Vernon Mountcastle

Overview of the neocortex

Cortical Columns and Pyramidal Neurons

3.1 Sparse Distributed Representations

Consider the binary representation of the integer 16 versus that of the integer 15. The two numbers are quite similar: they are only 1 apart, so they are as close

[1]Image courtesy of Wikipedia: https://en.wikipedia.org/wiki/Neocortex

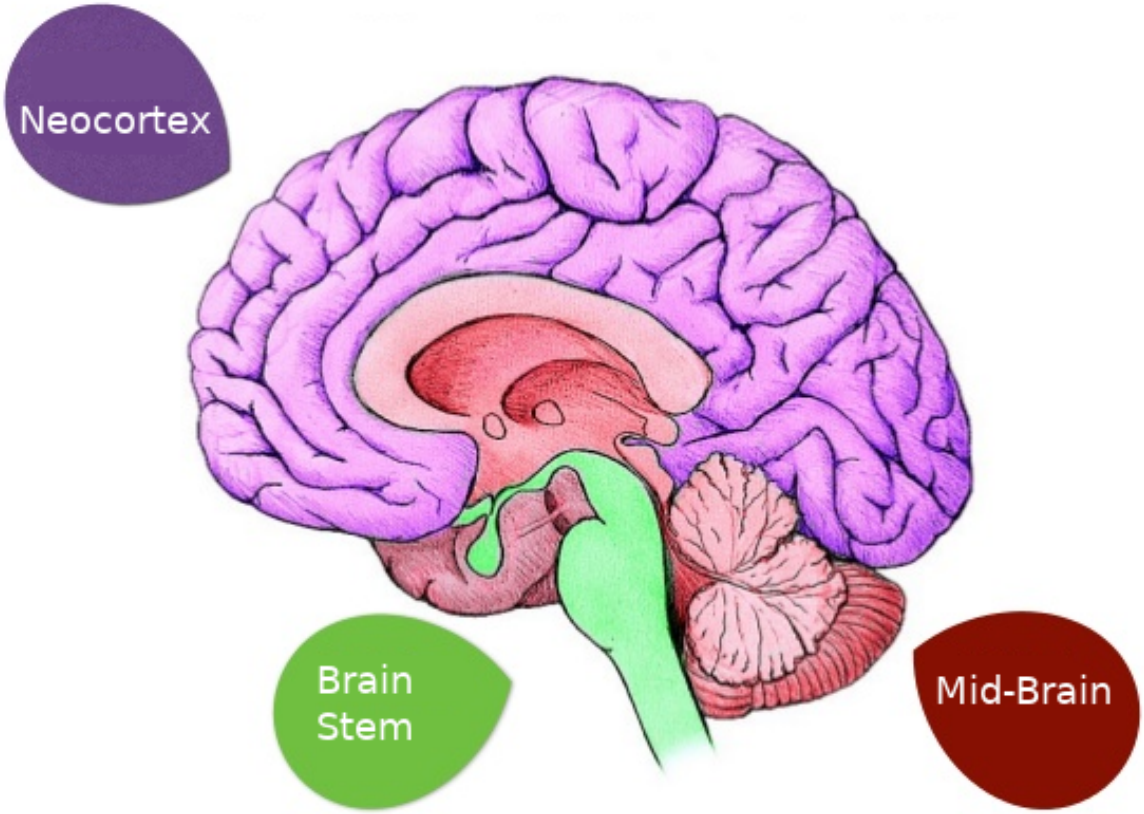


Figure 3.1: Overview of the Human Brain

as two integers can be. Yet their binary representations are [100000] and [011111] respectively. They have no shared "on" bits so despite their similarity, their binary representations reflect none at all. In fact, despite being as close as two integers can be (a Euclidean distance of 1), their binary representations have a Hamming distance, the number places in which two codewords differ, of 5, the maximum Hamming distance of two codewords of length 5 [1].

This means that our encoding does not preserve semantic similarity or a concept of distance between elements which is highly undesirable for an code because if there is some kind of error in the code we could end up decoding something meaning the

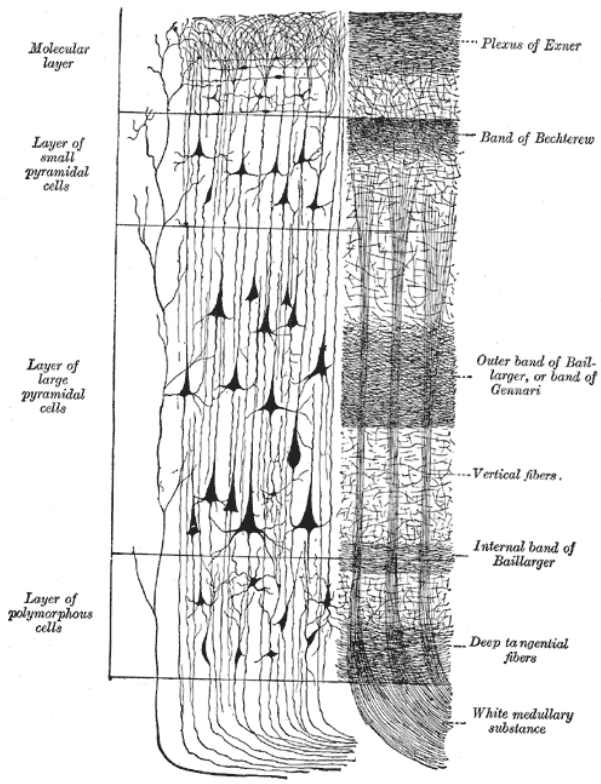


Figure 3.2: Column of the Neocortex[1]

opposite of what we were trying to convey. As an example, consider $\mathbb{Z}$ from 0 to 31 which is mapped to $GF(2)^6$ by their binary representation. The mapping of 31 is [111111] but a single error in transmission can easily lead to [011111] which would be decoded as 15. So a code Hamming distance (d_H) of one away ($\frac{1}{5}$ of the total metric) lead to an element 16 integers away ($\frac{1}{2}$ of the total metric). We would obviously like to avoid this so that errors in the transmission of our codes are either (1) correctable or (2) lead to a decoding that is as close as possible to our original element.

This is achievable by simply conveying more information in our code. In our binary representation any single error led to another valid codeword (a codeword

which decoded to an element of the input/output set) which meant that no errors could be detected or corrected. By expanding our code length, we increase the number of codewords (multiplying by the cardinality of the code alphabet for each character added) meaning that fewer errors will result in other valid codewords and can possibly be detected or corrected.

A key strategy with Sparse Distributed Representations is to encode semantic similarity, such as with our idea of distance in our motivating example. This helps us achieve our second goal because even if we increase the error-tolerance of our code, there is still some probability of an uncorrectable error and we would like that error to result in a codeword as close to the original codeword as possible. To give you a real world example imagine I am sending instructions to a aircraft and I need to tell it to turn down 7° to start its descent. Obviously, an error resulting in 8° or 6° being interpretted by the pilot are both preferable to 90°.

To achieve our goal, we employ sparse distributed representations or SDRs. Just as with traditional dense binary representations, we will represent sparse distributed representations as vectors over their code alphabet, in this case GF(2). We call them **sparse** distributed representations because these vectors typically only have a small proportion of the components as 1. We will use Numenta's notation of letting $w_{\overrightarrow{x}}$ denote the number of components in an SDR $\overrightarrow{x}$ that are 1, so $w_{\overrightarrow{x}} = \|x\|_1$ [2].

Given our definition of distance, we could say that two decodings of sparse distributed representations, a and b, are equal if and only if the $d_H(a, b) = w_a = w_b$. This would mean that both vectors would have to have the same dimensionality, same number of on bits, and all on and off bits would have to match. This definition is good for "equals," but suppose we have a single error in transmission or a single component of our distributed system fails, equality would thus fail. In order to be able preserve

the ability to subsample and thus to preserve fault tolerance, we therefore need a less stringent definition for decoding SDRs. Numenta refers to this as the *overlap*, which is

$$overlap(\overrightarrow{a}, \overrightarrow{b}) \equiv \overrightarrow{a} \cdot \overrightarrow{b} \equiv \sum_{0}^{n-1} a_i b_i \tag{3.1}$$

Thus, we say two SDRs $\overrightarrow{a}$ and $\overrightarrow{b}$ decode to the same element of the input space if and only if $overlap(a, b) \geq \theta$ where $\theta \leq w_a$ and $\theta \leq w_b$ [2]. Denote the function that determines if two sparse distributed representations decode to the same element of the input space using some θ and $overlap(\overrightarrow{a}, \overrightarrow{b})$, $match_\theta(\overrightarrow{a}, \overrightarrow{b})$ and it is a function from SDR×SDR $\longrightarrow$\{true, false\}.

Given a set of sparse distributed representations with the same dimension, $X = \{\overrightarrow{x_1}, \overrightarrow{x_2}, ..., \overrightarrow{x_n}\}$, we can union the vectors using the bitwise OR operation over the ith position of the vectors in the set to produce the ith position of $union(X)$ [2]. For example, given [0100] and [0010] the union would be [0110]. We say an SDR $\overrightarrow{y}$ is an element of the union of a set of SDRs, $\overrightarrow{X}$, if and only if $match_\theta(\overrightarrow{X}, \overrightarrow{y})$ [2].

3.2 Hierarchical Temporal Memory

Hierarchical Temporal Memory aims to implement the functions of the neocortex as faithfully as possible, and unsurprisingly uses the same atomic unit of the cell. Modeled after the pyramidal neurons of the neocortex, these cells each take their own inputs and make their own predictions which the layer combines into a cohesive prediction [21]. They have three states: active from feed-forward input, active from lateral input (predictive state), and inactive. These first two states correspond to

a short burst of action potentials in a neuron, and a slower steady burst of action potentials on a neuron respectively in our own brain.

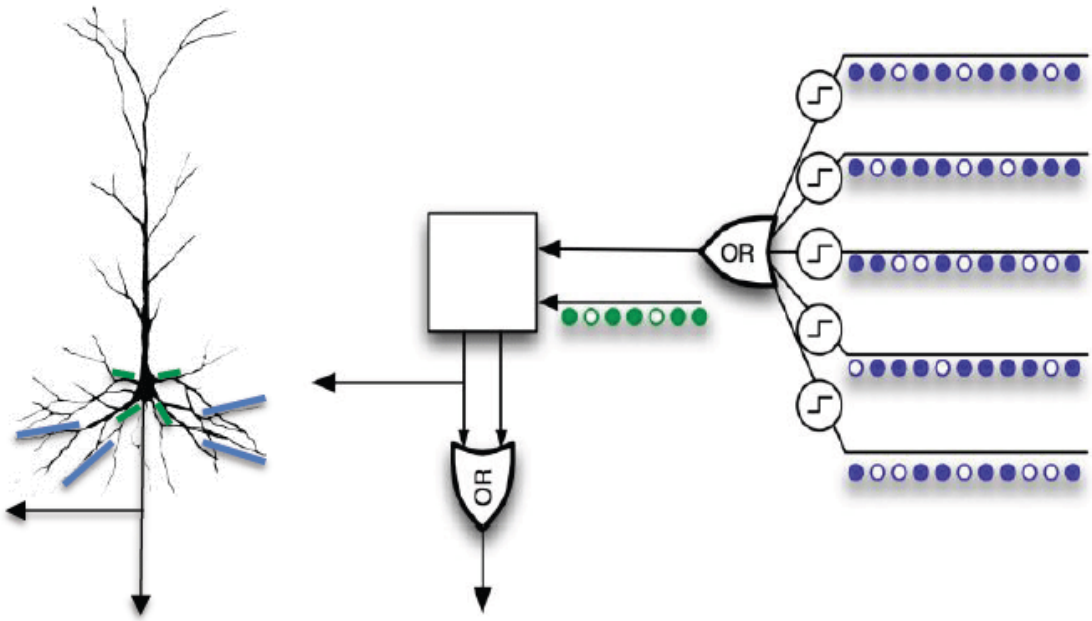


Figure 3.3: Pyramidal Neuron vs. Hierarchical Temporal Memory Cell

Just like actual pyramidal neurons, HTM cells also dendrite segments. Each cell has one proximal dendrite segment and a dozen or more distal dendrite segments [21]. Proximal dendrites accept feed-forward input and all cells in the same column respond to a similar feed-forward input due to a class of inhibitory cells. On the other hand, distal dendrites receive lateral input from nearby cells and their response is unique on a cell-by-cell basis dependent on the connected synapses on each cells distal dendrites. Figure 3.3 illustrates the similarities between a pyramidal neuron and an HTM cell.

talk about synapses

3.2.1 Encoder

3.2.2 Spatial Pooler

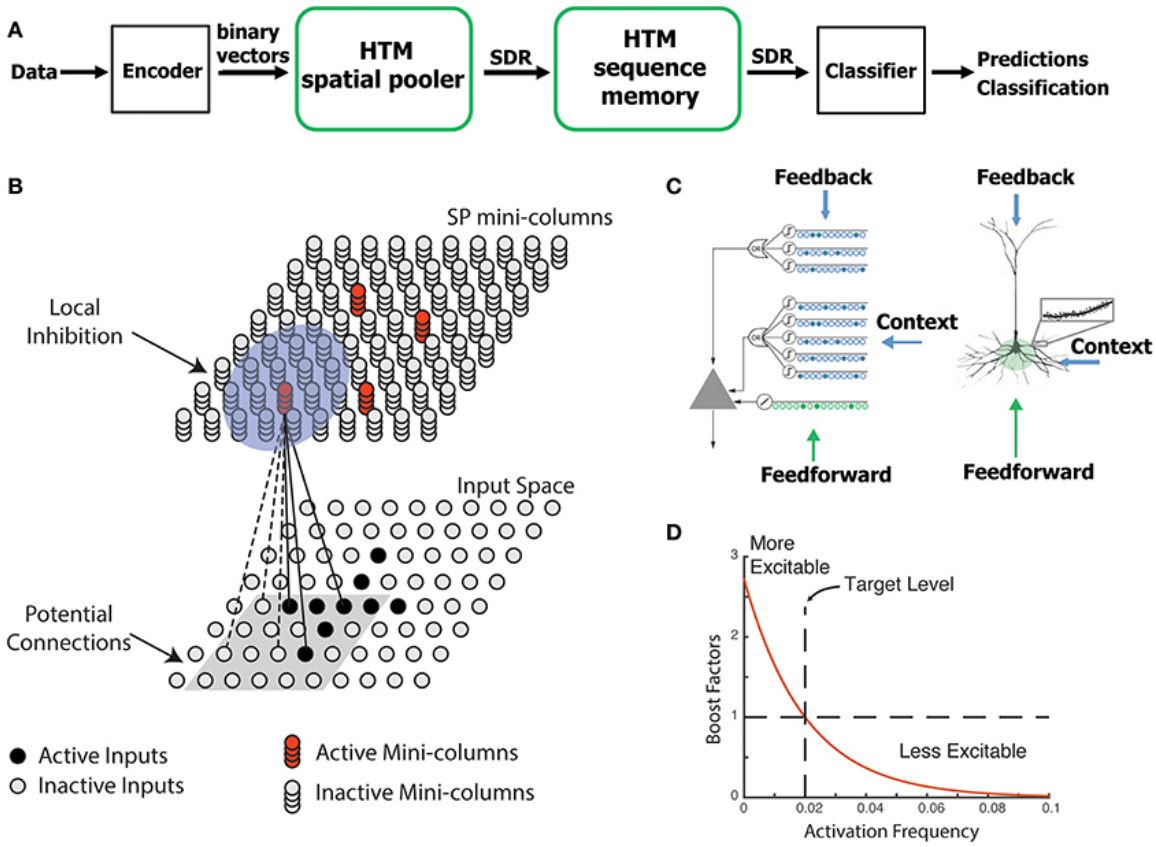


Figure 3.4: Spatial Pooler [11]

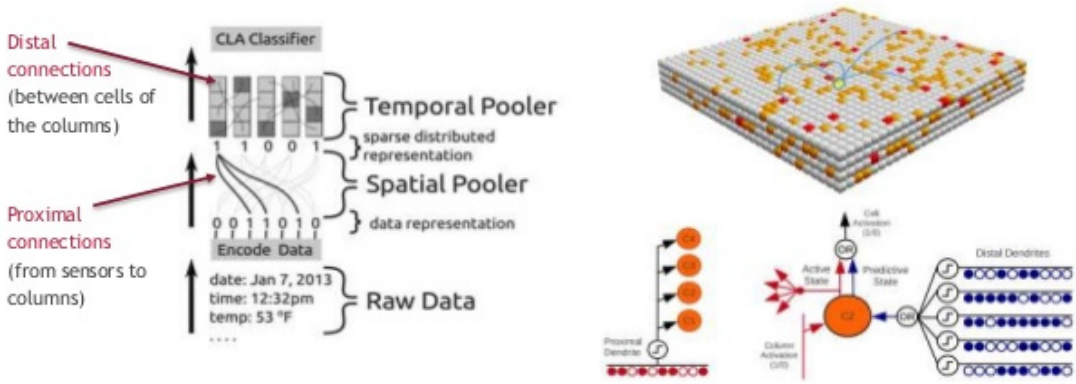


Figure 3.5: Spatial and Temporal Poolers

3.2.3 Temporal Pooler

3.2.4 Classifier

Once the system completes its processing of a time step's input, the system outputs a sparse distributed representation which may have gone through multiple layers and it is important to be able to decode the system's prediction in order for the system to be useful. Hierarchical Temporal Memory currently uses something called an SDR Classifier to decode the predictions of an HTM [13]. At its essence, an SDR Classifier is a single layer, feed forward, neural network [13].

SDR Classifiers are able to decode the information HTMs give about predictions n times steps in the future by maintaining a weight matrix [13]. The matrix's weights are adjusted to "learn" after each time step to reflect the correct weighting between the output vector at time t and the probability distribution of the input/output space at time $t-n$ [13]. This enables the matrix to reflect relationships between inputs and outputs n time steps apart. To determine the SDR Classifier's interpretation of an output at time $t+n$, the SDR Classifer takes in the HTM's output vector and uses

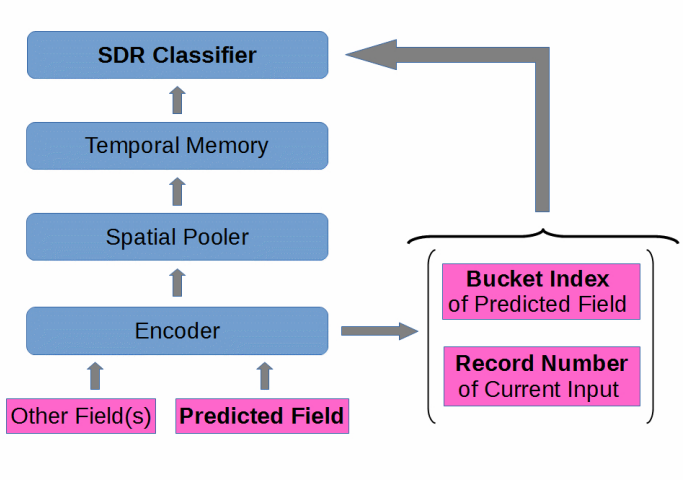


Figure 3.6: SDR Classifier

Softmax to determine the most likely decoding [13]. So for each of the k classes in the output space, the certainty that the output vector is referring to it is

$$y_j = \frac{e^{a_j}}{\sum_{i=1}^{k} e^{a_i}} \tag{3.2}$$

where a_j is the activation level of the jth element of the output space, calculated by multiplying the output vector by the jth column of the weight matrix element wise [13].

Chapter 4

Literature Review

4.1 Sequence Memory for Anomaly Detection and Prediction

Numenta's work

partners such as http://numenta.com/htm-for-stocks/ grokstream.com cortical.io

Masters theses on prediction

4.2 Spatial Data, Sensors, and Motor Response

Chapter 5

Experiment Design

5.1 Hardware and Software Specifications

I utilized the official Python (Python 2.7) implementation of Numenta's cortical learning algorithms, NuPIC[1] and it's Network API[2]. At the time of my work NuPIC was at version 1.05 and I chose to use this version. It should be noted that although much of NuPIC is in Python, many of its computationally intensive algorithms are implemented in C++ using their NuPIC.core repository[3] and it also relies on a number of external libraries such as `numpy` and `pandas`.

Due to the computationally intensive nature of my research and the limited time available to me, I also heavily relied on parallel processing to run my code in a reasonable time frame. In particular, the `multiprocessing` package in Python was indispensable to my research, allowing me to speed my code up a factor of up to 28x when running independent tests or swarming.

[1] NuPIC repository: https://github.com/numenta/nupic
[2] Network API docs: http://nupic.docs.numenta.org/stable/api/network/
[3] NuPIC.core repository: https://github.com/numenta/nupic.core

Dijkstra (Asus Vivobook)	
CPU	Intel i5-8250U, 4-Core, 1.60GHz
RAM	24 GB
OS	Linux Mint 19 Cinnamon
Galois (HP DL380 G7)	
CPU	2x Intel Xeon X5650, 6-Core, 2.70 GHz
RAM	192 GB
OS	Linux Mint 19 Cinnamon
Aslan (Type)	
CPU	2x Intel Xeon E5-2690, 10-Core, 3.00 GHz
RAM	252 GB
OS	Gentoo

Table 5.1: Hardware Specifications

On the hardware side of things, the majority of development and testing was conducted on my personal laptop, an Asus Vivobook with some small upgrades (see Table 5.1 for details). The code was then deployed to my HP DL380 G7 or [**ASLAN's type**] server which had more cores, more memory, and a higher clock speed allowing me to get faster results. For a synopsis of the hardware specifications, see Table 5.1.

Hardware and language

Numerical analysis-Epsilon comparisons

5.2 Particle Swarm Optimization

Introduced in by Russell Eberhart and James Kennedy in their 1995 paper, *A New Optimizer Using Particle Swarm Theory*, Particle Swarm Optimization has changed the way the world optimizes continuous nonlinear functions [5]. Swarming is fast and effective for optimization over complex multidimensional search spaces and lends itself well to parallelization due to its multi-agent approach, making it an excellent choice for optimizing parameters in neural networks and is implemented by Numenta for finding optimal parameters.

Usually abbreviated to PSO or *swarming*, it draws inspiration from bird flocks and schools of fish [14]. The concept is quite simple: it simulates particles on parameter space and at each time-step every particle evaluates fitness as a function of its location in parameter space, then moves towards some linear combination of its personal best and the overall best score among the particles using weights representing a cognitive constant or individuality (denoted c) and a social constant (denoted s).

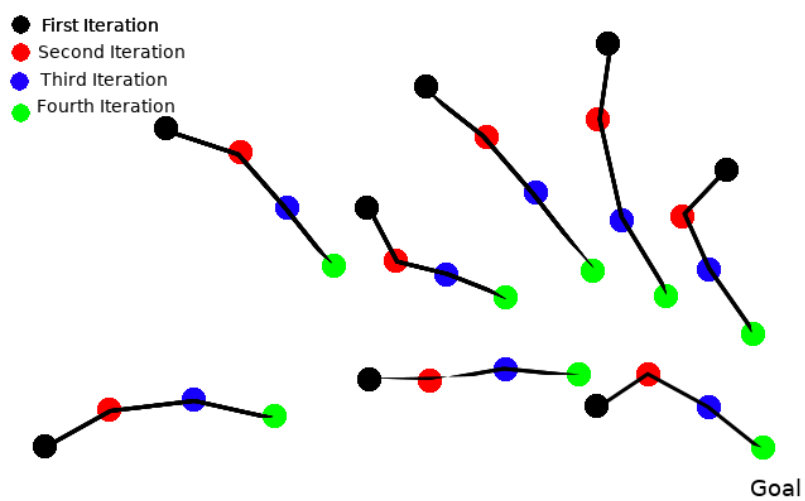


Figure 5.1: Visualization of the PSO Algorithm

The algorithm begins by initializing all particles with a random coordinate in the parameter space and a random velocity. Then at each time step, it calculates its fitness based on the function being optimized and the parameters associated with its position in parameter space. If this fitness is the best the particle has achieved, it is recorded as its personal best, denoted `pbest` and the particle with the best fitness is chosen as global best, denoted `gbest`. We then update the velocity and particle position of each particle in our swarm using the equations described in Equation (5.1) where v_t is the velocity of a particle at time step t, p_t is the position of a particle at time step t, and β_1, β_2 are uniformly distributed random variables with $\max(\beta)$ being a parameter of the algorithm [5].

$$\begin{aligned} v_{t+1} &= v_t + c \times \beta_1 \times (\text{pbest} - p_t) + s \times \beta_2 \times (\text{gbest} - p_t), \\ p_{t+1} &= p_t + v_{t+1}. \end{aligned} \tag{5.1}$$

5.3 Time Series Prediction and Fitting

Used Statsmodels to get Bayes Information Criterion for time series –¿ order (uses full exact MLE estimate)

fed order to fit ARMA model (exact maximum likelihood via Kalman filter) –¿ get lag poly

Wrote wrapper and swarming utility

LOTS of swarming

Chapter 6

Results and Discussion

Autoregressive			**Moving-Average**		
	μ	σ	μ	σ	
AR Lag 1	0.00349	0.07462	0.00276	0.05784	MA Lag 1
AR Lag 2	-0.00185	0.03404	-0.00147	0.04248	MA Lag 2
AR Lag 3	0.00134	0.02895	-0.00049	0.02883	MA Lag 3
AR Lag 4	-0.00039	0.03909	0.00192	0.03478	MA Lag 4
AR Lag 5	0.00093	0.02925	-0.00128	0.03096	MA Lag 5
AR Lag 6	0.00387	0.03895	-0.00175	0.03532	MA Lag 6
AR Lag 7	0.00042	0.01602	-0.00016	0.01331	MA Lag 7
AR Lag 8	.00008	0.00314	-0.0001	0.00788	MA Lag 8
MA Lag 1	-0.00461	0.06797	-0.00193	0.04801	AR Lag 1
MA Lag 2	0.00085	0.01665	0.00142	0.03004	AR Lag 2

Chapter 7

Further Work

Bibliography

[1] C. Adams and R. Franzosa, *Introduction to Topology: Pure and Applied*, Pearson Prentice Hall, Upper Saddle River, NY, 2008.

[2] S. Ahmad and J. Hawkins, *Properties of sparse distributed representations and their application to hierarchical temporal memory*, arXiv preprint arXiv:1503.07469, (2015).

[3] S. Ahmad and J. Hawkins, *How do neurons operate on sparse distributed representations? A mathematical theory of sparsity, neurons and active dendrites*, ArXiv e-prints, (2016).

[4] F. Åslin, *Evaluation of hierarchical temporal memory in algorithmic trading*, Master's thesis, Institutionen för Datavetenskap.

[5] A. Banks, J. Vincent, and C. Anyakoha, *A review of particle swarm optimization. part i: background and development*, Natural Computing, 6 (2007), pp. 467–484.

[6] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: Forecasting and Control*, John Wiley & Sons, Inc., Hoboken, N.J., 5 ed., 2016.

[7] B. G. Buchanan, *A (very) brief history of artificial intelligence*, AI Magazine, 26 (2005), pp. 53–60.

[8] F. Byrne, *Random distributed scalar encoder*. http://fergalbyrne.github.io/rdse.html, 7 2014.

[9] J. Connell and K. Livingston, *Four paths to ai*, Frontiers in artificial intelligence and applications, 171 (2008), p. 394.

[10] Y. Cui, S. Ahmad, and J. Hawkins, *Continuous online sequence learning with an unsupervised neural network model*, Neural Computation, 28 (2016), pp. 2474–2504. PMID: 27626963.

[11] ——, *The htm spatial pooler—a neocortical algorithm for online sparse distributed coding*, Frontiers in Computational Neuroscience, 11 (2017), p. 111.

[12] L. N. de Castro, *Fundamentals of natural computing: an overview*, Physics of Life Reviews, 4 (2007), pp. 1 – 36.

[13] A. Dillon, *Sdr classifier*. http://hopding.com/sdr-classifier, 2016. Online; accessed 9-April-2018.

[14] R. Eberhart and J. Kennedy, *A new optimizer using particle swarm theory*, in MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, IEEE.

[15] P. Gabrielsson, R. König, and U. Johansson, *Evolving hierarchical temporal memory-based trading models*, in Proceedings of the 16th European Conference on Applications of Evolutionary Computation, EvoApplications'13, Berlin, Heidelberg, 2013, Springer-Verlag, pp. 213–222.

[16] M. Galetzka, *Intelligent predictions: an empirical study of the cortical learning algorithm*, Master's thesis, University of Applied Sciences Mannheim, 2014.

[17] D. George and J. Hawkins, *Towards a mathematical theory of cortical micro-circuits*, PLOS Computational Biology, 5 (2009), pp. 1–26.

[18] L. Gugerty, *Newell and simon's logic theorist: Historical background and impact on cognitive modeling*, Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 50 (2006), pp. 880–884.

[19] J. Hawkins and S. Ahmad, *Why neurons have thousands of synapses, a theory of sequence memory in neocortex*, Frontiers in Neural Circuits, 10 (2016), p. 23.

[20] J. Hawkins, S. Ahmad, and Y. Cui, *A theory of how columns in the neocortex enable learning the structure of the world*, Frontiers in Neural Circuits, 11 (2017), p. 81.

[21] J. Hawkins, S. Ahmad, and D. Dubinsky, *Hierarchical temporal memory including htm cortical learning algorithms*, (2011).

[22] J. Hawkins, S. Ahmad, S. Purdy, and A. Lavin, *Biological and machine intelligence (bami).* Initial online release 0.4, 2016.

[23] J. Hawkins and S. Blakeslee, *On Intelligence*, Times Books, New York, NY, USA, 2004.

[24] T. Jones, *Artificial Intelligence: A Systems Approach with CD*, Jones and Bartlett Publishers, Inc., USA, 1st ed., 2008.

[25] J. Munkres, *Topology*, Prentice Hall, Upper Saddle River, NJ, 2 ed., 2000.

[26] Numenta, *Advanced nupic programming*, (2008).

[27] ——, *Principles of hierarchical temporal memory (htm): Foundations of machine intelligence*, October 2014.

[28] ——, *Numenta research: Key discoveries in understanding how the brain works*, August 2018.

[29] S. Purdy, *Encoding data for HTM systems*, CoRR, abs/1602.05925 (2016).

[30] J. R. Searle, *Mind design*, MIT Press, Cambridge, MA, USA, 1985, ch. Minds, Brains, and Programs, pp. 282–307.

[31] A. M. TURING, *I.—COMPUTING MACHINERY AND INTELLIGENCE*, Mind, LIX (1950), pp. 433–460.

[32] A. Vivmond, *Utilizing the htm algorithms for weather forecasting and anomaly detection*, Master's thesis, University of Bergen, 2016.

[33] F. D. S. Webber, *Semantic folding theory and its application in semantic fingerprinting*, CoRR, abs/1511.08855 (2015).

Appendices