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.

February 17, 2019

Contents

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 currently being developed by Numenta. It has stood out from other machine learning techniques due to high noise tolerance and learning capacity making 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 error in predictions of HTM networks that we asked to predict outputs of autoregressive integrated moving average (ARIMA) models, varying the parameters of both the networks and the ARIMA models. We hope to to find a relation between sets of HTM network parameters and ARIMA model parameters to better explain the functions of each part of the neocortex and cortical learning algorithms in sequence memory.

List of Figures

List of Tables

Chapter 1

Introduction

People have dreamed of AI for thousands of years

Logic and heuristics

Computational neuroscience approach

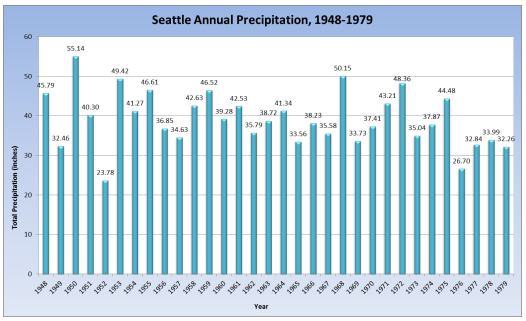
Briefly discuss Numenta and CLAs

Chapter 2

Time Series

Motivate the problem, time series are everywhere

(a) A chart of AMD stock prices



(b) Seattle precipitation, 1948-1979

Figure 2.1: Examples of Time Series [1]

Talk about ARIMA models

[1] Seattle precipitation image courtesy of Seattle Weather Blog

Chapter 3

Hierarchical Temporal Memory

3.1 The Neocortex

Vernon Mountcastle

Overview of the neocortex

Cortical Columns and Pyramidal Neurons

3.2 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 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

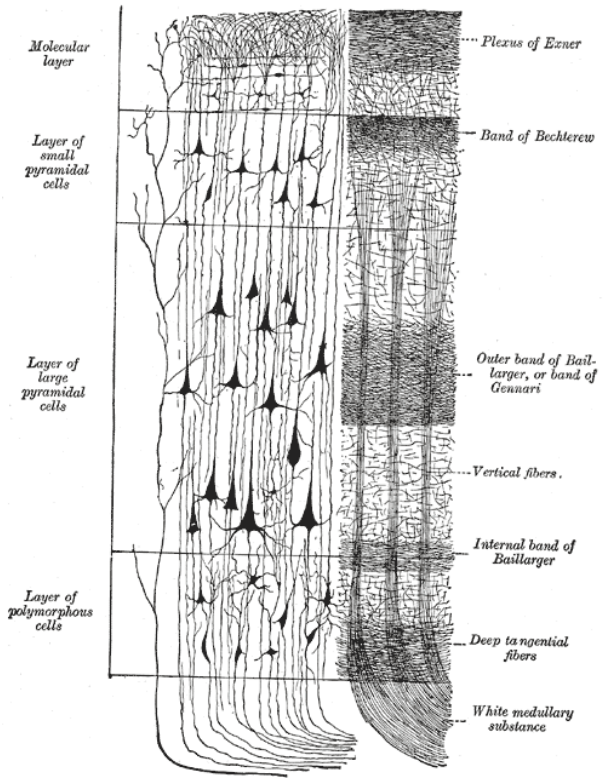


Figure 3.1: Column of the Neocortex

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 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

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

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 i^{th} position of the vectors in the set to produce the i^{th} 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].

3.3 Cortical Learning Algorithms

Sequence memory - SDRs and transitions

Variable order sequence memory - cells per minicolumn

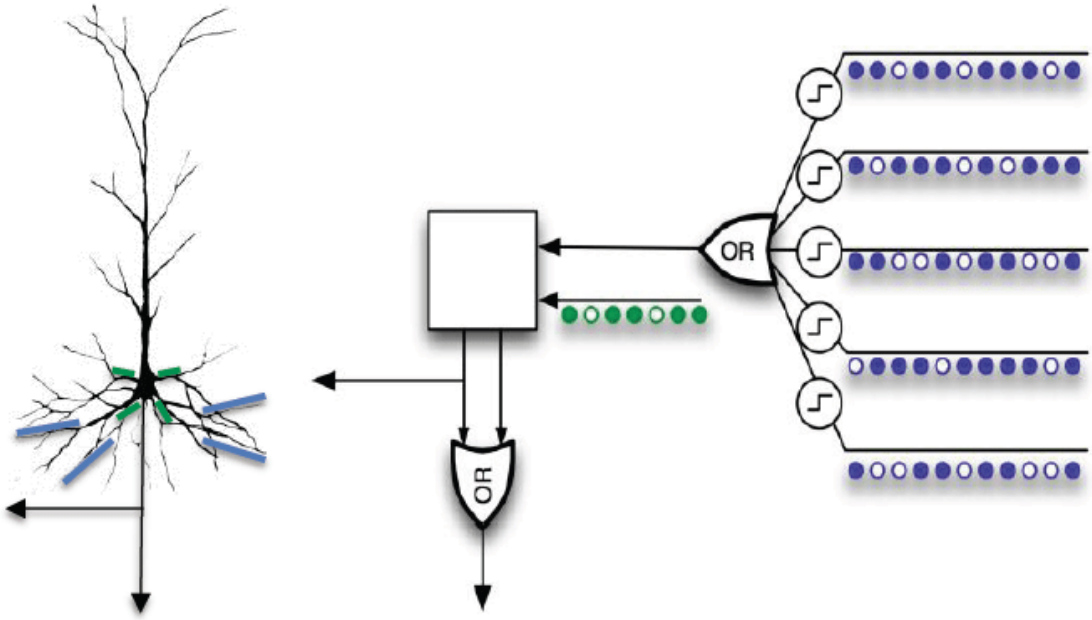


Figure 3.2: Hierarchical Temporal Memory Cell

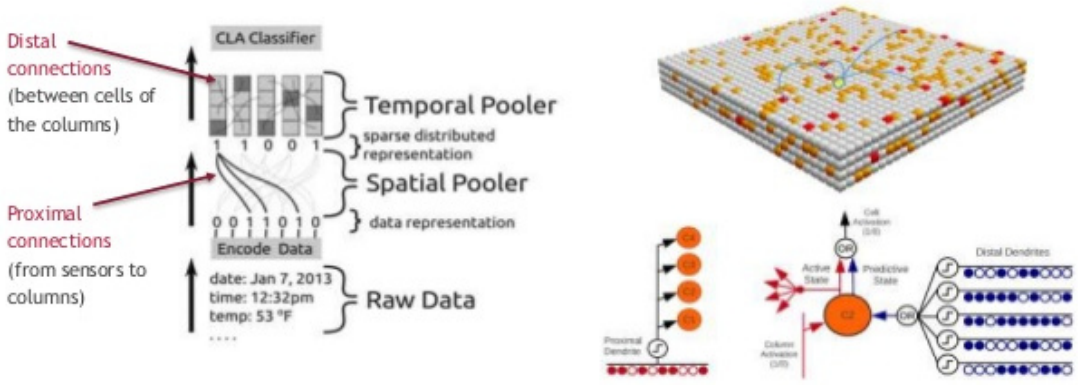


Figure 3.3: Spatial and Temporal Poolers

3.3.1 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 [10]. At its essence, an SDR Classifier is a single layer, feed forward, neural network [10].

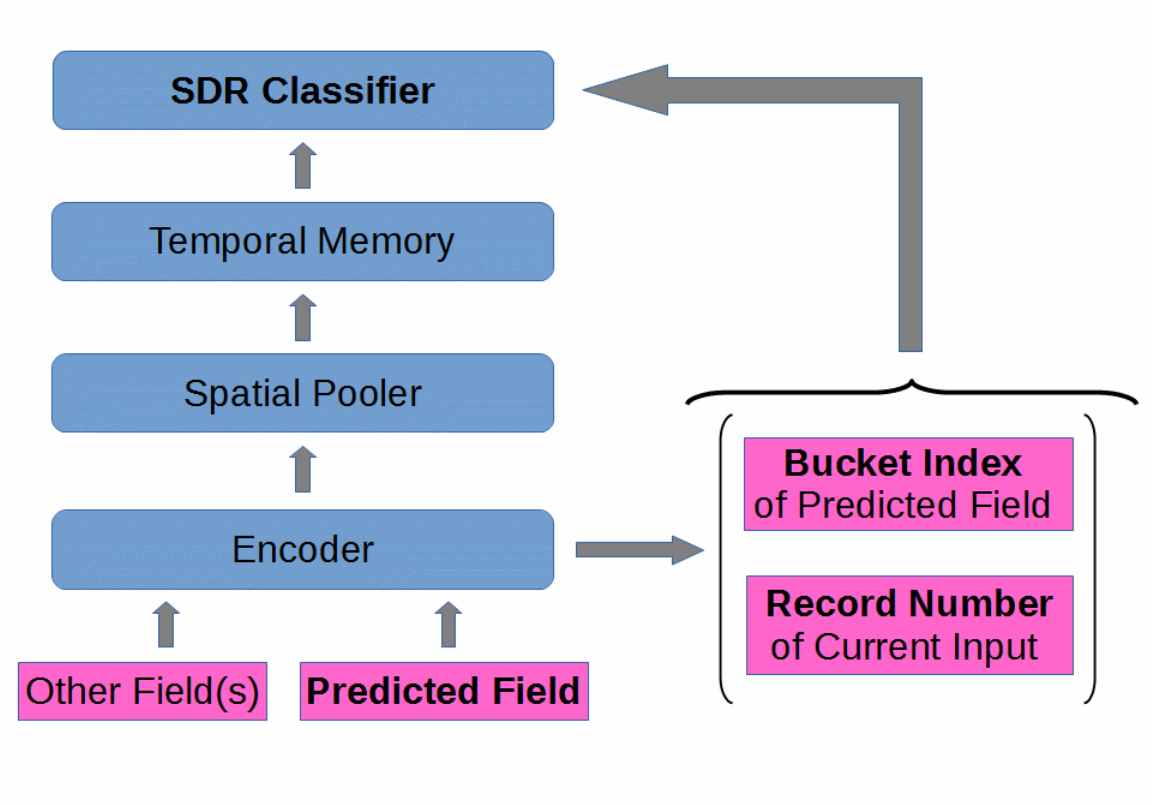


Figure 3.4: SDR Classifier

SDR Classifiers are able to decode the information HTMs give about predictions

n times steps in the future by maintaining a weight matrix [10]. 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 $t-n$ [10]. 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 Softmax to determine the most likely decoding [10]. So for each of the k classes in the output space, the certainty that the output vector is referring to it is

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

where a_j is the activation level of the j^{th} element of the output space, calculated by multiplying the output vector by the j^{th} column of the weight matrix element wise [10].

Chapter 4

Literature Review

Does anyone know if its able to do this?

It has been used in applications, look there

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

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 20× 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	Intel Xeon X5650, 6-Core, 2.70 GHz
RAM	192 GB
OS	Linux Mint 19 Cinnamon

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 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. Usually abbreviated to PSO or *swarming*, it draws inspiration from bird flocks and schools of fish [11]. The concept is quite simple: it simulates particles on parameter space and at each time-step each 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.

Wrote wrapper and swarming utility

LOTS of swarming

Chapter 6

Results and Discussion

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

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

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

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

[15] 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.

[16] 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.

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

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

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

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

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

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

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

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

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

Appendices