An Exploration of the Topological and Logical Properties of Hierarchical Temporal Memory Networks

Honors Research Paper

Alexander Michels

Mathematics and Financial Economics

Computer Science

supervised by
Dr. Carolyn Cuff
Dr. C. David Shaffer
Dr. William Procasky

August 27, 2018

Contents

List of Figures

1 Introduction

1.1 What is Intelligence?

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* [20]. 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 [20]. We reach "intelligence" when an interrogator cannot reliably distinguish between the two [20].

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" [13]. 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 [19]. 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 machines literally understand Chinese or it simulating the ability to understand Chinese [19]? 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 literally produce a machine capable of thought is the strong AI hypothesis [13]. Another useful term in use today is Artificial General Intelligence, which is an intelligence capable of performing any intellectual task that a human can [5].

1.2 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 [5]. 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 [5]. Pascal and Leibniz both designed calculating machines

capable of automated arithmetic, but proposing a calculator is far from what we think of as [5].

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 [5]. 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 [5]. Advancements in computational power, operating systems, and language design allowed computer scientists to demonstrate computational problem solving such as Arthur Samuel's 1952 groundbreaking checker-playing program written in assembly language and one of the first examples of evolutionary computation [5].

Newell, Shaw, and Simon's "Logic Theorist" program became the first artificial intelligence written for a computer in 1956 [11]. Through the use of heuristic algorithms, Logic Theorist was able to prove theorems and solve complex problems astonishingly well [11]. 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 [13].

Artificial intelligence became formally recognized as a field of study and got its name from the 1956 Dartmouth Artificial Intelligence Conference [5]. 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 [13]. 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" [13].

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 [13]. Algorithms developed and used for artificial intelligence found their way into camera auto-focus, anti-lock brakes, search engines, and medical diagnoses [13]. 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.

1.3 Approaches to Artificial 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 [7].

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 [7]. Examples include 'Fancy Logic' (second-order, non-monotonic, epistemic, deontic, modal, etc), Fuzzy Logic, Deep Language, Embodiment, and Quantum Computing [7]. 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 [7]. 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 [7].

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 [7]. 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 [7].

Lastly, we visit "Emulation" which is the school of thought that says we are better off copying intelligence than designing our own [7]. Neural simulation, neural networks, animal models, human development, sociality, and cortical learning algorithms all fit in this category [7]. 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" [7].

2 Logic, Fuzzy Logic, and Fuzzy Control Systems

3 Topological Concepts

In order to discuss *An Exploration of the Topological and Epistemic Properties of Hierarchical Temporal Memory Networks*, we will also need to briefly discuss what a topology is. A topology on a set X is a collection $\mathcal{T}$ of subsets of X having the following properties: the null set and X are elements of $\mathcal{T}$, any arbitrary union of elements of $\mathcal{T}$ is in $\mathcal{T}$, and any finite intersection of elements of $\mathcal{T}$ is in $\mathcal{T}$ [15]. When a topology $\mathcal{T}$ is specified on a set X, we call X a topological space [15]. A subset $\mathcal{U}$ of $\mathcal{T}$ is an open set of X and set $\mathcal{C}$ is closed if the compliment, $X - \mathcal{C}$, is an open set of X [1].

As arbitrary unions of elements of a topology and any finite intersections of elements of a topology are both also elements of the topology, it is generally more practical to specify a collection that is able to generate the topology than the entire topology itself. This is called a basis. If X is a set and $\mathcal{B}$ be is collection of subsets of X, then $\mathcal{B}$ is a basis of X if the following hold: (1) $\forall\, x \in X\ \exists\, B \in \mathcal{B}$ s.t. $x \in B$ and (2) if B_1 and B_2 are in $\mathcal{B}$ and $x \in B_1 \cap B_2$, then $\exists\, B_3 \in \mathcal{B}$ s.t. $x \in B_3 \subseteq B_1 \cap B_2$ [1].

4 Sparse Distributed Representations

4.1 Key Concepts

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 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 semantically 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 17° to start its descent. Obviously, 18° or 16° 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].

Sparse Distributed Representations work similar to Content Addressable Memory in which input is directly interpreted as a memory location [21]. As an example, the string "2+3" (assuming 8-bit ASCII) would be interpreted as a 256-ary number giving the memory location in an array of length 2^{24} where "5" is stored [21]. Cortical memory improves on this concept though because it has a mechanism to generate specific memory addresses for every memory cell meaning it has indirection-free semantic data storage leading to a constant amount of work to process information [21]. This also means that the ability to handle data in real time increases linearly with the number of available memory which was likely a large advantage in mammalian evolution [21].

4.2 Defining a Metric Space on SDRs

Consider the Hamming distance of two sparse distributed representations. It is a function, d_H : SDR $\times$ SDR $\longrightarrow \mathbb{R}$ with some special properties. It is the sum of the differences across the vectors which can be thought of as the boolean XOR operation and thus

$$d_H(\overrightarrow{x}, \overrightarrow{y}) \equiv \sum_{0}^{n-1} x_i + y_i$$

The Hamming distance of two SDRs is greater than or equal to zero for any two SDRs and zero if and only if the two SDRs are equal. Our d_H is also symmetric as $d_H(\overrightarrow{u}, \overrightarrow{v}) = d_H(\overrightarrow{v}, \overrightarrow{u})$ for all $\overrightarrow{u}$, $\overrightarrow{v}$. Lastly, suppose we have three n-dimensional sparse distributed representations, $\overrightarrow{x}$, $\overrightarrow{y}$, and $\overrightarrow{z}$, it must be the case that $d_H(\overrightarrow{x}, \overrightarrow{y}) + d_H(\overrightarrow{y}, \overrightarrow{z}) \geq d_H(\overrightarrow{x}, \overrightarrow{z})$. Thus we conclude that Hamming distance is a metric for n-dimensional sparse distributed representations.

Consider the set of all possible n-dimensional sparse distributed representations, $\{\overrightarrow{v} = [b_1, b_2, ..., b_n] \,|\, b_i \in \{0, 1\}, \forall\, i \in [1, n]\}$, which we will denote $\mathcal{SDR}^n$. If we combine our set $\mathcal{SDR}^n$ with our metric d_H we can construct a collection of ϵ-balls $\mathcal{B}$ where $\mathcal{B} = \{\overrightarrow{y} \mid d_H(\overrightarrow{x}, \overrightarrow{y}) < \epsilon, \epsilon > 0, \text{ and } \overrightarrow{x} \in \mathcal{SDR}^n\}$. Our collection, $\mathcal{B}$ is a basis for a topology, $\mathcal{X}(\mathcal{SDR}^n, d_H)$ on $\mathcal{SDR}^n$ called the metric topology induced by d_H.

4.3 Numenta's Quotient Topology

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] I will call the function that determines if two sparse distributed representations decode to the same element of the input space using θ and $overlap(\overrightarrow{a}, \overrightarrow{b})$, $match_\theta(\overrightarrow{a}, \overrightarrow{b})$ and it is a function from SDR$\times$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].

Hierarchical Temporal Memory makes use of the *union* function to fold the metric space into a quotient topology which glues some sets together which are semantically similar. This also enables the system to only deal with single SDRs rather than sets of SDRs because it can map any set of SDRs in the metric space to a set containing only one SDR. For example, consider three semantically similar elements of our input space {A}, {B}, and {C} which are mapped by our SDR encoding function f to [00001111], [11110000], and

[00111100] respectively. The sets {A,B} and {A,B,C} will have the same SDR if we use the *union* mechanism, [11111111], despite being {[00001111],[11110000]} and {[00001111],[11110000],[00111100]} in the metric space respectively.

This quotient topology on $\mathcal{SDR}^n$ is constructed with a partition X^* where the elements of X^* are the equivalence classes of the single vector sets of $\mathcal{SDR}^n$ where $\overrightarrow{x} \in \mathcal{SDR}^n \equiv \mathcal{U} \subseteq \mathcal{X}(\mathcal{SDR}^n, d_H)$ if and only if $d_H(\overrightarrow{x}, union(\mathcal{U})) = 0$. This means that each open set $\mathcal{U}$ in $\mathcal{X}(\mathcal{SDR}^n, d_H)$ is associated with the single vector set which is the same as $union(\mathcal{U})$. The union of X^* must be $\mathcal{X}(\mathcal{SDR}^n, d_H)$ because each open set in $\mathcal{X}(\mathcal{SDR}^n, d_H)$ can be mapped to a single n-dimensional sparse distributed representation through the *union* function and letting $\epsilon < 1$ we see that $\mathcal{X}(\mathcal{SDR}^n, d_H)$ contains a set containing each n-dimensional sparse distributed representation and only that point. Letting $p : \mathcal{X}(\mathcal{SDR}^n, d_H) \longrightarrow X^*$ be the surjective map carrying each point of $\mathcal{X}(\mathcal{SDR}^n, d_H)$ to the element of X^* containing it, we arrive at the quotient topology induced by p, X^*.

4.4 Encoding

Thus far we have discussed how sparse distributed representations form a topology defined the operations on SDRs in topological terms, but we have yet to discuss how to produce an sparse distributed representation. Sparse distributed representations are created using Semantic Folding Theory which is defined as "The process of encoding words, by using a topographical semantic space as a distributional reference frame into a sparse binary representational vector" [21].

There are some rules which we need our encoder to follow in order to preserve semantics. Let $\mathcal{A}$ be an arbitrary input space, let $d_{\mathcal{A}}$ be a metric on $\mathcal{A}$, and let f be our encoding function from $\mathcal{A}$ to $\mathcal{SDR}^n$. The output should have the same total number of bits for all inputs [18]. Formally, n is a parameter of f and $\forall x \in \mathcal{A}$, $dim(\overrightarrow{f(x)}) = n$. All output should also have the same sparsity

$$\frac{\left\|\overrightarrow{f(x)}\right\|_1}{dim(\overrightarrow{f(x)})} = \frac{\left\|\overrightarrow{f(y)}\right\|_1}{dim(\overrightarrow{f(y)})} \forall x, y \in \mathcal{A}$$

or similar sparsity as well as enough on-bits to handle noise and subsambling (20-25) [18].

Claim 1. *Formally, the notion of "semantic similarity or difference" can be represented by distance, and thus an encoding of a Sparse Distributed Representation is a mapping which attempts to preserve relative distances between elements of the input space.*

Therefore, there are a few properties we want our encoder to follow. For all $x, y \in \mathcal{A}$ if $d_{\mathcal{A}}(x,\ y) = 0$, then $d_H(\overrightarrow{f(x)},\ \overrightarrow{f(y)}) = 0$. This says that if two elements of the input space have a distance of zero, or are "semantically the same," they must map to the same vector in $\mathcal{SDR}^n$. This is just a formalization and generalization of already known concepts [18]. Additionally, because we want to preserve relative distances we want as many elements $x, y, z \in \mathcal{A}$ as possible to satisfy if $d_{\mathcal{A}}(x,\ y) > d_{\mathcal{A}}(x,\ z)$, then $d_H(\overrightarrow{f(x)},\ \overrightarrow{f(y)}) > d_H(\overrightarrow{f(x)},\ \overrightarrow{f(z)})$ and conversely, if $d_{\mathcal{A}}(x,\ y) < d_{\mathcal{A}}(x,\ z)$, then $d_H(\overrightarrow{f(x)},\ \overrightarrow{f(y)}) < d_H(\overrightarrow{f(x)},\ \overrightarrow{f(z)})$.

However, it is generally not possible to produce an encoder f capable of satisfying these conditions for all elements in $\mathcal{A}$. As an easy example, consider $\mathcal{A} = \mathbb{R}$. There does not exist an $n \in \mathbb{Z}$ such that $2^n = |\mathbb{R}|$, thus any f would rely on a mapping from a quotient space of $\mathbb{R}$ to $\mathcal{SDR}^n$. There exists two points a, b in a set identified together under the quotient topology, S, such that the Euclidean distance of a and b is greater

than or equal to the Euclidean distance of any other two points in a set identified together. By the Well Ordering Principle, either $a < b$ or $b < a$ and we will label the numbers such that $a < b$ and d(a,b) = c. $d_H(a,\ b) = 0$ because they are identified together under the quotient topology, but for some $0 > \epsilon > c$ a-ϵ is not in S because if it would cause a contradiction in the selection of a, b and $d_H(a, a-\epsilon) > 0$ by definition of a metric. Thus, we have $d(a, a-\epsilon) < d(a, b)$ and $d_H(a, a-\epsilon) > d_H(a, b)$.

It is important to note that this mapping preserves local information about distance, but generally the information about distance is less useful as the distance between elements of $\mathbb{SDR}^n$ grows. For example, suppose we have a Random Distributed Scalar Encoder with resolution 1.0 and 8 on-bits and suppose the first value it sees is 50.5. We will go with nupic which uses Python's *round*() function which rounds 0.5 up to 1.0, thus using the lower-bound topology on $\mathbb{R}$, denoted $\mathbb{R}_l$. This encoder will then have all elements of the clopen set $[50, 51)$ identified together as one SDR and the SDRs of the elements of $[49, 50)$ and $[51, 52)$ will each have a Hamming Distance of 1 from the SDRs of the elements of $[50, 51)$. With 8 on-bits, we can inductively add "buckets" until we reach $[58, 59)$ whose elements will map to an SDR that shares no on-bits with the SDR of an element of $[50, 51)$, thus they have a Hamming distance of 16, the maximum in this metric space. However, the SDR of $1000.2 \in [1000, 1001)$ also has a Hamming distance of 16 from the SDR of $[50, 51)$. So while locally, sparse distributed representations hold distance information about similarity, their information about how different they are is rather lacking.

//use Euclidean metric for fuzzy HTM??

//quotient space on $\mathbb{R}_l$ to make buckets?

//injective mapping quotient space to codes attempting to preserve distance, but transferred to a different metric

//talk about requirements to be metrizable

RDSE http://fergalbyrne.github.io/rdse.html

5 Hierarchical Temporal Memory

Hierarchical Temporal Memory is a technology that takes all the concepts we have briefly surveyed and combines them into a machine learning technology firmly rooted in the "Emulation" Path to Artificial Intelligence. It aims to capture the structure and algorithms of the neocortex in the hopes of delivering machines the intelligent though that the neocortex is thought to gift us [12]. Hierachical Temporal Memory networks (HTMs) are by definition a type of neural network, but are special because they model cells that are stacked in columns, which are organized in layers, in nodes, and in a hierarchy, following the layout of cells in the neocortex [12].

Hierarchical Temporal Memory follows what is known as a cortical learning algorithm [17]. The cortical learning algorithm can be broken down into a few top level steps: initialization and encoding, the Spatial Pooler, the Temporal Pooler, and decoding the information. Of course, in a system with more than one level of regions, there is a hierarchy of regions with children regions feeding into parent regions to interpret multiple time series together which further complicates the network [8]. Bringing all of these complex pieces together yields Hierarchical Temporal Memory, but it is important to start by looking at what makes everything work: the cells.

5.1 Cells

The cell in a Hierarchical Temporal Memory network is the most basic building block from which the structure is formed. To understand how the system works, it is vital to first comprehend the internal structures and functions of the HTM's cells. They are comprised of three main components: synapses, the proximal dendrite, and the distal dendrites [12].

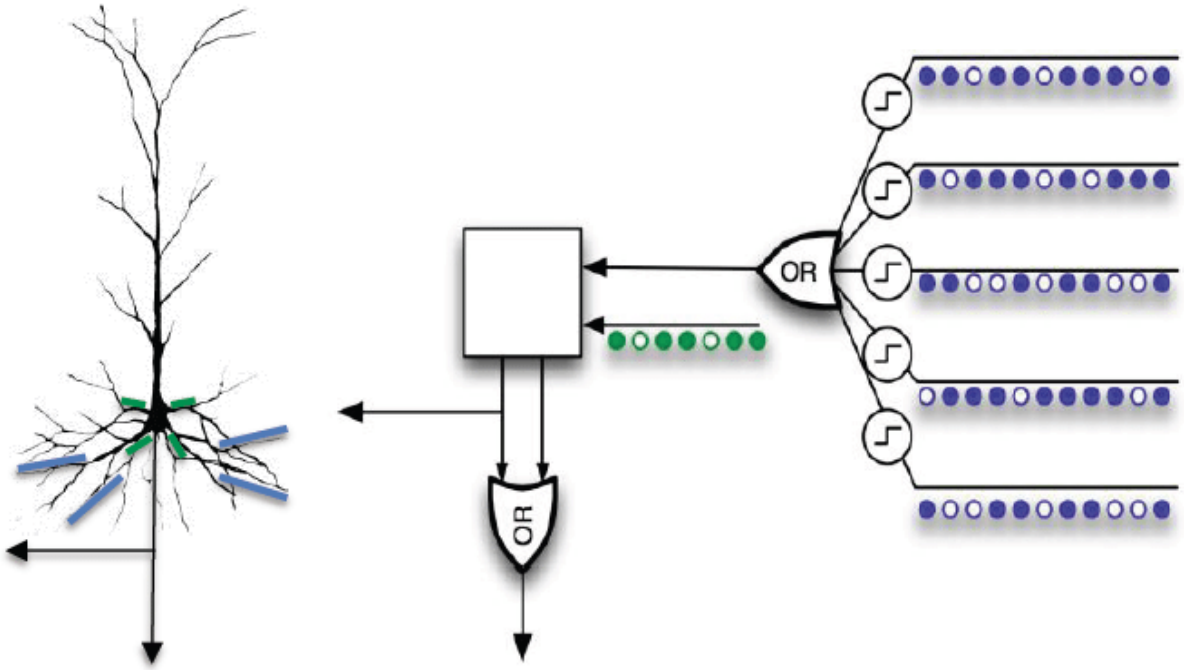

Figure 1: Hierarchical Temporal Memory Cell

Synapses in a Hierarchical Temporal Memory network represent a level of connection between two cells and are binary connected or not connected [12]. Although the connectedness of a synapse is binary, synapses have scalar "permanence" values which vary from 0.0 to 1.0 [17]. A synapse is considered connected or valid when its permanence is above a certain threshold which is a parameter of the network, and permanence values are updated during the spatial pooling portion of the cortical learning algorithm [12]. Synapses are used by the proximal and distal dendrites.

The proximal dendrite of a cell holds the synapses that bring a cell its feed-forward input [17]. It only holds a set of potential synapses which represents a proper subset of all the inputs to the node and a cell's feed-forward activation is the sum of the activity of these synapses which will be discussed further **Spatial Pooler** [12]. Neurons that are arranged in the same column share a proximal dendrite so that they can receive the same feed-forward response which is a desirable property for reasons we will talk about later [12].

Cells also have a list of distal dendrite segments [12]. Each distal dendrite segment contains a list of potential synapses to other cells in the node. Segments learn by updating based on forming connections to cells that were active due to feed-forward input in the previous time step in order to predict its own cell's activation [12]. When a cell has a segment which has a number of active synapses above a threshold, the cell is considered to be in a predictive state [17].

5.2 Initialization and Encoding

Hierarchical Temporal Memory needs a way to move from the set of possible inputs, the input set, to the sparse distributed representations. This is accomplished by an encoding function which is specified from the input set to some n-dimensional sparse distributed representation. A region's input space and output space are the same; an HTM takes in a time series over the space and predicts an element or set of elements of the input space it believes to come next [18]. Each region has an input space which could be a single input set or the Cartesian product of multiple input sets. As an example, {0,1,2,3,4} could be an input set and if a parent region receives input from child region which has {0,1} as its input space and another child region with {red, green, blue} as its input space, then the parent region's input space would be {(0, red), (1, red), (0, green), (1, green), (0, blue), (1, blue)}. An encoding function from an input space to n-dimensional sparse distributed representations is not guaranteed to be surjective or injective. We would not expect surjectivity, but could get it through nearest neighbor decoding, but injectivity is something that one would think would be preserved. However, because parent regions are allowed to take a slice of the output vector (not the "on" bits, just an arbitrary portion) when being fed to from multiple child regions, we cannot guarantee the property.

Before anything is sent through an HTM, each column is initialized with a list of initial potential synapses [12]. First we select a set of random inputs from the input set, then each input is represented by a synapse and given a random initial permanence value around the synapse validity threshold which are assigned such that each column has a bias towards the center of the input space [12]. Before we begin to send input to the system we also need an encoder which is a surjective function from the input space to our sparse distributed representations so that the HTM can interpret our time series.

5.3 Spatial Pooler

The Spatial Pooler is named so because the idea is to lump or pool together concepts that are semantically similar. It gives HTMs the ability to generalize and abstract. It accomplishes this through a couple of interesting steps that allow the system to preserve crucial information about the input in a sparse distributed representation but still generalize away some of the information in a algorithmic way. The fact that it is able to accomplish this through at all, let alone through the series of steps it takes is frankly incredible.

The algorithm begins by computing the overlap that the input vector, $\overrightarrow{v}$, has with the particular column [12]. This can be accomplished by generating what I will call $\overrightarrow{pd}$, the vector of connected synapses on the proximal dendrite where "1" represents a connected synapse and "0" can represent either a potential synapse which is not in the subset held by the proximal dendrite but exists in the node's set of potential synapses or a potential synapse in the column's proximal dendrite's set of potential synapses which is not connected. The column's overlap field is initially set to $overlap(\overrightarrow{pd},\ \overrightarrow{v})$. Then the columnn's overlap field is then compared to a minimum overlap parameter, and is set to zero if it is less than the parameter or multiplied by a boost function if the column's overlap field is greater than or equal to the parameter [12].

Next, the Spatial Pooler takes the set of columns with an overlap greater than zero and produces a subset that represents the most highly activated columns in each area of the node [17]. This is accomplished with a desired local activity parameter which is used in conjunction with an inhibition radius to calculate a threshold at which its active overlap is high enough and if the overlap is greater than that threshold, it is added to the set of "active" columns [12]. The threshold is calculated by using the integer desired local activity parameter $desiredLocalActivity$, to find the overlap of the column with the $desiredLocalActivity^{th}$ highest overlap score [12].

Lastly, the algorithm performs its learning stage [12]. This consists of updating the permanence values of potential synapses in the proximal dendrite of each active column, updating the boost of each column, and updating the inhibition radius [12]. For each potential synapse in the proximal dendrite of each active column, the permanence is incremented by a parameter if the synapse is active and decremented by the parameter if not, but always bounded between [0,1] [12]. Then the algorithm calculates a minimum desired firing rate for each cell to adjust the cell's boost accordingly and the cell's synapse permanences can be increased by a scalar if the cell is not having large enough overlap using a sliding average [12]. The learning phase finishes by adjusting the inhibition radius in order to ensure that the representations the system will produce will be of the desired sparseness [12].

5.4 Temporal Pooler

The Temporal Pooler is named as such because its job is to pool together patterns through time series that it finds to be similar. Thus it is the Temporal Pooler's job to take the spatial patterns and attempt to find patterns through sequences of them [17]. It is here that the cell structure of columns in HTMs matter because they allow the system to represent various patterns in different contexts [17]. In an HTM node with 4 cells per column, 100 columns, and a 4% activation rate, 4 columns represent each input, but each input can be represented $4^4 = 256$ ways. For example, the words "eight" and "ate" are homonyms and would be hard to tell apart if you heard the two without context, but in the sentences "I ate a cookie" and "I have eight pens" we are able to correctly distinguish between the two words solely because of their context.

The first job of the Temporal Pooler is to decide which cell in the column should become active due to

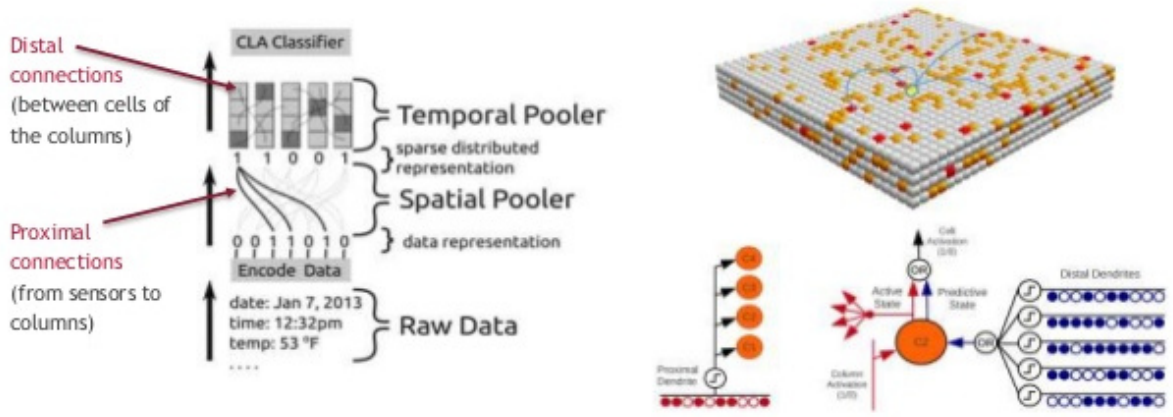

Figure 2: Spatial and Temporal Poolers

the column's activation [17]. If any cells in the column were in a predictive state in the previous time step, we say those cells predicted the input, and those cells are chosen [12]. If none of the cells were in a predictive state, we put all of the cells in the column into a active state and the best matching cell for predicting the particular input is selected to be the learning cell and is given a new segment on its distal dendrite [12]. The idea behind making all the cells active is to say, "We don't know how to interpret this, so all contexts are valid interpretations at this point" [17].

Secondly, the system needs to calculate the predictive state for each cell [12]. For each segment on the distal dendrite of each cell, if the segment is considered active, because enough of its synapses are active, the cell is put into a predictive state [17]. The cell also has active synapses added to that that segment and another segment that has the best match to the activity in the previous time step in order to learn [12]. These changes are not immediately implemented however, they are queued up and when the next set of feed-forward input comes, if the cell is selected as a learning cell (its prediction was correct) it is positively reinforced, else if the cell stops predicting it is negatively reinforced [12].

5.5 Hierarchy

The hierarchy of Hierarchical Temporal Memory comes from the ability to stack HTM regions in a tree like structure so that outputs of multiple HTM regions can be fed into a single parent region. So instead of interpreting the set of cells in a predictive state (that regions interpretation of its input) directly, we pass that information to another region with its own independent initialization. The parent region can also possibly receive another region's output or another raw input to contextualize the data. This ingenious structure comes from Jeff Hawkin's 2004 book "On Intelligence," where he proposed that larger objects are composed of smaller objects (sentences of words, words of letters, etc.) and this structure allows lower regions to identify and predict the individual components and then higher regions to connect these components into more complex ideas [3].

5.6 Decoding

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 [8]. At its essence, an SDR Classifier is a single layer, feed forward, neural network [8].

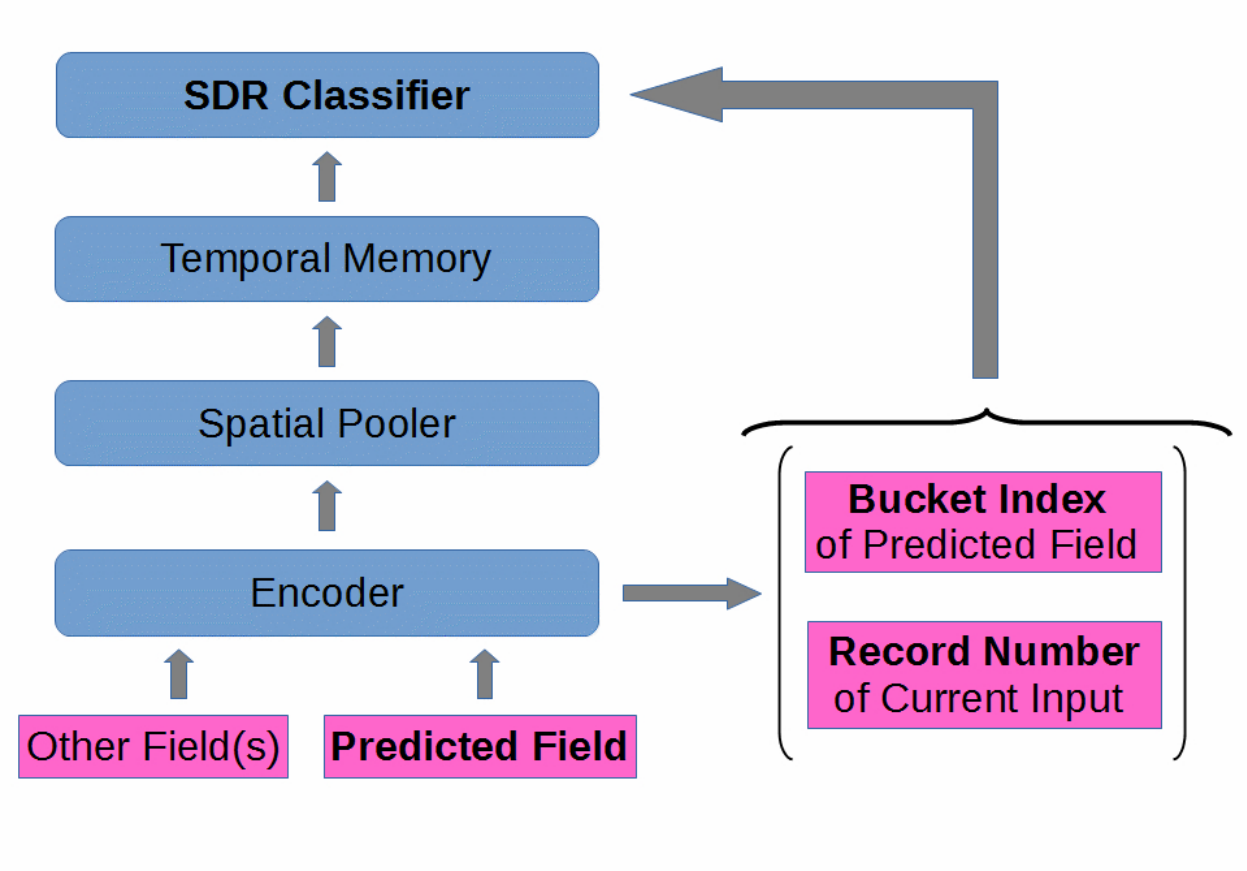

Figure 3: SDR Classifier

SDR Classifiers are able to decode the information HTMs give about predictions n times steps in the future by maintaining a weight matrix [8]. 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 $t - n$ [8]. 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 [8]. 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 [8].

6 Literature Review

Hierarchical Temporal Memory has sparked a great amount of interest in the artificial intelligence community as everyone scrambles to ask how this biologically-inspired approach will fit into the Machine Learning landscape going forward. How do we mathematically formalize this emulation of the neocortex in order to integrate HTMs into the tools and systems we use today? How does their performance compare to the tools in use today when it comes to typical tasks like signal processing and forecasting? Will HTMs become an industry standard, fade to irrelevance, or somewhere in between? Much work has been done in this field and there is still much to be to done.

Shortly after introducing the world to his theory of Hierarchical Temporal Memory, Jeff Hawkins along with Dileep George released a theory on the guiding mathematical principles in "Towards a Mathematical Theory of Cortical Micro-circuits". This mathematical framework is very high level, never diving past the node level of the HTM, but is quite interesting nonetheless. Under this model, Hierarchical Temporal Memory networks use Bayesian belief propagation as messages from one node represent a degree of certainly which is then used by the parent node to form its own inferences with their own degrees of certainty [10]. They claim each node "contains a set of coincidence patterns and a set of Markov chains defined over the set of coincidence patterns" [10]. The framework is unable to describe how to convert the belief messages to the messages that a node sends to its children and although it is incredibly interesting this loose description of the system seems quite lacking.

Some work has been done in formalizing the operation of the Spatial Pooler [14]. This work related the algorithms, operations, and data structures of the Spatial Pooler to already existing ones, finding similarities in order to advance the computational efficiency [14]. This mathematical framework is helpful for initializing and optimizing HTMs, but does not go as far as describing what a Hierarchical Temporal Memory network does to produce its forecasts.

One of the applications of Hierarchical Temporal Memory that has been explored in some depth is its use in algorithmic trading. Many genetic algorithms and other methodologies for developing automated traders are only good at finding optimal parameters for their training data, but suffer once exposed to new data, a problem referred to as over-fitting. The hope is that Hierarchical Temporal Memory's ability to abstract spatial and temporal patterns will allow for the production of automated traders that are more resilient to over-fitting. Preliminary work in this field seems promising [3].

An empirical study of Hierarchical Temporal Memory looked at its performance for biometric keystroke analysis, network data analysis, and website visitor analysis and concluded, "that the predictions made by HTM models are clearly comparable, if not better, than the ones of the state-of-the-art approaches" [9]. The study also reaffirmed the conclusion of those using it for algorithmic trading, that HTMs are not very prone to over-fitting because they learn to generalize with more data [3] [9]. This study also found that finding the optimal parameters for an HTM is a very computationally intensive process because of the dependence of the parameters on each other making the search exponential in parameters that need to be optimized [9].

7 Proposed Work

Hierarchical Temporal Memory is a branch of artificial intelligence grounded in Emulation, but I would like to take their strides in the field and see if can be improved using the innate topologies and inexact logic. I would like to implement my own Hierarchical Temporal Memory network using topological concepts to attempt to improve on the existing routines and enable broader logical frameworks to explore the performance and capabilities of such a system. Another possible advantage of using the topological properties is that I may be able to integrate topological or positional logic in order to enhance the system. Fuzzy logic and soft computing are another route I would like to take in improving Hierarchical Temporal Memory. It is my hope that this work will improve on the performance and forecasting ability of Hierarchical Temporal Memory.

I am going to work on the coding and decoding process to attempt to find a better process for interpreting the output of a Hierarchical Temporal Memory network. It is my hope that a formal framework for cortical learning algorithms using topological concepts will enable me to build an algorithm which is better able to specify the range of likely predictions. Graph algorithms such as breadth first search on an undirected graph representing the output space and coding theory tools are just a few among the dozens of techniques we could try by reducing the problem to a more concrete one.

I would like to use the floating point numbers and other non-Boolean values that are currently mapped to concrete states such as "active"/"not active" and "predictive"/"not predictive" and allow HTMs to work with them in these fuzzy states where cells will have degrees of activity. It is my hope that allowing the degrees of set membership and uncertainty to more explicitly propagate through the network, we will be able to extract the network's probabilistic beliefs about the subsequent time steps. Additionally, I believe that using prediction based on degrees of belief rather than Boolean belief will result in greater stability in the system as the system will be able to more readily adjust to new information in every time step leading to fewer paradigmatic changes resulting from Boolean shifts in lower regions.

The first step in my research would be using topological properties present in Hierarchical Temporal Memory networks to attempt to improve upon the current decoding scheme. It is my hope that using the topological structures present we will be able to produce a more efficient or accurate decoding mechanism than Sparse Distributed Representation Classifier which essentially tracks the correlation between outputs and events. Using metric spaces and graph algorithms, I believe it is possible to build a decoding mechanism that first "decodes" the output with certainty as a function of distance in the metric space and then multiplies a vectors of probabilities that the output is referencing each element of the input space by a weight matrix to produce the prediction at time n. This would mean that the uncertainty of decoding would be dealt with in the graph algorithms and the weights in the weight matrix would not have to bare as much of the burden.

The next step in my research would be to produce a fuzzy Temporal Pooler. The Temporal Pooler would keep the mechanism that decides which cell in a column should become active, but rather than a Boolean "predictive"/"non-predictive" state, cells would be able to represent a degree of predictive activity. There would be many challenges to overcome to accomplish this, such as changes to the learning mechanisms in the Temporal Pooler, possible changes to how the segments work, and whether the output for the cell is the raw activity level of the cell or possibly another value (such as a normalization using Softmax of the cells predictive states). The end result of this is to hopefully produce an HTM that can itself produce probabilistic predictions rather than relying on a probabilistic interpretation of its output.

After this is done, I would like to reconcile this fuzzy Temporal Pooler with the rest of the system. This

would mean producing a decoder that is able to handle the output of a fuzzy Temporal Pooler and convey which which uncertainties are a result of the probabilistic forecast and which are a result of an inexact decoding. From there it would be important to alter the Spatial Pooler to take in non-Boolean vectors for the system to work in a hierarchy. This could be done by mapping the vector to a Boolean vector or by altering the Spatial Pooler to allow for fuzzy logic.

The hopeful net result will be a Hierarchical Temporal Memory network that is more flexible, more stable, and more capable. Failing that, this exploration will explore the uses of the innate topological structures in Hierarchical Temporal Memory, help us define the boundaries of what Hierarchical Temporal Memory may and may not be able to achieve, and explore if and how the concepts of fuzzy logic, fuzzy control, and soft computing can improve upon Hierarchical Temporal Memory.

References

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

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

[5] B. G. Buchanan, *A (very) brief history of artificial intelligence*, AI Magazine, 26 (2005), p. 53.

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

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

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

[11] 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.

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

[13] M. T. Jones, *Artificial Intelligence: A Systems Approach*, Infinity Science Press, Hingham, M.A., 1 ed., 2007.

[14] J. Mnatzaganian, E. Fokoué, and D. Kudithipudi, *A mathematical formalization of hierarchical temporal memory's spatial pooler*, Frontiers in Robotics and AI, 3 (2017), p. 81.

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

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

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

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

[19] J. R. Searle, *Minds, brains, and programs*, Behavioral and Brain Sciences, 3 (1980), p. 417–424.

[20] A. M. TURING, *Computing machinery and intelligence*, Mind, LIX (1950), pp. 433–460.

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