CSCI-731 Project: Toward An Unsupervised, Incremental, Streaming and One-Shot Visual Class-Based Learning and Recognition System with Hierarchical Temporal Memory Theory

Brody Kutt
bjk4704@rit.edu

1. INTRODUCTION

1.1 Overview

In this project is designed a vision experiment using Hierarchical Temporal Memory (HTM) theory. HTM is a biologically plausible theory of the structure, function, organization and interaction of pyramidal neurons in the neocortex of the primate brain [14]. To be clear, HTM bears little resemblance to traditional machine learning and deep learning. Objectively, deep learning bears little to no resemblance to actual neuroscience. Where deep learning is encompassed by the act of complex functional approximation, HTM is founded upon massively distributed sparse encodings and Hebbian-like local learning strategies which are theorized to help provide for flexible, continuous and incremental sequence learning in cortex.

This experiment is intended to use HTM's core spatio-temporal learning algorithms in a way that, as far as I'm aware, has not been explored before. Historically in computer vision, classification of symbols in images such as digits has been taken to be an inherently static task. With state-of-the-art technologies such as deep convolutional neural networks (CNN) [22], the CNN is expected to receive a single instantaneous depiction of the symbol and perform static matrix operations on that image to calculate a classification output. At a high level, this is the way classification of images of all kinds has been done for decades. There is no concept of time in the process. Obviously, humans don't do anything truly statically. When humans observe an image, the orientation and focus of our eyes constantly change through time. For instance, one type of eye movement known as saccades are rapid, ballistic movements of the eyes that abruptly change the point of fixation [32]. Saccades can be elicited voluntarily, such as when surveying a room, but they also occur completely involuntarily whenever the eyes are open even when fixated on a target. The latter type of involuntary movement is theorized to take place because the fovea centralis (a bundle of cones on the retina which provides for the sharpest possible vision in the center of the visual field) is very small and it benefits us to expose it to as much relevant light as possible. The fovea centralis can be visualized in Figure 1.

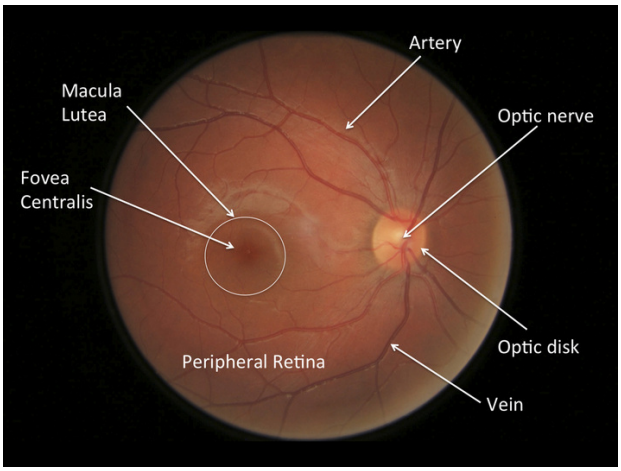

Figure 1: Anatomical visualization of a human eye featuring the location and relative size of the fovea centralis [2].

I will refer to the artificial center of the visual field as the focus window from here on out. Involuntary eye movement that takes place when focused on a stationary object is what I artificially create in this project. The focus window is manipulated manually to accomplish this. Directly observable in everyday life, when presented with a foreground subject in an image, our focus typically centers around its lines and curvature. The patterns of light hitting our retina are constantly changing as we reach a conclusion of its content which often happens extremely quickly. It is natural to envision the task of image classification as a temporal task with HTM in the way it is inherently a temporal task for humans.

In summary, the overarching goal that this exploration is intended to work toward is to frame a historically static computer vision task as a task of sequence learning and prediction such that it is made accessible to HTM. Doing so could potentially provide for a completely unsupervised, incremental, real-time and one-shot visual classification system. That goal is not going to be realized in this work alone. If anything, this work is meant to inspire future ex-

ploration and extensions to this work that uses HTM in this way. In this work, we wish to specifically test and explore three qualities that each are necessary, if not incredibly useful, to any intelligent agent sensing visual information in the real world.

1.2 Hypothesis

By framing a historically static image classification task as a streaming sequence learning and prediction task, we can show that HTM is capable of three things. Firstly is forming spatially invariant representations of classes with relatively few instances compared to deep learning. Secondly is reliably sensing when previously learned (and separately unlearned) information is being examined in the context of visual class-based learning. Thirdly is the ability to incrementally learn new classes on-the-fly without catastrophically forgetting previously learned information.

Note that this work is not meant to provide a concrete answer to this hypothesis due to a lack of breadth in testing. This work is only meant to be a proof of concept toward accepting or denying the hypothesis.

1.2.1 Why Do We Care?

One may rightfully ask why bother trying to use HTM in historically static classification tasks. There are a few reasons for this. Firstly, historically static classification tasks such as digit recognition have a large foundation in the computer vision, machine learning and broader artificial intelligence community. Benchmark datasets like MNIST are used in thousands of papers and can provide for clear unbiased comparison to traditional machine learning. Secondly, HTM boasts a number of practical benefits over traditional machine learning and deep neural networks. These qualities deem it significantly divergent from classical machine learning and even time-based deep learning models like Recurrent Neural Networks and Long-Short Term Memory networks. These qualities are enumerated below.

1. HTM systems inherently learn continuously and can autonomously adapt to changing data. They require no expensive offline training procedures.

2. HTM systems often need only a very small fraction of examples to learn from compared to what is typically required with deep neural networks [7].

3. HTM systems employ local learning strategies as opposed to global optimization procedures and thus they are not subjected to the phenomenon of catastrophic forgetting.

4. Pattern encoding in HTM is sparse and distributed thus the networks are robust to partial failures and subsampling [13].

5. HTM is inherently able to make multiple, simultaneous predictions that maintain discernibility as long as the patterns are large and sparse.

6. HTM is completely agnostic to the underlying data format as long as it has been encoded into a binary vector representation.

1.3 High-Level Experimental Design

Discussed more in Section 2, current software implementations of HTM feature a sparse and distributed memory model with local learning rules that is capable of storing, learning, recalling and predicting temporal sequences in an unsupervised, incremental and continuous fashion to meet the demands of real-time tasks. The fundamental output of an HTM network is the predicted state of cellular activation internal to the network for the next timestep. We can measure how well an input was predicted by comparing how similar our activation state is to our predicted state. Currently, this functionality is used for streaming anomaly detection tasks [5]. We will use this streaming measure of prediction quality for this experimental exploration.

I'm going to focus on simple hand-drawn symbols in images for this experiment. Using any symbol would suffice, but to keep things simple, I'm just going to use digits. In short, I will hard-code the functionality of eye movement through time while an HTM network is exposed to a drawn symbol in a black and white binary image. Through explicit hard-coded manipulation, I can move the artificial center of focus around the digit and let the HTM network learn the sequence of neuronal activation states associated with that digit and that pattern of movement. In humans, eye movements are controlled by behavior and attention feedback circuits which involve other brain structures [32]. The theory is not completely settled on exactly how this all works in the brain yet. So, for simplicity, I'm going to artificially create this movement in code.

1.3.1 Phase 1: Toward One-Shot Learning

I will expose the HTM network to an instance of a digit. The focus window will begin to move around the digit as if surveying it. If and when the HTM network reaches a point where it can correctly predict activation states beyond some reasonable threshold, we can conclude it has "learned" the digit in some sense since uncertainty in the network is low. Low streaming prediction error is indicative of the network experiencing something that it has previously learned. Note that this process takes place on only one single instance of the digit at a time. Depending on the results, I plan on only feeding in one or only a few representative samples of the digit in total. We repeat the process of exposing the HTM network to the digit instance until prediction errors have settled down for each instance while synaptic adaptations are taking place. It will be of interest to see if with sequential exposures the prediction errors take less time to settle down. After all instances have been processed, I can manually turn off learning to prevent any further modifications to any synapses. I will then present the learned HTM network with a new unseen instance of the same digit as well as instances of entirely new digits for comparison. The streaming prediction errors experienced when presented with an instance of the learned digit compared with those experienced with unlearned digits is the experimental variable. If the streaming prediction errors experienced with the unseen instance of the previously learned digit are significantly lower than with totally unlearned digits, we can conclude that the HTM network has learned something of value from a single or few instances of the digit. It will have used this previously gathered temporal knowledge to recognize a different instance of the same digit. This is sometimes referred to as one-shot learning.

1.3.2 Phase 2: Toward Incremental Learning

Using the same network, we can then turn learning back on and repeat a learning phase with one or few instances of an entirely new digit class. Once that is finished, we can turn learning back off again. The testing portion at this phase will include presenting the network with the same set of digit instances and seeing if the new digit class is recognized but looking specifically for if the previously learned digit was still remembered. The remembrance of the previously learned digit will be evaluated by the presence of noticeably lower streaming prediction error compared to examination with unlearned digits. This second phase of course lightly tests the capacity for incremental learning.

1.4 Why Keep It Simple?

The human visual system includes complex processing before any signal from the retina actually reaches cortex [1]. Furthermore, visual cortex is most often composed of several hierarchical processing layers [3]. These are both likely necessary to make sense of the extremely dynamic and colorful world we live in. Despite continuing progress, the standard HTM model includes only one hierarchical layer of processing due to incompleteness in neuroscience understanding. Unfortunately, hierarchical processing in the brain is more complex than simply stacking cortical regions on top of each other in a feedforward fashion such as is seen in deep learning. Fortunately, however, there is ample evidence that each cortical column in each region has a homogeneous structure and performs the same fundamental set of algorithms on its input signals [26]. Evolutionarily, this makes sense as a general purpose design can rapidly increase in size to quickly provide for new functional brain areas as opposed to needing to evolve new structures for each. Hierarchical layers are believed to provide for more sophisticated generalization power as well as information fusion and abstraction. However, a single hierarchical layer of cortex should in theory be able to generalize and create invariant representations of its input to some extent. So, we are going to keep the task simple for demonstration purposes and simple proof of concept.

The preprocessing that occurs in the human visual pathway before any signal reaches cortex would need to be modeled at the early stage of the encoder for a standard HTM model. The spatial pooler and temporal memory in HTM networks are meant to model cortex and only cortex which performs its functionality on sparse distributed patterns of electrical activity on its input fibers. The functionality of cortex is completely agnostic to the origin of its data. For reasons of time-constraints, I'm going to use the simplest possible encoder which simply relays the binary pixel data to the spatial pooler. In all reality, a more powerful and biologically plausible vision encoder would include much more complex and interesting preprocessing of the pixel data.

1.5 In the Context of Classification

Note that observing lower prediction error alone does not provide any discriminating information that can be used to know which previously learned digit is currently being viewed. A fundamental property of HTM, it is hypothesized that all knowledge is encoded into sparse distributed representations of neural firing activity in the brain. Current HTM systems employ an engineered method of mapping states of neural activity to data values whether they be scalars or discrete nominal labels. For our task, the state of which neurons are currently predicted to become active in the next timestep would be mapped to what class label the network currently believes it is experiencing. In the rest of this section, I'll call this mapping function from neuronal prediction states to class label as the mapping model.

In humans, higher levels in the cortical processing hierarchy tend toward more general and invariant knowledge representations. In human vision, for example, it has been noted that neuron firing patterns become more stable across forms of visual variance at higher levels of visual processing [11]. For example, similar neurons fire at the top level of IT in the human visual system whenever a human face is being viewed no matter who it is or how they are situated in the visual field.

For current HTM systems, especially when working with only one hierarchical layer, we would need to build up a mapping model that continuously maps the current neuronal prediction states to class labels every timestep. Note that this could be done in an unsupervised and streaming manner *if* digits can be learned incrementally and events of high streaming prediction error show to be reliably indicative of new information being presented. With these abilities in place, in the event that previously learned information is determined to be currently under examination, we could augment our representation in the mapping model of whatever label the mapping model deems is most probably being examined with the new information. In the event that previously unlearned information is determined to be currently under examination, we can tell our mapping model to start learning a new class. Provided that the prediction states are separable and meaningful, data labels could simply tell us after the fact which learned class corresponds to which human-interpretable digit name. For a fully functioning classification system, we would additionally need to evaluate the separability and meaningfulness of the digit-wise neuronal prediction states. For reasons of time constraints, I'm not going to be doing that in this experiment. Refer to Section 1.2 for the goals of this work.

Lastly, note that the act of classification will take place over time as everything else is assumed to do so. This means the judgment of what classification label to assign to the network's current input can be continuously adjusted as the focus window continues to move about the image. Intuitively, the confidence of a label would likely grow the more that the focus window is exposed to the digit. Evidence for a classification label would likely increase and converge as the maximal amount of information has been absorbed by the artificial fovea.

2. BACKGROUND

For the sake of brevity, I'm not going to detail the entire HTM theory in terms of its underlying philosophical motivations, concrete biological evidence of design and driving ideology. For that information, the reader is directed to [14]. Instead, I will detail HTM networks as they are implemented in an open-source HTM implementation known as NuPIC [39].

The Numenta Platform for Intelligent Computing (NuPIC) [39] is an open-source HTM software implementation that was originally created by Numenta in June, 2013. Numenta's implementation of HTM has undergone many changes since its original conception. At the time of writing this, the most current version of NuPIC is v1.0.3. NuPIC v1.0.3 is ar-

chitected in a way that allows for experimental code to be developed in Python but more efficient implementations of the core HTM algorithms to be developed in C++ in the "NuPIC Core." There exists many open-source variants on HTM implementations with various algorithmic differences and in various languages such as Java and Clojure [28, 15]. However, the codebase that is freely available in NuPIC is the original HTM codebase that was used in [5]. Thus, NuPIC is the underlying implementation of HTM that is used in this work.

This section details the algorithms and structures of the most current generation of HTM implementations. This is not to be confused with the older, previous work from Numenta.

2.1 High-Level Design

An "HTM model" as implemented in NuPIC and discussed in this work is composed of a collection of components each with a unique purpose. There are 3 core components to the HTM models used in this work: the encoder, the spatial pooler and the temporal memory (or sequence memory) region. These HTM models are specifically designed to work with data that contains a temporal component. A high level illustration of HTM's core components and how they fit together in an HTM model is shown in Figure 2.

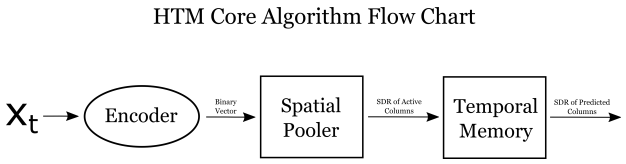

Figure 2: Illustration of a full HTM model's core components.

The data point at each timestep, x_t, is first converted into a binary vector representation that encodes the semantics of the data. The binary vector representation need not be sparse. From that binary vector representation, the spatial pooler converts it into a sparse distributed representation (SDR) that represents which columns in the temporal memory region have been activated by the feedforward input. The temporal memory algorithm uses those activated columns to preferentially activate the cells in each column that are currently in a depolarized "predicted" state, determine the next set of predicted cells for the next time step and lastly update all the synaptic weights to implement learning. Thus, the primary input to an HTM model is the timestep data and the primary output is the next timestep prediction of the state of active columns. The prediction of the next timestep state versus what actually got activated by the spatial pooler is what is used to perform prediction-based anomaly detection.

Furthermore, it's important to understand the terminology and organization of the model architecture. Within an HTM network, the most basic building block is an HTM neuron. Each neuron contains many dendritic segments and each dendritic segment contains many potential synapses to other cells in the network which represent lateral basal connections. These HTM neurons are organized into columns. In terms of the six layer anatomy of the neocortex [18], the set of columns in an HTM network is made to model layers two and three. Each cell in a column shares all feedforward connections to the spatial pooler output. That is, feedforward connections are only unique to columns. A visualization of the network at various levels of abstraction is shown in 3.

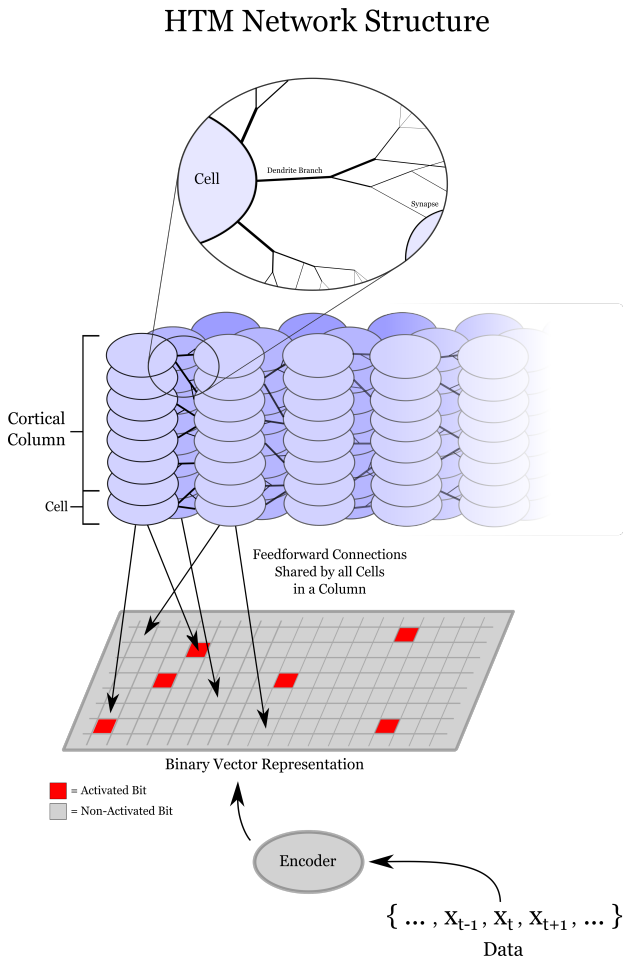

Figure 3: Visualization of an HTM network at various levels of abstraction.

Lastly, all synapses in HTM are modeled in the same way. Each potential synapse is associated with a permanence value that determines if a synapse has formed yet. If the synapse permanence is beyond a minimum permanence threshold, the synapse if formed and it contributes to NMDA spikes and action potentials. If the synapse is not formed, it does not have any effect. Note that synapses with a maximum permanence value and synapses with a permanence value at the minimum threshold have the same effect on spiking. This is okay because the network is built on massively distributed patterns of activity. Also, a synapse with maximum permanence has seen lots of reinforcement and can be interpreted as harder to forget since it would take many permanence decrements to erase the synapse compared to a synapse at minimal threshold permanence. This synapse permanence concept is visualized in Figure 4.

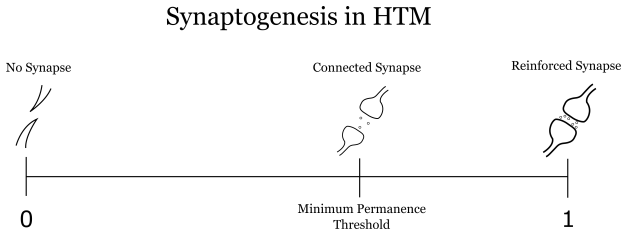

Figure 4: Visualization of synapse formation in HTM.

2.2 HTM Neuron

One of the foundational modeling details of NuPIC is the introduction of the HTM neuron. The HTM neuron can be considered an increase in modeling complexity and biological plausibility from the original perceptron model as originally introduced in [35]. Excluding numerous mathematical refinements, Rosenblatt's model is the neuron model seen throughout nearly all deep learning technology in some form. Rosenblatt's original perceptron model only goes so far as to consider the proximal synaptic input on pyramidal neurons. By this we mean the input to the neuron which can directly lead to action potentials; often called the feedforward input. A visual comparison of neuron model structure can be seen in Figure 5. Figure 5 comes from [13].

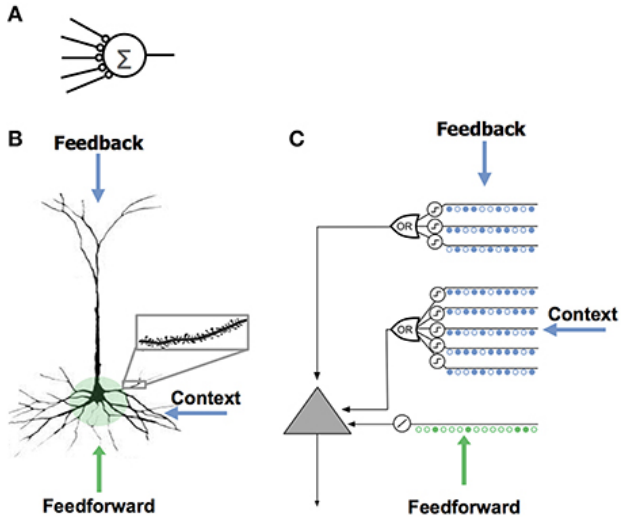

Figure 5: A visualization of different neuron models for comparison purposes. A. A perceptron model featuring feedforward integration. B. A pyramidal neuron with three synaptic integration zones. C. An HTM neuron with three synaptic integration zones [13].

The HTM neuron model incorporates signals from three different input zones. These three zones are the proximal zone, the basal zone and the apical zone. The proximal (or feedforward) zone consists of the dendrites in close proximity to the cell soma which can directly lead to an action potential. The basal zone includes lateral basal dendritic segments which branch profusely as they stem outwards towards other cells and cortical regions. The inputs on these dendrites can be considered to provide contextual information from neighboring cells in the same cortical region [33, 30, 42] and are hypothesized to enable the neuron to learn transitions of network states. These dendrites do not typically lead directly to an action potential but instead bring the cell into a slightly depolarized "prediction" state through an NMDA spike also known as a "dendritic spike." [9, 24]. Experimental results show that simultaneous activation of synapses in close proximity on a basal dendritic branch will combine in a non-linear fashion and cause an NMDA dendritic spike [21, 24, 37, 36]. It is believed the slightly depolarized cell is primed to fire early in this case and locally inhibit its neighbors in the process. This creates a competitive sequence of prediction and activation in which the cortex is biased towards activating it's currently predicted state. The apical zone refers to the input coming from the dendrites which stem from the apex of the soma. As opposed to proximal dendrites, the distal apical tuft usually receives inputs from more distant cortical and thalamic locations. The effects on the cell body from apical dendritic spikes are largely unknown though there exist some theories on their true function. These dendrites have long been believed to relay feedback information from higher levels of cortical processing in the form of expectation or bias [38, 20]. The interaction between action potentials and proximal, basal and apical synaptic integration is still a hot topic of research [34]. More discussion of the HTM neuron and its properties can be seen in [13].

2.3 Sensory Encoders

All different kinds of sensory information arriving on the input fibers of the brain must first be encoded into a common representation. The receptors on the retina along with the optic nerve in a human eye can be seen as a biological encoder: they convert external stimuli (light) into a form that is readable by the brain while encoding important semantics [29]. Semantics in terms of light hitting the retina is referring to things like where photons are colliding with the retina and with what frequency with respect to other areas of the retina. The cochlea is another example of a biological encoder which encodes the frequency and amplitude of sound waves into a sparse pattern of neuron activity using the stimulation of tiny hairs [41].

Encoding real-world streaming data into binary vector representations is an integral step of the HTM model. This is the job assigned to what are known as sensory encoders (or simply encoders) which represent the first step in the HTM algorithm. Sensory encoders play a role for HTM systems that is analogous to the role that biological encoders play for the brain. The input to an encoder is a timestep data value and the output is a binary vector. Note that the output of the encoder need not be in an SDR format. That is to say the output binary vector need not be sparse or distributed. The design considerations for sensory encoders are enumerated below.

1. It's very important that the encoder captures the semantic characteristics of the data (which are defined by the engineer) into the output binary vector. Semantics are captured by overlapping bits. This is to say semantically similar data should result in binary vectors with a high number of overlapping active bits relative to non-semantically similar data.

2. In general, encoders must be deterministic in the sense

that the same input should always produce the same output. Along those lines, the output of a sensory encoder always has the same dimensionality. The output binary vector is always the same size with activity distributed among all the bits.

3. Lastly, every possible binary vector that comes out of a sensory encoder should have a similar sparsity across the input space. This is necessary to work with the spatial pooler in its intended way. Enough density should be present, also, so that it is sufficiently robust to variation and subsampling for your application.

Consider, as a simple example, a binary vector output space of 6 bits.

$$\begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 0\end{bmatrix}$$

Say you want to encode integers while defining the semantic similarity between them to be their closeness on the number line. One possible way to define the encoding is shown below.

$$1 = \begin{bmatrix}1 & 0 & 0\\ 0 & 0 & 0\end{bmatrix} 2 = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} 3 = \begin{bmatrix}0 & 0 & 1\\ 0 & 0 & 0\end{bmatrix} 4 = \begin{bmatrix}0 & 0 & 0\\ 1 & 0 & 0\end{bmatrix}$$

$$5 = \begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 0\end{bmatrix} 6 = \begin{bmatrix}0 & 0 & 0\\ 0 & 0 & 1\end{bmatrix}$$

Notice how the entire representation space has been used. In other words, the representation of each integer is evenly distributed across the entire space. This is desirable to ensure we are using the maximal representation capacity available to us. Using the entire output space, we were able to encode six integers total. However, notice that the above encoding fails to capture any of the desired semantic similarity in the data. Each integer has exactly no overlap with every other integer and thus no similarity is expressed between any integer. Alternatively, consider this possible encoding shown below.

$$1 = \begin{bmatrix}1 & 1 & 0\\ 0 & 0 & 0\end{bmatrix} 2 = \begin{bmatrix}0 & 1 & 1\\ 0 & 0 & 0\end{bmatrix} 3 = \begin{bmatrix}0 & 0 & 1\\ 1 & 0 & 0\end{bmatrix}$$

$$4 = \begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 0\end{bmatrix} 5 = \begin{bmatrix}0 & 0 & 0\\ 0 & 1 & 1\end{bmatrix}$$

Now, only numbers that are next to each other on the number line have a single bit of overlap with each other and thus some of the desired semantics have been encoded. But, notice we were only able to encode 5 integers using the entire representation space this time. We could go a step further as shown in the next possible encoding below.

$$1 = \begin{bmatrix}1 & 1 & 1\\ 0 & 0 & 0\end{bmatrix} 2 = \begin{bmatrix}0 & 1 & 1\\ 1 & 0 & 0\end{bmatrix} 3 = \begin{bmatrix}0 & 0 & 1\\ 1 & 1 & 0\end{bmatrix} 4 = \begin{bmatrix}0 & 0 & 0\\ 1 & 1 & 1\end{bmatrix}$$

Now, notice how integers adjacent to each other on the number line will have two bits of overlap and integers that are within two integers of each other will have one bit of overlap. This means the underlying semantics have been encoded to a finer level of detail than the second possible representation. However, we were only able to represent four integers total.

This is a contrived example but it illustrates the trade off between the level of semantic similarity that is encoded, otherwise known as resolution, and the breadth of possible states to be represented when the output size is fixed. In reality, there exist many possible ways to encode integers, decimals, etc. as well as non-numeric data types but this trade off is always one of many concerns. Encoding is rarely defined so explicitly per possible value. Instead, encoders are typically defined as functions on the input space. It is ultimately up to the engineer to make these decisions about encoding and it depends entirely upon the specific application's needs.

In theory, the underlying data, x_t, that is fed into the sensory encoder may be in any form with any number of attributes. Your temporal data may be in the form of scalar values, nominal classes, ordered values, text, images and anything else imaginable. The only requirement is that the data be encoded into a binary vector format that captures the important semantics of the data. In design consideration number one, it is mentioned that the engineer may define the semantics of the data however they please. This choice is entirely dependent on the underlying goal of the application. By designing the encoder, you are explicitly telling the HTM model what data are semantically similar and what aren't. There have been many proposed encoder designs as a result for even seemingly simple data formats like scalar values. A more in-depth discussion on encoders than what is offered in this work can be found in [31].

2.4 Spatial Pooler

Overview.

The job of the spatial pooler is to make sense of the messy, irregular input coming from the sensory encoders. In the brain, each cortical region receives converging feedforward input from multiple locations all with varying sizes and signal sources. Each region has no concept of where the given signal is coming from and what it's characteristics will be such as the number of input axons the signal comprises. With multiple feedforward input sources all mixed together, it doesn't make sense that a region could uniquely process all of these inputs without some common representation that it builds and operates on. The spatial pooler aims to create efficient normalized representations of all it's input at a fixed size and sparsity while retaining the signal's semantic information so that the temporal memory algorithm can make sense of it. This is achieved chiefly through Hebbian-like learning principles, homeostatic excitability control, topological organization of sensory cortical connections and activity-dependent structural plasticity. All of these mechanisms are observed computational principles of the mammalian neocortex [6, 10, 40, 19, 43].

Concretely, the spatial pooler is named as such because the algorithm learns to group together spatially similar patterns into a common representation. Input patterns that are spatially similar are determined as such because they share a large number of co-active bits in the encoder's output space. This has the effect of helping to learn invariant representations of abstract patterns where semantically similar patterns are naturally grouped together in the output representation. In an end-to-end HTM system (encoder, spatial

pooler and temporal memory), the spatial pooler provides the SDR representation of the input that the temporal memory algorithm learns to predict. A single region in an HTM system is organized into a set of columns where each column has a set of associated neurons. The spatial pooler takes as input the output from the sensory encoders which are semantically encoded binary vectors. The output is an SDR of bits where each bit represents the activity state of columns in the HTM model. In that sense, the spatial pooler only operates at the level of entire columns in the cellular representation of the model. The spatial pooler decides which columns should become active and the temporal memory region decides from there which cells in the column to become active and which become depolarized. In the context of the three synaptic integration zones of a pyramidal neuron, the spatial pooler models the feedforward connections into the proximal dendrites. A visualization of the spatial pooler's structure and functionality is shown in Figure 6. Figure 6 comes from [8].

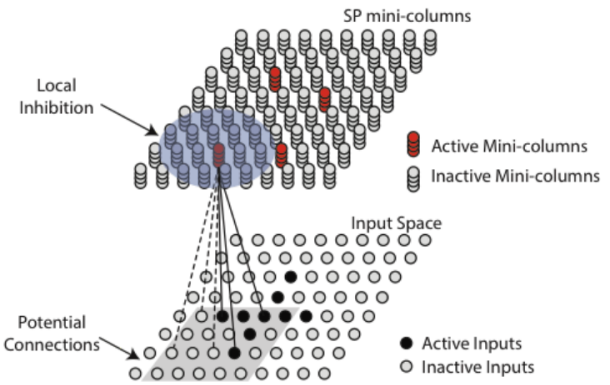

Figure 6: A visualization of the spatial pooler's structure and functionality. Each column in the network is associated with a subset of potential connections to the input which is the output of the sensory encoder. Topological organization is encapsulated by the radius of the candidate area of potential synapse connections for each column. Local inhibition ensures only a small fraction of the columns which receive the most activation on it's connections are activated within a given neighborhood. Synapse weights between columns and input cells are adjusted according to a competitive system of reinforcement and punishment based on activity levels. In addition, homeostatic excitatory control, called "boosting," increases the relative excitability potential of columns that are historically inactive for the purposes of ensuring utilization of the entire space [8].

Algorithm Details.

The spatial pooler algorithm can be organized into three separate phases following an initialization.

1. **Initialization:** The spatial pooler must first be initialized before any input is received or any learning takes place. This is done only once. Each column has a candidate radius of input bits from which potential synapses may form called a potential pool. The initial list of potential synapses for each column is chosen randomly within its potential pool. Additionally, each potential synapse that is chosen is given a random permanence value to start. The permanence value is constrained to be between 0 (no synapse) and 1 (strongest possible synapse). These initial permanence values are chosen to be within a small range around the minimum permanence required to form a synapse. This ensures synapses can be connected and disconnected quickly upon the start of training as needed. The permanence values for each potential synapse for each column are also chosen to be higher towards the center of the candidate area on the input for that column to reflect the desired topological organization.

2. **Phase 1 - Compute Column Overlap Scores:** After the spatial pooler connections have been initialized, it is ready to take in input vectors. Recall these input vectors are the output of the sensory encoders thus they represent the signal source. The first step of processing these input vectors with the spatial pooler is to compute the overlap score for each column's synapses with that vector. The overlap score for each column is computed as the number of connected synapses (synapses that have a high enough permanence value) with active inputs multiplied by a boost value that attempts to mimic homeostatic control. We include a stimulus threshold at this point to ignore trace activations which are not significant. If the overlap score does not breach the stimulus threshold, it is set to 0.

3. **Phase 2 - Inhibition:** The next step in the spatial pooler algorithm is to select which columns should remain active after a process of local inhibition takes place. The inhibition radius is always proportional to the average receptive field size across all columns in the layer. The number of active columns allowable per inhibition area is chosen in advance as a parameter. If the number of active columns per inhibition area is chosen to be k, that means a column will stay active if it's overlap score is greater than the score of the k^{th} highest overlap score within its inhibition radius. This process of inhibition helps keep the sparsity of activation relatively constant. Inhibition is thought to help balance out excitatory firing in the mammalian cortex as they increase and decrease together. This balance is believed to be critical for proper cortical function [16].

4. **Phase 3 - Learning:** The last phase for a given input vector is the learning phase. The idea is to update the synapse permanence values for each potential synapse for each column based on a local competition-based learning mechanism inspired by Hebbian learning [6]. For columns that stay active through the inhibition process, for each of its potential synapses, if the synapse is active, its permanence value is incremented. Otherwise, if the synapse is inactive, the permanence value is decremented.

   Boosting factors are also updated in this step. Recall boosting comes into play in phase 1. Updating the boosting factors involves measuring the time-averaged activation level for each column (known as an active

duty cycle) and the time-averaged activity level of the column's neighbors. The boost factor is updated based on the difference between these two measurements. If a column has a low active duty cycle compared to its neighbors, it is incremented and vice versa. From an information theoretic perspective, it would be ideal for all columns to have a similar active duty cycle in every neighborhood. This boosting mechanism is inspired by numerous studies on homeostatic regulation of neuronal excitability [10]. Alternatively, if a column's time-averaged overlap score (known as an overlap duty cycle) is too low, its connected synapse's permanence values are increased.

Additionally, the inhibition radius (applying to all columns in a layer) is recomputed at the end of Phase 3 as the average receptive field size in the layer across all columns. The potential pool of synapses does not ever change for a column. However, the receptive field of a column only includes connected synapses (those with permanence values greater than the threshold). So, a column might have a large potential pool but a small receptive field if it has only formed synapses with the input connections nearest the center of it's potential pool (those potential synapses near the center are initialized to higher values because they are "closer" topologically). These receptive fields change over time as the duty cycle and representational burden of columns fluctuates, so they must be recomputed to determine the extent of lateral inhibition between columns. In other words, there should be more inhibition going on for a region which is composed of columns with large receptive fields since in general that layer will have more active columns for its inputs. Increased inhibition in that scenario proportional to the average receptive field size helps ensure sparsity is stable.

Algorithm Properties.

There exist notable and desirable properties of the spatial pooler algorithm which are a direct result of its design. These properties help the temporal memory algorithm learn useful patterns in data in a continuous and flexible fashion. It is believed that the mammalian neocortex uses these same properties to facilitate learning in the real world. These properties are enumerated below.

1. **A Common Representation:** The spatial pooler forms representations of its input with a fixed size and sparsity. This is important because the input signals to a cortical region may arrive from many different sources and consist of varying numbers of cells that change over time. A common representation from which to process on ensures useful synaptic adaptations can take place in temporal memory. In addition, fixed sparsity ensures each pattern has a similar probability of being detectable. If the sparsity of input activation was allowed to vary, input patterns with high activation density would be much easier to detect than input patterns with low activation density. This would result in a high false positive rate for high density patterns and a high false negative rate for low density patterns.

2. **Utilizing the Whole Space:** Through homeostatic excitability control, the spatial pooler is able to make use of the entire capacity of the network to create efficient and useful representations of the input. Neurons which fire disproportionately frequently or infrequently do not convey as much information as a balanced distribution of firing. In other words, a neuron that plays a role in many different patterns is accordingly less able to discern between it's patterns and the information gained by its activation is less interesting. Conversely, a neuron that plays a role in very few or no patterns is going to be fired very infrequently and is going to play a small role in detecting patterns on the input in general. What NuPIC calls "boosting" (discussed above) is a method of ensuring the neuron firing patterns are efficiently distributed across the entire space of possible neurons. This is done by throttling the relative excitability potential of cells that are firing frequently and the converse. The biological counterpart of boosting is the observed modulation of synaptic efficacy and membrane excitability in neural activity in the brain that is thought to constrain neural plasticity and contribute to the stability of neural function over time [10].

3. **Robustness to Noise:** The act of pooling together semantically similar patterns into a single output representation adds a layer of noise robustness to the system. Real world data signals as well as patterns of cellular activity resulting from responses of sensory neurons can vary widely for a given stimulus. These small variations in the input can make the job of recognizing and making sense of them near impossible if not mitigated by invariant representations. Related to this, the spatial pooler is resilient to trivial patterns by maintaining synapse permanence values and minimum thresholds for activation. A "survival of the fittest" environment is in place when it comes to patterns that *are* and *are not* recognized. A potential synapse needs sufficient activation over time to drive its permanence value up and lead to a fully formed synapse. If the synapse is not fully formed, it does not contribute to any action potentials nor dendritic spikes. This prevents connections that are anomalous, erroneous or random from forming and influencing the system at all. A minimum threshold for action potentials and dendritic spikes prevent patterns from being recognized unless sufficient confidence is in place conveyed by the presence of a number of connected active synapses beyond a minimum threshold. Without the threshold, many false positive activations would occur. This enables only the most clear and frequent patterns to dominate.

4. **Fault Tolerance:** Biological evidence for fault tolerance in the mammalian brain has been observed in patients with traumatic brain injuries such as a stroke. These patients experience damage to relatively large portions of their cortex. Typically sensory, cognitive and motor abilities may be diminished upon initial injury but are followed by substantial recovery [27, 12]. The spatial pooler's output representation is naturally fault tolerant due to the mathematical properties of SDRs (see the effects of subsampling in [4]) and the concept of self-adjusting receptive fields. By homeo-

static excitability control, establishing minimum thresholds for activation, maintaining a large pool of potential synapses for each column and adjusting each synapse permanence based on activity level, the population of columns in an HTM layer will learn to best represent it's input with it's given network parameters. If the column population size is reduced (or increased), the learning rules will dynamically adjust the output representation to fit the data with the new network characteristics.

5. **Online Learning:** Organic brains are highly "plastic." Regions of the neocortex are able to represent entirely different things in response to environmental changes. While fault tolerance refers to the spatial pooler's ability to tolerate internal faults, online learning is referring to the fact that it can dynamically adjust to changing patterns of the underlying data signal. The learning properties of the spatial pooler (at the level of individual synapses) are completely local and massively parallel. The adjustment of one synapse is independent of every other synapse. No offline training procedures are necessary as the data changes. As the distribution of patterns change in the data signal, so will the network in a completely autonomous fashion.

Further Reading.

For further discussion on the spatial pooler and experimental results, the reader is referred to [8]. For a mathematical formalization of the spatial pooler, the reader is referred to [25].

2.5 Temporal Memory

Overview.

The temporal memory algorithm simulates the hypothesized function of lateral basal dendritic connections emerging from the cell soma on pyramidal neurons in cortical regions. This kind of lateral synapse integration between neurons in the same region is believed to learn and predict transitions in the network state of activation in the brain. This is analogous in function to the recurrent connections on recurrent neural networks and long-short term memory (LSTM) networks [23, 17]. The goal of the temporal memory algorithm in terms of the entire HTM model is to learn and predict which cells (and associated columns) will be activated next given the past activation states.

Each static state from the spatial pooler takes the form of a sparse distributed pattern of co-active columns in the network. However, further detail at the level of individual cells within columns provides additional representational capacity which includes past context. The same static input state which consists of a subset of active columns can take many different forms at the level of what individual cells in its associated columns are active. Each different possible state for a subset of columns represents a unique context in which we are observing that input.

The HTM network variables that are adjusted within the temporal memory algorithm include the predicted state and activation state for each cell in each column and the synapse permanences on each lateral basal dendritic segment on each cell on each column. The proximal synapses connecting columns to feedforward input from the spatial pooler are not used or adjusted in this algorithm. The fundamental input to the temporal memory algorithm is a subset of columns chosen to be active and the fundamental output is a prediction of cells (and associated columns) to be activated immediately next. The entire temporal memory algorithm can be explained as an initialization followed by a three phase process, similar to the spatial pooler.

Algorithm Details.

1. **Initialization:** Initializing the lateral basal connections must be done before any processing of input takes place. This need only be done once. Each dendritic segment on each cell is given a set of potential synapses to other cells in the layer with a random nonzero permanence value. These initial permanence values are chosen randomly with some synapses being above the minimum permanence threshold (forming a synapse) and some not.

2. **Phase 1 - Compute the Activation State:** Computing the new activation state of the cells of each column is the first step to take when a new input is received. Recall the input is a subset of columns that have been chosen to become active. If the column that a cell is contained in was not chosen and is not activated in the input, there is no chance that that cell is going to become active. If the containing column is chosen, there are two possible cases for the cells within the column to become active. If there are no predicted cells anywhere in the chosen column, all cells within the column become activated. This is known as "bursting" and it stimulates new growth of synapses. Alternatively, if there exist some cells in a predicted state (from the previous iteration) then those cells and only those cells will become active.

   Activating only the cells in a predicted state in an active column models the context in which a sequence state has been observed. In other words, an individual sequence state is represented as a subset of active columns but the individual cells active within those columns represent the context in which that sequence state is being observed. In this way, a single sequence state can take many different forms based on what came before it at the level of individual cells. In the example provided in [13], this allows the network to know to predict "D" and not "Y" if it knows "ABCD" and "XBCY" and it sees "ABC" as the input.

3. **Phase 2 - Compute the Predicted State:** The second phase is to compute the new cells that should be put into a predicted state based on the new activation state. The lateral basal synapses on cells that have a large enough permanence will be activated if they are connected to an active cell. If enough activated synapses are simultaneously present on a dendritic branch, an NMDA spike occurs and the cell from which the dendritic branch originates will be slightly depolarized and put into a "predicted" state. This is controlled by a threshold hyperparameter representing the minimum number of activated synapses on a dendritic branch necessary to cause an NMDA spike. If the minimum threshold is not breached, the cell is not put into a predicted state. Recall that these

predicted states are predictions for the next timestep. Concretely, a cell in a predicted state means that the network predicts that the column in which the cell belongs will be activated in the next timestep.

4. **Phase 3 - Learning:** The last phase of the temporal learning algorithm is the synaptic adaptations that incorporate learning. This phase is similar to phase 3 in the spatial pooler algorithm with some additional details.

   The learning rule is still Hebbian-like in the sense that it rewards synapses that correctly predicted a column activation. There are two possible ways a dendritic branch can have its synapses reinforced. If a cell was correctly predicted (was previously depolarized and became active in phase 1), any dendritic branches off the cell that breached the minimum threshold and underwent an NMDA spike have its synapses reinforced. Secondly, if an activated column was unpredicted, there exists a need to choose a dendritic branch that will represent that pattern in the future. We simply choose the dendritic branch across all cells in the column that had the most synaptic input (even though it was below the threshold) to have its synapses reinforced.

   Reinforcing the synapses on the dendritic branches that were chosen in either of the two possible ways above is Hebbian-like. The goal is to reward synapses with active presynaptic cells (increase the permanence value) and punish those (decrease the permanence value) that don't.

   Lastly, in the event that a cell was predicted but did not become active, we apply a small decay to the synapses on the dendritic branches of that cell that caused it to become depolarized. This punishes those synapses that lead to an entirely incorrect prediction.

Algorithm Properties.

There are some interesting properties of the temporal memory algorithm that will be enumerated and discussed here. These help the reader understand the design choices and the expected model behavior.

1. **Utilizing SDR Properties:** The mathematical properties of SDRs are used at nearly every step in the temporal memory algorithm and help it to function desirably. In particular, subsampling of patterns is done on nearly every dendritic branch of each cell. In a population of cells, only approximately 2% of cells are going to be active at any given time due to local inhibition in the spatial pooler. Thus, a static pattern of activity is composed of roughly 2% of cells and some lateral dendritic branches on different cells might each end up learning to recognize this pattern. One might initially think the dendritic branch ought to be connected to all 2% of cells to recognize the pattern but in reality the value of connecting to only a small fraction of those active cells to successfully recognize the pattern is practically the same. If the network is connected to only a tenth of those cells, for example, the probability that those cells are activated for a different pattern is extremely small as long as the pattern is large and sparse. This also allows the dendritic branches to connect to multiple patterns at once. There is a chance of false positive pattern detection if connections to multiple patterns exist on one dendritic branch; such as if partial activation on each pattern sum together to initiate an NMDA spike despite no single pattern appearing fully. As long as the network is large and sparse, the chance of this happening is minuscule. See [4] for the mathematics to back up these claims.

2. **The Effects of Multi-Cell Columns:** One of the many parameters in an HTM network is to choose how many cells should be allocated to each column in a layer. This can vary from one cell per column to arbitrarily many. However, choosing one cell per column is effectively like building a Markovian model that is only capable of first-order memory on each transition. Within first-order memory, the system only remembers what happened immediately previous to the current input and does not use any other past information. This obviously severely limits the capabilities of the network for any temporal problems with sufficiently complex patterns. Any input would always produce the same prediction regardless of what happened before. Each different pattern represents a different context of previously activated columns that came before it. Thus, the same input pattern can be modeled in many different ways dependent on a variable amount of past context. The predictions that arise from a column activation pattern include a variable amount of past context as well. This is because if an active column contains any predicted cells, only those cells will become active. Accordingly, only the lateral synapses attached to those specific cells will become active which will significantly shape the prediction for the next timestep.

3. **Existing Limitations:** There exist some limitations with the current implementation of temporal memory that arise in practice. Firstly, it is known the number of different ways the same input can be represented in the network is very large even for modest sized networks. This means the representational capacity is large but the downside is that the number of possible contexts is often so large that repeated occurrences of an input even in the same context of interest are often seen as brand new because of an arbitrary, ultimately unwanted, amount of past context coming into play. In other words, there doesn't exist any way to consolidate and generalize among different contexts so as to recognize the semantic similarity between inputs with similar contexts and translate this into more intelligent prediction. The ability to generalize among contexts is called "temporal pooling." In order for an input in a given context to be recognized we know it had to have been seen before and lateral basal synapses would have had to form to lead those cells to be predicted. Each new context essentially demands a new set of lateral basal synapses which equates to a very massive number of these lateral synapses needing to be learned. The system will generally become more stable over time as synapses are forming but in practice this problem of fragility with respect to variable context often arises.

   NuPIC has engineered several attempted solutions to this problem: resetting and backtracking. Resetting

is the process of manually telling the network when a pattern has ended (and when a new one has begun). When a reset is called upon, the network state is reset and lateral connections from the end of one pattern to the beginning of another are forcibly not learned. This alleviates the above problem because when transitions are constantly learned between patterns, the amount of context being represented can run awry spanning multiple patterns in the past that may have no true bearing on the current data signal state. This resetting phenomena is clearly not autonomous, however, and is impossible for data without clearly defined patterns. Secondly, backtracking is implemented in NuPIC which is a way to manually search for previously known contexts in the event of high bursting activity. In other words, if an input comes along that is unknown (not well predicted) the algorithm manually searches for a context in which the input *was* known in the short-term past and locks onto that pattern (assumes we are in it) instead of building a new context. It is like manually trying to find when in the short-term past did a new pattern likely start since context in the representation has been lost. While this does sometimes help, it is limited by the fact that we have to store an unintuitive and inflexible number of past network states to search through for a fitting context. It is in no way guaranteed that the best fitting context is going to be reachable in the short-term past at all. Additionally, it is an engineered solution without any known biological plausibility. It is the belief of the author that mechanisms of attention could alleviate the problem of autonomously identifying the beginning and ends of patterns and hierarchical processing could alleviate the issue of consolidating and generalizing among contexts.

Further Reading.

For further discussion and formalization on temporal memory and experimental results, the reader is referred to [13] and [7].

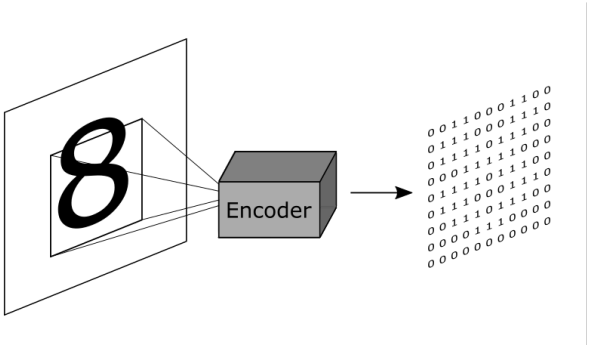

Figure 7: Illustration of how the focus window is oriented on an image and translates to a binary vector representation of the digit that is used as input to the spatial pooler.

Only the pixels that are within the current focus window are conveyed to the spatial pooler. The focus window is manually shifted around the image frame to implement some semblance of saccadic movement.

3. METHODOLOGY

I'm going to be using the most well-known, open-source implementation of HTM known as NuPIC [39] for my experiments. The modules I will be using from NuPIC are the C++ implementations of the spatial pooler and the temporal memory algorithm. The code can be utilized from a Python interface. For the encoder, I will implement my own "encoder" that simply uses the raw binary pixel data as the binary vector representation that is fed to the spatial pooler as input. The specific binary vector at any given time is a subset of the the full image. This encoding scheme is illustrated in Figure 7.

3.1 Experimental Data

Each image has dimensions of 32 by 32 pixels. Each digit is fit within a 16 by 20 pixel box. The box containing the digit is placed in the center of the 32 by 32 pixel frame. Our focus window is 22 by 22 pixels thus each digit instance can always fit within it. In order to include variance, we use instances of each digit that are originally depicted in different fonts before the binarization takes place. Each digit instance along with its font name is shown in Figure 8.

Figure 8: Visualization of each digit used and its font family.

There are some key assumptions on the data that we make in this experiment to provide for a very controlled environment. A simple and controlled environment is purposefully created for the sake of clear demonstration. This work is not intended to detail a fully functioning classification system. I only wish to do a proof of concept. Each digit used in this experiment adheres to the following strict criteria.

- All images are strictly single channel and binary.
- The orientation of the drawn symbol inside the image is irrelevant. The fixation point is assumed to be manually placed such that it is perfectly oriented with the symbol in terms of rotation and translation.
- The focus window contains only the desired symbol and nothing else. If time permits, I will experiment with occlusion and noise.
- Each digit instance is unique but all the digit classes are clearly distinguishable from each other. There is no ambiguity.

3.2 How Saccadic Movement is Modeled

The focus window is chosen to be a 22 by 22 pixel square. This is the window in the image that the HTM network is currently examining, akin to a human eye's fovea centralis. The total range of movement is defined to be five pixels toward the top of the image and 5 pixels toward the bottom of the image. The focus window begins perfectly centered in the image frame. It is well known that saccades are not continuous [2], thus the movements proceed skipping a distance of three pixels at a time. These skipping motions cycle through the range of motion moving to the top of the range to the bottom of the range and back around continuously. Doing the math, it takes 20 three pixel skips to eventually reach the same place in the movement cycle that you began. Defining the centered position as position 0, the entire ordered sequence of movement is defined below.

$$\{0, +3, +4, +1, -2, -5, -2,\ 1,\ 4,\ 3,\ 0, -3, -4, -1,$$

$$2,\ 5,\ 2, -1, -4, -3,\ 0\}$$

This pattern of movement is repeated for each digit where the focus window begins in the center of the image frame. In addition to the variance introduced by varying the font family, this varying focus window position forces the network to learn non-trivial characteristics about the digits in order to predict its input well.

3.3 Phase 1

Phase 1 will consist of using a single instance of the digit 8 drawn in the Calibri font for training purposes. The training process will consist of 20 cycles of the movement sequence defined in Section 3.2 while synapses are allowed to take place. The synapse updates on the spatial pooler will learn to pool together spatially similar input. The synapse updates in the temporal memory region will learn to predict sequences of cortical column activation as the movement continues. Note that no label is being fed to the HTM network. It inherently has no idea what digit it is looking at. Also note that this training phase is completely relatively instantaneously.

The testing piece of phase 1 will consist of first turning off any further synapse updates. This means the network is completely prevented from learning anything else about any other digit instance. The learned network will then be exposed to every digit across the five different font families undergoing four cycles of the movement sequence. The average prediction error response is captured and recorded during this time. I hypothesize with this experiment the network will be able to learn sophisticated characteristics of the digit from a single instance that it is able to generalize to four other instances of the digit from different font families.

3.4 Phase 2

Phase 2 will begin with the exact same learned network state from phase 1. The network will have only been trained on a single instance of the 8 digit. In this phase, the network will then undergo a training phase of 20 movement cycles on a single instance of the digit 7 also from the Calibri font family. Note these training periods occur incrementally and not simultaneously. This second training procedure will be followed by the same testing procedure after synapse updates has again been turned off. I hypothesize the network will

learn something new about the second digit but will also remember what it knew about the first digit.

Then, another sequential training procedure will take place on a single instance of the digit 2 also from the Calibri font family. The same testing procedure will follow this third sequential training procedure to test what the network has learned and what it still remembers. Ideally, the network will still remember what it learned from the previous training procedures.

4. RESULTS & DISCUSSION

In this section I detail the results discovered while executing the experiments outlined in Sections 3.3 and 3.4.

4.1 Phase 1

The results obtained during phase 1 are displayed in Figure 9.

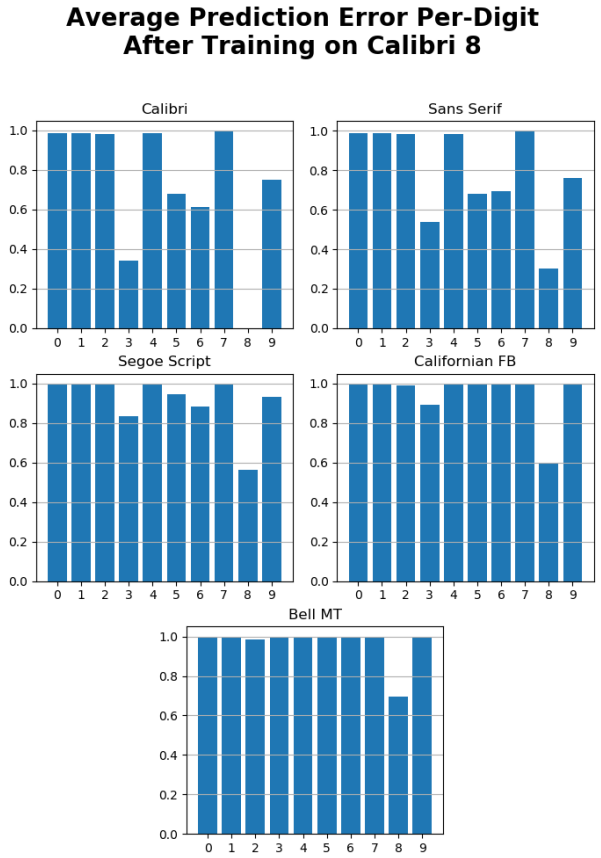

Figure 9: Visualization of the average testing prediction error after training the HTM network on a single instance of 8 in the Calibri font family.

Notice in Figure 9 how the digit 8 reflects the lowest average prediction error across all five font families by a relatively large margin of at least 20%. Interestingly, notice the drop in average prediction error in digit 3 across several font families. Once can easily make the argument of the geometric and thus semantic similarity between digits 8 and 3. It makes sense that 3 would experience a drop in prediction error. To a lesser extent, the rounded bottoms of digits 5 and 6 also seem to cause a drop in average prediction error.

4.2 Phase 2

The results obtained during the first part of phase 2 are displayed in Figure 10.

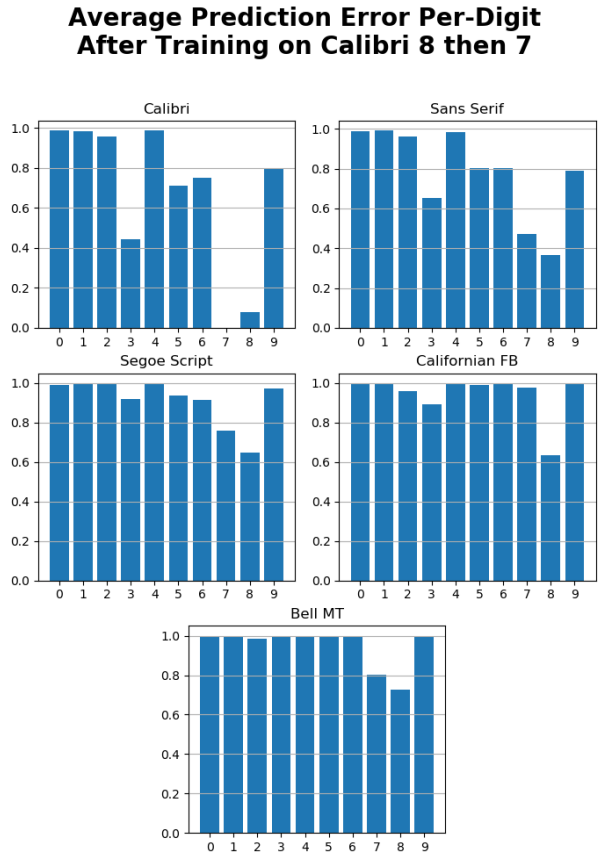

Figure 10: Visualization of the average testing prediction error after training the HTM network on a single instance of 8 in the Calibri font family followed sequentially by training on a single instance of 7 in the Calibri font family.

In Figure 10, we notice the lowest prediction error occurs when testing against both digits 7 and 8. The only blunder is seen with the Californian FB font family where it's knowledge representation of 7 is not useful to recognize that instance of 7. Importantly, notice that the prediction error experienced while testing against 8 is minimally perturbed compared to when we only trained on the single instance of 8. It appears the network even remembers what it derived about the digit 3 from its exposure to 8.

The second piece of phase 2 is to introduce another training procedure of a single instance of 2. The results from this are shown in Figure 11.

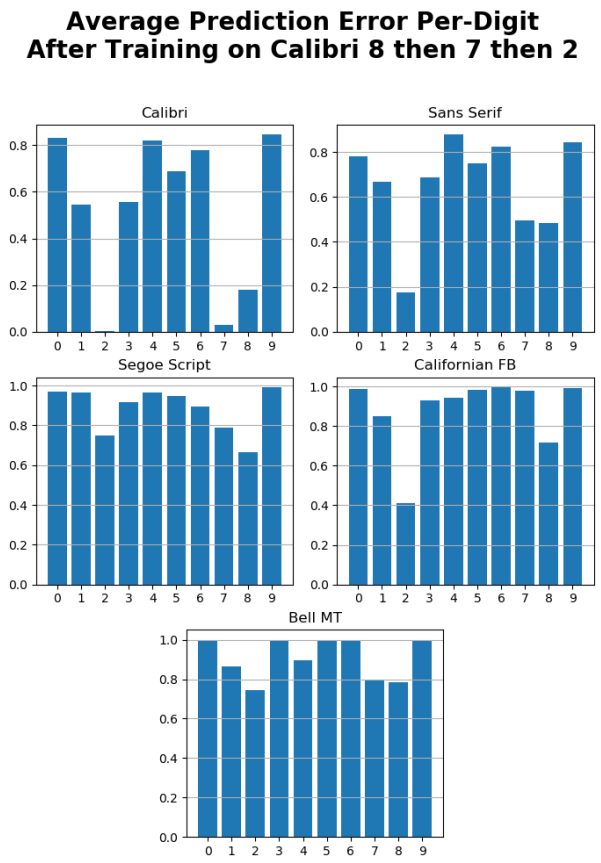

Figure 11: Visualization of the average testing prediction error after training the HTM network on a single instance of 8 followed sequentially by training on a single instance of 7 followed sequentially again by a training procedure of a single instance of the digit 2. All training digits are in the Calibri font family.

Notice in Figure 11 that the digits 2, 7 and 8 result in the lowest average prediction error response across all five font families. It appears that despite undergoing sequential training procedures to learn new digit class representations, the old information about 7 and 8 is not forgotten.

5. CONCLUSIONS

We see in this work supporting evidence for the hypothesis that HTM networks are capable of one-shot and incremental learning. These learning properties can be attributed to the spatial pooler's ability to generalize spatial information and the distributed local knowledge representation combined with local Hebbian learning rules in the temporal memory region respectively. Note that these learning properties are inherent to the spatial pooler and temporal memory itself. These properties can be expected to appear completely regardless of the networks input data. Analogous to the human brain and its input fibers, the spatial pooler and temporal memory don't actually ever have any contact with the original data source. They only ever see patterns of binary vector representations analogous to patterns of electrical activity on the physical brain's input fibers.

Note that this works fails to provide a large breadth of testing for the desired learning properties. It remains unclear if these properties would persist in more robust training scenarios. It is likely that the incremental learning properties of temporal memory are going to entirely depend upon the parameter settings including the representational size of the network (to account for intermingling of representations) and things like the magnitude of synapse permanence decrements due to inactivity. As is repeated several times previously, this work is not meant to introduce a fully functioning classification system nor provide a concrete answer to the proposed hypothesis. This work is meant to inspire future research in this domain especially as HTM continues to grow and develop. Most importantly, to reach a fully functioning classification system, one would need to assess the spatial separability of the digit-specific neuronal prediction states that HTM produces.

References

[1] Basic visual pathway. URL http://www.bioon.com/bioline/neurosci/course/basvis.html.

[2] Anatomy. URL https://ocularmanifestofsystemicdisease.weebly.com/anatomy.html.

[3] The primary visual cortex by matthew schmolesky. URL http://webvision.med.utah.edu/book/part-ix-psychophysics-of-vision/the-primary-visual-cortex/.

[4] Subutai Ahmad and Jeff Hawkins. Properties of sparse distributed representations and their application to hierarchical temporal memory. *CoRR*, abs/1503.07469, 2015. URL http://arxiv.org/abs/1503.07469.

[5] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134âĂŞ147, 2017. doi: 10.1016/j.neucom.2017.04.070.

[6] Yoonsuck Choe. *Hebbian Learning*, pages 1305–1309. Springer New York, New York, NY, 2015. ISBN 978-1-4614-6675-8. doi: 10.1007/978-1-4614-6675-8_672. URL https://doi.org/10.1007/978-1-4614-6675-8_672.

[7] Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *CoRR*, abs/1512.05463, 2015. URL http://arxiv.org/abs/1512.05463.

[8] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. The htm spatial pooler: a neocortical algorithm for online sparse distributed coding. *bioRxiv*, 2016. doi: 10.1101/085035. URL https://www.biorxiv.org/content/early/2016/11/02/085035.

[9] Srdjan D Antic, Wen-Liang Zhou, Anna Moore, Shaina Short, and Katerina Oikonomou. The decade of the dendritic nmda spike. 88:2991–3001, 11 2010.

[10] Graeme W. Davis. Homeostatic control of neural activity: From phenomenology to molecular design. *Annual Review of Neuroscience*, 29(1):307–323, 2006. doi: 10.1146/annurev.neuro.28.061604.

135751. URL https://doi.org/10.1146/annurev.neuro.28.061604.135751. PMID: 16776588.

[11] James J. Dicarlo, Davide Zoccolan, and Nicole C. Rust. How does the brain solve visual object recognition? *Neuron*, 73(3):415âĂŞ434, 2012. doi: 10.1016/j.neuron.2012.01.010.

[12] D. E. Erb and J. T. Povlishock. Neuroplasticity following traumatic brain injury: a study of gabaergic terminal loss and recovery in the cat dorsal lateral vestibular nucleus. *Experimental Brain Research*, 83(2):253–267, Jan 1991. ISSN 1432-1106. doi: 10.1007/BF00231151. URL https://doi.org/10.1007/BF00231151.

[13] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10, 2016. doi: 10.3389/fncir.2016.00023.

[14] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, 2004. ISBN 0805074562.

[15] htm community. htm-community/comportex, Sep 2016. URL https://github.com/htm-community/comportex.

[16] JeffryS. Isaacson and Massimo Scanziani. How inhibition shapes cortical activity. *Neuron*, 72 (2):231 – 243, 2011. ISSN 0896-6273. doi: https://doi.org/10.1016/j.neuron.2011.09.027. URL http://www.sciencedirect.com/science/article/pii/S0896627311008798.

[17] L. C. Jain and L. R. Medsker. *Recurrent Neural Networks: Design and Applications*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1999. ISBN 0849371813.

[18] J. H. Kaas. Neocortex in early mammals and its subsequent variations. *Ann. N. Y. Acad. Sci.*, 1225:28–36, Apr 2011.

[19] Jon H Kaas. Topographic maps are fundamental to sensory processing. *Brain Research Bulletin*, 44(2):107 – 112, 1997. ISSN 0361-9230. doi: https://doi.org/10.1016/S0361-9230(97)00094-4. URL http://www.sciencedirect.com/science/article/pii/S0361923097000944.

[20] Victor AF Lamme, Hans SupÃĺr, and Henk Spekreijse. Feedforward, horizontal, and feedback processing in the visual cortex. *Current Opinion in Neurobiology*, 8(4):529 – 535, 1998. ISSN 0959-4388. doi: https://doi.org/10.1016/S0959-4388(98)80042-1. URL http://www.sciencedirect.com/science/article/pii/S0959438898800421.

[21] M. E. Larkum, J. J. Zhu, and B. Sakmann. A new cellular mechanism for coupling inputs arriving at different cortical layers. *Nature*, 398(6725):338–341, Mar 1999.

[22] Yann Lecun, LÃl'on Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[23] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[24] G. Major, M. E. Larkum, and J. Schiller. Active properties of neocortical pyramidal neuron dendrites. *Annu. Rev. Neurosci.*, 36:1–24, Jul 2013.

[25] James Mnatzaganian, Ernest Fokoué, and Dhireesha Kudithipudi. A mathematical formalization of hierarchical temporal memory cortical learning algorithm's spatial pooler. *CoRR*, abs/1601.06116, 2016.

[26] Vernon B. Mountcastle. An organizing principle for cerebral function: The unit model and the distributed system. In Gerald M. Edelman and Vernon V. Mountcastle, editors, *The Mindful Brain*, pages 7–50. MIT Press, Cambridge, MA, 1978.

[27] Randolph Nudo. Recovery after brain injury: Mechanisms and principles. 7:887, 12 2013.

[28] Numenta. numenta/htm.java, Sep 2017. URL https://github.com/numenta/htm.java.

[29] Clyde W. Oyster. *The human eye: structure and function*. Sinauer Associates, 2006.

[30] L. Petreanu, T. Mao, S. M. Sternson, and K. Svoboda. The subcellular organization of neocortical excitatory connections. *Nature*, 457(7233):1142–1145, Feb 2009.

[31] Scott Purdy. Encoding data for HTM systems. *CoRR*, abs/1602.05925, 2016. URL http://arxiv.org/abs/1602.05925.

[32] Dale Purves. Types of eye movements and their functions, Jan 1970. URL https://www.ncbi.nlm.nih.gov/books/NBK10991/.

[33] J. C. Rah, E. Bas, J. Colonell, Y. Mishchenko, B. Karsh, R. D. Fetter, E. W. Myers, D. B. Chklovskii, K. Svoboda, T. D. Harris, and J. T. Isaac. Thalamocortical input onto layer 5 pyramidal neurons measured using quantitative large-scale array tomography. *Front Neural Circuits*, 7:177, 2013.

[34] Srikanth Ramaswamy and Henry Markram. Anatomy and physiology of the thick-tufted layer 5 pyramidal neuron. *Frontiers in Cellular Neuroscience*, 9, 2015. doi: 10.3389/fncel.2015.00233.

[35] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[36] J. Schiller and Y. Schiller. NMDA receptor-mediated dendritic spikes and coincident signal amplification. *Curr. Opin. Neurobiol.*, 11(3):343–348, Jun 2001.

[37] J. Schiller, G. Major, H. J. Koester, and Y. Schiller. NMDA spikes in basal dendrites of cortical pyramidal neurons. *Nature*, 404(6775):285–289, Mar 2000.

[38] Nelson Spruston. Spruston n. pyramidal neurons: dendritic structure and synaptic integration. nat rev neurosci 9: 206-221. 9:206–21, 04 2008.

[39] Matthew Taylor, Scott Purdy, breznak, Chetan Surpur, Austin Marshall, David Ragazzi, Subutai Ahmad, numenta ci, Andrew Malta, Pascal Weinberger, Akhila, Marcus Lewis, Richard Crowder, Marion Le Borgne, Yuwei, Christopher Simons, Ryan J. McCall, Mihail Eric, Utensil Song, keithcom, Nathanael Romano, Sagan Bolliger, vitaly krugl, hernandezurbina, James Bridgewater, Ian Danforth, Jared Weiss, Tom Silver, David Ray, and Luiz Scheinkman. numenta/nupic: 1.0.3, September 2017. URL https://doi.org/10.5281/zenodo.891005.

[40] S. B. Udin and J. W. Fawcett. Formation of topographic maps. *Annual Review of Neuroscience*, 11 (1):289–327, 1988. doi: 10.1146/annurev.ne.11.030188.001445. URL https://doi.org/10.1146/annurev.ne.11.030188.001445. PMID: 3284443.

[41] D.B. Webster, A.N. Popper, and R.R. Fay. *The Mammalian auditory pathway: neuroanatomy*. Springer handbook of auditory research. Springer-Verlag, 1992. ISBN 9780387976785. URL https://books.google.com/books?id=x6HwAAAAMAAJ.

[42] Yumiko Yoshimura, Hiromichi Sato, Kazuyuki Imamura, and Yasuyoshi Watanabe. Properties of horizontal and vertical inputs to pyramidal cells in the superficial layers of the cat visual cortex. *Journal of Neuroscience*, 20(5):1931–1940, 2000. ISSN 0270-6474. URL http://www.jneurosci.org/content/20/5/1931.

[43] Karen Zito and Karel Svoboda. Activity-dependent synaptogenesis in the adult mammalian cortex. *Neuron*, 35(6):1015âĂŞ1017, 2002. doi: 10.1016/s0896-6273(02)00903-0.