4-15-2016

A Mathematical Formalization of Hierarchical Temporal Memory's Spatial Pooler for use in Machine Learning

James W. Mnatzaganian
jwm3457@rit.edu

A Mathematical Formalization of Hierarchical Temporal Memory's Spatial Pooler for use in Machine Learning

James W. Mnatzaganian

A Mathematical Formalization of Hierarchical Temporal Memory's Spatial Pooler for use in Machine Learning

James W. Mnatzaganian
April 15, 2016

A Thesis Submitted
in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in
Computer Engineering

R·I·T | Kate Gleason
College of ENGINEERING

Department of Computer Engineering

A Mathematical Formalization of Hierarchical Temporal Memory's Spatial Pooler for use in Machine Learning

James W. Mnatzaganian

Committee Approval:

Dr. Dhireesha Kudithipudi *Advisor* Date
Associate Professor

Dr. Ernest Fokoué Date
Associate Professor

Dr. Andreas Savakis Date
Professor

To my loving wife, Haley.

Thank you for believing in me, and being there with me through the thick of it all.

I love you.

Acknowledgments

I firstly and formally am thankful for my heavenly father. This work would not have been possible without His gracious hand guiding me all the way. John 14:14 (NIV) "You may ask me for anything in my name, and I will do it."

I am grateful for my advisor, Dr. Dhireesha Kudithipudi. Thank you for your guidance and support. I was truly grateful to have you as my advisor.

I am thankful for my committee members Dr. Ernest Fokoué and Dr. Andreas Savakis. Thank you Dr. Fokoué for the mathematical support. Thank you Dr. Savakis for your time, and for helping keep my expectations realistic.

I am thankful for our research team at the NanoComputing Research Lab. All of you have helped me more than you will ever realize. James Thesing, thank you for all of the times you diligently listened to me, and for your input. Levs Dolgovs, thank you for our late night discussions. Qutaiba Saleh, thank you for our discussions. Dr. Cory Merkel, thank you for your general help on everything. Amanda Hartung, thank you for your extreme diligence – keep up the good work! Lennard Streat thanks for all of the HTM input. Everyone else who I did not mention, thank you all for your general friendship and support. This truly was a team effort! ☺

I am very thankful for the staff at RIT's research computing. Without their support and the use of their equipment, my experiments would still be running!

I am thankful for Kevin Gomez of Seagate Technology for his support.

I am thankful for the Numenta team for their input.

I am also thankful for my friends that provided help and support. In particular, Matthew Filmer – Thanks for reminding me that MS ! = PhD and for the LaTeX help.

Lastly, I am extremely grateful for my family. Without all of you keeping me straight, my head would have come off!! I love you all dearly. You have all supported me and believed in me from the beginning. Thank you for not only tolerating me, but showing me much love, during my season of business.

Abstract

Hierarchical temporal memory (HTM) is an emerging machine learning algorithm, with the potential to provide a means to perform predictions on spatiotemporal data. The algorithm, inspired by the neocortex, consists of two primary components, namely the spatial pooler (SP) and the temporal memory (TM). The SP is utilized to map similar inputs into generalized sparse distributed representations (SDRs). Those SDRs are then utilized by the TM, which performs sequence learning and prediction. One challenge with HTM is ensuring that proper SDRs are generated from the SP. If the SDRs are not generalizable, the TM will not be able to make proper predictions.

This work focuses on the SP and its corresponding output SDRs. A single unifying mathematical framework was created for the SP. The primary learning mechanism was explored, where a maximum likelihood estimator for determining the degree of permanence update was proposed. The boosting mechanisms were studied and found to only be relevant during the initial few iterations of the network. Observations were made relating HTM to well-known algorithms such as competitive learning and attribute bagging. Methods were provided for using the SP for classification as well as dimensionality reduction. Empirical evidence verified that given the proper parameterizations, the SP may be used for feature learning.

Similarity metrics were created for scoring the SDRs produced by the SP. The overlap metric proved that the SP is extremely robust to noise. The SP was able to produce similar outputs for a given input, provided the noise did not cause the input to change classes. This overlap metric was further utilized to create a classifier for novelty detection. The SP proved to be able to withstand more noise than the well-known support vector machine (SVM).

Contents

List of Figures

List of Tables

List of Symbols

Term	Description
$\exists!$	Uniqueness quantification.
n	Number of patterns (samples).
p	Number of inputs (features) in a pattern.
m	Number of columns.
q	Number of proximal synapses per column.
ϕ_+	Permanence increment amount.
ϕ_-	Permanence decrement amount.
ϕ_δ	Window of permanence initialization.
ρ_d	Proximal dendrite segment activation threshold.
ρ_s	Proximal synapse activation threshold.
ρ_c	Desired column activity level.
κ_a	Minimum activity level scaling factor.
κ_b	Permanence boosting scaling factor.
β_0	Maximum boost.
τ	Duty cycle period.
s	Integer index bounded by $[0, n)$.
r	Integer index bounded by $[0, p)$.
i	Integer index bounded by $[0, m)$.
j	Integer index bounded by $[0, m)$.
k	Integer index bounded by $[0, q)$.
$\boldsymbol{U}$	Set of inputs for all patterns. $\boldsymbol{U} \in \{0,1\}^{n \times p}$.
$\boldsymbol{U}^{(tr)}$	Set of inputs for all training patterns. $\boldsymbol{U}^{(tr)} \in \{0,1\}^{n \times p}$. n in this context represents the number of rows in $\boldsymbol{U}^{(tr)}$.
$\boldsymbol{U}^{(te)}$	Set of inputs for all testing patterns. $\boldsymbol{U}^{(te)} \in \{0,1\}^{n \times p}$. n in this context represents the number of rows in $\boldsymbol{U}^{(te)}$.
$\boldsymbol{W}$	Set of SP output SDRs for all patterns. $\boldsymbol{W} \in \{0,1\}^{n \times m}$.

Term	Description
$\boldsymbol{W}^{(tr)}$	Set of SP output SDRs for all training patterns. $\boldsymbol{W}^{(tr)} \in \{0,1\}^{n \times m}$. n in this context represents the number of rows in $\boldsymbol{U}^{(tr)}$.
$\boldsymbol{W}^{(te)}$	Set of SP output SDRs for all testing patterns. $\boldsymbol{W}^{(te)} \in \{0,1\}^{n \times m}$. n in this context represents the number of rows in $\boldsymbol{U}^{(te)}$.
$\vec{\boldsymbol{c}}$	Set of all columns indices. $\vec{\boldsymbol{c}} \in \mathbb{Z}^{1 \times m}, \vec{\boldsymbol{c}} \in [0, m)$.
$\boldsymbol{\Lambda}$	Source column indices for each proximal synapse on each column. $\boldsymbol{\Lambda} \in \{r\}^{m \times q}$.
$\boldsymbol{\Phi}$	Set of permanences for each column. $\boldsymbol{\Phi} \in \mathbb{R}^{m \times q}$.
$\boldsymbol{X}$	Set of inputs for each column. $\boldsymbol{X} \in \{0,1\}^{m \times q}$.
$\overline{\boldsymbol{X}}$	Overall mean of $\boldsymbol{X}$.
$\vec{\boldsymbol{b}}$	Set of boost values for all columns. $\vec{\boldsymbol{b}} \in \mathbb{R}^{1 \times m}$.
$\boldsymbol{H}$	Neighborhood mask for all columns. $\boldsymbol{H} \in \{0,1\}^{m \times m}$.
$\boldsymbol{Y}$	Bit-mask for the proximal synapses' activations. $\boldsymbol{Y} \in \{0,1\}^{m \times q}$.
$\vec{\boldsymbol{z}}$	Attribute mask. $\vec{\boldsymbol{z}} \in \{0,1\}^{1 \times p}$.
$\vec{\boldsymbol{\alpha}}$	Overlap for all columns. $\vec{\boldsymbol{\alpha}} \in \{0,1\}^{1 \times m}$.
$\vec{\hat{\boldsymbol{\alpha}}}$	Sum of the active connected proximal synapses for all columns. $\vec{\hat{\boldsymbol{\alpha}}} \in \mathbb{Z}^{1 \times m}$.
$\vec{\boldsymbol{\gamma}}$	ρ_c-th largest overlap (lower bounded by one) in the neighborhood of $\vec{\boldsymbol{c}}_i \ \forall i$. $\vec{\boldsymbol{\gamma}} \in \mathbb{Z}^{1 \times m}$.
$\boldsymbol{\delta\Phi}$	Proximal synapses' permanence update amount, following the original terminology. $\boldsymbol{\delta\Phi} \in \mathbb{R}^{m \times q}$.
$\boldsymbol{\delta\Psi}$	Proximal synapses' permanence update amount, following the new terminology. $\boldsymbol{\delta\Psi} \in \mathbb{R}^{m \times q}$.
$\overrightarrow{\boldsymbol{\eta^{(a)}}}$	Set of active duty cycles for all columns. $\overrightarrow{\boldsymbol{\eta^{(a)}}} \in \mathbb{R}^{1 \times m}$.
$\overrightarrow{\boldsymbol{\eta^{(o)}}}$	Set of overlap duty cycles for all columns. $\overrightarrow{\boldsymbol{\eta^{(o)}}} \in \mathbb{R}^{1 \times m}$.
$\overrightarrow{\boldsymbol{\eta^{(min)}}}$	Set of minimum active duty cycles for all columns. $\overrightarrow{\boldsymbol{\eta^{(min)}}} \in \mathbb{R}^{1 \times m}$.

Term	Description
$\boldsymbol{D}$	Distance between an SP column and its corresponding connected synapses' source columns. $\boldsymbol{D} \in \mathbb{R}^{m \times q}$.
σ_0	Inhibition radius.
$\vec{\hat{\boldsymbol{c}}}$	Set of active columns. $\vec{\hat{\boldsymbol{c}}} \in \{0,1\}^{1 \times m}$.
$\vec{\hat{\boldsymbol{\phi}}}$	Set of learned attribute probabilities. $\vec{\hat{\boldsymbol{\phi}}} \in (0,1)^{1 \times p}$.
$\vec{\hat{\boldsymbol{u}}}$	Reconstructed input. $\vec{\hat{\boldsymbol{u}}} \in \{0,1\}^{1 \times p}$.
$\hat{\theta}_{MLE}$	MLE of θ.
$\boldsymbol{J}$	Matrix of ones with dimensions $m \times q$.
t	Scaling factor equal to $m \times q$.
κ	Scaling parameter defined such that $\frac{\kappa}{\overline{\boldsymbol{X}}} \in [0,1]$ and $\frac{\kappa}{1-\overline{\boldsymbol{X}}} \in [0,1]$.
θ	Probability of an input being active.
$\vec{\boldsymbol{ic}}_r$	Event of input r connecting to column i. $\vec{\boldsymbol{ic}} \in \{0,1\}^{1 \times p}$.
$\boldsymbol{AC}_{i,k}$	Event that proximal synapse k is active and connected on column i. $\boldsymbol{AC} \in \{0,1\}^{m \times q}$.
$\vec{\boldsymbol{\lambda}}$	Random vector governing the count of connections between each input and all columns. $\vec{\boldsymbol{\lambda}} \in \mathbb{Z}^{1 \times p}, \vec{\boldsymbol{\lambda}} \in [0, q+m]$.
λ'	Random variable governing the number of unconnected inputs.
$\vec{\boldsymbol{ai}}_i$	Random variable governing the number of active inputs on column i. $\vec{\boldsymbol{ai}} \in \mathbb{Z}^{1 \times m}, \vec{\boldsymbol{ai}} \in [0, q]$.
a	Random variable governing the average number of active inputs on a column.
$\overline{\boldsymbol{ac}}\vec{\boldsymbol{c}}_i$	Random variable governing the number of active and connected proximal synapses for column i. $\overline{\boldsymbol{ac}}\vec{\boldsymbol{c}} \in \mathbb{Z}^{1 \times m}, \overline{\boldsymbol{ac}}\vec{\boldsymbol{c}} \in [0, q]$.
at	Random variable governing the number of columns having at least ρ_d active proximal synapses.
act	Random variable governing the number of columns having at least ρ_d active connected proximal synapses.

Term	Description
π_x	Random variable that is equal to the overall mean of $\mathbb{P}(\boldsymbol{X})$.
π_{ac}	Random variable that is equal to the overall mean of $\mathbb{P}(\boldsymbol{AC})$.
$\vec{\boldsymbol{\pi}}_w$	Random variable that represents the mean across the rows of $\boldsymbol{W}^{(tr)}$.
μ_u	Uniqueness metric. A metric for evaluating the quality of output SDRs of the SP. A value of zero indicates that all SDRs are identical and a value of one indicates that all SDRs are unique.
μ_o	Overlap metric. A metric for evaluating the quality of output SDRs of the SP. A value of one indicates that all SDRs are identical and a value of zero indicates that all SDRs are unique.
ao	Average overlap across all pair-wise vectors in $\boldsymbol{W}$.
mo	Maximum overlap on two given vectors in $\boldsymbol{W}$.
$\vec{\boldsymbol{w}}_0$	Base class representation for all of the SDRs in $\boldsymbol{W}^{(tr)}$.

List of Functions

Name	Description
$\mathrm{I}(k)$	Indicator function. Returns '1' if parameter k is true and '0' otherwise.
$\mathrm{B}er$	Bernoulli distribution.
$\mathrm{B}in(k; n, p)$	PMF of a binomial distribution, where k is the number of successes, n is the number of trials, and p is the success probability in each trial.
$\mathrm{m}in(\vec{\boldsymbol{v}})$	Returns the minimum value in $\vec{\boldsymbol{v}}$.
$\mathrm{m}ax(\vec{\boldsymbol{v}})$	Returns the maximum value in $\vec{\boldsymbol{v}}$.
$\mathrm{k}max(S, k)$	Returns the k-th largest element of S.
$clip(\boldsymbol{M}, lb, ub)$	Clips all values in the matrix $\boldsymbol{M}$ outside of the range $[lb, ub]$ to lb if the value is less than lb, or to ub if the value is greater than ub.
$\mathrm{u}pdate\_active\_duty\_cycle(\vec{\boldsymbol{c}})$	Updates the moving average duty cycle for the active duty cycle for each $\vec{\boldsymbol{c_i}} \in \vec{\boldsymbol{c}}$.
$\mathrm{u}pdate\_overlap\_duty\_cycle(\vec{\boldsymbol{c}})$	Updates the moving overlap duty cycle for the overlap duty cycle for each $\vec{\boldsymbol{c_i}} \in \vec{\boldsymbol{c}}$.
$\beta\left(\overrightarrow{\boldsymbol{\eta_i^{(a)}}}, \overrightarrow{\boldsymbol{\eta_i^{(min)}}}\right)$	Updates the boost values for each $\vec{\boldsymbol{c_i}} \in \vec{\boldsymbol{c}}$.
$\mathrm{d}(x, y)$	Distance function. Returns the distance between x and y.
$\mathrm{p}os(c, r)$	Returns the position of the column indexed at c, located r regions away from the current region.
$\mathrm{h}(\vec{\boldsymbol{c}}_i)$	Neighborhood function. Returns the neighbors for $\vec{\boldsymbol{c}}_i$.

Acronyms

2D two-dimensional

3D three-dimensional

ANN artificial neural network

API application program interface

CLA cortical learning algorithm

CNN convolutional neural network

CV cross-validation

HTM hierarchical temporal memory

i.i.d. independent and identically distributed

mHTM math hierarchical temporal memory

MLE maximum-likelihood estimator

MNIST modified National Institute of Standards and Technology

NuPIC Numenta platform for intelligent computing

PMF probability mass function

SDR sparse distributed representation

SOM self-organizing map

SP spatial pooler

SPD spatial pooler dataset

SPNDC spatial pooler novelty detection classifier

SVM support vector machine

TM temporal memory

TP temporal pooler

VQ vector quantization

Chapter 1

Introduction

Hierarchical temporal memory (HTM) is a machine learning algorithm that was inspired by the neocortex and designed to learn sequences and make predictions. In its idealized form, it should be able to produce generalized representations for similar inputs. Given time-series data, HTM should be able to use its learned representations to perform a type of time-dependent regression. Such a system would prove to be incredibly useful in many applications utilizing spatiotemporal data. One instance for using HTM with time-series data was recently demonstrated by Cui et al. [1], where HTM was used to predict taxi passenger counts. The use of HTM in other applications remains unexplored, largely due to the evolving nature of HTM's algorithmic definition. Additionally, the lack of a formalized mathematical model hampers its prominence in the machine learning community. This work aims to bridge the gap between a neuroscience inspired algorithm and a math-based algorithm by constructing a purely mathematical framework around HTM's original algorithmic definition.

HTM models, at a high-level, some of the structures and functionality of the neocortex. Its structure follows that of cortical minicolumns, where an HTM region is comprised of many columns, each consisting of multiple cells. One or more regions form a level. Levels are stacked hierarchically in a tree-like structure to form the full network. Within HTM, connections are made via synapses, where both proximal and distal synapses are utilized to form feedforward and neighboring connections,

respectively.

The current version of HTM is the predecessor to HTM cortical learning algorithm (CLA) [2]. In the current version of HTM the two primary algorithms are the spatial pooler (SP) and the temporal memory (TM). The SP is responsible for taking an input, in the format of a sparse distributed representation (SDR), and producing a new SDR. In this manner, the SP can be viewed as a mapping function from the input domain to a new feature domain. In the feature domain a single SDR should be used to represent similar SDRs from the input domain. The algorithm is a type of unsupervised competitive learning algorithm that uses a form of vector quantization (VQ) resembling self-organizing maps (SOMs). The TM is responsible for learning sequences and making predictions. This algorithm follows Hebb's rule [3], where connections are formed between cells that were previously active. Through the formation of those connections a sequence may be learned. The TM can then use its learned knowledge of the sequences to form predictions.

HTM originated as an abstraction of the neocortex; as such, it does not have an explicit mathematical formulation. Without a mathematical framework, it is difficult to understand the key characteristics of the algorithm and how it can be realized. In general, very little work exists regarding the mathematics behind HTM. Hawkins et al. [4] recently provided a starting mathematical formulation for the TM, but no mentions to the SP were made. Lattner [5] provided an initial insight about the SP, by relating it to VQ. He additionally provided some equations governing computing overlap and performing learning; however, those equations were not generalized to account for local inhibition. Byrne [6] began the use of matrix notation and provided a basis for those equations; however, certain components of the algorithm, such as boosting, were not included. Leake et al. [7] provided some insights regarding the initialization of the SP. He also provided further insights into how the initialization may affect the initial calculations within the network; however, his focus was largely on the network

initialization. The goal of this work is to provide a complete mathematical framework for HTM's SP. Additionally, this work aims to provide a basis for utilizing HTM's SP in machine learning.

1.1 Research Statement and Contributions

This work was created with the belief that *formalized spatiotemporal algorithms, inspired by the neocortex, will advance the machine learning of tomorrow.* As such, the primary objective of this work was to formalize the SP, and to apply that formalization in the field of machine learning.

The aforementioned formalization was created in Chapter 4. This mathematical framework took into consideration all aspects of the SP and brought them together under a single unifying framework. In addition to the primary operation of the SP, the initialization as well as the boosting mechanisms were described. The initialization work provides a probabilistic approach for determining the initial expected output of the SP; thereby, creating an easy technique for choosing suitable parameters. The model for boosting provided a premises for evaluating the effectiveness of the strategy, where it was found that the current boosting technique will typically make little difference in the SP's learned representations.

In addition to those contributions, the framework provided a foundation for studying the primary learning mechanism of the SP. Through that study, a plausible explanation for the origin of the permanence update amount was created. An estimator was developed to optimize the permanence increment and decrement amounts. The SP was also described in terms of traditional machine learning algorithms, where it was found that the SP is similar to attribute bagging with a competitive learning network as the base learner.

An open-source Python implementation of the aforementioned framework was developed (see Section 5.1). This implementation of the SP was unique, because it

specifically allowed the SP to be utilized in the same manner as any other comparable machine learning algorithm. This implementation will allow future researchers to easily utilize the SP in their present or future studies.

Using the created mathematical framework and software solution, the SP was demonstrated in a number of areas within machine learning (see Section 5.4 and Section 6.2). In addition to the traditional use case of classification, the SP was used to perform feature extraction, dimensionality reduction, and novelty detection. A method was also created for using the SP's output, along with its permanences, to recreate the input.

Two methods were created to evaluate the quality of the SP's SDRs (see Section 6.1). Those metrics, the uniqueness metric and the overlap metric, proved to provide a suitable means for evaluating the SP's SDRs. Additionally, those metrics were used to show that the SP is extremely robust to noise and is able to create *similar* SDRs for *similar* inputs. The result of the latter provides confirmation that the SP algorithm is able to perform is primary goal of feature mapping.

1.2 Document Structure

Chapter 2 discusses the necessary background relating to HTM. Additional information is provided discussing the algorithm's history. Chapter 3 provides an overview of the SP. All components utilized by the SP are explained, in their original terms. A high-level overview of the operations of the SP is provided, with corresponding pseudocode.

Chapter 4 introduces the mathematical model created for the SP. The notation utilized by this work is explained. Additionally, insights into the SP's primary learning mechanism are provided.

Chapter 5 provides detailed instructions on how to begin using the SP for machine learning related tasks. Data encoding is explained, with details provided for various

encoders. An explanation on how to train the SP is provided. Feature learning is discussed, with a demonstration of multi-class classification.

Chapter 6 discusses how to evaluate the SDRs produced by the SP. One of the introduced metrics is used to create a classifier. That classifier is then demonstrated in the context of novelty detection.

Chapter 7 summaries this work. It also provides some final remarks regarding other aspects of the SP algorithm.

Chapter 2

Background

2.1 Memory-Prediction Framework

A theory of brain function, known as the memory-prediction framework, was developed by Hawkins [8] to describe the manner in which the mammalian neocortex stores and utilizes memory. The theory proposes the notion of a single algorithm to describe the processing of all cortical information.

In this theory, a bi-directional, tree shaped hierarchy of regions are used to process data. Inputs enter at the lowest levels of the hierarchy, and information flows through the system in both a feedforward and feedback manner. In each region, an invariant representation of the information is created and stored in the form of patterns. This type of data storage is an autoassociative memory, where patterns are retrieved based on similarities to past patterns.

Each time an input is presented to the system, the current region infers as to what the input represents, passing its knowledge up the hierarchy, representing a feedforward flow. The region, storing past-occurrences of feedforward inputs, is able to combine feedback from higher levels with its memory and the current feedforward input to make a prediction about the next input.

Through the changing of input, each region is able to learn sequences of patterns. The higher regions are able to use the representations of the lower regions to learn

Sunday : 11100000000000000000000000
Monday : 01110000000000000000000000
...
Saturday : 11000000000000000000000001

Figure 2.1: Example SDR encoding for the days of the week. In this example, Sunday is similar to both Saturday and Monday, denoting a periodic pattern.

sequences of higher-order objects. This process allows higher levels to become more invariant, as less details are propagated forward through the network.

2.2 Sparse Distributed Representation (SDR)

HTM uses the concept of sparse distributed representations (SDRs), where a given input is encoded into a quantized binary representation. This binary representation has a small number of active bits. SDRs are primarily used to allow a type of similarity between inputs to occur.

For example, an SDR of the days of the week, might look something like the vectors shown in Figure 2.1. In that figure, Sunday shares two bits with Monday, implying that Sunday and Monday are *similar*. In this case, "similar" represents days of the week that are close in order. Additionally, Sunday shares two bits with Saturday, thus denoting a periodic pattern, as the first and last elements are similar.

If the number of overlapping bits were to increase, the resolution would also increase, allowing for an increase in step size between encodings. Additionally, an increase in bit overlap improves the robustness of the system, as only a single bit overlap is required to classify two inputs as similar.

SDRs are utilized by an HTM in a number of places. Before an HTM is able to use input data, it must first be encoded by an appropriate encoder (see Section 5.2 for more details). The encoder is used to map the input from its current form into its corresponding SDR form. While encoders are not explicitly part of an HTM they are a necessary requirement for the system.

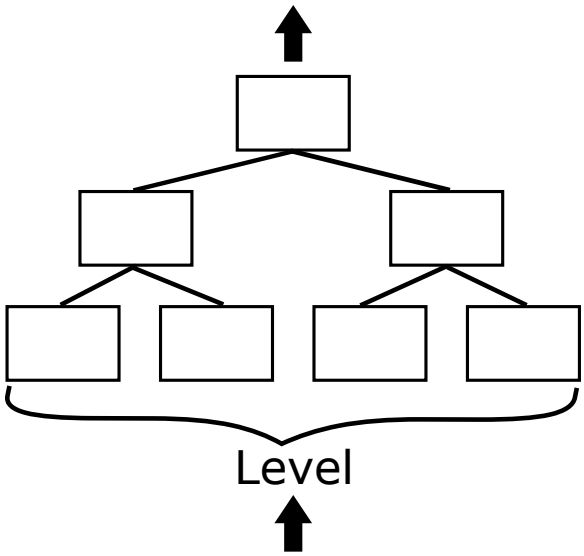


Figure 2.2: A generic HTM, consisting of three levels. The first level is at the bottom of the hierarchy and the last level is at the top of the hierarchy. The inputs are fed bottom-up, with the final output produced from the last level.

Within an HTM, SDRs are used whenever data is stored or read. This most prominently appears as an output from the SP, the current state of the TM, and a prediction from the TM.

2.3 Hierarchical Temporal Memory (HTM)

Hierarchical temporal memory (HTM) is a machine learning algorithm, created by Hawkins and George [9] that is based on the memory-prediction framework. A basic depiction of the system is shown in Figure 2.2. That system is comprised of three levels. The inputs to the network are fed bottom-up from level one up through level three. The overall output of the system is produced at the last level.

Originally, HTM, was developed as a system capable of performing invariant visual pattern recognition [10]. That version of HTM, known as Zeta 1, was built upon a Bayesian framework. After Zeta 1, Zeta 2 was developed.

The Zeta algorithms were succeeded by the cortical learning algorithms (CLAs). The CLAs were defined to be a category of algorithms utilized in HTM, explicitly

known as the HTM CLAs [2]. The CLAs were comprised of the spatial pooler (SP) and the temporal pooler (TP). Following HTM CLA came the current version of HTM, which is simply known as HTM [4]. This version of HTM is similar to HTM CLA, with the most noticeable difference being that the TP is replaced by temporal memory (TM).

HTM is under active development and as such, its details are changing. For purposes of clarity, this work primarily follows the definitions from the original HTM CLA whitepaper [2]. To ensure that the most recent details are provided, the latest version of HTM was used as a reference. All areas of ambiguity are explained as they appear. Terminology is used to follow the current version of HTM, to help remove any future confusion.

2.3.1 Zeta Algorithms

Zeta 1 HTM refers to the HTM described in [10]. This type of HTM is constructed with nodes. Each box in Figure 2.2 represents a node, with multiple nodes comprising a level. The nodes are the basic building blocks of Zeta 1 HTM and include both the algorithmic and memory components of the system. Nodes are used to learn spatial invariant representations of input spaces.

The nodes have two phases of operations. The first phase, learning, is where the node creates internal representations of the input. The second phase, sensing / inference, is where the node produces an output for the input. During the learning phase, three operations occur: memorization of patterns, learning transition probabilities, and temporal grouping. In this context, a "pattern" refers to the input the node receives at a given instance, in time.

During the memorization phase, unique occurrences of each pattern are labeled and stored. During the learning transition probabilities phase, a Markov chain is maintained. Each vertex in the chain corresponds to a stored pattern. The link

between two vertices corresponds to the probability of the pattern occurring. The probability is simply a unity normalized value corresponding to the percentage the transition from one pattern to another pattern occurred.

During the temporal grouping phase, patterns are clustered using agglomerative hierarchical clustering. Each cluster in the hierarchical clustering dendrogram corresponds to a set of patterns that are likely to follow one another, in time. Those clusters are known as temporal groups.

Once the learning phase has completed, the sensing / inference phase may occur. In this phase, each node produces an output based on an instantaneous input. The closeness of the input pattern to the patterns stored in memory is calculated. The closeness is measured by the Euclidean distance between all vertices in a temporal group. The node's output is a unity normalized vector containing the probabilities of the input pattern matching each temporal group.

Chapter 3

Spatial Pooler (SP) Overview

HTM is comprised of multiple levels, with each level consisting of one or more regions. The term level refers to the HTM's structure, where a level contains all components within a single rank of the hierarchy. A region refers to the functionality within the level. Each region is comprised of one or more columns consisting of one more cells. The cells act as the fundamental functional unit within an HTM. An example HTM architecture is depicted in Figure 3.1. This chapter explains those individual components, as well as the spatial pooler (SP) and its role within HTM.

3.1 HTM Components

An HTM is comprised of a number of components, many of which follow a naming convention inspired from neuroscience. Structurally, an HTM is a tree-shaped hierarchy of levels. Each level consists of one or more regions, which are used to carry out the basic operation of the system. A region is defined to create an SDR of an input, represent that input in the context of previous inputs, and perform inference based on that new representation [2]. In practice, a region is used as a building block to perform one distinct function, namely encoding the input, performing spatial pooling, or executing the temporal memory operations. As such, to represent a true region

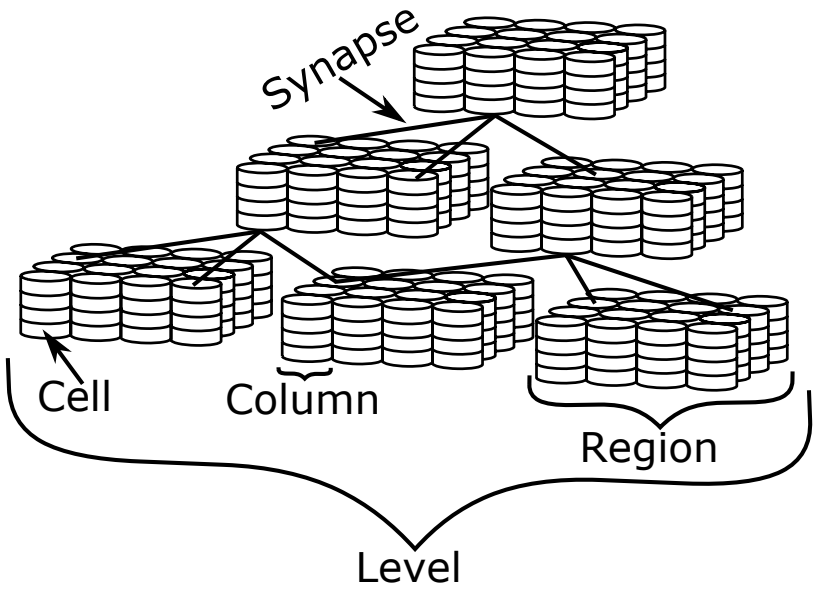


Figure 3.1: An HTM consisting of three levels. The first level is at the bottom of the hierarchy and the last level is at the top of the hierarchy. The inputs are fed bottom-up, with the final output produced from the last level. Each level contains one or more regions comprised of columns of cells. Connections between cells, within a region, also occur.

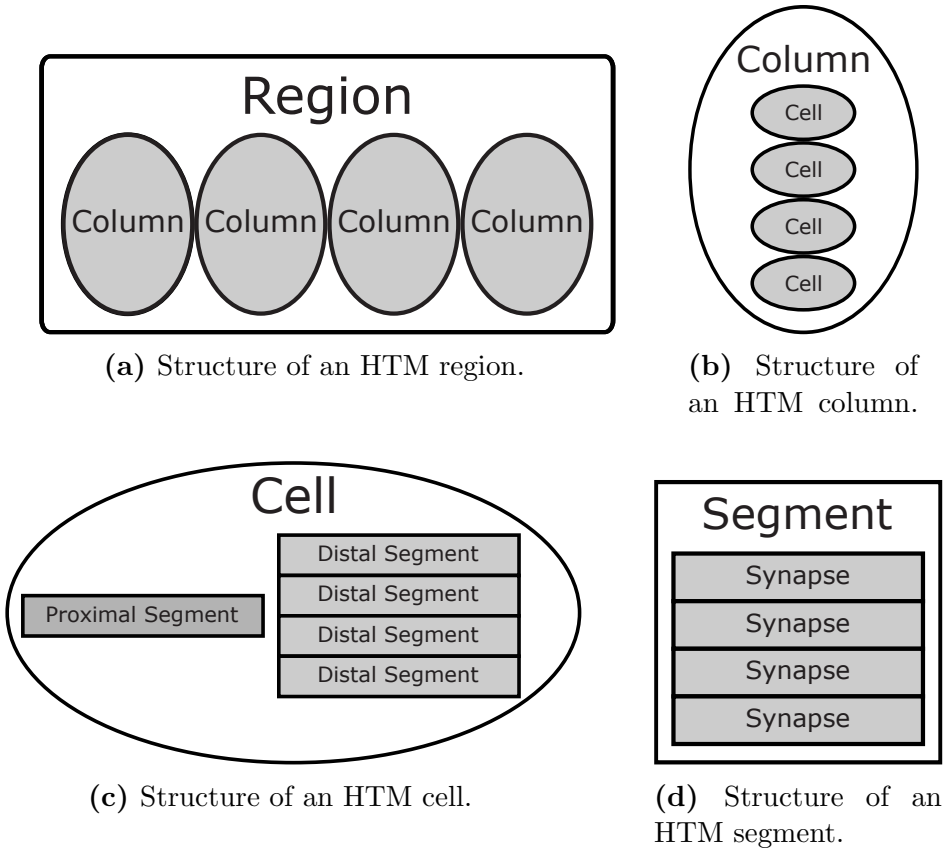
both an SP region and a TM region must be combined[1].

A region may be n-dimensional in size; however, for simplicity, it is convenient to work with a three-dimensional (3D) structure and two-dimensional (2D) data. One area where that 3D structure is desired is in computer vision, where the input is in the form of images. For simplicity, all further examples will assume a 3D structure with 2D data, such as the HTM previously shown in Figure 3.1. In addition to regions, an HTM contains a number of other components, such as columns, cells, segments, and synapses. Figure 3.2 shows a breakdown of the structure of those components.

As previously mentioned, a region is made up of columns of cells. A column may contain one or more cells. Columns within an SP region will have one cell[2] and columns within a TM region will have one or more cells. Each cell in a column receives the same feedforward input through a shared proximal segment. Those cells also

[1]It is common to split a complete HTM region into an SP and a TM region. This distinction eases both the implementation, utilization, and understanding of the system. In light of that, when the word region is utilized it will typically refer to either an SP or a TM region.

[2]Since SP columns only have one cell, the concept of cells may be ignored, with each column acting as a cell.



(a) Structure of an HTM region.

(b) Structure of an HTM column.

(c) Structure of an HTM cell.

(d) Structure of an HTM segment.

Figure 3.2: Structure of the various components inside of an HTM. A region (a) is comprised of columns (b) of cells (c) of segments (d) of synapses. There may exist multiple types of each subcomponent, except the proximal segment, which is limited to one per column.

receive different lateral inputs through distal segments. The activation of a column is determined by the proximal segment. When the proximal segment becomes active, the entire column is active.

A proximal segment is used to connect feedforward input, via synapses, to a column. A proximal segment becomes active when the number of active synapses exceeds a threshold. Each proximal segment has a fixed set of potential synapses, i.e. synapses may attach and / or detach from a proximal segment.

A distal segment is used to connect cells within a region. Each cell may have multiple distal segments. As with the proximal segment, when the number of active synapses exceeds a threshold the distal segment becomes active. Unlike a proximal segment, a distal segment has a dynamic set of potential synapses, allowing for new

potential synapses to be created[3].

A synapse is the lowest level unit inside of an HTM. A synapse has a direct connection with a cell, where the cell's state determines the synapse's input[4]. Each synapse has an associated permanence, which is a scalar between zero and one, inclusive. If the permanence is above a threshold, the synapse is connected; otherwise, it is disconnected. The larger the permanence the more difficult it is for the synapse to detach, likewise, the smaller the permanence the more difficult it is for the synapse to attach. A synapse becomes active if its input is active[5]. As in artificial neural networks (ANNs), synapses have weights; however, the weight of an HTM synapse is binary, whereas the weight of an ANN synapse is scalar. The weight of an HTM synapse is simply '1' when the synapse is connected and '0' when it is disconnected[6].

A cell in an HTM is analogous to a neuron in an ANN, having all activity entering the cell and the corresponding output exiting the cell. The output of a cell takes the form of three states — active, predictive, and inactive. The active state occurs when the cell's proximal segment becomes active. The predictive state occurs when at least one of the distal segments becomes active. In all other conditions, the cell is inactive. It is possible for the cell to be in both the active and predictive states. This would occur when at least part of the predicted pattern is the same as the current pattern.

If a cell is in the active state, cells that are connected to that cell by distal segments may also become active. Those cells would then enter the predictive state, collectively, indicating what the next input should be. Using those states, the output of a region

[3]A limit may be placed on the maximum number of synapses per segment, reducing the network complexity.

[4]If the cell is active, the input is '1'; otherwise, the input is '0'. For connections with the SP, it is usually more convenient to observe the column's state, where an active column results in an input of '1' and an inactive column results in an input of '0'. For the lowest region (where the region is directly connected to the HTM's input), a synapse's input is a single bit of the input SDR.

[5]It is common to refer to the state of the synapse as a function of its input and its permanence. For example, an "active connected" synapse would be a synapse whose input is active and whose permanence exceeds the connected threshold.

[6]The weight may be perceived at as the logical value of the synapse's connection state, i.e. '1' for connected and '0' for unconnected.

is found by taking the bitwise logical OR of the active states and predictive states of the region's cells[7].

3.2 SP Properties

The SP must utilize all columns to represent the input. This attribute eliminates dead columns, as all columns, even those that rarely receive an active input, will still be utilized in representing the input. This also ensures that the weaker columns have some degree of representation. The SP uses the concept of boosting, where columns with a low activation rate are forced to become active[8], to assist in achieving this property.

The percentage of active columns in a given range must remain constant. The number of active columns is fixed within an inhibition radius. This "radius" refers to a set of columns spatially located around a given column. The radius is dynamic and may range from only direct neighbors to the size of the entire region.

The inhibition radius is proportional to the size of the receptive field of the columns. The receptive field refers to the portion of the input that a column is able to connect with. This field exhibits plasticity in that synapses may dynamically become connected or disconnected from the column's proximal segment, thus changing the column's input.

In addition to those three properties, the SP must also avoid learning trivial patterns and avoid forming extra connections. To avoid learning trivial patterns, a threshold is applied to the proximal segment, requiring that a certain number of synapses must be active. To avoid forming extra connections, permanence's are both incremented and decremented during the SP's learning phase. If the synapse

[7]This is the output that would be passed as a feedforward input to the next region. If this region is the final region, it may be desirably to use either the active or predictive state, rather than both.

[8]The columns must still be connected to enough active synapses. If a column is poorly connected, such that its synapses are connected to inputs that rarely become active, it is likely that these columns will still become dead.

is connected to an active column that survived the inhibition phase, the synapse's permanence is incremented; otherwise, it is decremented.

3.3 SP Operation

The SP consists of three phases, namely overlap, inhibition, and learning. In this section the three phases will be presented based on their original, algorithmic definition. This algorithm follows an iterative, online approach, where the learning updates occur after the presentation of each input. Before the execution of the algorithm, some initializations must take place.

3.3.1 Initialization

Initialization occurs upon instantiation of the SP and is used to prepare the SP for operation. During initialization, each column randomly connects with a user-defined number of inputs via the formation of potential proximal synapses. The standard assignment allows for potential synapses to form across any distances[9]. The synapses connecting to a column are connected via the column's proximal segment. After forming the synapses, their respective permanences are randomly initialized. The permanences are set within a small range around the threshold used for determining the connectivity of a synapse. Additionally, the permanence values must be a function of their distance from the column they are attached to, where synapses traveling less distances are assigned larger permanence values.

3.3.2 Phase 1: Overlap

The first phase of the SP is known as overlap, and it is used to compute the degree of overlap between a column and its current input. This is done by summing the active

[9]This gives the synapses the ability to connect to any input bit. In other words, the number of synapses per column and the number of input bits (the size of the input to this region) are binomial coefficients.

Algorithm 1 SP phase 1: Overlap

```
1: for all col ∈ sp.columns do
2:     col.overlap ← 0
3:     for all syn ∈ col.connected_synapses() do
4:         col.overlap ← col.overlap + syn.active()

5:     if col.overlap < pseg_th then
6:         col.overlap ← 0
7:     else
8:         col.overlap ← col.overlap ∗ col.boost
```

synapses connected to the column. If the sum is too low, the overlap is set to zero; thus, removing this column from the potential set of active columns. The overlap is then boosted, by simply multiplying the overlap by that column's boost.

The operation of this phase is shown algorithmically in Algorithm 1. In Algorithm 1, the SP is represented by the object *sp*. The method *col*.*connected_synapses*() returns an instance to each synapse on *col*'s proximal segment that is connected, i.e. synapses having permanence values greater than the permanence connected threshold, *psyn_th*. The method *syn*.*active*() returns '1' if *syn*'s input is active and '0' otherwise. *pseg_th*[10] is a parameter that determines the activation threshold of a proximal segment, such that there must be at least *pseg_th* active connected proximal synapses on a given proximal segment for it to become active. The parameter *col.boost* is the boost for *col*, which is initialized to '1' and updated according to Algorithm 4.

3.3.3 Phase 2: Inhibition

The second phase is known as inhibition, and it is used to determine the set of active columns. Within a column's inhibition radius, the largest overlap, up to a user-defined amount, is obtained. If this column's overlap is at least as large as that

[10]This parameter was originally referred to as the minimum overlap; however, it is renamed in this work to allow consistency between the SP and the TM.

Algorithm 2 SP phase 2: Inhibition

```
1: for all col ∈ sp.columns do
2:     mo ← kmax_overlap(sp.neighbors(col), k)
3:     if col.overlap > 0 and col.overlap ≥ mo then
4:         col.active ← 1
5:     else
6:         col.active ← 0
```

overlap, the column is chosen as one of the active columns. Hence, the columns within the inhibition radius of the column being examined are inhibiting that column.

The operation of this phase is shown algorithmically in Algorithm 2. In Algorithm 2, k$max_overlap(C, k)$ is a function that returns the k-th largest overlap of the columns in C. The method $sp.neighbors(col)$ returns the columns that are within *col*'s neighborhood, including *col*, where the size of the neighborhood is determined by the inhibition radius. The parameter k is the desired column activity level. Line 2 in Algorithm 2 computes the k-th largest overlap out of *col*'s neighborhood. A column is then said to be active if its overlap is greater than zero and the computed minimum overlap, *mo*.

3.3.4 Phase 3: Learning

The third phase is the learning phase. This phase is used to update the permanence values of all of the synapses, the boost values, and the inhibition radius. This phase is optional and may be disabled. Disabling it, would cause the SP to no longer perform any learning processes. In this phase, the permanence of all of the potential synapses is updated for each active column. If the synapse is active (and thereby contributing to the column activating) its permanence is incremented; otherwise, its permanence is decremented. After the permanences have been updated, the duty cycles and boost are updated for each column.

The duty cycle refers to a moving average of the frequency of an event. Two duty

cycles exist for the columns, the active duty cycle and the overlap duty cycle. The active duty cycle refers to how often a column has been active after the inhibition phase. The overlap duty cycle refers to how often a column's overlap has exceeded the minimum overlap threshold. Before the duty cycles are updated, the minimum active duty cycle is calculated, which is by default equal to one percent of the largest active duty cycle within the inhibition radius. The duty cycles are then updated and the minimum active duty cycle is used to update the boost.

If the active duty cycle is above the minimum active duty cycle then the boost is set to one; otherwise, the boost is linearly increased. To aid the activation of synapses, if the overlap duty cycle is less than the minimum active duty cycle, the permanences of each synapse connected to a column is increased by ten percent (by default) of the connected permanence threshold.

The final step in the learning phase involves globally updating the inhibition radius. The inhibition radius is set to be the size of the average receptive field. This is found by finding the average distance that a connected synapse is from its column, i.e. the distance is measured from the column's position to the position of the input bit connected via the synapse.

The operation of this phase is shown algorithmically in Algorithm 3. In Algorithm 3, $syn.p$ refers to the permanence of syn. The functions min and max return the minimum and maximum values of their arguments, respectively, and are used to keep the permanence values bounded in the closed interval [0, 1]. The constants $syn.psyn\_inc$ and $syn.psyn\_dec$ are the proximal synapse permanence increment and decrement amounts, respectively.

The function $max\_adc(C)$ returns the maximum active duty cycle of the columns in C, where the active duty cycle is a moving average denoting the frequency of column activation. Similarly, the overlap duty cycle is a moving average denoting the frequency of the column's overlap being at least equal to the proximal segment activation

Algorithm 3 SP phase 3: Learning

```
    # Adapt permanences
 1: for all col ∈ sp.columns do
 2:     if col.active then
 3:         for all syn ∈ col.synapses do
 4:             if syn.active() then
 5:                 syn.p ← min(1, syn.p + syn.psyn_inc)
 6:             else
 7:                 syn.p ← max(0, syn.p − syn.psyn_dec)

    # Perform boosting operations
 8: for all col ∈ sp.columns do
 9:     col.mdc ← 0.01 ∗ max_adc(sp.neighbors(col))
10:     col.update_active_duty_cycle()
11:     col.update_boost()

12:     col.update_overlap_duty_cycle()
13:     if col.odc < col.mdc then
14:         for all syn ∈ col.synapses do
15:             syn.p ← min(1, syn.p + 0.1 ∗ psyn_th)

16: sp.update_inhibition_radius()
```

threshold. The functions *col.* u*pdate_active_duty_cycle*() and *col.* u*pdate_overlap_duty_cycle*() are used to update the active and overlap duty cycles, respectively, by computing the new moving averages. The parameters *col.odc*, *col.adc*, and *col.mdc* refer to *col*'s overlap duty cycle, active duty cycle, and minimum duty cycle, respectively. Those duty cycles are used to ensure that columns have a certain degree of activation.

The method *col.* u*pdate_boost*() is used to update the boost for column, *col*, as shown in Algorithm 4, where *maxb* refers to the maximum boost. It is important to note that the whitepaper did not explicitly define how the boost should be computed. This boost function was obtained from the source code of Numenta's implementation of HTM, Numenta platform for intelligent computing (NuPIC) [11].

The method *sp.* u*pdate_inhibition_radius*() is used to update the inhibition radius. The inhibition radius is set to the average receptive field size, which is computed as

Algorithm 4 Boost Update: $col.\,update\_boost()$

1: **if** $col.mdc == 0$ **then**
2: $\quad col.boost \leftarrow maxb$
3: **else if** $col.adc > col.mdc$ **then**
4: $\quad col.boost \leftarrow 1$
5: **else**
6: $\quad col.boost = col.adc * ((1 - maxb)/col.mdc) + maxb$

the average distance between all connected synapses and their respective columns in the input and the SP.

Chapter 4

SP Mathematical Framework

4.1 Notation

The operation of the SP (see Section 3.3) lends itself to a vectorized notation. By redefining the operations to work with vectors it is possible not only to create a mathematical representation, but also to greatly improve upon the efficiency of the operations. The notation described in this section will be used as the notation for the remainder of this work.

All vectors will be lowercase, bold-faced letters with an arrow hat. Vectors are assumed to be row vectors, such that the transpose of the vector will produce a column vector. All matrices will be uppercase, bold-faced letters. Subscripts on vectors and matrices are used to denote where elements are being indexed, following a row-column convention, such that $\boldsymbol{X}_{i,j} \in \boldsymbol{X}$ refers to $\boldsymbol{X}$ at row index[1] i and column index j. Element-wise operations between a vector and a matrix are performed column-wise, such that $\vec{\boldsymbol{x}}^T \odot \boldsymbol{Y} = \vec{\boldsymbol{x}}_i \boldsymbol{Y}_{i,j}\ \forall i\ \forall j$.

Let I(k) be defined as the indicator function, such that the function will return 1 if event k is true and 0 otherwise. If the input to this function is a vector of events or a matrix of events, each event will be evaluated independently, with the function returning a vector or matrix of the same size as its input. Any variable with a superscript in parentheses is used to denote the type of that variable. For example,

[1] All indices start at 0.

Table 4.1: User-defined parameters for the SP

Parameter	Description
n	Number of patterns (samples)
p	Number of inputs (features) in a pattern
m	Number of columns
q	Number of proximal synapses per column
ϕ_+	Permanence increment amount
ϕ_-	Permanence decrement amount
ϕ_δ	Window of permanence initialization
ρ_d	Proximal dendrite segment activation threshold
ρ_s	Proximal synapse activation threshold
ρ_c	Desired column activity level
κ_a	Minimum activity level scaling factor
κ_b	Permanence boosting scaling factor
β_0	Maximum boost
τ	Duty cycle period

$\vec{\boldsymbol{x}}^{(y)}$ is used to state that the variable $\vec{\boldsymbol{x}}$ is of type y.

All of the user-defined parameters are defined in Table 4.1[2]. These are parameters that must be defined before the initialization of the algorithm. All of those parameters are constants, except for parameter ρ_c, which is an overloaded parameter. It can either be used as a constant, such that for a column to be active it must be greater than the ρ_c-th column's overlap. It may also be defined to be a density, such that for a column to be active it must be greater than the $\lfloor \rho_c * num\_neighbors(i) \rfloor$-th column's overlap, where $num\_neighbors(i)$ is a function that returns the number of neighbors that column i has. If ρ_c is an integer it is assumed to be a constant, and if it is a scalar in the interval (0, 1] it is assumed to be used as a density.

Let the terms s, r, i, j, and k be defined as integer indices. They are henceforth bounded as follows: $s \in [0, n)$, $r \in [0, p)$, $i \in [0, m)$, $j \in [0, m)$, and $k \in [0, q)$.

[2]The parameters κ_a and κ_b have default values of 0.01 and 0.1, respectively.

4.2 Initialization

Competitive learning networks typically have each node fully connected to each input. The SP; however, follows a different line of logic, posing a new problem concerning the visibility of an input. As previously explained, the inputs connecting to a particular column are determined randomly. Let $\vec{\boldsymbol{c}} \in \mathbb{Z}^{1\times m}, \vec{\boldsymbol{c}} \in [0, m)$ be defined as the set of all columns indices, such that $\vec{\boldsymbol{c}}_i$ is the column's index at i. Let $\boldsymbol{U} \in \{0,1\}^{n\times p}$ be defined as the set of inputs for all patterns, such that $\boldsymbol{U}_{s,r}$ is the input for pattern s at index r. Let $\boldsymbol{\Lambda} \in \{r\}^{m\times q}$ be the source column indices for each proximal synapse on each column, such that $\boldsymbol{\Lambda}_{i,k}$ is the source column's index of $\vec{\boldsymbol{c}}_i$'s proximal synapse at index k. In other words, each $\boldsymbol{\Lambda}_{i,k}$ refers to a specific index in $\boldsymbol{U}_s$.

Let $\overrightarrow{\boldsymbol{ic}}_r \equiv \exists! r \in \boldsymbol{\Lambda}_i \ \forall r$, the event of input r connecting to column i, where $\exists!$ is defined to be the uniqueness quantification. Given q and p, the probability of a single input, $\boldsymbol{U}_{s,r}$, connecting to a column is calculated by using (4.1). In (4.1), the probability of an input not connecting is first determined. That probability is independent for each input; thus, the total probability of a connection not being formed is simply the product of those probabilities. The probability of a connection forming is therefore the complement of the probability of a connection not forming.

$$\begin{aligned} \mathbb{P}(\overrightarrow{\boldsymbol{ic}}_r) &= 1 - \prod_{k=0}^{q} \left(1 - \frac{1}{p-k}\right) \\ &= \frac{q+1}{p} \end{aligned} \tag{4.1}$$

It is also desired to know the average number of columns an input will connect with. To calculate this, let $\overrightarrow{\boldsymbol{\lambda}} \equiv \sum_{i=0}^{m-1} \sum_{k=0}^{q-1} \mathrm{I}(r = \Lambda_{i,k}) \ \forall r$, the random vector governing the count of connections between each input and all columns. Recognizing that the probability of a connection forming in m follows a binomial distribution, the

expected number of columns that an input will connect to is simply (4.2).

$$\mathbb{E}\left[\vec{\boldsymbol{\lambda}}_r\right] = m\mathbb{P}(\vec{\boldsymbol{ic}}_r) \tag{4.2}$$

Using (4.1) it is possible to calculate the probability of an input never connecting, as shown in (4.3). Since the probabilities are independent, it simply reduces to the product of the probability of an input not connecting to a column, taken over all columns. Let $\lambda' \equiv \sum_{r=0}^{p-1} \mathrm{I}(\vec{\boldsymbol{\lambda}}_r = 0)$, the random variable governing the number of unconnected inputs. From (4.3), the expected number of unobserved inputs may then be trivially obtained as (4.4). Using (4.3) and (4.2), it is possible to obtain a lower bound for m and q, by choosing those parameters such that a certain amount of input visibility is obtained. To guarantee observance of all inputs, (4.3) must be zero. Once that is satisfied, the desired number of times an input is observed may be determined by using (4.2).

$$\mathbb{P}\left(\vec{\boldsymbol{\lambda}}_r = 0\right) = (1 - \mathbb{P}(\vec{\boldsymbol{ic}}_r))^m \tag{4.3}$$

$$\mathbb{E}[\lambda'] = p\mathbb{P}\left(\vec{\boldsymbol{\lambda}}_r = 0\right) \tag{4.4}$$

Once each column has its set of inputs, the permanences must be initialized. As previously stated, permanences were defined to be initialized with a random value close to ρ_s, but biased based on the distance between the synapse's source (input column) and destination (SP column). To obtain further clarification, NuPIC's source code [11] was consulted. It was found that the permanences were randomly initialized, with approximately half of the permanences creating connected proximal synapses and the remaining permanences creating potential (unconnected) proximal synapses. Additionally, to ensure that each column has a fair chance of being selected during inhibition, there are at least ρ_d connected proximal synapses on each column.

Let $\boldsymbol{\Phi} \in \mathbb{R}^{m \times q}$ be defined as the set of permanences for each column, such that $\boldsymbol{\Phi}_i$ is the set of permanences for the proximal synapses for $\vec{\boldsymbol{c}}_i$. Each $\boldsymbol{\Phi}_{i,k}$ is randomly initialized as shown in (4.5), where Unif represents the uniform distribution. Using (4.5), the expected permanence value would be equal to ρ_s; thus, $q/2$ proximal synapses would be initialized as connected for each column. To ensure that each column has a fair chance of being selected, ρ_d should be less than $q/2$.

$$\boldsymbol{\Phi}_{i,k} \sim \mathrm{U}nif(\rho_s - \phi_\delta, \rho_s + \phi_\delta) \tag{4.5}$$

It is possible to predict, before training, the initial response of the SP with a given input. This insight allows parameters to be crafted in a manner that ensures a desired amount of column activity. Let $\boldsymbol{X} \in \{0,1\}^{m \times q}$ be defined as the set of inputs for each column, such that $\boldsymbol{X}_i$ is the set of inputs for $\vec{\boldsymbol{c}}_i$. Let $\overrightarrow{\boldsymbol{ai}}_i \equiv \sum_{k=0}^{q-1} \boldsymbol{X}_{i,k}$, the random variable governing the number of active inputs on column i. Let $\mathbb{P}(\boldsymbol{X}_{i,k})$ be defined as the probability of the input connected via proximal synapse k to column i being active. $\mathbb{P}(\boldsymbol{X}_i)$ is therefore defined to be the probability of an input connected to column i being active. Similarly, $\mathbb{P}(\boldsymbol{X})$ is defined to be the probability of an input on any column being active. The expected number of active proximal synapses on column i is then given by (4.6). Let $a \equiv \frac{1}{m} \sum_{i=0}^{m-1} \sum_{k=0}^{q-1} \boldsymbol{X}_{i,k}$, the random variable governing the average number of active inputs on a column. Equation (4.6) is then generalized to (4.7), the expected number of active proximal synapses for each column.

$$\mathbb{E}[\overrightarrow{\boldsymbol{ai}}_i] = q\mathbb{P}(\boldsymbol{X}_i) \tag{4.6}$$

$$\mathbb{E}[a] = q\mathbb{P}(\boldsymbol{X}) \tag{4.7}$$

Let $\boldsymbol{AC}_{i,k} \equiv \boldsymbol{X}_{i,k} \cap \mathrm{I}(\boldsymbol{\Phi}_{i,k} \geq \rho_s)$, the event that proximal synapse k is active and connected on column i. Let $\overrightarrow{\boldsymbol{ac}}_i \equiv \sum_{k=0}^{q-1} \boldsymbol{AC}_{i,k}$, the random variable governing

the number of active and connected proximal synapses for column i. Let $\mathbb{P}(\boldsymbol{AC}_{i,k}) \equiv \mathbb{P}(\boldsymbol{X}_{i,k})\rho_s$, the probability that a proximal synapse is active and connected[3]. Following (4.6), the expected number of active connected proximal synapses on column i is given by (4.8).

$$\mathbb{E}[\overrightarrow{\boldsymbol{ac}}_i] = q\mathbb{P}(\boldsymbol{AC}_{i,k}) \tag{4.8}$$

Let $\mathrm{B}in(k; n, p)$ be defined as the probability mass function (PMF) of a binomial distribution, where k is the number of successes, n is the number of trials, and p is the success probability in each trial. Let $at \equiv \sum_{i=0}^{m-1} \mathrm{I}\left(\left(\sum_{k=0}^{q-1} \boldsymbol{X}_{i,k}\right) \geq \rho_d\right)$, the random variable governing the number of columns having at least ρ_d active proximal synapses. Let $act \equiv \sum_{i=0}^{m-1} \mathrm{I}\left(\left(\sum_{k=0}^{q-1} \boldsymbol{AC}_{i,k}\right) \geq \rho_d\right)$, the random variable governing the number of columns having at least ρ_d active connected proximal synapses. Let π_x and π_{ac} be defined as random variables that are equal to the overall mean of $\mathbb{P}(\boldsymbol{X})$ and $\mathbb{P}(\boldsymbol{AC})$, respectively. The expected number of columns with at least ρ_d active proximal synapses and the expected number of columns with at least ρ_d active connected proximal synapses are then given by (4.9) and (4.10), respectively.

In (4.9), the summation computes the probability of having less than ρ_d active connected proximal synapses, where the individual probabilities within the summation follow the PMF of a binomial distribution. To obtain the desired probability, the complement of that probability is taken. It is then clear that the mean is nothing more than that probability multiplied by m. For (4.10) the logic is similar, with the key difference being that the probability of a success is a function of both $\boldsymbol{X}$ and ρ_s,

[3] ρ_s was used as a probability. Because $\rho_s \in \mathbb{R}, \rho_s \in (0, 1)$, permanences are uniformly initialized with a mean of ρ_s, and for a proximal synapse to be connected it must have a permanence value at least equal to ρ_s, ρ_s may be used to represent the probability that an initialized proximal synapse is connected.

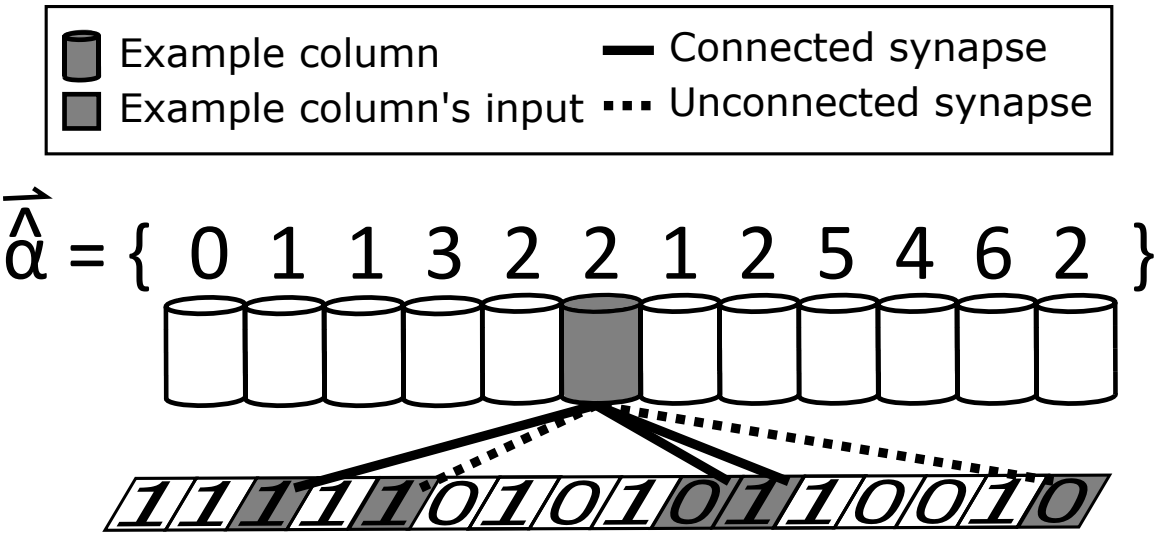


Figure 4.1: SP phase 1 example where $m = 12$, $q = 5$, and $\rho_d = 2$. It was assumed that the boost for all columns is at the initial value of '1'. For simplicity, only the connections for the example column, highlighted in gray, are shown.

as it was in (4.8).

$$\mathbb{E}[at] = m \left[1 - \sum_{t=0}^{\rho_d - 1} \mathrm{B}in(t; q, \pi_x) \right] \tag{4.9}$$

$$\mathbb{E}[act] = m \left[1 - \sum_{t=0}^{\rho_d - 1} \mathrm{B}in(t; q, \pi_{ac}) \right] \tag{4.10}$$

4.3 Phase 1: Overlap

Let $\vec{\boldsymbol{b}} \in \mathbb{R}^{1 \times m}$ be defined as the set of boost values for all columns, such that $\vec{\boldsymbol{b}}_i$ is the boost for $\vec{\boldsymbol{c}}_i$. Let $\boldsymbol{Y} \equiv \mathrm{I}(\boldsymbol{\Phi}_i \geq \rho_s)\ \forall i$, the bit-mask for the proximal synapse's activations. $\boldsymbol{Y}_i$ is therefore a row-vector bit-mask, with each '1' representing a connected synapse and each '0' representing an unconnected synapse. In this manner, the connectivity (or lack thereof) for each synapse on each column is obtained. The overlap for all columns, $\vec{\boldsymbol{\alpha}} \in \{0, 1\}^{1 \times m}$, is then obtained by using (4.11), which is a function of $\vec{\hat{\boldsymbol{\alpha}}} \in \mathbb{Z}^{1 \times m}$. $\vec{\hat{\boldsymbol{\alpha}}}$ is the sum of the active connected proximal synapses for all columns, and is defined in (4.12).

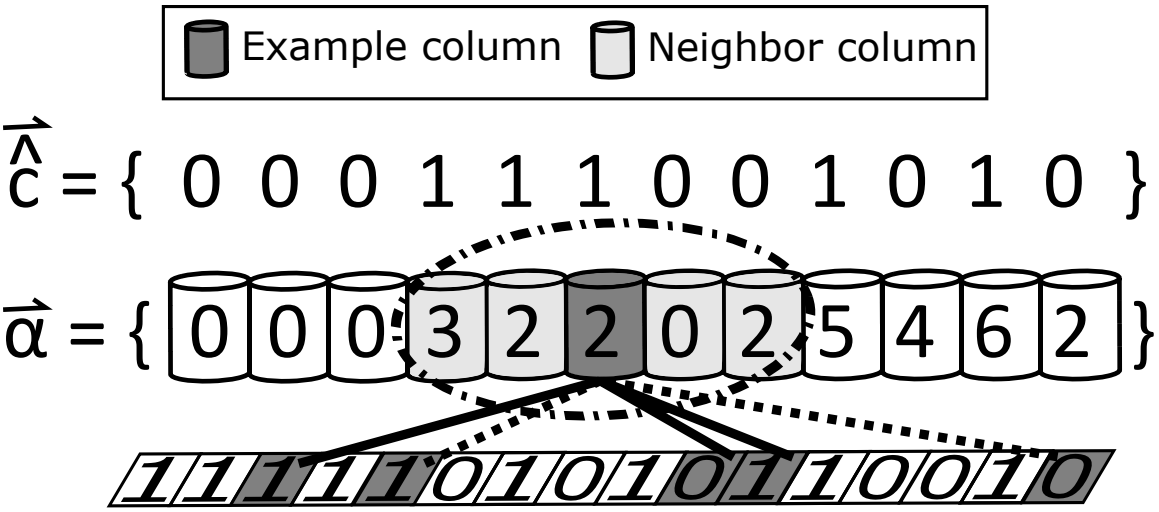


Figure 4.2: SP phase 2 example where $\rho_c = 2$ and $\sigma_o = 2$. The overlap values were determined from the SP phase 1 example.

Comparing these equations with Algorithm 1, it is clear that $\vec{\hat{\boldsymbol{\alpha}}}$ will have the same value as *col.overlap* before line five, and that the final value of *col.overlap* will be equal to $\vec{\boldsymbol{\alpha}}$. To help provide further understanding, a simple example demonstrating the functionality of this phase is shown in Figure 4.1.

$$\vec{\boldsymbol{\alpha}} \equiv \begin{cases} \vec{\hat{\boldsymbol{\alpha}}}_i \vec{\boldsymbol{b}}_i & \vec{\hat{\boldsymbol{\alpha}}}_i \geq \rho_d, \\ 0 & otherwise \end{cases} \quad \forall i \tag{4.11}$$

$$\vec{\hat{\boldsymbol{\alpha}}}_i \equiv \boldsymbol{X}_i \bullet \boldsymbol{Y}_i \tag{4.12}$$

4.4 Phase 2: Inhibition

Let $\boldsymbol{H} \in \{0,1\}^{m \times m}$ be defined as the neighborhood mask for all columns, such that $\boldsymbol{H}_i$ is the neighborhood for $\vec{\boldsymbol{c}}_i$. $\vec{\boldsymbol{c}}_j$ is then said to be in $\vec{\boldsymbol{c}}_i$'s neighborhood if and only if $\boldsymbol{H}_{i,j}$ is '1'. Let k$max(S,k)$ be defined as the k-th largest element of S. Let m$ax(\vec{\boldsymbol{v}})$ be defined as a function that will return the maximum value in $\vec{\boldsymbol{v}}$. The set of active columns, $\vec{\hat{\boldsymbol{c}}} \in \{0,1\}^{1 \times m}$, may then be obtained by using (4.13), where $\vec{\hat{\boldsymbol{c}}}$ is an indicator

vector representing the activation (or lack of activation) for each column. The result of the indicator function is determined by $\vec{\boldsymbol{\gamma}} \in \mathbb{Z}^{1\times m}$, which is defined in (4.14) as the ρ_c-th largest overlap (lower bounded by one) in the neighborhood of $\vec{\boldsymbol{c}}_i \ \forall i$.

Comparing these equations with Algorithm 2, $\vec{\boldsymbol{\gamma}}$ is a slightly altered version of mo. Instead of just being the ρ_c-th largest overlap for each column, it is additionally lower bounded by one. Referring back to Algorithm 2, line 3 is a biconditional statement evaluating to true if the overlap is at least mo and greater than zero. By simply enforcing mo to be at least one, the biconditional is reduced to a single condition. That condition is evaluated within the indicator function; therefore, (4.13) carries out the logic in the if statement in Algorithm 2. Continuing with the demonstration shown in Figure 4.1, Figure 4.2 shows an example execution of phase two.

$$\vec{\hat{\boldsymbol{c}}} \equiv \mathrm{I}(\vec{\boldsymbol{\alpha}}_i \geq \vec{\boldsymbol{\gamma}}_i) \ \forall i \tag{4.13}$$

$$\vec{\boldsymbol{\gamma}} \equiv \mathrm{m}ax(\mathrm{k}max(\boldsymbol{H}_i \odot \vec{\boldsymbol{\alpha}}, \rho_c), 1) \ \forall i \tag{4.14}$$

4.5 Phase 3: Learning

Let $clip(\boldsymbol{M}, lb, ub)$ be defined as a function that will clip all values in the matrix $\boldsymbol{M}$ outside of the range $[lb, ub]$ to lb if the value is less than lb, or to ub if the value is greater than ub. $\boldsymbol{\Phi}$ is then recalculated by (4.15), where $\boldsymbol{\delta\Phi}$ is the proximal synapse's permanence update amount given by (4.16)[4].

$$\boldsymbol{\Phi} \equiv clip\left(\boldsymbol{\Phi} \oplus \boldsymbol{\delta\Phi}, 0, 1\right) \tag{4.15}$$

[4]Due to $\boldsymbol{X}$ being binary, a bitwise negation is equivalent to the shown logical negation. In a similar manner, the multiplications of $\vec{\hat{\boldsymbol{c}}}^{\boldsymbol{T}}$ with $\boldsymbol{X}$ and $\neg\boldsymbol{X}$ can be replaced by an AND operation (logical or bitwise).

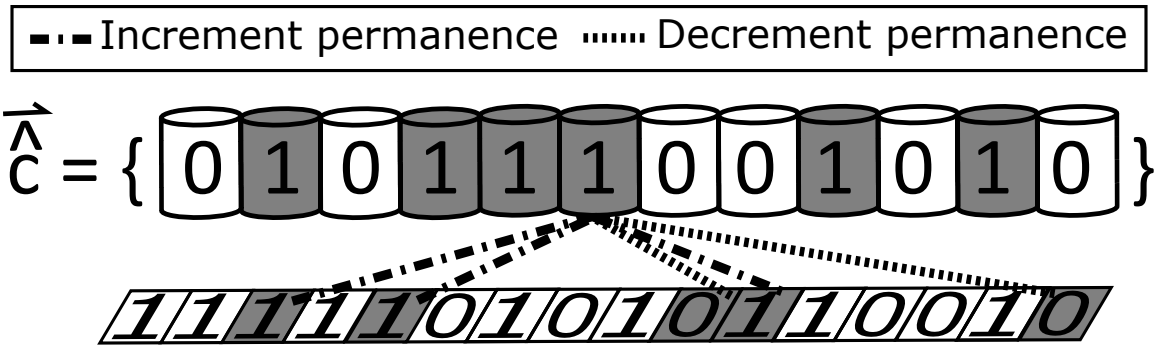


Figure 4.3: SP phase 3 example, demonstrating the adaptation of the permanences. The gray columns are used denote the active columns, where those activations were determined from the SP phase 2 example.

$$\boldsymbol{\delta\Phi} \equiv \vec{\hat{\boldsymbol{c}}}^{T} \odot (\phi_{+}\boldsymbol{X} - (\phi_{-}\neg\boldsymbol{X})) \tag{4.16}$$

The result of these two equations is equivalent to the result of executing the first seven lines in Algorithm 3. If a column is active, it will be denoted as such in $\vec{\hat{\boldsymbol{c}}}$; therefore, using that vector as a mask, the result of (4.16) will be a zero if the column is inactive, otherwise it will be the update amount. From Algorithm 3, the update amount should be ϕ_{+} if the synapse was active and ϕ_{-} if the synapse was inactive. A synapse is active only if its source column is active. That activation is determined by the corresponding value in $\boldsymbol{X}$. In this manner, $\boldsymbol{X}$ is also being used as a mask, such that active synapses will result in the update equalling ϕ_{+} and inactive synapses (selected by inverting $\boldsymbol{X}$) will result in the update equalling ϕ_{-}. By clipping the element-wise sum of $\boldsymbol{\Phi}$ and $\boldsymbol{\delta\Phi}$, the permanences stay bounded between [0, 1]. As with the previous two phases, the visual demonstration is continued, with Figure 4.3 illustrating the primary functionality of this phase.

Let $\overrightarrow{\boldsymbol{\eta}^{(\boldsymbol{a})}} \in \mathbb{R}^{1\times m}$ be defined as the set of active duty cycles for all columns, such that $\overrightarrow{\boldsymbol{\eta}_{\boldsymbol{i}}^{(\boldsymbol{a})}}$ is the active duty cycle for $\vec{\boldsymbol{c}}_{\boldsymbol{i}}$. Let $\overrightarrow{\boldsymbol{\eta}^{(\boldsymbol{min})}} \in \mathbb{R}^{1\times m}$ be defined by (4.17) as the set of minimum active duty cycles for all columns, such that $\overrightarrow{\boldsymbol{\eta}_{\boldsymbol{i}}^{(\boldsymbol{min})}}$ is the minimum active duty cycle for $\vec{\boldsymbol{c}}_{i}$. This equation is clearly the same as line 9 in

Algorithm 3.

$$\overrightarrow{\boldsymbol{\eta}^{(min)}} \equiv \kappa_a \, max\left(\boldsymbol{H}_i \odot \overrightarrow{\boldsymbol{\eta}^{(a)}}\right) \ \forall i \tag{4.17}$$

Let u*pdate_active_duty_cycle*($\vec{\boldsymbol{c}}$) be defined as a function that updates the moving average duty cycle for the active duty cycle for each $\vec{\boldsymbol{c}_i} \in \vec{\boldsymbol{c}}$. That function should compute the frequency of each column's activation. After calling u*pdate_active_duty_cycle*($\vec{\boldsymbol{c}}$), the boost for each column is updated by using (4.18). In (4.18), $\beta\left(\overrightarrow{\boldsymbol{\eta}_i^{(a)}}, \overrightarrow{\boldsymbol{\eta}_i^{(min)}}\right)$ is defined as the boost function, following (4.19)[5]. The functionality of (4.18) is therefore shown to be equivalent to Algorithm 4.

$$\vec{\boldsymbol{b}} \equiv \beta\left(\overrightarrow{\boldsymbol{\eta}_i^{(a)}}, \overrightarrow{\boldsymbol{\eta}_i^{(min)}}\right) \ \forall i \tag{4.18}$$

$$\beta\left(\overrightarrow{\boldsymbol{\eta}_i^{(a)}}, \overrightarrow{\boldsymbol{\eta}_i^{(min)}}\right) \equiv \begin{cases} \beta_0 & \overrightarrow{\boldsymbol{\eta}_i^{(min)}} = 0 \\ 1 & \overrightarrow{\boldsymbol{\eta}_i^{(a)}} > \overrightarrow{\boldsymbol{\eta}_i^{(min)}} \\ \overrightarrow{\boldsymbol{\eta}_i^{(a)}} \frac{1-\beta_0}{\overrightarrow{\boldsymbol{\eta}_i^{(min)}}} + \beta_0 & otherwise \end{cases} \tag{4.19}$$

Let $\overrightarrow{\boldsymbol{\eta}^{(o)}} \in \mathbb{R}^{1\times m}$ be defined as the set of overlap duty cycles for all columns, such that $\overrightarrow{\boldsymbol{\eta}_i^{(o)}}$ is the overlap duty cycle for $\vec{\boldsymbol{c}}_i$. Let u*pdate_overlap_duty_cycle*($\vec{\boldsymbol{c}}$) be defined as a function that updates the moving average duty cycle for the overlap duty cycle for each $\vec{\boldsymbol{c}}_i \in \vec{\boldsymbol{c}}$. That function should compute the frequency of each column's overlap being at least equal to ρ_d. After applying u*pdate_overlap_duty_cycle*($\vec{\boldsymbol{c}}$), the permanences are then boosted by using (4.20). This equation is equivalent to lines 13 – 15 in Algorithm 3, where the multiplication with the indicator function is used to accomplish the conditional and clipping is done to ensure the permanences stay

[5]The conditions within the piecewise function must be evaluated top-down, such that the first condition takes precedence over the second condition which takes precedence over the third condition.

within bounds.

$$\boldsymbol{\Phi} \equiv clip\left(\boldsymbol{\Phi} \oplus \kappa_b \rho_s \, \mathrm{I}\left(\overrightarrow{\boldsymbol{\eta}_i^{(o)}} < \overrightarrow{\boldsymbol{\eta}_i^{(min)}}\right), 0, 1\right) \tag{4.20}$$

Let $\mathrm{d}(x, y)$ be defined as the distance function[6] that computes the distance between x and y. To simplify the notation[7], let $pos(c, r)$ be defined as a function that will return the position of the column indexed at c located r regions away from the current region. For example, $pos(0, 0)$ returns the position of the first column located in the SP and $pos(0, -1)$ returns the position of the first column located in the previous region. The distance between $pos(\vec{\boldsymbol{c}}_i, 0)$ and $\mathrm{pos}(\boldsymbol{\Lambda}_{i,k}, -1)$ is then determined by $\mathrm{d}(pos(\vec{\boldsymbol{c}}_i, 0), pos(\boldsymbol{\Lambda}_{i,k}, -1))$.

Let $\boldsymbol{D} \in \mathbb{R}^{m \times q}$ be defined as the distance between an SP column and its corresponding connected synapses' source columns, such that $\boldsymbol{D}_{i,k}$ is the distance between $\vec{\boldsymbol{c}}_i$ and $\vec{\boldsymbol{c}}_i$'s proximal synapse's input at index k. $\boldsymbol{D}$ is computed following (4.21), where $\boldsymbol{Y}_i$ is used as a mask to ensure that only connected synapses may contribute to the distance calculation. The result of that element-wise multiplication would be the distance between the two columns or zero for connected and unconnected synapses, respectively[8].

$$\boldsymbol{D} \equiv (\mathrm{d}(pos(\vec{\boldsymbol{c}}_i, 0), pos(\boldsymbol{\Lambda}_{i,k}, -1)) \odot \boldsymbol{Y}_i \; \forall k) \; \forall i \tag{4.21}$$

The inhibition radius, σ_0, is defined by (4.22). The division in (4.22) is the sum of the distances divided by the number of connected synapses[9]. That division represents the average distance between connected synapses' source and destination columns,

[6] The distance function is typically the Euclidean distance.

[7] In an actual system the positions would be explicitly defined.

[8] It is assumed that an SP column and an input column do not coincide, i.e. their distance is greater than zero. If this occurs, $\boldsymbol{D}$ will be unstable, as zeros will refer to both valid and invalid distances. This instability is accounted for during the computation of the inhibition radius, such that it will not impact the actual algorithm.

[9] The summation of the connected synapses is lower-bounded by one to avoid division by zero.

and is therefore the average receptive field size. The inhibition radius is then set to the average receptive field size after it has been floored and raised to a minimum value of one, ensuring that the radius is an integer at least equal to one. Comparing (4.22) to line 16 in Algorithm 3, the two are equivalent.

$$\sigma_o \equiv max\left(1, \left\lfloor \frac{\sum_{i=0}^{m-1}\sum_{k=0}^{q-1} \boldsymbol{D}_{i,k}}{max(1, \sum_{i=0}^{m-1}\sum_{k=0}^{q-1} \boldsymbol{Y}_{i,k})} \right\rfloor\right) \tag{4.22}$$

Once the inhibition radius has been computed, the neighborhood for each column must be updated. This is done using the function $\mathrm{h}(\vec{\boldsymbol{c}}_i)$, which is dependent upon the type of inhibition being used (global or local) as well as the topology of the system[10]. This function is shown in (4.23), where $\vec{\boldsymbol{\zeta}}$ represents all of the columns located at the set of integer Cartesian coordinates bounded by an n-dimensional shape. Typically the n-dimensional shape is a represented by an n-dimensional hypercube.

$$\mathrm{h}(\vec{\boldsymbol{c_i}}) \equiv \begin{cases} \vec{\boldsymbol{c}} & \textit{global inhibition} \\ \vec{\boldsymbol{\zeta}} & \textit{local inhibition} \end{cases} \tag{4.23}$$

4.6 Boosting

It is important to understand the dynamics of boosting utilized by the SP. The SP's boosting mechanism is similar to DeSieno's [12] conscience mechanism. In that work, clusters that were too frequently active were penalized, allowing weak clusters to contribute to learning. The SP's primary boosting mechanism takes the reverse approach by rewarding infrequently active columns. Clearly, the boosting frequency and amount will impact the SP's learned representations.

The degree of activation is determined by the boost function, (4.19). From that

[10]For global inhibition, every value in $\boldsymbol{H}$ would simply be set to one regardless of the topology. This allows for additional optimizations of (4.14) and (4.17) and eliminates the need for (4.22) and (4.23). For simplicity only the generalized forms of the equations were shown.

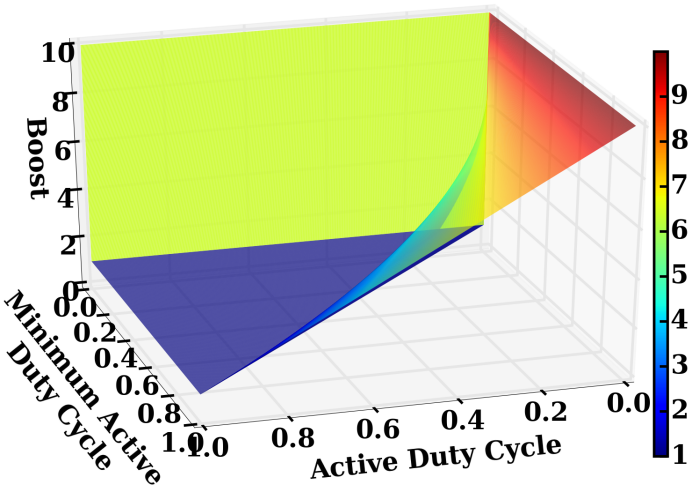


Figure 4.4: Demonstration of boost as a function of a column's minimum active duty cycle and active duty cycle.

equation, it is clear that a column's boost is determined by the column's minimum active duty cycle as well as the column's active duty cycle. Those two values are coupled, as a column's minimum active duty cycle is a function of its duty cycle, as shown in (4.17). To study how those two parameters affect a column's boost, Figure 4.4 was created. From this plot it is found that the non-boundary conditions for a column's boost follows the shape $1/\widetilde{\eta_i^{(min)}}$. It additionally shows the importance of evaluating the piecewise boost function in order. If the second condition is evaluated before the first condition, the boost will be set to its minimum, instead of its maximum value.

To study the frequency of boosting, the average number of boosted columns was observed by varying the level of sparseness in the input for both types of inhibition, as shown in Figure 4.5. For the overlap boosting mechanism, (4.18), very little boosting occurs, with boosting occurring more frequently for denser inputs. This is to be expected, as more bits would be active in the input; thus, causing more competition to

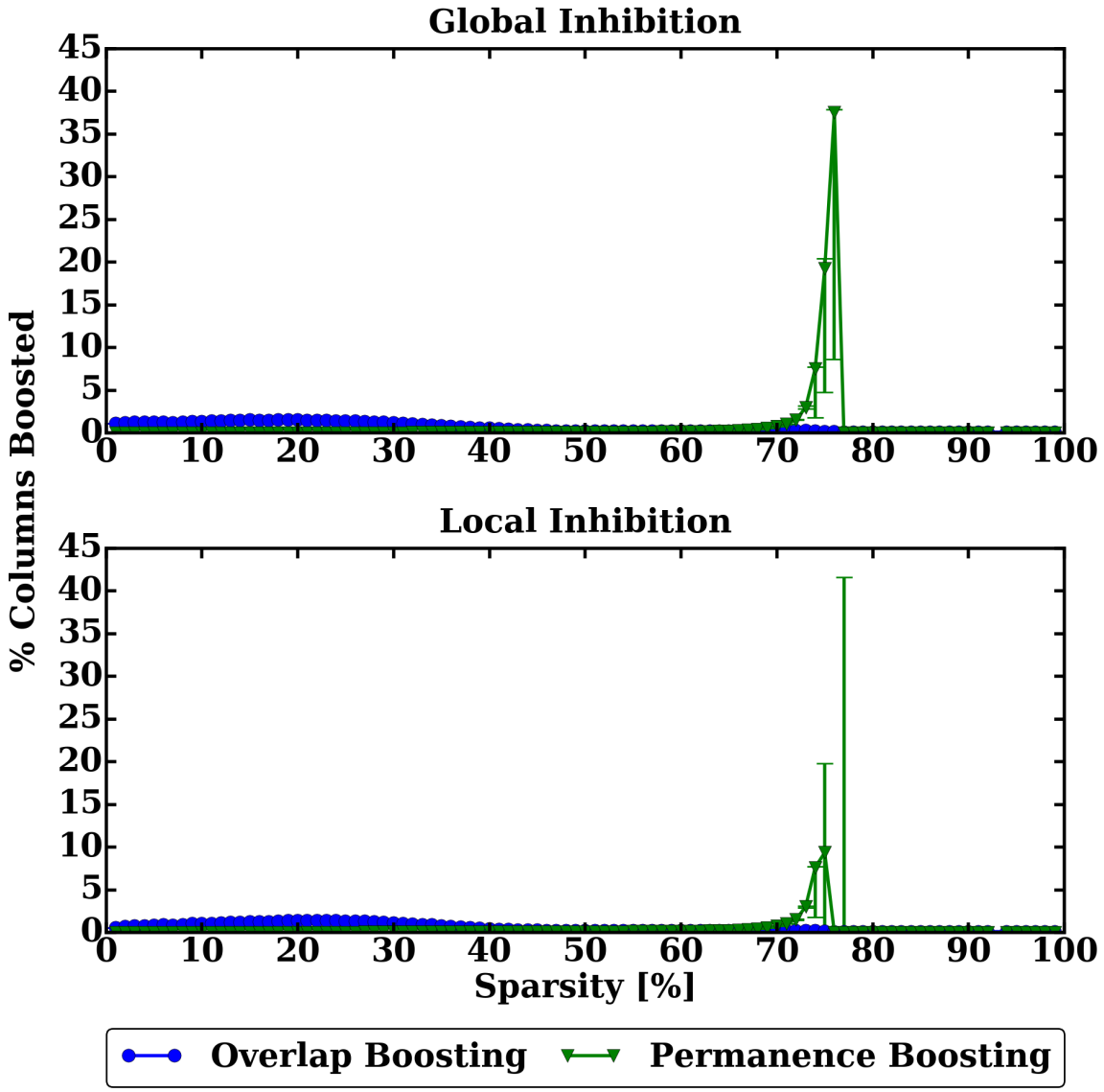


Figure 4.5: Demonstration of frequency of both boosting mechanisms as a function of the sparseness of the input. The top figure shows the results for global inhibition and the bottom figure shows the results for local inhibition[11].

occur among the columns. For the permanence boosting mechanism, (4.20), boosting primarily occurs when the sparsity is between 70 and 76%, with almost no boosting

[11]The inputs to the SP consisted of 100 randomly generated bit-streams with a width of 100 bits. Within each bit-stream, bits were randomly flipped to be active. The sparseness is then the percentage of non-active bits. Each simulation consisted of 10 epochs and was performed across 10 trials. The SP's parameters are as follows: $m = 2048$, $p = 100$, $q = 40$, $\rho_d = 15$, $\rho_s = 0.5$, $\phi_\delta = 0.05$, $\rho_c = \lfloor 0.02 * m \rfloor$, $\phi_+ = 0.03$, $\phi_- = 0.05$, $\beta_0 = 10$, and $\tau = 100$. Synapses were trimmed if their permanence value ever reached or fell below 10^{-4}. On the figure, each point represents a partial box plot, i.e. the data point is the median, the upper error bar is the third quartile, and the lower error bar is the first quartile.

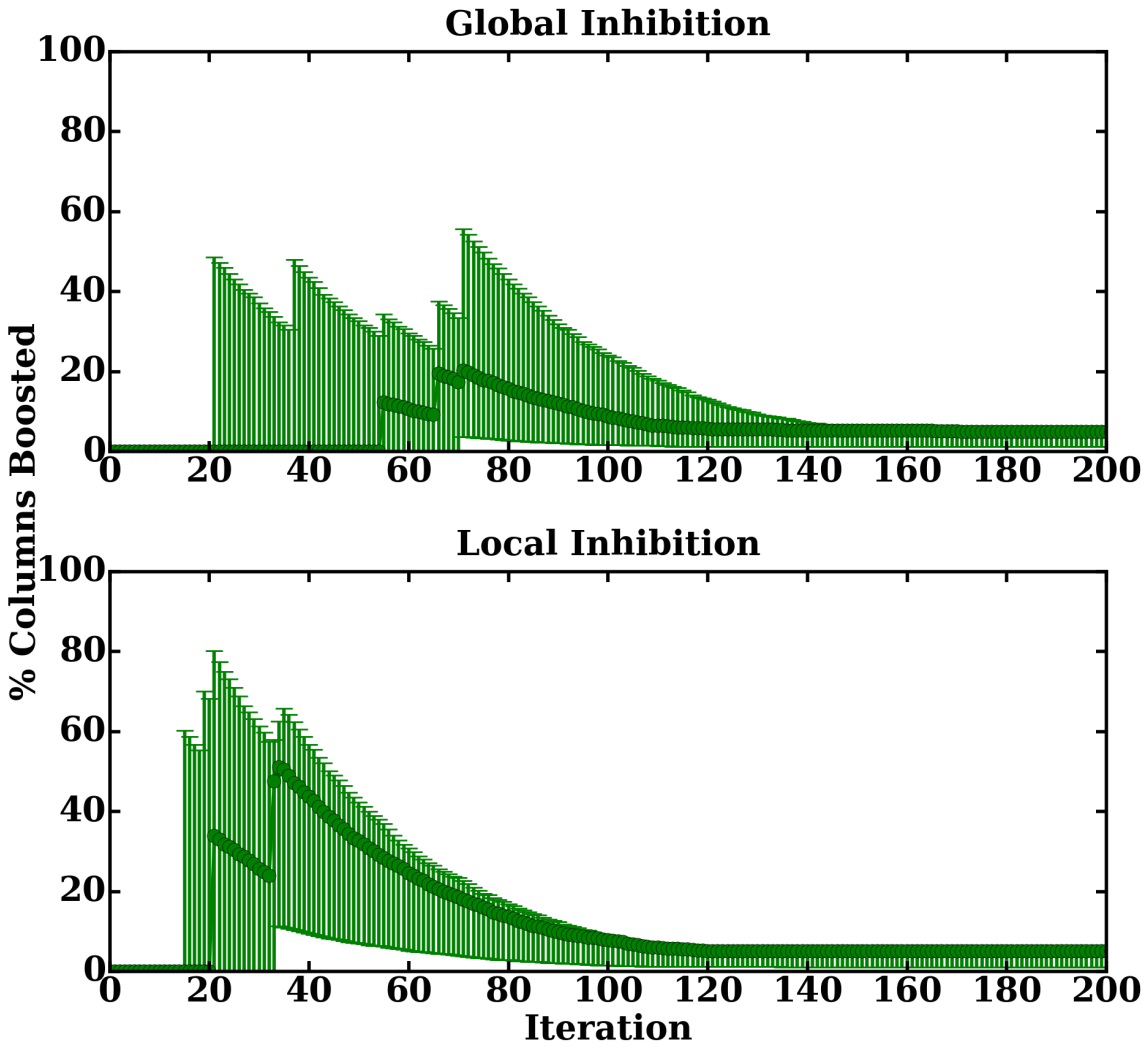


Figure 4.6: Frequency of boosting for the permanence boosting mechanism for a sparsity of 74%. The top figure shows the results for global inhibition and the bottom figure shows the results for local inhibition Only the first 200 iterations were shown, for clarity, as the remaining 800 propagated the trend.

occurring outside of that range. Given the extremely large error bars at some of those points, it is evident that the random initialization plays a large role in the permanence boosting mechanism. This is likely due to that mechanism's dependency on the degree of overlap between a column and its input, i.e. the inputs that a column randomly connects with will greatly influence the frequency of permanence boosting.

To further explore the permanence boosting mechanism, its boosting frequency was plotted for a sparsity of 74% in Figure 4.6. It is shown that there is a delayed

start, followed by a decaying exponential that falls until its minimum level is reached, at which point the overall degree of boosting remains constant. Additionally, this decaying exponential trend was common among the sparsities that resulted in a noticeable degree of permanence boosting. Comparing this to DeSieno's work [12], both boosting mechanisms follow a right-skewed decaying exponential. It is also interesting to note the size of the error bars. This once again reinforces the notion that network initialization plays an important role.

Given that both boosting mechanisms generally occur infrequently, for inputs that were expected to cause the largest degree of boosting, it is concluded that these boosting mechanisms are secondary learning mechanisms, with the primary learning occurring from the permanence update in (4.15). This observation allows resource limited systems (especially hardware designs) to exclude boosting, while still obtaining comparable results; thereby, greatly reducing the complexity of the system.

4.7 Exploring the Primary Learning Mechanism

To complete the mathematical formulation it is necessary to define a function governing the primary learning process. Within the SP, there are many learned components: the set of active columns, the neighborhood (through the inhibition radius), and both of the boosting mechanisms. All of those components are a function of the permanence, which serves as the probability of an input bit being active in various contexts.

As previously discussed, the permanence is updated by (4.16). That update equation may be split into two distinct components. The first component is the set of active columns, which is used to determine the set of permanences to update. The second component is the remaining portion of that equation, and is used to determine the permanence update amount.

4.7.1 Plausible Origin for the Permanence Update Amount

In the permanence update equation, (4.16), it is noted that the second component is an unlearned function of a random variable coming from a prior distribution. That random variable is nothing more than $\boldsymbol{X}$. It is required that $\boldsymbol{X}_{i,k} \sim \text{B}er(\mathbb{P}(\boldsymbol{X}_{i,k}))$, where B$er$ is used to denote the Bernoulli distribution. If it is assumed that each $\boldsymbol{X}_{i,k} \in \boldsymbol{X}$ are independent and identically distributed (i.i.d.)[12], then $\boldsymbol{X} \overset{i.i.d.}{\sim} \text{B}er(\theta)$, where θ is defined to be the probability of an input being active. Using the PMF of the Bernoulli distribution, the likelihood of θ given $\boldsymbol{X}$ is obtained in (4.24), where $t \equiv mq$ and $\overline{\boldsymbol{X}} \equiv \frac{1}{t}\sum_{i=0}^{m-1}\sum_{k=0}^{q-1}\boldsymbol{X}_{i,k}$, the overall mean of $\boldsymbol{X}$. The corresponding log-likelihood of θ given $\boldsymbol{X}$ is given in (4.25).

$$\begin{aligned}\mathcal{L}(\theta;\boldsymbol{X}) &= \prod_{i=0}^{m}\prod_{k=0}^{q}\theta^{\boldsymbol{X}_{i,k}}(1-\theta)^{1-\boldsymbol{X}_{i,k}} \\ &= \theta^{t\overline{\boldsymbol{X}}}(1-\theta)^{t-t\overline{\boldsymbol{X}}}\end{aligned} \tag{4.24}$$

$$\ell(\theta;\boldsymbol{X}) = t\overline{\boldsymbol{X}}\log(\theta) + (t - t\overline{\boldsymbol{X}})\log(1-\theta) \tag{4.25}$$

Taking the gradient of the joint log-likelihood of (4.25) with respect to θ, results in (4.26). Ascending that gradient results in obtaining the maximum-likelihood estimator (MLE) of θ, $\hat{\theta}_{MLE}$. It can be shown that $\hat{\theta}_{MLE} = \overline{\boldsymbol{X}}$. In this context, $\hat{\theta}_{MLE}$ is used as an estimator for the maximum probability of an input being active.

$$\boldsymbol{\nabla}\ell(\theta;\boldsymbol{X}) = \frac{t}{\theta}\overline{\boldsymbol{X}} - \frac{t}{1-\theta}(1-\overline{\boldsymbol{X}}) \tag{4.26}$$

[12]Recall that $\boldsymbol{X}_i$ contains the input observed by $\vec{\boldsymbol{c}}_i$, which is a subset of $\boldsymbol{U}_s$. Each $\boldsymbol{X}_i \in \boldsymbol{X}$ is determined independently, such that all $\boldsymbol{X}_i \in \boldsymbol{X}$ are guaranteed to be i.i.d. Each $\boldsymbol{X}_{i,k} \in \boldsymbol{X}_i$ is identically distributed, but not independent, thereby invalidating the i.i.d. assumption. However, if the initialization was altered such that the selection from $\boldsymbol{U}_s$ was done with replacement, then all $\boldsymbol{X}_{i,k} \in \boldsymbol{X}$ would be guaranteed to be i.i.d.

Taking the partial derivative of the log-likelihood for a single $\boldsymbol{X}_{i,k}$ results in (4.27). Substituting out θ for its estimator, $\overline{\boldsymbol{X}}$, and multiplying by κ, results in (4.28a). κ is defined to be a scaling parameter and must be defined such that $\frac{\kappa}{\overline{\boldsymbol{X}}} \in [0, 1]$ and $\frac{\kappa}{1-\overline{\boldsymbol{X}}} \in [0, 1]$. Revisiting the permanence update equation, (4.16), the permanence update amount is equivalently rewritten as $\phi_+\boldsymbol{X} - \phi_-(\boldsymbol{J} - \boldsymbol{X})$, where $\boldsymbol{J} \in \{1\}^{m\times q}$. For a single $\boldsymbol{X}_{i,k}$ it is clear that the permanence update amount reduces to $\phi_+\boldsymbol{X}_{i,k} - \phi_-(1 - \boldsymbol{X}_{i,k})$. If $\phi_+ \equiv \frac{\kappa}{\overline{\boldsymbol{X}}}$ and $\phi_- \equiv \frac{\kappa}{1-\overline{\boldsymbol{X}}}$, then (4.28a) becomes (4.28b). Given this, $\boldsymbol{\delta\Psi}$ is presented as a plausible origin for the permanence update amount. Using the new representations of ϕ_+ and ϕ_-, a relationship between the two is obtained, requiring that only one parameter, κ, be defined. Additionally, it is possible that there exists a κ such that ϕ_+ and ϕ_- may be optimally defined for the desired set of parameters.

$$\frac{\partial}{\partial\theta}\ell(\theta; \boldsymbol{X}_{i,k}) = \frac{1}{\theta}\boldsymbol{X}_{i,k} - \frac{1}{1-\theta}(1 - \boldsymbol{X}_{i,k}) \tag{4.27}$$

$$\boldsymbol{\delta\Psi}_{i,k} \equiv \frac{\kappa}{\overline{\boldsymbol{X}}}\boldsymbol{X}_{i,k} - \frac{\kappa}{1-\overline{\boldsymbol{X}}}(1 - \boldsymbol{X}_{i,k}) \tag{4.28a}$$

$$\equiv \phi_+\boldsymbol{X}_{i,k} - \phi_-(1 - \boldsymbol{X}_{i,k}) \tag{4.28b}$$

4.7.2 Discussing the Permanence Selection

The set of active columns is the learned component in (4.16), obtained through a process similar to competitive learning [13]. In a competitive learning network, each neuron in the competitive learning layer is fully connected to each input neuron. The neurons in the competitive layer then compete, with one neuron winning the competition. The neuron that wins sets its output to '1' while all other neurons set their output to '0'. At a global scale, this resembles the SP with two key differences.

The SP permits multiple columns to be active at a time and each column is connected to a different subset of the input.

Posit that each column is equivalent to a competitive learning network. This would create a network with one neuron in the competitive layer and q neurons in the input layer. The neuron in the competitive layer may only have the state of '1' or '0'; therefore, only one neuron would be active at a time. Given this context, each column is shown to follow the competitive learning rule.

Taking into context the full SP, with each column as a competitive learning network, the SP could be defined to be a bag of competitive learning networks, i.e. an ensemble with a type of competitive learning network as its base learner. Recalling that $\boldsymbol{X} \subseteq \boldsymbol{U}_s$, each $\boldsymbol{X}_i$ is an input for $\vec{\boldsymbol{c}}_i$. Additionally each $\boldsymbol{X}_i$ is obtained by randomly sampling $\boldsymbol{U}_s$ without replacement. Comparing this ensemble to attribute bagging [14], the primary difference is that sampling is done without replacement instead of with replacement.

In attribute bagging, a scheme, such as voting, must be used to determine what the result of the ensemble should be. For the SP, a form of voting is performed through the construction of $\vec{\boldsymbol{\alpha}}$. Each base learner (column) computes its degree of influence. The max degree of influence is q. Since that value is a constant, each $\vec{\boldsymbol{\alpha}}_i$ may be represented as a probability by simply dividing $\vec{\boldsymbol{\alpha}}_i$ by q. In this context, each column is trying to maximize its probability of being selected. During the inhibition phase, a column is chosen to be active if its probability is at least equal to the ρ_c-th largest probability in its neighborhood. This process may then be viewed as a form of voting, as all columns within a neighborhood cast their overlap value as their vote. If the column being evaluated has enough votes, it will be placed in the active state.

Chapter 5

SP for Machine Learning

Since its conception, there has been a gap between HTM and the machine learning community. This gap has largely been a result of the algorithm's origin. Many prominent machine learning algorithms originated from a rigours mathematical background, whereas HTM is based on the operating principles of the neocortex. A mathematical framework for the SP was presented in Chapter 4. This chapter is designed to bridge the gap between HTM theory and traditional machine learning concepts.

5.1 Software Implementation

It was desired to create a software implementation of the SP, based on the presented mathematical framework, that is readily usable by the machine learning community. Such a framework was created in Python and is dubbed math HTM (mHTM)[1] [15]. The implementation was made to be fully compatible with the popular Python machine learning library, scikit-learn [16].

The implementation of the SP is single-threaded; however, multiple forms of parallelizations for performing cross-validation (CV) and parameter optimization exist. Those parallelizations are supported for both local machines and clusters. The code is additionally platform independent and should run on any system containing the

[1]This implementation has been released under the MIT license and is available at: `https://github.com/tehtechguy/mHTM`. The application program interface (API) is documented at: `http://techtorials.me/mHTM`.

prerequisites. To use the code on a cluster, the cluster must be running SLURM, and as such must be a valid Linux distribution.

5.2 Data Encoding

An HTM learns on the data that is provided to it, as well as how that data is presented. Mapping an arbitrary input to a format that an HTM understands is handled by an encoder. This input must always be an SDR. Each bit in the SDR represents the activation state ('0' or '1' for inactive and active, respectively) of the columns from the previous region in the HTM. This input then serves as the feedforward input to the next region in the HTM.

In the case of the input to the system, the input data must be mapped from its representation to a suitable SDR. This mapping is performed by an encoder. The most trivial mapping occurs when the input is already in the form of an SDR. In that case, a unity encoder may be used, where the raw input is simply passed through.

Regardless of the input type, if it can be mapped to an SDR, the network will be able to process it. Choosing an appropriate encoder is critical, as a coarse-grained encoder (small number of bits per state) may not allow for enough distinction between inputs and a fine-grained encoder (large number of bits per state) may demand large memory and computation requirements or result in too large of an input distinction.

In addition to encoding, decoding is critical. An HTM region's output is an SDR. To obtain the original input, that SDR must be related to the input SDR, and then decoded. The process of mapping from a region's output SDR to its input SDR is explained in Section 5.3. With that in mind, the purpose of an encoder is to be able to take a supported input, convert it to an SDR, and then decode that SDR to obtain the original input. With a lossless encoder it is always possible to obtain the original input; however, a lossy encoder will only be able to obtain an input close to the original input. The closeness of a lossy encoder's decoded value to the original

value depends on the granularity of the encoder.

The ensuing subsections will explain the details of a few encoders[2]. While other encoders exist, most inputs may be satisfied through a unity encoder, category encoder, or a scalar encoder. In the event of multiple inputs, a multivariate encoder may be used.

5.2.1 Category Encoder

A category encoder is used to encode categorical data. Each category is represented by a user-defined set of contiguous bits, known as a bin. Each bit in the bin is set to the active state if its category is the one being encoded; otherwise, it is set to the inactive state. The bin for each category is concatenated together to form the final representation.

The number of bits in each bin is specified by the user. An increase in bits increases the degrees of freedom for each input. It also ensures that each bin is able to be properly observed. Since an HTM network only cares about active bits, having multiple active bits for a given input allows the system to more readily utilize its resources. Having a large number of bits also provides some benefits for noise tolerance, as the loss of some bits within a bin will not necessarily result in an incorrect encoding. The tradeoff exists with the desired amount of memory, as the total size of the SDR increases proportionally to the number of bits in the bin. This may then result in increasing the number of columns in the HTM.

In addition to defining the number of bits, the user may specify the bit length. The encoding will always be scaled to fit within the bit length. If too many bits are created, the number of bits for a given bin is decremented until the bit length is satisfied or there is no longer enough bits left to encode the data[3]. If the encoding

[2]Due to the informal definition of the encoders, the actual implementation of an encoder will vary based on the work. The encoders presented in this work provide details for one possible implementation.

[3]This occurs when the number of categories is greater than the number of bits.

is shorter than the desired bit length, it will be padded on the right-hand side with zeroes.

This implementation requires categories to be integers in the set of zero up to the number of categories less one. While categories can technically be defined to be any arbitrary object, integers simplify the design. Additionally, the number of categories must be known at the time of creation. Having a discrete set of categories, represented as integers, converts the categorical encoding problem into a simplified scalar encoding problem. This reduction allows this encoder to use a scalar encoder to perform the actual encoding and decoding, while still obtaining the desired result.

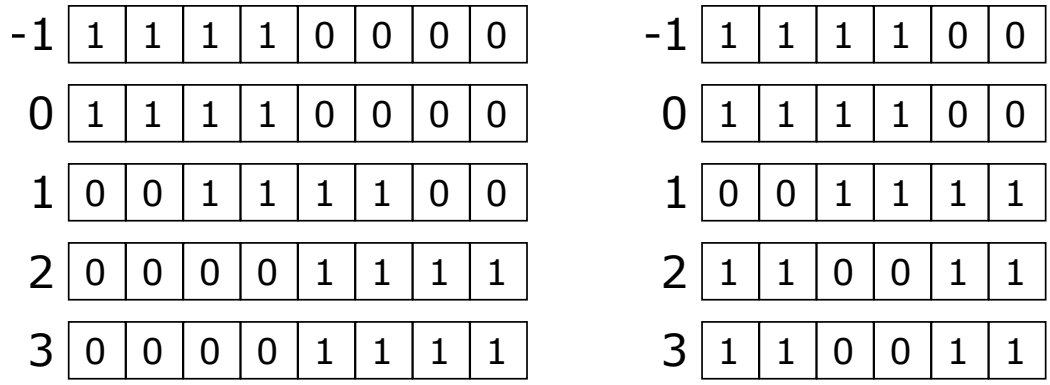
A category encoder is inherently a lossless encoder, as there exists a 1:1 relationship between encodings and categories. If the provided input to decode is lossy, the most likely category will be returned as the output. This decoding process is explained further in Section 5.2.2.

5.2.2 Scalar Encoder

A scalar encoder is used to encode scalars. Unlike the category encoder, the scalar encoder has an infinite set of possible inputs. That set must still be mapped to an SDR; therefore, like the category encoder, a finite set of bins is required. Determining the proper number of bins is very important, as this will set the level of granularity for the quantization process. Unfortunately, the current method for determining this parameter is through experimentation.

In addition to the number of bins, the encoder must know the minimum and maximum values of the data it is encoding. This allows the quantization to be uniformly distributed across that range. If a value is found that lies outside of that range, it is clamped to its respective limit (the minimum or maximum value).

As with the category encoder, the number of bits and the number of active bits are valid input parameters with a requirement of at least one of them being supplied. Two



(a) Without wrapping **(b)** With wrapping

Figure 5.1: Scalar encoding example demonstrating the cases without (a) and with wrapping (b). For both cases, four active bits and three bins were used with the bin overlap set to 70%. The encoding bounds are zero and two for the minimum and maximum, respectively.

additional parameters exist to further customize the encoding process: bin overlap and wrap. Those parameters are both techniques to allow a commonality between bins. This commonality enables a native way to denote similar inputs, but by doing so, the complexity for discretizing the bins is increased.

Bin overlap is a percentage used to define the degree of overlap between bins. For example, if four active bits are used with a bin overlap of 70%, there will be, at most, two bits of overlap ($\lfloor bin\ overlap * \#\ active\ bits \rfloor$) on either side of the bin. This means that the two right-most bits of the bin representing the value immediately lower than the current value will be shared with the two left-most bits of the current bin, and that the two right-most bits of the current bin will be shared with the two left-most bits of the bin representing the value immediately larger than the current value. This encoding is depicted in Figure 5.1a.

Wrap is used as a flag to denote if the encoding should wrap around to the other side. This can be used as a means to denote that the last element in the set shares some commonality with the first element in the set. In the case of the previous bin overlap example, if the current bin was the last bin, its two right-most bits would be the first two bits of the encoding. This encoding is depicted in Figure 5.1b.

Algorithm 5 Decode the SDR, *sdr*, to a scalar value.

```
    # Find the set of best matching bin indexes
 1: scalar_encoder_bins ← get_all_scalar_encoder_bins()
 2: bitwise_bins ← bitwise_and(bin, sdr) ∀bin ∈ scalar_encoder_bins
 3: n ← len(bitwise_bin) − 1
 4: bin_sums ← Σ_{b=bitwise_bin[0]}^{bitwise_bin[n]} (b) ∀bitwise_bin ∈ bitwise_bins
 5: best_bin_sum ← max(bin_sums)
 6: best_bin_indexes ← []
 7: i ← 0
 8: for all bin_sum ∈ bin_sums do
 9:     if bin_sum = best_bin_sum then
10:         best_bin_indexes.append(i)
11:     i ← i + 1

    # Determine the bin's value
12: median_value ← mean(get_bin_median(bin_ix)) ∀bin_ix ∈ best_bin_indexes
13: min_value ← mean(get_bin_min(bin_ix)) ∀bin_ix ∈ best_bin_indexes
14: max_value ← mean(get_bin_max(bin_ix)) ∀bin_ix ∈ best_bin_indexes
```

A scalar encoder is a lossy encoder, as an infinite number of scalars must be mapped to a finite number of bins. Due to this mapping challenge decoding SDRs into scalars is a non-trivial problem, where multiple decoding schemes may be used. The decoding scheme presented in this work aims to map a given SDR to the bin that it best matches. To do this, the given SDR is bitwise AND'd with each valid bin representation. The sum of the active bits is found, and each bin having a sum equal to the max sum is selected as the best bin. The average scalar representation of those best bins is taken as the decoded scalar value. The pseudocode for this procedure is shown in Algorithm 5. Since a bin is defined to be a range of scalars, the median bin value, minimum bin value, and maximum bin value are all computed. In most cases, the median bin value should be sufficient, but by also using the minimum and maximum bins a more precise representation of the number is obtained.

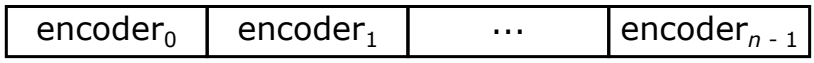


Figure 5.2: Depiction of a multivariate encoder with n encoders.

5.2.3 Multivariate Encoder

A multivariate encoder is a special type of encoder that supports the encoding of multiple inputs. This encoder takes multiple encoders as inputs and uses those encoders to create a new encoding. This encoder supports utilizing any other valid encoder, such as the unity encoder, scalar encoder, or category encoder. It additionally supports as many encoders as desired. An encoding from this encoder is obtained by concatenating the encodings of all of the internal encoders. The encodings are concatenated based on the order that they were originally provided to the multivariate encoder. This structure is depicted in Figure 5.2.

To encode a set of values, a list of tuples containing the value to encode and the encoder's index is supplied, respectively. For each encoder, if a corresponding tuple is supplied it is encoded using its encoder; otherwise, its encoding is an array of zeroes.

For decoding, this encoder iterates through all of its encoders, passing them the relevant portion of the input SDR to decode. The return value is then a list of tuples containing the encoded value and the encoder's index, respectively.

5.3 Methodology of Operation

HTM is defined to be an online algorithm, allowing an iterative training approach. In this manner, patterns are observed sequentially, with the system adapting after each pattern. It is possible to disable the learning mechanisms. Doing so enables the ability to extract the representation of the network after the provided set of training samples. This online learning approach is further to be applied for hierarchical regions, where each region performs learning after each pattern. While this approach works by

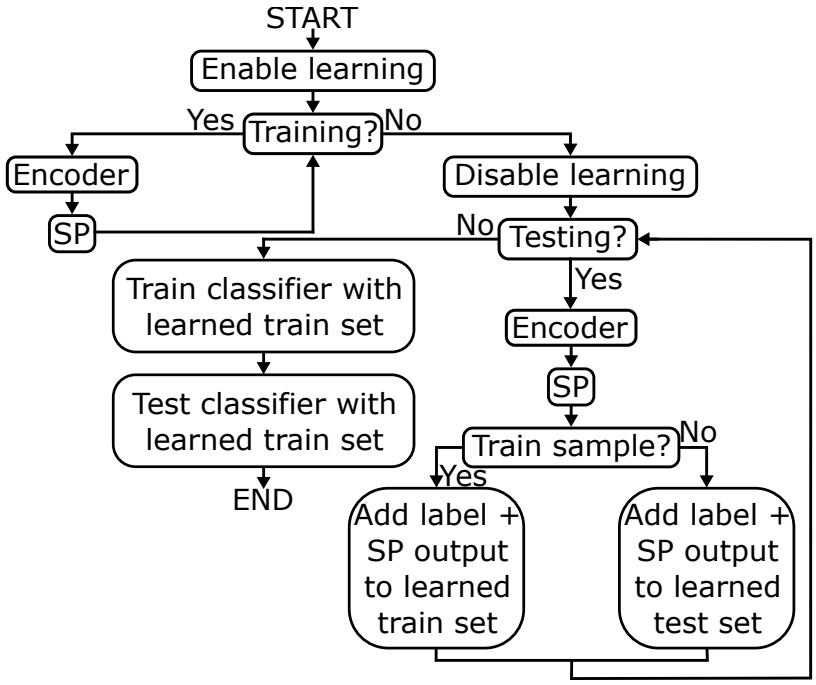


Figure 5.3: Procedure for classifying with a single SP.

design, it creates some difficulty in understanding which representation corresponds to which input.

The training strategy utilized in this work is shown in Figure 5.3. The SP is set to enable learning. For each training epoch, each training sample is encoded and sent to the SP. The SP then processes that encoded sample. Once the training process has finished, the learning mechanisms in the SP are disabled. The encoded training and testing sets are passed to the SP, respectively. The corresponding output of the SP, along with the class label are stored. Those new training and testing sets are passed to the classifier where it is then trained and used to compute the overall score.

If there exist multiple SP regions, the output of the previous SP is used as the input for the current SP. Assuming that only a final overall score is desired, only the output from the last SP is needed to train the classifier. It is also possible to use the representation of the other regions. In that case, the output of the desired region would need to be stored for both the training and testing sets.

In this hierarchical training scheme, each SP is training at the same time. It

may be desirable to instead train each level in the hierarchy independently. This would ensure that subsequent levels are only processing a static set of SDRs, since the lower level would no longer be training. This approach has other benefits, such as the ability to store intermediate outputs[4]. Unfortunately, if a constant stream of data is expected, this approach will likely have issues. This is because the system will have switched from training on a static set of SDRs to a new dynamic set; thus, the training context will have changed.

In the event that a TM follows an SP, a new methodology is required. The input to the TM will be the output from the SP; however, because it is unknown what the output represents, it must first be mapped to the input. This is done by using a classifier as shown in Figure 5.3. That classifier may then be used to easily tell which output SDR (from the SP below the TM) produced which input. In this manner, the output of the TM may be directly related back to the input of the system. This is obviously required, as it is needed to determine the validity of the TM's predictions.

Another interesting situation occurs with the use of TM. The TM needs to learn what input SDR follows the current input SDR. If the online learning approach is utilized, the set of input SDRs to the TM will be dynamic, thereby complicating the process of predicting the next SDR. Given these circumstances, it is likely that breaking the training into multiple parts will be desirable.

5.4 Feature Learning

A classical use case for the SP is to act as a pre-processing step for a classifier. In this context, the SP is learning a new representation of the input that should be easier for the classifier to classify. This is also the primary job of the SP, i.e.

[4] A system built to only utilize one SP could be trained by sequentially training each SP. Once the first SP has been trained, its output (with learning disabled) could be stored for both the training and testing sets. Those sets can then be used as input for a new system consisting of the SP for the next level.

mapping *similar* inputs to *similar* output SDRs. It is therefore clear that the SP is performing feature learning. This section explores what that entails both theoretically and experimentally.

5.4.1 Probabilistic Feature Mapping

It is convenient to think of a permanence value as a probability. That probability is used to determine if a synapse is connected or unconnected. It also represents the probability that the synapse's input bit is important. It is possible for a given input bit to be represented in multiple contexts, where the context for a specific instance is defined to be the set of inputs connected, via proximal synapses, to a column. Due to the initialization of the network, it is apparent that each context represents a random subspace; therefore, each column is learning the probability of importance for its random subset of attributes in the feature space. This is evident in (4.16), as permanences contributing to a column's activation are positively reinforced and permanences not contributing to a column's activation are negatively reinforced.

If all contexts for a given input bit are observed, the overall importance of that bit is obtained. Multiple techniques could be conjured for determining how the contexts are combined. The most generous method is simply to observe the maximum. In this manner, if the attribute was important in at least one of the random subspaces, it would be observed. Using those new probabilities the degree of influence of an attribute may be obtained. Let $\vec{\hat{\boldsymbol{\phi}}} \in (0,1)^{1\times p}$ be defined as the set of learned attribute probabilities. One form of $\vec{\hat{\boldsymbol{\phi}}}$ is shown in (5.1)[5]. In (5.1), the indicator function is used to mask the permanence values for $\boldsymbol{U}_{s,r}$. Multiplying that value by every permanence in $\boldsymbol{\Phi}$ obtains all of the permanences for $\boldsymbol{U}_{s,r}$. This process is used to project the SP's

[5]The function max was used as an example. Other functions producing a valid probability are also valid.

representation of the input back into the input space.

$$\vec{\hat{\boldsymbol{\phi}}} \equiv max\left(\boldsymbol{\Phi}_{i,k}\,\mathrm{I}(\boldsymbol{\Lambda}_{i,k} = r)\ \forall i\ \forall k\right)\ \forall r \tag{5.1}$$

5.4.2 Dimensionality Reduction

The learned attribute probabilities may be used to perform dimensionality reduction. Assuming the form of $\vec{\hat{\boldsymbol{\phi}}}$ is that in (5.1), the probability is stated to be important if it is at least equal to ρ_s. This holds true, as $\vec{\hat{\boldsymbol{\phi}}}$ is representative of the maximum permanence for each input in $\boldsymbol{U}_s$. For a given $\boldsymbol{U}_{s,r}$ to be observed, it must be connected, which may only happen when its permanence is at least equal to ρ_s. Given that, the attribute mask, $\vec{\boldsymbol{z}} \in \{0,1\}^{1\times p}$, is defined to be $\mathrm{I}\left(\vec{\hat{\boldsymbol{\phi}}} \geq \rho_s\right)$. The new set of attributes are those whose corresponding index in the attribute mask are true, i.e. $\boldsymbol{U}_{s,r}$ is a valid attribute if $\vec{\boldsymbol{z}}_r$ is true.

5.4.3 Input Reconstruction

Using a concept similar to the probabilistic feature mapping technique, it is possible to obtain the SP's learned representation of a specific pattern. To reconstruct the input pattern, the SP's active columns for that pattern must be captured. This is naturally done during inhibition, where $\vec{\hat{\boldsymbol{c}}}$ is constructed. $\vec{\hat{\boldsymbol{c}}}$, a function of $\boldsymbol{U}_s$, is used to represent a specific pattern in the context of the SP.

Determining which permanences caused the activation is as simple as using $\vec{\hat{\boldsymbol{c}}}$ to mask $\boldsymbol{\Phi}$. Once that representation has been obtained, the process follows that of the probabilistic feature mapping technique, where $\mathrm{I}(\boldsymbol{\Lambda}_{i,k} = r)$ is used as a second mask for the permanences. Those steps will produce a valid probability for each input bit; however, it is likely that there will exist probabilities that are not explicitly in {0, 1}. To account for that, the same technique used for dimensionality reduction is applied,

by simply thresholding the probability at ρ_s. This process is shown in (5.2)[6], where $\vec{\hat{\boldsymbol{u}}} \in \{0,1\}^{1\times p}$ is defined to be the reconstructed input.

$$\vec{\hat{\boldsymbol{u}}} \equiv \mathrm{I}\left(\left[max\left(\boldsymbol{\Phi}_{i,k}\vec{\hat{\boldsymbol{c}}}_i\,\mathrm{I}(\boldsymbol{\Lambda}_{i,k}=r)\;\forall i\;\forall k\right)\geq\rho_s\right]\right)\;\;\forall r \tag{5.2}$$

5.4.4 Experimental Results

mHTM was used to empirically investigate the performance of the SP. Since the SP should perform well with inherently spatial data, it was desired to evaluate the SP with a well-known computer vision task. Additionally, because the SP requires a binary input, it was desired to work with images that were originally black and white or could be readily made black and white without losing too much information. Another benefit of using this type of image is that the encoder may be de-emphasized, allowing for the primary focus to be on the SP. With those constraints, the modified National Institute of Standards and Technology's (MNIST's) database of handwritten digits [17] was chosen as the dataset.

The MNIST images are simple 28×28 grayscale images, with the bulk of the pixels being black or white. To convert them to black and white images, each pixel was set to '1' if the value was greater than or equal to $^{255}/_{2}$ and '0' otherwise. Each image was additionally transformed to be one-dimensional by horizontally stacking the rows. The SP has a large number of parameters, making it difficult to optimize the parameter selection. To help with this, $1,000$ independent trials were created, all having a unique set of parameters. The parameters were randomly selected within reasonable limits[7]. Additionally, parameters were selected such that $\mathbb{E}[\lambda'] = 0$. To reduce the computational load, the size of the MNIST dataset was reduced to 800

[6]The function max was used as an example. If a different function is utilized, it must be ensured that a valid probability is produced. If a sum is used, it could be normalized; however, if caution is not applied, thresholding with respect to ρ_s may be invalid and therefore require a new thresholding technique.

[7]The following parameters were kept constant: $\rho_s = 0.5$, 30 training epochs, and synapses were trimmed if their permanence value ever reached or fell below 10^{-4}.

training samples and 200 testing samples. The samples were chosen from their respective initial sets using a stratified shuffle split with a total of five splits. To ensure repeatability and to allow proper comparisons, care was taken to ensure that both within and across experiments the same random initializations were occurring. To perform the classification, a linear support vector machine (SVM) was utilized[8]. The input to the SVM was the corresponding output of the SP.

Three comparisons were explored for both global and local inhibition: using the set of active columns as the features (denoted as "column"), using $\vec{\hat{\phi}}$ as the features (denoted as "probabilistic"), and using the dimensionality reduced version of the input as the features (denoted as "reduction"). For each experiment the average error of the five splits was computed for each method. The top 10 performers for each method were then retrained on the full MNIST dataset. From those results, the set of parameters with the lowest error across the experiments and folds was selected as the optimal set of parameters.

The results are shown in Table 5.1 and Table 5.2 for global and local inhibition, respectively. For reference, the same SVM without the SP resulted in an error of 7.95%. The number of dimensions was reduced by 38.27% and 35.71% for global and local inhibition, respectively. Both the probabilistic and reduction methods only performed marginally worse than the base SVM classifier. Considering that these two techniques are used to modify the raw input, it is likely that the learned features were the face of the numbers (referring to inputs equaling '1'). In that case, those methods would almost act as pass through filters, as the SVM is already capable of determining which features are more / less significant. That being said, being able

[8]When a single SP region is used, the input is typically projected into a higher dimensional space. The SP, by definition, also produces a sparse output. Based on those details, any classifier following an SP would need to be able to work well with both high dimensional and sparse data. Since SVMs perform well with that type of data, the decision for using an SVM followed naturally. Additionally, an SVM should also work well with the raw input, allowing for a fair comparison between using the SP and not using the SP. A linear kernel was used over a nonlinear one, because the dimensionality is large. Also, using a linear kernel has the added benefit of greatly improving the execution time.

Table 5.1: SP Performance on MNIST using Global Inhibition[9]

Method	Error
column	7.70%
probabilistic	8.98%
reduction	9.03%

Table 5.2: SP Performance on MNIST using Local Inhibition[10]

Method	Error
column	7.85%
probabilistic	9.07%
reduction	9.07%

to reduce the number of features by over two thirds, for the local inhibition case, while still performing relatively close to the case where all features are used is quite desirable.

Using the active columns as the learned feature is the default behavior, and it is those activations that would become the feedforward input to the next level (assuming an HTM with multiple SPs and / or TMs). Both global and local inhibition outperformed the SVM, but only by a slight amount. Considering that only one SP region was utilized, that the SP's primary goal is to map the input into a new domain to be understood by the TM, and that the SP did not hurt the SVM's ability to classify, the SP's overall performance is acceptable. It is also possible that given a 2D topology and restricting the initialization of synapses to a localized radius may improve the accuracy of the network. Comparing global to local inhibition, comparable results are obtained. This is likely due to the globalized formation of synaptic connections upon initialization, since that results in a loss of the initial network topology.

To explore the input reconstruction technique, a random instance of each class from MNIST was selected. The input was then reconstructed as shown in Figure 5.4. The top row shows the original representation of the inputs. The middle row shows

[9]The following parameters were used to obtain these results: $m = 936$, $q = 353$, $\rho_d = 14$, $\phi_\delta = 0.0105$, $\rho_c = 182$, $\phi_+ = 0.0355$, $\phi_- = 0.0024$, $\beta_0 = 18$, and $\tau = 164$.

[10]The following parameters were used to obtain these results: $m = 786$, $q = 267$, $\rho_d = 10$, $\phi_\delta = 0.0425$, $\rho_c = 57$, $\phi_+ = 0.0593$, $\phi_- = 0.0038$, $\beta_0 = 19$, and $\tau = 755$.

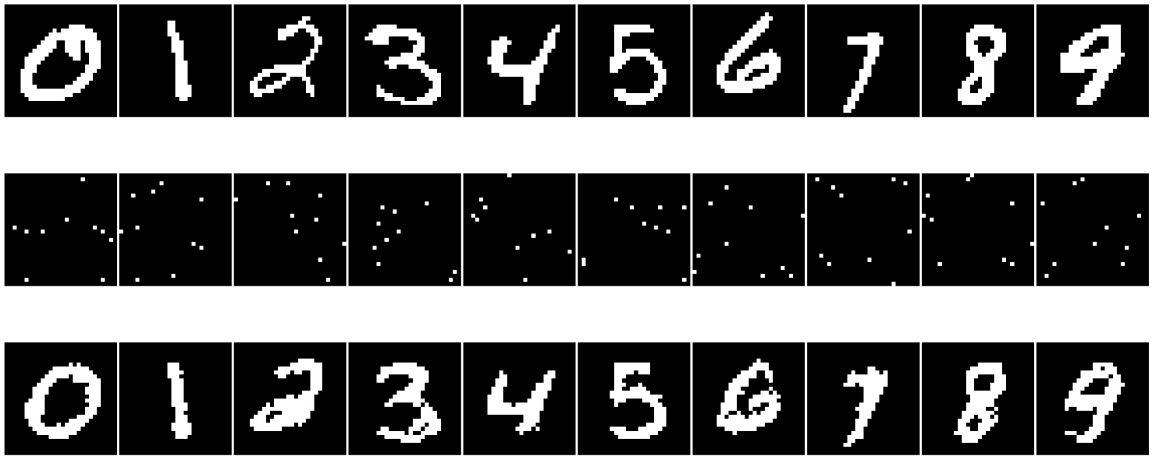


Figure 5.4: Reconstruction of the input from the context of the SP. Shown are the original input images (top), the SDRs (middle), and the reconstructed version (bottom).[11]

the SDR of the inputs. The bottom row shows the reconstructed versions. The representations are by no means perfect, but it is evident that the SP is indeed learning an accurate representation of the input.

[11]The following parameters were used to obtain these results: $m = 784$, $q = 392$, $\rho_d = 10$, $\phi_\delta = 0.01$, $\rho_c = 10$, $\phi_+ = 0.001$, $\phi_- = 0.002$, ten training epochs, global inhibition, and boosting was disabled. The number of columns was set to be equal to the number of inputs to allow for a 1:1 reconstruction of the SDRs.

Chapter 6

SDRs and Novelty Detection

6.1 Evaluating the SP's SDRs

The primary purpose of the SP is to map *similar* inputs to *similar* output SDRs. If it is able to perform this mapping, the TM should be able to make the proper predictions. Proving that the SP is doing such operations can be difficult, since the definition of *similar* changes by application. Additionally, there is a lack of methodology as well as metrics for proving that the SP is actually producing the desired result. This section provides some useful metrics for evaluating the SP's output SDRs[1].

6.1.1 Uniqueness Metric

The simplest technique for determining similarity is to evaluate the uniqueness of the sample set. If all of the SDRs are unique there is no similarity, and if all of the SDRs are identical there is complete similarity.

Let $\boldsymbol{W} \in \{0,1\}^{n \times m}$ be defined as the set of SDRs produced by the SP from input $\boldsymbol{U}$, such that $\boldsymbol{W}_s$ represents the SDR for input $\boldsymbol{U}_s$ and $\boldsymbol{W}_{s,i}$ represents bit i in $\boldsymbol{W}_s$. $\boldsymbol{W}_s$ is said to be unique if $\exists! \vec{\boldsymbol{w}} \in \boldsymbol{W} : (\vec{\boldsymbol{w}} = \boldsymbol{W}_s)$. Let nu be defined as the number of unique $\vec{\boldsymbol{w}} \in \boldsymbol{W}$. Let μ_u be defined as the uniqueness metric as shown in (6.1). μ_u is the number of unique SDRs in $\boldsymbol{W}$, normalized between zero and one. The metric

[1]The definitions are explicitly made in the context of the SP; however, the metrics may be used to evaluate any arbitrary input, provided that the appropriate substitutions are made. The definitions were made to be explicit for the SP to reduce the complexity of the equations.

is equal to '1' when all $\overrightarrow{\boldsymbol{w}} \in \boldsymbol{W}$ are unique and '0' when there are no unique SDRs. When used in the context of evaluating the SP's output SDRs, an ideal value for μ_u is zero.

$$\mu_u \equiv \frac{nu - 1}{n - 1} \tag{6.1}$$

This metric readily shows if the SP is performing perfectly; however, it is possible for this metric to be equal to one even when all of the SDRs are similar. That condition can occur if the SDRs differ by a relatively low number of flipped bits. To obtain better insight, it is wise to use this metric in conjunction with another metric.

6.1.2 Overlap Metric

Overlap is the distance metric used by the SP (see (4.11)). The SP works to maximize this distance, such that the number of overlapping bits within two given SDRs will be as large as possible. Since overlap is used within the SP, it is natural to use it as a similarity metric.

Referring to (4.11), overlap is nothing more than the dot product between two vectors. By using overlap as the distance metric, it is possible to compute the similarity of a collection of vectors. This is done by computing the average overlap across all pair-wise vectors. It can then be normalized by dividing that term by the maximum possible overlap value. The maximum overlap value that two vectors may obtain is the number of active bits in the vector whose sum of active bits is the second largest of the collection.

Let *ao* be defined as the average overlap across all pair-wise vectors in $\boldsymbol{W}$, as given by (6.2). In (6.2), the summations compute the sum of the overlap between all pair-wise vectors. To obtain the average, that term is divided by the the number of

elements in that sequence, which is the $(n-1)$th triangular number.

$$ao \equiv \frac{2\sum_{s=0}^{n-2}\left(\sum_{u=s+1}^{n-1}\left(\boldsymbol{W}_s \bullet \boldsymbol{W}_u\right)\right)}{n(n-1)} \tag{6.2}$$

Let mo be defined as the maximum overlap of two given vectors in $\boldsymbol{W}$, as given by (6.3). Since the overlap is the sum of the number of overlapping bits between two vectors, the maximum possible overlap is the number of the active bits in the vector having the second largest number of active bits. This number is technically biased, as it is possible that the actual maximum overlap is less than this value; however, it does provide a valid upper bound.

$$mo \equiv \mathrm{k}max\left(\sum_{i=0}^{m-1} \boldsymbol{W}_{s,i} \forall s, 2\right) \tag{6.3}$$

Let μ_o be defined as the overlap metric, as shown in (6.4). μ_o is the average overlap, normalized between zero and one. By using mo as an upper bound, it is ensured that this value may never exceed one. mo is additionally raised to one if it is zero to avoid division by zero. If μ_o is one, all of the SDRs are identical. If it is zero then all of the SDRs are unique. If it is somewhere in-between, it can provide a means of measure for determining the similarity amongst the SDRs.

$$\mu_o \equiv \frac{ao}{\mathrm{m}ax(mo, 1)} \tag{6.4}$$

6.1.3 SP Dataset

A custom dataset was developed for evaluating the SP's output SDRs. This dataset, simply named "SP dataset" (SPD)[2], is a generic dataset designed specifically for use in exploring the capabilities of the SP. SPD is a dynamic dataset whose specifics depend upon the provided user input.

[2]This dataset is included in mHTM. The class name is "SPDataset" and it is located in "mHTM.datasets.loader".

The dataset consists of only a single class that is utilized for generating SDRs. The user defines the number of bits that each SDR has, as well as the percentage of bits that should be active, i.e set to '1'. Based on those two parameters, the base class is created. The base class consists of a user-defined, randomly activated number of bits in an SDR initialized to be all zeroes. This base class represents the ideal representation for the dataset.

SPD may contain as many samples as the user desires. To generate samples, noise is added to the base class. A user-specified percentage of bits, i.e. the degree of noise, is inverted. Those bits are randomly determined, such that the exact number of desired bits to be inverted will be inverted.

SPD provides an easy way to determine how effective the SP is at producing generalizable SDRs. Since the determination of active bits is random and the other parameters are user-definable, SPD can be used to create a generalized representation for a specific dataset. It also has the added benefit of not being exclusively tied to a specific underlying dataset, but rather the state of the input. Recalling that the input that each column observes, $\boldsymbol{X}_i$, is randomly determined from the current input, $\boldsymbol{U}_s$, SPD should be able to approximate any single-class dataset. In that context, the number of bits in the SDR becomes q. The percentage of bits to make active is approximated by $\overline{\boldsymbol{U}}$. The degree of noise is approximated by subtracting the overlap metric of the input from one[3].

Since the dataset has only one class, the output from the SP would ideally be a single SDR. Additional samples are added to the dataset by perturbing the true representation of the input. Those samples are a function of the original input and noise, as such, it is easy to evaluate the SP's resilience to noise in the input.

It is also possible to use a modified version of SPD, where multiple classes are cre-

[3]The overlap metric of the input is μ_o, where all instances of $\boldsymbol{W}$ are replaced by $\boldsymbol{U}$ and all constants referring to the dimensions of $\boldsymbol{W}$ are altered to refer to the dimensions of $\boldsymbol{U}$. This same concept may be used to compute the overlap metric (or uniqueness metric) of any arbitrary input.

ated. Any secondary classes can be explicitly set to have as much similarity (overlap) with the primary class (and each other) as desired. This technique makes it possible to determine if the SP is not only producing similar SDRs for the primary class, but also filtering out classes that are not of the primary class.

6.1.4 Experimental Results

Using mHTM, the uniqueness and overlap metrics were evaluated[4]. A single SP was configured based on manual experimentation[5]. SPD was used as the dataset. 500 samples were created, each having 100 bits. 40% of the bits were set to be active. The experiment consisted of varying the degree of noise utilized to generate the dataset's samples. The noise was varied from 0 – 100%. For each percentage, ten trials were used, such that the repeatably of the experiment could be studied. In a trial, the random state for both the SP as well as the dataset were varied. The same random states were used across the various amounts of noise, to ensure a fair comparison.

During an individual trial, the SP was trained on all of the data in SPD. The SP was then tested on that same exact data. The uniqueness and overlap metrics were computed for both the SP's predicted SDRs as well as the raw input. The results are shown in Figure 6.1 and Figure 6.2 for the uniqueness and overlap metrics, respectively[6].

In an ideal situation, the SP would produce the same representation for all inputs until the noise level becomes large enough to change what the base class' representation is. For this experiment, that transition is expected to occur when the noise is

[4]These metrics, along with a number of other metrics, are all part of the "SPMetrics" class which is located in "mHTM.metrics".

[5]The SP parameters were $m = 200$, $p = 100$, $q = 75$, $\rho_d = 15$, $\rho_s = 0.5$, $\phi_\delta = 0.5$, $\rho_c = 50$, $\phi_+ = 0.001$, and $\phi_- = 0.001$. Synapses were trimmed if their permanences ever reached or fell below 10^{-4}, all boosting mechanisms were disabled, and ten training epochs were used. Those specific parameters were used, because they worked well for Section 6.2.2. This allowed for a nice comparison between the two experiments. Other parameters were found to produce "better" results. Those were avoided to ensure the presented data would not be misleading.

[6]Each point in the figure represents a partial box plot, i.e. the data point is the median, the upper error bar is the third quartile, and the lower error bar is the first quartile.

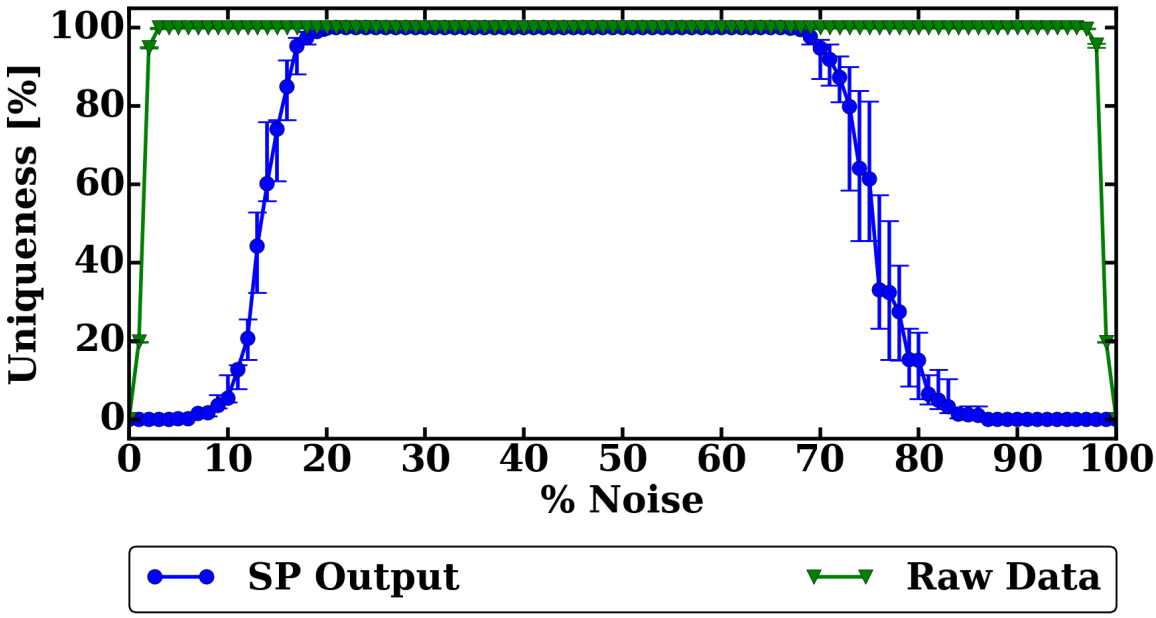


Figure 6.1: Uniqueness metric on SPD for varying degrees of noise.

around 20%. That number represents half of the number of active bits in the base class. With that much noise, it is possible for half of the originally active bits to be inverted. At that point, the base class would be on the verge of being lost, since only half of the bits between the base class and that new SDR would be overlapping. In reality, it is unlikely that those bits would be selected, so the SDRs should still be "similar" until the noise has exceeded 20%.

Referring to the results from the uniqueness metric, Figure 6.1, the SDRs in the dataset were completely unique once the amount of noise reached only 3%. At that point, the SP would now be responsible for filtering out the produced noise from the dataset. The SP was able to produce perfect representations until the noise reached 4%. It was then able to produce near perfect representations until the noise reached 7%, where the uniqueness metric increased from 0.10% up to 1.40%. It was not until the noise reached 14% that the uniqueness of SDRs exceeded 50%. Continuing from there, once the noise reached 21% all of the SDRs were unique. Based on that, it is likely that with a noise of 21% or more, it will be very hard to classify the output SDRs as similar. Before that level had been reached, it should have been possible to

classify the SDRs to be similar, assuming an appropriate classifier was used. Given that it was not until a noise level of 14%, where the majority of the SDRs became unique, the SP proved itself to be robust to noise in the input data. Additionally, the random state of the SP, as well as the dataset, appeared to have almost no affect on the SP's output representations, provided that the SP was still producing very few unique SDRs.

Referring to the results from the overlap metric, Figure 6.2, the raw data's overlap reached 50% at about a 21% noise level. Once the dataset's overlap fell below 50%, the original base class began to become lost. At that point, the SP was still able to produce output SDRs with a large amount of overlap, boasting an overlap of about 86%. Additionally, the SP's overlap did not reach 50% until the noise level was about 35%, at which point the overlap of the raw data was about 41%. This indicates that the SP was able to generalize the input SDRs even in the presence of a substantial amount of noise. Depending on how much noise is desired to be tolerated, the SP may have been slightly overfitting the data. That is because it was still able to classify the SDRs to be of the same type, even when the base class began to become lost.

It was also observed that the overlap metric proved to be robust to the random variation between trials, having almost no variation, regardless of the degree of noise. That property is extremely useful, as it alleviates the concern regarding the SP's initialization. It also implies that the SP should be robust to noise, regardless of the dataset, provided that the samples stay within a certain degree of noise from each other, and that the SP is properly trained. This is because any other given dataset (consisting of a single class), is also just a set of SDRs representing a common class. Assuming that those SDRs do indeed represent the same class, they should all be centered around a common base class SDR. That class could be approximated by computing the average bit activations (the average down the rows of $\boldsymbol{U}$) and thresholding each of those averages to either a '0' or a '1'.

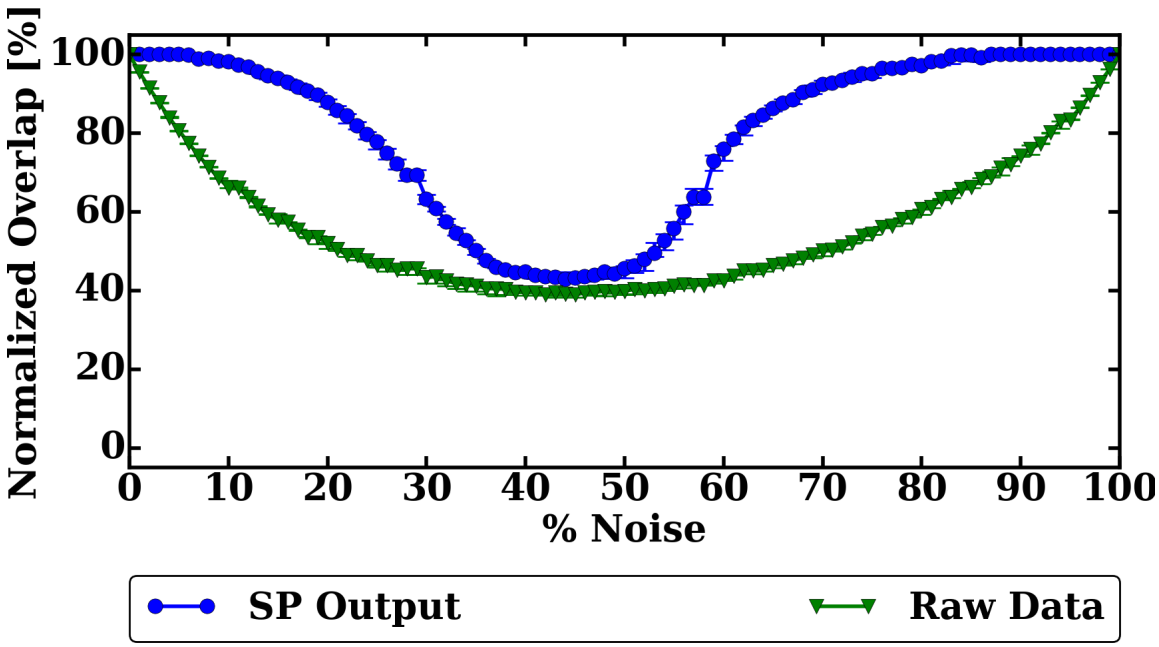


Figure 6.2: Overlap metric on SPD for varying degrees of noise.

From these results, it is clear that the SP is indeed creating a valid mapping from the input SDRs to a new set of SDRs. The uniqueness metric provided insight into how much noise the system can trivially tolerate; however, the metric was blind to the similarity between SDRs having almost identical representations. The overlap metric proved to be more robust, showing that the SP was able to produce recognizable SDRs up until the amount of noise became so large that the base class had changed.

These results indicate that it is possible for the SP to group a collection of input SDRs into a single category. This experiment; however, does not provide insight into the degree to which two overlapping or non-overlapping base classes may be separated. Additionally, it is possible to find a set of parameters for the SP that would result in a very tight distribution. While this at first glance would appear that the SP is performing extremely well, it may be misleading, as it is likely that the SP is over aggressively grouping SDRs together, i.e. overfitting. For this experiment, some overfitting is occurring; however, the degree is minimal, given that once the original base class becomes lost, the SP quickly regards the SDRs to be dissimilar.

6.2 Novelty Detection

Given that the metrics in Section 6.1 demonstrated that it is possible to determine the similarity among SDRs and that the SP was able to produce similar outputs for similar inputs, it is likely that the SP should be able to perform novelty detection. In novelty detection, the classifier is trained on the base class. It is then charged with classifying new items to either be within the base class or to be novel, i.e. outliers. The goal is to be able to easily detect any outliers. This section will explore utilizing the SP as a novelty detector. An emphasis will be placed on the SP's overall accuracy, i.e. it will be judged by how well it is able to accurately separate the base and novel classes.

6.2.1 SP Novelty Detection Classifier

To be able to use the SP for novelty detection, a classifier was designed. This classifier is dubbed SP novelty detection classifier (SPNDC). SPNDC utilizes the concept of the overlap metric at its core. Let $\boldsymbol{U}^{(tr)}$ and $\boldsymbol{U}^{(te)}$ be defined as the input SDRs for the training and testing datasets. The corresponding output SDRs for $\boldsymbol{U}^{(tr)}$ and $\boldsymbol{U}^{(te)}$ are $\boldsymbol{W}^{(tr)}$ and $\boldsymbol{W}^{(te)}$, respectively. The SP should be trained on $\boldsymbol{U}^{(tr)}$. After training, both $\boldsymbol{W}^{(tr)}$ and $\boldsymbol{W}^{(te)}$ are produced.

Using $\boldsymbol{W}^{(tr)}$, this classifier begins by computing the probability that a given bit within an SDR in $\boldsymbol{W}^{(tr)}$ should be active. This probability is defined to be $\vec{\boldsymbol{\pi}}_w$ as given by (6.5)[7]. That probability is then thresholded such that all bits in $\vec{\boldsymbol{\pi}}_w$ that are at least equal to 0.5 are set to '1' and all other bits are set to '0'. This thresholded vector, defined to be $\vec{\boldsymbol{w}}_0$, is now an SDR. $\vec{\boldsymbol{w}}_0$ is taken to be the generic representation

[7] n in this context represents the number of vectors in $\boldsymbol{W}^{(tr)}$.

for all of the SDRs in $\boldsymbol{W}^{(tr)}$, i.e. $\vec{\boldsymbol{w}}_0$ represents the base class.

$$\vec{\boldsymbol{\pi}}_w \equiv \frac{1}{n} \sum_{s=0}^{n-1} \boldsymbol{W}_s^{(tr)} \tag{6.5}$$

To determine whether a given input is of the base class or the novel class, SPNDC computes μ_o between $\vec{\boldsymbol{w}}_0$ and the provided input. If μ_o is at least equal to 0.5, the input is classified to be part of the base class; otherwise, it is classified to be part of the novel class. Applying that strategy to all of the SDRs in $\boldsymbol{W}^{(tr)}$ and $\boldsymbol{W}^{(te)}$, the full dataset is classified.

6.2.2 Experimental Results

Using mHTM and SPNDC, an experiment was conducted to study the SP's ability to perform novelty detection. A single SP was configured after performing some manual optimizations[8]. A variant of SPD was used as the dataset. 1,000 samples were created, 800 of those samples were used to form $\boldsymbol{U}^{(tr)}$. Each sample contained 100 bits, and 40% of the bits were set to be active for the base class. A second dataset was created, also using SPD. Instead of this datset's base class being randomly constructed, as with the first dataset, it was created based on the original dataset's base class. This new base class was adjusted to have a certain percentage of overlapping bits with the original base class. If this percentage was zero, the original base class and the new base class would have no overlapping active bits. If the percentage was 100, the two base classes would have all 40 of their active bits overlapping, i.e. they would be identical.

Similar to the experiment used in evaluating the metrics (Section 6.1.4), the amount of noise used to generate additional samples was varied. Additionally, the percentage of overlapping bits between base classes was varied. All possible pairs of

[8]The SP parameters were $m = 200$, $p = 100$, $q = 75$, $\rho_d = 15$, $\rho_s = 0.5$, $\phi_\delta = 0.5$, $\rho_c = 50$, $\phi_+ = 0.001$, and $\phi_- = 0.001$. Synapses were trimmed if their permanence value ever reached or fell below 10^{-4}, all boosting mechanisms were disabled, and ten training epochs were used.

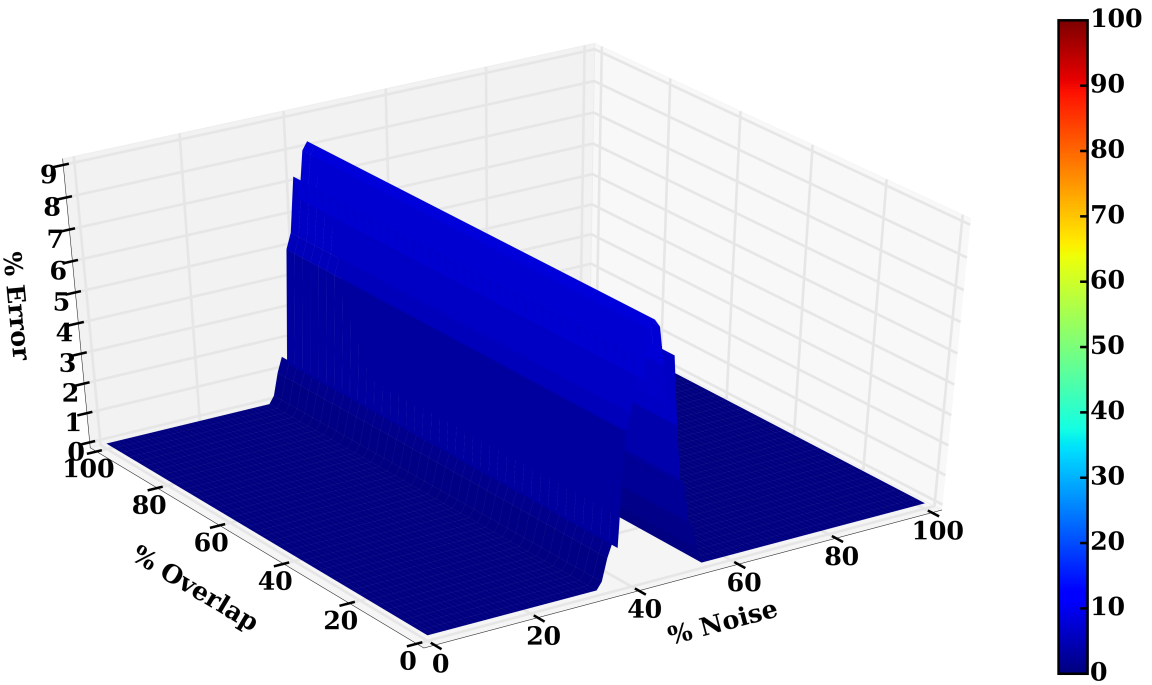
those values were varied, such that data was generated for each single noise and bit overlap pair. For each one of those pairs, ten trials were conducted. In a trail, the random state for both the SP as well as the dataset were varied. The same random states were used across the various amounts of noise, to ensure a fair comparison.

$\boldsymbol{U}^{(te)}$ was created by using the remaining 200 SDRs from the first dataset as well as an additional 200 SDRs generated from the second dataset. The first dataset was used to represent the base class and the second dataset was used to represent a different, novel, class. Evenly dividing the number of samples in each class allows for the overall accuracy to be evenly representative of each class, such that an accuracy of 50% indicates that the classifier was unable to distinguish between the base and novel classes.

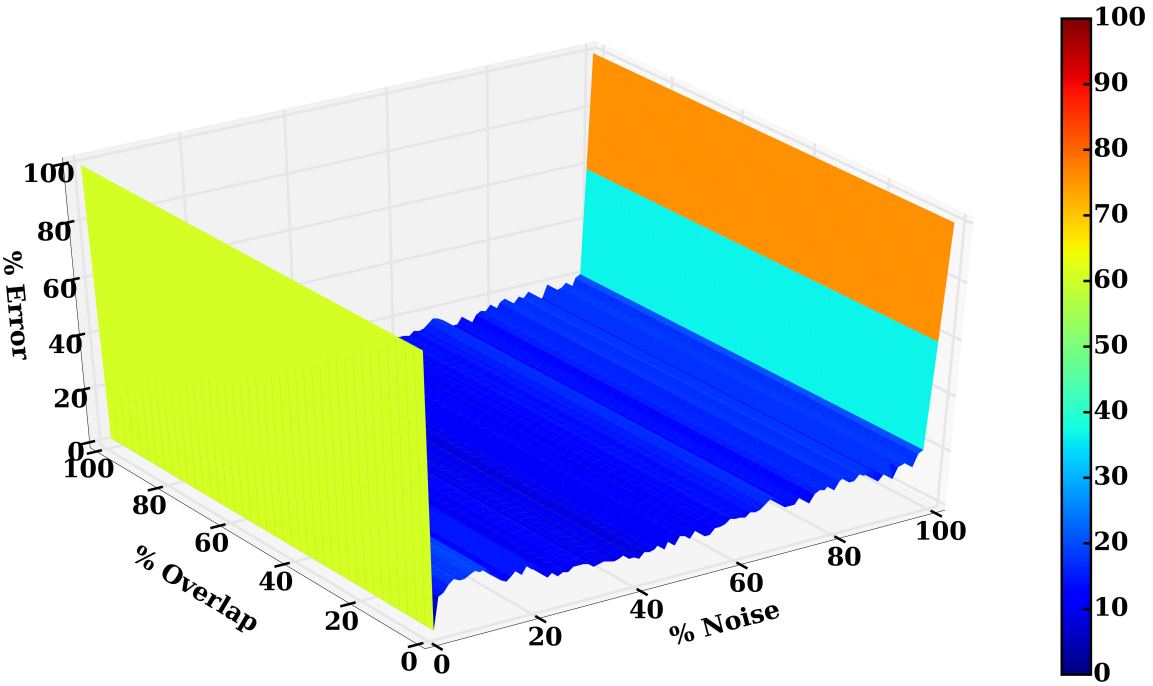
In addition to using the SP with SPNDC, a one-class SVM was used. The SVM utilized a linear kernel and had ν equal to 0.1. ν was set to allow a training error of 10%. A larger value of ν resulted in the SVM poorly classifying the base class and a smaller value resulted in it poorly classifying the novel class.

The training results for the SP and the SVM are shown in Figure 6.3. The SP (Figure 6.3a) shows that it was able to produce a very good fit, across the board. There did not appear to be any impact on training error with regards to overlap. For noise, the error did not increase above 0% until the noise reached 34%, at which point the error was only 0.25%. The maximum training error occurred for a noise of 43%, with the error being 8.25%. Given these results, it is clear that the SP was able to accurately detect the base class, even in the presence of noise or when the overlap between the base classes was large.

The SVM (Figure 6.3b) struggled at the boundary conditions. When the classes were made to be identical, the SVM was unable to classify them correctly. This is to be expected, since there is no distinction between the base and novel classes. To provide some additional visibility, the noise was restricted to be between 5 and 95%,



(a) SP training error.



(b) SVM training error.

Figure 6.3: Novelty detection, training, results for the SP (a) and the SVM (b).

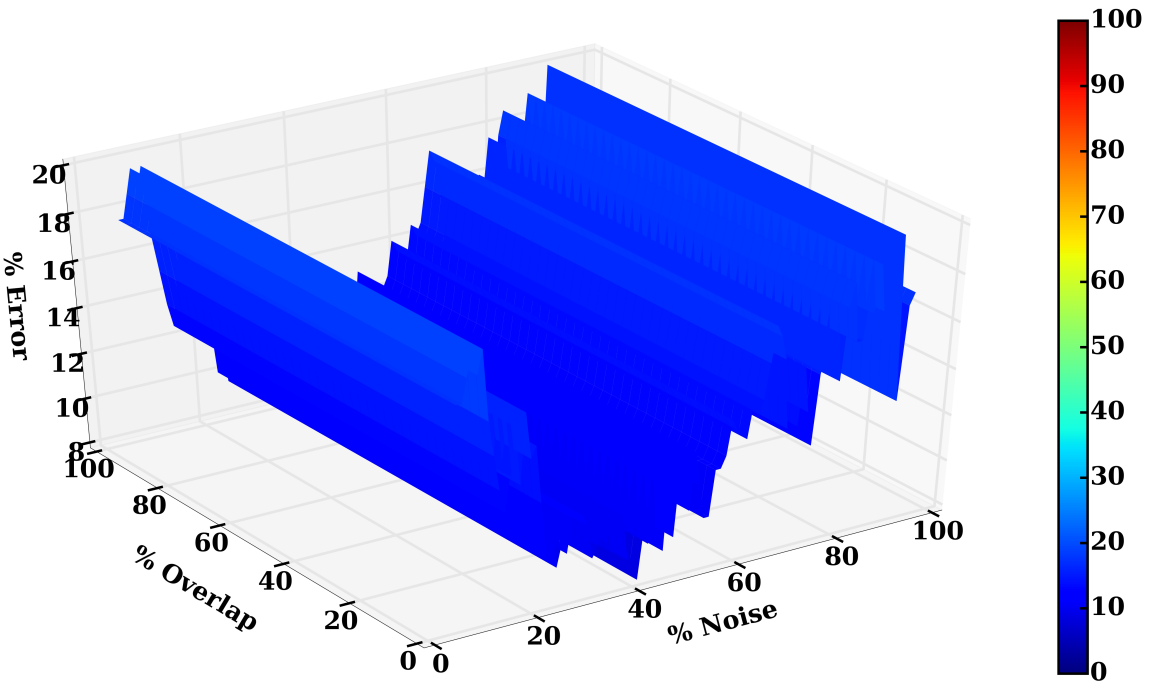


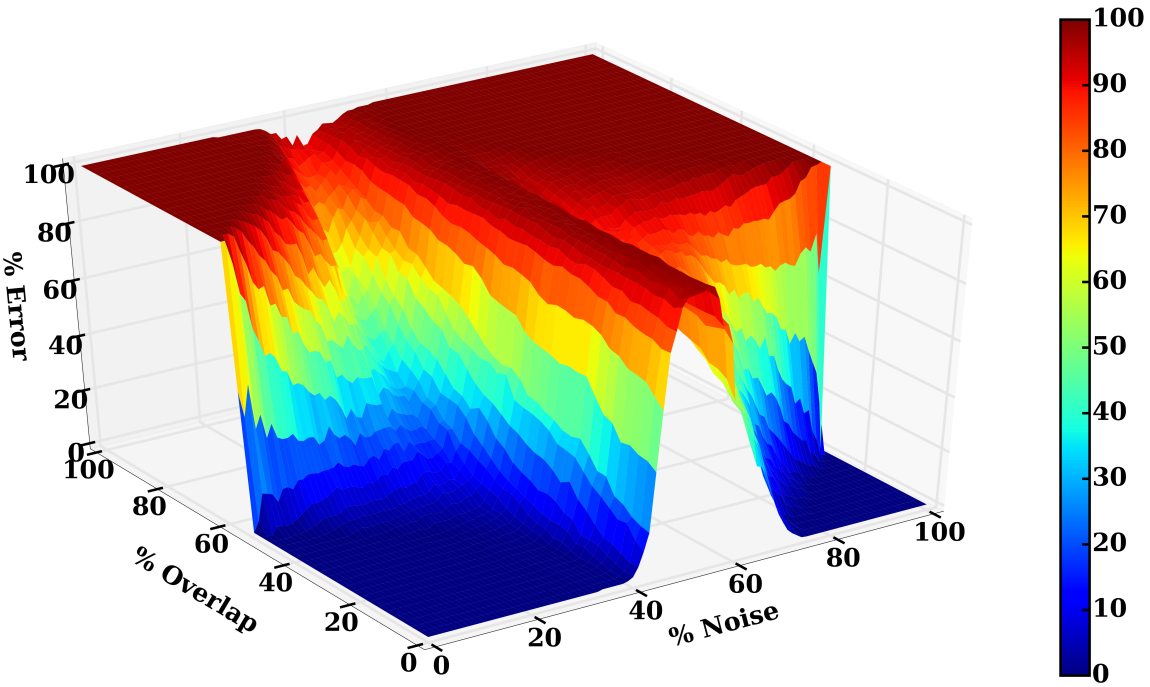
Figure 6.4: Restricted version of Figure 6.3b, where the noise ranged between 5 and 95%.

as shown in Figure 6.4. From that figure it is clear that the SVM did not fit the base class as well as the SP. The lowest error the SVM obtained was 8%[9], which was only marginally better than the SP's worst error. The highest error the SVM reached was 20%, which indicates that it was struggling to fit the data, since the expected error should be 10%. As with the SP, varying the overlap did not appear to affect the error.

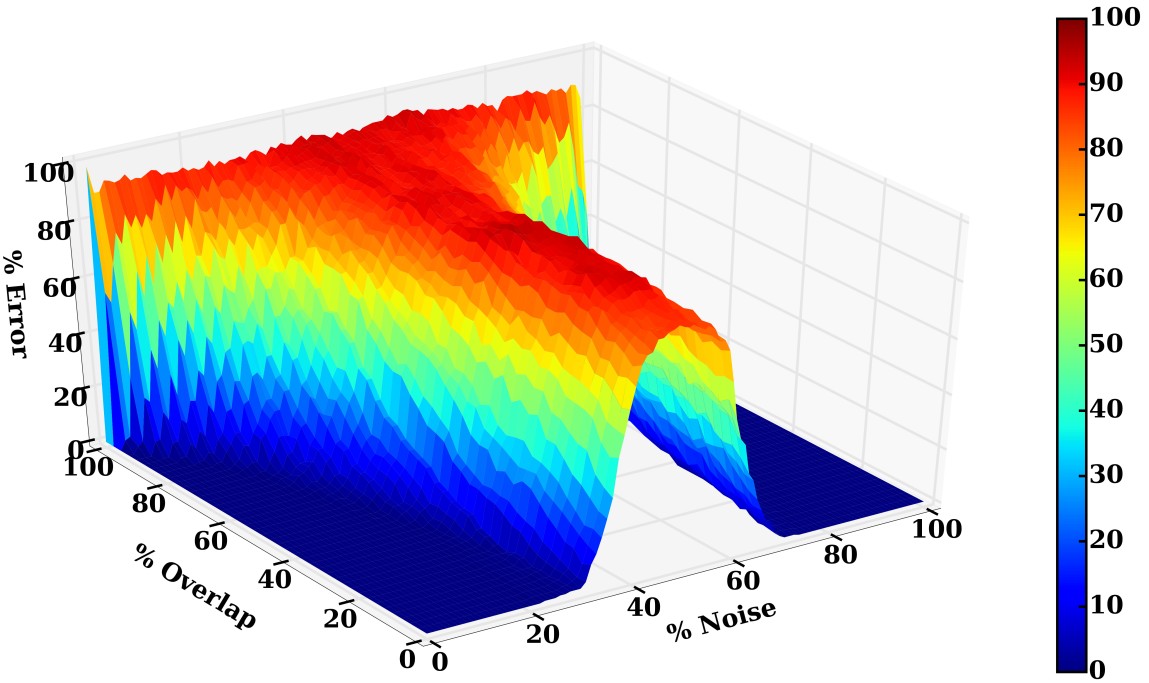
The testing results for the SP and the SVM are shown in Figure 6.5. The SP (Figure 6.5a) showed a robustness to noise. The error only began to increase once the noise reached about 40%. Unlike the training results, varying overlap had a large impact on the test error. The SP was able to perform well, provided that the overlap was below 40%. Once the overlap exceeded that threshold, the error steadily increased to its maximum value.

The SP's results closely followed the results that occurred from the metric experiments (Section 6.1.4). Once the noise exceeded 35%, it became difficult to classify

[9]That occurred when the noise reached 35%. Interestingly, the SVM's best fitting error was near where the SP began to no longer have a perfect fit.



(a) SP testing error.



(b) SVM testing error.

Figure 6.5: Novelty detection, testing, results for the SP (a) and the SVM (b).

the data. Forcing the two base classes to have overlapping bits greatly increases the complexity of the problem. Once 50% of the bits are overlapping, it should no longer be possible to separate the two classes. Based on the results in Figure 6.5a, that theory is supported. If however, both the overlap and noise are at their respective thresholds, it could be possible to get "lucky" due to the chaotic nature of the input. This was observed in Figure 6.5a, where the error was not 100% even when 90+% of the bits were overlapping.

Compared to the SP, the SVM (Figure 6.5b) was not as robust to noise. Once the noise was around 30%, the SVM's error began to struggle with performing the correct classifications. The amount of noise the SVM could tolerate was further reduced as the number of overlapping bits increased. Unlike the SP, the SVM was able to tolerate a much larger degree of overlapping bits, without sacrificing the error rate. This is likely a result of the SVM's ability to take into consideration not only the active bits, but also the inactive bits. There is likely a correlation between an input's class and which bits remain inactive, within a given dataset. Since the SP is restricted to learning based on the active bits, it is unable to take that into consideration. It may; however, be possible to construct a classifier (or use a pre-existing one) that is able to take that into consideration, utilizing $\boldsymbol{W}^{(tr)}$ as the training set.

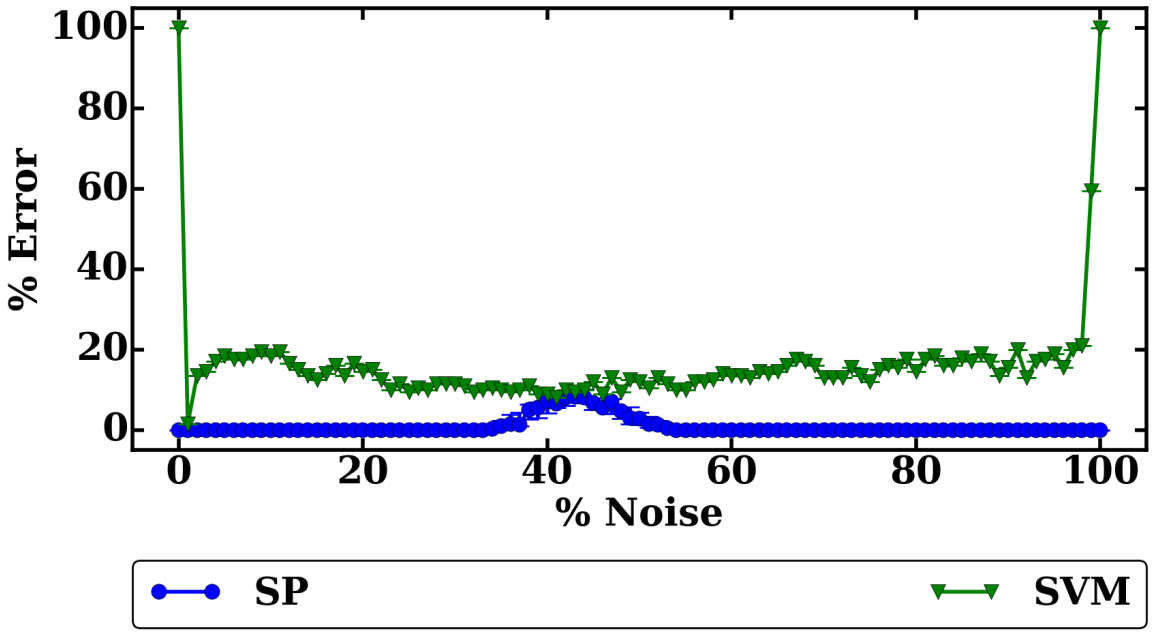
Revisiting Figure 6.5a, it was desired to further study the details of how well the SP performs on the verge of chaos. To study this, two slices from both Figure 6.3 and Figure 6.5 were taken. The first slice was taken with the overlap fixed at 0% and the noise varied. The results for training and testing for both the SP and the SVM are shown in Figure 6.6. Figure 6.6a reiterates that the SP was more robust than the SVM to noise in the training set. The SVM's error was around 20%, whereas the SP's was around 0%, excluding a slight bump where the noise became 40%. Figure 6.6b shows that the SP also performed better on the test set. The SP's test error did not begin to increase until around 40%, whereas the SVM's test error began to increase

at about 30%. Figure 6.6 can be used as a baseline, showing how the two algorithms should respond to noise in an ideal situation.

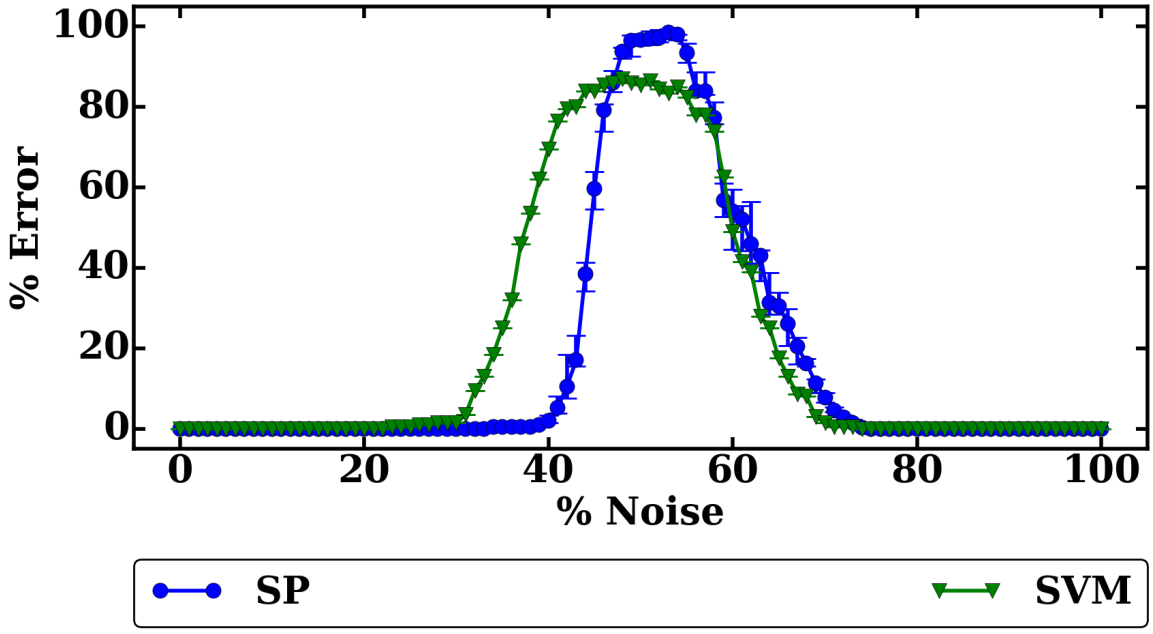
The second case was studied with the noise fixed at 35% and the overlap varied. This high level of noise was chosen, as it marks the point at which the SP was just beginning to struggle. The results for training and testing for both the SP and the SVM are shown in Figure 6.7. The training results, Figure 6.7a, were perfect for both classifiers. The SP had a constant zero error and the SVM had a constant 10% error.

The test results, Figure 6.7b, showed a very interesting trend. With no overlap, the SVM's error started at just above 20%. This error then steadily increased as the number of overlapping bits increased. With an overlap of only 20%, the SVM was already bordering an error rate of 50%. Based on this, it is clear that the SVM is unable to handle the chaotic nature of this dataset.

The SP proved to be incredibly robust, having only 0.5% test error until the overlap reached 25%. Even with the chaotic nature of the input, the SP did not begin to show any substantial increase in error until the overlap was about 40%. At that point, the SP's error was only 6%, whereas the SVM's error had already grown to 60%. As expected, the error bars for the SP became non-negligible once the overlap became large. Taking that into consider, the SP did not reach an error of 50% (with error) until the overlap reached 70%.

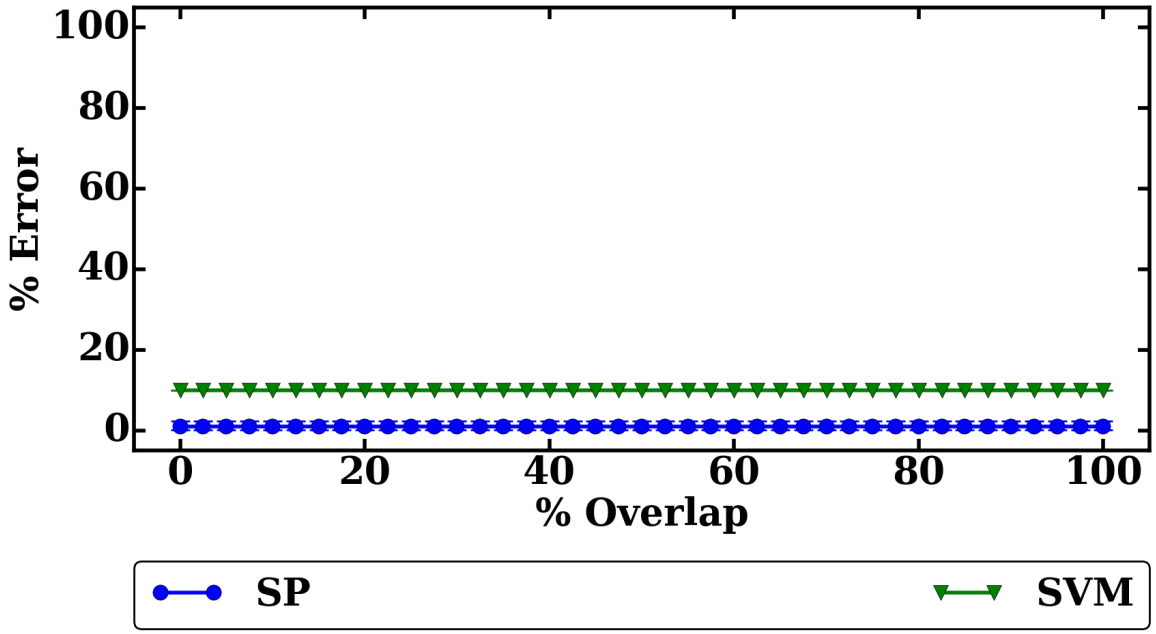


(a) Training error.

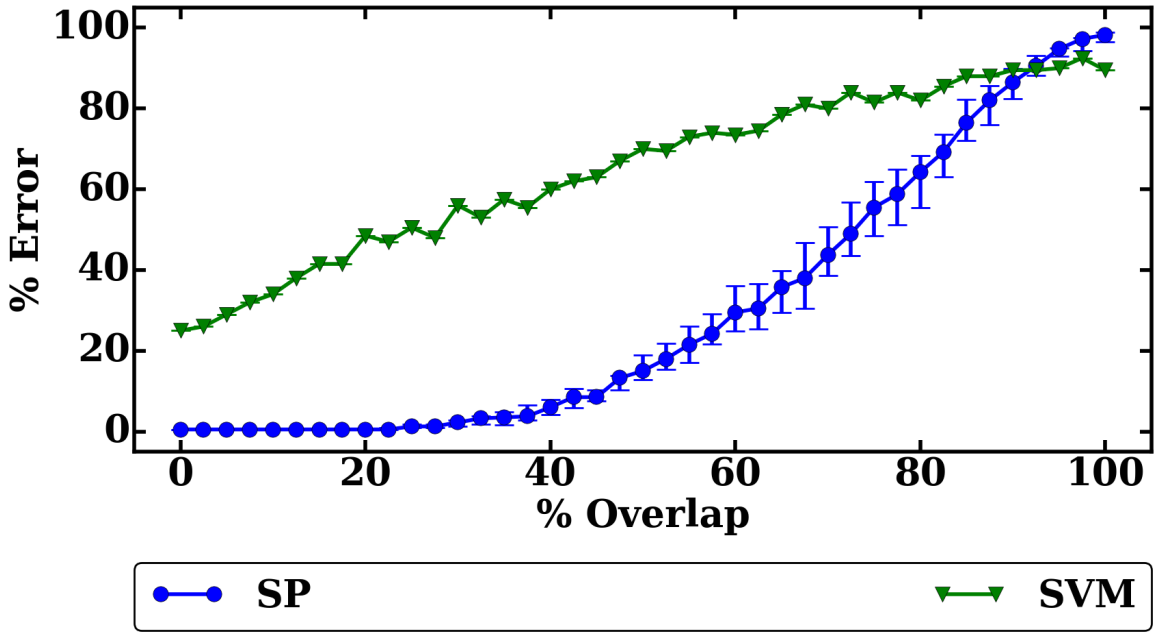


(b) Testing error.

Figure 6.6: Novelty detection results for non-overlapping inputs. The training (a) and the testing (b) results are shown.



(a) Training error.



(b) Testing error.

Figure 6.7: Novelty detection results for samples generated with a noise variation of 35%. The training (a) and the testing (b) results are shown.

Chapter 7

Final Remarks

7.1 Summary of Work

This work took the biologically inspired HTM SP algorithm and created a mathematical framework for it. Using that framework, it was demonstrated how the SP can be used within machine learning. In particular, the SP was demonstrated to perform feature learning. This was demonstrated in the context of a multi-class classification problem. It was shown how using the SP's learned features, the SP may be used to perform dimensionality reduction. Additionally, a reverse-mapping technique was created. That reverse-mapping allows for the SP's output to be converted back into the input domain.

The primary learning mechanism of the SP was explored. It was shown that the mechanism consisted of two components, the permanence selection and the degree of permanence update. A plausible estimator was provided for determining the degree of permanence update, and insight was given on the behavior of the permanence selection. The behavior was compared to well-known machine learning algorithms.

Methods for evaluating the SP's produced SDRs were provided. One such method utilized the SP's inherent distance metric, overlap. It was shown that the SP is extremely robust to noise. The SP was able to group inputs as similar until the noise became so large that the original input class was lost.

Using the created overlap metric, the SP was designed for use in novelty detection. A classifier was created to provide the SP with this capability. The SP was then evaluated for different levels of noise. It was also evaluated with respect to the similarity between the base and novel classes. The SP proved to be more robust to noise than an SVM.

The conducted work may be used to bridge the gap between HTM and the machine learning community. Substantial evidence was presented showing the use of the algorithm. Additionally, an open-source implementation of the algorithm was created. That implementation, mHTM, includes all of the code utilized in this work. mHTM should prove to be a solid foundation for future studies, as well as for using the SP as-is in other machine learning venues.

7.2 Parameter Optimization

The topic of parameter selection and how to optimize those parameters is always important in any parameterized algorithm. The SP, unfortunately, has a very large number of parameters. As shown in this work, finding suitable parameters is a non-trivial process. Techniques such as randomly searching a reduced space and manually tunning the parameters were utilized. Those techniques are by no means the best methods for determining such parameters. It is quite likely there exists a different set of parameters for which better results may be obtained.

There are; however, some general guidelines to follow. The permanence increment and decrement amounts should be made low ($\sim 10^{-3}$). They can somewhat be thought of as learning rates. A theory regarding optimizing those was presented in Section 4.7.1. If these parameters are set to be too large, the SP produces a very coarse mapping. If the parameters are set to be too small, a fine grain mapping occurs. These parameters are also directly related to determining the number of epochs.

In general, it is possible to use a very low number of training epochs. This is

much unlike traditional neural networks that may require hundreds of epochs. It is even possible to get away with only using a single training epoch; however, this will typically not produce the best results. Having ten or so epochs seems to be sufficient when the permanence update amounts are of a 10^{-3} magnitude. Additional epochs did not appear to provide any extra benefit and may in fact lead to overfitting.

The permanences should always be randomly initialized. Initializing the permanences based on the synapses' relative distance between their source and destination columns produced extremely poor results in all examined cases. When initializing the permanences, using a small window will allow the permanences to become connected more readily. This comes at the risk of too greedily forming synaptic connections. Some benefits may be found adjusting this parameter; however, in general using 0.5 should be sufficient.

Trimming permanences seemed to have little effect. Depending on the implementation, this may improve performance. If that is the case, it is recommended to use trimming. A value of 10^{-4} was found to have no negative impact.

Boosting greatly slows down the execution of the network. In rare cases it may prove to be useful; however, in general it was found to not add any benefit. Based on that, it is highly recommended to disable boosting. The current boosting mechanism requires improvement, but it is likely that it can be improved to address this issue.

Global inhibition was found to perform similarly to local inhibition. Global inhibition has the added benefit of significantly reducing execution time. It is recommended to only use local inhibition when it is clear that the application will benefit from it. Refer to Section 7.4 for additional remarks.

After using those guidelines only four parameters remain to be discussed: the number of columns, the number of active columns, the number of synapses, and the segment threshold. For optimizing these parameters, experimentation will be required. It is possible to use a search optimization algorithm to optimize the param-

eters. That being said, there are still some general guidelines that may be used to reduce the search space.

The number of synapses should be as large as possible. Referring back to Section 4.7.2, it was discussed how the SP may be performing a type of attribute bagging. In this context, the number of synapses dictates how many attributes each column will observe. If this number is equal to the number of input bits, then all attributes will be observed. In certain circumstances, this may be ideal; however, in general it can lead to overfitting. It is best to choose a large number of synapses, such that an adequate representation of the sample is selected.

The segment threshold is tightly coupled with the number of synapses parameter. This parameter may often be fickle, such that changing it by as little as ± 1 can drastically affect the results. The safest choice is always zero. Using zero ensures that there will always be adequate representation of the sample. While this is the safest, it is not the best. This parameter should be optimized to be as large as possible. As the value increases in size, the SP will become more robust to noise. If the parameter is too large, it is possible to completely lose any sort of representation, and if it is too small it is likely that too much noise is being introduced into the system. The discussion in Section 4.2 may be used to help determine this parameter's upper bound.

Choosing the appropriate number of columns is tricky. A good starting point is to set it to be equal to the number of bits in the input. This will provide a minimally adequate level of coverage. It is possible to utilize a small number of columns, i.e. less than the number of input bit; however, doing so may result in a less stable network. If there are too many columns, the network will be more likely to overfit the training data. If the input data is very chaotic, it may be likely that more columns will be required. As a starting upper bound, it should not be necessary to use more than $4\times$ the number of input bits. This is by no means a guarantee, but rather a

starting point. It was found that increasing the number of columns quickly provided diminishing returns. Additionally, too many columns hurt the system's accuracy. Ideally, this parameter should be optimized to be as small as possible. The smaller this parameter is, the shorter the execution time will be.

The last remaining parameter is the number of active columns. This parameter may also be set to be a percentage. No real difference was observed when using a constant vs. a density. For simplicity, the use of a constant is recommended. This parameter is tightly coupled with the number of columns, and should always be smaller than the number of columns. Ahmad et al. [18] suggest that this parameter be set to make m/p very small (e.g. 0.02). Having a small ratio has shown to improve the training accuracy; however, when the ratio is too small a loss in the testing accuracy was observed. This was found to occur for both the multi-class classification and novelty detection cases. Based on that evidence, it is recommended to use a ratio of m/p in the range of 0 – 30%. Since there is a tight coupling of all of the parameters, it is still likely that a low ratio will work, given alternative parameters.

Keep in mind that all of these provided suggestions are what was found to work best for this work. For any given application, experimentation should be used to determine the "ideal" selection of parameters.

7.3 Scalability

When considering using the SP for use in an application, it is important to evaluate how well it will scale. The exact scalability may vary based on the specific implementation. mHTM was designed to scale with respect to the number of columns. In this design paradigm, each phase becomes embarrassingly parallel. If this is exploited, this will obviously greatly reduce the computation time; however, it will also increase memory and power consumption.

One factor affecting scalability is the number of synapses. If the implementation

is made to scale with respect to the number of columns, it will likely not be able to scale with respect to the number of synapses. Adding additional synapses will increase both computation time as well as memory consumption.

Given that the SP will likely be used for spatial related tasks or in conjunction with the TM, it is very important to take into consideration the size of the input. If the input is an RGB image, then it will first need to be encoded. If it is desired to keep information on each channel, the input size is effectively tripling (a result of the three channels). If color is required to be kept, the pixel values will need to be encoded into an SDR. This could result in each pixel consuming many bits. For example, if each pixel is desired to be represented by only five bits, and the image size is a mere 28×28 pixels, p would become 11,760[1]. From this simple example, it becomes apparent that to construct a large-scale system, parallelizations will be required.

For improved performance, custom hardware may be created. Given the simplistic nature of this algorithm, building such hardware should be readily achievable.

7.4 Hierarchical Topologies

Very few examples have demonstrated utilizing HTM in a hierarchical fashion. Zhang et al. [19] mentioned the use of a hierarchy; however, the specifics of the design are vague. Preliminary work was conducted while creating this work to explore the potential of a hierarchical design. Unfortunately, said work has not yet been finished; however, some preliminary statements may still be made.

A hierarchical design may prove useful for forming higher-level representations of the input. Hierarchical designs in traditional neural networks (aka deep learning) have proven to be immensely successful. In the context of the SP, it is believed that

[1]This is a fabricated encoding scheme for demonstration only. It is entirely possible that preprocessing may be done to the image before it is encoded and / or a different encoding scheme may be utilized.

some changes to the initialization of the network should first occur.

The original design allows synapses to randomly form anywhere within the input space. This is fine for using a single SP; however, when used hierarchically, this type of initialization losses all of the topology originally present within the input. One possible work around is to construct many SP regions within the region and to have those regions connecting to a specific window (i.e. each SP region will have a unique receptive field). While that technique would work, it would be better to redefine the initialization method.

One such method would be to restrict columns to forming connections within a given window. If the latter technique is used, each column would be effectively acting as a mini region. If the number of bits that the column connects with is small enough, it is believed that this technique should perform well. Additionally, it is presumed that each column should be fully connected to all bits within its receptive field. If the receptive fields are constructed to be overlapping, this topology will resemble that of a convolutional neural network (CNN), which has proven to perform exceptionally well in computer vision tasks.

With the aforementioned initialization, it is important to take into consideration how that will affect the utilization of local inhibition. When local inhibition is used, neighboring columns are competing with one another. With the original, random, global initialization method, the original input topology is lost, such that localized competition becomes somewhat meaningless[2]. If however, the input topology is preserved, as with the aforementioned design change, local inhibition now has an immense amount of meaning. Instead of looking at the top columns (global inhibition), each column, within a neighborhood, will have the ability to inhibit other columns. This adds an additional level of competitiveness to the system and will likely improve the SP's results.

[2]This is a likely explanation for why global and local inhibition performed similarly, in this work.

Assuming this new initialization scheme[3], constructing a feed-forward hierarchy is relatively straight forward. Subsequent levels would use the output of the previous level as input. If feedback is desired, a new design will need to be created.

While it may not be difficult to physically construct a hierarchical design, a lot of unknowns exist, regarding how it should be trained and utilized. Using an online training technique will require the most amount of resources, as the full network will be required to be present in memory. If each level is trained sequentially, it is possible to simulate a multi-level design with only ever using a single SP. That method; however, has the downside of not being able to accept new input, since the learning would have incrementally built off the previous level's representation.

Adding levels will also greatly increase the network computation time, so if the hierarchy is not required, it is best to avoid it. Additional concerns regarding parameter selection and optimization are also created. If online learning is used, finding proper parameters will likely be arduous. Offline learning may allow the ability to reuse the ideas in Section 7.2, reducing the complexity.

In theory, a hierarchy should allow for more complex representations to be learned. Additional research is still required to determine the full potential of this idea. Additionally, how this will affect the TM is something that must be understood. One possible construction (classical) is to sandwich an SP with a TM and to stack those levels on top of one another. This would allow for learning an intermediate temporal relationship; however, it is likely that only the top-level sequence is desired to be learned. In such a case, it would make most sense to stack SP levels, with a single TM region on top of that stack.

[3]NuPIC has the option to use localized regions, which is similar to this idea. Additionally, openHTM [20] was known to utilize a localized connection scheme years before this work or NuPIC began to utilize it.

7.5 Future Work

Exploring the use of the proposed initialization scheme as well hierarchical designs should be explored. Given that HTM was designed to be hierarchical, it is important to demonstrate that capability.

The claims made in Section 4.7.1 regarding the selection of the permanence increment and decrement amounts need to be experimentally validated. They should be studied under two contexts. In the first context the parameters should be adaptive, i.e. set based on the current input pattern. In the second context they should be static, i.e. based on the general distribution.

The work done with novelty detection has the potential to be expanded. It is theorized that HTM works best with a single signal. If that theory is correct, it would be best to construct an SP for each input class. This could be tested on the MNIST dataset. A voting method could be used for classification. Alternatively, the output from the overlap metric may be passed onto a classifier. A third option would be to pass the raw output SDRs to a classifier. In this option, each SP would only train on a single class. New inputs would need to be passed to each SP. Those outputs would need to be passed to the classifier. Assuming the classifier's decision function produces a probability, it would be possible to use that probability to determine class membership.

This work should be extended for the TM. Taking a similar approach could provide many insights into the TM. Once that work has been completed, a full system-level model for describing HTM can be created. HTM should be explored in many spatiotemporal applications. One particularly interesting application would be anomaly detection in video streams.

It is very likely that expanding this work will produce experiments of a larger scope. If that occurs, the implementation should be parallelized. If a hardware

version of HTM is created, it could be used to rapidly increase the scope of HTM-related experiments.

Bibliography

[1] Y. Cui, C. Surpur, S. Ahmad, and J. Hawkins, "Continuous online sequence learning with an unsupervised neural network model," *arXiv preprint arXiv:1512.05463*, 2015.

[2] "Hierarchical temporal memory including htm cortical learning algorithms," Available at http://numenta.com/assets/pdf/whitepapers/hierarchical-temporal-memory-cortical-learning-algorithm-0.2.1-en.pdf, 2011, accessed on 2014-11-02.

[3] D. O. Hebb, *The organization of behavior: A neuropsychological approach*. John Wiley & Sons, 1949.

[4] J. Hawkins and S. Ahmad, "Why neurons have thousands of synapses, a theory of sequence memory in neocortex," *Frontiers in Neural Circuits*, vol. 10, no. 23, Mar. 2016.

[5] S. Lattner, "Hierarchical temporal memory-investigations, ideas, and experiments," Master's thesis, Johannes Kepler Universität, 2014.

[6] F. Byrne, "Encoding reality: Prediction-assisted cortical learning algorithm in hierarchical temporal memory," *arXiv preprint arXiv:1509.08255*, 2015.

[7] M. Leake, L. Xia, K. Rocki, and W. Imaino, "A probabilistic view of the spatial pooler in hierarchical temporal memory," *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 9, no. 5, pp. 1111–1118, 2015.

[8] J. Hawkins and S. Blakeslee, *On intelligence*. Macmillan, 2007.

[9] J. Hawkins and D. George, "Directed behavior using a hierarchical temporal memory based system," Available at http://appft1.uspto.gov/netacgi/nph-Parser?Sect1=PTO1&Sect2=HITOFF&d=PG01&p=1&u=/netahtml/PTO/srchnum.html&r=1&f=G&l=50&s1=20070192268.PGNR.

[10] D. George, "How the brain might work: A hierarchical and temporal model for learning and recognition," Ph.D. dissertation, Stanford University, 2008.

[11] "Numenta platform for intelligent computing (NuPIC)," Available at https://github.com/numenta/nupic, commit bd8e6a9.

[12] D. DeSieno, "Adding a conscience to competitive learning," in *Neural Networks, 1988., IEEE International Conference on*. IEEE, 1988, pp. 117–124.

[13] D. E. Rumelhart and D. Zipser, "Feature discovery by competitive learning," *Cognitive science*, vol. 9, no. 1, pp. 75–112, 1985.

[14] R. Bryll, R. Gutierrez-Osuna, and F. Quek, "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets," *Pattern recognition*, vol. 36, no. 6, pp. 1291–1302, 2003.

[15] J. Mnatzaganian, E. Fokoué, and D. Kudithipudi, "A mathematical formalization of hierarchical temporal memory's spatial pooler," *arXiv preprint arXiv:1601.06116*, 2016.

[16] "scikit-learn," Available at http://scikit-learn.org/stable/index.html, accessed on 2016-29-01.

[17] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

[19] X. Zhang, J. Zhang, A. B. Rad, X. Mai, and Y. Jin, "A novel mapping strategy based on neocortex model: Pre-liminary results by hierarchical temporal memory," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 476–481.

[20] "openhtm," Available at https://sourceforge.net/projects/openhtm/, accessed on 2016-03-26.

Glossary

active state

A cell state indicating that the cell became active due to feedforward input from its proximal segment.

bin

A contiguous set of bits within an encoded SDR that represent a particular subset of inputs.

boost

The process of increasing the activation of columns, such that weaker columns will eventually become active.

cell

A fundamental unit within an HTM, analogous to a neuron in an ANN.

column

A collection of one or more cells within a region.

cortical learning algorithm

Terminology used in the previous version of HTM, denoting an algorithm that enables learning in an HTM, namely the SP and the TP.

distal

An adjective used to describe a synapse or a segment. This refers to a connection from lateral input.

encoder

A unit used to convert an input to an SDR. An encoder is used before sending the data to the HTM.

feedback

The flow of data from a higher hierarchical level to a lower hierarchical level.

feedforward

The flow of data from a lower hierarchical level to a higher hierarchical level.

inactive state

A cell state indicating that the cell is inactive.

inhibition radius

The set of columns that may be disabled after the active columns have been determined. This "radius" is spatially centered around each column.

level

A reference to the structure of a single rank within the hierarchy of an HTM.

permanence

A scalar indicating the strength of a synapse connection.

potential synapse

A synapse that could become part of a specific segment, i.e. a synapse that is not currently connected, but could become connected in the future.

predictive state

A cell state indicating that the cell became active due to lateral input from one or more distal segments.

proximal

An adjective used to describe a synapse or a segment. This refers to a connection from feedforward input.

receptive field

The set of inputs that may be connected to a column. This is determined by the distance from a column to all of its connected synapses.

region

The highest-level functional unit within a level. If the level only contains one region, the region would be used to describe the functionality of the level.

segment

A dendrite segment, in the context of HTM, where a segment houses a collection of synapses.

sparse distributed representation

A type of binary encoding, where a small percentage of bits are active at a time. Each unique configuration of active bits represents a specific input.

spatial pooler

One of the primary algorithms in HTM. It is used to form an SDR of the input.

synapse

A connection between cells and / or columns.

temporal memory

One of the primary algorithms in HTM. It is used to form representations of sequences of input patterns.

temporal pooler

One of the CLAs. It is used to form representations of sequences of input patterns.