# REPORT DOCUMENTATION PAGE

Form Approved OMB NO. 0704-0188



| 1. REPORT DATE (DD-MM-YYYY) | 2. REPORT TYPE | 3. DATES COVERED (From - To) |
|---|---|---|
| 30-08-2013 | Final Report | 2-Sep-2009 - 1-Aug-2013 |

4. TITLE AND SUBTITLE

Brain Behavior Evolution during Learning:
Emergence of
Hierarchical Temporal Memory

5a. CONTRACT NUMBER
W911NF-09-1-0481

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER
611102

6. AUTHORS

Donald A. Drew

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES AND ADDRESSES

Rensselaer Polytechnic Institute
Office of Sponsored Research
110 8th Street
Troy, NY 12180 -3522

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSOR/MONITOR'S ACRONYM(S)
ARO

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
56793-MA.1

12. DISTRIBUTION AVAILIBILITY STATEMENT

Approved for Public Release; Distribution Unlimited

13. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not contrued as an official Department of the Army position, policy or decision, unless so designated by other documentation.

14. ABSTRACT

In this report, we summarize two attempts to ascertain whether structure arises in Hopfield brain models subject to Hebbian learning. The first attempt uses graph measures to derive numerical scores for networks, and then apply data mining methods to separate the networks in parameter space. The second attempt uses one- and two-event chains to relate synapse strength to connection information. Learning methods are applied to brains of different types to see if changes can be detected. No Hebbian learning method caused tree-like structure to develop.

15. SUBJECT TERMS

brain model, Hopfield network, Hebbian learning

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 15. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | Donald Drew |
| UU | UU | UU | UU | | 19b. TELEPHONE NUMBER 518-276-6903 |

**Report Title**

Brain Behavior Evolution during Learning:
Emergence of
Hierarchical Temporal Memory

**ABSTRACT**

In this report, we summarize two attempts to ascertain whether structure arises in Hopfield brain models subject to Hebbian learning. The first attempt uses graph measures to derive numerical scores for networks, and then apply data mining methods to separate the networks in parameter space. The second attempt uses one- and two-event chains to relate synapse strength to connection information. Learning methods are applied to brains of different types to see if changes can be detected. No Hebbian learning method caused tree-like structure to develop.

---

**Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:**

**(a) Papers published in peer-reviewed journals (N/A for none)**

Received Paper

**TOTAL:**

**Number of Papers published in peer-reviewed journals:**

---

**(b) Papers published in non-peer-reviewed journals (N/A for none)**

Received Paper

**TOTAL:**

**Number of Papers published in non peer-reviewed journals:**

---

**(c) Presentations**

**Number of Presentations:** 0.00

---

**Non Peer-Reviewed Conference Proceeding publications (other than abstracts):**

Received Paper

**TOTAL:**

**Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):**

---

**Peer-Reviewed Conference Proceeding publications (other than abstracts):**

Received Paper

**TOTAL:**

**Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):**

---

**(d) Manuscripts**

Received Paper

**TOTAL:**

**Number of Manuscripts:**

---

**Books**

Received Paper

**TOTAL:**

**Patents Submitted**

---

## Patents Awarded

## Awards

### Graduate Students

| NAME | PERCENT_SUPPORTED | Discipline |
|---|---|---|
| Oswaldo Sanchez | 1.00 | |
| Andrew Warner | 1.00 | |
| **FTE Equivalent:** | **2.00** | |
| **Total Number:** | **2** | |

### Names of Post Doctorates

| NAME | PERCENT_SUPPORTED |
|---|---|
| **FTE Equivalent:** | |
| **Total Number:** | |

### Names of Faculty Supported

| NAME | PERCENT_SUPPORTED | National Academy Member |
|---|---|---|
| Donald A Drew | 0.05 | |
| **FTE Equivalent:** | **0.05** | |
| **Total Number:** | **1** | |

### Names of Under Graduate students supported

| NAME | PERCENT_SUPPORTED | Discipline |
|---|---|---|
| Julienne LaChace | 0.00 | Mathematics |
| Francis Cheng | 0.00 | Mathematics |
| **FTE Equivalent:** | **0.00** | |
| **Total Number:** | **2** | |

### Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

| | |
|---|---|
| The number of undergraduates funded by this agreement who graduated during this period: ..... | 2.00 |
| The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields: ..... | 2.00 |
| The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields: ..... | 1.00 |
| Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale): ..... | 1.00 |
| Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering: ..... | 0.00 |
| The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense ..... | 0.00 |
| The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields: ..... | 0.00 |

## Names of Personnel receiving masters degrees

| NAME |
| --- |
| **Total Number:** |

## Names of personnel receiving PHDs

| NAME |
| --- |
| Oswaldo Sanchez |
| Andrew Warner |
| **Total Number:** 2 |

## Names of other research staff

| NAME | PERCENT_SUPPORTED |
| --- | --- |
| **FTE Equivalent:** | |
| **Total Number:** | |

## Sub Contractors (DD882)

## Inventions (DD882)

## Scientific Progress

Brain connections were characterized by a set of weights connecting neurons. A number of graph measures were identified to characterize the networks. Data mining methods were used to determine how different graphs types could be distinguished. In addition, two-event chain firing frequency data were generated as a function of connection strength. Hebbian learning methods were applied to the different types of brains, but no evidence was obtained that indicated that structure emerged from learning. Indeed, Hebbian learning (and all of its variations) led to networks that are symmetric (and therefore not brain-like) and were interconnected with a Gaussian distribution of weights.

## Technology Transfer

Final Report for ARO

# Brain Behavior Evolution during Learning: Emergence of Hierarchical Temporal Memory

Donald A. Drew, PI
Department of Mathematical Sciences
Rensselaer Polytechnic Institute,
Troy, NY 12180

## Contents

**Abstract**

In this report, we summarize two attempts to ascertain whether structure arises in Hopfield brain models subject to Hebbian learning. The first attempt uses graph measures to derive numerical scores for networks, and then apply data mining methods to separate the networks in parameter space. The second attempt uses one- and two-event chains to relate synapse strength to connection information. Learning methods are applied to brains of different types to see if changes can be detected. No Hebbian learning method caused tree-like structure to develop.

# 1 Introduction

In this report, we summarize two attempts to ascertain whether structure arises in Hopfield brain models subject to learning. Brains function through a process whereby neurons "fire" in response to a stimulus, that firing then induces other neurons to "fire," resulting in a cascade of information. The growth and formation of the brain structures is governed by genetic information and implemented in each organism (including humans) in an environment of proteins and enzymes. However, the equivalent of a schematic or wiring diagram at the level of individual neurons does not exist, and indeed, it is believed that brains are continually rewiring in the process of learning.

Implementation of the genetic code specifies the types and number of neurons as well as the general patterns of connections, but leaves the details of neuronal wiring up to the adaptive processes of development. It is generally presumed that the acquisition of complex skills, such as speech, vision and movement, achieved with a little supervision during the first years of life, is due to adaptive processes of neuronal re-organization and synapse strengthening and reconnection operating within and upon the existing processing structures[2].

To say the least, the brain is an extremely complex system, and after a century of work we have a basic understanding of the complex processes within a single neuron and the biophysical chemistry of synapses [1, 24, 48, 49]. However, the dynamics of how a functioning network of neurons lead to memory gain and loss, behavior, consciousness, creativity or any of the byproducts of the brain, are still poorly understood. To wit, the brain has about 40 billion neurons each having at least 1,000 but as many as 10,000 synapses, for a total of about 30 trillion synapses. Neurons are the key component and computational unit of the brain, organized into the several regions of the brain. The cerebral cortex [4] is the structure in the brain that plays a key role in memory, attention, perceptual awareness, thought, language and consciousness. It is pretty well accepted that the cortex is relatively uniform in appearance and structure [32]. That is, the area for seeing, the occipital lobe, does not have a drastically different structure from the region for hearing, the temporal lobe. What differs is which organ each region connects to; therefore, there should be a common algorithm for all signals (inputs), independent of any particular function or sense, and this algorithm should be as simple as possible. This is because despite its complexity, the brain has to function and react at time scales less than a second. These limitations on the speed of the chemical and electrical reactions in the brain suggest that its storage and retrieval must be as simple and efficient as possible. Desirable models for the brain should have great speed and possibly parallel structure, be robust, have plasticity (the ability to change or adapt), deal with complexity, but most importantly, be as simple as possible.

The brain does not compute the answers to problems; instead, it retrieves the answers from memory stored in the neurons. For example the brain does not compute 2+2, but remembers that it is 4. This is very similar to how mathematical models called neural networks work. These networks can learn patterns, store them and then retrieve them. The use

of models that borrow some of the procedures from biology, but are not bogged down by the actual size of the brain, or all the intricate chemical and physical properties, may shed some light on how we learn. They may one day help answer the question that drives computational neuroscientists: How can an incomplete description (inputs to our brain), encoded within neural states, be sufficient to direct the survival and successful adaptive behavior of a living system [4]?

Many of the computational abilities of the brain, such as learning and memory retention, are a result of the modification of the efficacy of the synapse response [20]. Learning is the acquisition of new knowledge and skills while memory is the storing, retention, and retrieval of what has been learned. During the learning process, the weights connecting the neurons are altered; some are strengthened while some are weakened. The connections between the neurons give the network its structure. As the weights change, the structure of the network changes.

If we understand what algorithm the brain uses to learn, we would be able infer things about artificial intelligence, the process of machine learning. Algorithms that can reproduce the computational properties of the brain can help us understand how these properties may arise. We would be able to make models that predict human behavior such as addictions and reactions to certain inputs, how people learn, or why they learn certain things quickly and easily, but other learning comes with difficulty, if at all.

Hawkins and Blakeskee [18] hypothesize that the brain has evolved many hierarchical structures in order to manage the complexity of accomplishing both learning and memory formation. The main goal of this project is to examine different learning schemes, such as Hebbian learning, and investigate how they result in a restructuring of the connections between neurons. We postulate that a brain structure, established early in the life of the organism, is modified by the way the connections evolve during learning, causing this hierarchical arrangement to emerge. It is plausible, then, that the brain arranges the connections between neurons hierarchically as a result of the way it learns.

First we will further define the concept of hierarchy. Then we will look at how neuronal states can be modeled by neural networks and how those network change as they are trained on data. We also define how we will use graph theory to represent the networks and graph theory measures to distinguish the networks. We then discuss how we implement various learning algorithms in the networks, and how changes in hierarchy are tracked using data mining techniques. We present the issue of using of a threshold to convert networks to graphs, and how it affects our results.

Complex systems with complex behaviors, like the brain with all its processes, can emerge from the simple interactions of simpler components (neurons) [22]. Through the interplay of the relatively simple and homogeneous neurons, the brain is able to store and retrieve stable memories, make categorical generalizations and adapt in novel situations. Emergence of these computational capabilities from the collective behavior of a large number of simple precessing elements has been modeled by Hopfield, Carpenter and Grossberg [22, 21, 23, 14].

Hopfield's brain models used neurons with elementary properties and

networks with little structure, nonetheless, computation properties emerged, allowing the networks to correctly recall memories and resolve ambiguities with the capacity for some generalization and time ordering of memories [22]. Carpenter and Grossberg used network dynamics to learn recognition categories and observed in their network an architecture that self-organized and self-stabilized, illustrating that "the whole is much greater than the sum of its parts both in human experience and in self-organizing model" [14]. They found the emergent properties of parallel network interactions allowed for novel inputs to recall a category by learning invariant properties of the familiar exemplars of that category.

## 1.1 Hierarchy

According to Hawkins, in a hierarchical system, some elements are in an abstract sense "above" and "below" others [18]. There is a concept of levels and elements are ranked or sorted into levels where information propagates up or down these levels sequentially. What makes one region "higher" or "lower" than another is how they connect to one another. The region with the initial input is considered the lowest level. Data then propagates up the hierarchy to other regions. There is a flow of information from the lower neuron to the higher ones. There can also be lateral connections between neurons on the same level. For example, in the brain the lowest region in the hierarchy could be the primary sensory area, called V1, where information from the senses first arrives in the cortex. V1 then feeds information to other regions such as V2 and V4, and those regions connect to the top region, IT[18].

Hierarchy is found not only in the brain, but virtually all complex systems including social networks, power grids, and the Internet. The universal advantage of such hierarchical forms is that they are efficient and robust against disruptions that might threaten the goal of the system. That so many systems naturally evolve into a hierarchical structure seems to be more than a coincidence, indicating that for many complex interactions, a hierarchical system may be optimal [7, 35].

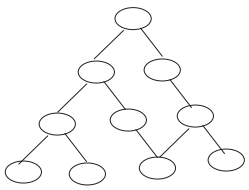


Figure 1.1: Hierarchy

Figure 1.1 shows an example of a hierarchical structure. In this type of structure, nodes of a higher level have access to more neurons than those from lower levels. There is a convergence of information in the higher levels. In the brain the lower neurons might receive visual information from seeing a dog over the observation period. Each neuron would receive

vastly changing signals from individual receptors on the eye. Neurons higher in the hierarchy would be fed this data and recognize they are looking at tails, fur, paws, while the neurons above those would recognize back, head, leg and finally the top one would recognize dog. This hierarchy allows for something as complex a recognizing a dog to be possible. How we model hierarchy is explained further in section 4.1.

## 1.2 McCulloch and Pitts Model and Hopfield Networks

There are several different types of neurons arranged in the spatial network in a brain. These neurons consist of a cell body, several dendrites, and an axon, and are classified, at least partly, by geometric structure and complexity. Communication between neurons occurring at synapses is due to two types of activity: electrical, most notably due to the action potential, a wave of potential change that propagates along the dendrites and axons; and chemically, with release of neurotransmitters and ions at synapses. These processes are complex, and it is possible that nuances in them may modulate brain activity and memory. For the purposes of studying the behavior of networks, it is impractical, if not impossible, to describe these details, and we do not attempt to do so. We model the brain by a moderately large set of neurons with discrete states ("on" and "off", or "firing" and "resting"). We assume that any neuron contributes to the activation of any other by superposition, subject to a threshold. Learning is the modification of the interaction weights and the synaptic thresholds. An event for this brain is a sequence of neural states for discrete times, starting from a given state, and stimulated by a sequence of input states to a small fraction of the neurons. During an event, the weights will change. Learning has the effect of making the neural states resulting from the same starting state and the same stimulus to be different for a brain that has undergone some learning process and one which has not.

Donald O. Hebb formulated a theory stating that synaptic plasticity results from the simultaneous (or immediately successive) activation of presynaptic and postsynaptic neurons [19]. That is, repeated and persistent stimulation of both neurons increases the strength of their synaptic connection. As a result, if one is activated, the other is likely to activate as well. This has come to be described by the rubric "Neurons that fire together wire together" [19]. Synaptic strengths are also weakened as a result of their disuse, that is, "If you dont use it you lose it."

A simple iconic example of Hebbian learning can be seen in the conditioning of Pavlovs dog (Figure 1.2). Suppose the dog has three neurons, A, B and C, for which the sight of food is enough to excite neuron C, which in turn excites neuron B and causes salivation. Also suppose that in the absence of food, sound from a ringing bell excites neuron A but does not excite neuron B. Then applying simultaneous sight and sound stimulation causes salivation and may increase a putative connection between neuron A and B. After repeated simultaneous stimulation the sound of the bell may induce salivation without the sight of any food. The dog is now conditioned (hardwired) so that the response to the bell and the

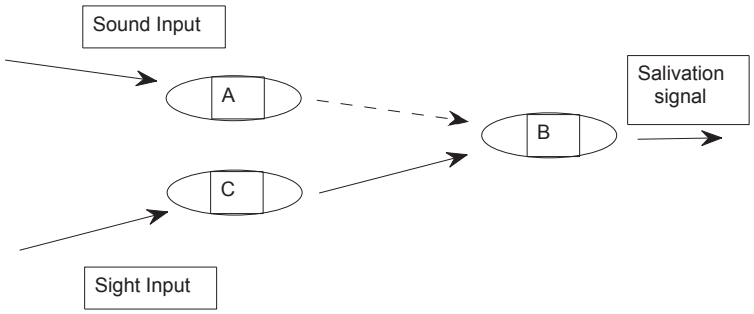



Figure 1.2: Three neurons learning

response to the food are closely related [16, 34]. Note that the essence of this conditioning response is that "neurons" A and B fire together, and became "connected" in terms of their responses.

McCulloch and Pitts proposed a simple model of the interactions of neurons as a response to the stimulus of connecting neurons and whether the stimulus exceeds a threshold, $\tau_j$ . It is a discrete time neuron network model that assigns to neuron $j$ the states of "on" ($s_j(t) = 1$) or "off" ($s_j(t) = 0$) depending on the state at the previous time $t - 1$ and on inputs to certain "sensory" neurons. That is, the state of neuron $j$, $s_j$ , at time $t$ is a function of the states of all the neurons at that connect to it at time $t - 1$. We model the behavior of the network by

$$s_j(t) = H\left(\sum_i w_{i,j}(t-1)s_i(t-1) - \tau_j\right), \tag{1.1}$$

where $H$ is the Heaviside function,

$$H(x) = \begin{cases} 1 & x \geq 0, \\ 0 & x < 0 \end{cases} \tag{1.2}$$

and the weight $w_{i,j}(t)$ represents the strengths of the synapse connecting neuron $i$ to neuron $j$ at time $t$, and $\tau_j$ is the threshold for neuron $j$.

In this model, memory is associated with the weight matrix $w_{i.j}(t)$ and learning is a process by which $w_{i,j}(t)$ changes dependent on previous $w_{i,j}(t-1)$ values and previous neuronal states.

$$w_{i,j}(t) = f(w_{i,j}(t-1), s_i(t-1), s_j(t-1)). \tag{1.3}$$

The goal of most neural networks is to find the optimal weights so as to learn patterns, classify or cluster data, make predictions, and correct for

missing or corrupt data [39, 42]. The Hopfield network is based on the idea that memory is associative. We associate names with faces or countries with locations [17]. Hopfield networks change by Hebbian learning; that is, if neurons are on at the same time the connection (weight) is strengthened. One starts out with a fixed given set of p patterns, $(\xi^1, \xi^2, \ldots \xi^p)$ input to our neurons. The Hopfield Network assumes the patterns are the minimum of the energy function defined by

$$E(x) = \frac{-1}{2} x^T \overline{W} x \tag{1.4}$$

where $\overline{W}$ is the connection matrix defined by

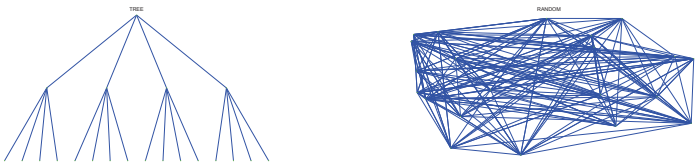
$$\overline{W} = \frac{1}{n} \sum_{\mu=1}^{p} \xi^\mu (\xi^\mu)^T \tag{1.5}$$

Here $\xi^\mu$ is a vector of zeros and ones, and the domain of $E$ are the possible patterns we can have [4, 20]. Basically the weight matrix that has "learned" the patterns $(\xi^1, \xi^2, \ldots \xi^p)$ is the linear combinations of the outer products of these pattern vectors. It is called the covariance matrix, and is symmetric. We explore learning algorithms in section 5.

## 1.3 Graph Theory

In order to determine whether learning results in a change of the hierarchical structure we must quantify hierarchy. Figure 1.3 shows an example of a tree and a random network. A tree is clearly hierarchical while a random network is not.

We use graph theory to represent brain models. A graph, $G =< V, E >$, is a set of vertices, $V$, some pairs of which are connected by links called edges, $E$. A graph can be represented by an adjacency matrix, $A$, where $A(i, j) = 1$ represents a connection fromi to $j$ and $A(i, j) = 0$ otherwise. If the connection between nodes $i$ and $j$ is bidirectional, then $A(i, j) = A(j, i)$. A weighted graph allows for entries other than just 0 or 1, representing strengths of connection, lengths, capacities or any relationship we can quantify between $i$ and $j$ [13].




(a) Tree Graph (b) Random Graph

Figure 1.3: A Tree Graph and a Random Graph

The weight matrix $W(t)$, is an matrix that specifies how strongly neurons $i$ and $j$ are connected. If we round the values in a weight matrix to either 0 or 1, with the assumption that neurons with sufficiently low weights between them do not affect each other, we can construct a thresholded graph of the network. That is, we round low values to 0 and higher values to 1. Therefore, though there may be a connection between two neurons, if it is too weak we do not count it. In order to use graph theory tools we need to classify two nodes as connected or not; as a result, we must map the weight matrix in a matrix of zeros and ones. To do this we must round the matrices. From a weight matrix, $W$, define the adjacency matrix $A^\tau$ at the threshold $\tau$ by

$$A^\tau_{i,j} = H\left(|w_{i,j}| - \tau\right), \tag{1.6}$$

where $H$ is the Heaviside function. The values used for the threshold affect the results. We investigate this by varying the threshold.

We use graph theory measures such as expansion [38, 37], eccentricity, efficiency, [36], clustering coefficient and average minimum path length [45] to rank graphs in terms of increasing hierarchy. We propose to determine a region in a low dimensional space of such parameters to quantify hierarchical graphs in an empirical way. Each dimension is a measure of some aspect of the graph. We shall generate several known hierarchical and non-hierarchical networks, and calculate the values of the parameters for each.

Once an appropriate measure of hierarchy has been established, we examine the behavior of Hopfield networks under a variety of learning algorithms. During training we study the evolution of the graph in this low dimensional space. We shall evaluate the propensity of learning algorithms to cause networks to migrate toward the hierarchical region in this space. This project yields a measure to study the emergence of hierarchy in brains as well as in other networks and graphs. It is important to be able to distinguish systems that are hierarchical from those that are not because it allows us to better understand how their structures affect their dynamics. We explore graph theory measures in section 4.1.

## 1.4 Data Mining

### 1.4.1 Support Vector Machine Learning

Support Vector Machine learning is a form of supervised learning. We present the SVM learning mechanism with $n$ inputs and their corresponding desired output. That is we present it with a set of $n$ patterns $\{x_i, y_i\}$, where $x_i \in R_n$ and $y_i \in \{+1, -1\}$. In order for SVM to successfully associate input with correct output patterns, the data must belong to either one of two groups and be linearly separable. Examples are a patient either having cancer or not, an email being a scam or not, a computer crashing or not. SVM is similar to regression in that it fits a line (or a hyperplane) to separate the two classes. However, unlike regression, where the line is trying to interpolate the data, SVM just linearly separates an $n$-dimensional space into two parts: one side for one class and the other

side for the other class. Figure 1.4 is an example of classifying with SVM [25].

### 1.4.2 Principal Component Analysis

Principal Component Analysis (PCA) is a method used in regular statistical analysis which extracts a basis to define a lower dimensional vector space. The procedure is to find n eigenvectors corresponding to the optimal basis for a $n$ dimensional representation of the vectors with minimal reconstruction error. The basis of the vector space is changed so as to place the maximal variance of the data along the eigenvector with the greatest eigenvalue, the second largest variance along the eigenvector with the second largest eigenvalue, and so on. Variance is a measure of the spread of the data set. The virtue of using the eigenvalues is that they rank the directions of the space in terms of variations along those dimensions, and this ranking is often related to their importance.

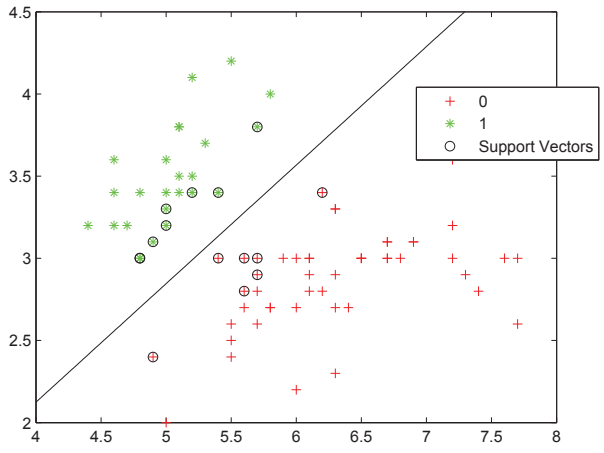
We can use exactly all the decomposed eigenvectors from the covariance matrix and represent all the data in terms of the linear combination of the eigenvectors. They reorient the data so that it can be more easily analyzed for feature extraction. Ideally we want to represent the greatest amount of information with the least amount of memory. With PCA we could use only the first few eigenvectors because they explain the directions in which the information varies the most, and neglect the last few eigenvectors since they explain very little. In other words, the basis projects the data from their original $m$ dimensional space onto a $n$ dimensional subspace, with $n < m$, spanned by these $n$ vectors resulting in a dimensionality reduction that often retains most of the intrinsic information of the data. Actually the choice of how many eigenvectors to keep is problem specific. For example if the vector space data is in $R^{200}$, whether or not the problem reduces to $R^{150}$ or even $R^{20}$ depends on how much variance is explained by the eigenvectors with the larger eigenvalues [4, 12].

### 1.4.3 Canonical Variate Analysis

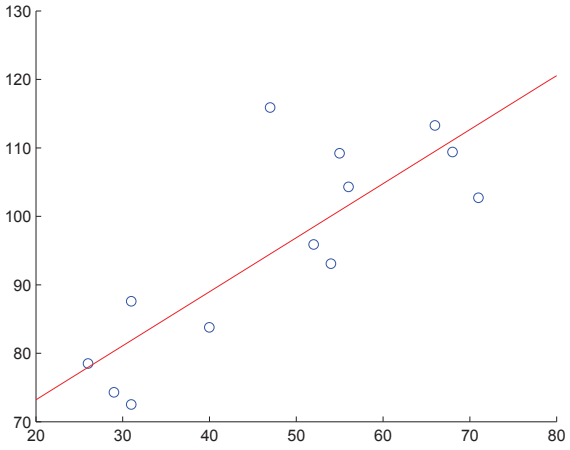
Canonical Variate Analysis (CVA) is a method using a basis to define a lower dimensional vector space that maximizes the distance of data points in different groups and minimizes distances of data points in the same group. In instances when the data is not linearly separable, CVA would a better option than PCA.

### 1.4.4 Parzen Windows

Given a set of data points all belonging to the same group, we can determine the probability that a new point belongs to that group. We assume the data given all belong in one group $T$, and then construct an underlying probability density function, $P$, that could have given rise to the data. This can be done by fitting a Gaussian distribution to the data. We calculate $P(x \,|x \in T\,)$ by determining the sample mean and standard




(a) SVM




(b) Regression

Figure 1.4: Classifying vs. Regression

deviation, and substituting them into the distribution

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{1.7}$$

or in higher dimensions

$$f(x) = \frac{1}{(2\pi)^{\frac{k}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{1.8}$$

where $\Sigma$ is the covariance matrix, $|\Sigma|$ is its determinant, $k$ is the dimension of the space, and $\mu$ is the vector of means.

When analyzing data that could belong to any one of several different sets, the data points in each group determine separate means and standard deviations, and requires that we fit each of them with its own distribution. Then given a new point, we can calculate to which group the point belongs by determining which group assigns the largest probability to the new point. Parzen windows are used to decide to which group the nodes belong. We explore data mining to define and quantify hierarchy in section 4.2.1.

## 1.5 Event Chains

An event chain is an occurrence of certain events in a given order. Event chains of length one are just the events themselves. Event chains of length two are two events that took place in a given order. In the McCulloch-Pitts model implemented as a Hopfield network, a two-event chains would occur when one neuron fires one time step after the other.

It is clear that if a string of neurons fire one after the other with about the same frequency that any one of them fires, that the firing usually occurs in that sequence. From that, we can infer that the neurons are connected from neuron to neuron in that string. The neurons may be connected to other neurons outside the string, but from that firing pattern, the connectedness of the neurons is evident.

To utilize this idea, we construct brains out of certain reasonable construction rules, and interrogate them for a correlation between the two-event chain firing frequency and the connection weight $w_{ij}$. This relates function, namely the firing frequency, to the structure, the connection weight distribution.

## 1.6 Learning Algorithms

Neural Networks are a set of processing nodes (elements, units, links, neurons) with connections between them. Their structure and purpose vary, but typically they are arranged in levels and the connections changed by the data with which they are presented [17].

A learning algorithm is a way to change the weights of the matrix representing connections between neurons. The magnitude of the weights is equal to the strength of the connection. The weights carry the information that has been learned. Changes in the weight of the matrix occur in response to learning. We allow the activity of the neurons to train the

weight matrix. For example, Hebbian learning implies that if two neurons are repeatedly on at the same time, the connection between them should increase. Gradually updating weights is analogous to human learning because synaptic connections are strengthened or weakened while learning occurs.

Developing an algorithm to model human learning can be guided by biological plausibility. An important feature of such an algorithm is its ability to store information in a set of nodes so that the information can be recalled and altered. Neuroscientists now understand that an increase in the rate of neurotransmitter released by the presynaptic cell is responsible for the increase in the strength of the synapse which is interpreted as learning [3]. Hebbian learning mathematically exploits this relationship, assuming that neurons that are frequently on at the same time develop the strongest synapses [17].

## 1.7 Hebbian Learning

Hebbian learning is any learning that strengthens the connection of neurons that fire simultaneously and weakens those that do not. There are several ways to change the weight matrix based on the activity of the neurons.

The Hopfield Network uses a way to choose the weights for the McCulloch-Pitts model so that memories are content-addressable and insensitive to small changes [20, 21, 22, 23]. It chooses the weight matrix to be a linear combination of the patterns $\xi_i^\mu$, $\mu = 1, 2, \ldots, p$

$$w_{i,j} = \frac{1}{n} \sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu, \tag{1.9}$$

where $p$ is the number of patterns and $n$ is the number of neurons. The symmetry of the Hopfield network implies that all neurons have the same effect on other neurons as those neurons have on them. This restricts the interpretation of the model in biological terms because neuronal connections are not bidirectional. Specifically, neurons are connected from synapse to dendrite. Thus, the symmetry of the Hopfield model contradicts the assumption of biological plausibility. We consider variations to Hebbian learning in section 5.

## 1.8 Goal

We are interested in the idea: Hierarchy can emerge in a system as a result of how a network learns; a network that changes weights after it is given an input, extracting patterns from the input. We examine structure in two quite different ways, (i) using graph measures and data mining, and (ii) by analysis of structure related to function.

We train networks with Hebbian-like learning algorithms and, as we do so, trace whether hierarchy changes or not using graph measures, and how the function changes.

We are interested in the measures of a tree and tree-like structures because they are hierarchical; so we use the trees as our prototype for hierarchy. We calculate the probability that a given graph is one of several

different graph types, in particular a tree. We then take the graph and alter its structure during training by rewiring it connections. We calculate the probability of the graph being a tree after each training and if the probability that it is a tree increases, then we say its hierarchy increases. We explore different starting values and different thresholds and find that threshold is very important. One threshold value might lead us to conclude the hierarchy is increasing while another might contradict those results. We also find different training algorithms lead to different results. We present results in Section 5 and conclusions in Section 7.

The functional approach appears to be useful in determining whether it is possible to distinguish brains with "different" structures. We text the method against brains with "damage", more specifically, we compare Waxman networks that are connected locally to similar networks with patches of neurons that do not fire.

# 2 The Brain

We begin by examining the neuroscience of the brain in order to devise a strategy to extract the information relevant to the problem. The brain is an extremely complicated organ comprised of many components, and that to model the entire brain is cumbersome at best, and unfeasible at worst. Modeling assumptions become increasingly drastic simplifications as we try to encompass more realistic complexity of the brain.

## 2.1 The Neocortex

For the purposes of this research, we are concerned with the activity of the neocortex. The neocortex is part of the cerebral cortex, the outermost area of the brain. The cerebral cortex is involved in processing sensory information, delivering motor commands, and is believed to be involved with thought, memory, learning and intelligence. Our focus is on modeling behavior of the neocortex based on the measurable structure and functionality, followed by making inferences about the less tangible concepts such as thought or intelligence. We will start by discussing the neocortex as a whole and delve into relevant specifics.

As an example of the substructure of the brain at this order of functionality, the neocortex is a subregion of the cerebral cortex. Furthermore, we know that the neocortex focuses on sensory information. Each sense is processed in subregions within the neocortex. These are labeled accordingly. For example, the areas of the neocortex that handle vision are the V1, V2, and V4 regions. Each of these regions, for example V1, have further subregions of V1.

As we see in Figure 2.1 these regions are connected and connect to even more regions in a hierarchical structure. For example, information in V1 flows to V2, is processed, then flows to V4 and so on. The lower regions of the hierarchy handle more basic inputs directly from the senses and interpret them to the next level of complexity. As we move up the hierarchy the data passed through by the lower regions is interpreted and

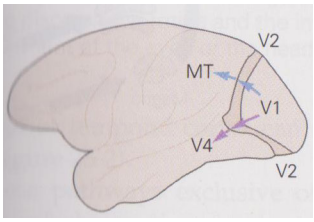



Figure 2.1: Areas of the neocortex that handle visual sensory data, labeled V1, V2, V4 and MT

more complex interpretations are made. The area shown in the figure as MT is the middle temporal area. As we move up in the hierarchy we eventually join the senses together to get a more complete interpretation of what we are seeing, touching, smelling, etc. In conjunction with sight, suppose we consider touch and hearing (or audition). Each sense would have its own hierarchy, but eventually join together as seen in Figure 2.2.

In order to develop an operational understanding we examine the composition of the neocortex itself. The neocortex, in humans, is a layer roughly 2mm thick and comprised of special connections organized in six layers. The six layers are comprised of neurons and vary in both size and in number of neurons in the layers. For example, layer one is a thinner layer and tends to consist of the axons of a neuron more than the cell bodies themselves. Layer two or four however may be thicker and will undoubtably contain the stoma (cell bodies) of neurons in addition to the axons and dendrites.

In Figure 2.2(ii) the stoma is represented as a pyramid or star shape. Regardless of which subregion of the neocortex, there are six layers in every region. Since there is consistent structure and composition of each the subregions, the actions in each regions must be the same. This means that the operations being performed in a region such as V2 are the same being performed in A4, an auditory region. We'll investigate this further as well, but it is important to recognize the implication of this. Recent research suggests that "rewiring" of visual and auditory inputs can occur in the brain of a ferret. This research supports the idea that, although not exact, the behavior in the auditory region A1 is able to mimic the performance of the visual region V1. In their book "On Intelligence", Hawkins and Blakeslee state simply "Cortex is cortex".

### 2.1.1 Cortex Organization

We next examine the composition of a region at the neuron level so we can describe just how the inputs in a region are handled and what occurs to produce their outputs. Using stains on the cortex shows a structure that is consistent with the representation in Figure 2.2.

The use of stains enables the examination of the neurons within a slice of cortex. In Figure 2.3 we can confirm many of the structural issues addressed already. Notice that there is a clear distinction between layers on the left and center images with the Nissl-stain. It is widely accepted

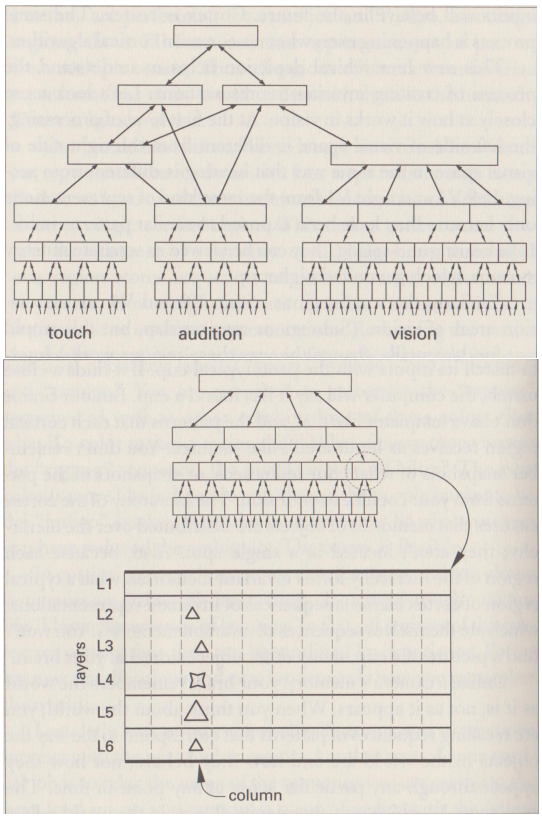



Figure 2.2: (i) Hierarchy for inputs from senses of Touch, Hearing, and Sight (ii) Cortical sheet for a given region

that the cortex physically consists of six layers. Note that layer one at the top of the image contains mostly axons and few neuronal cell bodies. The Golgi-stained cortex shown in Figure 2.3 reveals an important structure for neurons in the layers. It is noticeable that the axons from neurons in lower layers such as layer 4 and 5, tend to extend directly upward towards layer 1. The pillars that are formed by these axons create neuron "columns". The columns in the neocortex are not perfectly straight, but from a simplistic view, the image in Figure 2.2 seems like an accurate enough representation.

The exact definition of a column and the effect of the arrangement of the neocortex into columns is debatable; however, there is an agreement that these columns do exist. In some literature they are referred to as cortical minicolumns. The columns are comprised of around 100 neurons each. A critical point is that the neurons within a given column tend to be active together. Suppose we examine the V1 region which takes the sensory input coming to us through our eyes. The input into the cortical region will be in layer 4 or 5. Beginning at layer 4 or 5, the neurons send

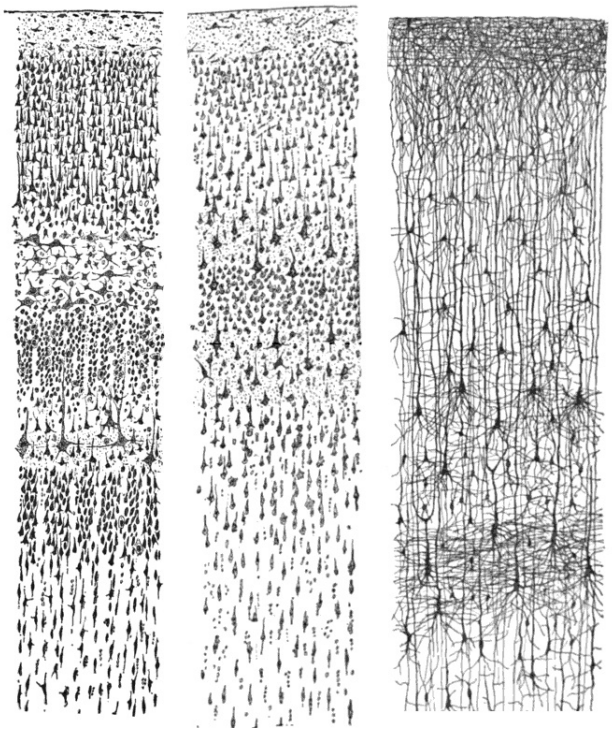


Figure 2.3: (i) Nissl-stain on visual cortex (ii) Nissl-stain on motor cortex (iii) Golgi-stain on cortex

a signal through the layers where the signal eventually leaves layer 1 and travels up the hierarchy to the V2 region, and so on. This is illustrated in Figure 2.4.

A model for behavior of the neocortex requires recognition of these columns as a basis for the model. Referring to Figure 2.4, the neuron represented by the star in layer 4 will fire in response to the input signal it receives from other regions of the brain or perhaps a signal generated from one of the senses. The firing of this neuron will more than likely cause the neurons in the above layers in the same column, to fire. We could model this sequence of firings at the neuron level and consider the propagation of information in the column as separate events, but due to the likelihood that the column will fire when the star shaped neuron fires, it may be more detail than is necessary. Instead of looking at whether a large sequence of individual neurons in an organized sructure (specifically, a column) is firing or not, we consider whether the column as a whole will "fire" or not. We offer the interpretation of all methods and results in this report as firing of neuronal columns instead of individual neurons. However, we shall call the units that interconnect and fire "neurons" throughout this report.

The neurons in any given column take input from many other sur-

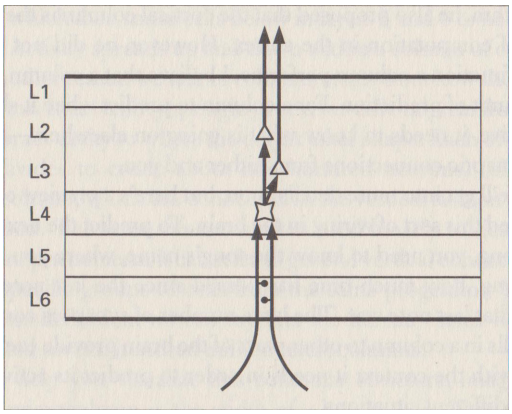



Figure 2.4: Signal traveling through the cortical layers (Hawkins and Blakeslee)

rounding columns. Figure 2.5 shows three distinct columns differentiated by their shade of brown. We can see from this representation how these three columns interact with each other. These interactions are an important since the signal coming into the neocortex travels through this hierarchy by means of these interactions. For example, it has been shown that the V1 region of the neocortex takes the input from our eyes and is able to determine simple attributes such as the orientation of linear structures.

With conditioning and learning, the brain becomes more adept at dealing with recognition of stimuli that are similar to previously learned stimuli. It is clear that the interactions between columns are extremely important to information processing. This will determine what we see, hear, taste, etc. We also know that the columns are not indivisible entities, so we must account for the relation between their components. If we want to know how columns interact, we must see how their parts interact, and for that we need to understand how the synapses between neurons behave.

### 2.1.2 Neuron Type and Interaction

This brain structure, namely, columns which fire in response to stimuli from sensory information, or from interactions with other columns, needs to be modulated with a learning process. There are several types of neurons which interact with all systems of the body, from regulating temperature to causing muscle contraction. Not all neurons are the same, in fact, not even all neurons in the same area of the brain such as the neocortex, behave alike. There are more than 40 types of neurons. Some of the most important are shown in Figure 2.6. Since we are only concerned with the neurons within the neocortex, it is clear that only the Projection and Local interneurons are of interest. The Projection neurons constitute roughly 80% of all neurons within the cortex and have a pyramidal shape. These neurons are primarily in layers 3, 5 and 6. The remaining roughly

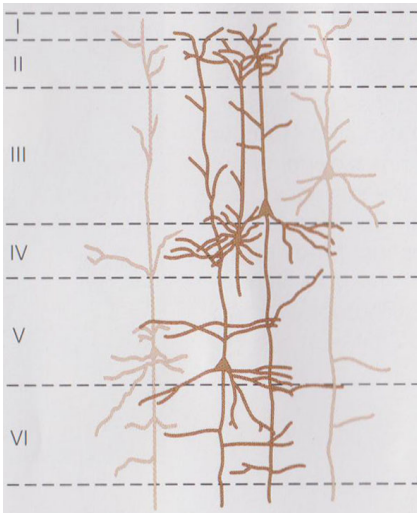



Figure 2.5: Representation of multiple columns through cortical layers

20% of neurons are the Local Interneurons.

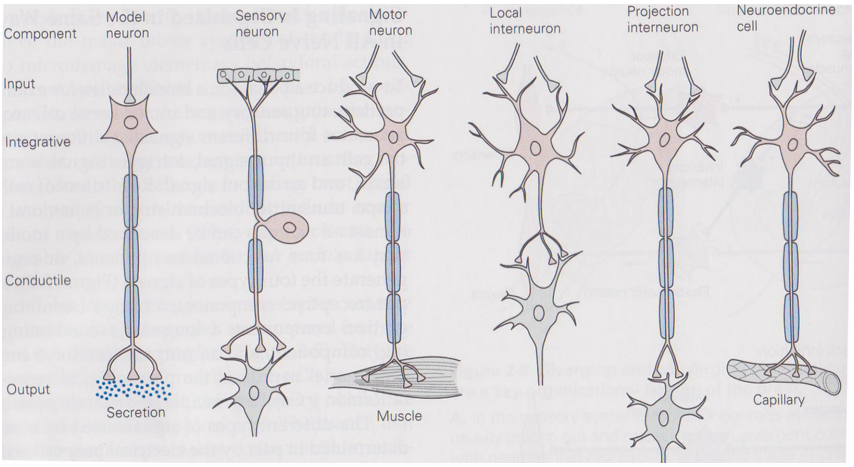



Figure 2.6: Neuron models based on function

Local Interneurons are often referred to as stellate neurons due to their starlike shape, motivating the use of stars to represent them, in Figure 2.2 and Figure 2.4.

If two neurons (Neuron A and Neuron B) have an excitatory connection from Neuron A to Neuron B, when Neuron A fires, it attempts to make Neuron B fire. With an inhibitory connection, when Neuron A fires, it attempts to suppress Neuron B from firing.

Whether a neuron will fire or not then becomes a matter of integrating

the effects of the incoming excitatory and inhibitory pulses. Each neuron has an activation state that is modified by the incoming pulses. Each neuron fires when its activation state surpasses a certain point. We call this the threshold of that neuron. It is also worth noting that after a neuron fires there is a period of time where the neuron is less likely to fire. This refractory period will not be implemented in the present model.

Given this information we can develop a model of neuronal column interaction in terms of the integration of individual neurons interacting within their respective columns.

# 3 Brain Model

In this section, we derive the model that we use to describe the brain. The model must act in a manner that simulates, to a reasonable degree of accuracy, the complex behavior of neural firing and interaction within the brain.

## 3.1 McCulloch and Pitts Model for Neurons

We describe neural interaction by the model proposed by McCulloch and Pitts[57]. This widely accepted simplification reduces a neuron to a state of being either on (represented by the number '1') or off (represented by the number '0'). In addition, we assume that neural events occur at discrete times. The state of neuron $i$ at time $t$, $s_i(t)$ changes at discrete times, $(t = 0, 1, 2, ...)$. This is discussed further in Section 3.1

Next, the interaction between the neurons is measured in terms of a "weight". This weight quantifies both the strength of the interaction between two neurons, and whether that connection is inhibitory or excitatory. We will define the weight from neuron $i$ to neuron $j$ as $w_{i,j}$. Since it is not necessary for this interaction to be symmetric, we make a distinction between $w_{i,j}$ and $w_{j,i}$.

When neuron $i$ is on, neuron $j$ is receiving an impulse from neuron $i$ of weight $w_{i,j}$. If the total impulse from all active neurons connected to neuron $j$ is large enough, it will cause neuron $j$ to fire (turn on) at the next instant. We can measure the incoming impulse to neuron $j$ by the sum

$$\sum_i w_{i,j} s_i(t)$$

Here

$$s_i(t) = \begin{cases} 1 & : \text{if neuron } i \text{ is on at time } t \\ 0 & : \text{if neuron } i \text{ is off at time } t \end{cases}$$

**Interpreting McCulloch and Pitts** The McCulloch-Pitts model has been a widely accepted method for neuron to neuron connection, however in light of the above discussion, neurons in the neocortex are connected in columns, which are then connected column to columns. This implies that for interpretation in terms of biological "intelligence", the model should focus more on the actions of the columns as a whole rather than on the individual neurons.

We shall interpret the model proposed by McCulloch and Pitts for neurons, instead for columns. Thus, we shall assume that columns are either "on" or "off". In this model the firing of individual neurons within a column is not modeled; instead, the model assumes that the effects of the individual neurons in each column results in the column being "on" or "off". Thus, we can view the total effect of one column on another column. Each column has a net excitatory or inhibitory effect on neighboring columns and each column will require a certain amount of excitatory contribution in order to fire. This describes the same model proposed by McCulloch and Pitts, just with different base units. Even so, we shall use the terminology "neuron" throughout this report.

**Other Interaction Simplifications** The discrete nature of temporal changes merits more explanation. The totality of activation to a neuron (and therefore to a neuronal column) from neuronal coulmns currently firing has a cumulative effect on neuronal column $j$. On a continuous time scale it would be impossible for the impulses from two neuronal columns (say $i$ and $k$) to reach column $j$ at the exact same time. This means when column $j$ is analyzing whether or not to fire, it would have to allow for some length of time $\alpha$ in which to accept impulses. The column would sum the impulses that were received for that amount of time and then fire or not fire accordingly. This argument does assume that all the neuronal columns are in sync (all update at the same time), but still gives credence to our simplification. We will use a time scale with $\alpha = 1$.

Now with the sum above, adapted under our new simplifications, we can make a statement about the condition of neuronal column $j$ at the next time step

$$s_j(t+1) = \begin{cases} 1 & : \sum_i w_{i,j} s_i(t) - \tau_j \geq 0 \\ 0 & : \sum_i w_{i,j} s_i(t) - \tau_j < 0 \end{cases}$$

Here $\tau_j$ is a threshold value that the sum of impulses must surpass in order for column $j$ to fire. We see by the update rule that the weights between columns share a role with the threshold values $\tau_j$. Using matrix notation and the Heaviside function $H(x)$ we have

$$S(t+\alpha) = H\left(\begin{bmatrix} w_{1,1} & w_{1,2} & \cdots \\ w_{2,1} & \ddots & \\ \vdots & & \end{bmatrix} S(t) - \bar{\tau}\right)$$

Here $\S(t) = [s_1(t), s_2(t), ..., s_n(t)]$ is a vector containing the states of the columns at time $t$. The thresholds are also represented in vector notation $\bar{\tau}$.

It is important to note that $w_{i,i} = 0$ since a neuron has neither an excitatory or inhibitory effect on itself.

# 4 Structure

Hierarchy is an arrangement of objects with an order where objects are ranked one above the other. Objects higher up in the chain, though

fewer in number, have a greater influence than objects lower in the chain. Information flows from the lower regions to the higher regions with the higher objects having the greatest convergence of information. That is, the higher nodes have more nodes connecting to them than do lower nodes. If information then propagates back down from a higher region, those nodes can influence more nodes than those in lower regions. Hierarchy resolves into a topological organization that is commonly found in several complex systems. As an organization tool, it detects and measures significant features intrinsic to networks.

Knowledge clearly involves hierarchies. One important hierarchical component is classification. As an example, taxonomic groupings of individual organisms in biology sort animals into different groups at varying levels: Kingdom, Phylum, Class, Order, Family, Genus and Species. In particular, the beagle dog is classified in the Animalia Kingdom, Chordata Phylum, Mammalia Class, Carnivora Order, Canidae Family, Canis Genus, and Canis Familiaris Species [50]. This classification is based on DNA similarities or differences among the different organisms [51]. The top level, Kindgom, would sort plants, animals, and parasites, into different groups while at the lowest level cats and dogs would be sorted in different species. When we learn, we classify knowledge by using more rudimentary attributes such as color, size, skin type (fur, scales) and locomotion (biped, quadruped, walk, run) to construct the taxonomy of the objects of everyday life. For example, when first learning the notion of animal, a young child might consider chipmunks, tree squirrels and marmots one and the same. But, as this child matures intellectually, they would learn to distinguish them. Similarly, over time the child would be able to differentiate bulls from bison, ducks from geese, and beagles from bichons, despite their sometimes subtle differences.

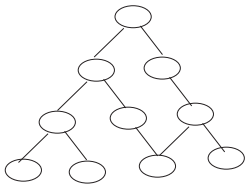


Figure 4.1: Hierarchical Arrangement

We have an intuitive concept of hierarchy. Figure 4.1 shows a prototype of a hierarchical network. But, in order to distinguish when one system is less hierarchical than another one, we need to be able to do more than just "look" at the system. We know that graphs like trees are hierarchical, but that random graphs are not. In addition, the structure of brains suggests that not all connection models are appropriate.

Braitenberg [84]claimed that are three types of neurons: Pyramidal cells, Stellate cells, and Matrinotti cells. These cells have varying properties, but the stellate cells in particular have a property related to in-

terconnections, specifically, they have axons and dendrites that extend only locally. The pyramidal cells can connect into the white matter which could in turn transfer an impulse a relatively long distance, spanning several columns.

We first quantify brains using the idea of connectedness. We do this by examining different graph types and discussing what they have in common and what they do not. To do this, we calculate different measures on various graph types, then use data mining tools obtain a measure for increasing and decreasing "hierarchical-ness." We then examine a more functionality-based idea of brain structure using the ideas of event chains. In the present paradigm, event chains are related to the idea of hierarchy in the following way: Consider the event of a sequence of firing starting with the top node, followed by any node in the next level, followed by the firing of a node in the next level, etc. The set of event chains that have as the firing of the top node as initial event, the firing of any of the nodes in the next level, etc., functionally describes the behavior of the hierarchical structure.

There are several quantifiable measures of graphs that have been used in graph theory. First we discuss the different graph types, followed by a description of the measures.

## 4.1 Graph Methods

### 4.1.1 Graph Types

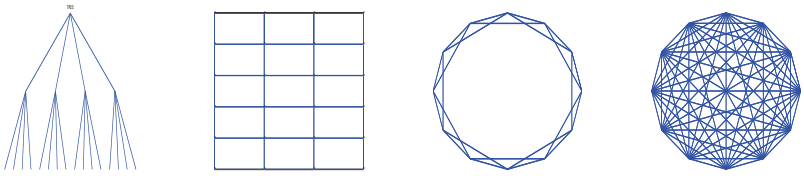
| Graph Type | |
|---|---|
| Tree | Connected graph with no simple circuits |
| Retangular | Nodes in lattice(grid) connected to clsoest points |
| Ring | Nodes in Ring connect to $n$ closest neighbors |
| Complete | All nodes connected |
| Null | No nodes connected |
| Random | Nodes connected randomly |
| Small World | Nodes connected with small pathlength, high clustering |
| Waxman | Nodes in grid connected with probability based on distance |
| Scale Free | Nodes more likely to connect to highly connected nodes |

Table 6.1 Graph Types

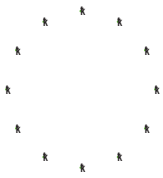
We consider graphs with $n$ nodes and connections among the nodes. These graphs can be expressed by the adjacency matrix, $A$.

The first type of graph we consider is the *Tree*. A tree is a connected graph with no simple circuits, *i.e.*, a graph that has no closed paths. In other words, in a tree, there are no paths with the same starting point and end point. If a path starts at any node, $i$, and follows a set of edges, it cannot end up back at node $i$.

The *Rectangular Array (Mesh)* and *Ring* are two other non-random graphs. In a Rectangular Array the nodes are located at rectangular lattice points and connected to their neighbors directly to the left, right, above and below. This concept can be generalized to a three dimensional rectangular array. In a Ring, the nodes can be arranged in a circle and

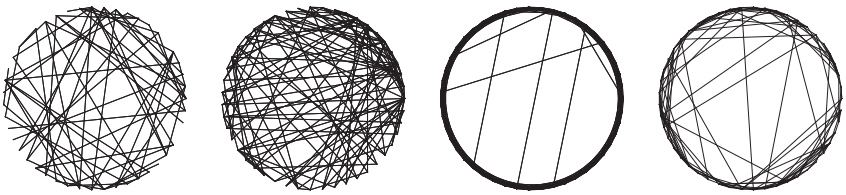


(a) Tree (b) Rectangular (c) Ring (d) Complete



(e) Null

Figure 4.2: Graph Types



(a) Random (b) Scale Free (c) Small World (d) Waxman

Figure 4.3: Graph Types

connections exist only between each node and its k nearest neighbors along the circle. A *Complete* graph is one in which all nodes connect to all other nodes, and a *Null* graph is one where none of nodes connect to any of the other nodes, including themselves. Small examples of graphs are illustrated in Figure 4.2.

*Random* graphs (ER) were first studied by Erdos and Renyi [6]. They are generated by taking a fixed number $c$ edges and using them to connect $n$ nodes. There are a total of $n(n1)$ possible connections. Each possible connection of $i$ to $j$ has a probability

$$P(i,j) = \frac{c}{n(n-1)}. \tag{4.1}$$

One property of ER graphs is a rapidly decaying degree distribution. The degree of a node $i$, DEG($i$), refers to the number of edges at node $i$. In the ER graphs, nodes with high degrees are highly unlikely [8]. There are several variations on how to generate random networks. Some can be found in [11, 9].

Three other random networks we consider are *Small World*, *Waxman* and *Scale Free*. Small World networks model the phenomena called "six degrees of separation," where any two people in the world are at most six steps away, and the cliquish nature of people, where the friends of a given person are likely to be friends with one another. Small World networks are sparse networks with with small average path length (six steps) and a high clustering coefficient (high cliquishness). They were introduced by Watts and Strogatz [45, 6, 44] and have been applied to model website linking, disease spreading, the well known actor-to-actor linkage, and the prestigious Erdos number. They are generated by taking a ring, where the nodes are connected to their $k$ closest neighbors, then randomly deleting a connection and adding one back randomly between any two nodes that were not previously connected. This is repeated one node at a time until a suitable clustering coefficient and path length is obtained [44, 43].

Waxman networks start from a rectangular array structure and connect the nodes based on a probability that depends on distance. In this network, nodes that are closer together are more likely to be connected. That is, the probability that node $i$ connects to node $j$ depends on how far apart they are. An example of a connection probability density is

$$P(i,j) = \alpha \exp\left(\frac{d(i,j)}{\beta L}\right) \tag{4.2}$$

where $d(i,j)$ is the distance from $i$ to $j$, $L$ is the maximum distance in the graph and $\alpha$ and $\beta$ are parameters [46, 44].

Scale free networks have recently been used by Barabasi [6, 44] to model networks that have nodes with extremely high degrees. These super connected or "super nodes" are found in real world networks, such as the Internet, the brain, and models for spreading of cancer and HIV [6]. These networks are robust to random node loss, but quickly become disconnected if the "super nodes" are removed. To create a scale free network a fixed connected ER model and is allowed to grow one node at time. Each time a node, $j$, is added, we add a fixed constant number, $m$, of links from that new node to the existing nodes. The existing nodes receive the new connection with a probability proportion to their corresponding degree. With this algorithm, nodes that are already highly connected are more likely to receive one of the new connections [5]. These networks reflect a theme described as "the rich get richer." The probability that the new node $j$ connects one of its $m$ links to a node $i$ is given by

$$P(i,j) = \frac{\mathrm{DEG}(i)}{\sum_k \mathrm{DEG}(k)} = \frac{\sum_k A(i,k)}{\sum_{i,k} A(i,k)} \tag{4.3}$$

The number of links added, $m$, could also be allowed to vary by picking $m$ using a Poisson distribution with mean $M$. Examples of random graphs are shown in Figure 4.3.

### 4.1.2 Graph Measures

Once we have the Adjacency matrix, $A(i,j)$ of a given graph, $G$, we calculate different measures [26]. The underlying matrix, Und defines an

undirected graph from an arbitrary graph. It is defined as

$$\mathrm{Und}(i,j) = \begin{cases} 1 & A(i,j) = 1 \text{ or } A(j,i) = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

The underlying matrix symmetrizes the directed graph into an undirected graph by making a connection from $i$ to $j$, and $j$ to $i$, if at least one connection between the two exists. If the graph, $G$, is undirected then $A(i,j) = \mathrm{Und}(i,j)$.

The first measure is the *Number of Nodes*, NV

$$\mathrm{NV} = \#\text{ of Vertices (Nodes).} \tag{4.5}$$

It refers to the number of vertices, nodes or neurons in the graph, $G$. Similarly, we can examine the *Number of Edges*, NE($i$) of each node,

$$\mathrm{NE}(i) = \sum_j A(i,j) \tag{4.6}$$

This refers to the number of nodes incident on node $i$. There can be an important relationship between NE and NV . For example, trees of $n$ nodes always have $n1$ edges. Random Graphs on average have NV $\star\, p$ edges, where $p$ is the probability that a connection exists between any two given nodes. Ring graphs have NV$k/2$ edges, where $k$ is the number of edges emanating from a vertex. Small world graphs derived from rings also have $N \star k/2$ edges because small world graphs are just rings with some of their edges redistributed. NE($i$) is calculated for every node $i$. The distribution of NE can be informative. For a scale free network the distribution follows a power law distribution. The *Average Degree of Nodes*, DEG

$$\mathrm{DEG} = \frac{\sum_i NE(i)}{NV} = \frac{\sum_{i,j} A(i,j)}{NV}. \tag{4.7}$$

is average degree over all the nodes in the graph. The maximum degree, MDEG is maximum degree over all the nodes,

$$\mathrm{MDEG} = \max_i(NE(i)). \tag{4.8}$$

A path in a graph is a sequence of edges that begins at a node of the graph and concatenates edges of the graph, always connecting pairs of adjacent nodes. The path length is the number of edges used to get from node $i$ to node $j$. If there are multiple paths connecting $i$ and $j$, we only use the shortest one. If no path exists we assign the path as being of length $NV + 1$. The distance matrix, $D(i,j)$, is a matrix where the $(i,j)$ entry is the length of the shortest path required to get from node $i$ to $j$. The Mean Path length, $VD$, is a measure of how many steps it takes to get from one node to another on average. The minimum path length between all pairs of nodes is calculated and then averaged.

$$\mathrm{VD} = \frac{\sum_{i,j} D(i,j)}{NV}. \tag{4.9}$$

We also calculate the longest and shortest path length over all connected nodes. This gives two measures, *Max Path Length* or *Diameter*, DIA,

$$\mathrm{DIA} = \max_{i,j}(D(i,j)) \tag{4.10}$$

and *Minimal Path Length* or *Radius*

$$\mathrm{RAD} = \min_{i,j}(D(i,j)) \tag{4.11}$$

We define the *Eccentricity*, ECC($i$), of node $i$ as the maximum of its finite distances (not those set to $NV+1$) to all other nodes. That is, from each node $i$, we calculate the distances to the farthest reachable node,

$$\mathrm{ECC}(i) = \max_{j}(D(i,j)), \tag{4.12}$$

then average over all nodes to get the average eccentricity.

$$\mathrm{MECC} = \frac{\sum_i \mathrm{ECC}(i)}{NV} \tag{4.13}$$

The next measure was introduced by Watts and Strogatz [44] to indicate on average how many connections exist between the immediate neighbors of a given vertex. For every node $i$ determine the nodes that $i$ connect to, $M(i)$. Suppose that node $i$ connects to $M(i)$ neighbors. The number of connections between each of those $M(i)$ neighbors is a measure of clustering. This quantity is normalized by dividing by the maximum possible connections between these nodes, $M(i)(M(i)1)$ to get the clustering coefficient of node $i$.

$$\mathrm{Clus}(i) = \frac{\sum_{(k,j)\in\omega(i)} A(k,j)}{M(i)(M(i)-1)} \tag{4.14}$$

where$\omega(i)$ is the set of the nodes that node $i$ connects to and $M(i) = \mathrm{NE}(i)$. The clustering coefficient is averaged over all nodes to find the *Mean clustering coefficient*, MC

$$\mathrm{MC} = \frac{\sum_i \mathrm{Clus(i)}}{NV}. \tag{4.15}$$

A value of measure equal to one means all the nodes $i$ connect to one another. For a completely connected graph the measure would have a value of 1, while for a tree the value would be zero.

The *Expansion* of a node $i$, $E(i,h)$ are the nodes that can be reached from node $i$ in $h$ steps that cannot be reached in fewer steps. Once a node has been counted at some $h' < h$ it is no longer counted at distance $h$. That is, even though node $i$ could reach node $j$ in $h'$ or $h$ steps, the expansion only considers the smaller distance.

$$\mathrm{EX(i,h)} = ||E(i,h)|| \tag{4.16}$$

is the number of nodes in $E(i,h)$. In a completely connected graph starting from any node all other nodes can be reached in one step. In a rectangular mesh with $n$ nodes typically $||\mathrm{EX}(i,h)|| \sim h^2/N$, while a $k$-ary tree or a random graph of average degree $k$ has $||EX(i,h)||$ proportional to $k^h/N$ [38]. The magnitude of the expansion is averaged over all nodes i to arrive at the *Average Expansion*, MEX, of the graph.

$$\mathrm{MEX} = \frac{\sum_{i,h} \mathrm{EX}(i,h)}{\mathrm{NV\ MH}} \tag{4.17}$$

where MH is the maximum distance.

The next measure is *Clustering Level*, CL. The expansion of a node $i$ is used to calculate how connected nodes are at each height, $h$. We average over the maximum height and starting nodes $i$.

$$\mathrm{CL}(i,h) = \frac{\sum_{(j,k)\in\kappa} A(j,k)}{||\kappa||(||\kappa|| - 1)}. \tag{4.18}$$

where $\kappa$ is the set of nodes in $E(i,h)$, that are distance $h$ from node $i$ and $||\kappa||$ is the number of nodes, $\mathrm{EX}(i,\kappa)$. $\mathrm{CL}(i,h)$ calculates the clustering coefficient [40] at each height of the expansion for every node $i$ and MCL is the *Average clustering level*,

$$\mathrm{MCL} = \frac{\sum_{i,h} \mathrm{CL}(i,h)}{\mathrm{NV\ MH}}. \tag{4.19}$$

The next measure is *Average Feedback*, MFK. The expansion $E(i,h)$ is used to calculate the number of connections from level $h$ to level $h-1$ over all possible connections for each give node $i$.

$$\mathrm{FB(i,h)} = \sum_{(j,k)\in\theta} A(j,k) \tag{4.20}$$

where $\theta$ is the set of nodes in $E(i,h)$ that connect to nodes in $E(i,h-1)$. We average over the distance and all nodes to get the *Average In Degree*, MFK,

$$\mathrm{MFK} = \frac{\sum_{i,h} \mathrm{FK}(i,h)}{\mathrm{NV\ MH}}. \tag{4.21}$$

The next measure is *Average Out Degree*, MOD. Using the expansion $\mathrm{EX}(i,h)$ we calculate $\mathrm{Outdeg}(i,h)$, the connections from level $h$ to the next level $h+1$ with respect to node $i$.

$$\mathrm{Outdeg}(i,h) = \sum_{(j,k)\in\psi} A(j,k). \tag{4.22}$$

The next measure is *Average Out Degree*, MOD. Using the expansion EX(i, h) we calculate $\mathrm{Outdeg}(i,h)$, the connections from level $h$ to the next level $h+1$ with respect to node $i$.

$$\mathrm{MOD} = \frac{\sum_{i,h} \mathrm{Outdeg}(i,h)}{\mathrm{NV\ MH}} \tag{4.23}$$

The next measure counts how many *Cycles of Order N* (such as $N3$, cycles of order 3, $N4$ cycles of order 4, and $N5$ cycles of order 5). A cycle of order $n$ is a path whose end node is the same as starting node, with no other repeated nodes (other than the starting and ending nodes).

The next measure quantifies the *Average Centrality*, AC, over all nodes

$$\mathrm{AC} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{SC}(i), \tag{4.24}$$

where each SC($i$) is the centrality of node $i$, a measure of how important node $i$ is in the graph. It is calculated by counting the number of subgraphs that make up a cycle starting and ending at a given vertex $i$.

$$\mathrm{SC}(i) = \sum_{ij} \frac{A^k(i,i)}{k!} \tag{4.25}$$

and $A^k(i,i)$ is the $i$th diagonal element of the $k$th power of the adjacency matrix $A$, $A^k$.

The measure *Reciprocity*, Reci

$$\mathrm{Reci} = \frac{\sum_{ij} A(i,j)A(j,i)}{M} \tag{4.26}$$

where $M$ is the total number of edges, $M = \sum_{ij} A(i,j)$. The reciprocity is the fraction of bidirectional edges, which can be normalized as the *Edge Reciprocity*,

$$\mathrm{EdgeReci} = \frac{\mathrm{Reci} - \overline{A}}{1 - \overline{A}}, \tag{4.27}$$

where $\overline{A}$ is the average of the adjacency matrix, $\overline{A} = \left(\sum_{ij} A(i,j)\right)/N^2$

The next measure is the *Average Matching Index*, AMI, The matching index is calculated for each edge $(i,j)$, where $i$ is one endpoint and $j$ the other. It is the number of other matching connections between node $i$ and node $j$ [8].

$$\mathrm{AMI} = \frac{\sum_{k\neq i,j} A(i,k)A(j,k)}{\sum_{k\neq j} A(i,k) \sum_{k\neq i} A(j,k)} \tag{4.28}$$

A low value of AMI implies the edge plays an important role as a shortcut. We average over all nodes.

A connected component is a subgraph in which each node can reached by paths from any other node in its subgraph. A breadth-first search is a graph search algorithm that begins at the chosen root (starting) node and examines all of its neighboring nodes constructing paths by adding neighboring nodes, until it reaches all possible nodes. The graph is decomposed into its connected components, $C(i)$ by the breadth-first algorithm.

Once we find the components we can calculate the *Connectivity* of the graph [7, 26].

$$\mathrm{Con} = 1 - \frac{V_c}{\frac{N(N-1)}{2}}, \tag{4.29}$$

where $V_c$ is the number of pairs of nodes that have no path between them in the underlying matrix and $N(N1)/2$, the maximum value for $V_c$, is the total number of pairs of nodes that could exist. Suppose there are $r$ components, where each $C(i), i = 1, \ldots, r$ is a component, and each component has $||C(i)||$ nodes. Then

$$V_c = \sum_{i\neq j} ||C(i)||\,||C(j)|| \tag{4.30}$$

Each graph is either completely connected, completely disconnected or somewhere in between. A connectivity of 1 means that the graph is fully

connected, that is, $A(i,j) = 1$, $\forall i,j$. A Null network has connectivity zero.

The next measure is *Direction*,

$$\mathrm{D} = 1 - \left[ \frac{V_D}{\frac{N(N-1)}{2}} \right] \tag{4.31}$$

where $V_D$ is the number of times a connection between $j$, and $i$ occurs given that a connection between $i$, $j$ exists[7].

$$V_D = \sum_{i,j} A(i,j)A(j,i). \tag{4.32}$$

The next two measures are calculated on the individual components then summed over all the components [7]. *Efficiency* is a measure of path redundancy between nodes. The more paths that exist between two nodes the less efficient the network becomes. These paths are superfluous. Extra links only add costs and use of up resources in networks. For a given component the most efficient graph is a tree. Since a tree of $N$ nodes has $N-1$ edges, $V_E(i)$ are the number of edges over $N-1$.

$$\mathrm{Eff} = 1 - \left[ \frac{\sum_i V_E(i)}{\max V_E} \right] \tag{4.33}$$

where

$$V_E(i) = N_i - (||C(i)|| - 1) \tag{4.34}$$

is the number of connections that exist in component $i$, and $N_i$ is the number of edges in component $i$, and

$$\max V_E = \sum_i (||C(i)|| - 1)(||C(i) - 2) - N_i \tag{4.35}$$

is the maximum number of connections that could exist summed over all the components.

*Least Upper Boundedness* (LUB) is a measure of the presence of a unity-of-command. It is a measure of the ratio between the number of nodes that act as "bosses" (higher nodes) to those that act as "subordinates" (lower nodes). A high value would imply that there are few bosses, and a low value would imply that there are too many. A LUB (boss) for any two nodes exists if there is a third node from which there is a path to each. If the nodes are directly connected, then we can choose the third node to be one of the two.

$$\mathrm{LUB} = 1 - \left[ \frac{V_L}{\max V_L} \right] \tag{4.36}$$

where $V_L$ is the number of pairs of nodes that have no LUB between them in each component summed over all components and $\max V_L$ is the maximum number of pairs of points that could possibly have no LUB in each component summed over all components [7].

$$\max V_L = \sum_i \frac{(||C(i)|| - 1)(||C(i) - 2)}{2} \tag{4.37}$$

| Data Mining Methods | |
|---|---|
| PCA | Reduces dimensionality by aligning data along direction of greatest variance |
| CVA | Reduces dimensionality by transforming data minimizing distances between points in the same group and maximizing distances between points in different groups |
| SVM | Sorts data inot one of two groups using a hyperplane |
| Parzen Windows | Finds the probability of a data point belonging to any given group |

Table 1: Data Mining Methods

The next measure is another way to measure *Efficiency*,

$$\text{Eff2} = \frac{1}{(N+1)N} \sum_{i,j} \frac{1}{D(i,j)}, \tag{4.38}$$

where $D(i,j)$ is the pathlength from node $i$ to node $j$. The measure quantifies the efficiency by equating it to the summation of the reciprocal of the path lengths. The farther apart nodes are the less efficient the network is [8]. The *Harmonic Mean* is the reciprocal of the efficiency,

$$H = \frac{1}{\text{Eff2}}. \tag{4.39}$$

This measure is more appropriate for graphs with more than one connected component [8].

The next measure is the *Pearson Correlation Coefficient* calculated by

$$r = \frac{\frac{1}{M}\sum_{j>i} \text{NE}(i)\text{NE}(j)A(i,j) - \left[\frac{1}{M}\sum_{j>i}(\text{NE}(i)+\text{NE}(j))A(i,j)\right]^2}{\frac{1}{M}\sum_{j>i}\frac{1}{2}(\text{NE}(i)^2+\text{NE}(j)^2)A(i,j) - \left[\frac{1}{M}\sum_{j>i}(\text{NE}(i)+\text{NE}(j))A(i,j)\right]^2} \tag{4.40}$$

where $M$ is the total number of edges. If $r > 0$ then nodes of high degree tend to connect with nodes of high degree, and if $r < 0$ then nodes of high degree tend to connect with nodes of low degree [8].

The last measure is the *Entropy* of the degree distribution

$$H = -\sum_{k} P(k)\log(P(k)), \tag{4.41}$$

where $P(k)$ is the probability that a node has a $k$ edges. This measure describes the heterogeneity of the network. The more uniform the degree distribution (i.e. all $k$s are equally likely) the higher the entropy.

## 4.2 Classification

### 4.2.1 Data Mining

Data mining is the process of extracting interesting, useful and novel information from data [10]. Machine learning provides methods to mine

| | ER | Waxman | SmallWorld | Null | Complete | Tree | Ring | ScaleFree |
|---|---|---|---|---|---|---|---|---|
| DEG | 0.07127 | 0.03419 | 0.06409 | 0.00000 | 1.00000 | 0.00135 | 0.02670 | 0.01067 |
| MDEG | 0.10414 | 0.05073 | 0.25433 | 0.00000 | 1.00000 | 0.00271 | 0.02670 | 0.14514 |
| MC | 0.07120 | 0.49799 | 0.00355 | 0.00000 | 1.00000 | 0.00000 | 0.71053 | 0.06128 |
| VD | 0.00260 | 0.00567 | 0.00482 | 1.00000 | 0.00126 | 0.98974 | 0.02560 | 0.00404 |
| MECC | 0.00399 | 0.01033 | 0.01033 | 1.00000 | 0.00126 | 0.99867 | 0.05060 | 0.00564 |
| RAD | 0.00266 | 0.00799 | 0.00799 | 1.00000 | 0.00126 | 0.00135 | 0.05060 | 0.00398 |
| DIA | 0.00399 | 0.01332 | 0.01332 | 1.00000 | 0.00126 | 0.01215 | 0.05060 | 0.00664 |
| Eff2 | 0.53225 | 0.28954 | 0.28954 | 0.00126 | 1.00000 | 0.00492 | 0.11251 | 0.34770 |
| H | 0.00250 | 0.00460 | 0.00460 | 1.00000 | 0.00126 | 0.27417 | 0.01184 | 0.00382 |
| MH | 0.00400 | 0.01031 | 0.00399 | 0.00000 | 0.00126 | 0.01031 | 0.05067 | 0.00564 |
| MEX | 0.25011 | 0.11605 | 0.00533 | 0.00126 | 0.50000 | 0.00135 | 0.02564 | 0.19209 |
| MCL | 0.05191 | 0.19819 | 0.40675 | 0.00000 | 0.49937 | 0.00000 | 0.45561 | 0.01828 |
| MFK | 0.28617 | 0.17998 | 0.00327 | 0.00000 | 0.50000 | 0.87790 | 0.29487 | 0.23502 |
| MOD | 0.28617 | 0.15364 | 0.00483 | 0.00000 | 0.50000 | 0.00000 | 0.29487 | 0.23502 |
| N3 | 0.00036 | 0.00056 | 0.21887 | 0.00000 | 0.99622 | 0.00000 | 0.00048 | 0.00001 |
| N4 | 0.00003 | 0.00001 | 0.21789 | 0.00000 | 0.99496 | 0.00000 | 0.00001 | 0.00000 |
| N5 | 0.00000 | 0.00000 | 0.25433 | 0.00000 | 0.99370 | 0.00000 | 0.00000 | 0.00000 |
| Energy | 0.00000 | 0.00000 | 0.00000 | -0.00000 | 0.00000 | 0.34657 | 0.00000 | 0.00000 |
| Reci | 1.00000 | 0.78463 | 0.00007 | 0.00000 | 1.00000 | 0.00000 | 1.00000 | 1.00000 |
| EdgeReci | 1.00000 | 0.77702 | 0.00000 | -0.00000 | 1.00000 | -0.00135 | 1.00000 | 1.00000 |
| Con | 1.00000 | 1.00000 | 1.00000 | 0.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| D | 0.00000 | 0.35441 | 0.35441 | 1.00000 | 0.00000 | 1.00000 | 0.00000 | 0.00000 |
| Eff | 0.92997 | 0.96710 | 0.96710 | 1.00000 | 0.00000 | 1.00000 | 0.97460 | 0.99065 |
| LUB | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 | 1.00000 |
| R | 1.07240 | 1.03551 | 0.05340 | 0.00000 | 1.99874 | 1.00270 | 1.02667 | 1.02482 |
| NV | 750.00000 | 750.00000 | 750.00000 | 793.00000 | 793.00000 | 740.00000 | 750.00000 | 752.00000 |

Table 2: Calculated measures for numerous graphs

such data. We use data mining techniques to choose a low-dimensional subspace made up by combinations of the graph measures to represent the data. We consider graphs types such ER (random), Small World, Waxman, Scale Free, Complete, Ring, Null, and Rectangular. We generate these graphs with 450 and 750 nodes, and calculate all 28 measures discussed in 4.1.2. Table 2 shows the measures discussed in 4.1.2 calculated for different graphs.

We use four different techniques to map a high dimensional measure space into a low dimensional subspace so inferences can be made as to where hierarchical graphs live. The four techniques are Principal component Analysis (PCA), Canonical Variate Analysis (CVA), Support Vector Machine (SVM) and Parzen Windows (PW).

**Principal Component Analysis (PCA)** Suppose $X$ represents a set of graph measure data, where each row of $X$ represents the data calculated from each of the $m$ different graphs, and each of the $p$ columns represents a different measure.

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ x_{21} & x_{22} & \dots & x_{2p} \\ \dots & \dots & \dots & \dots \\ x_{m1} & x_{m2} & \dots & x_{mp} \end{bmatrix}, \tag{4.42}$$

| | | | | | | |
|---|---|---|---|---|---|---|
| 8.4777e-001 | 4.2320e-001 | 2.1704e-001 | 1.8229e-001 | 1.1182e-001 | 6.5579e-002 | 5.4536e-002 |
| 4.3284e-002 | 9.6150e-003 | 9.4873e-003 | 2.8256e-003 | 2.3813e-003 | 2.7228e-004 | 1.6424e-004 |
| 3.0886e-005 | 1.8647e-005 | 1.5152e-005 | 5.6883e-006 | 3.7396e-006 | 3.5133e-007 | 2.2624e-007 |
| 5.4453e-008 | 4.0002e-008 | 1.5075e-008 | 9.2445e-009 | 3.9128e-009 | 1.5646e-009 | 7.0329e-017 |

Table 3: The twenty-eight eigenvalues calculated from data

where $x_{ij}$ represents measure $j$ for graph $i$ and $[x_{i1}, x_{i2}, \ldots, x_{ip}]$ is called the feature vector for graph $i$. Principal Component Analysis changes the basis in the high dimensional space and thereby transforms the data $X$ to a new set of axes so it can be studied in a lower dimensional space. The data is realigned so that the greatest variances in the data are in the directions of the first few eigenvectors of the covariance matrix, COV. The covariance matrix is calculated using $X$, by summing up the outer product of the rows of $X$. Let $\mathrm{COV}(i,j)$ represent the covariance matrix of the data $X$

$$\mathrm{COV} = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(X_i - \overline{X})^T}{n}, \tag{4.43}$$

where $X_i$ corresponds to row $i$ of matrix $X$ and $\overline{X}$ is the mean of all the data vectors,

$$\overline{X}_j = \frac{\sum_{i=1}^{m} x_{ij}}{m} \tag{4.44}$$

Using the PCA algorithms we calculate a new matrix $FM$ from $X$

$$FM = \begin{bmatrix} FM_{11} & FM_{12} & \ldots & FM_{1r} \\ FM_{21} & FM_{22} & \ldots & FM_{2r} \\ \ldots & \ldots & \ldots & \ldots \\ FM_{m1} & FM_{m2} & \ldots & FM_{mr} \end{bmatrix}, \tag{4.45}$$

where the rows of $FM$ correspond to the feature data expressed in a lower dimension($r << p$), that is, the column size has decreased and each row is a new shortened feature vector.

We perform PCA on data from different graph types. This method does not require previous knowledge about graph types, e.g. which graphs are trees, which are Waxman graphs, etc. PCA is able to map the different types of graphs based the similarities and differences between their measures in the lowdimensional space. When the points are plotted, they appear to be sorted so that each graph lies in a different region. That is different graphs types are sorted because the measures indicate they are fundamentally different graphs.

The PCA algorithm does allow the data to be represented in terms of its full eigenvector decomposition. But, if the data is well represented by fewer eigenvectors then it is not necessary to use all the eigenvectors. In order to choose which eigenvalues to keep we examine at the magnitude of their corresponding eigenvalues. The eigenvectors with the largest eigenvalues are more important than the rest.

**Evaluating PCA Results** We generate ER, Small World, Waxman, Tree, Null, Complete graphs and calculate their feature vectors. The

| | EV1 | EV2 | EV3 | EV4 | EV5 | EV6 |
|---|---|---|---|---|---|---|
| DEG | 2.3993e-001 | 2.3464e-001 | -3.5604e-002 | 4.6825e-003 | 4.7999e-002 | 1.3882e-002 |
| MDEG | 2.3492e-001 | 2.3552e-001 | 7.0542e-002 | 3.0199e-003 | 3.3737e-002 | 6.2826e-002 |
| MC | 2.0526e-001 | -3.4974e-002 | -1.3949e-001 | 1.0444e-001 | 2.6558e-001 | -4.6147e-001 |
| VD | -2.0160e-001 | 2.3260e-001 | -2.1563e-001 | -1.5464e-001 | 5.3194e-002 | -3.0645e-002 |
| MECC | -2.0359e-001 | 2.1031e-001 | -2.5172e-001 | -1.3942e-001 | 1.0988e-001 | 4.1141e-002 |
| RAD | -1.8158e-001 | 1.9450e-001 | -2.2378e-001 | 3.0747e-001 | 6.5504e-002 | 6.4812e-002 |
| DIA | -1.8283e-001 | 1.9523e-001 | -2.2132e-001 | 3.0514e-001 | 6.4539e-002 | 6.5024e-002 |
| Eff2 | 2.4515e-001 | 1.3462e-001 | 7.6129e-002 | 2.8124e-003 | -2.9240e-001 | 9.6493e-002 |
| H | -1.9863e-001 | 2.5850e-001 | -1.7242e-001 | 1.2863e-001 | -2.8135e-002 | -8.9599e-002 |
| MH | -1.3984e-002 | -1.6136e-001 | -2.7432e-001 | 1.1470e-001 | 4.0034e-001 | 5.1337e-001 |
| MEX | 2.3447e-001 | 1.1959e-001 | -8.8395e-002 | -6.9403e-003 | -3.6694e-001 | 7.4972e-002 |
| MCL | 1.6128e-001 | -9.3556e-003 | 2.6573e-001 | 1.1499e-001 | 4.6582e-001 | -2.6829e-001 |
| MFK | 1.0047e-001 | -2.3878e-002 | -3.3450e-001 | -4.4915e-001 | 1.0328e-001 | 1.0371e-001 |
| MOD | 2.2120e-001 | -1.1412e-001 | -3.0180e-001 | 1.3041e-001 | 4.2039e-002 | 1.6751e-001 |
| N3 | 2.2324e-001 | 2.5430e-001 | 4.8883e-002 | 5.4574e-003 | 1.2913e-001 | 2.8143e-002 |
| N4 | 2.2406e-001 | 2.5419e-001 | 4.3366e-002 | 5.4377e-003 | 1.3001e-001 | 3.6365e-002 |
| N5 | 2.2001e-001 | 2.5507e-001 | 7.1865e-002 | 5.7114e-003 | 1.3814e-001 | 4.0420e-002 |
| SC | 2.1217e-001 | 2.2725e-001 | -8.3882e-002 | 7.3346e-003 | 2.1615e-002 | 2.6324e-002 |
| Energy | -7.7200e-002 | 7.2156e-002 | -1.0149e-001 | -5.7658e-001 | 8.5422e-002 | -1.6246e-002 |
| Reci | 1.8242e-001 | -2.3942e-001 | -2.5385e-001 | 1.3762e-001 | -1.0614e-001 | -2.1681e-002 |
| EdgeReci | 1.8248e-001 | -2.3900e-001 | -2.5451e-001 | 1.3810e-001 | -1.0490e-001 | -2.0136e-002 |
| Matching | -1.1200e-002 | 3.7831e-002 | 3.3239e-001 | 6.5190e-003 | 2.2408e-001 | 5.3806e-001 |
| Con | 1.7802e-001 | -2.4985e-001 | 1.3388e-001 | -2.6723e-001 | 7.1382e-002 | 1.0986e-001 |
| D | -2.0792e-001 | 2.4888e-001 | 3.0392e-002 | -1.7358e-001 | -6.7091e-003 | -1.1805e-001 |
| Eff | -2.4061e-001 | -2.2880e-001 | 7.8140e-002 | -4.3939e-003 | -2.7851e-002 | 1.5935e-002 |
| LUB | -9.8188e-017 | 1.1301e-016 | -5.7885e-017 | 6.2778e-017 | 5.3169e-016 | -4.0335e-017 |
| R | 2.4016e-001 | -5.3330e-002 | -2.6683e-001 | -1.4856e-001 | -5.9234e-002 | -5.7822e-002 |
| NV | 1.5874e-002 | 1.6849e-001 | 1.0288e-001 | 4.5268e-002 | -3.8692e-001 | 2.2326e-001 |

Table 4: The first six eigenvectors

data is represented in 28 dimensional space as discussed in Section 4.2.1. Table 3 shows the first 28 eigenvalues and Table 4 shows the first six corresponding eigenvectors. Note that $\lambda_1 = 8.5 \times 10^{-1}$, $\lambda_2 = 4.2 \times 10^{-1}$, $\lambda_3 = 2.17 \times 10^{-1}$, etc. This suggests that the first eigenvector "explains" about twice as much of the variance as the second, which in turn "explains" about twice as much as the third, etc. Using the eigenvectors shown in Table 4 we convert the data set to a lower dimensional space. Figure 4.4 shows the data projected on each of the first four eigenvectors, each subfigure showing the effect of using the next eigenvector.

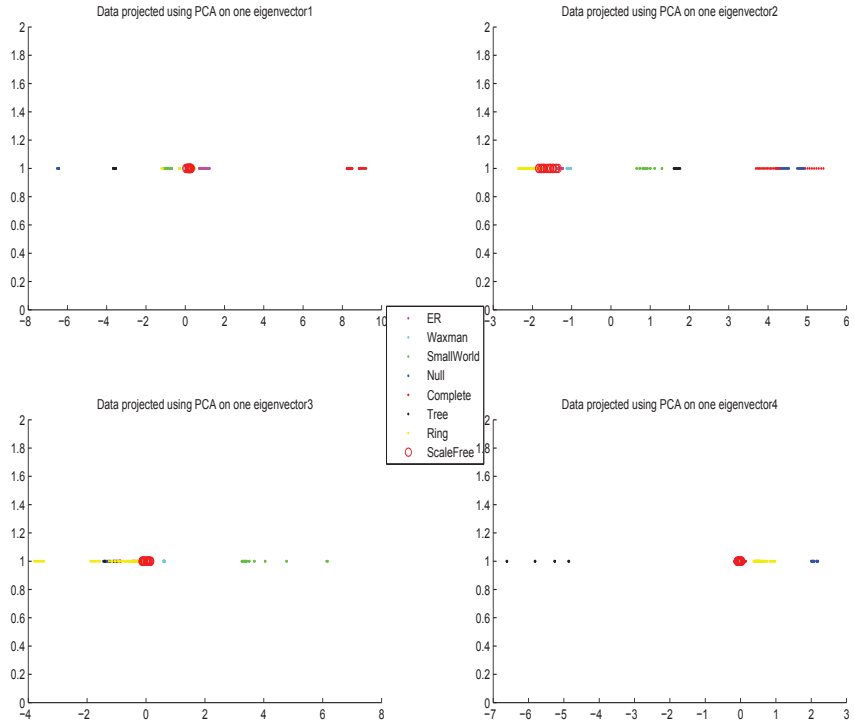



Figure 4.4: Data projected using PCA on each of the first four eigenvectors

The first graph in Figure 4.5 shows that if we project the data along the first eigenvector we can sort out the points that belong to Null (blue), Tree (black), Complete (red), ER (magenta) groups but that Rings (yellow), Waxman (cyan), Scale Free (red loops), and Small World (green) are intertwined. In Figure 4.5, using the second eigenvector seems to indicate that we are able to sort out all the groups except for Null (blue) and Complete (red). Using the third eigenvector allows us to only sort out the points that belong to the Waxman (cyan) and Small World (green) graphs; with the third eigenvector the rest of the graphs remain intertwined. Using

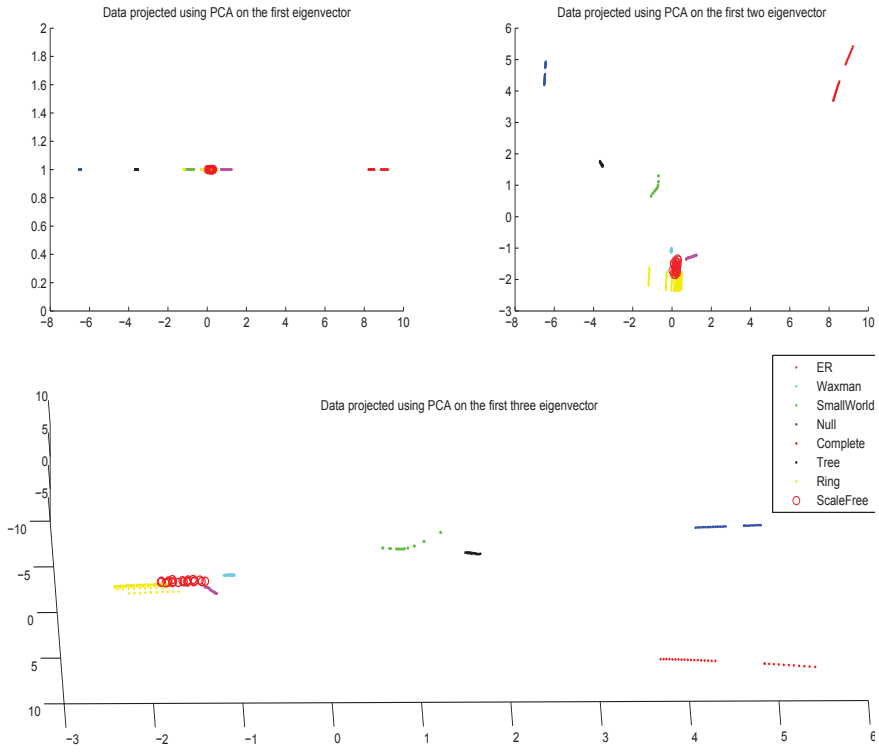



Figure 4.5: Data projected using PCA on one, two, and three eigenvectors

the fourth eigenvector the only points not intertwined are those belonging to Tree, Null, and Ring. Using three eigenvectors appears to be sufficient for sorting out the data points. Figure 4.5 shows the data projected on the first eigenvector, the first two eigenvectors and the first three eigenvectors. From all three images of Figure 4.5 it is clear the data points remain intertwined unless we at least use three eigenvectors.

**Assessing the Validity of PCA** In order to assess the validity PCA as a method for sorting graphs, we generate new graphs of several different types and use the transformation extracted from the training data to classify them. Using the transformation extracted from our training data, we generate new graphs of several different types and test how they are classified. The new test points are graphed over the original points. We show two examples. In Figure 4.5(a) the original trees are the solid black points, the blue rings are the new test trees and the other graphs are all in magenta. In Figure 4.5(b) the training rings are the yellow points, the test rings are the cyan circles, and the other graphs are in magenta. The new feature vectors projected on the first three eigenvectors appear to be

well predicted. Specifically, trees are still classified as trees and rings are still classified as rings.

We also explored the possibility of using fewer measures to do the analysis of the graphs, in particular, using only the first nine or the first thirteen measures instead of the full twenty-eight. However, we found that the graphs were not as well separated . The data remains too clustered to distinguish some of the groups. Also new points were not as well classified with fewer measures as they were with all twenty-eight.

**Canonical Variate Analysis (CVA)** Using PCA does not require any knowledge of the groups to which the data belong. If, however, we know which data points correspond to trees, which correspond to random, which correspond to small world, etc, we can use CVA [10, 28, 30]. CVA uses information about which class (group/graph type) a data point (feature vector) belongs to. It maximizes the ratio of the variation of the data from different groups to the variation of the data from the same group (inter-class)

$$\frac{\phi_1^T S_{\text{inter}} \phi_1}{\phi_1^T S_{\text{intra}} \phi_1} \tag{4.46}$$

where $S_{\text{inter}}$ is the interclass scatter matrix, $S_{\text{intra}}$ is the intraclass scatter matrix, and $\phi_1$ is the first eigenvector of $S_{\text{intra}}^{-1} S_{\text{inter}}$.

Results were also obtained using CVA to project the same data set as analyzed by PCA. These results are not shown here, but are shown in with the learning results in subsection 5. While qualitative results appear to be the same, we note that the individual groups are more clustered than they were with the PCA algorithm as shown in this section. This indicates that CVA is somewhat better that PCA.

**SVM** Sorting or classifying data is a common process in machine intelligence. We separate the training data points into one of two classes (hierarchical or not in our case), and then decide to which class new data points should be assigned. SVM finds hyperplanes that can separate sets of points into two classes. In n dimensions, the equation of a hyperplane is

$$W^T X + \beta = 0. \tag{4.47}$$

where $W = (w_1, \ldots, w_n)^T$ , $X = (x_1, \ldots, x_n)$ and $\beta \in R^1$. The hyperplane divides the data into two of classes, where the points $X_\xi$ "above" the plane are such that $W^T X_\xi + \beta > 0$ and those "below" the plane satisfy $W^T X_\xi + \beta < 0$. Training for a SVM involves choosing the values of the hyperplane parameters $W$ and  to achieve the best separation by maximizing the distance between the hyperplane and both sets of points. This is equivalent to minimizing $||W||^2$ subject to $c_i(W^T X_\xi + \beta) > 0$, where $i$ ranges over the elements in the training set with $c_i = 1$ if the point corresponds to the points that are "above" the hyperplane and $c_i = -1$ otherwise. To implement this method for determining if a graph is hierarchical, the points correspond to graph measures discussed in Section 4.1.2, along with the value $c_i = 1$ for trees and $c_i = -1$ otherwise. SVM is implemented by generating several trees that are clearly hierarchical and several

random networks. The best hyperplane is then calculated to separate the data in the training set.

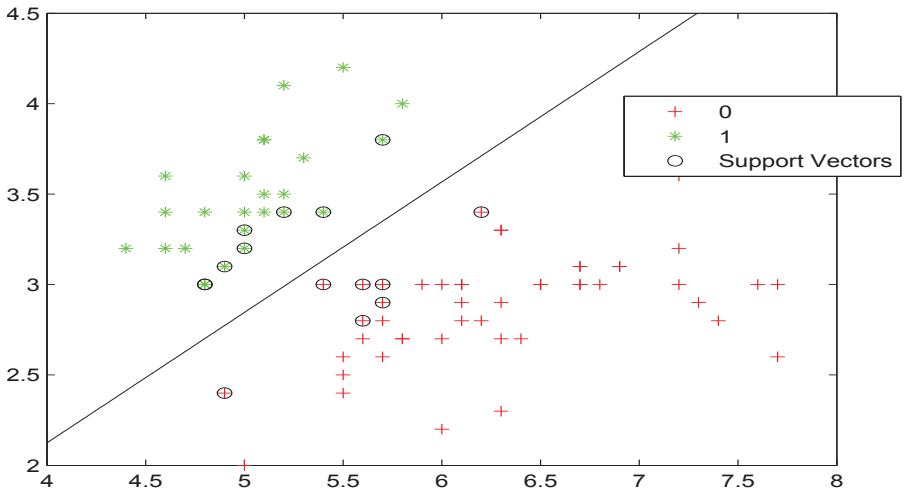



Figure 4.6: Examle of SVM separating data

If $X$ is a feature vector for a given graph, then the signed distance from a point to a plane, $d(X) = (WX + \beta)/||W||$ is a measure of how hierarchical a given graph is. We generate several known hierarchical and non-hierarchical networks, calculate feature vectors for them and determine the boundary of the region specified by these points. In Figure 4.6 we show an example of data being sorted by SVM. We will only be concerned with the distance $D(X_v)$ of a point $X_v$ to this separating hyperplane as the measure of hierarchy for that point $X_v$.

SVM therefore has a measure of hierarchy that allows the determination of how hierarchical a graph is. Specifically the (signed) distance from the feature vector of the graph to the hyperplane quantifies hierarchy. If we start with a random network, train the network, and the point representing it moves closer to the hyperplane or crosses the hyperplane into the tree region its hierarchy is increasing.

**Probability Methods** In Figure 4.5 we can see that there is slight overlap between data points of different classes. For example, some Rings and Scale Free Graphs are very close. In order to classify data points, we use probability to group the data. We do this in two, three and four dimensions. The fourth dimension cannot be easily viewed. We fit a Gaussian to each of the groups using

$$f(x) = \frac{1}{(2\pi)^{\frac{k}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right), \tag{4.48}$$

where $\Sigma$ is the covariance matrix, $|\Sigma|$ is its determinant, $k$ is the dimension of the space, and $\mu$ is the mean. Then given a new point we can assign it a probability of belonging to any one group. Figure 4.7 shows the probability distributions over the training data in one dimension.

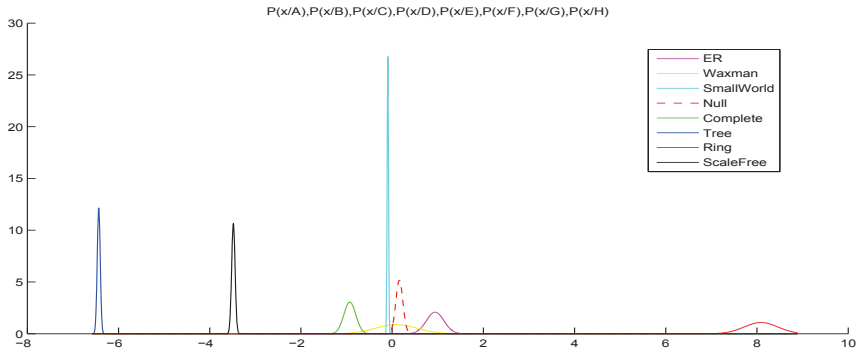



Figure 4.7: Examle of SVM separating data

Table 5 shows a subset of the training data set and how it is classified by the Gaussian distribution in one, two, three and four dimension. The first column is what we believe them to be: Random Graphs are denoted by 1,Waxman Graphs by 2, Small World by 3, Null Graphs by 4, Complete by 5, Tree by 6, Ring by 7, and Scale Free by 8. The remaining four columns indicate how they are classified by Gaussian distributions of one, two, three, four dimensions, respectively. For the most part the method classifies the points correctly. The last row is the fraction of the graphs that were correctly classified by the Gaussian distribution. This is expected, given that the Gaussian distributions were built on these points. The Gaussians of the third and fourth dimensions did the best job in predicting the correct group. Therefore, in the remainder of this paper, we will use a three dimensional probability distribution.

Eight graphs types are used in this analysis: Tree-[A], ER-[B], Waxman[C], Complete-[D] and Null graphs- [E], small world [F], Scale Free[G], Rings[H]. The probability that a given graph has measure $x$ is then calculated, given that it belongs to one of the corresponding groups. These probabilities are denoted by $P(x|A)$, $P(x|B)$, $\ldots$, $P(x|H)$. The feature vector data for each of the groups is then fit with their own Gaussian distributions by calculating the corresponding sample means and standard deviations. Bayes theorem is used to calculate the probabilities

$$P(A|x),\ P(B|x), \ldots, P(H|x). \tag{4.49}$$

These functions express the probability that a graph with feature vector $x$ belongs to group $T$, where $T = A, B, C, D, E, F, G$, or $H$, respectively. Bayes Theorem gives

$$P(T|x) = \frac{P(T \cap x)}{P(x)} = \frac{P(T)P(x|T)}{P(x)} \tag{4.50}$$

| | Should Be | Gets Classified 1d | 2d | 3d | 4d |
|---|---|---|---|---|---|
| DataPoint | 3 | 3 | 3 | 3 | 3 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 2 | 2 | 2 | 2 | 2 |
| DataPoint | 2 | 2 | 2 | 2 | 2 |
| DataPoint | 1 | 1 | 1 | 1 | 1 |
| DataPoint | 7 | 7 | 7 | 7 | 7 |
| DataPoint | 2 | 2 | 2 | 2 | 2 |
| DataPoint | 6 | 6 | 6 | 6 | 6 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 1 | 1 | 1 | 1 | 1 |
| DataPoint | 5 | 5 | 5 | 5 | 5 |
| DataPoint | 8 | 2 | 7 | 8 | 8 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 7 | 7 | 7 | 7 | 7 |
| DataPoint | 5 | 5 | 5 | 5 | 5 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 7 | 7 | 7 | 7 | 7 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 6 | 6 | 6 | 6 | 6 |
| DataPoint | 7 | 7 | 7 | 7 | 7 |
| DataPoint | 3 | 3 | 3 | 3 | 3 |
| DataPoint | 3 | 3 | 3 | 3 | 3 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| DataPoint | 3 | 3 | 3 | 3 | 3 |
| DataPoint | 3 | 3 | 3 | 3 | 3 |
| DataPoint | 1 | 1 | 1 | 1 | 1 |
| DataPoint | 4 | 4 | 4 | 4 | 4 |
| Percent Correct | 100 | 86.66 | 95.55 | 100 | 100 |

Table 5: Training data classification/Percent correct

In order to calculate $P(T)$, the number of graphs that belong to the corresponding group is counted and divided by the total number of training data points that are used. To calculate $P(x)$, count the data points that fall in a neighborhood around point $x$. Currently we use an $n$-dimensional square as the neighborhood. The probabilities must sum to one. That is, any graph must belong to one of the groups. All graphs are either complete, Null, or somewhere in between. There are two ways to accomplish this. One is to add a ninth group, Other [I], and calculate $P(I|x)$ using

$$P(I|x) = 1 - \sum_{T=A,B,\ldots,H} P(T|x) \tag{4.51}$$

The other is to normalize each on each point $x$, the probabilities $P(A|x)$, $P(B|x)$, $P(C|x)$, etc, by dividing them all by

$$\sum_{T=A,B,\ldots,H} P(T|x) \tag{4.52}$$

at each value of $x$. We found that neither of these methods gave satisfactory results. Points that were not training data were either not classified in any group or always in the "other" category. The problem is that points too far from the centers have effectively a probability of zero. A Gaussian distribution dies off too fast. Instead we chose to use Parzen windows, resulting in a method which is better at predicting points farther away from the center.

**Parzen Windows** Parzen windows forms the basis of a method that takes the discrete data points, represents them as Dirac deltas and convolves with a Gaussian function [10, 12]. It calculates the discrete distribution choosing the maximum of

$$P(A|x),\ P(B|x), \ldots, P(H|x) \tag{4.53}$$

to decide to which group the data point $x$ belongs. This method provides a way of filling in the empty spaces that the previous method could not.

## 4.3 Functional methods

Event chains have the potential of being able to reflect the structure. That is, if two neurons fire in succession with a certain frequency, it seems logical that the connection strength between them should be proportional to that frequency. Even at the most basic level we can see how analysis of event chains can help us analyze the system. A neuron that is connected to neurons with larger excitatory weights will fire more often and, in turn, we should see more chains involving that neuron. Some simple data that could be extracted from this includes the firing rate, the number of times a neuron is expected to fire over $n$ time steps. This raw data is important as is comparative data such as neuron $i$ fires $m$ times in $n$ time steps, and neuron $j$ fires $p$ times in $n$ time steps. These pieces of data allow us to make assumptions about the structure of the connections from the behavior of the neurons alone. Dissection and evaluation of

neurotransmitters to determine these connections might be be possible, but could not serve as a diagnostic tool on a living patient. We expect that neurons are more likely to connect to other neurons that are nearby.

Event chains are sequential actions. In this case, if neuron $i$ fires at time step $n$ and then neuron $j$ fires at time step $n+1$, the "two-event chain" $\{i, j\}$ occurs. Note that neurons $i$ and $j$ both produce a "one-event chain" as well. It is important to note that if we want to use this tool to give us some sense of which neurons are causing others to fire, we must only consider those that could possibly be connected. We can use the connection matrix to see which event chains are meaningful.

We consider connection matrices that have spatial structure. Instead of randomly placing neurons in a planar region in $\mathbf{R}^2$ we will use an upright square lattice. By doing this, we can define neurons being geometrically near and have the same effect on each neuron. For the present, we assume that any neuron is connected to any other neuron within a given distance, and not connected to neurons farther away than that distance.

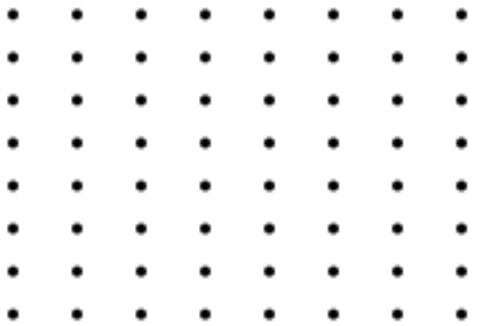


Figure 4.8: 64 Points arranged in a square lattice

Consider the lattice shown in Figure 4.8. In order to take advantage of the nature of stellate cells and local connections, we consider distance as calculated by the standard Euclidean norm. Neurons will only connect with other neurons that are less than a given distance away.

Figure 4.9 shows neurons connected based on varying distances. As the distance $r$ increases the points within Euclidean distance r are shown in Figure 4.9.

### 4.3.1 Combining Interaction and Structure

In order to eliminate the sequential firing that would occur if two neurons are connected to each other, we implement a refactory period. We assume that a neuron has a period of time after firing where it is incapable or less likely to fire. This period, called the refractory period, could be thought of as a "cool down" period. This naturally prevents a situation where $w_{i,j} > \tau_j$ and $w_{j,i} > \tau_i$ would cause the two neurons to fire on each other indefinitely if there were no other inhibitory effects present. The assump-

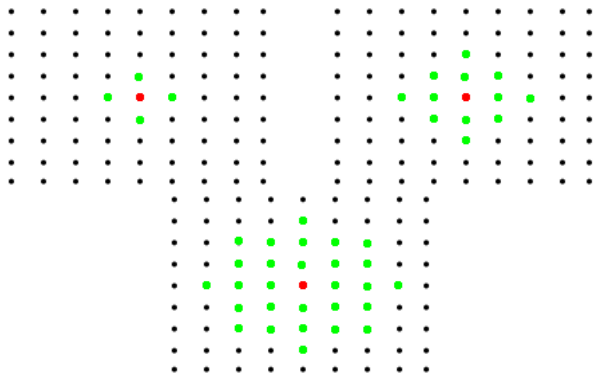


Figure 4.9: Distances of length 1,2 and 3 on an 81 point lattice

tion of a refractory period prevents a neuron from immediately feeding back, but other connection scenarios where the neurons fired in such a way as to loop back to an earlier firing neuron. These small "loops" of neurons may result in a self sustaining loop. These loops are not consistent with how the brain is perceived to operate, and we will address their effect further in Section 4.3.4. To avoid this, we will implement the model for event chains so that no neuron is allowed to fire more than once during the propagation of a single stimulus.

Since we assume finite lattices of neurons, it is necessary to consider how the neurons near the edges are connected. One possibility is that the neurons near the edges are connected in the form of a tube or a torus. This idea is a way to simulate larger samples of neurons by creating a periodic array. However, this creates more of a patchwork model of the same neuronss with the same weights. It could also be argued that this violates the locality argument defined by distance since the region itself is not physically a tube or torus. We assume the lattice does not support connections on its edges. In other words, we assume that there is no "wrapping" effect. This prevents loops and also has some structural significance.

### 4.3.2 Extracting Data from Event Chains

We can use two event chain data to gather connection information. We do this by taking advantage of the definition for statistical independence. Two events A and B are independent if

$$P(A|B) = P(A) \rightarrow \frac{P(A|B)}{P(A)} = 1$$

Thus, two events are independent if the probability of an event happening is the same regardless of whether the other event is happening. If $\frac{P(A|B)}{P(A)} \neq 1$, the events are not independent, so the outcome of one event modifies the outcome of the other.

$\frac{P(A|B)}{P(A)} > 1$ implies a positive correlation
$\frac{P(A|B)}{P(A)} < 1$ implies a negative correlation

We adapt this to two-event chains. Consider two neurons; for example, suppose neuron $i$ fires $f_i$ times and neuron $j$ fires $f_j$ times in $N$ time steps. This could be interpreted as frequencies, so that $P(\{i\}) = f_i/N$, $P(\{j\}) = f_j/N$. If the two-event chain of $j$ following $i$ occurs $f_{\{i,j\}}$ times in the same $N$ time steps, then the two-event chain $\{i, j\}$ occurs with frquency $P(\{i, j\}) = f_{\{i,j\}}/N$. We have that

$$\frac{P(A \cap B)}{P(A)P(B)} = \frac{P(\{i, j\})}{P(\{i\})P(\{j\})}.$$

If this ratio is greater than 1, then neuron $j$ fires more frequently than expected if the neurons were independent. This allows us to infer an excitatory weight or effect from neuron $i$ onto neuron $j$. It is worth noting that asymmetry is maintained and the assumptions about $w_{i,j}$ will not affect $w_{j,i}$. Since two-event chains preserve order, if $P(\{i, j\}) \neq P(\{j, i\})$ then we could still have $w_{i,j} \neq w_{j,i}$.

### 4.3.3 Simulations and Data

The model from Section 3 is used to simulate processes in the cortex, for different "brains". A set of weights and thresholds determine a "brain" that is then interrogated for its behavior. We wish to determine whether different types of brains behave differently. To do this, we postulate schemes for determining a set of equivalent brains. For the present work we generate a large number of "equivalent" brains by using a given connection scheme, given threshold value (same for all neural columns) and statistically generated interaction weights. By generating multiple "brains" from the same scheme we can compare information inherent to the general model and not to one specific weight assignment. For each brain, we find two-event trees by stimulating random neurons. The statistics of the two-event trees depend on the values of the weights and threshold.

**Assigning Preliminary Values** We consider the effects of three major components: two are network connectivity parameters, namely weights and thresholds. Two-event trees are triggered by exciting a number of neurons.

**Locality Distance** Locality distance will determines how many neurons are connected.This determines what weights and threshold values lead to reasonable "brain"-like behavior. Locality is the area around a neuron that we dictate is close enough for connections to be formed. It depends on the total number of neurons. A large connection distance with few neurons would cause all neurons in the model to be affected by the actions of one neuron. When we compute the occurrences of one or two-event chains in short bursts we are able to simulate a large brain, that is, one with many neurons. For these runs, the data was recorded for about 6400 neurons. There is a correlation between number of neurons and the time needed to run for stable data. The more neurons there are,

the longer the simulations must run so that a pulse can propagate through all the neurons. This propagation of a single pulse from initiation until no further neurons fire is called a cascade. Imagine if we excite a cascade of neurons, we can see that the larger the number of neurons, and the further from the edges the stimulation, the longer each cascade will take. It will require many cascades to get enough data to obtain reliable statistics. For example, the case of 6400 neurons, which has 38 times more neurons than the 169 neuron case. For the larger "brain", in order to obtain the same level of accuracy in the statistical data would take 38 times longer.

We discuss the topic of locality, for the 169 neuron case. We have already stated that we can get an idea for how many neuron are connected to a certain neuron as $\pi r^2$ where $r$ is the locality distance. When a cascade is induced, the number of neurons affected by a given neuron is an important parameter. In addition, the percent of neurons responding to all neurons is a parameter of interest.

| | | Active Columns | | | | |
|---|---|---|---|---|---|---|
| Dist. | Effected Col. | 1 | 2 | 3 | 4 | 5 |
| 1 | 4 | 2.3669% | 4.7052% | 6.9741% | 9.1890% | 11.3512% |
| 2 | 12 | 7.1429% | 13.7755% | 19.9344% | 25.6534% | 30.9638% |
| 3 | 28 | 16.6667% | 30.5556% | 42.1296% | 51.7747% | 59.8122% |
| 4 | 48 | 28.5714% | 48.9796% | 63.5569% | 73.9692% | 81.4066% |

Table 6: The Effect of Locality Distance

We see in Table 6 the various scenarios given an active neuronal column in the case of 169 total neurons. If we are interested in the effect of one neuron being active, we can see what percentage of all other neurons are within this distance. For example, with a distance of one, the neurons located in the cardinal directions would be affected. That is $\frac{4}{168} \approx 0.023810$ or 2.3810%. However, there is a more useful interpretation of this. We can also view this percentage as the probability a random neuron would be affected by this neuron being active. These considerations assume that edge effects are not important.

Using the cumulative distribution function for the binomial distribution we can find the probability that a neuron will fire given that a certain number of neurons are already active. The probability that exactly a neuron fall within the distance of $k$ out of the $n$ active neurons is

$$\binom{n}{k} p^k (1-p)^{n-k}$$

Where $p$ is the probability a neuron will activate if one other neuron is active with a certain distance. By summing over the possibility that any number of neurons are affecting a random neuron, we have

$$\sum_{k=1}^{n} \binom{n}{k} p^k (1-p)^{n-k}$$

**Threshold and Weight Assignment** Given a locality distance of two, a neuron is connected to twelve nearby neurons. Next we define threshold and weight values. We can start with the simple assumption that weights are distributed over a range of both positive and negative values to represent the excitatory and inhibitory values respectively. We show data for distributions of weights on the interval $[-0.2, 0.4]$, chosen to have more positive than negative values in the hope to force an overall excitatory effect.

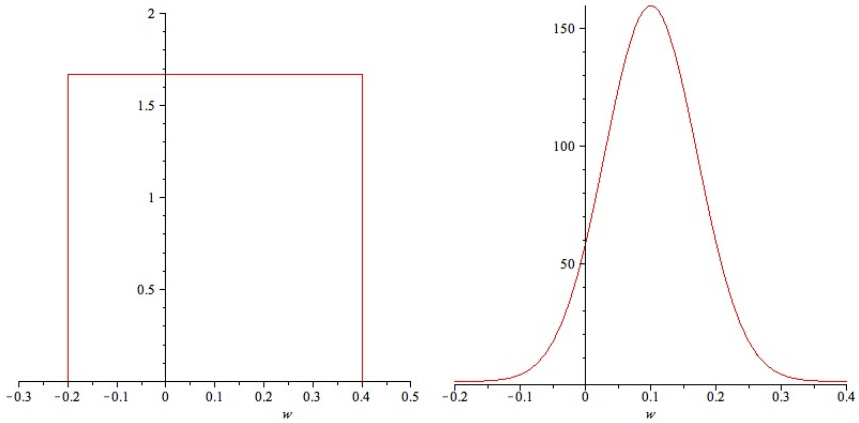



Figure 4.10: Potential Single Distributions Over the Weight Range

In Figure 4.10 we show two different distributions covering the same weight span. The simplest case is the uniform distribution where the probability of assigning any weight to an interaction is the same. The normal distribution would place a higher likelihood around a certain weight (in this case 0.1) with a lower probability of assigning a higher or lower weight value to a connection.

It is physically realistic to assume two separate distributions for excitatory and inhibitory connections, since these different connection types are related to structurally different neurons. Figure 4.11 shows two normal distributions for positive and negative weights.

We will begin with the uniform distribution, and then modify it to see the effect of the distribution on the behavior of the model.

To address the issue of a threshold use the idea the mean for a uniform distribution over $[a, b]$ is $\mu = \frac{a+b}{2}$. In our case $\mu = 0.1$ so we need the threshold to be above 0.1, otherwise the model would clearly fire too easily. Through trial, we were able to determine that a threshold value of $\tau = 0.32$ was able to maintain a firing rate for the columns that was neither too frequent or infrequent. Since the expected firing frequency for three neurons is 0.3 we know it will take roughly 3 or 4 active neurons on average within the locality distance for a neuron to fire.

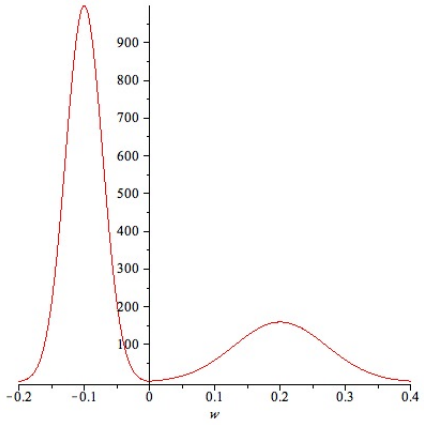



Figure 4.11: Potential Dual Distributions Over the Weight Range

### 4.3.4 One and Two-Event Chains

We will begin by retrieving the data from one and two event chains. We can then plot the weights $w_{i,j}$ against the ratio

$$R_1 = \frac{P(\{i,j\})}{P(\{i\})P(\{j\})} \tag{4.54}$$

where $P(\{i\})$ and $P(\{j\})$ are the probabilities of firing of neural columns $i$ and $j$, respectively, and $P(\{i,j\})$ is the probability of the two event chain, $i$ fires, followed by $j$ firing. From here we can find the relationship between this ratio and the connection weights. Finally, we can modify the threshold value or distribution to isolate the effect of those conditions.

For reliable data, we run the simulations and approximate the probabilities in eq. (4.54) by frequencies until any large variability within the data is eliminated. In Figure 4.12 we can see the output for one-event chains for a specific model with uniformly distributed weights over $[-0.2, 0.4]$. Here the position of the neuron is located by cartesian coordinates. Although these plots are for much larger spaces than we will be using, these larger plots accentuate some attributes of structural significance.

Figure 4.12 demonstrates the effect of the refractory period. As we discussed in Sec. 4.3.1, we restricted a neuron to firing only once during a cascade. This simulates a scenario where the refractory period is so long that no neuron will fire twice under one instance of a stimuli. This scenario can be see in Figure 4.12(ii). We can see that the firing rates of neurons under a long refractory period seem to form groups locally. This "blotting" is consistent with activity within an actual neocortex. The accompanying image, Figure 4.12(i) implements a refractory period by increasing the threshold of a neuron after it fires. This eliminates the chance a neuron will fire continuously. However, to allow the neuron to fire further along in the cascade, the threshold is lowered with each time step back to its originally designated level. This leads to a problem of

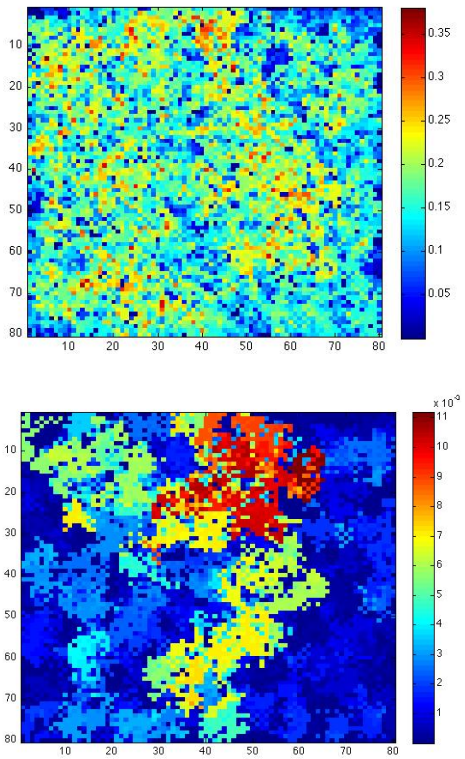



Figure 4.12: Activity in (i) One-Event Chains for 6400 neuron model with short refractory period (ii) One-Event Chains for 6400 neuron model with long refractory period

cyclic neuron firing. The easiest way to imagine this is to picture a cycle of neurons, such that each neuron can cause the next clockwise neuron to fire. If we were to excite one neuron, the cascade would begin, and by the time the cascade reached the originally excited neuron, the threshold will have returned to its original value. This will cause the original neuron to fire again and so the ring is constantly active. Figure 4.12(i) then becomes data for multiple loops and it detracts from the meaning of the firing rates of each neuron. As we increase the length of the refractory period, we reduce the possibility of these firing cycles repeating. What remains is the underlying features inherent to the locally defined structure.

Figure 4.13 shows the geometry in that connections are are nearly diagonal and about 13 units on either side of the diagonal. This banding structure which results from spatial proximity of neurons which are not nearby in index. The firing rate can be found by locating the first neuron in the chain on the $x$-axis and the second neuron on the $y$-axis. Since $w_{i,j} = w_{j,i}$ is not mandated, the firing rate for $(x, y)$ does not necessarily

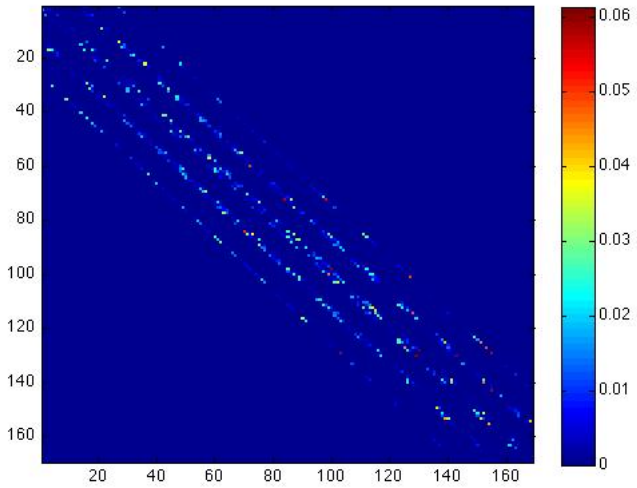



Figure 4.13: Activity in Two-Event Chains for 169 neuron model with long refractory period

equal that for $(y, x)$.

### 4.3.5 Weight and Event-Chain Correlation

By comparing $w_{i,j}$ to the ratio $P(\{i,j\})/P(\{i\})P(\{j\})$ for the frequency of the two-event chain {i,j}, we can tangibly assess the correlation between weight value and firing rates.

Figure 4.14 confirms that the ratio from 4.54 increases as the weight of the connection between the neurons $w_{i,j}$ increases. This is to be expected, since we saw in Section 4.3.2 that a positive correlation between two neurons would yield a larger ratio. There are two elements to this plot that require further attention.

First, we see that there seems to be a discontinuity in the behavior of the graph at the threshold value $\tau = 0.32$. This is because if $w_{i,j} < \tau$ neuron $i$ requires additional excitatory neurons to be active on $j$ for $j$ to fire. However, if $w_{i,j} \geq \tau$, neuron $i$ can excite neuron $j$ by itself. This may only be negated by the effect of other inhibitory neurons. Although not unlikely, the numbers used make it twice as likely that another active neuron will have an excitatory rather than inhibitory connection with $j$.

Secondly, we see that the growth in the ratio seems to be exponential as the weight of the connection increases. The assumption is that we can make a guess on the structure of the interaction by the behavior of the brain. Since this data will be fit to an approximation based on its behavior it would be simpler to deal with an easier function than an exponential.

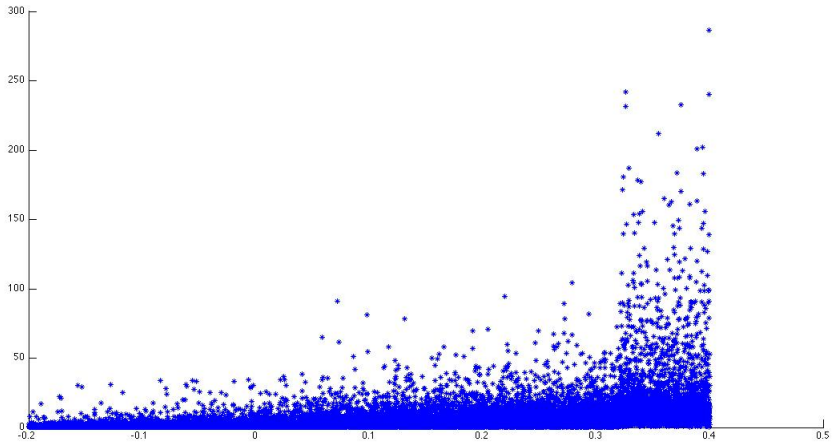



Figure 4.14: The Weight and Ratio (4.54) Comparison Data for $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$

Taking the natural log of Eq. 4.54 we have

$$R_2 = log_e\left(\frac{P(\{i,j\})}{P(\{i\})P(\{j\})}\right) \tag{4.55}$$

If the dependence in Figure 4.14 is exponential, the ratio $R_1$ can be modeled by a function of the form

$$R_1(w_{i,j}) = ae^{mw_{i,j}}$$

Here $a$ and $m$ are constants.By modeling ratio 4.55, we would have a function $R_2$ of the form

$$\begin{aligned} R_2 &= log_e\left(R_1(w_{i,j})\right) = log_e\left(ae^{mw_{i,j}}\right) \\ &= log_e(a) + log_e(e^{mw_{i,j}}) = c_1 w_{i,j} + c_0 \end{aligned} \tag{4.56}$$

Where $b$ is constant and equivalent to $log_e(a)$. This allows us to roughly approximate this function in terms of a best fit line.

Again, we are able to see the impact of the threshold at $\tau = 0.32$. Attempting to fit one line to the entire set of weight values $[-0.2, 0.4]$ would be problematic and not represent the behavior of neuron pairs with weights above or below threshold. Since there is such a large deviation in the model after the threshold value, the compensation would modify the best fit line dramatically. This would cause an inaccurate approximation for weights below the threshold, with considerable errors for negative weights. It would be in our best interest to model these two sections separately, for $[-0.2, 0.32)$ and $[0.32, 0.4]$.

**Conditionality of the Model** There are points being omitted in the data due to the nature of the model itself. Suppose we revisit the ratio given by Eq.(4.55)

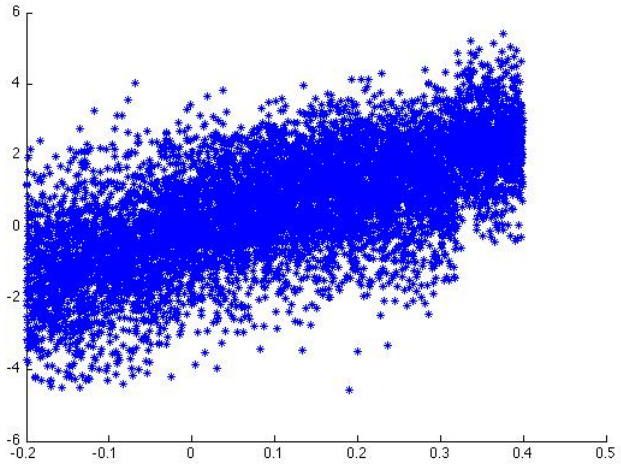



Figure 4.15: The Weight and Ratio (4.55) Comparison Data for $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$

$$R_2 = \log_e \left( \frac{P(\{i,j\})}{P(\{i\})P(\{j\})} \right). \tag{4.57}$$

It is possible that $P(\{i,j\}) = 0, P(\{i\}) \neq 0$, and $P(\{j\}) \neq 0$. In this case $R_2$ is not defined.

These points clearly can not be used in the model data itself since they represent the case where where the ratio $R_2$ is not defined. However, they represent important information, since these data points are events where neurons $i$ and $j$ did not fire during a cascade. We can simply place a conditional argument onto the data. This serves to validate the data further, eliminating the margin of error created from ignoring these points. It also serves as another point of comparison between two different realizations of brains. The number and the distribution of these non-responding points can be compared as well, to extract further behavior from the model. It is clear that we will see a higher probability of such points from connections with lower weighted values. Also note that since the neural cascade begins with multiple random neurons being activated, the denominator in Eq.(4.55) is unlikely to be zero.

### 4.3.6 Correlating the Data

In Figure 4.16 we see the linear approximations for the data properly separated around the threshold value. The sample data provided comes from multiple weight matrices under the uniform distribution between $[-0.2, 0.4]$. Using multiple weight matrices helps eliminate bias due to a certain construction of a single weight matrix. Supposing we only sampled one weight matrix, even randomly distributed, it could have perhaps a

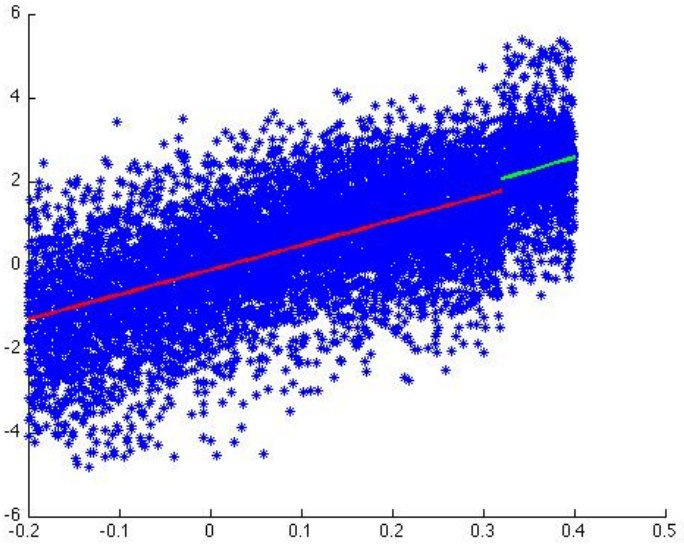



Figure 4.16: The Weight and Ratio (4.55) Comparison Data with Linear Approximations for $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$

grouping of larger weights in a certain area. Consequently the best fit line would approximate more accurately for similar weight matrices with this grouping and less so for others.

Using multiple samples of weight matrices chosen from a set of equivalent weight matrices helps eliminate partiality to a certain construction of a single weight matrix. If we only sampled one weight matrix, even though its weights were randomly distributed, a non-uniform distribution of weights could have resulted in skewed data, perhaps due to an accidental a grouping of larger weights in a certain area. Consequently a fit of the data would approximate more accurately for similar weight matrices with this particular accidental grouping and less so for others.

Here we determine the constants $c_0, c_1$ to find the equation for the linear approximation $p(w_{i,j}) = c_1 w_{i,j} + c_0$. Since this is an overdetermined system, the values found for $c_0$ and $c_1$ are found to minimize the difference, or residual, between the data and the line. Standard least squares gives

$$c_1 = \frac{n^2 \sum_{i,j} R_2(w_{i,j}) w_{i,j} - (\sum_{i,j} w_{i,j})(\sum_{i,j} R_2(w_{i.j}))}{n^2 \sum_{i,j} w_{i,j}^2 - (\sum_{i,j} w_{i,j}^2)^2}$$

$$c_0 = \frac{(\sum_{i,j} w_{i,j}^2)(\sum_{i,j} R_2(w_{i.j})) - (\sum_{i,j} w_{i,j})(\sum_{i,j} R_2(w_{i,j}) w_{i,j})}{n^2(\sum_{i,j} w_{i,j}^2) - (\sum_{i,j} w_{i,j})^2}. \quad (4.58)$$

Figure 4.17 shows data and the piecewise linear approximation for weights drawn from one or two normal distributions, respectively. The

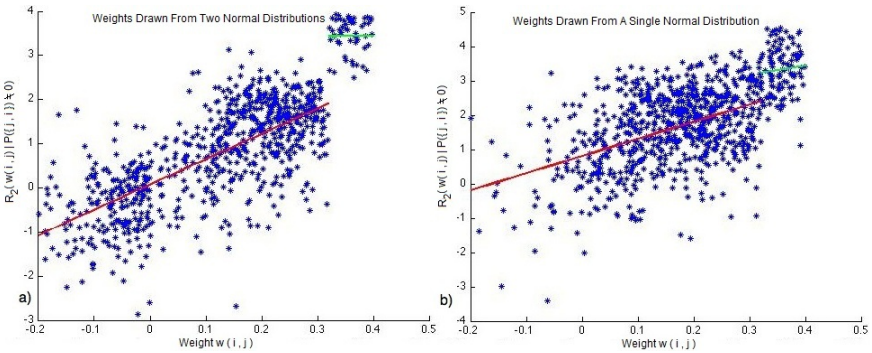



Figure 4.17: The Weight and Ratio (4.55) Comparison Data with Linear Approximations for a) $w_{i,j} \in [-0.2, 0.4]$ drawn from two normals, $\theta = 0.32$ b) $w_{i,j} \in [-0.2, 0.4]$ drawn from a single normal, $\tau = 0.32$

linear approximation may likely be less efficient in these cases, but the concept still holds. Next we describe how to use this data to make an assessment about structure from behavior.

### 4.3.7 Behavior and Structure

Figure 4.16 shows the correlation between the behavior witnessed (ratio $R_2$) and the structure (weights $w_{i,j}$). However, this data was generated starting with the weight values. The inverse problem is to find the weight distribution from two-event frequencies. However, we can use the statistical approach to classify brains, including brain models that have been subjected to learning.

The most simple and straightforward way to do this is by creating a linear approximation as we did in Figure 4.16 and then using the equation of this line as the prediction. We would first solve for the constants $c_0$ and $c_1$ by Eq.(4.58) and generate the approximation $R(w_{i,j}) = c_1 w_{i,j} + c_0$ from the reference data. Now on a new weight matrix, we retrieve the $R(w_{i,j})$ values from the behavior and use it in our reference equation to approximate the weights $w_{i,j}$. This does assume uniformity in all brains such that they are relatively similar and can be generated through a single reference, but this is unavoidable. Also, since we control the distribution and connectivity of weights, the variations will be limited.

It is clear that the linear approximation alone could generate adequate results. As Figure 4.16 shows, there appears to be a distribution of weights near the linear approximation, and that this distribution is significantly spread out. To remedy this, we can use the linear approximation value as the mean, and develop a statistical description for describing $w_{i,j} = F(R_2)$, where $F$ denotes an appropriate distribution. Without further information, the assumption of a normal distribution seems appropriate.

## 4.4 Weight Model and Prediction

In this section we determine a suitable weight given the observed behavior between two neurons. We begin with a simple linear regression assuming homoscedasticity, and refine it as we progress towards a sliding window with accompanied weighted mean. We are able to improve the model as we abandon a purely analytical model towards a simulation model.

### 4.4.1 Linear Regression Model

We have already defined how we would find the best fit line in Section 4.3.5. From here we develop the probability distribution for the weights

$$f_{n^2}(w_{i,j}|R_2), \beta_0, \beta_1, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{n^2/2}} \exp\left[-\frac{1}{2\sigma^2}(w_{i,j} - \beta_0 - \beta_1 R_2)^2\right]. \tag{4.59}$$

We can rewrite the best fit line in terms of $R_2$

$$R_2 = c_1 w_{i,j} + c_0 \rightarrow w_{i,j} = \frac{1}{c_1} R_2 - \frac{c_0}{c_1}. \tag{4.60}$$

To fit the form of Eq.(4.59) we have approximations for $\beta_0$ and $\beta_1$

$$\hat{\beta}_1 = \frac{1}{c_1}, \hat{\beta}_0 = \frac{c_0}{c_1}. \tag{4.61}$$

By putting these approximations back into Eq.(4.59) and maximizing the results, we can solve for the corresponding variance value

$$\hat{\sigma}^2 = \frac{1}{n^2} \sum (w_{i,j} - \hat{\beta}_0 - \hat{\beta}_1 R_2)^2, \tag{4.62}$$

where the sum is over the appropriate ensemble. In this work, we take the sum over all connections in the network, and over a selection of brain models.

The distributions are therefore generated from the simulations in which we observe both $R_2(w_{i,j})$ and $w_{i,j}$. From this description we can extract all the parameters for the distributions above. Once we have obtained the distributions for a large sample of data, we can use it to derive the weights in other models. Since we can obtain $R_2$ from behavior alone, we can use this observed data and the distributions to determine weights in the new observations.

### 4.4.2 Model Flaws and Improvements

The brains that we describe in this Section have a statistical distribution of connection weights. The data produced by cascades on these brains show scatter around a nearly linear regression.

If we examine Figure 4.15 we notice that although given a weight, there is a relatively small range of ratio values, the reverse is much more complex. For example, if we look at the distribution for $R_2 = 0$, we can see that it is relatively flat. This gives us a sensible approximation for the connection strength between two neuronal columns, but the variability in

this model is very large. This holds true for many other ratios because of the relatively small slope of the best fit line.

Drawing from a uniform distribution $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$, we can create a model as in Eq.(4.59). Once we establish the parameters, we generate a new set of weights that remain unknown. We use the observed behavior along with the model to estimate the unknown weights that were generated. From this exercise we observed only a 30% increase in accuracy over randomly guessing the weight. What this means is that given no information at all, a random weight from the uniform distribution will be on average 0.2 in magnitude different from the weight we are trying to estimate. Using the observed data and model, we improve this only to an average 0.14 in magnitude different between the estimate and the actual weight value. From this approach, we see that the possibility of determining the weight of the connection between two neurons, given the firing rate $R_2$ is difficult due to the spread of the data. The more realistic hope for this inverse process is to determine the distribution of connection weights, but even this seems to offer little improvement over selecting weights from a uniform distribution

We reformulate this question into the following: Can we determine if brains chosen from a set of brains (for example, all brains of 169 neurons with locality distance two and with a uniform weight distribution) are different from brains chosen from some other set? This becomes even more relevant when we examine the topic of learning.

Before we move on to the improvements for the model, we must discuss the validity in assuming a normal distribution model for the data. For least-squares regression we see that a regression line along with a normal distribution is commonly implemented to represent the spread of data [86]. To this end we compare our data with sample data from the standard normal distribution. We create a normal probability plot by taking the ordered points from our data and plot them against the ordered points from the standard normal distribution.

Figure 4.18 shows a Rankit plot, here a linear relationship between the ordered points draw from $N(0, 1)$ and the ordered data points. This indicates that the data we are using follows a normal distribution. The data is gathered within a window of consideration where $w_{i,j} = 0$ and $\Delta h = 0.1$, so there are slight deviations for $R(w_{i,j})$ values that are further from $w_{i,j}$. However, the linearity remains quite accurate and gives credibility to our assumption.

We can improve even further on this assumption by using the Kolmogorov-Smirnov test [87]. We can compare the one-dimensional distribution that results from our observed $\hat{R}(w_{i,j})$ values to that of the standard normal. We will need to standardize our distribution to have mean $\mu = 0$ and $\sigma = 1$. We use the distributions mean, $\hat{\mu}$, and standard deviation $\hat{\sigma}$ to generate a new distribution of points $R(w_{i,j}) = (1/\hat{\sigma}) * (\hat{R}(w_{i,j}) - \hat{\mu})$. The distributions of this new set of points can now be directly compared to points generated from the normal distribution $N(0, 1)$.

Figure 4.19 shows the similarity in the cumulative distribution functions for our distribution and that of the normal, $N(0, 1)$. The Kolmogorov-Smirnov test has the null hypothesis that the vector of sample points, $R(w_{i,j})$, has a standard normal distribution. The test rejects the null

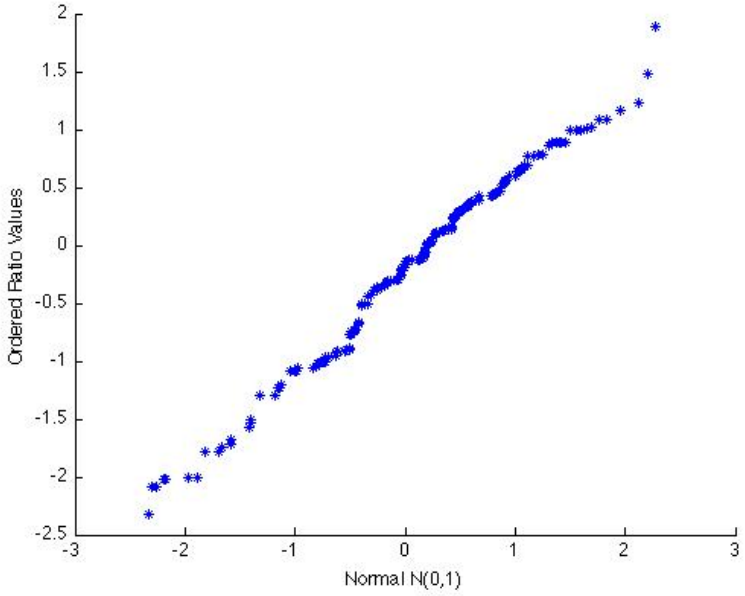



Figure 4.18: The ordered data points $R(w_{i,j})$ for $-0.05 \leq w_{i,j} \leq 0.05$ plotted against ordered data points from $N(0,1)$

hypothesis at the 5% significance level. Using the kstest in Matlab allows us to utilize this test and confirm that the data points used in Figure 4.19 do not reject the null hypothesis and therefore, the variable can be considered normally distributed with 95% confidence.

### 4.4.3 Numerical Improvement

It is possible to make improvements to the model by increasing the degrees in the polynomial used to fit the data. Instead, we examine an approach to generating a correlation based on a sliding window technique that allows the determination of a numerical approximation to the regression fit of the data.

To construct the numerical model, we first define the number of divisions and width of the window of consideration. To define a value $R_2(w_{i,j})$ for a set of weights, consider data in a narrow window around the value $w_{i,j}$. We can see in Figure 4.20 that we can use $n$ subdivisions in the weight axis, with each window having width $\Delta h$. Note that if $\Delta h$ is too small, most windows will not contain enough points for a statistical analysis. On the other hand, if $\Delta h$ is too large, trends in the data will be lost or skewed. Here the mean of the ratio is calculated numerically with the window of consideration defined between $w_{i,j} - \frac{\Delta h}{2}$ and $w_{i,j} + \frac{\Delta h}{2}$. Points within the window affect the mean, assigned to the center of the interval, based on their distance in weight value to the mean weight. If we look,

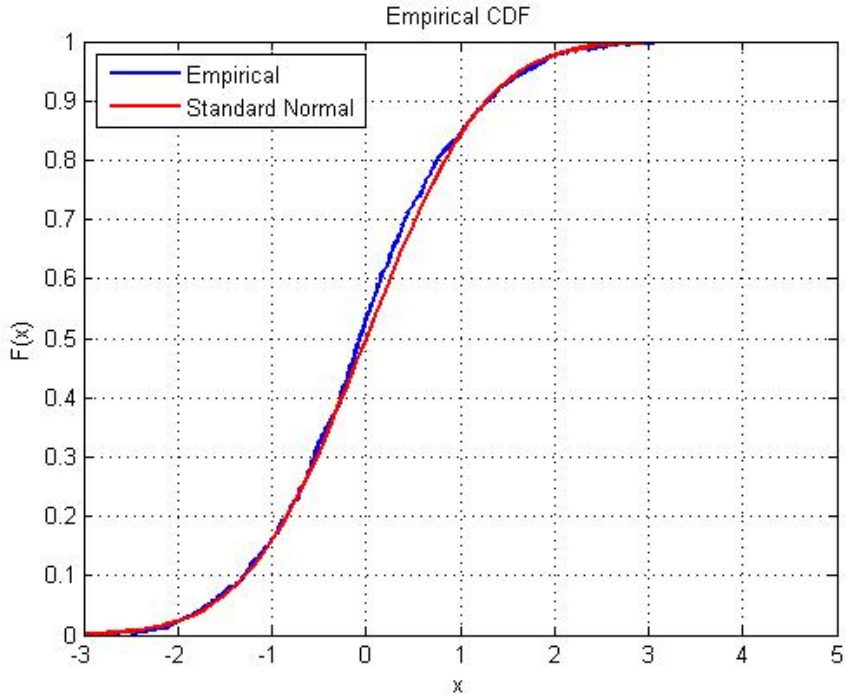



Figure 4.19: The normalized data points $R(w_{i,j})$ for $-0.12 \leq w_{i,j} \leq -0.1$ compared by their cdf against data points from $N(0,1)$

for example, at the sample point A in Figure 4.20, we see that it is within the window of consideration. However, since this point is rather far away from the mean weight value, it is weighted accordingly and therefore will impact the calculated ratio mean less than a point that is closer to the mean weight. The factor by which the point is considered in the mean ratio calculation is defined by a linear relationship with respect to distance. A point that is halfway between the $w_{i,j}$ being considered and the outer window of consideration, will have half the weight of a point with a weight value $w_{i,j}$. Choosing $n = 1000$, and a $\Delta h = 0.1$, we can create a model much like we did previously.

Figure 4.21 shows the result of this process for 30 brains with uniform distributions, each interrogated 10,000 times. The averages for each brain over the windows is shown in red and green, representing the new mean regression below and above the threshold respectively. Also shown is an average over all 30 realizations represented in blue and yellow. Here we see the improvement over the modeling used in Figure 4.16. We see that a best fit line overestimates for lower weight values and underestimates at higher weight values. In determining weights, we see how this improvement alone warrants the change from a more analytical model to the numerical one. At this point we could reevaluate the use of the natu-

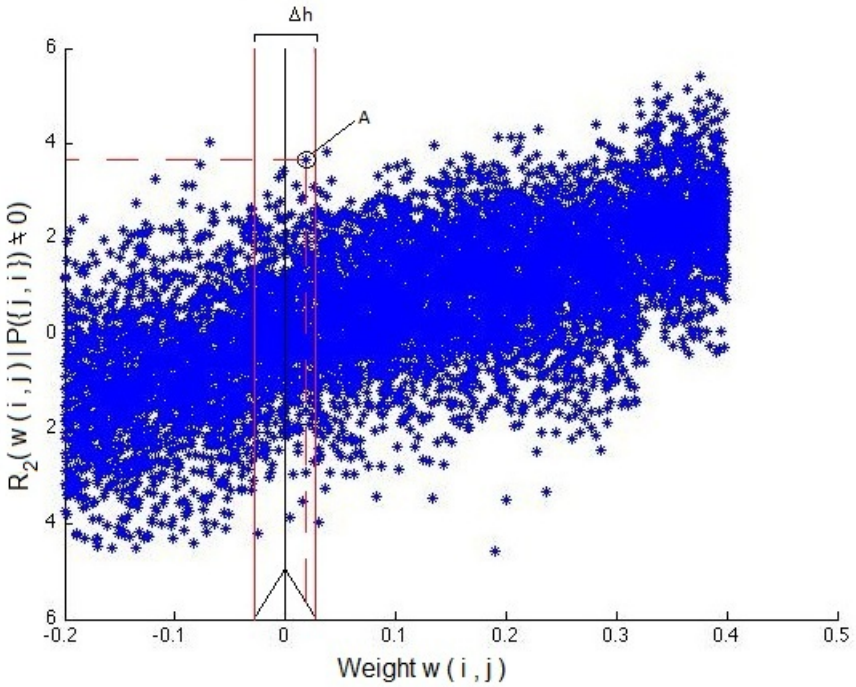



Figure 4.20: The Weighted Mean for a Given $w_{i,j}$ with Sample Point A within the Window of Consideration

ral log in the definition for the y-axis, seen in Eq.(4.55). However, there are advantages to this formulation even though the resultant graph is not truly linear. We see $R_2(w_{i,j}) = 0$ has the interpretation of having a two-event chain occurrence that follows logically from the conditions of the one-event chains and independence. This allows for a quick assessment of positive or negative correlation between two neurons. Since this should occur when the weight between two neurons is zero, we can assess any change in the model from containing $(0, 0)$ and the implications of this.

Likewise, we can numerically calculate the standard deviation using the weighted means. This is done in the same manner as described above through the definition for variance.

$$\sigma = \sqrt{E[X^2] - \mu^2} = \sqrt{E[X^2] - E[X]^2}. \tag{4.63}$$

So we can find the standard deviation numerically through calculating the weighted expected values numerically as we did above. Since the focus is no longer on extracting the weight values, the variance and standard deviation are calculated in a window around each value of $w_{i,j}$, not the linear regression line $R_2(w_{i,j})$ as previously. Since our aim is now to compare two brains, the variability of behavior given the weight becomes a more tangible point of comparison.

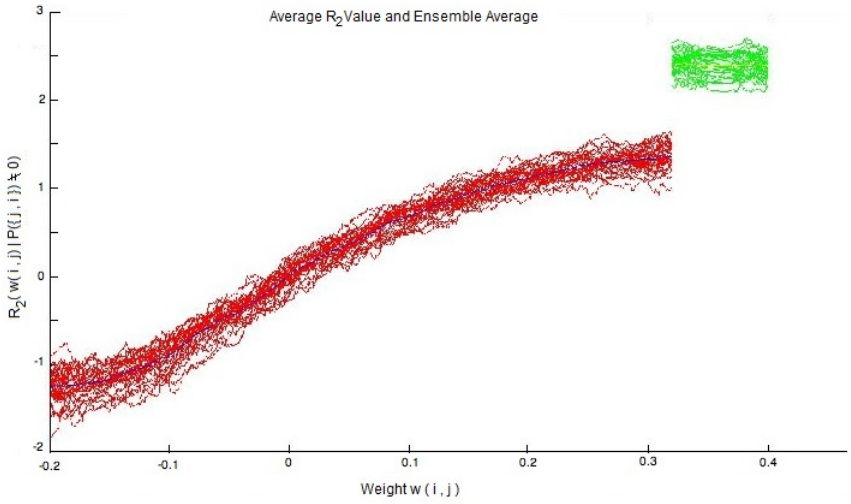



Figure 4.21: The Weight and Ratio (4.55) Comparison Data with Numerical Approximation for $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$

### 4.4.4 Normal Distribution Modeling

We have seen up to this point the behavior of a network of neural connections when they are drawn uniformly from $w_{i,j} \in [-0.2, 0.4]$ with threshold $\tau = 0.32$. Appealing to the biology of neurotransmitter released at synapses, it seems sensible that the strengths of the connections obey a standard normal distribution. This assumption will help address three issues that are potentially problematic with the uniform distribution. First, there is no biological reason to assume all weights are equally likely. It is logical to assume the connections that are extremely strong are less likely. To implement a uniform distribution, there must be a set of bounds, and this creates points where the behavior around them is also not biologically justifiable. Finally, clustering is not as dominant an issue. A few strong positive connections within a vicinity of a brain can more easily have their impact negated by a few strong negative connections.

A normal distribution of weights removes the drawbacks due to a uniform distribution, and can be used unbounded to eliminate the impossibility of certain weights being assigned. The largest impact is in regards to the clustering. The impact of a few large weights becomes greater if the surrounding weights are drawn, for example, from a mean zero normal with small variance.

To do meaningful analysis on normally distributed weights, we create a normal distribution similar to the uniform distributions used above. Since the relationship of the weights to the threshold can be crucial to obtaining meaningful cascades, that is, cascades that are neither too short or too long, this gives us a good starting point and allows us to identify any visible changes from changes in the weight distribution. We see for a normal of mean $\mu = 0$ and $\sigma = 0.2$ the behavioral plot is similar to Figure

4.21 for the uniformly distributed weights.

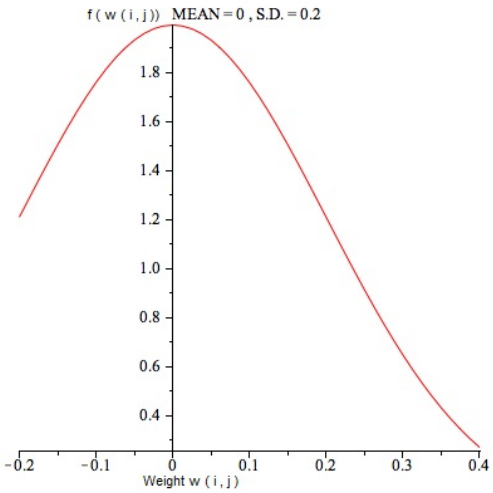



Figure 4.22: The Normal Weight Distribution for $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$ with $\mu = 0$ and $\sigma = 0.2$

The aqua and blue lines represent the mean data below the threshold and the green and yellow lines represent the mean above the threshold. Magenta and red lines are used to indicate the variance. The general shape of the graph is similar, with minor differences that we will discuss in Section 4.5.1.

Both the uniform and normal distributions have one last noteworthy issue, namely symmetry. The normal distribution will be symmetric around the mean. Just as the uniform distribution restricted us with regards to the proportion of negative and positive connections, this symmetry inherent in the normal does the same. For this we propose the possibility of using two distributions, one for the inhibitory connections and one for the excitatory connections. Using the normal distribution for both would allow us to generate them simply, but provide a bit more customization.

### 4.4.5 Reduction to Observable Data

Before we implement the statistical data, we must address a glaring detail. The data being analyzed in Figure 4.21 contains both the observable data and the weight value between two neurons. We have already abandoned the idea of extracting the weight data from the observable firing frequency data, but we must determine whether or not we gain any addition information by examining behavior independently from the weights. In other words, data describing the frequency at which two neurons fire together with respect to the individual firing rates is of little use if we know the connectivity between them. The importance of firing frequency results comes from the fact that the connectivity cannot be determined by the

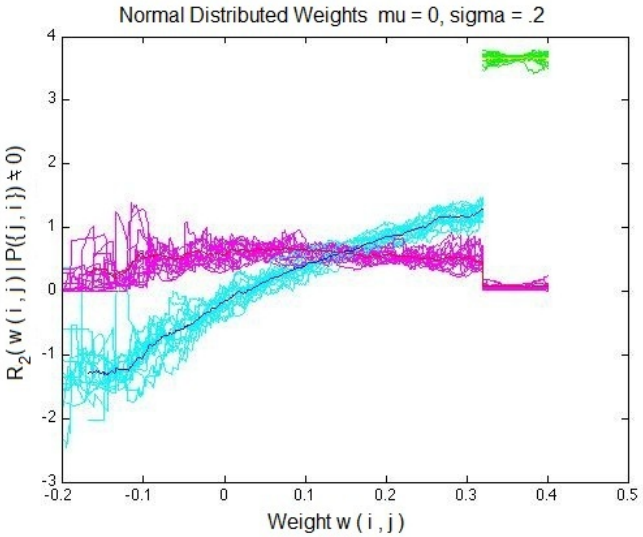



Figure 4.23: The Weight and Ratio (4.55) Comparison Data with Numerical Approximation for Normally Distributed $\mu = 0$ $\sigma = 0.2$ weights $w_{i,j} \in [-0.2, 0.4], \tau = 0.32$

weight alone. A connection with a negative weight may still fire frequently due to being in proximity to a positive weight of larger magnitude. This is one of the reasons for the variability in the data. A uniform distribution allows for equal likelihood of all weights within the range considered, but this would not be true in the case of weights drawn from a normal distribution, or potentially from two normals. Suppose we took $N(0, 1)$ to generate the weights, then a value such as $w_{i,j} = -2.5$ could have a significant effect. Such a weight is unlikely in the distribution and it would also be extremely unlikely to have a positive weight of equal or greater magnitude to combat its effect. There is also the complexity of having to adjust the locality distance or the threshold. Information gained with observed behavior can therefore add to our understanding of the brain model more than looking only at weight values.

Even though we are able to extract useful information from the frequency versus weight data, we also consider the information that can be extracted by analyzing behavior (i.e., firing frequency) data alone. This can be analyzed by projecting the data onto the y-axis. Projection will give us a distribution over $R_2$ that can be compared using several models. Since we have a monotone plot, any variability that would be apparent in the firing frequency versus weight plot is also apparent in the projection. For example, changing the threshold will cause the "jump" at that value in the plot to occur at a different position. This is obvious in a plot such as Figure 4.21, where we can infer that the projection on the firing frequency axis will also include a cluster of values with higher ratio values.

## 4.5 Predicting Brain Differences without Using Weight Data

Using the numerical results developed in Section 4.3.4, we can address the comparison of the two event data generated by two different brain models. We can vary values such as the weight distribution, connectivity, and threshold to determine their impact on the behavior of the model. The Kolmogorov Smirnov test allows us to evaluate whether a set of data is likely to have been generated from a given distribution. This test shows whether a data set was likely drawn from a normal distribution. Unfortunately, there is no numerical value to quantify the "goodness of fit" for the data to the distribution. However, at a higher level we can observe data as falling within a "class". Furthermore, the Kolmogorov-Smirnov test also allows us to directly compare two sets of data.

Since we invoke the use of a normally distributed model, it is first necessary to discuss ways to compare two normal distributions. Once this is determined, we can compare the data from varying models. We can attempt to determine the conditions on a weight matrix given its observable data and weight values. Finally we can try make this determination without access to the weight values, using only the projection data.

Using a sliding window, we create a set number of bins over the range of weights and represent the bins behavioral data by means of a normal distribution with a given mean and standard deviation. We compare models by creating these same bins over the weight range and compare the normal distributions that define the behavioral data using the Hellinger distance. The Hellinger distance computes the difference between the two distributions and returns a value between zero and one for each bin. Viewing the data bin by bin allowed us to see which weight values were more prone to show behavioral changes when models were modified. Alternatively we used a Riemann sum over all the Hellinger distances for each bin to give one value summarizing the difference between the models.

Earlier in Section 4.4, we attempted to determine the weight value $w_{i,j}$ for a connection from neuron $i$ to neuron $j$, given its behavior as determined by its frequency ratio value $R_2(w_{i,j})$. The distribution of weights for various brain models all had significant variability in the weights, given a value of the two-event frequency. Also, there is not a large discrepancy between the ratio values of different weights. These two combined factors made it far too difficult to determine an accurate weight value given only the behavioral ratio value. The difficulty is fundamental, since it is relatively straightforward to interrogate the brain and retrieve the behavioral data, but behavioral data can result from different connections with different weights. Moreover, extracting data on physical weight values would require an invasive technique (such as examining synapses) and therefore, the data is less obtainable. For these reasons, we discuss how to make useful observations based on behavioral data only. To this end, we attempt to remove the knowledge of the weight component that is required in the previous plots.

### 4.5.1 Hellinger Distance

The Hellinger distance compares the similarity between two probability distributions with respect to a third probability measure. Formally the definition of the Hellinger distance between probability measures X and Y is

$$H^2(X,Y) = \frac{1}{2}\int\left(\sqrt{\frac{dX}{d\lambda}} - \sqrt{\frac{dY}{d\lambda}}\right)^2 d\lambda. \tag{4.64}$$

Here $X$ and $Y$ are probability measures with respect to the probability measure $\lambda$. We can define this in terms of probability theory by defining $\lambda$ as the Lebesgue measure. $\frac{dX}{d\lambda}$ and $\frac{dY}{d\lambda}$ are now probability distributions.

We can define $P_n(w^n_{i,j})$ and $Q_n(w^n_{i,j})$ as the probability distributions at the $n^{th}$ subdivision in the weight axis where $w^n_{i,j}$ is the center of a bin as in Figure 4.20. The formulas are simplified if we can treat both of these probability distributions as normal distributions, in that case we define $P_n(w^n_{i,j}) \sim N(\mu_{1,n}, \sigma^2_{1,n})$ and $Q_n(w^n_{i,j}) \sim N(\mu_{2,n}, \sigma^2_{2,n})$[95]. Each mean and standard deviation is denoted by its corresponding distribution and $n^{th}$ subdivision. The final squared Hellinger distance for the two normals $P \sim N(\mu_1, \sigma_1^2)$ and $Q \sim N(\mu_2, \sigma_2^2)$

$$H^2(P_n, Q_n) = 1 - \sqrt{\frac{2\sigma_{1,n}\sigma_{2,n}}{\sigma^2_{1,n}+\sigma^2_{2,n}}}e^{-\frac{1}{4}\frac{(\mu_{1,n}-\mu_{2,n})^2}{\sigma^2_{1,n}+\sigma^2_{2,n}}}. \tag{4.65}$$

Similar to the use of the Kolmogorov-Smirnov test used in Section 4.4.2, we can verify that for a subdivision of weights, the corresponding $R(w_{i,j})$ values in this subdivision are normally distributed. This allows us to analyze the difference between two weight vs. ratio plots by comparing them for each subdivision.

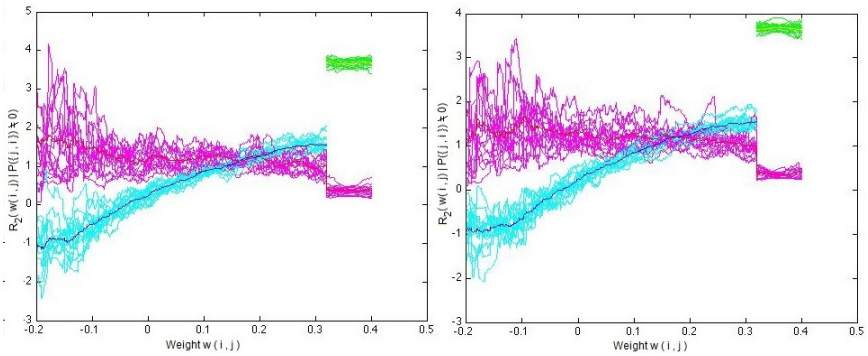



Figure 4.24: Two Weight vs. Ratio plots generated from uniform weights

The plots being compared in Figure 4.24 are quite similar. For both plots, the weights were generated from a uniform distribution and as we can see the differences between the two resultant weight vs. ratio plots are very small. For all 500 weight subdivisions, we compare the normally distributed ratio values for two plots using the Hellinger distance.

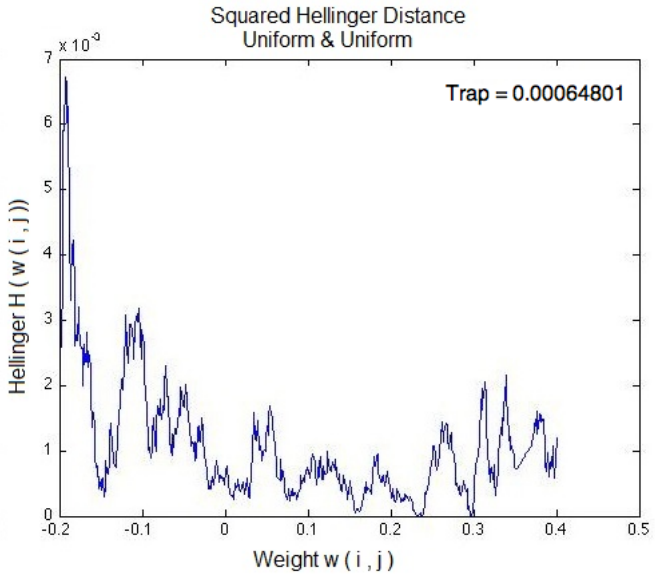



Figure 4.25: The Hellinger comparison of two Weight vs. Ratio plots generated from uniform weights

For each data point in Figure 4.25 we see the Hellinger distance between the weight subdivisions for the two plots in Figure 4.24. We see that the largest discrepancy occurs for the lower weight values. This is to be expected since neuronal firing that involves low weighted connections is less common and is largely a product of a statistical anomaly. This can also be argued to a lesser degree for the largest of the weights. In order for a neuron to fire if it has connecting weights substantially below the threshold it must be stimulated by the other connections around it. We can also use this data to create a single value to determine the "closeness" of two weight vs. ratio plots. The idea is to use the mean value theorem from calculus showing that the integral of a function is the mean value of the function, times the length of the interval. By taking the area under the Hellinger values $H(w_{i,j})$, we get a single value to represent the overall distance between the two. Due to the nature of the data, numerical integration by means of the trapezoidal rule accomplishes the desired result. For example, with 500 subdivisions and $w_{i,j} \in [-0.2, 0.4]$ we have

$$\int_{-0.2}^{0.4} H(w_{i,j}) \approx \frac{0.3}{500} \sum_{k=1}^{500} H(-0.2+k\frac{0.6}{500}) + H(-0.2+(k-1)\frac{0.6}{500}). \quad (4.66)$$

The entire plot in Figure 4.25 falls below a Hellinger distance of 0.016. Since the Hellinger distance between the two is small, we conclude that the ensemble average of the brains compared are very similar. Indeed, the Hellinger distance in this case sets a threshold below which brains

are indistinguishable from those with uniformly distributed weights with locality 2.

Next, we compare the weight vs. ratio plots generated from uniform weights against those from normally distributed weights. Figure 4.25 shows weights generated by a normal with mean $\mu = 0$ and standard deviation $\sigma = 0.2$. Again, with this data we see that the lower weight values take on a larger range of ratios. We also see that since our distribution is less likely to generate lower weight values, there are fewer data points, leading to an even greater disparity in the mean ratio values for these subdivisions.

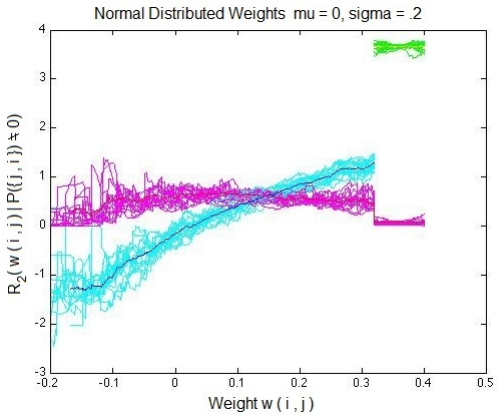



Figure 4.26: Weight vs. Ratio plot generated from normally distributed weights with $\mu = 0$, $\sigma = 0.2$

Even though we observe that Figure 4.26 is different from those in Figure 4.24 just by inspection, application of the Hellinger distance (see Figure 4.27) shows by how much.We can see that the difference in Hellinger distance is sufficiently large that brains generated with a uniformly distributed weights and normally distributed weights can be distinguished.

Quantitatively, the Hellinger distance between the brains compared in Figure 4.24 is 0.00064801, we see that the Hellinger distance between uniformly distributed and normally distributed is 0.05109837. This should be expected since these two plots are fundamentally different. The effect of the threshold, $\theta = 0.32$ becomes noticeable as well. Since these brains are fundamental in how we interpret our event-chain data, a means of comparison between such plots is extremely useful.

### 4.5.2 Detecting Brain Size

We use the Hellinger distance to see the effect of increasing the number of neurons being considered. For many simulations on the brain models, we used 169 neurons to define the weight matrix. By increasing the number of neurons in the model, we reduce the relative importance of boundary neurons. These boundary neurons have fewer connections and so there is

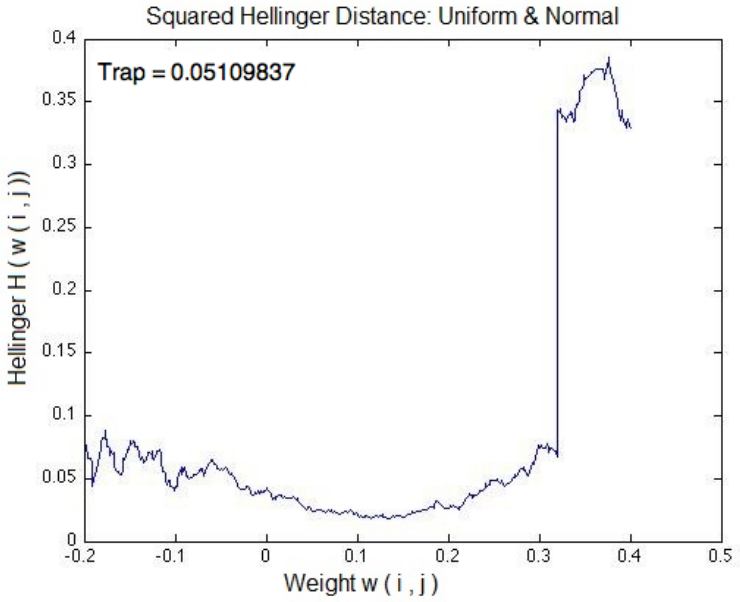



Figure 4.27: Hellinger distance between a plot generated from normal weights with $\mu = 0$, $\sigma = 0.2$ and a plot generated from uniformly distributed weights

potential for the data to be skewed. To evaluate this effect, we compare different brain models of varying amounts of neurons using the Hellinger distance.

Figure 4.28(a) shows the effect of reducing the number of neurons in the weight matrix. Likewise Figure 4.28(b) shows the effect of increasing the number of neurons and therefore reducing the impact of the boundary. In both cases the effect is minimal, and within the range of reasonable randomness in the interrogation process. Both values for the mean Hellinger distance are less than observed when comparing the two identical brain models in Figure 4.25. We can conclude that the effect of moving from $n^2$ neurons to $(n \pm 1)^2$ neurons is almost negligible for values of $n$ around 13. We can increase the number of neurons further to see how much influence the boundary effect has.

Figure 4.29 shows two cases with a larger difference in the number of neurons. Figure 4.29(b) is for the 100 and 400 neuron cases, which are the smallest and largest samples considered during simulations. Figure 4.29(a) quantifies the difference between the 169 neuron case that is more commonly used throughout the thesis, and the 400 neuron case. The difference is 10 to 40 times larger than what we saw when only considering a step from $n^2$ to $(n+1)^2$, depending on the value of $n$ we consider. However, most importantly, the Hellinger distance is still considerably less than the one seen in Figure 4.27, comparing a weight matrix of uniformly distributed weights to a matrix of normally distributed weights. We notice that the weight values that differ the most are larger in magnitude. When

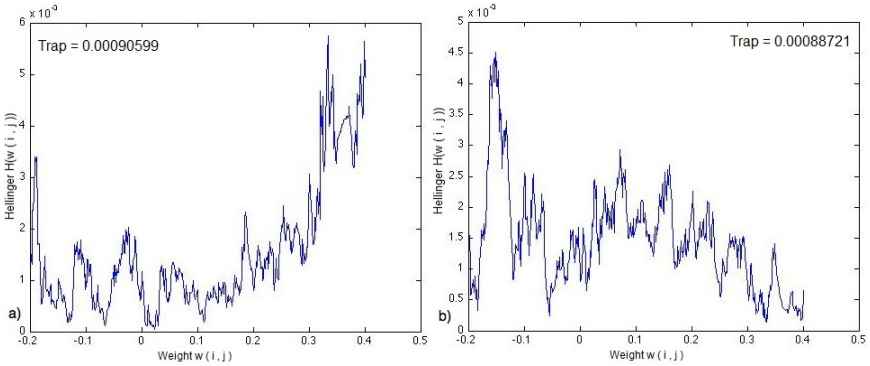



Figure 4.28: (a) Hellinger distance comparing a 144 neuron brain model to a 169 neuron model. Both weight matrices are generated using a uniform distribution.(b) Hellinger distance comparing a 169 neuron brain model to a 196 neuron model. Both weight matrices are generated using a uniform distribution

computing the interaction, the sum of connected neurons will be impacted more by these neurons. This means that larger positive weights will have more unique impacts, as we saw in the learned models. Since we only consider cases where the two-event chain data is available, the negative weights with larger magnitude are rarer and have more variability. Both of these factors are why we see these larger differences. It is also important to have an understanding of the comparison beyond simply the number of neurons.

| Neuronal Columns | Distance 2 Connections | Lost Boundary Connections | % of Lost Connections | Potential Connections |
|---|---|---|---|---|
| 100 | 1004 | 196 | 16.33% | 10000 |
| 144 | 1492 | 236 | 13.66% | 20736 |
| 169 | 1772 | 256 | 12.62% | 28561 |
| 225 | 2404 | 296 | 10.96% | 50625 |
| 400 | 4404 | 396 | 8.25% | 160000 |

Table 7: A comparison of connections and the influence of boundaries given the number of neurons under consideration

Table 7 gives pertinence to the effect of increasing the number of neurons used in the weight matrix. The 400 neuron case has roughly 2.5 times as many connections than the 169 neuron case that is primarily used throughout the thesis. Yet even with such a large relative increase in the number of columns, we see a fairly minor difference in the data produced. In the 169 neuron case, without interference from a boundary, each neuron would connect to twelve other neurons given a connection distance of size two. When we introduce the boundary 12.62% of those connections are not viable. In the 400 neuron case this is reduced to 8.25% and yet we see a small difference in the behavior when compared using the Hellinger distance.

From Figure 4.29(b) and Table 7 we can conclude further behavior

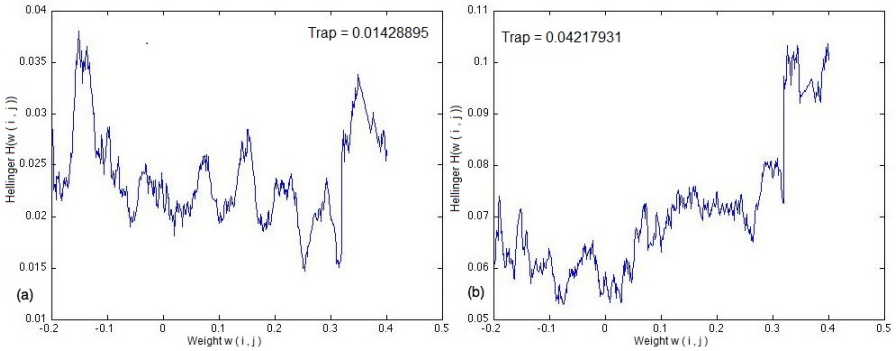



Figure 4.29: (a) Hellinger distance comparing a 169 neuron brain model to a 400 neuron model. Both weight matrices are generated using a uniform distribution.(b) Hellinger distance comparing a 100 neuron brain model to a 400 neuron model. Both weight matrices are generated using a uniform distribution

resulting from the increase in the number of neurons. The number of connections that are not viable due to the boundary increases from 16.33% to 8.25% when the number of neurons is increased from 100 to 400. This is twice the change we see from the 169 and 400 case, however the mean Hellinger distance is 0.014289. We see from the data that as we increase the number of neurons, we are lowering the fraction of neurons that are affected by the boundary, and changing the behavior by less each time. We conclude that any gain in accuracy within the model comes from an increase in the number of connections we use, and even then would most likely have relatively similar results to a model with fewer neurons.

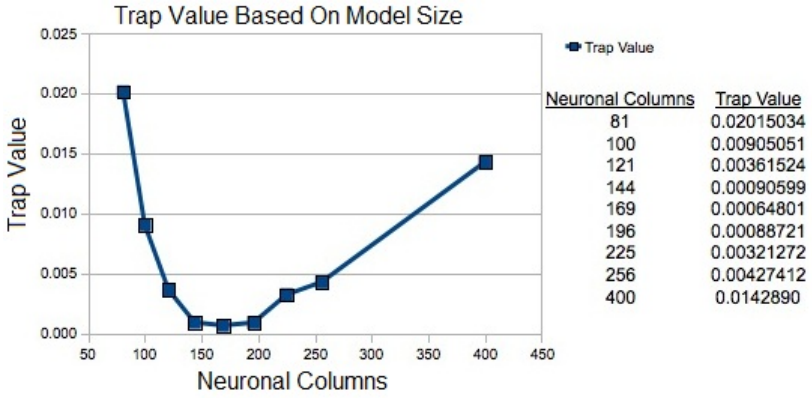



Figure 4.30: The Trapezoidal Sum over the Hellinger distances comparing a uniformly distributed brain model of 169 neurons to that of models with varying neurons

One last point to be made is that we are looking at unlearned models

for comparison. As we see in Table 7, given $n$ current neurons, there are $24\sqrt{n} - 8$ new connections by increasing $n$ to the next perfect square. However, the learned model does not restrict connections to a locality distance of two. Table 7 shows a much more rapid increase in the number of viable connections if we do not have this restriction. If we increase the model to limit the effect of the boundaries, we must be able to evaluate a considerably larger number of new connections when implementing learning. Revisiting the comparison of a 169 neuron model to the 400 neuron model, we see the learned model would potentially have over $130,000$ more connections, or over five and a half times as many. With these considerations, we postulate that the improvements gained by increasing the size of the model slightly does not show much change, and a sufficiently large size increase to make an impact in the data worth considering would incur massive increases in storage and run times. Figure 4.30 verifies this assumption for the model containing 169 neurons. Almost any model is limited by computer resources and even so, will not be close to any realistic biological brain size [96]. As an example, in the eye of a rodent, the external plexiform layer there are on the order of $10^9$ synapses [97]. The present model with full connectivity would only contain around 28,500 connections and increasing the model size to match that of the human V1 region in the neocortex would take an astonishing increase in computer resources.

### 4.5.3 Detecting Damaged Neurons

In the previous subsection, we focused on the boundary effect and how not having connectivity would alter the results. We claim that a finite array of neurons could be embedded in an array of inactive neurons without any change in behavior. For example, if our 169 neurons was laid inside a square annulus of unresponsive neurons, the results would remain the same. It therefore seems appropriate to address the impact of damage on the behavior of the model by considering regions of inactive neurons interior to the brain.

When submitting the model to damage, we simply remove any connectivity or weight associated with a certain subset of neurons. This way, it contributes neither an inhibitory or excitatory effect. It is important that we also design a damage scenario that is not a reformulation of the previous problem with increased or decreased numbers of neurons.

We consider a damaged brain constructed by severing a single horizontal line of neurons through the middle of the square lattice. Since we damage only a single row and the connectivity allows for distance two, we still have partial connectivity between the two regions.

Figure 4.31 shows the connectivity for the neurons for this amount of damage. Since the locality distance is two, had the damage been applied to two adjacent center rows, the analysis would have been similar to that of a smaller rectangular lattice without damage. Thus, the damage shown in Figure 4.31 can only be classified as damage and not simply a restructuring of the boundaries.

In this case, there are 156 active neurons, and therefore the potential for 1872 connections. Given the boundary connections and the damage,

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 5 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 8 | 11 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 11 | 8 |
| 8 | 11 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 11 | 8 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 6 | 8 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 8 | 6 |
| | | | | | | | | | | | | |
| 6 | 8 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 8 | 6 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 8 | 11 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 11 | 8 |
| 8 | 11 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 11 | 8 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 5 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 5 |

Figure 4.31: Representation of connectivity of 169 neurons given a row of damaged neurons. The value listed is the number of connections active for a neuron in that position

there are only 1526 active connections. We have 18.48% of potential connections lost to these two conditions. In comparison, a case with no damage and 81 neurons has 18.00% of potential connections lost due to the boundary conditions alone. However the average Hellinger distance between our damaged model and the 169 neuron undamaged model is only 0.00118332. Examination of Figure 4.30 shows that the 81 neuron case has mean Hellinger distance of 0.02015034. Therefore the effect of the damage on the model is far less drastic than reducing the number of neurons considered.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 5 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 8 | 11 | 12 | 12 | 11 | 11 | 11 | 11 | 11 | 12 | 12 | 11 | 8 |
| 8 | 11 | 12 | 11 | 9 | 8 | 8 | 8 | 9 | 11 | 12 | 11 | 8 |
| 8 | 11 | 11 | 9 | | | | | | 9 | 11 | 11 | 8 |
| 8 | 11 | 11 | 8 | | | | | | 8 | 11 | 11 | 8 |
| 8 | 11 | 11 | 8 | | | | | | 8 | 11 | 11 | 8 |
| 8 | 11 | 11 | 8 | | | | | | 8 | 11 | 11 | 8 |
| 8 | 11 | 11 | 9 | | | | | | 9 | 11 | 11 | 8 |
| 8 | 11 | 12 | 11 | 9 | 8 | 8 | 8 | 9 | 11 | 12 | 11 | 8 |
| 8 | 11 | 12 | 12 | 11 | 11 | 11 | 11 | 11 | 12 | 12 | 11 | 8 |
| 7 | 10 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 11 | 10 | 7 |
| 5 | 7 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 8 | 7 | 5 |

Figure 4.32: Representation of connectivity of 169 neurons given a five by five grid of damaged neurons. The value listed is the number of connections active for a neuron in that position

In Figure 4.32 we propose another damage scenario. Again, we avoid a case where the damage mimics boundary condition problems that were already addressed. Here the active neurons form a square annulus around

a five by five damaged area at the center of our 169 neuron square lattice. The damage to the grid reduces the number of active neurons to 144, and the unrestricted number of connections to 1728. The number of connections in Figure 4.32 is 1376, leading to 20.37% of connections being lost due to the effect of the boundary and damage. Applying the same Hellinger distance metric to this model in comparison to the undamaged 169 neuron case, we find the MHD (mean Hellinger distance) is 0.00128543. Since there are more connections lost due to this damage than with a single row of damage neurons, it is not surprising that we have a slightly higher trapezoidal sum. We note that this number is still substantially less than if the loss of connections came purely from the boundary.

The ability for the model to handle damaged connections has been found to occur even in models that have learned and retained their ability of associative memory. David MacKay writes, "The network can be severely damaged and still work fine as an associative memory. If we take the 300 weights of the networks... and randomly set 50 or 100 of them to zero, we still find that the desired memories are attracting stable states." [98] Although we do not address it in this thesis, learning and memories are inextricably intertwined and explaining how learning occurs would also make a statement about the way memories are formed [99].

What we have found is that the effect of lowering the connectivity depends on the impediment. The behavior of the interaction of neurons is based on propagating a flow of impulse. What we can see from the examples is that decreasing the number of neurons we consider and increasing boundary effects impedes this flow more than for the damaged cases. We note that although there are more damaged neurons and less connectivity for the brains used to generate Figure 4.32 than in Figure 4.31, they differ from the undamaged model by nearly identical amounts. This suggests a new variable representing the interruption of flow on a cascade of impulses. By comparing models with similar percentages of lost connections, as we have above, we can compare the relative impacts of the impediments themselves.

# 5 Learning Algorithms

| | |
|---|---|
| Hebbian | Incrementally changes weight if nodes are on simultaneously |
| Anti-Hebbian | Weights decrease if neurons fire together |
| Oja | Allows network to forget patterns it does not see very often |
| Asymptotic | Incorporates an Asymptotic limit |
| Bilinear | Weights increase on product of activity and decreases only if one fires |
| Covariance | Decreases the weight if one neuron is on but not the other not |
| Rate | Increases weights like Hebbian except it also constantly |
| Depressed | Like Hebbian but always reduces the weight |

Table 8: A summary description of each learning algorithm

In this Section, we consider learning models applied to Hopfield networks, and how they affect the graph structure. In this paradigm, the network is represented by weights $w_{ij}$ represents the strength of the synaptic connection from neuron $i$ to neuron $j$. Learning consists of modifying the weight matrix so that some connections are strengthened and some are weakened. We shall assume that all learning algorithms can be written as a increment to the weights. That is, we assume that learning algorithms have the form

$$w_{ij}(t+1) = w_{ij}(t) + \mathcal{L}(t). \tag{5.1}$$

Networks can be unsupervised or supervised. Unsupervised networks take unlabeled data and discover patterns, features, regularities, correlations, or categories intrinsic to the data. They also discard redundancy in the information by lowering the energy of the system. Unsupervised networks include Hopfield networks and competitive learning networks [4]. Supervised networks are directly manipulated so that they learn the data. As they do so there is a quantitative measure of error that determines how to change the connection weights so as to learn the data. Supervised networks are given data in the form of input and targets, the targets being the desired response of the neural network to the input [29].

It is difficult to program a computer to learn how to drive a car or distinguish the difference between a dog and cat, so we turn to how humans learn for inspiration. We are capable of learning from data: Extracting patterns, categorizing into groups and discriminating between two or several similar items. For example a four year old could tell that a maple tree is a tree, even if all she might have seen are oak trees in her life. Trying to get a computer to do that or to perform handwriting recognition or drive a car or fly a plane on autopilot would require thousands of different "if", "else", "then" statements; and even then the program might not be able to cope with a novel environment. Instead of trying to teach the program, we let it learn from experience. We present it with data and have it find similarities in the data. For example, we show the program pictures of different kinds of trees and pictures that are not; then given a new picture it should be able to tell you whether it is a tree. After more learning the program could even learn to discriminate between different types of trees.

The mathematical background of why different learning algorithms can

learn on given input and then generalize on new data is extensive and well developed by Vapnik and Chervonenkis [41] based on the assumption that the given data (known values) is randomly selected so a model built from the data would be valid outside the data. Similar to polling, assumptions can be made about the entire population using a smaller sample size as a reference. It is with these assumptions that we use Support Vector Machine learning (SVM), Principal Component Analysis (PCA), Canonical Variate Analysis (CVA) and Parzen Windows (PW) to classify data to recognize hierarchy in networks.

We study unsupervised learning. The learning algorithm can depend on a specified set of patterns to be learned, $\xi^\mu$ and the neuronal state, $w_{ij}(t)$, i.e.,

$$\mathcal{L}(t) = \mathcal{L}(\xi^\mu, w_{ij}(t)). \tag{5.2}$$

For unsupervised learning, the learning alorithm does not depend on $t$ explicitly, as would be the case when a supervisor is changing the learning rule. For the hierarchy analysis of Section 6, we must convert the weight matrix to an adjacency matrix. That is, we convert a brain model to a graph. To do this, we need to know whether or not a connection exists. We use a fixed threshold,$\tau$, to convert the weights to connections. The adjacency matrix corresponding to threshold, $\tau$, becomes

$$A^\tau(i,j) = \begin{cases} 1, & |w_{ij}| > \tau, \\ 0, & \text{otherwise}, \end{cases} \tag{5.3}$$

where $A^\tau(i,j)$ is the adjacency matrix at threshold $\tau$. Thus, we only consider a link to exist if the weight is sufficiently large, that is, if the absolute value of the weight is greater than a given threshold, $\tau$. Thus the adjacency matrix for a given network depends on the threshold chosen.

We examine the evolution of graph measures as discussed in Section4.1.2 as the networks are trained to learn patterns. Specifically, we examine the changes in the graph measures as the weights change in response to learning patterns.

Hebbian learning is any learning algorithm that strengthens the connection of neurons that are on simultaneously and weakens those that are not, much like the potentiation and depression of neurons during memory formation. There are several ways to change the synaptic strengths between neurons, represented by a weight matrix, based on the activity of the neurons. A network of $N$ nodes is trained on a set of $p$ $N$-length binary vector patterns $\xi_i^\mu$, where $\mu = 1, 2, \ldots, p$ and $i = 1, \ldots, N$. Each entry of $\xi_i^\mu$ is either a 0 or a 1. The Hamming distance is defined to be the number of elements by which two vectors differ. The Hamming distance between two vectors $\xi_i^j$ and $\xi_i^k$ is calculated by

$$\sum_i \xi_i^j(1-\xi_i^j) + (1-\xi_i^k)\xi_i^j. \tag{5.4}$$

We restrict that each training pattern have at least a Hamming distance of $N/2$ to the other patterns. We also do not train with too many patterns to guarantee that the network is not over trained, a condition in which a network can no longer learn. We follow the rule of thumb that restricts

the number of patterns that a Hebbian network can learn to be 0.138 $N$[20].

The Hopfield Network couples many McCulloch-Pitts neurons. It arises from a desire that memories be content-addressable and insensitive to small change [20]. The most fundamental implementation of Hebbian learning automatically chooses the weight matrix to be a linear combination of the learned patterns.

$$w_{ij}^{\mathrm{H}ebb} = \frac{1}{n}\sum_{\mu=1}^{p} \xi_i^\mu \xi_j^\mu \tag{5.5}$$

The symmetry of the Hopfield network implies that all neurons have the same effect on other neurons as those neurons have on them. This may be the biggest flaw in the model because neuronal connections are not bidirectional. Specifically, neurons are connected from synapse to dendrite. Thus, the symmetry of the Hopfield model contradicts the assumption of biological plausibility. Nonetheless, due to the ubiquity of the Hebbian learning algorithm, it is included in the consideration of the emergence of hierarchy.

There are variations to Hebbian learning on the Hopfield network. For example we each weight, $w_{ij}$, can be incrementally changed in order to learn new patterns.

$$\mathcal{L}^{\mathrm{Hebb}} = \eta \xi_i^\mu \xi_j^\mu \tag{5.6}$$

where $\eta$ is the acquisition rate. A synapse weight between two neurons is increased if two neurons are on at the same time. The acquisition rate is usually taken to be small, so a single presentation changes the weights only slightly. In order to really learn a pattern, the pattern must be presented several times, if $\eta$ is small enough.

The Anti-Hebbian algorithm incrementally decreases the weight from $i$ to $j$ if node $i$ and node $j$ are on simultaneously.

$$\mathcal{L}^{\sim\mathrm{Hebb}} = -\lambda \xi_i^\mu \xi_j^\mu \tag{5.7}$$

The above formulas only allow for synapses to get stronger in magnitude. The Hebbian algorithm could cause the weights to go off to positive infinity and the Anti-Hebbian algorithm could cause the weights to go off to negative infinity. This forces the weight strengths to approach infinity over time. This is addressed in a variation of Hebbian learning called the covariance learning rule, which allows for weights to be weakened:

$$\mathcal{L}^{\mathrm{cov}} = \eta(\xi_i^\mu - \overline{x}_i)(\xi_j^\mu - \overline{x}_j) \tag{5.8}$$

where $\overline{x}_i = \sum_{i=1}^{p} \xi_i^\mu / p$. The covariance rule decreases the weight if one neuron is on but not the other not. Thus if the neurons are not firing together, their weight strength decreases. A flaw of this algorithm is that the weight strength goes up if both are off at the same time. This algorithm above still has the problem that the weight strength might approach infinity. In a more realistic algorithm, the weights have to decrease so that none of them will overpower the rest The brain has mechanisms that naturally decrease weights corresponding to unused synapses throughout the brain, so that no individual synapse is much stronger than the rest. One

solution for this problem is to use normalization. This can be done by setting an upper bound on the maximum value of $w_{ij}$ and not allowing any of the weights go over that predetermined maximum value.

$$w_{ij}(t+1) = min(w_{ij}(t) + \mathcal{L}, 1) \tag{5.9}$$

if $w_{ij}(t) > 0$, and

$$w_{ij}(t+1) = max(w_{ij}(t) + \mathcal{L}, -1) \tag{5.10}$$

if $w_{ij}(t) < 0$.

However, over time, some or all of the weights may still approach the maximum value resulting in the algorithm effectively no longer learning. Two other ways to normalize include using multiplicative or subtractive scaling. We will not consider these alternatives. Instead we will stop training early so that we do not exceed reasonable weight values [20].

Another learning algorithm is Oja's rule [20, 47, 33] which allows the network to forget things it does not see very often.

$$\mathcal{L}^{\text{Oja}} = \lambda \xi_i^\mu (\xi_j^\mu - w_{ij}(t)) \tag{5.11}$$

The weights between two nodes are lowered when one of them, say $j$, is on but the other one, say $i$, is not. The Oja algorithm allows the weights to decrease but if they do, the reduction is slowed as the weight gets smaller. If neuron $j$ is on but neuron $i$ is not, then $\mathcal{L}^{\text{Oja}}$ is negative.

In the Rate algorithm, weights at time $t$, $w_{ij}(t)$ equal the change in weight $\mathcal{L}^{\text{Oja}}$ of the Oja algorithm

$$w_{ij}(t+1) = \mathcal{L}^{\text{Oja}}(t). \tag{5.12}$$

The Bilinear algorithm increases the weight if both neurons are on at the same time, but decreases if only one of them fires,

$$\mathcal{L}^{\text{bil}}(t) = \lambda \xi_i^\mu \xi_j^\mu - \eta \xi_i^\mu - \alpha \xi_j^\mu. \tag{5.13}$$

Constant Depression is an algorithm that increases the weight in the same way the Hebbian algorithm

$$\mathcal{L}^{\text{CD}} = \eta \xi_i^\mu \xi_j^\mu - \lambda w_{ij}(t). \tag{5.14}$$

We also train using different initial networks to learn the patterns. Each network is represented by a vector with the different measures in each entry of the vector. The different networks are ER (random), Waxman, Small world, Ring, and Rectangular. We train with networks of size 450 and 750. Results can be found in Section 6.

# 6 Results from Learning Algorithms

In this Section we present the results of the study of the emergence of hierarchy in Hopfield networks. We present the graph types and sizes we use, then we show the feature vectors from the graphs after they have been transformed by Principal Component Analysis (PCA) and Canonical Variate Analysis (CVA). Then using the transformed data we show

how Support Vector Machine (SVM) and Parzen Windows (PW) classify the data. After that we discuss each of the seven different training algorithms, followed by comparisons between results using thresholds for converting networks to graphs, and interesting results. We present examples of training that seem to increase the hierarchy and those that do not. We compare how SVM and Parzen windows agree or disagree about the classification of graphs on a case by case basis.

## 6.1 Results for Initial Networks

We created ER graphs,Waxman graphs, Small world graphs, Null graphs, Complete graphs, Trees graphs, Rings graphs, and Scale Free graph at 450 and 750 neurons. The ER, Waxman, and Small World graphs were generated from three different expected average degree distributions. Different m-ary Trees were generated using different heights. Figure 4.6 shows the data projected onto the first three eigenvectors by PCA for 450 neurons and Figure 19 shows the data projected by PCA for 750 neurons.

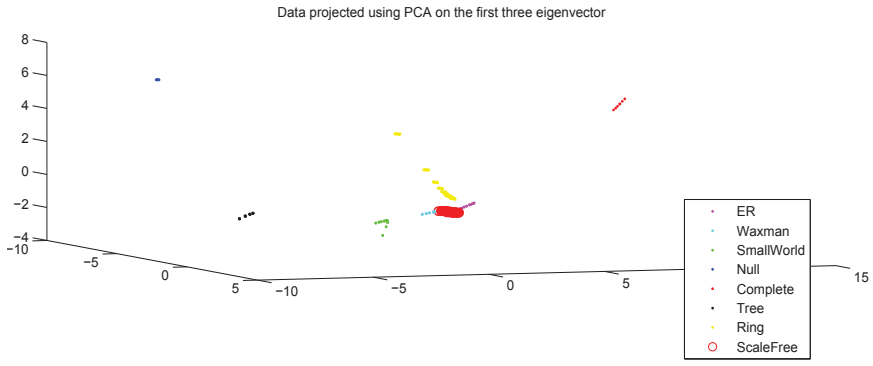



Figure 6.1: Data of size 450 projected with PCA

In Figures 6.1 and 6.2 the ER graphs are represented by the magenta points, the Waxman graphs by the cyan points, the Small World graphs by the green points, the Trees graphs by the black points, the Null graphs by the blue points, the Complete graphs by the red points, the Ring graphs by the yellow points and the Scale Free graphs by the red circles. We can see that the types of graphs are well separated by using PCA with the set of graph measures adapted.

Figure 6.3 shows the data projected onto the first three eigenvectors by CVA for 450 neurons and Figure 6.4 shows the data projected by CVA space for 750 neurons. The types of graphs are better separated by CVA

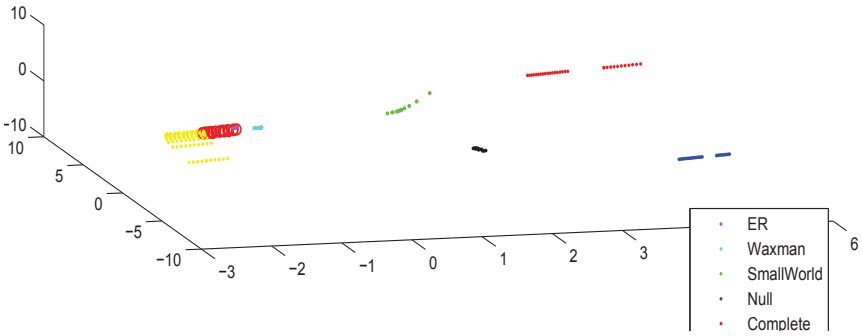



Figure 6.2: Data of size 750 projected with PCA

than by PCA.

In order to use the SVM algorithm we only consider Tree graphs and Random graphs. Figure 22 shows the data from the Trees and Random Graphs transformed by PCA then separated by SVM. If the data transformed by CVA had been used instead and then separated by SVM a similar graph would have resulted.

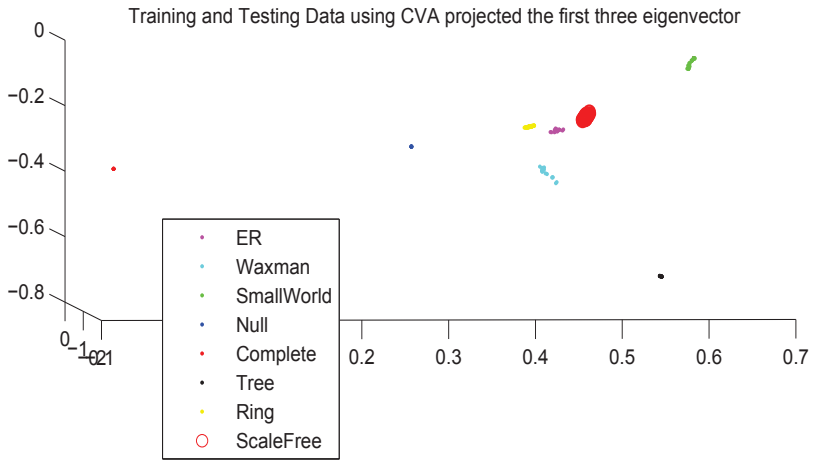



Figure 6.3: Data of size 450 projected with CVA

### 6.1.1 Results from Training

Next the networks are trained, using the original Hebbian algorithm. The initial point is the feature vector of a Ring graph. The graph is trained for 53 iterations. In Figure 6.6 we can see how the points are classified by the

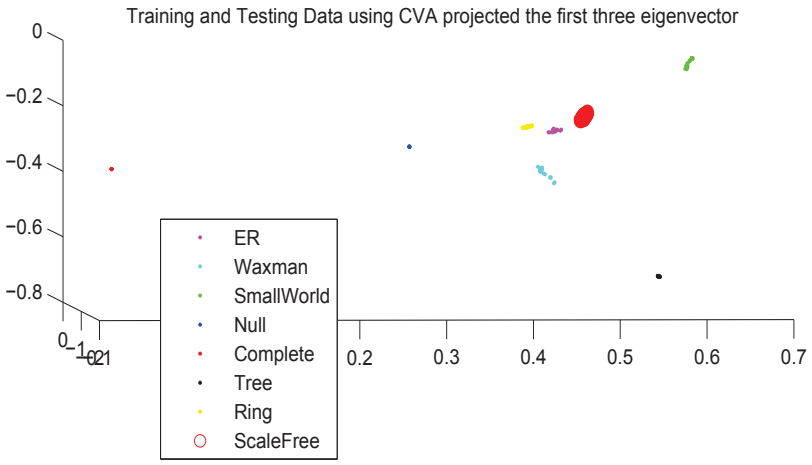



Figure 6.4: Data of size 750 projected with CVA

Parzen windows (PW) as the networks are trained for the 53 iterations. For the Hebbian algorithm applied to a random network, the brain model does not get more hierarchical as learning occurs. The graph is assigned to the group corresponding to the maximum of the probabilities after each learning step. The probabilities of a graph being a tree (cyan line) or not (magenta line) at each iteration of the training are shown in Figure 6.7. There is basically no change in the lines for any of the thresholds.

We calculate the distance of the Hebbian points to the hyperplane separating the graph types as they learn. Figure 6.8(a) shows the distances from the hyperplane for the PCA data after each training. Figure 6.8(b) shows the distance from the hyperplane for the CVA data after each training. A negative distance implies that the data point lies on the side of the random graphs. Before the network is trained the point is closer to the hyperplane, i.e, it is less random, but not a tree. After the first training it becomes more random but does not change after that.

### 6.1.2 More Interesting Result

In the Previous section an increase in hierarchy as a result of Hebbian learning was not observed. These results are similar for other learning algorithms; in particular, neither the Anti-Hebbian algorithm nor the Oja algorithm showed an increase in hierarchy. This was supported by the PCA and CVA data using both PW and SVM. We do not present that data. Instead we present data where changes were observed.

Using the Bilinear algorithm led to interesting behavior. Figure 6.9 shows that the evolution of different probability distributions during learning as a function of the threshold value used to convert the network to graphs. For low threshold values, the probability that the graphs become

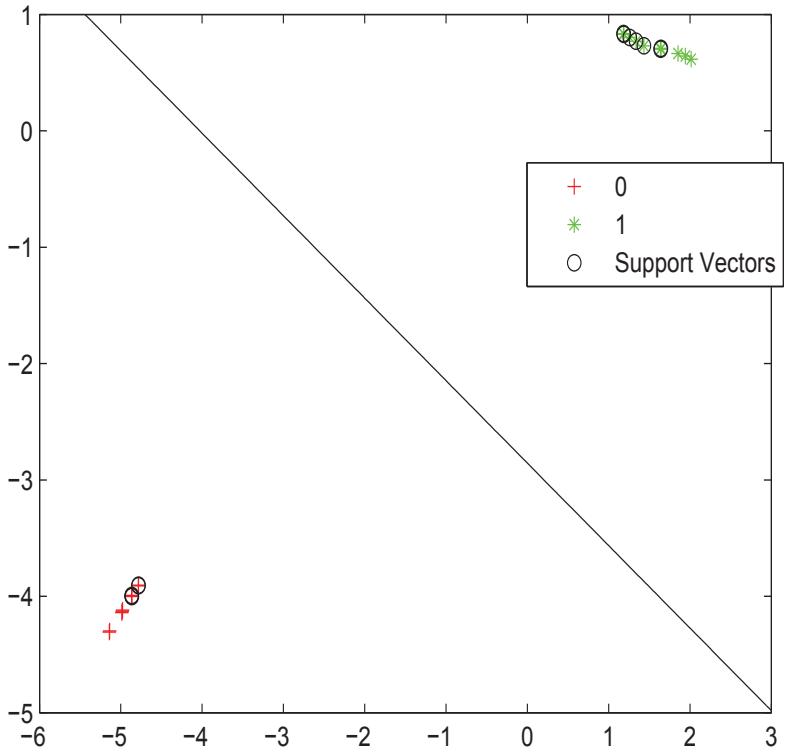



Figure 6.5: PCA data separated by SVM

complete are higher. However as the threshold increases the probability that the graph becomes more tree-like increases. Of particular interest are the behaviors for threshold values of 0.8 and 0.9. The results are supported by the CVA data shown in Figure 6.10 and by the Tree or Not distributions in Figure 6.11. The SVM distances for these two thresholds are shown in Figures 29(a) and 29(b). We point out that the value of the threshold parameter played a major role in deciding whether a graph was becoming more hierarchical. For example if a threshold value of 0 or 1 were used, no change would have been observed. With a threshold of 0 the method results in a Complete graph and with a threshold of 1 the method produces a Null graph. We also trained with the Bilinear algorithm for 30 more iterations to obtain the results in Figure 6.13. Each training number represents 10 training iterations. We see that eventually more training would lead to a decrease in the probability of being a tree for the threshold values of 0.8 and 0.9 but would also decrease the probability of remaining Complete graphs for the lower threshold values. Next we studied the Depressed algorithm. Similar to the results of the Bilinear algorithm, an increase in hierarchy was achieved. In Figure 6.14 and Figure 6.15 we can see that the probability that the network is similar

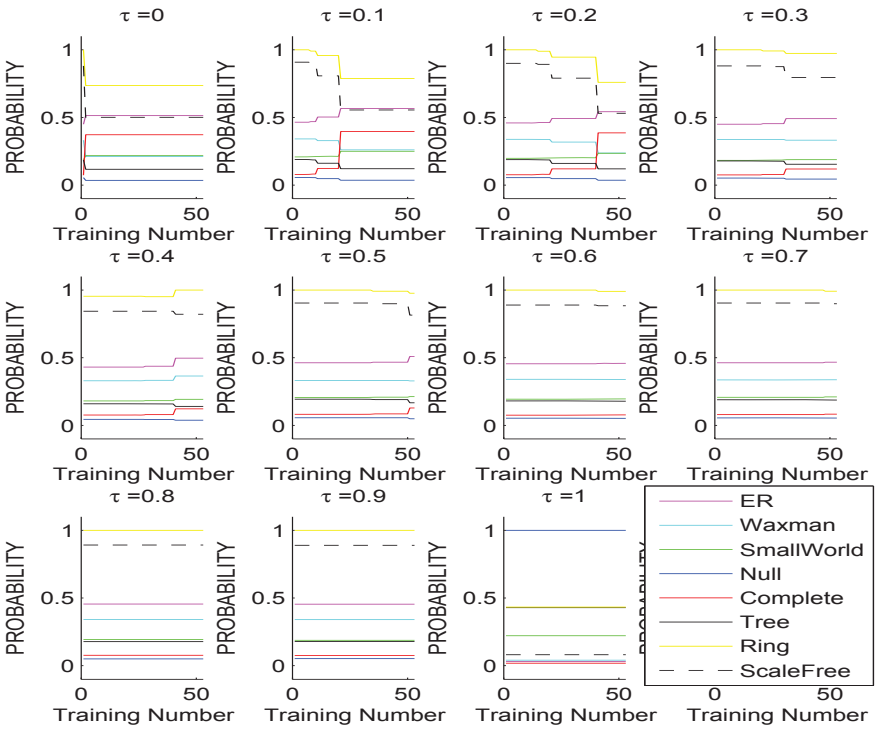



Figure 6.6: Probability distributions for Hebbian algorithm with PCA data

to a tree increases. The analysis for this graph was also strongly influenced by the threshold value. , In particular in Figures 6.16(a), (b), (c) with threshold values of 0.5, 0.7, and 0.9, respectively, increasing threshold values led to increasing probabilities of being a tree using the SVM algorithm. Another interesting example was the graph trained on the Rate algorithm in Figure 6.17 where the probability that the graph was a tree increases at first but then oscillated back and forth. Figure 6.18 shows that SVM supports the same result where the distance from the hyperplane goes up and down. Finally, using the Covariance algorithm the graphs were becoming complete graphs as shown in Figure 6.19 for every threshold value.

## 6.2 Functionality Changes During Learning

In this section we apply the approach of using two-event chains to describe changes in learned brains. We interrogate a set of brains that start from a rectangular array of neurons connected with various connection rules to

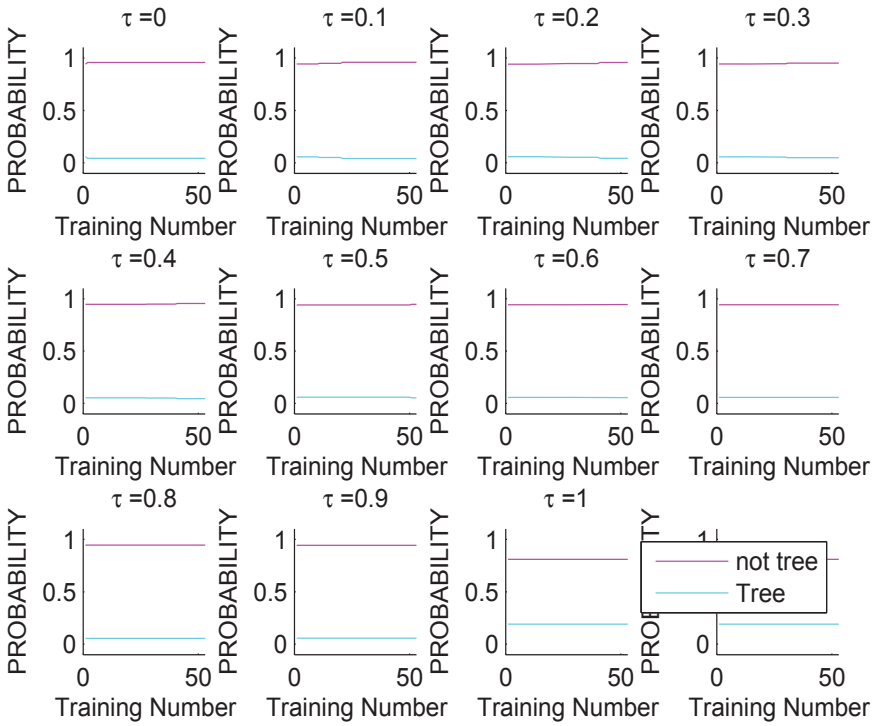


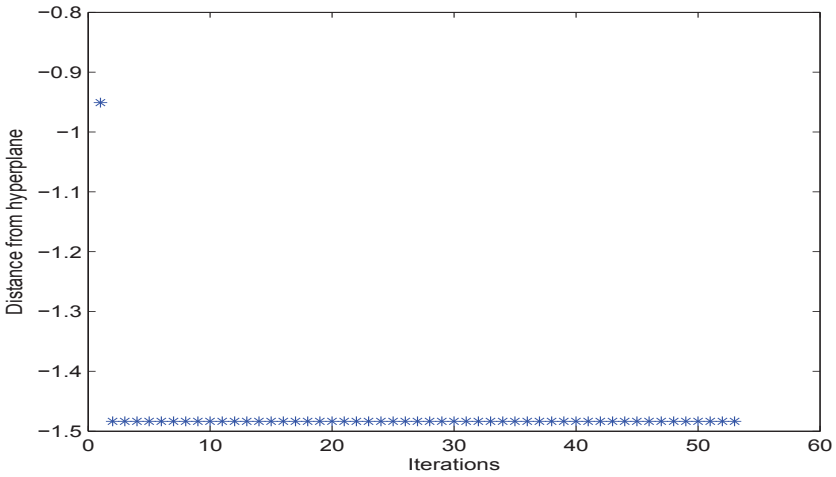

Figure 6.7: Tree or not a tree

a distance 2. We then train these brains using Sanger's rule for a fairly large number of iterations. In Section 4.4.2 we argued that an unlearned brain had normally distributed behavior ratios over a small set of weight values. After we subject the model to learning, we confirm that a learned model also has this normally distributed behavior.

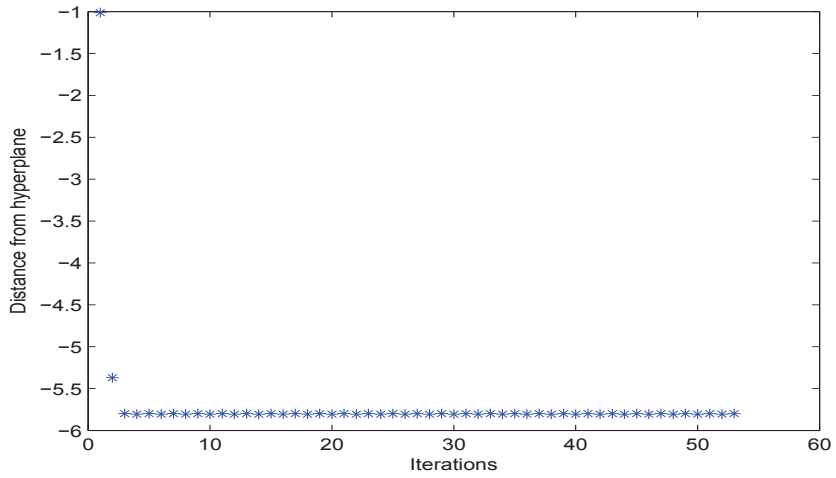
Figure 6.22 shows histograms of two-event frequency ratios for two very different brains, one the initial weight matrix with connection distance two and a uniform distribution of weights, and the other after learning using Sanger's rule for 7000 learning steps. Both histograms have the same number of points allowing for easier comparison. Here we see that there is a clear distinction between the two. It becomes apparent that we are able to distinguish a learned brain model from an unlearned one using the distribution of two-event frequency ratios.

We can use the Hellinger distance to compare a learned plot like Figure 6.21(b) to an unlearned plot.

As we see in Figure 6.23 there is considerable difference between the learned model and the unlearned model. If we compare this to the Hellinger




(a) PCA data Distance from Hyperplane (-) Random, (+) Tree




(b) CVA data Distance from Hyperplane (-) Random, (+) Tree

Figure 6.8: Distances from hyperplane using SVM for the Hebbian algorithm

plots in Figure 4.25 and Figure 4.26, we see an appreciable difference. Again, there is a larger difference towards the higher values. There are two reasons for this, first we notice for the distribution of weights in a learned model as seen in the red curve in Figure 6.22, there are fewer weight values in this higher range. This means that the data will be less accurate and fluctuate more. Second, since there is less variation the behavior for each weight value, the range of behaviors has increased. Com-

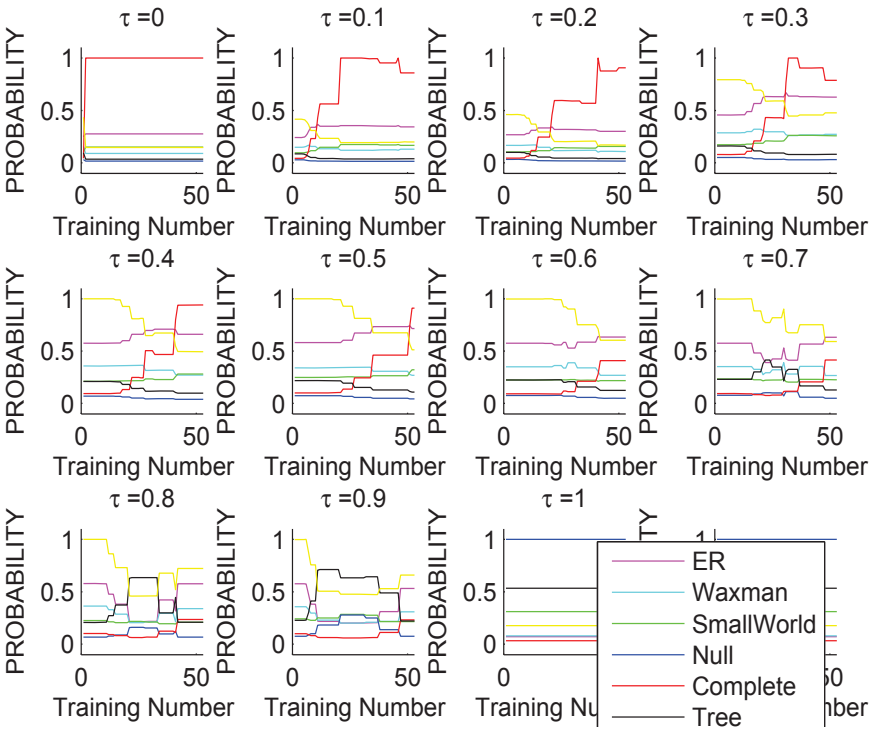



Figure 6.9: Probability distributions for bilinear algorithm with PCA data

paring Figure 6.21 to Figure 4.21 shows the larger ratio value for larger weights. Since the variance in the learned plot is also small, the possible ratio values for a large weight in the learned case will be considerably different from those in the unlearned case.

### 6.2.1 Comparing Learned Brains to Random Networks

Figure 6.24 shows the result of taking a locally connected brain with a using this method in estimating the weight values of a learned model. This figure shows a distribution of the magnitude difference between the two weights. The estimated weight values differ, on average, by 0.0436 from the actual weight values. Figure 6.21 shows the distribution of weight values in a learned brain model. In the learned brain model, new connections are formed and we see from Figure 6.21 that the magnitude of the weight values in the learned model are smaller on average than those in our unlearned uniformly distributed case. Even though this procedure makes it easier to generate a weight value from a two-event frequency

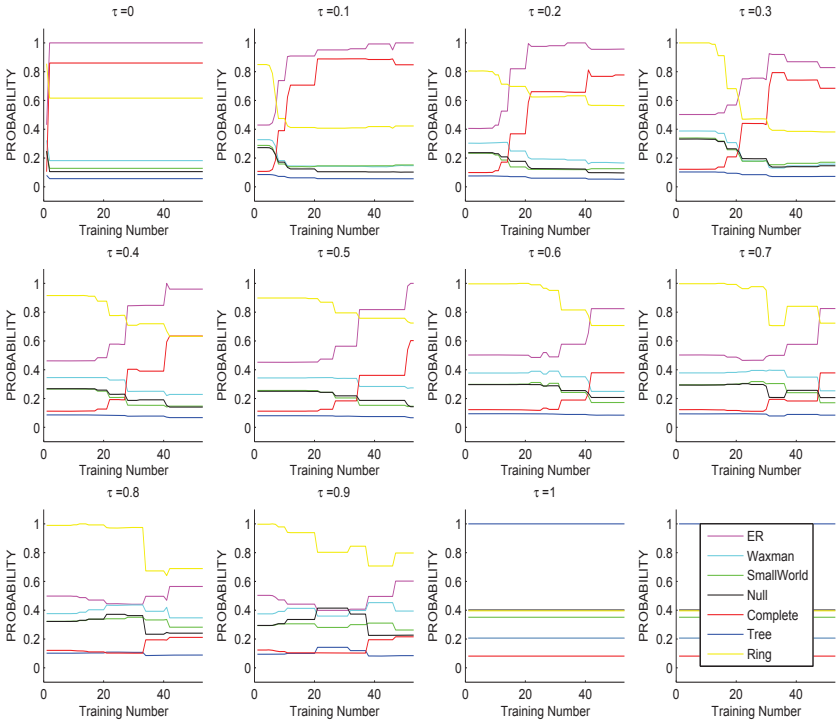



Figure 6.10: Probability distributions for bilinear algorithm with CVA data

value, there is now a smaller range in which most learned weight values exist. This means that although the model is better suited for generating weight values from frequency values, the new weight distribution does not seem to be a correct model. Therefore, we still find that this approach in approximating weights on behavior alone can not be done effectively by these means. In addition to this, we are only able to make estimations on weight values when there is behavioral information available. In many cases, for smaller weight values, there are no two event-chain results. In our learned model, 48.33% of the connections did not have an associated behavior. Longer run times would reduce this by a small amount, but when coupled with the mediocre inferences our model can make with the data we do have, it is not worth pursuing.

The convergence of the learned brain weight matrices prompted us to examine the learning algorithms. All of the algorithms used herein (all unsupervised learning algorithms with patterns chosen at random) led to

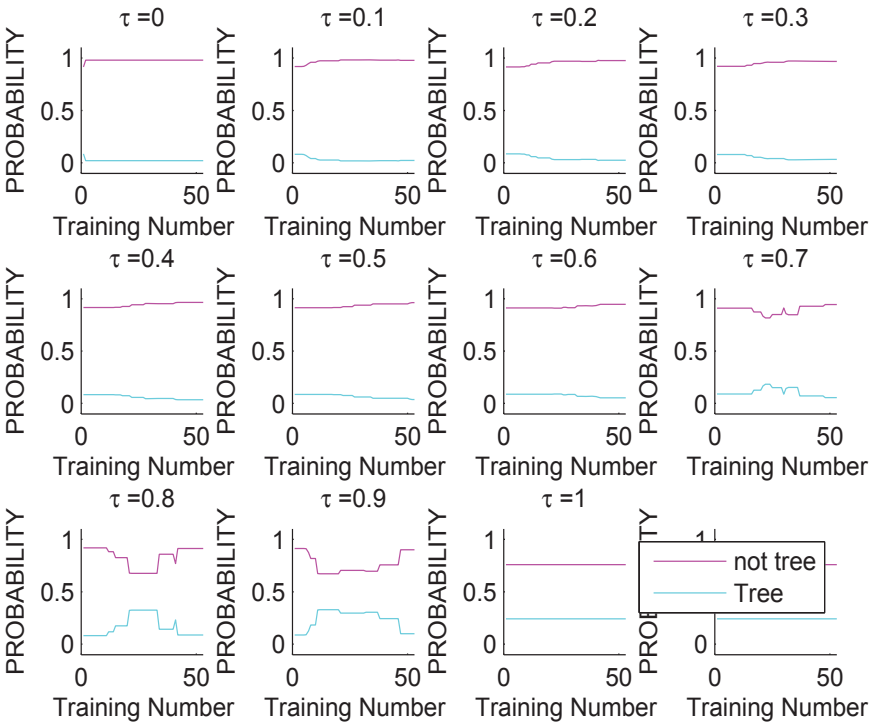
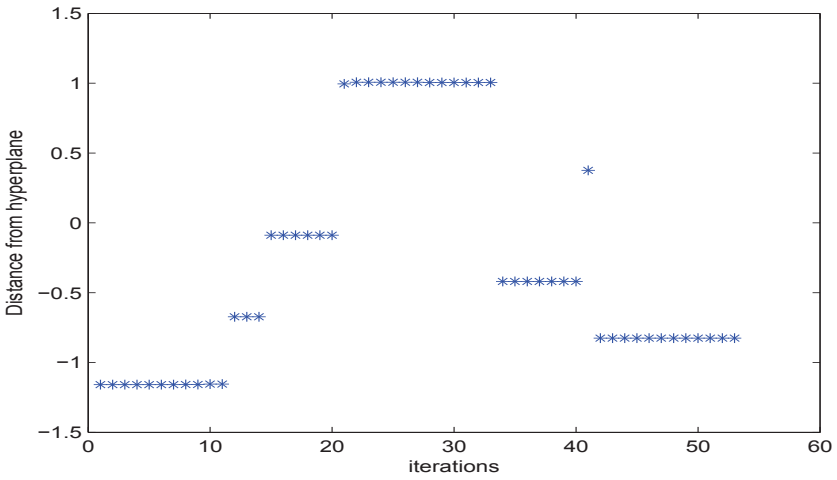



Figure 6.11: Tree or not for bilinear algorithm

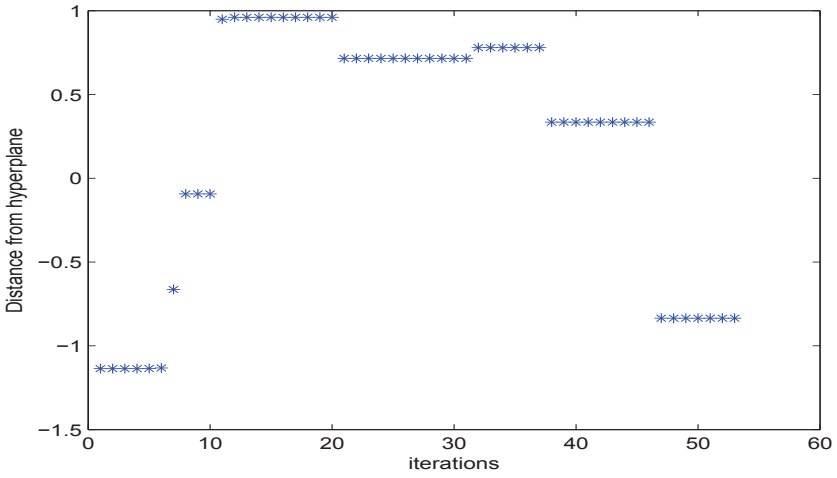
learned weight matrices of the form

$$w_{ij}^{Hebb} = \sum_{\mu=1}^{p} \lambda^{\mu} \tilde{\xi}_i^{\mu} \tilde{\xi}_j^{\mu} \tag{6.1}$$

where $\tilde{\xi}^{\mu}$, $\mu = 1, \ldots, p$ are normalized pattern vectors, which turn out to be eigenvectors of the converged weight matrix, with eigenvalue $\lambda^{\mu}$. While that is a "structure" for the weight matrices, it is not hierarchical.




(a) Threshold Value of 0.8




(b) Threshold Value of 0.9

Figure 6.12: Distance from hyperplane using SVM for the bilinear algorithm

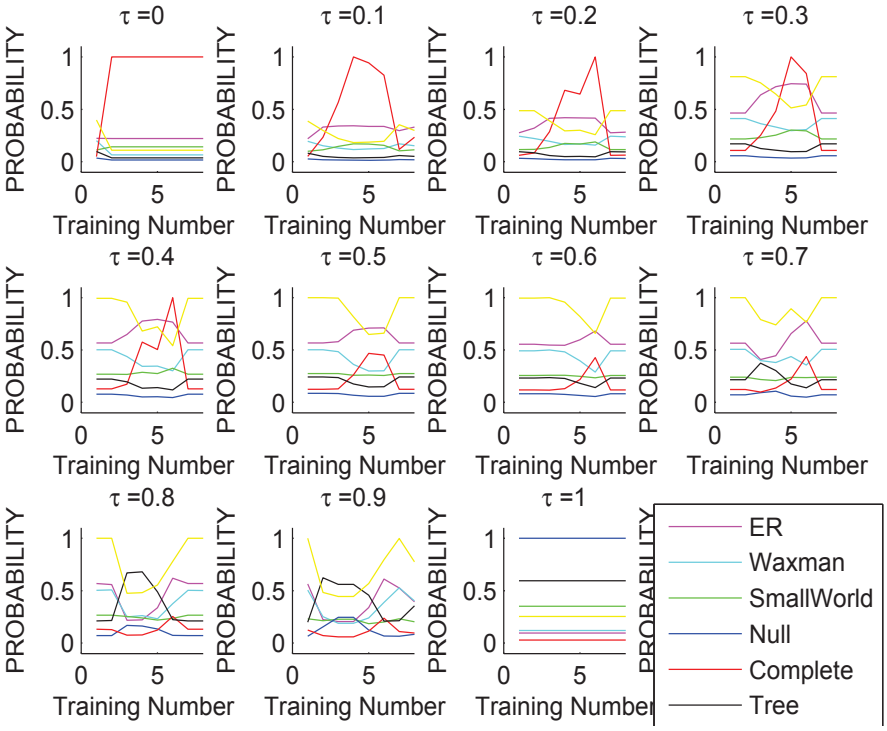



Figure 6.13: Bilinear algorithm trained longer

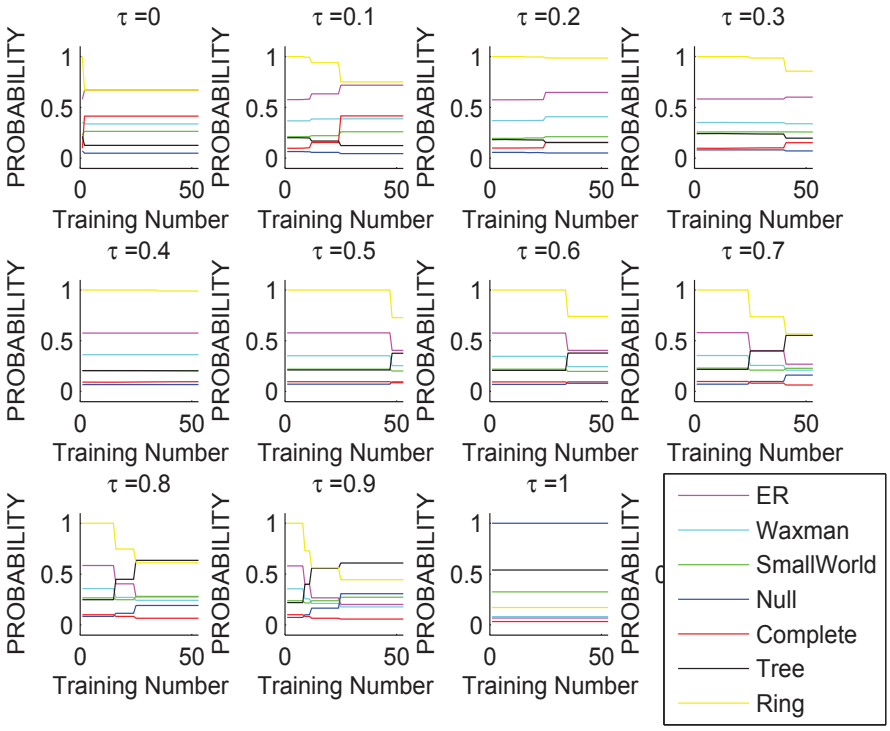



Figure 6.14: Probability distributions for depressed algorithm

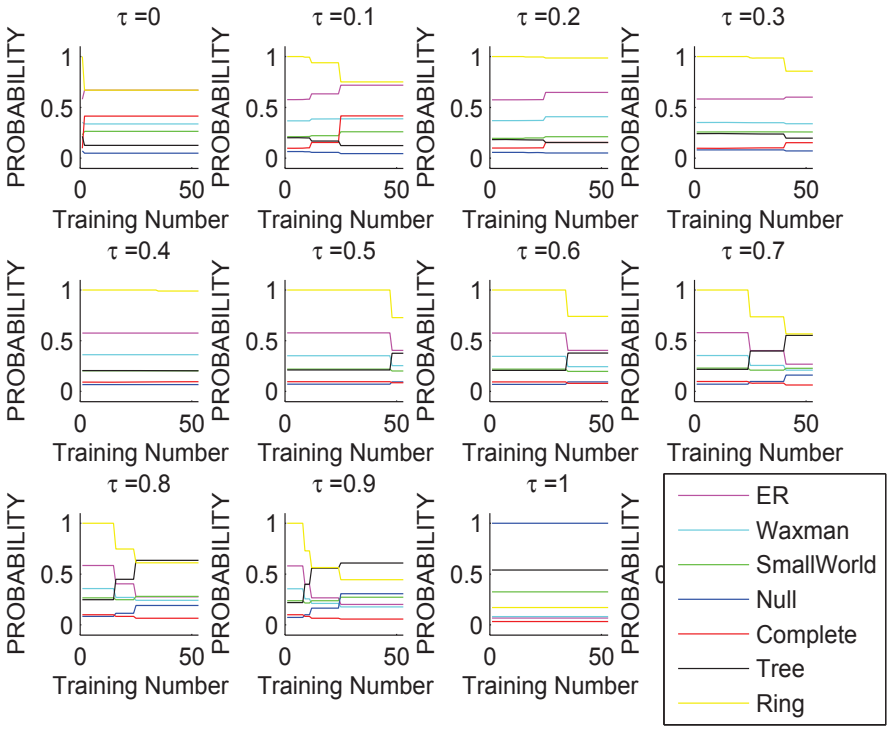
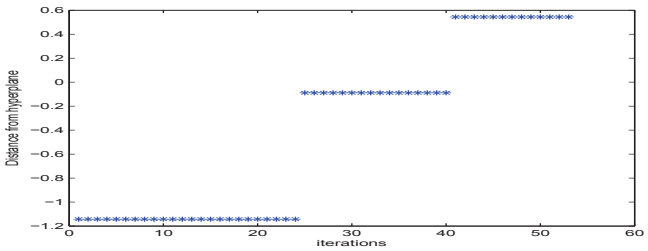
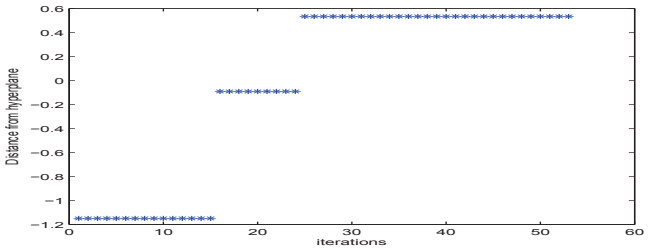



Figure 6.15: Tree or not for depressed algorithm




(a) Threshold Value of .5




(b) Threshold Value of .7

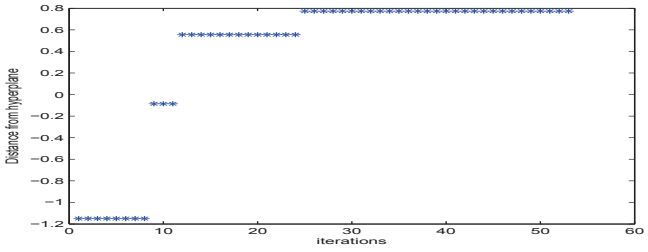



Figure 6.16: Distances from hyperplane using SVM for the for depressed algorithm

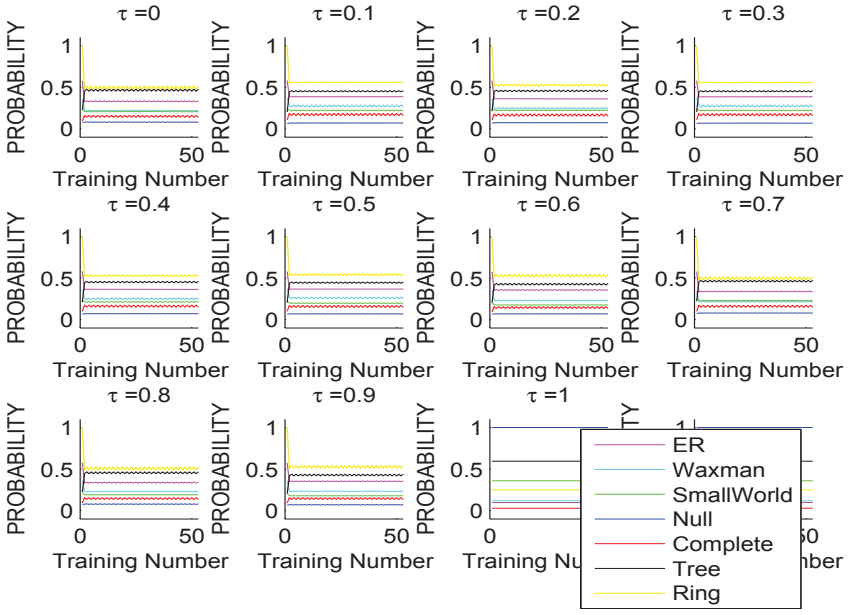



Figure 6.17: Probability distributions for rate algorithm

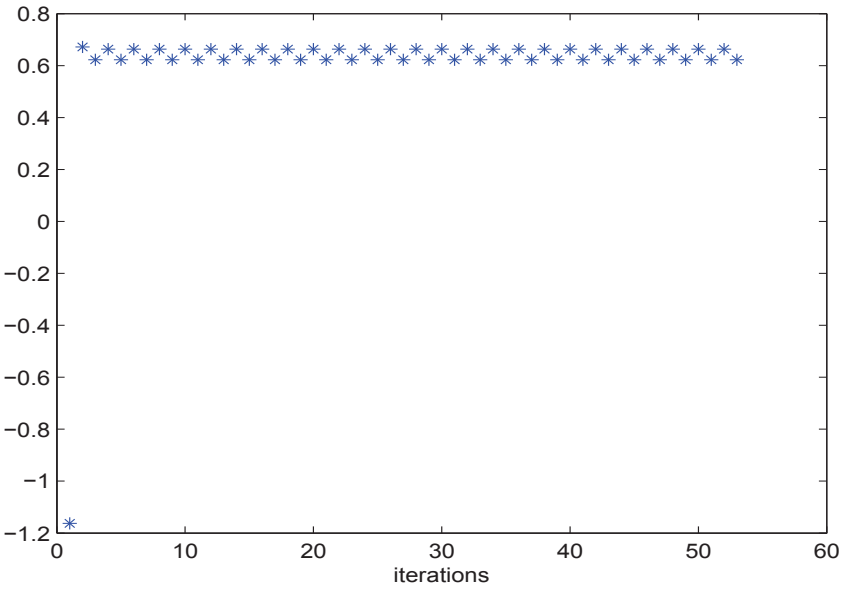



Figure 6.18: Distances from hyperplane using SVM for the threshold value of 0.1 for the rate algorithm

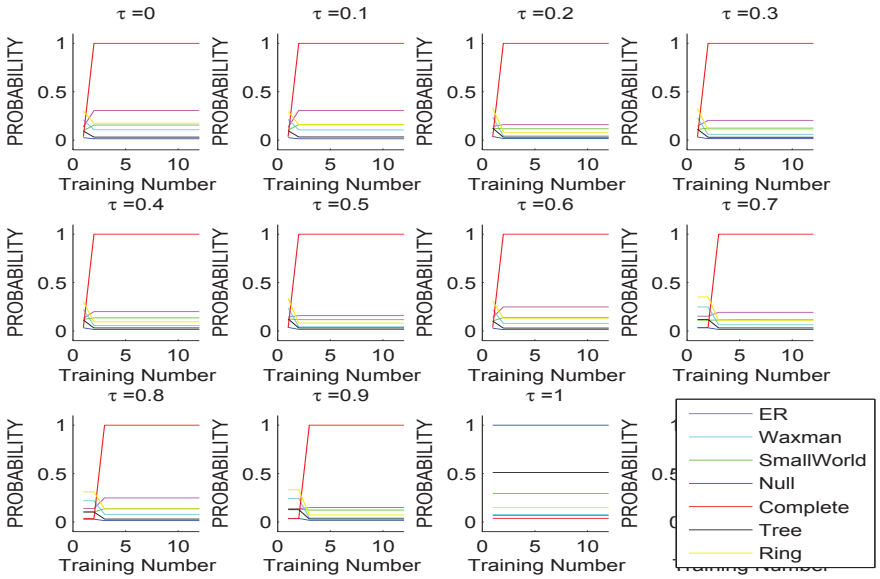



Figure 6.19: Probability distributions for covariance algorithm

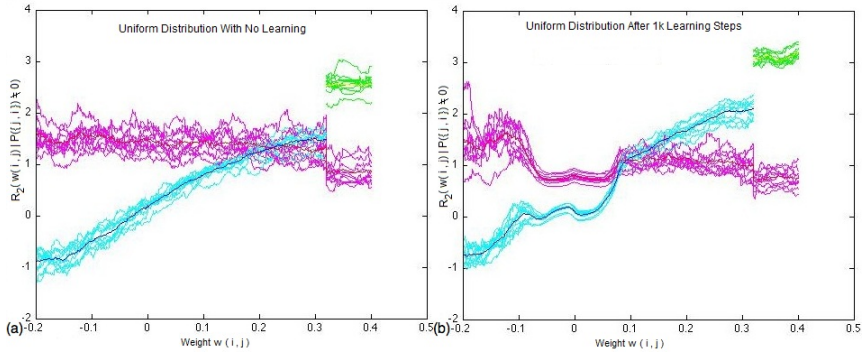
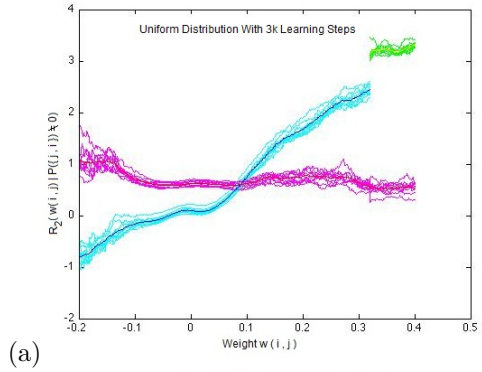
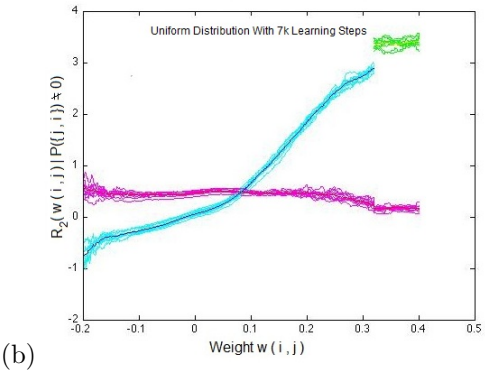



Figure 6.20: (a) The comparison plot for weights distributed uniformly over $[-0.2, 0.4]$ with no learning. (b) The comparison plot for weights after learning by Sanger's rule for 1000 time steps




(a)




(b)

Figure 6.21: (a) The comparison plot for weights after learning by Sanger's rule for 3000 times steps. (b) The comparison plot for weights after learning by Sanger's rule for 7000 time steps

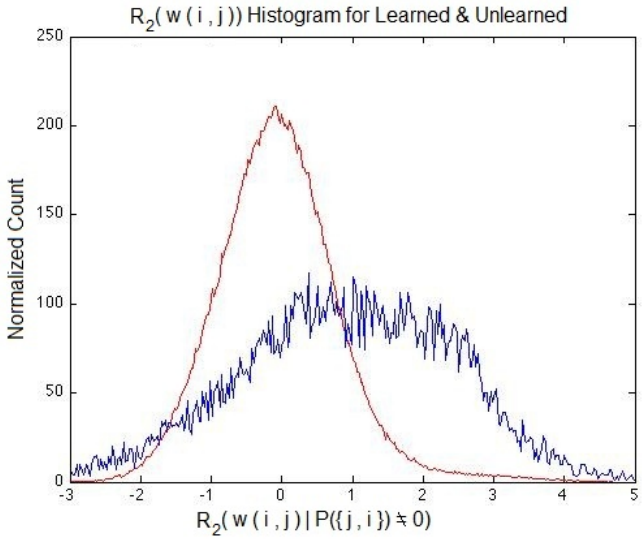



Figure 6.22: The behavior only histogram plot for learned and unlearned brain models. The red plot is generated by learned ratio values, and the blue plot is generate by unlearned ratio values

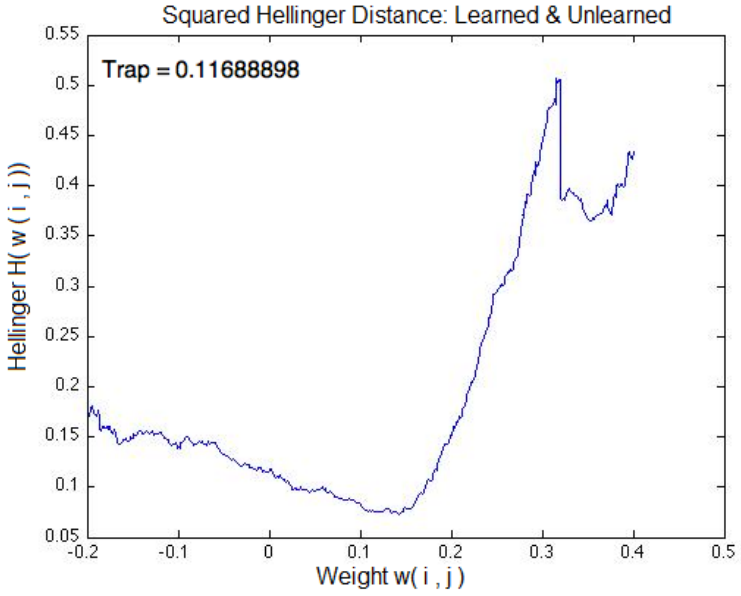



Figure 6.23: Hellinger distance comparing an unlearned 169 neuron brain model, to a 169 neuron learned model

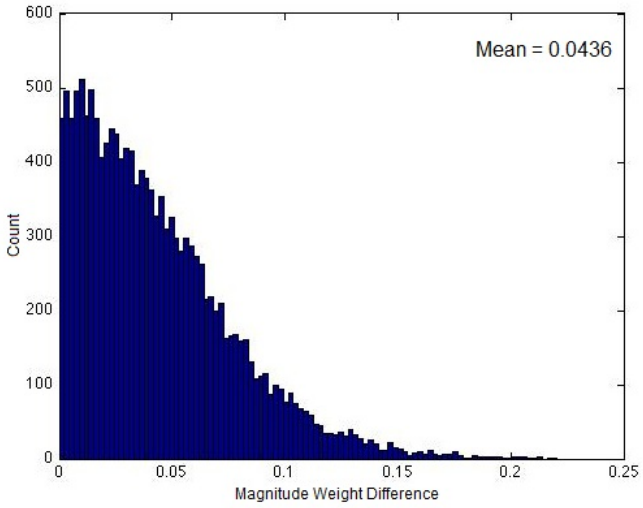



Figure 6.24: The histogram of differences between the estimated weight values of a learned brain model and the actual weight values

# 7 Conclusion and Future Directions

## 7.1 Summary

The purpose of this work was to determine if structures (including hierarchy) emerged from the use of Hebbian-like learning algorithms on a Hopfield network. This would imply that the brain may arrange connections between its neurons hierarchically as a result of how it learns. That is, the information encoded in the synapses is arranged hierarchically, as a result of how they learn. Typically networks that are used to model the brain assume a hierarchical structure from the onset and are trained with the nodes in hierarchy [15].

This work started with the idea of determining if hierarchy emerged due to learning in brain models. The paradigm brain for this work was the Hopfield network, which learns by changing the weights of synaptic connections between neurons. The goal of the neuron is to respond appropriately to the combination of the two inputs (inhibitory and excitatory). Graph measures were defined to quantify hierarchy. Two event chains were also used to characterize brain models in terms of their actual function.

In order to describy hierarchy in networks, we first studied graphs. We took several graph types, including Small World, Waxman and Tree graphs, and calculated various measures on each graph using their adjacency matrices. Each graph was then represented by an $n$-dimensional data point, i.e, a feature vector, where each dimension was a different measure calculated for the graph. Using Principal Component Analysis (PCA) and Canonical Variate Analysis (CVA) transformed the high dimensional data onto a lower-dimensional space. Using this transformed data the probability, $P(T|X)$ was calculated. This quantity is the probability of belonging to any of the graph types, say $T$ , given any point, $X$, in that lower dimensional space. The brain model starts from a weight matrix with entries other than 0 or 1. However, to calculate the graph types the weight matrix must be transformed into an adjacency matrix, with the entries of 0 and 1. This is accomplished as a function of threshold $\tau$. Any weight exceeding the threshold is considered a connection. Specifically, we used

$$A^{\tau}(i,j) = H(|w_{i,j} - \tau), \tag{7.1}$$

to generate the adjacency matrix from the weight matrix.

After the graph measures were evaluated using PCA and CVA methods, a measure of hierarchy was determined by using trees as the prototype of hierarchy. The differences in the algorithms, though sometimes subtle, lead to very different results. The algorithms change the connection weights based on the interaction of any two neurons; the essence of the algorithms being whether both neurons are on, both neurons are off, or just one neuron is on. Different learning algorithms are examined and each react differently to the same input and the same initial weight matrix.

We did not assume a priori that the network had a hierarchical structure. The emergent structure was governed by a strengthening and weak-

ening of synaptic weights based on repeated exposure to a set of patterns to be learned. Most of the algorithms did not lead to an increase in hierarchy. However, an increase in hierarchy was observed in the Bilinear and Depressed Algorithms, and to a lesser extent in the Rate algorithm. These observations were dependent on the range of threshold values used. In fact, as the threshold value was increased, an increase in hierarchy was observed, supported by PW and SVM for both PCA and CVA data sets. This was somewhat expected because increasing the threshold effectively removes a number of weaker links from a graph altering the structure. However the increase in hierarchy was not just a result of having fewer links, or else the results of other algorithms would have also shown an increase in hierarchy. The increase in hierarchy was due to changes in the way the connections between neurons were evolving during learning.

For the functional work, The event chain data evaluates the behavior of a neuron using a method similar to frequency analysis. We used the information gathered from two-event chains and one-event chains to relate the behavior to the strength of the connection between two neurons. The data showed a correlation between connection weight values between pairs of neuronal columns and the frequency of the two-event chain of the sequential firing of that pair of neuronal columns.

We attempted to use the behavioral data alone to recreate the weight matrix. In the unlearned model we concluded that the distribution of weight values the corresponding behavior has too large a variance to accurately approximated the weights. For the learned model we found the learning process changed the distribution of the weights to be primarily a normal distribution, but not localized, as we believe that brain models should be.

To extract information from the firing frequency versus connection weight data from various brain models, we used the Hellinger distance between two probability density distributions as a metric. Noting that Hellinger distance is easy to calculate for two normal distributions, we used the Kolmogorov-Smirnov test to confirm that for a small range of weight values the corresponding behavioral data could be represented as a normal distribution with mean and standard deviation which varies with connection weight. We found that brain behavior led to normally distributed firing frequency ratios for all brain models studied, this regardless of the distribution of the weights or whether or not the brain model has undergone a learning process.

From these two tools alone we quantified the behavioral difference between models of varying sizes. We determined the effect of increasing or decreasing the number of neuronal columns under consideration. This gave us an understanding of the relation between increasing the model size and behavior. Under similar methodology, we determined the impact of damaged neuronal columns on behavior and how this relates to the boundary effects seen when changing our model size.

By comparing two models that are identical we set a baseline difference that can be attributed to randomness and sampling errors. From this point we changed various properties of the model and measured any significant changes. We found that we could see considerable differences when we used a normally distributed set of weights that favored weights

with smaller magnitudes.

Finally we implemented multiple learning algorithms and determined the most suitable to analyze further. Beginning with a fairly simple Hebbian or Anti-Hebbian learning algorithm, we determined that the weights in the model evolve to extremely large magnitudes. We implemented Sanger's learning algorithm on the weight matrices and let the connection distance and weights be changed based entirely on the algorithm. This was done in steps to allow for analysis of the learning progression. The new connections and their corresponding strengths were then run through the same Kolmogorov Smirnov test to ensure it could be compared fairly with our unlearned models.

The distinct behavior of the learned model allowed us to use only behavioral data to distinguish it from an unlearned model. Using the Kolmogorov Smirnov test we were able to confirm that, on behavior alone, we can distinguish a learned brain from an unlearned brain. The only shortcoming to this method is that the result tells us whether or not they are similar within a margin of error, but not a quantifiable difference like with the Hellinger distance.

By examining the distribution of weights in the learned brain model we reproduced the learned model using weights drawn at random from the same distribution. We could confirm that behavior remained the same and that behavior is determined almost entirely by these two parameters.

## 7.2 Lessons Learned

After studying the functional method, we concluded that all Hebbian-like learning methods led to brains that were not structured. Hierarchy did not emerge from Hebbian learning. The apparent reason for this is that Hebbian learning methods lead to networks with a symmetric connection weight matrix. This is non-biological, and apparently, precludes structures that "flow" from lower to higher (or *vice versa*). The issue is not strengthening connections that were used more frequently; instead, it appeares to be that the methods imposed patterns, $\xi^\mu$, $\mu = 1, \ldots, p$, and the patterns force the symmetry of the weight matrix. The different methods merely alter the rate of approach to the symmetric matrix, or keep it from growing large.

We have concluded that a brain model must have a fixed set of input neurons and a fixed set of output neurons, and that the learning proceeds from applying an input pattern to the input neurons, comparing the output from the output neurons over several time steps, and changing the weights used to "reward" or "punish" for degree of correctness in the output. This is, in essence, a supervised learning method. It requires a "parent" to decide if the output answer was better or worse.

In a corrollary to this idea, the PCA idea of expanding a matrix in its eigenvectors might be replaced by expanding a matrix in terms of its trees, since trees are a fundamental unit in hierarchy.

# References


[1] Hillel Adesnik, Guangnan Li Matthew J. During, Samuel J. Pleasure, and Roger A. Nicoll. NMDA receptors inhibit synapse unsilencing during brain development. Proc Natl Acad Sci U S A 105:5597-5602. 2008.

[2] Veronica A. Alvarez and Bernardo L. Sabatini, Anatomical and Physiological Plasticity of Dendritic Spines, Annual Review of Neuroscience, Vol. 30: 79-97

[3] Bear, Mark F., Connors, Barry W. Paradiso, Michael A. Neuroscience Exploring the Brain. LippincottWilliams & Wilkins. 2001.

[4] Ballard, Dana H. An Introduction to Natural Computation. Cambridge: The MIT Press. 1997.

[5] Albert-Laszlo Barabasi and Reka Albert. Emergence of Scaling in Random Networks. Science Vol 286 15. October 1999

[6] Barabasi, Albert-Laszlo. Linked: How Everything Is Connected to Everything Else and What It Means. Reissue ed. New York: Plume. 2003.

[7] Kathleen M. Carley, Michael J. Prietula, eds. Computational organization theory. Hillsdale, N.J. : Lawrence Erlbaum Associates. 1994.

[8] L. da Fontoura Costa, Rodrigues, F. A.; Travieso, G.; Boas, P. R. Villas. "Characterization of complex networks: A survey of measurements" Advances in Physics 56.1 (2007). 02 Jul. 2010

[9] L. da Fontoura Costa, G. Traviesoa, and C.A. Ruggiero. Complex grid computing. Instituto de Fisica de Sao Carlos, Universidade de Sao Paulo, Caixa Postal 369, 13560-970, Sao Carlos, SP, Brazil

[10] Duda, Richard O., Peter E. Hart, and David G. Stork. Pattern Classification. New York: Wiley-Interscience. 2000.

[11] P. Erdos and A. Renyi. On the evolution of random graphs. Publ. Math. Inst. Hungar. Acad. Sci. 5 17. 1960.

[12] Fukunaga, Keinosuke. Introduction to Statistical Pattern Recognition, Second Edition (Computer Science and Scientific Computing Series). Toronto: Academic Press. 1990.

[13] Golumbic, Martin Charles. Algorithmic Graph Theory and Perfect Graphs. Academic Press. 2004.

[14] Gail A Carpenter, Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine, Computer Vision, Graphics, and Image Processing, v.37 n.1, p.54-115, Jan. 1987.

[15] J. Hawkins & D. George. Hierarchical Temporal Memory: Concepts, Theory, and Terminology, http://www.numenta.com/htm-overview/education/Num enta HTM Concepts.pdf, Numenta Inc. Date Last Accessed, 08/25/2010. 67

[16] Freeman, James. Simulating Neural Networks with Mathematica. Reading, MA: Addison-Wesley Publishing Company. 1994.

[17] Freeman, James and David Skapura. Neural Networks: Algorithms, Application and Programming Techniques. Reading, MA: Addison-Wesley Publishing Company. 1991.

[18] Hawkins, Jeff and Sandra Blakeslee. On Intelligence. New York : Times Books. 2004.

[19] Hebb, D. The organization of behavior : a neuropsychological theory. Mahwah, N.J. : L. Erlbaum Associates. 2002.

[20] Hertz, John, et al. Introduction to the Theory of Neural Computation. Westview Press. 1991.

[21] Hopfield, J.J and Tank, D. W. Neural computation of decisions in optimization problems. Biological Cybernetics 55, 141-146. 1985

[22] Hopfield, J.J. Neural networks and physical systems with emergent collective computational properties. Proc. Nat. Acad. Sci. (USA) 79, 2554-2558. 1982.

[23] Hopfield, J.J. Neurons with graded response have collective computational properties like those of two-state neurons. Proc. Nat. Acad. Sci. (USA) 81, 3088-3092. 1984.

[24] E. Kandel & J.H. Schwartz. Molecular biology of learning: modulation of transmitter release, Science, Vol 218, Issue 4571, 433-443. 1982.

[25] Kecman, Vojislav. Learning and Soft Computing: Support Vector Machines, Neural Networks, and Fuzzy Logic Models (Complex Adaptive Systems). London: The Mit Press. 2001.

[26] Kolaczyk, Eric D.. Statistical Analysis of Network Data: Methods and Models (Springer Series in Statistics). New York: Springer. 2009 .

[27] Koza, John R, On the Programming of Computer by Means of Natural Selection. MIT Press. 1998.

[28] Johnson, Richard A., and Dean W. Wichern. Applied Multivariate Statistical Analysis. Alexandria, VA: Prentice Hall. 2007.

[29] MacKay, David. Information Theory, Inference, and Learning Algorithms. Cambridge, UK. Cambridge University Press. 2003.

[30] Mclachlan, Geoffrey J.. Discriminant Analysis and Statistical Pattern Recognition (Wiley Series in Probability and Statistics). New Ed ed. New York: Wiley-Interscience. 2004. 68

[31] Mountcastle, Vernon. Perceptual Neuroscience: The Cerebral Cortex. Cambridge: Harvard University Press. 1998.

[32] Mountcastle, Vernon B. An Organizing Principle for Cerebral Function: The Unit Model and the Distributed System. Cambridge: MA. MIT Press. 1978.

[33] Oja Erkki. Simplified neuron model as a principal component analyzer. Journal of Mathematical Biology (3): 267-273. 1982.

[34] Pierce W.D. &,. Cheney, C.D. Behavior analysis and learning, Lawrence Erlbaum Associates. 2004.

[35] Simon, Herbert A. The sciences of the artificial. Cambridge, Mass. : MIT Press. 1981.

[36] O. Sporns, Graph theory methods for the analysis of neural connectivity patterns In: R. Kotter, Editors, Neuroscience Databases. A Practical Guide, Kluwer, pp. 171-186. 2002.

[37] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger, Network topology generators: degree-based vs. structural, Proc. ACM SIGCOMM, pp. 147 - 160. 2002.

[38] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, W. Willinger, On characterizing network hierarchy, Technical Report 03-782, Computer Science Department, University of Southern California. 2001.

[39] Baker, Tanya I., Marc Benayoun, Adam Seth Dickey, Nicho Hatsopoulos, Michael Lusignan, and Pascal Wallisch. Matlab for Neuroscientists: An Introduction to Scientific Computing in Matlab. Toronto: Academic Press. 2008.

[40] A. Tsonis, K. Swanson, and G.Wang. Estimating the clustering coefficient in scale-free networks on lattices with local spatial correlation structure. Physica A: statistical mechanics and its applications, vol. 387, pp. 5287- 5294. 2008.

[41] Vladimir N. Vapnik, The Nature of Statistical Learning Theory. Springer. 1995.

[42] Vogels TP, Rajan K, Abbott LF. Neural networks dynamics. Annu Rev Neurosci 28: 357-376. 2005.

[43] Watts D.J. & Strogatz, S. H. Collective dynamics of small-world networks, Nature 393 (6684); 409-10. 1999.

[44] Watts, Duncan J. Six Degrees: The Science of a Connected Age. New York: W. W. Norton & Company. 2004. 69

[45] Watts, Duncan. Small Worlds. Princeton, NJ: Princeton University Press. 1999.

[46] B.M.Waxman Routing of multipoint connections. IEEE J. Select. Areas Commun. 6(9), 1617-1622. 1988

[47] Y. Munakata and J. Pfaffly, Hebbian learning and development, Dev. Sci. 7, pp. 141-148. 2004.

[48] Yuste, R. & Denk, W. Dendritic Spines as Basic Functional Units of Nueronal Integration, Nature, 375, 682-4. 1995.

[49] Yuste, R & Bonhoeffer, T., Morphological Changes in Dendritic Spines Associated with Long-Term Synaptic Plasticity, Annual Review of Neuroscience Vol. 24; 1071-1089. March, 2001.

[50] "Happiness is taxonomy: four structures for Snoopy - page 8. http://findarticles.com/p/articles/mi m0FWE/is 3 7/ai 99011617/pg 8/. Date Last Accessed, 08/25/2010.

[51] Biological classification -Wikipedia, the free encyclopedia.Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Biological classification. Date Last Accessed, 08/25/2010.

[52] E. Kandel et al., *Principles of Neural Science*, 4th ed. New York: McGraw-Hill Medical, 2000, pp. 28, 325-326, 458, 549, 1247-1254.

[53] S. Blakeslee and J. Hawkins, *On Intelligence*. New York, NY: St. Martin's Griffin, 2005, pp. 40-142.

[54] J. Hertz et al., *Introduction to the Theory of Neural Computation*. Boulder: Westview Press, 1991, pp. 3, 200-327.

[55] T. D. Sanger, "Optimal unsupervised learning in a single-layer linear feedfoward neural network," *Neural Networks*, vol. 2, pp. 459-473, 1989.

[56] A. V. Rangan et al., "Quantifying neuronal network dynamics through coarse-grained event trees," *PNAS*, pp. 10990-10995, August 2008.

[57] McCulloch, W. and Pitts, W., A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 7:115 - 133, (1943).

[58] J. Hawkins, "Learn like a human: Why can't a computer be more like a brain?," *IEEE Spectr.*, pp. 22, April 2007.

[59] D. J. Felleman and D. C. Van Essen, "Distributed hierarchical processing in the primate cerebral cortex," *Cerebral Cortex*, pp. 2, 1991.

[60] O. Sanchez, "Emergence of hierarchy in networks," Ph.D. dissertation, Dept. Math., Rensselaer Polytechnic Inst., Troy, NY, 2010.

[61] G. M. Shepherd, "The neuron doctrine: A revision of functional concepts," *Yale J. Biology and Medicine*, vol. 45(6), pp. 589, 1972.

[62] M. P. Young, "The organization of neural systems in the primate cerebral cortex," *Proc. Roy. Soc. London, Ser. B*, pp. 17-18, 1993.

[63] J. R. Newton and M. Sur, "Rewiring cortex: Functional plasticity of the auditory cortex during development," *Plasticity and Signal Representation in the Auditory Syst.*. Brooklyn, NY: Springer, 2005, pp. 127-137.

[64] S. Ramon y Cajal, *Comparative Study of the Sensory Areas of the Human Cortex*. Charleston, SC: BiblioLife, 2010, pp. 1-80.

[65] C. Johansson and A. Lansner, "Towards cortex sized artificial neural systems," *Neural Networks*, vol. 20(1), pp. 4, 2007.

[66] M. Abeles, *Corticonics Neural Circuits of the Cerebral Cortex*. New York, NY: Cambridge University Press, 1991, pp. 9.

[67] M. Penrose, *Random Geometric Graphs*. New York, NY: Oxford University Press, 2003, pp. 2-3.

[68] S. Song et al., "Competitive hebbian learning through spike-timing-dependent synaptic plasticity," *Nature America Inc.*, pp. 919-926, 2000.

[69] N. Pathak et al., "A generalized linear threshold model for multiple cascades," *IEEE Int. Conf. on Data Mining*, pp. 965-970, 2010.

[70] G. G. Blasdel, "Orientation selectivity, preference, and continuity in monkey striate cortex," *J. Neuroscience*, pp. 3139-3161, 1992.

[71] D. J. Amit, *Modeling Brain Function The World of Attractor Neural Networks*. New York, NY: Cambridge University Press, 1989, pp. 15.

[72] R.F. Thompson and W.A. Spencer, "Habituation: A model phenomenon for the study of neuronal substrates of behaviour," *Psychological Review*, vol. 73(1), pp. 16-43, 1996.

[73] V. Garcia-Hoz, "Signalization and stimulus-substitution in Pavlov's theory of condition," *Spanish J. Psychology*, vol. 6, pp. 168-176, 2003.

[74] H. Adesnik et al., "NMDA receptors inhibit synapse unsilencing during brain development," *PNAS*, vol. 105(14), pp. 5599, 2008.

[75] G. Piccinini, "The first computational theory of mind and brain: A close look at McCulloch and Pitt's "Logical calculus of ideas immanent in nervous activity"," *Synthese*, vol. 141(2), pp. 188, 2004.

[76] S. Hayman, "The McCulloch-Pitts model," *Neural Networks*, vol. 6, pp. 4438-4439, 1999.

[77] L. da F. Costa et al., "Characterization of complex networks: A survey of measurements," *Advances in Physics*, vol. 56(1), pp. 167-242, 2007.

[78] W. Gerstner and W. M. Kistler, "Mathematical formulations of hebbian learning," *Biological Cybernetics*, pp. 2, 2002.

[79] D. Marr, "A Theory for Cerebral Neocortex," *Proc. Roy. Soc. London, Ser. B, Biological Sci.*, vol. 176, pp.197, 1970.

[80] T. P. Vogels et al., "Neural network dynamics," *Annu. Review Neuroscience*, pp. 357-376, 2005.

[81] J.J. Hopfield, " 'Neural' computation of decisions in optimization problems," *Biological Cybernetics*, pp. 143, 1985.

[82] A. Scott, *Neuroscience A Mathematical Primer*. New York, NY: Springer-Verlag, 2002, pp. 41-42.

[83] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Natl. Academy Sci.*, vol. 79, pp. 2554-2558, 1982.

[84] V. Braitenberg, *On the Texture of Brains: An Introduction to Neuroanatomy for the Cybernetically Minded*. Berlin, Germany: Springer-Verlag, 1977, pp. 1-127.

[85] L. N. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA: SIAM, 1997.

[86] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers with Personal Computer Applications*. New York, NY: McGraw Hill, 1985, pp. 289-294.

[87] F. J. Massey Jr., "The Kolmogorov-Smirnov test for goodness of fit," *J. American Stat. Assoc.*, vol. 46, pp. 68-78, 1951.

[88] L. R. Squire and E. R. Kandel, *Memory: From Mind to Molecules*. New York, NY: Henry Holt and Company, LLC, 1999, pp. 34-37.

[89] P. Dayan and N. D. Daw, "Decision theory, reinforcement learning, and the brain," *Cognitive, Affective, & Behavioral Neuroscience*, pp. 429-453, 2008.

[90] A. M. Hermundstad, "Learning, memory, and the role of neural network architecture," *PLoS Computational Biology*, vol.7, pp. 2, 2011.

[91] Y. Munakata and J. Pfaffly, "Hebbian learning and development," *Develop. Sci.*, pp. 141-148, 2004.

[92] E. Oja, "A simplified neuron model as a principal component analyzer," *J. Math. Biology*, pp. 267-273, 1982.

[93] Y. P. Shimansky, "Biologically plausible learning in neural networks: A lesson from bacterial chemotaxis," *Biological Cybernetics*, pp. 380, 2009.

[94] W. H. Calvin, *How Brains Think*. New York, NY: Basic Books, 1996, pp. 59.

[95] E. Torgersen, *Comparison of Statistical Experiments*. New York, NY: Cambridge University Press, 1991, pp. 411.

[96] J. Feng, *Computational Neuroscience A Comprehensive Approach*. Boca Raton, Florida: Chapman & Hall/CRC, 2004, pp. 293.

[97] S. Pomeroy et al., "Postnatal construction of neural circuitry in the mouse olfactory bulb," *J. Neuroscience*, pp. 1964, 1990.

[98] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. New York, NY: Cambridge University Press, 2003, pp. 510.

[99] J. E. LeDoux, *Synaptic Self: How Our Brains Become Who We Are*. New York, NY: Penguin Books, 2002, pp. 1799.