Spring 5-1-2023

Artificial Dendritic Neuron: A Model of Computation and Learning Algorithm

Zachary Hutchinson
University of Maine, zachary.s.hutchinson@maine.edu

ARTIFICIAL DENDRITIC NEURON: A MODEL OF COMPUTATION AND LEARNING ALGORITHM

By

Zachary Hutchinson

B.A. University of Houston, 1999

B.A. Hunter College, 2014

A DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

(in Computer Science)

The Graduate School

The University of Maine

May 2023

Advisory Committee:

Roy Turner, Associate Professor of Computer Science, Advisor

James Fastook, Professor of Computer Science

David Hiebeler, Professor of Mathematics

Phillip Dickens, Associate Professor of Computer Science

Bruce Segee, Professor of Electrical and Computer Engineering

ARTIFICIAL DENDRITIC NEURON: A MODEL OF COMPUTATION AND LEARNING ALGORITHM

By Zachary Hutchinson

Dissertation Advisor: Professor Roy Turner

A Lay Abstract of the Dissertation Presented
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
(in Computer Science)
May 2023

Dendrites are root-like extensions from the neuron cell body and have long been thought to serve as the predominant input structures of neurons. Since the early twentieth century, neuroscience research has attempted to define the dendrite's contribution to neural computation and signal integration. This body of experimental and modeling research strongly indicates that dendrites are not just input structures but are crucial to neural processing. Dendritic processing consists of both active and passive elements that utilize the spatial, electrical and connective properties of the dendritic tree.

This work presents a neuron model based around the structure and properties of dendrites. This research assesses the computational benefits and requirements of adding dendrites to a spiking artificial neuron model. A list of the computational properties of actual dendrites that have shaped this work is given. An algorithm capable of generating and training a network of dendritic neurons is created as an investigative tool through which computational challenges and attributes are explored.

This work assumes that dendrites provide a necessary and beneficial function to biological intelligence (BI) and their translation into the artificial intelligence (AI) realm would broaden the capabilities and improve the realism of artificial neural network (ANN) research. To date there have been only a few instances in which neural network-based AI

research has ventured beyond the point neuron; therefore, the work presented here should be viewed as exploratory. The contribution to AI made by this work is an implementation of the artificial dendritic (AD) neuron model and an algorithm for training AD neurons with spatially distributed inputs with dendrite-like connectivity.

DEDICATION

Professor Shawn Ell

ACKNOWLEDGEMENTS

First, I would like to thank my wife, Almut Rochowanski, for her love, patience and support. Research rarely belongs solely to the person standing in the lab or sitting at the computer. Mine is no exception.

Next, I would like to thank Professor Roy Turner who patiently waded through my offbeat ideas and feeble text. He never failed to provide good advice or let me leave his office without a positive mind. I am thankful for the encouragement and friendship of Professors James Fastook and Phillip Dickens which kept me going early on. Professor David Hiebeler stoked and structured my love of computer simulations. It is far too easy to lose perspective inside an amorphous, in-progress dissertation, so I am very thankful for Professor Bruce Segee who looked in and suggested I carve out a small bit.

I have a special thanks for the late Professor Shawn Ell. Without Shawn, this work does not exist. In my second semester, I showed up at his office and expressed my interest in neuroscience, his world. His response: *let's work on something.* His vision, that the importance of neuroscience extends beyond its domain, lives on. And I thank his lab students for tolerating an outsider and his outsider comments.

I thank my parents and grandparents for bringing me up in an environment that valued learning, creativity, exploration and craftsmanship. With regards to family, I am profoundly blessed.

Finally, I would like to thank my fellow students. The best part of education is being surrounded by people trying to figure stuff out. I cannot imagine completing this without the opportunity to absorb energy from those around me. To Mark Royer, Brian Toner and Mike Cressey: gentlemen, every Saturday afternoon was an honor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Nevertheless, children still populate the slums. The struggle for existence is no simple affair, and things happen which no mathematics can foretell.

- D'Arcy Wentworth Thompson, *On Growth and Form* [153]

CHAPTER 1
INTRODUCTION

> You're welcome to study neuroscience, but please try and help them instead of learn from them.
>
> Marvin Minsky[117]

1.1 Statement of Purpose

The purpose of this work is to create and demonstrate a neural model of computation based on several purported properties of biological dendrites. From this model, an artificial dendritic neuron (AD neuron) is derived for use in artificial intelligence work. Lastly, an algorithm by which AD neurons can be trained is proposed and tested.

1.2 Motivation

Advances in neuroscientific study have been a source of inspiration for artificial intelligence (AI) research since the latter's inception. The brain's ability to solve complex real world problems while adapting to a dynamic environment embody the practical goals of AI. Neuroscience-inspired solutions are wide spread throughout AI. They are visible in approaches to learning, memory, planning, computer vision and artificial neural networks (ANN) [69]. Such neuroscience-inspired solutions are often the result of advances in our understanding of the brain.

One area of neuroscience research that has made significant strides in recent decades is the study of sub-neural structures. One such structure is the dendrite. Dendrites are the elongated root-like extensions through which a neuron receives a significant portion of its input. The biological dendrite is crucial to neural computation. It integrates and propagates arriving signals from their points of origin to the cell body. Many of the

sub-neural mechanisms underlying learning and memory are located in the dendrite. Its shape enables and influences neural processing. In neurons with elaborate dendrites, by the time a single input signal reaches the neuron body, it has been filtered, transformed and integrated with hundreds, if not thousands, of other signals. Dendrites allow a single neuron potentially to integrate thousands of inputs and respond to hundreds of distinct patterns [70, 72]. Ultimately, biological neural networks and high-level intelligence could not exist without them [99, 54, 57].

However, sub-neural networks, such as the dendrite, are largely absent from ANN implementations. The majority of artificial neural networks are comprised of the point neuron which is a neuron model whose afferent, or incoming, connections are made only at the cell body. The point neuron integrates its inputs linearly and simultaneously. By contrast, dendritic input integration is a nonlinear process which happens locally, repeatedly and in parallel over the length of the dendritic tree. The result of this temporal and spatial disbursement is that a dendritic neuron is more computationally equivalent to a network of point neurons than the point neuron itself [17].

While the ANN community has taken an increased interest in dendrites in recent years in search of new bio-inspired forms, AI-centered work on dendrites remains very small. Past work has not produced a clear path forward in the form of a widely adopted model. Dendritic structures are often created by reusing ANNs which miss or make it difficult to model unique characteristics. The benefits of adding dendritic structures is unclear or varies from work-to-work. A thorough return to the biological source, that could identify their function and benefits and produce a useful abstraction, is not visible in the research. To aid in this endeavor, this work proposes a new neuron model, the artificial dendritic (AD) neuron.

An AD neuron represents a computational improvement over the point neuron. The AD neuron on its own is capable of recognizing and selecting between independent patterns formed by subsets of its input space. Because of the spatial ordering of inputs along the

dendritic tree, independent patterns do not only interact with the cell body but can combine within the dendrite. Such interaction forms composite patterns and selects between competing ones.

A neural-level comparison between point and dendritic models may not apply to the network level. Are networks of AD neurons (ADNN) an improvement over classical ANNs? AI currently lacks a domain-appropriate model to investigate this open question. Providing an answer is important because it carries with it ideas for topological improvements to ANNs and suggestions about the functional role of dendrites in biological networks. To move forward, an AD model is necessary.

1.3 The Case for a Biological Simplification

Biological dendrites are a diverse group. Similar to fingerprints, it is likely no two are perfectly identical. Variations in structure and composition suggest each dendrite combines information differently into a specialized signal. These differences contribute to the production of a unique neural input-output relationship which brings further diversification to the brain network. Given such diversity between variants, is it possible to represent dendrites using a single model and retain their computational complexity?

If the previous question is to be answered, the biological dendrite must be simplified to a set of properties without sacrificing all the benefits of dendritic computation.

A fundamental set of properties has several benefits. A simple AD model can serve as a foundation to explore further complexity. It is possible that certain dendritic variations or specializations are beneficial to certain types of tasks. A simplified foundation potentially can accommodate dendritic specialization by being compatible with a range of optional properties. Next, weeding out dendritic idiosyncrasies reduces their complexity and variability, simplifying the implementation and the training process. Finally, certain dendritic properties are common to many types of networks including ANNs. Focusing on

common network properties potentially opens the AD neuron to traditional and AI forms of network analysis. This could give them wider applicability and a general appeal.

1.4 The Five Attributes of Artificial Dendrites

The AD neuron is defined around the abstraction of five general attributes of biological dendrites which contribute to their purported computational abilities. These five attributes are *dispersed inputs*, *shape*, *compartmentalization*, *dendritic spikes* and *plasticity*. Each of these attributes contributes to learning within the artificial dendritic neuron.

1.4.1 Dispersed Inputs

Synapses, the points on a neuron which receive input from other neurons, are dispersed across the entire surface of the neuron with a substantial number arriving on its dendrite. Each input is unique in that it arrives at a difference point on the neuron and is subject to differing local properties. Signal arriving on the dendrite propagate through the tree. Since each arrival point is unique, paths to the soma are also unique. As a signal propagates it is subject to alterations due to local activity and wider dendritic properties.

Abstracted, each path through the artificial dendrite is a unique function. Dispersion creates an order among neural inputs. Order creates a set of unique paths to the soma, one for each input. Each unique path is composed of a series of transformations of the propagating signal. The series of transformations can be represented by a function which defines how a signal arriving at a specific input location impacts somatic output.

1.4.2 Shape

Biological dendrites come in many forms. The shape of the dendrite determines which signals it receives from its surroundings. For the artificial dendrite, the shape of a dendrite is defined by the number and length of its unique paths. A dendrite's shape determines the minimum, maximum and average distances between input positions. A shape with many

branches facilitates independent, parallel computations. A long thin shape devoid of branches allows for detailed sequential computation.

1.4.3 Compartmentalization

In general dendrites are isopotential structures, meaning different branches of a dendrite can have different electrical potentials [70, 12]. This phenomenon belies an underlying electrical compartmentalization. Compartmentalization of the dendrite allows one neuron to possess multiple integrative units where input is combined quasi-independently.

The artificial dendrite translates this attribute by dividing its dendritic tree into multiple compartments. Within each compartment arriving signals integrate linearly. Signals propagating between compartments undergo a nonlinear transformation. This local linear bias divides a network into a set of sub-networks. Compartments allow a single neuron to evaluate subsets of its inputs independently and repeatedly. The partitioning of inputs into subsets potentially allows each compartment to recognize a sub-pattern within the input space. The artificial dendrite, composed of multiple compartments, recognizes composite patterns (i.e., patterns comprised of patterns).

1.4.4 Dendritic Spikes

The dendrites of some neuron types are capable of threshold behavior in the form of dendritic spikes. Dendritic spikes, in general, are caused by certain input received along a specific section of the dendrite. While the profile, behavior and cause of dendritic spikes varies by type, they are an internal signal produced by the dendrite for a larger part of the neuron.

For the artificial dendrite, each compartment defines which combinations of input signals are allowed to propagate forward and which do not. The concept of a dendritic spike is interpreted to be a compartment-level meta-signal indicating the presence of a recognized sub-pattern.

1.4.5 Plasticity

Plasticity is the ability of a neuron to alter local properties to increase or decrease the impact of a local signal on a neuron's wider behavior. Much of neural plasticity in the adult brain is attributed to repeated or coinciding activity both within and between neurons. Different parts of the dendrite are subject to different, possibly independent, plasticity mechanisms. The artificial dendrite is concerned with two: synaptic and branch plasticity. Synaptic plasticity raises or lowers the scaling of a signal arriving at a point on the dendrite. Branch plasticity raises or lowers the scaling of a signal moving from one branch of the dendrite to another. The artificial dendrite has the ability to alter properties which affect the strength of signals at their points of arrival *and* the strength as signals move from one point, or compartment, to another.

1.5 The Artificial Dendritic Neuron

The artificial dendritic neuron is a computational unit consisting of a set of inputs and a single output. Neural output occurs when input arriving at the soma, the root of the neuron, reaches a threshold. However, unlike the point neuron, the AD neuron's inputs are spatially ordered and connected to the cell body through a dendritic tree rather than to the cell body itself. The distance between inputs defines how the tree itself alters propagating signals. Because of this, the AD neuron possesses a topological bias that favors some input patterns over others. Figure 1.1 highlights this difference in input connectivity.

In the AD neuron, an afferent signal arrives at a distinct point along the tree and must propagate down a unique path to arrive at the cell body. As a signal propagates, it integrates both linearly and nonlinearly with other signals either arriving directly along its path or originating from other parts of the tree. Each input affects certain paths through the dendritic tree while having little to no effect on others. Finally, each tree passes to the cell body one composite signal. Because of these properties, the majority of computation in an AD neuron happens within the dendritic tree.

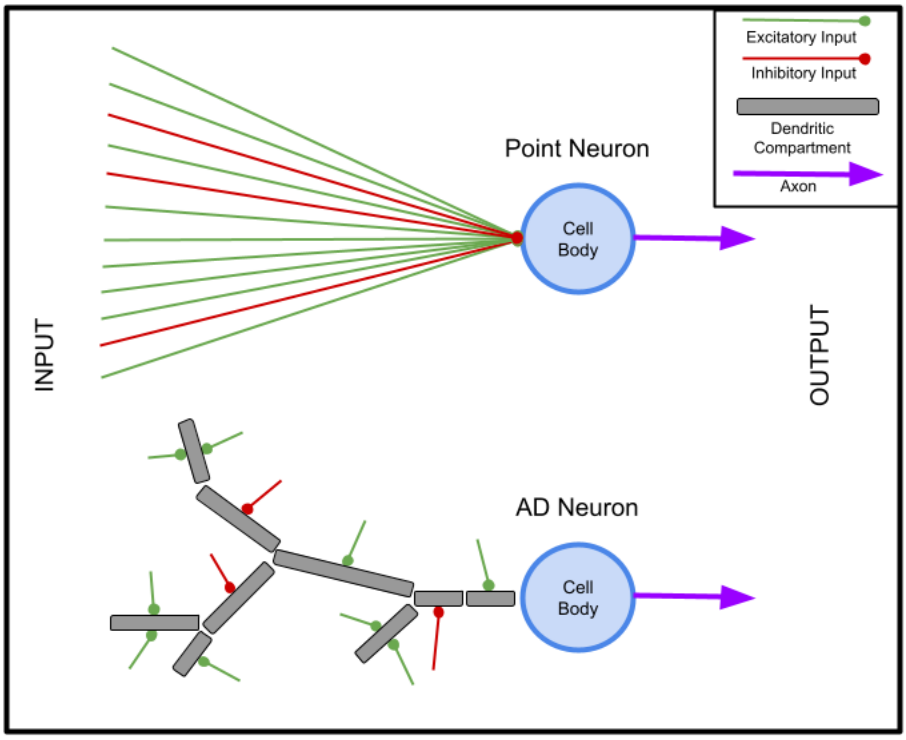


Figure 1.1: Top: point neuron with a mix of excitatory (positive weights) and inhibitory (negative weights) inputs. Bottom: AD neuron with the same number and type of inputs distributed over a dendritic tree.

In this work, the AD neuron is given in three different forms. The basic model describes a conversion of the five properties given above into a working computational model. The basic model is minimal in that it makes few assumptions about any domain to which it might be applied, limitations on dendritic shape or connectivity, or the types of signals it receives. An extended model of the AD neuron fills in many of these undefined aspects and is presented with two purposes. First, it is an example of how the basic model can be extended to further define its behavior. Second, it provides a foundation on which a trainable, third version of the AD neuron is constructed. The final model is the trainable version of the extended model. Various training algorithms are presented and tested to demonstrate its capabilities. Together the three models form a process that can be forked at any point to create alternatives. We hope that defining the AD neuron through this constructivist presentation will aid in the formulation of such alternatives.

1.6 Training the Artificial Dendritic Neuron

1.6.1 Neural Learning

Both artificial and biological neurons learn by capturing the relationship between input and the response it produces. The artificial neuron, for example, learns by exposure to training examples. The correctness of its response forms an error signal which is used to manipulate the neuron's input parameters.

Learning in a biological neuron is more complex but is still thought to be, at least partially, driven by an evaluation of input and response. It is made up of quasi-independent processes that operate over various time scales and affect different parts of the neuron.

A model of neural learning, then, answers two questions:

- Which neural properties are altered during the learning process?
- How are input-response relationships captured to make these alterations?

The artificial neuron alters the weights of its inputs based on the correctness of its response. The biological neuron alters its electrical, chemical and physical properties in response to input-output timing (i.e., spike-time dependent plasticity), chemical signals in the form of neurotransmitters, and the activity of nearby neurons.

How then does the AD neuron answer these two questions?

1.6.2 Artificial Dendritic Neuron Learning

The AD neuron is trained through alterations to its five properties. Through these alterations, the AD neuron builds its own dendritic tree. Input activity and subsequent neural responses move the inputs to new positions and alter their weights. These new positions are then used to build a new dendritic tree by (re)connecting inputs and redefining compartments. Figure 1.2 depicts this cycle.

Learning in the AD neuron is driven by the temporal relationship of input activity and neural responses. The temporal proximity of activity at two inputs or between an input

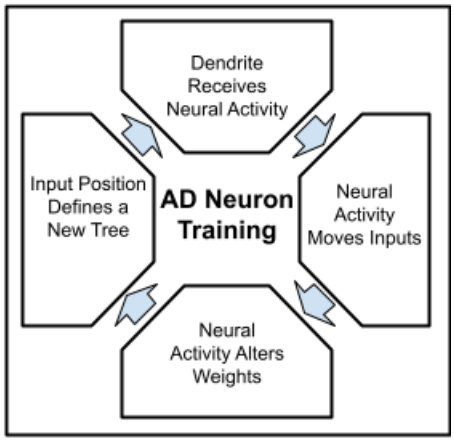


Figure 1.2: The AD Neuron Training Cycle

and the cell's response is assumed to carry meaningful information. By capturing such co-activity the AD neuron's parameters can be altered to improve task performance. How does this happen?

The artificial dendrite begins completely disconnected. First, co-activity among inputs is used to either cluster inputs together or drive them apart. The direction of movement depends on whether the activity contributes to a desirable neural response. Similarly, the timing of input activity and neural response determines an input's proximity to the cell body. These two relationships--input-to-input and input-to-output--determine where each input is positioned within the dendritic field. Next, the position of inputs within the field is used to generate the actual dendritic tree. The dendritic field is the N-dimensional space inhabited by the inputs. This process connects neurons to each other and, ultimately, to the cell body. This is done using a modified minimum spanning tree algorithm. Network distance is used to cluster inputs and compartmentalize the tree. The resulting dendritic network carries both the feed-forward signals which drive the neural response and the back-propagating signals which govern weight changes.

In summary, it is the timing of activity arriving at or leaving from the AD neuron which contributes to the construction of the dendritic tree.

1.7 Overview

The rest of the proposal is organized in order from background information to remaining work. Chapter 2 focuses on the biological inspirations for this work. It begins with a brief description of dendrites and then focuses on several properties which are important to dendritic information processing. Chapter 3 describes the related AI work which has explored artificial dendrites. Chapter 4 introduces the AD neuron model, giving a base and extended variant. It discusses how dendrites expand the artificial neuron and the complexity this brings. Chapter 5 describes through a series of fixed (i.e., without learning) experiments the behavior capabilities of the AD neuron. Chapter 6 presents a suite of algorithms capable of training the AD neuron's various properties. Chapter 7 walks through the results of a series of experiments which investigate the behavior of the individual dendritic learning algorithms. Chapter 8 benchmarks the learning capability of the AD neuron as a whole using several standard data sets. Finally, chapter 9 concludes the work by summarizing results and listing a myriad of alternative idea and future work which has been generated by this dissertation.

CHAPTER 2

BACKGROUND - NEUROSCIENCE

After decades of experimentation and modeling, neuroscience possesses a rich characterization of dendritic function and its various contributions to neural computation; however, a unified map of dendritic function is far from complete. Theoretical work is problematic due to the seemingly varied attributes both across and within dendritic types. It is likely the morphological classification of neurons, and thereby dendrites, in equal parts resolves and adds to the confusion. Individual theories or aspects of dendritic computation overlap making a segmented conversation difficult.

A passage from an article by Payeur et al.,[125] in reference to information selection, sums up the spirit of the sprawling nature of the discussion surrounding dendritic computation:

> Obtaining an adequate understanding of the full computational role of compartmentalized information poses a significant theoretical challenge. In this direction, dendritic states have been referred to as a teacher or target, a prediction, an error signal, a plasticity regulator, an associative signal or attentional signal. We do not expect all dendrites to have the same function, nor do we believe these interpretations to all be mutually exclusive [125].

With this quote in mind, a focus was placed on the more general computational characteristics attributed to dendrites rather than those that contribute to any given specialized cognitive task. As is common in neuroscience work, it is important to define the scale of one's view in terms of neuro-architecture. Since the topic is the dendrite, the scale spans a spectrum that views the neuron as both a set of subneural components (or compartments) and as a whole. It is tempting when explaining neural and, even more so, dendritic computation to reach for the underlying detail of neurotransmitters and the electrochemical properties of membrane tissues. This short survey of dendritic computation

has tried to avoid too much magnification. Similarly, it does not argue for the neurobiological realism of the AD neuron. Instead, it focuses on a generalized, utilitarian description of dendrites that summarizes the neurological inspiration, related research and foundation for this work.

2.1 Dendrites

2.1.1 Neural Communication

Neurons send and receive information through different parts of the cell. Generally, output leaves the neuron via the axon and enters a cell through the dendrite. Both axon and dendrite extend from the cell body. However, dendrites can take on far more elaborate shapes. Whereas the axon is typically a single fiber with relatively few branches, the dendrite can have hundreds of branches. Furthermore, a single neuron can have many distinct dendrites emanating from different sides of the cell body.

While axons and dendrites send and receive information, they do not actually make contact with each other. Where the two come in close proximity, the dendrite can grow small extensions, called spines, that further narrow the gap between axon and dendrite. Where many spines reach for the axon, the axon can swell to form a bouton. Still, neither of these make a physical connection and the remaining space between the two is called a synapse. These elements can be seen in Figure 2.1 which depicts an actual cortical pyramidal neuron.

At the synaptic cleft, electrical signals moving outward along the axon are converted to neurotransmitters. These transmitters flood the narrow space between axon and dendrite. They are quickly received and taken in by the dendrite where they are converted locally back to an electrical signal. Once in the dendrite, the signal propagates onward.

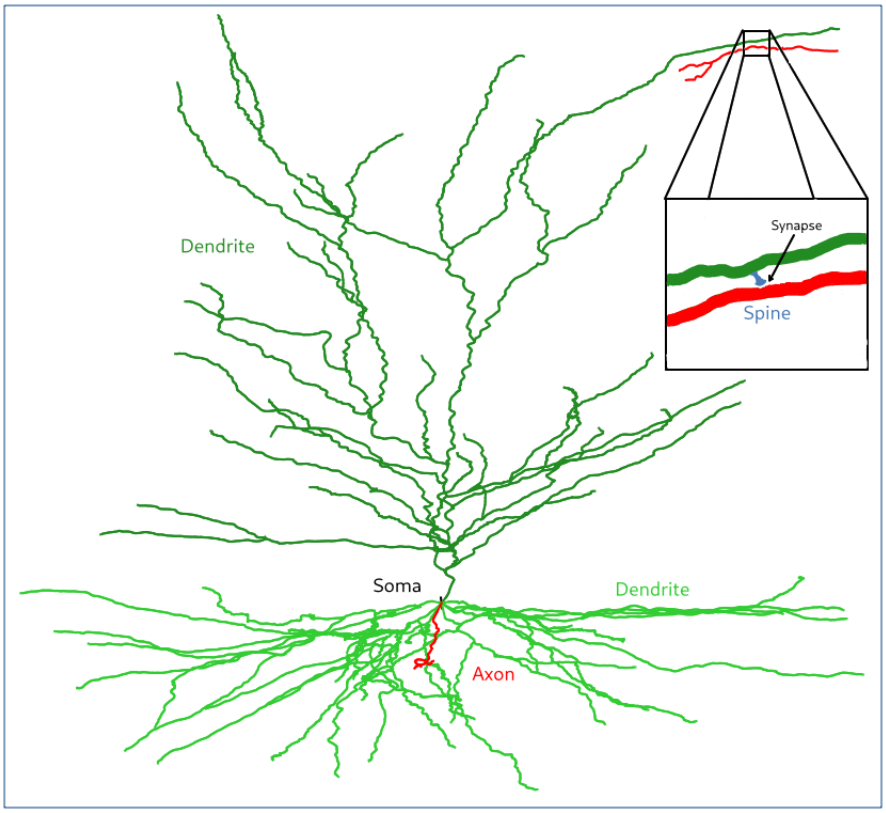


Figure 2.1: A labeled cortical pyramidal neuron. The neuron has one apical dendrite (dark green) and several distinct basal dendrites (light green). Neuron data comes from neuromorpho.org [10, 86]. Visualization created personally using Python3. Zoomed in area and spine drawing were done by hand for demonstration purposes.

2.1.2 Role of Dendrites

The dendrite appears to have two main roles. By their elongated shape, dendrites allow the neuron to make many more connections to other neurons than it could if it lacked them. This is a role shared with the axon. Second, the dendrite itself integrates and processes incoming signals locally before they ever reach the soma, or cell body. It is this second, computational role that is the focus of this work.

The elongated morphology of the dendrite makes possible the high degree of connectivity in the nervous system, providing more surface area without impeding common neural spaces [149]. For example, the space-filling, mossy dendritic arbor of Purkinje cells of the cerebellum has been estimated to make anywhere from 69,000 to 223,000 connections [55] over a surface area 100 times the size of its cell body [162].

(a) (b)

Figure 2.2: Examples of two types of dendrites. Left: Bipolar, e.g., cortical pyramidal cells. Right: Space-filling, e.g., Purkinje cells. Drawings were created by synthesizing several textbook examples.

Dendrites take on many shapes. Visually, many types resemble the roots of plants or the branches of a tree (Fig 2.2). Shapes range from a single branch reaching for a specific input space to densely packed, space-filling structures which capture the activity of the surrounding area (Figure 2.2). There are many characterizations of dendritic shapes (cylindrical, conical, laminar, planar, and even adendritic) [149]. Dendrites are further categorized by the location of their root. Certain types of dendrites originate from different regions of the cell body.

Dendrites are the computational workhorse of the biological neuron; they are not merely arms by which neurons reach distant signals. The dendrite performs active and passive computational duties by transforming, filtering and integrating signals within the tree. This innate ability to transform incoming signals enhances the overall computational power of the neuron [38]. Furthermore, dendrites possess the means to evaluate spatiotemporal patterns locally. This means that signals arriving at one part of the dendrite can be evaluated with some independence and do not immediately affect the arrival of others.

> [...] compartmentalization of information processing endows neurons with a second processing layer that boosts the computational capacity of the neuron by at least an order of magnitude compared to that of a thresholding point neuron.[124]

The following subsections cover briefly the purported computational properties of dendrites which have influenced this work. A full treatment of any one of these dendritic properties is beyond the scope of this work; therefore, the discussion is focused on how each influenced the creation of the AD neuron and its four properties.

2.2 Dendritic Diversity

Dendrites do not all behave the same way. Dendritic behavior is the result of a myriad of interdependent, active and passive properties [125]. These properties vary in kind, density or distribution between dendritic types.

For example, dendritic morphology, or shape, determines the neuron's capacity to *listen* to surrounding activity. The densely packed dendritic branches of the Purkinje neurons (Fig. 2.2b) are thought to capture all signals moving through a local space. Similarly, the stellate neuron type samples a local space through separate dendrites extending from all sides of the cell body. This is in contrast to biconical and spindle shapes whose dendrites extend from opposite sides of the soma allowing it to sample spatially distinct fields [149, 38]. The dendritic field is the physical space occupied by the dendrite within which it can make connections. These spaces interlock with those occupied by the dendrites and axons of other neurons. This interlocking geometry of axonic and dendritic fields--their orientation and location--contribute to the specialization of each neuron type.

The variability in dendritic behavior cannot be ascribed to differences in shape alone. Ion channels are small apertures in the membrane of the neuron which allow the influx or efflux of different ions. Ion channels are responsible for holding or changing the membrane's electrical potential. More specifically, ion channels allow specific types of

positively or negatively charged ions (calcium, sodium, potassium, etc.) to pass into or out of the cell and thereby change, in a site-by-site manner, the polarization of the cell. Local densities of ion channels determine how synaptic sites respond to input and how current moves from one location to another [76].

Dendritic spines and the connections they make to other neurons contribute to dendritic diversity. Spines are dynamic. Throughout the life of an organism new spines grow and old ones are pruned [18]. Changes in spines weaken or strengthen the connections between neurons [68]. The location of spines on the parent dendrite as well as the density of local spine populations impacts the role of the signals received from other neurons [13, 165].

Dendrites also differ in their active responses. Dendritic spikes are rapid swings in the neuron's membrane potential originating in the dendrite [98]. They are associated with different ion channels, and the input requirements for each type varies dramatically [139, 123]. Dendritic spikes come in several types. The active response can differ by which input patterns are capable of causing it and in the nature of the response (i.e., length, strength, etc.) [105]. For some dendrites, clustered input patterns appear to determine the post-synaptic response [60]. Other neurons with dendrites of a different type respond equally well to inputs scattered across the dendrite [173, 174].

Research suggests that each dendrite makes a unique contribution to the neural circuit and its general type allows it to fulfill basic roles. Seemingly, this would make any generalization of their computational abilities more difficult; however, dendritic diversity has positively influenced the current project. The biological diversity lent a freedom to the design of the AD neuron implementation. It suggested that there is no single, best implementation. With enough research and experimentation, we might identify a multitude of implementations which focus on different dendritic properties and are best suited for different tasks. Furthermore, it inspired the design of the training algorithms. If shape, order and compartmentalization determine a dendrite's role, these characteristics should be allowed to vary on a neuron-to-neuron basis. In other words, they should be trainable.

2.3 Dendritic Information Processing

> Individual nerve cells convert the incoming streams of binary pulses into analog, spatially distributed variables [...]. A number of transformations can be applied to these variables besides subtraction and addition: low- and band-pass filtering, normalization, gain control, saturation, amplification, multiplication and thresholding [88].

Neuroscience research indicates that dendrites heavily process afferent, or incoming, signals before they reach the cell body. Dendrites are thought to filter, combine, exclude, amplify, modulate, remember and even predict signals. This information processing depends on a suite of passive and active elements. Specifically, this work identifies synapse location, signal transport, dendritic morphology, dendritic spikes and forms of synapse and branch plasticity as five crucial components of the dendritic information processing toolkit.

While this work condenses the dendrite to five properties that determine information processing, neuroscience does not have a unified dendritic model that covers the functional significance of all of its elements. Some of them, especially across different neuron types, appear to be incompatible (e.g. cluster-sensitive versus scatter-sensitive [158]). As it stands, the causes of dendritic processing are better understood than its purposes. Surveys of dendritic information processing capabilities struggle to move beyond case studies and into a theoretical treatment [99, 105, 125].

The five components mentioned above helped to identify the four properties of the AD neuron. In this section, they are described and their connections to the four properties are highlighted.

2.3.1 Signal Transport

Signal transport describes how the propagation of a single signal from one point in the dendrite to another alters the signal itself. What are the effects, if any, of this movement? Which active or passive mechanisms cause alteration?

The movement of current back and forth along the dendritic tree is one of the longest studied properties of the dendrite [162]. For more than a century, cable theory has been used to describe how electrical signals move through a series of connected cylinders each having a specific radius and length [75, 36].

A signal moving between synapse and soma is subject to the electrical characteristics of the shape and length of each cylinder which together comprise the dendritic tree. Biological neurons, and thereby dendrites, have a capacitive membrane that has low electrical resistivity and through which charge leaks into the surrounding fluid. The leaky membrane reduces the strength of the signal. Additionally, internal, or axial, resistance (parallel versus perpendicular movement) further attenuates the signal coming from distal branches. The passive electrical model of dendrites predicts that signals impinging distal locations arrive at the soma (or vice versa) severely attenuated.

Signals take the form of post-synaptic potentials (PSP). PSPs can be either excitatory (EPSP) or inhibitory (IPSP). PSPs are the polarization or depolarization of a post-synaptic cell's membrane due to the movement of positively or negatively charged ions into or out of the cell. The region of polarization/depolarization moves bidirectionally from the initial site due to adjacent ion channels opening. Ion channels open or close under certain conditions (e.g., voltage differences or ligands) depending on their type. The treatment of PSP movement is essential for non-isopotential neuron models with distributed input. The distance a PSP travels from its site of initiation determines the electrical compartmentalization of a dendrite (or part thereof) with respect to a single synapse.

However, experiments on the integration of PSPs in brain slices do not always reinforce cable theory's predictions [122]. The problem is that the dendritic network is a dynamic entity. The dendrite is affected by the signals it carries. The discovery of active properties [98] complicated the view that current propagation through the dendrite is completely passive [105] and that distal signals arrive at the soma highly attenuated [65].

Ion channels, and thus the dendritic branch, respond to the presence of current. Dendrites are comprised of multiple types of ion channels. Movement of PSPs is both nonlinear and uneven. Ion channels are not uniformly distributed [66]; therefore, the effects of movement differ between neurons but also between parts of the same neuron.

Dendritic segments are capable of altering their efficacy for signal transport. Branch-strength potentiation (mentioned later in Sec. 2.3.5) is a mechanism by which entire segments of a dendrite can increase or decrease their strength. Activity within specific dendritic branches can have an enhanced or diminished impact on the somatic response over its sibling branches [106, 92] based on recent activity. Branch-strength potentiation complicates a within-neuron model of signal transport uniformity by predicting that not all dendritic distances are measured the same and even the distance over the same segment varies with activity and time.

The circular dependency of network state and signal have strongly influenced the direction of the current work. As a signal moves from one point in the dendrite to another it is altered by the dendrite's local properties and current state. Simultaneously, localized sections of the dendrite can be potentiated or depressed by the activity either arriving at or passing through it. This is particularly true when passing and arriving information collide. Such events are thought to be crucial to changes in synaptic strength. Signal transport has influenced how the role of distance between synapses has been implemented and how colliding signals can amplify (or suppress) one another.

2.3.2 Synapse Location

The functional significance of a synapse's location in relation to the cell, but also to its sibling synapses, remains an open question. And to answer it for any given synapse potentially requires a detailed understanding of the processes that drive neurogenesis, dendritic morphogenesis and spinal and synaptic growth. It also requires an understanding of the role of activity and experience in these processes.

For example, Koch and Segev suggested that:

> Synaptic clustering requires a learning rule that encourages simultaneously active synapses to cluster in adjacent dendritic regions, whereas uncorrelated synapses should have no privileged spatial relationship to each other [88].

Generally, research considering the question of a synapse's location has focused on two separate relationships: synapse-to-synapse and synapse-to-soma. The former relationship is evident in neuroscience research investigating dendritic spikes. Dendritic spikes in certain neuron types have been shown to depend on multiple inputs arriving in close temporal and spatial proximity [60]. Distal branches are thought to compartmentalize local synapses [174]. By themselves, signals arriving at these distant branches attenuate before reaching the cell; however, when received within a short window of time, they can generate a dendritic spike. A dendritic spike, from an informational perspective, can be thought of as a unified, possibly superlinear response, to a group of individual signals. Branches, in some instances, communicate with the rest of the dendrite as a whole (e.g., dendritic spikes).

Next, what is the functional consequence of a synapse's distance to the cell body itself? Early models of dendrites predicted that signals from distal synapses suffered attenuation before arriving at the soma [133]. Research into EPSP efficacy [172], inhibition [83] and learning rules [93] reinforce this idea. However, other evidence from the cortex [24], the hippocampus [104, 7] and spinal motor neurons [81, 6] indicates that some neurons possess active and passive properties that reduce or eliminate the distance-based attenuation. This indicates that there is a need, within the dendritic tool kit, for both the compartmentalization and democratization of synapse-soma interactions.

The synapse-to-soma relationship could indicate a more general role. One theory classifies signals as either driving somatic responses or providing contextual modulation [144, 1]. The literature freely admits that the definition of *driver* and *modulator* is unclear and certainly differs between neuron types. However, a driver input is associated with external or sensory input; whereas, a modulator is more contextual and incapable of

producing a neural response on its own. This last point--the ability of a synapse to be heard by the soma--is at the heart of the definition. Whether a synapse is heard depends on location. Location is crucial with drivers occupying proximal spot on the dendrite and modulators making more distal connections.

Similar to the driver-modulator theory, other research associates the relative location along the proximal-distal axis of cortical pyramidal neurons with one of two types of modulatory influence. Distal synapses lower the spiking threshold of more proximal segments of the dendrite; whereas, proximal synapses provide a gain boost for distal signals [13]. Whereas the driver-modulator theory defines distal and proximal synapses in relation to the soma, this one does not. Synaptic locations are relative to other synapses. A single synapse, for instance located along a midpoint, could play both roles.

The above research presents the idea that the location and order of synapses and compartments within the dendrite determines their computational role. In fact, each synapse might have as many roles as it has sibling synapses. This point influenced the disentangling of the synapse-to-soma from the synapse-to-synapse relationship. The spatial proximity of synapses determines the time window for signal overlap and interplay. The combined distances (angular and radial) between synapses determines the formation of compartments. Synapse-to-soma relationships determine a synapse's direct influence over the neural response. In other words, those synapses close to the soma have the last word.

2.3.3 Dendritic Morphology

The shape of a dendrite is important to the behavior of a neuron. Morphological variations are enough to suspect that shape and purpose are inextricably linked [113]. While dendrite morphology is one of the determining features of neuron categories, the functional significance of both type-specific and type-invariant features remains largely undefined [44]. This is in large part due to the dependence of such definitions on the functional significance of neuron types themselves.

Ramon y Cajal first suggested that neurons benefit from the spatial and energy efficiency of dendrites [30]. Efficiency and function are associated with dendritic shape. Both explain why the process of evolution might favor their current form; however, it is unclear to what extent the requirements of each has determined current forms.

Dendritic shapes are not arbitrary. They minimize the costs of maximizing the number of potential connections to other neurons [166, 167]. A brain devoid of the elaborate spiny branching structures enabling neural connectivity would occupy a space five orders of magnitude larger than one with them [32]. However, the need for efficiency alone does not determine topology. Research suggests that shape is determined by innate growth patterns that are susceptible to modification by local activity. For example, a best-fit of the shape of the dendrite of lobula plate tangential cell in flies can be produced by a version of Prim's spanning tree algorithm [129] that includes a branch factor bias [37]. This suggests that neurons of a similar class are likely determined by an identical genetic "growth program" [40]. This suggests that arborization during morphogenesis is, at least partly, governed by rules that place restrictions on the path length from the dendritic root to any given connection. This also implies that for any given pair of synapses, their *order*, or their relative proximity to the cell body, is likely determined more by a need for efficiency and less so by any ordering requirements for local computation.

One indication that shape impacts behavior comes from modeling work investigating how dendritic topology impacts the response profile of a neuron. Not all neurons spike the same when sufficiently depolarized. Some produce a single spike; others burst. Computational models focusing on dendritic topology suggest that a dendrite's shape and mean path length contribute to the neuron's firing frequency [161] and activation pattern (i.e. tonic or bursting) [48].

Other research suggests that shape contributes to the neuron's ability to store and retrieve patterns. Pattern recognition performance was examined by exhaustively generating and testing all possible dendritic trees with 22 synapses. Results suggested that

topologies with less asymmetry and a shorter mean depth (or a smaller mean electrotonic path length for tapered dendrites) show a stronger recall of stored vs. novel patterns [44]. Asymmetric dendrites are unbalanced trees where certain parts branch more than others and synapses are unevenly distributed. Mean path length is the average length of all paths through the tree measured from the root to all terminal branches.

Another thread of modeling work investigated the connection between shape and behavior by developing algorithms to generate dendrites capable of producing certain behaviors [28]. The algorithm used a stochastic process dependent on local branch properties to produce the length and diameter of further branches. Research based on the work by Burke [28] has shown that a neuron exhibiting specific behavior can be created through an evolutionary algorithm capable of building dendritic segments [156]. These works reinforce the idea that shape can determine function and demonstrates that dendritic shapes which produce a specific function can be algorithmically generated.

The idea that a variety of dendritic shapes can be reproduced by a growth algorithm [39] has strongly influenced the current work's approach to synaptic connectivity.

2.3.4 Dendritic Spikes

Dendrites, like the soma, are capable of non-linear, suprathreshold events in the form of dendritic spikes. Dendritic spikes are viewed as branch- or compartment-level signalling mechanisms that can unify and amplify groups of signals enabling them to reaching the soma or other parts of the dendrite that, individually, they would not reach. In this way, they could function as sub-pattern or coincidence detectors [71] within a larger set of inputs. Their generation depends on synchronous input to a dendritic region [142].

Dendritic spikes are the result of a localized feedback among ions channels which depolarizes a region of the dendrite's membrane [145]. Due to the uneven distribution of these channels over the length of the dendrite, the primary driver of a dendritic spike can differ by location [8]. Therefore, the cause, location and effect of a dendritic spike varies.

It is likely that different types of spikes serve different purposes. For example, calcium-based spikes in layer-5 cortical pyramidal neurons originate in the apical trunk. It is thought that these spikes are the product of colliding forward distal input and a back-propagating action potential generated by a somatic spike [89]. This collision results in a burst of neural responses. Together this suggests that colliding dendritic spikes allow the neuron to respond to relevant input arriving at different compartments.

Dendritic spikes can also play a local role within parts of the dendrite. The fact that not all dendritic spikes reach the soma suggests their impact is not neuron-wide. Furthermore, whether or not a spike reaches the soma depends on current circomstances. The extent of spike propagation depends on compartmentalization which can be dynamically regulated [71]. Because dendrites can partition input regions into electrically independent compartments, spikes that do no move from a specific region can still affect local processes, such as learning, without impacting other compartments [126, 8].

Compartment-level signalling suggests the somatic response is the product of a set of sub-patterns rather than a set of unique inputs. This, in theory, allows for a low-energy recognition of partial patterns that on their own are not significant to the wider neural network. In other words, at an elemental level the number of neurons does not need to equal the number of patterns. And not all patterns are created equal across all contexts.

2.3.5 Multi-tiered Plasticity

Dozens of learning models exist that partially describe neural plasticity and include phenomenological treatments as well as low-level processes. There are two models of plasticity that are most pertinent to our work. First, synaptic plasticity is the product of the timing of input and response *and* the local state of the neuron (e.g. voltage) [34, 35]. In other words, the interaction between input and response, does not happen in isolation from activity at sibling synapses. Second, neural plasticity is not confined to the synapse or even to the growth and atrophy of dendritic spines. It appears to be a multilayered

process. In additional to synapses, entire branches increase or decrease their effectiveness to elicit somatic responses. This dendrite branch-level learning is known as branch-strength potentiation (BSP) [100, 92]. This section provides a short overview of these two influences.

Synaptic plasticity is one of the most researched properties of neurons and is far too large a topic to cover here in any depth. Synaptic plasticity is thought to be a major component in learning and memory. The majority of approaches to synaptic plasticity utilized either in neuroscience models or within spiking neural networks are based on the relative spike times of pre- and post-synaptic neurons. If an upstream neuron spikes prior to a downstream neighbor within a small temporal window, the learning model assumes that upstream activity contributed to the downstream response. The intervening connection is, thereby, strengthened. This is referred to as Hebbian learning [74] or spike-time dependent plasticity (STDP) (for an excellent history of the subject, see [108]). Likewise, if the order is reversed, the connection is weakened. Famously, synaptic learning based on correlated activity was summarized by Carla Shatz as "cells that fire together wire together" [143]. Hebbian or STDP learning is the basis for many network learning models, and much of its attraction is due to it being an unsupervised process. However, STDP is inherently unstable because it rewards repeated activity with an efficacy for more of the same activity [116, 2]. Subsequent research has suggested modifications to this basic rule. For example, the Bienenstock-Cooper-Murnau (BCM) rule suggests that the thresholds for synaptic potentiation and depression shift based on the post-synaptic spike rate [19]. Recent work taking note of the frequency dependency of synaptic change and the interaction of spike triplets has produced a model unifying temporal coincidence and membrane voltage [34, 35]. Plasticity, like much else, depends on neuron type. There are instances where, *in vivo*, the STDP rule appears strictly violated and the sign of synaptic change depends on local dendritic dynamics [93]. Additionally, the presence of certain neurotransmitters and the dendrite-dependent back-propagation of the somatic response play a role in synaptic change.

Above the synaptic level, dendritic branches are thought to compartmentalize input into separable units. Research suggests that dendritic branches can exist in a "strong" or "weak" state allowing for more robust propagation of dendritic spikes over greater, forward (toward the soma) distances [100]. BSP is the result of a change in a K^+ channel responsible for the forward and backward propagation of somatic and dendritic spikes. Dendritic spikes in potentiated branches exert greater control over somatic membrane potential. This change is caused by correlated local and somatic activity.

Once elevated, BSP affects input arriving at all synapses along the branch and not only those synapses that caused the potentiation. If branches represent input patterns, then BSP allows the coupling of pattern-to-response irrespective of the individual inputs comprising a pattern instance. Pattern instances and the activity they generate are known to change over time, suggesting that activity shifts from one synapse to another even in the continued presence of the same pattern. BSP ensures that such shifts do not result in a failure of recognition at the neuron level. Modeling work based on Losonczy et al 2008 [100] showed that BSP might contribute to pattern separation and feature binding [92].

From the above discussion, the current work takes the view that dendrites are multi-layered learners. A multi-layered learner is one in which the mechanisms affecting neural change play different roles and can, to some degree, change independently. They are multi-layered because each mechanism alters the dendrite at different scales with some impacting specific locations and others impacting entire branches.

In general, synaptic plasticity refers to changes to a single input to the neuron. Its impact is local. If compartmentalization creates quasi-independent pattern detectors out of multiple synapses, synaptic plasticity would have the effect of altering a synapse's role within in recognizing the pattern. It determines how important a synapse is in triggering (or inhibiting) the pattern.

BSP, on the other hand, increases or decreases the ability of all signals originating within a dendritic branch to elicit a neural response. It might represent a

compartmental-level form of plasticity. If a dendritic compartment is in some way synonymous with an input pattern, then BSP could bind a pattern or collection of patterns to the current context. In this way, BSP does not impact the dendritic branch's ability to recognize a pattern. Rather, it temporarily alters the pattern's status within the larger dendritic network.

In summary, this multi-layered plasticity potentially allows for the learning of multiple patterns and switching between them without causing catastrophic forgetting.

2.4 Are We Neural Mathematics?

This chapter has tried to summarize the view that dendrites are themselves processing elements by highlighting several properties that support this theory. But before we move on, we must acknowledge that there is a crucial assumption hiding silently in this research. Looking at structure in the living tissue and trying to draw from it a model capable of reproducing its ability requires a language capable of describing the mind-matter (or meaning-matter) intersection. I am skeptical that the mathematical and logical language of AI as it is today can model this intersection. There is a strong overlap between what certain neuroscientists are trying to see in the brain and the struggle of AI researchers to explain why their tools do or do not work. I suspect the language of real intelligence is the physical brain. We speak to computers through interpreters capable of transforming human language into binary. To speak brain[1] , we need interpreters that move meaning between human language and neural structures.

AI is not the only field dependent on inadequate language. The history of research into dendritic processing also tends to use the language of mathematics and logic[22, 87, 99]. It is easy to create from the diverse research a panorama that portrays the entire brain, not to mention individual cells, as a deterministic machine. For example, the placement of synapses along the dendritic tree and the nature of each signal create relationships similar

[1] As dumb as that phrase sounds.

to addition, subtraction, multiplication and division or logical combinations of AND, OR and NOT. A partial reason for this retreat-into-math is that neuroscience lacks the experimental tools to develop a method for interpreting the compositional aspect of brain activity. The problem is too large, the scale is too small and the change in state too quick to take enough measurements at a moment in time to understand which micro phenomena compose which macro activity. So the computational nature of the brain has been the domain of computer models. It is hard to resist superimposing parts of the model onto the brain when the artificial robustly mimics the biological.

Ultimately, whether or not brain activity can be reduced, at any level, to a form of math is a contentious issue for those who study it. But one thing seems to be clear. Whatever individual neurons perform to create collectively our ability to perceive, think, remember and react, it is enabled by the parallel, distributed structure of the dendrite.

CHAPTER 3
BACKGROUND - ARTIFICIAL INTELLIGENCE

3.1 Overview

Unlike in neuroscience, dendrites in AI-related neuron-based learning are extremely rare. However, several works have cited dendrites as the basis for their research [50, 177, 84]. The scope of dendritic influence ranges from simple modifications of point neurons to novel systems. As inspiration they cite the purported computational abilities of dendrites, the benefits of deeper network topologies and, in general, the belief that an increase in biological fidelity is desirable.

What is missing from the literature is an AI perspective on computational properties of dendrites. In general, AI lacks a computational model of dendrites. To our knowledge, no AI work has proposed one or drawn one from the plethora created for neuroscientific research. Thus far AI-side research into dendrites largely has been grafted onto traditional approaches to neural networks. And one cannot look only at AD-themed work for answers. Works which do not carry the name 'dendrite' concern themselves with the same problems (e.g. pruning or network sparsity).

The AI-related work on dendrites thus far can be characterized as sparse and disconnected. All approaches have inherited something from general neural network research. Lateral borrowing between projects is all but nonexistent. This is a good thing! The voices are few but the approaches are many. While the goal of this dissertation is to add a model, it does not try to unify or end the search or select from among the implementations but rather find common elements. It is the belief of this thesis that AI research into dendrites is at an early stage when creativity and imagination are key. We should explore the space.

The following discussion on AI research is divided into topics that, in some way, relate to artificial dendrites. The origins of these topics are diverse. Some were selected because they are important to this work and some are derived from common topics or concerns within the body of related works.

The discussion is limited only to those works that state an interest in either dendrites or general ANN research with obvious relevant elements. When one looks at the computational nature of dendrites described by neuroscience research and ANN research, one sees many parallels. We cannot possibly point them all out.

For this reason, the history of artificial neural network research is omitted; there are better forums for it [179]. Certain types of networks that arguably could contribute to the discussion are also omitted to retain focus on specific properties. For example, long-short term memory networks use a complex chain of neurons and connections as their elementary construction for certain layers. There are some similarities between each layer's chain of separate computations and the concept of the dendrite. However, such comparisons are beyond the scope of this work as that would entail the comparison between two complex entities one of which is poorly understood. Lastly, there seems to be multiple research paths within neural network literature converging on a set of AD-related ideas. To this end, we acknowledge the likelihood that our field will arrive at a utilitarian understanding of dendrites without taking the direct approach advocated by this work.

3.2 Overview

Previous work has focused on dividing inputs into independent, sub-neural channels [152, 50, 177] or tree-like subnetworks whose shape is predetermined [177, 84]. Each channel or sub-neural node integrates its input independently from the others before passing the result forward to either an aggregate channel, node or the cell body itself. Local integration is nonlinear which models the idea that dendritic branches are isopotential and that the shape of the dendrite contributes to a signal.

Several projects have been successful in demonstrating this last point. Research by John Elias showed that reassigning inputs to different locations on a predetermined dendritic tree is sufficient to alter the neuron's response profile [50, 52, 51]. In Jones 2020 the order in which signals are integrated impacts network accuracy [84]. On the other hand, Wu 2018 randomizes the connections from the previous layer to the dendritic structure of the next [177] inspired by randomized attribute bagging [27]. These works draw inspiration from neuroscientific theories about input placement, tree shape, and signal integration.

3.3 Dendritic Shape

3.3.1 From Artificial Dendrites

The brain's dendritic diversity suggests no one shape fits all input. So we must begin by asking a basic question: which dendrite shape should we use?

Behind this question lurks another concerning how for some problem or problem domain we find an optimal shape. On the surface there are two answers to this question? First, we develop, through trial and error, a method for selecting a predetermined shape. Second, we develop a method to train shapes using problem data.

To our knowledge, only one work (other than ours) in the AI domain is working on the latter method [155]. The rest opt for the first method.

Before describing how related works have handled dendrite shape, it is important to note that other works (apart from Todo 2014) have not placed an emphasis on shape selection. While, in general, shape is identified as being an important consideration, the dendritic attribute driving previous work is not shape but input separation (see Section 3.4). This does not mean related works have not made decisions regarding shape--only that it is a secondary concern.

In Todo 2014 (one of several papers describing a similar approach to building artificial dendrites [155, 152, 131, 130, 164] that originates in [150]) neurons are expanded to four layers: synaptic layer, dendrite layer, membrane layer and somatic layer. The somatic,

synaptic and membrane layers together form an entity similar to a point neuron. Individually they fill the roles of transfer function, input summation and connection weights respectively. The novelty comes in the dendrite layer.

The dendrite layer is comprised of N dendrites. Each dendrite hosts some number of inputs (the synaptic layer). Dendrites use of a version of logical AND when combining signals arriving from the synaptic layer. Similar to the Identity Law of Boolean logic, synapses with a weight of 1 prune themselves. And, similar to the Null Law, synapses with weight 0 prune an entire dendrite. The combined dendritic signal is passed on to the membrane layer where dendritic output is summed. The number of dendrites and synapses per dendrite are hyper-parameters for a given neuron. The ultimate shape of each dendrite and the number of dendrites depends on a final pruning process, informed by synaptic weight.

The approach of Todo 2014 can accommodate any initial dendritic shape as dendrites are allowed to contain branches. Instantiating a dense forest of dendritic trees and connections and allowing a weight-focused training algorithm to find a suitable sub-tree is advantageous. Unlike a purely genetic approach or the approach taken by this work, pruning based on weights simplifies the training process. Dendrite shape does not depend on synaptic location or finding a method for connecting synapses. In fact, synaptic location within the dendrite is commutative.

Pruning connections based on weight to reduce the size of networks is common in ANN work. The primary benefit (at least in the context of a final pruning stage) is the production of a smaller, more efficient network. The final result is a sub-graph of the original. Their approach is also capable during post-training analysis of identifying inputs that are common to multiple dendrites. These can be used to redraw the dendritic tree to a more complex (but equivalent) shape. It is possible this could aid in explainability.

But there is one disadvantage that is inherited from all neural networks with static, hand-selected topologies. Network initialization must ensure the starting set of dendrites

and the placement of inputs contains an adequate sub-graph. For the experiment described in the paper, there are few enough inputs that all N (10 in this case) dendrites receive input from all sources. In this case, finding an adequate sub-graph depends on initialized weights rather than the selection of sets of inputs.

Wu 2018 ([177]) describes a dendritic neural network (DENN) in which dendrites comprise a single pre-layer to a neuron. Each dendrite hosts a randomly selected subset of the previous layer's efferent connections. Dendrites sum the input values received through each connection. Neural output is different; the neuron itself outputs the max value received from all dendrites.

Shape in this case is a tree of depth one, rooted at the soma, with d children. The authors equate the number of dendrites (or its shape) with an increase in network complexity over traditional feed-forward networks. Since each dendrite receives input from a subset of neurons in the previous layer it creates sparse representations in the hidden layers. In general, sparse representations duplicate individual inputs across multiple subsets of the full set of features. This has been shown to reduce catastrophic forgetting [97]. Typical forms of sparse representations (grouping by tile coding or radial basis) place an importance on locality. Locality is the idea that the information processed at a particular part of a network should share similar features. Locality has influenced our work as well. However, in Wu 2018, connections are randomly selected for each dendrite. Locality of information in the dendritic layer is left to chance.

The advantage of this topology is that it is amenable to existing ANN implementations. The shape does not contribute to a drastic increase in complexity since layer-to-layer connections are still all-to-all without redundancy. The limitation of single-layer equal-size dendrites prevents the DENN from exploring unbalanced or deep dendrites with multiple compartments. However, this limitation of balance is not endemic to the model but its formulation within the paper. There appears to be nothing preventing the authors from randomizing placement such that one dendrite receives more inputs than another.

Jones 2020 present a similar approach albeit taken to an extreme with respect to dendritic tree length [84]. They model dendrites as binary trees in which synapses only exist at the leaf nodes. Dendritic depth is $\log_2(N)$ where N is the number of synapses. Pairs of inputs are combined and passed through a ReLU transformation. Pairs of branch outputs are likewise transformed and so on until the final remaining two are combined at the soma (root). Dendritic neurons were also tested with multiple dendritic trees (k-tree) with k varying from 1 to 32. Results indicated that increasing k resulted in better performance overall and is comparable to a 2-layer feed-forward network where the hidden layer has $2k$ neurons. The initial pairing of inputs determines the chain of subsequent pairings and Jones 2020, interestingly, noted that pairings based on the natural locality of inputs (e.g., adjacent pixels) produced better results than random pairings. Shape, in this case, is determined by the number of inputs to a neuron.

The dendritic shape used in Elias 1992 ([50, 52, 51] is, like Jones 2020, a balanced binary tree. The dendritic tree used in [51] has a depth of 4. Synapses connect at predefined points. Each non-leaf branch has 8 excitatory and 8 inhibitory connection points where synapses can be formed. Leaf branches have 9 of each connection type. In total the dendritic tree has 256 connection points. The number of connection points exceeds the number of inputs to allow for many synaptic orderings. Only one shape appears to have been tested. This is likely due to the implementation being partially done in hardware.

The researcher does not explicitly state why this specific shape was selected. However, the text focuses on the number of synapse orderings over the tree. Since reordering connections forms the basis of the training algorithm, this shape seems to have been selected to increase the possible orderings without creating too large of a search space. Shape, then, is important as it increases the number of possible synaptic orderings. Increasing the range of synaptic orderings also increases the dendrite's capacity for multiple sparse representations. In other words, each dendrite (or dendritic compartment) by its shape and location is capable of learning one of a set of sparse representations.

Which one it receives can change. Lastly, it should be noted that this is the only related work in which synapse ordering along the dendrite matters.

Overall, previous work emphasizes compartmentalization over shape. The actual shape and depth of the tree seems to be a secondary concern or a result of the particular approach to compartmentalization or as a side effect of the chosen implementation. Certainly, there is wisdom in choosing the simplest shape when the role of any given shape is unknown. The chosen shapes are balanced which make for easier implementations. With unbalanced shapes one has to contend with the fact that, unless constrained, large dendrites will shout down smaller ones. Or distal connections will be lost along lengthy paths. Balance, then, is beneficial.

While dendrites are viewed as vehicles of sparse representations, the contribution of shape remains poorly understood. It is the opinion of this author that understanding the computational uses of shape in this respect will require the comparison of many forms: deep, wide, balanced and unbalanced. Shape, however, does not stand on its own; its effects depend strongly on how input is integrated within and across dendritic compartments and trees. This is the focus of the next section but first let us briefly walk through what traditional ANN work has to say about network shape.

3.3.2 From Neural Networks

While there has been little artificial dendrite-related work on selecting or training for a specific shape or family of shapes, there is a growing body of ANN work on selecting useful neural network topologies for a particular problem [119, 107, 115]. Nearly all ANN work focuses on the two most general topological aspects of a neural network: depth and width. Width corresponds to the number of neurons in a single layer and depth is defined as the number of layers. Network shape has been linked to network expressivity [132].

Expressivity is a theoretical measure of a network's ability to approximate a complex function. We can equate expressiveness with performance. Research has shown that a

2-layer network is universally expressive in that it is a universal function approximator[41]; however, such a shallow network, by today's standard, in order to be expressive enough to approximate a particular function, could need to be exponentially wide in the size of the input [102]. Whereas, the same function could be approximated with much deeper but narrower network. It is the general consensus that depth and width both share the burden of a network's potential [120]. Much of our insight into the depth versus width argument comes in the form of a search for dimension inequalities for a class of functions [49, 151, 94, 114, 128]. In other words, how wide must a network be to approximate the same functions that can be handled by some depth, and vice versa. The abstract, mathematical analysis of shape largely views networks as rectangles. More granular empirical insight is lacking therefore particulars are the result of trial and error.[1]

However, for certain special subtypes that contain distinct connectivity, there are additional shape-related hyperparameters to consider. Residual connective networks (ResNets) [73] contain connections that skip one or more layers. The number of bypassed layers as well as the number of connections becomes a topological choice. Convolutional networks (CNN) have layers whose individual elements examine a small section of the overall input, unlike fully connected layers [91]. For these, users must choose the kernel size ($N \times N$) to which to apply a filter. Filters themselves can be interpreted as a type of connectivity and therefore define a shape. Recurrent networks, of which there are many types, contain neurons that receive, perhaps indirectly, their own output. Like ResNets, recurrent connections are chosen during construction and, generally, not a part of training.

Shape, then, can be difficult to disentangle from other network properties that contribute to its performance. It is important, however, that we gain some granular insight. This is particularly true in light of research into identifying sparse networks. At this stage, it is not clear how applicable topology research using large, fully connected networks applies to sparsely connected AD neurons except to say, as in the case of the Lottery

[1]Also called 'Grad Student Descent' by this paper[62]. We are not amused and await the revolution.

Ticket Theory (see Section 3.7), that they guarantee the existence of a smaller network with some level of accuracy. The general advantages of depth and width are applicable to AD neurons. For example, Jones 2020 shows that increasing the number of independent dendrites (width) improves performance [84]. A study of how the different sparse shapes of pruned networks affect performance would shed some light on the subject but, to our knowledge, this remains an open question [96].

From a general network point of view, ANN variants capture just a small selection of all regular, symmetric network shapes, not to mention imbalanced, irregular or asymmetric ones. As previously stated, there will be overlap of the properties that make ANN and dendrite topologies useful. However, the difficulty in identifying those today, beyond the low-hanging fruit of depth and width, is that typically ANNs start densely and uniformly connected. In other words, they start by containing all shapes less than or equal to their current dimensions. Therefore, ANN topology research has focused on defining initial networks that are guaranteed to contain a network of smaller or equal size that can solve the problem. For the artificial dendrite to be useful (and not just a rehash of neural network pruning), it should be able to drastically constrain the size of its initial shape and rather precisely refine to a final one. Dendritic research, with respect to shape, should attempt to redefine neural networks as a set of connected, heterogeneous, sparse networks (i.e., dendritic neurons). Arguably, the most important contribution dendritic research can make to neural network based AI is give it the capability to innovate network shape, even at the neuron or micro level.

3.4 Quasi-independent Integration

Quasi-independent integration (QII) describes the idea that dendrites allow the neuron to combine signals arriving in close proximity on a branch or segment in some degree of isolation from signals arriving elsewhere on the tree. The qualifier *quasi* indicates that this isolation is not complete in the neurological setting but segments the dendrite into leaky

compartments. In practice, this leakiness manifests in one dendritic compartment integrating the output of other compartments. QII, then, turns the dendritic tree into a computational entity consisting of layers of parallel integrators. Quasi-independent integration is, arguably, the predominant idea motivating previous AI work on dendrites. So why is this important? And how has it been implemented?

Related AI work argues in favor of dendrites based on the importance of network depth and width in traditional ANNs. Network dimension allows for independent regions of computation and computational repetition. In overview, the argument is made that if network dimensions are important, then a neuron with one or more dimensions could be beneficial. The predominant method to give neural entities dimension is through some form of QII and connectivity. QII is important because it creates dimension.

Previous work also emphasizes the pre-somatic integration of subsets of inputs, i.e., compartmentalization. Parceling inputs into compartments increases the potential complexity of neural computation (see Section 4.6). In all these works, experimental evidence shows that a variety of approaches to compartmentalization can empower a single neuron.

Implementations of QII typically use a two-step process similar to the perceptron: a linear combination followed by a nonlinear output function. Todo 2014 (and related works) combines the output produced by compartments through an approximation of Boolean-OR via a soft-max function (Equation 3.1), where I is the resulting input given either to the soma or to the next compartment[155]. d_j is the dendritic output of the j^{th} dendrite. I, then, passes on the maximum output of all dendrites. Each d is the result of a soft-min (or Boolean-AND) of all synapses within a compartment. The authors state that their dendritic model is given by a kind of disjunctive normal form (DNF). As stated in the paper, having a DNF as a dendritic model has significant implications for the computational abilities of dendrites since they can approximate any Boolean function given

enough dendrites and synapses. This wide computational range depends on AND-OR (DNF) compartmentalization.

$$I = \frac{\sum_j d_j e^{v d_j}}{\sum_j e^{v d_j}} \tag{3.1}$$

In Hussain 2014, AD neurons consist of m soma-rooted branches[79]. Each branch has exactly k synapses. Signals within a branch are integrated as a weighted sum. These combined signals pass through a nonlinear transformation function, $b(x) = x^2$. The text states that the square of x, the branch output, was used because of it is easier to implement in hardware. A constraining factor in this work (similar to Elias 1992) is that it is or is designed to be implemented in hardware. In this work, the denritic neuron is shallow (one level of soma-rooted dendrites) but wide. The work shows that increasing k, synapses per branch, increases performance and states that an increase in m, the number of branches, does so as well; however, details about the latter, which is of more interest to this topic, are not given.

Jones 2020 shows that a mix of sequential and parallel compartmentalization leads to improved performance[84]. This work embodies an extreme form of independent integration in that it maximizes the number of compartments. Dendrites form a k-tree, where k gives the number of dendritic trees rooted at the soma. Each dendritic tree is itself a sparse ANN with the shape of a binary tree. Neurons are arranged in layers and each neuron represents a dendritic compartment. Each neuron in each tree receives input from only two neurons in the previous layer, except the leaf layers which receive two external inputs. This gradual narrowing of the layers creates a deep topology that is dependent on the size of the input vector. Their research shows that as k (the number of dendritic trees) increases so does performance. Significant differences in performance are best seen comparing $k = 1$ to $k = 32$, the largest network tested. First, this reinforces that composite signals, or signals comprised of earlier integrated signals, performed over the length of the dendritic tree, are important to an appropriate neural response. Second, higher values of k,

or more soma-rooted dendrites, produce better results. The paper infers that duplicate inputs spread across different dendrites has some benefit. While the exact contribution of multiple dendrites is not discussed, it seems likely that such a configuration gives the dendritic neuron a set of weights for each input that could prevent learning from being trapped in local minima due to initial values. In other words, signals arriving on independent trees form a quorum from which it is more likely the majority is correct. This is reminiscent of techniques involving multiple copies or voices that have been applied as meta-heuristics to ANN learning, such as Harmony Search[61] or consensus for distributed learning[95]. It also influenced our decision to allow for multiple, soma-rooted dendrites.

Wu 2018's approach to QII is similar to Jones 2020 in that it is based on a modified feed-forward neural network (FNN)[177]. Its depth, or vertical compartmentalization, differs in that it consists of one layer of dendrites. Each dendrite outputs a weighted sum of their inputs compartmentalized by a linear rectifier: $\sigma(x) = \max(0, x)$.

In summary, related work has used QII to give neurons both vertical (length of dendrites) and horizontal (number of dendrites and number of branches) dimension. Generally, the former consists of two steps: combination and transformation. The combination step is just a receiving step and is typically a simple sum., but it can also be more complex, such as a selection as in the case of Todo 2014. Transformation is an interpretive step applied to the combination. Since most implementations of artificial dendrites is derived from ANN research, transformation has been implemented using one of the transfer functions common to ANNs (ReLU or tanh). Horizontal QII, on the other hand, gives a neuron completely independent input channels. Jones 2020 uses them to duplicate signals. Todo 2014 uses them to permute signals.

3.5 Input Placement

With dendritic neurons, input does not necessarily arrive at a single location. This fact raises several novel problems concerning neural network instantiation. For example,

assuming we have a neuron with some number of dendritic trees, which inputs connect to which which tree? If a tree consists of more than one quasi-independent segment, which inputs fall into which segments? This is a hard problem because it requires we have an understanding of the meaning carried by an input and how it should combine with others. Neuroscience can answer this only generally in most cases. An implementation cannot live on broad strokes alone.

This section briefly covers how previous work approached this problem. In summary, related work places inputs using one of two methods: locality in the feature space or randomization. In this respect, our work is unique in that input placement is a product of training rather than random selection or pruning strategies.

Wu 2018 randomized input placement[177]. k outputs from the previous layer are randomly selected without replacement to be input for one of d dendrites. Each input, then, is unique to one dendrite. Jones 2020 tested their architecture on both connection strategies--locality and randomization--and found that the latter decreased performance[84]. Todo 2014 takes a selection-less approach; all inputs are duplicated across all dendrites[155]. Random initial weights are used to give each input a unique voice in each dendrite in spite of repetition. Hussain 2014 does not explicitly state the placement strategy used; however, a figure in the paper (Figure 1) suggests inputs are sparsely distributed and can make multiple connections[79].

At its core, placement is an initial value problem and certainly amenable to selection strategies used for other parameters, like weights. Some works show that placement can be avoided if the training algorithm has the capacity to silence individual connections and even dendrites by adjusting weights[130]. But this is only possible if a neuron begins fully and redundantly connected. So the degree of connectivity plays a role in placement strategy. Like many other aspects of the artificial dendrite, which strategy is selected

appears to be driven by certain underlying details, such as an implicit model dendritic computation and/or limitations of the training algorithm[2] .

The concerns of input placement do not end with the question: which input, which dendrite? In the general case, inputs share a dendrite with other inputs. Therefore, placement must concern itself with the effects of input-input interactions with respect to dendritic (or compartmental) membership and ordering. *Dendritic membership* is concerned with whether all inputs have an equal voice with respect to computing the output of a branch. *Input ordering* asks whether the order of arrangement on a branch impacts dendritic output. With the exception of Elias 1992, no work has explored whether the degree of dendrite (or compartment) membership or input ordering is important. Elias 1992 makes use of input locations along fixed dendrites. By altering the point of contact for the set of signals, their work shows that placement and ordering are enough to learn nontrivial tasks, such as MNIST digits. All previous and current AD models combine input signals arriving at the same dendrite associatively. In other words, membership to a specific dendrite is complete.

Understandably, the AI-related work on dendrites largely avoids the topic directly. At this early stage, there are no strong motivators for a particular placement strategy. Inspired by this point, we determined to develop a method to train input positions. In other words, let the problem dictate where they should connect. Almost any adoption of dendritic neurons by the AI community will require significant research into the effects of different placement strategies on performance and complexity.

3.6 Training

AD neuron train likewise faces additional concerns. This is not to suggest that AI-related dendritic work must address every concern or make all aspects of the dendrite trainable. These concerns fall into two general categories: attributes that impact the

[2]The work of this thesis is no exception.

dendrite and attributes that impact the input (or connection). For the dendrite, training affects the number of dendrites and their shapes (e.g., length, number of branches, etc.). For the input, training affects position and weight. But the two categories are linked. For example, input position can determine the shape of a dendrite (per the implementation in this thesis). Or, a fixed shape, as in Elias 1992, dictates where inputs can connect. The majority of related AI works begin with the dendrite because it simplifies the search space for input position.

In Elias 1992, training consists of moving inputs between predefined locations [51]. Although it is left unstated, Elias' work makes the case that input configurations over the dendritic tree constitute a very large state space due to nonlinearities between the predefined positions. Predefined locations reduce the search space. Training involves moving input connections from one pre-defined location to another. A connection list forms a network's code which is optimized using a genetic algorithm. A fitness for each population of connection lists is generated by comparing an ideal output against the actual output. Elias 1992 notes that, while the number of trees is $\frac{N!}{(N-M)!}$, where N is the number of connection points and M the number of inputs, due to the fact that convergent and divergent connections are allowed, the actual number is far higher. Interestingly, the trained network, capable of recognizing hand-written digits, utilized a high-degree of convergence; however, the author does not comment on the contribution of multiple sensors connecting to the tree at the same location. Elias' approach is an interesting one. A broadly tuned set of initial connection points could be used to identify a general input order. And final positions could be tweaked by a refining process.

No other related work, to our knowledge, shifts inputs from one location to another. Some shrink the set of input arriving on a branch by zeroing weights.

Dendrite shape is likewise not trainable in the related AI work. Todo 2014 allows for dendrites to shrink due to weight modification but they have no mechanism to grow or branch.

Weight modification strategies include standard back-propagation used in typical ANNs (Jones 2020), the absence of weights altogether (Elias 1992) and weights modified by evolutionary means in tandem with other network parameters (Wang 2020).

Synaptic and dendritic training concerns are linked. A good example of this is in the question of whether the dendritic tree impacts the training of weights. Jones 2020 and Wu 2018 subject the weight training signal to the shape of the tree; however, since both implementations are directly or strongly based on sparse ANNs as dendrites, weight training can use the same methods used by ANNs: back-propagation. Similarly to Elias 1992, Wang 2020 uses an evolutionary algorithm to train network parameters. The authors make use of *differential evolution* to train three critical parameters: weight, threshold and a value governing how responsive the synapse is to input. Differential evolutionary algorithms use single dimensional changes within a possibly multidimensional search space to improve a population of candidate solutions. If the change results in improved performance, the *agent* moves to the new place in the search space.

Weights are the primary vehicle of neuron and network learning in AD neuron research. This is the approach with which we are most comfortable and for which we have the most tools. But it ignores the big addition AD neurons bring to neural networks: non-associative, distributed input. The need to simplify the search space or only train one attribute at a time is certainly beneficial for complexity reasons; however, selecting weight as the initial candidate is, in the estimation of this thesis, the wrong one. Each related work, in spite of their differences, suggests that shape and input placement play a role in dendritic computation.

3.7 Sparse Networks

Formulations of the AD neuron borrow heavily from ANN research. For example, several of the related works use specialized, multilayer ANNs to represent AD neurons. There are trends in ANN research that appear to be converging toward certain properties

endemic to AD neurons. Specifically, this is true for *sparse networks* or *sparse connectivity*. Generally defined, sparse connectivity describes a network where neurons in the i^{th} layer do not receive input from every neuron in the $(i-1)^{st}$ layer. This short section covers the idea of sparse networks and how they compare to the AD neuron.

Superficially, sparse connectivity has nothing to do with the neuron model itself. From the point of view of the neuron, the input it receives is a topological concern and what is done with that input is the domain of the neuron. In other words, sparse connectivity is a network property. However, previous work on artificial dendrites have taken the view that a single AD neuron is better matched by a network of points neurons [141]. If this view is correct, the impetus for and benefits of sparse connectivity in ANNs might provide clues about the benefits of the artificial dendrite. First, why are sparsely connected ANNs attractive?

Training neural networks is a time and therefore energy intensive process[148]. Given that the majority of the world's energy is produced by nonrenewable, carbon dioxide emitting sources, high energy AI is undesirable. More complex tasks typically require larger networks[154]. Large, dense networks have millions, if not billions, of connections. Training involves updating each connection millions, if not billions, of times. This does not end with training (as researchers often forget). Once the trained model is deployed, every decision made by every instance of the model must again pass input through the network. Large, dense networks in their current forms are environmentally unsustainable and these monetary and environmental costs limit research and deployment. Interest in sparse connectivity, then, is driven partially by the desire to reduce the rate of network growth with respect to problem complexity[78, 5].

Networks can be altered such that a reduction in size still produces the same results. Research has shown that fully connected networks are overparameterized[67]. Overparameterization simply means that the model contains more information than is needed. Parameter reduction comes in several forms but *pruning* is most relevant to this

discussion as it produces subnetworks that when viewed on their own resemble AD neurons[3] . Pruning removes connections from the trained network whose weights have become zero[90]. Trained networks typically contain a substantial number of zero weights. Additionally, nonzero weights can be pruned through an ancillary calculation that encourages connections that do not strongly contribute to the outcome to go to zero. This is referred to as *regularization*.

Regularization pushes networks toward simpler solutions (or solutions whose fit can be approximated with a minimum order polynomial) and away from overfitting to a specific data set, i.e. generalization[183]. Sparse topologies have also shown an increased ability to generalize in some cases[101, 31]. Pruning is typically performed after the training process and therefore makes only the final model more efficient.

Pruning research has produced what is called the *Lottery Ticket Hypothesis* that states:

> A randomly-initialized, dense neural network contains a subnetwork that is initialized such that--when trained in isolation--it can match the test accuracy of the original network after training for at most the same number of iterations.[58]

In other words, a dense network contains a much smaller network with the same capability. This smaller network can be very sparse compared to the original, pruned by up to 98.5%. Identifying winning tickets can profoundly reduce the costs of a deployed network; however, there is a catch on the front end. Although work on identifying when (as early as possible) during the training process winning tickets can be selected, they cannot be discovered except through training the dense network[59].

Perhaps a dendritic approach to network instantiation could help identify the rough shape of these winning tickets. Being able to narrow the possible topologies could reduce the resources required by the training process. For example, dendritic branches that do not

[3]There are other forms of parameter reduction such as quantization which reduces the number of bits used by each weight.

contribute to neural output can be removed or disabled during the training process. If dendrites do in fact equate to one or more traditional point neuron layers, then a network will require fewer layers to accomplish the same task.

3.8 Biologically Plausible Credit Assignment

A second line of research that has experimented with artificial dendrites involves the search for an alternate form of credit assignment. Credit assignment is the problem of identifying and altering upstream contributions to a downstream result. Neural network research has spent significant effort pondering the lack of a direct biological equivalent for weight transport, also known as back propagation. Weight transport solves the problem of credit assignment by sending a non-local signal back through the entire network (dendrite-to-axon). There is no evidence from neuroscience to support the idea that higher order neural signals involved in determining the correctness of a response are sent *directly*, in a chain-like manner, back through the brain to the point of input [16]. This problem has motivated several researchers to ponder whether weight transport requires a separate feedback pathway, or whether the error signal can be treated as input in its own right using artificial dendrites [134]. The approach to weight transport has not influenced the work of this thesis; however, the idea that dendrites represent different, independent functions has.

Guerguiev et al., 2017 [64] suggests that weight transport can be solved by partitioning neuronal input into a two-part dendrite consisting of an apical and basal compartment. The basal compartment relays sensory information; whereas, the apical compartment carries a feedback signal used for credit assignment. In other words, basal input receives the preceding layer's interpretation of the input based on current weights. The apical input receives an ideal signal, or how the network should respond to a given input. The MNIST handwritten image data set was used to train their network. Each image passes through three phases. During the first phase, beginning at t_0, the image is presented to the network's 784 inputs and the network is allowed to respond based on current weights. At

t_1, a plateau potential (the integral of membrane voltage from t_0 to t_1) is calculated based on the feed-forward network activity. During the second phase, the correct output neuron receives external input driving it to fire at a maximum rate. All other outputs are minimized. At t_2, the end of the second phase, a second plateau potential is calculated. The difference between these two potentials is then used to define target firing rates for the hidden layer, and the output layer's target firing rate is the difference between the firing rates in phases one and two. These differences define the loss function for each layer and determine the final weight modification.

Learning is based on feeding optimal results back into the hidden layer via a second input channel. The apical input, transmitting perfect downstream information, allows for local loss computation. It offers a method by which feedback enters the neuron and concludes that separation between feed-forward and feedback signals is necessary for deep learning. While Bittner's research [21] on plateau potential driven weight increases supports the use of apical dendrites in learning, a gap left in the discussion is in how apical input alters basal weights. Research suggests that distal apical tufts can cause somatic burst firing which, in turn, can cause basal depolarization (see [134] for an overview and in depth references). As their focus is on creating a more biologically plausible method for back-propagation, the connectivity and placement of synapses is uniform across all hidden layer neurons.

The evidence that back propagation, as used in deep learning, is not biologically feasible is the motivation stated by Sacramento 2018 [136, 45]. Their model has similarities to the network and dendritic topology of Guerguiev et al. 2017 [64]. To inform upstream connections about downstream outcomes, neurons are given three compartments: apical and basal dendrites and the soma. As in [64], apical dendrites integrate feedback and lateral information while basal dendrites handle feed-forward signals. Using a three-layer network, with one hidden layer, learning is directed by a teaching signal originating in the output layer. The teaching signal is fed back into apical inputs to the hidden layer. This

forms an error signal for the somatic response. Weights between the input and hidden layer are adjusted using this error signal. Unlike [64], learning is continuous and does not require separate forward and backward phases, or the calculation of average firing rates. Leaky, current-based models were used for the experiment.

The goal of our work was not to find a biologically-plausible credit assignment. A better understanding about how the brain performs error calculations will undoubtedly improve our use of neural networks. However, these two studies (Guerguiev 2017 and Sacramento 2018) taken together suggested something more general to us. They suggested that dendrites can provide regions of independent computation. Independent computation can accommodate error signal integration but it could also be used in a more general sense. Dendrites (or branches) could introduce subtle pattern variations or multiple types of the same pattern.

3.9 Signal Propagation

Thus far we have focused on the effects of carving up dendrites into a connected set of quasi-independent signal integrators. And we have said nothing about what happens either at the boundaries of these segments or between them. AI related work has said very little about them. Todo 2014 selects the maximum signal from all dendrites as the signal to move into the next segment or soma. This suggests dendrites should possess a winner-takes-all ability. Implicitly it suggests that multiple dendritic-level signals do not integrate but experience a form of signal saturation.

Only Elias 1992 acknowledges that the effects of signal propagation could be important. In this work, inputs arrive at different preset locations on the dendrite. The structure of the dendritic tree resembles a binary tree; therefore, connections made toward the leaves of the tree are farther from the neuron than those at the top. The paper demonstrates that the implementation is capable of reproducing Rall's early study on directional selectivity[162], and target recognition is based on a threshold that appears dependent on

some degree of distal-to-proximal activation. Propagation delay is a necessary component for recognition. Unfortunately, Elias 1992 does not contain information about either the actual delays or tests in which they were varied. Of the three representations for delay mentioned above, dendritic propagation delay seems only able to force coincidences at a micro-time scale based on wire length and propagation speeds.

Other works use either weighted or unweighted connections between pre-somatic layers or pre-somatic layers and the soma. The method by which traditional ANNs handle the effects of signal propagation is strongly evident in this as the practice seems to be borrowed without discussion. Practically, this makes sense, but it does gloss over the fact that the same computational model is being used for two types of connections: synaptic and dendritic. Neuroscience and AI seem to agree that a decent, entry-level model of the synapse is a scalar, but there is no consensus on the latter.

There are other methods to model signal propagation. Biological dendrites leak current into the surrounding fluid; therefore, a signal traveling along a dendrite can be modeled by a rate of decay. Another way to interpret signal propagation is by the use of time. The time it takes for a signal to move from one place on the dendrite to another could have computational significance. Our implementation uses this latter method to determine the effects of dendrites.

Finally, signal propagation, specifically the idea that time carries meaning, is a great influence on our work. Our implementation of the AD neuron and our method for training its properties depends on the timing of signals arriving at or moving along the dendrite. Before moving on to the model, we would like to note one work in particular, although not necessarily AI-related, that influenced our handling of propagation delay.

Smith 2018 proposes a *Space-time algebra* to aid in modeling temporal networks. Temporal networks pass discrete events from one element to another and are not limited to neural networks. Smith's formalism is built around the *space-time function* (s-t functions) that satisfies three properties. 1) S-t functions are computable. 2) They obey temporal

causality (later events cannot change earlier results). 3) Results are invariant to temporal shifts. For temporal networks, input is transformed into a series of spikes. Likewise, output of the network is converted back into domain-specific values. However, within the network, the only values present are temporal differences. Functional units (i.e., neurons) accept temporal delays as input and produce delays as output. Delays are based off some fixed moment in time. Computable s-t functions, then, are necessarily bounded in that they limit delays to some fixed window of time.

In Smith's own words, "the time it takes to communicate a value *is* the value, and the time it takes to compute a value *is* the value (emphasis his)". This idea that the delay is the value ties in with the general approach taken by this thesis that a dendrite is a spatial embodiment of a temporal pattern. The sequence (or 'volley' used by Smith) of spikes determines local intensities and the neural response. There is something very profound in this idea, that Smith summarizes nicely:

> Evolution has shown itself to be a very clever engineer across a broad range of biological systems. And, it seems compelling that a clever engineer might use the flow of physical time as a resource because it possesses some ultimate engineering advantages: the flow of physical time is free (it happens whether we like it or not), it requires no space, and it consumes no energy[146].

3.10 Summary

In a nutshell, AI work on dendrites takes its cues from current work on state-of-the-art ANNs. These works are based on the general idea that deeper topologies improve performance. While this leading statement is an over simplification and does not take into account, for example, overparameterization and sparsity, its root idea is *the* driver in AI's exploration of the dendrite. What is missing, however, is the possibility for AD neurons create even greater and more diverse forms of depth.

The low-hanging fruit of dendritic shapes appears to be a single-layer of fixed input compartmentalization prior to the soma. Such an expansion of the basic perceptron model has a low overhead compared to more elaborate dendrite-like tree structures. Training can still take advantage of back-propagation or other ANN learning techniques. But previous work has not demonstrated whether dendrites are a new kind of depth or just more of the same. Related work has not demonstrated the benefits of input placement or compartmentalization. These works are primarily trying to answer an unstated, preliminary and very necessary question (at least with regards to computing): 'how can this be done?' Although it sounds blind and without a hypothetical basis, it is required to answer, 'is it useful?'

AI work on artificial dendrites is in its infancy. The majority of related works on the subject date from the last few years. Very few, if any, fundamental questions, given by the topics in this chapter, have concrete answers. There is as yet no comprehensive model of dendritic computation from either the AI or neuroscience side. AI work has focused on applications rather than methodologies and abstractions. This is due, in part, to the complex, multi-layered computational descriptions of the biological source. Neuroscience has many rich descriptions of dendritic learning, computation and behavior; it has built detailed simulations of single dendritic neurons down to the smallest distal tuft, but it has not produced a unified computational model that can guide an understanding of the dendrite's role across all neuron types.

Scattered approaches to dendrites within AI and the focus on real-world application is also due to the expectations and bias currently within the AI community for results and actionable implementations. Again, research that simply posits 'how' without positive results is less publishable[109]. The results must carry (justify) the research. This is not meant to be an outright condemnation but a characterization that explains why an infant sub-field (AD neurons) is cloaked in the premature question, 'what can it do?' than 'how can it be done?'. Surely, there exists a form of scientific technical debt in judging research

solely based on whether it solves some benchmark task no matter how opaque the solution[62]. Of course, one can also argue that if an approach produces good results, it will generate follow on research into the nature of those benefits. After all, we have made strides in explaining the workings of ANNs. Much of our understanding has come after a demonstration of usefulness.

The approach of this thesis follows these related works in several ways. It is exploratory; therefore, it is concerned more with 'how' than improving current results. It uses biology as a kind of proof that point neurons are not the end of the road for artificial neuron models. However, it differs in one important way. Rather than focus exclusively on an implementation, it focuses on how neurons can be modeled more abstractly. This is the topic of the next chapter.

CHAPTER 4
ARTIFICIAL DENDRITIC MODEL OF COMPUTATION

4.1 Introduction

This chapter describes an artificial dendritic model of computation. Formulating artificial dendrites as a model of computation (versus an implementation only) provides a blueprint for future implementations and provides a language to compare it to other models.

All computational models highlight or capture different properties of a particular system. The artificial dendritic (AD) neuron model places an emphasis on the dispersion of inputs over a dendritic tree which combines signals hierarchically and nonlinearly. It assigns a meaning to the separation between input positions caused by this dispersion. It defines how separation affects the integration of signals arriving at different points. It specifies a set of constraints on the shape of the tree. Our model attempts to capture the computational significance of the separation between connected points on the dendritic tree.

The AD model highlights two computational aspects of the biological dendrite: compartmentalization and separation. Dendrites can maintain varying electrical potentials at different points; i.e., they are not isopotential. This is thought to divide the dendritic tree into compartments that form quasi-independent computational units. The process of dividing the dendritic tree into independent units is *compartmentalization*. Within each compartment, synaptic inputs interact more immediately and linearly, whereas signals between compartments are subject to nonlinear transformations [17, 127, 70]. The degree of these transformations depends on the characteristics of the dendrite (e.g., length, diameter, ion channels, membrane capacitance, etc.). A full biological simulation of the dendrite is (at the present time) of no use to AI-related work. It must be simplified. Therefore, we group these characteristics under a single term, *separation*. The degree of

separation between two compartments (or more generally, two locations on the dendrite) describes the nonlinear transformation that takes place when a signal moves between them. In describing the nonlinear transformation, separation also describes each one's computational independence or lack thereof.

The rest of this chapter gives a high level overview of the AD neuron and how it uses the ideas of compartmentalization and separation.

4.2 Model Overview

The dendritic model is made of *compartments* and *connections*. A compartment is a disjoint subset (or partition) of a neuron's set of inputs. A connection describes an input-output relationship between two compartments and models their separation. Putting these together, an AD neuron is a set of connected compartments. Within compartments, input signals combine linearly, whereas, between compartments signals undergo a nonlinear transformation. The nonlinear transformation of a particular signal is determined by the separation between compartments.

A comparison to the more common point neuron model might help elucidate the additional complexity of the dendritic model. Figure 4.1 shows the main difference between the dendritic and point neuron models. In both figures, two inputs (A and B) arrive at a neuron. The left figure depicts a point neuron. Point neurons sum the values of all inputs. This linear combination models an identical point (or compartment) of arrival on the neuron. The right figure depicts a dendritic neuron. Signals arrive at different points or different compartments. The separation (dashed gray line) between these locations creates a new, potentially more complex relationship (indicated by the question mark) between the two signals. The dendritic model attempts to define this new relationship determining how each signal, arriving at different locations, interacts.

Unlike previous work on artificial dendritic neurons, our model is versatile in that it does not define the artificial dendritic neuron around one specific shape. Our model is

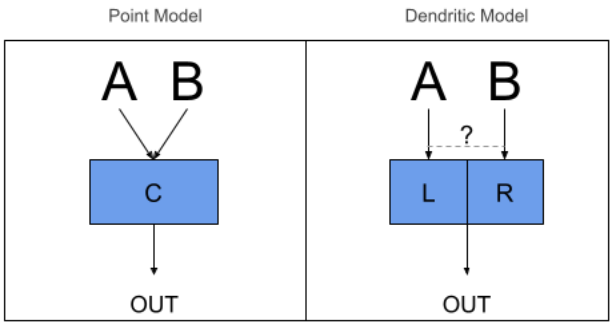


Figure 4.1: Point model versus the dendritic model. A and B are inputs. Arrows show the flow of information. Blue rectangles, C, L and R, are neurons (point neuron) or compartments (dendritic neuron). Dashed gray line and question mark indicate that the dendritic neuron uses separation to define the relationship between both input positions on the dendritic tree and dendritic compartments.

composed of reusable and repeatable elements--compartments and connections--that allow it to take on many different shapes without losing the ability to define how signals interact. In this example, the two compartments, L and R, appear to be equal (same size, same position) with respect to the output and each other. But we can imagine a different arrangement in which L and R are not equal in some aspect. One might be closer to the output than the soma. One might be the point of arrival for more signals than the other. One might have more influence over neural output than the other.

In the rest of the chapter, we first describe how dendritic neuron behavior differs from point neuron behavior using a metaphor involving a more familiar system: a lock and key. Next, we present the AD neuron model in its basic form for clarity. Finally, we give an extended version that fleshes out the model for use in a trainable implementation, and we describe the computational impact of the extended model.

4.3 Neurons and Inputs as Locks and Keys

Biological and artificial neurons are frequently described by an all-or-nothing behavior. This description says that neurons respond to correct input and not at all to incorrect input. Correct/incorrect input can be defined differently for each situation. It could be the magnitude of input, the rate at which input arrives or the absence of input. While many

models of neural behavior contain a gray area of partial response, all-or-nothing behavior is the predominant one and it is usually implemented using a single threshold. If the input is above the threshold, the neuron responds; otherwise, it remains silent.

To help us understand the differences between the dendritic and point models, it might be profitable to expand this description into a metaphor: the neuron and input as lock and key. A lock is a prime example of all-or-nothing behavior. If one applies the correct key, the lock opens. If one applies an almost correct key, it does not open. The pins must be at the right height or a series of digits must match value and order.

The point neuron is like a lazy combination lock that will open so long as the sum of the current digits exceeds some preset value. Set every dial to nine and voila! The dendritic neuron model proposed here is like a real combination lock. Larger numbers are not more correct. It will only unlock if each dial matches some preset value. However, this metaphor only describes one compartment of the dendritic neuron, thus the metaphor is actually more complex. Each dial itself can be another combination lock and the child lock contributes a single digit to the parent lock. This child-parent relationship can continue down to as many levels as desired. Each parent is unlocked by the unlocking of all of its children.

The dendritic neuron as combination lock is visualized in Figure 4.2. Left and right figures depict the same combination lock. The left lock is open because all actual input (top number in each green box) matches the expected input (bottom number). The right lock is closed because two input sources sent an incorrect value. Top left received a 2 when it expected a 3. This mismatch propagates all the way to the root. And middle right expected a 1 but received a 9. This shows how even a partial mismatch deep in the dendritic tree can keep the lock from opening.

Our point and dendritic neuron locks behave differently. Figure 4.3 depicts the differences in behavior of these two neuron models. The point neuron begins to unlock once the input vector reaches a threshold. The dendritic neuron is capable of defining multiple,

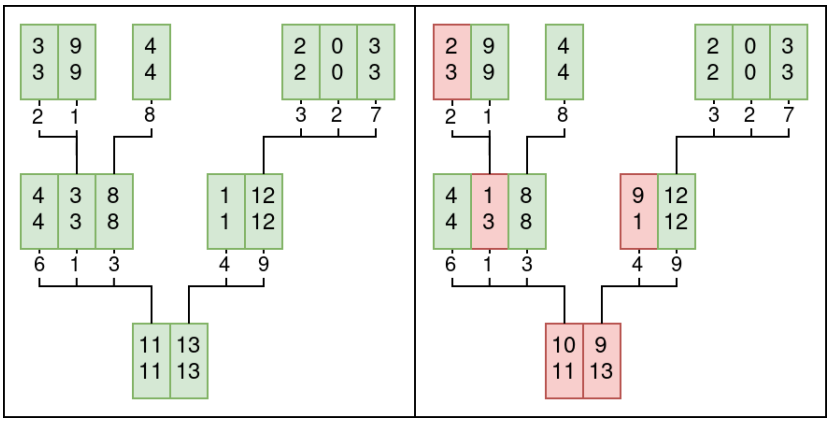


Figure 4.2: A dendritic neuron represented as a multi-level combination lock. Each box is a dial. If the upper value (the actual input) matches the lower value (the expected input), the dial emits a 1, else it emits a 0. The output of each dial is scaled by a weight, the value along each line connecting the dials. Non-leaf dials only unlock (emit a 1) when all their children unlock. Green boxes are unlocked. Red are locked. LEFT: All values match; the lock opens. RIGHT: Two leaf values do not match. A 3 was expected in the top left but it received a 2. This error cascades into the next series causing a mismatch (1 instead of 3) and finally to the root which expected an 11 but received a 10. On the right, a 1 was expected but a 9 received which caused a mismatch of 9 to 13.

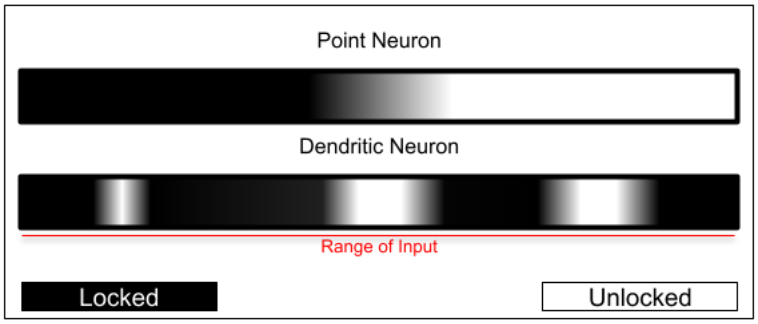


Figure 4.3: One dimensional representation of input vectors as keys.

non-contiguous sub-ranges in which the neuron unlocks or activates. This figure shows the problem along a single dimension and is solely for the purpose of visual representation. In reality, the dendritic neuron's key is a hierarchy of multidimensional keys.

The benefit of the dendritic model when viewed through this metaphor is that it more precisely defines which keys can and cannot unlock it. It can define one or more 'Goldilocks' ranges and each range can be as wide or narrow as necessary. For example, Figure 4.2 requires perfectly matching values but ranges could have softer bounds thus

allowing for partial unlocking. Another option is a parent compartment that only unlocks if some subset of its children are unlocked. For example, if the weights from the top right cluster were altered from 3-2-7 to 12-6-6, then the parent would unlock if the left child and no other unlocked or if the middle and right children unlocked together but not the left.

With this metaphor as a guide, let us examine the actual model.

4.4 Simplified Model

The AD neuron model is comprised of two fundamental elements: compartments and connections.

4.4.1 Compartment

The dendritic neuron model is a connected network of compartments (Figure 4.4). Conceptually, compartments model sub-trees with the dendritic tree. Figure 4.4 shows how compartments form an abstraction of the dendritic tree by grouping branches into separate entities. Because compartments represent swaths of the dendritic tree, they can encapsulate the location of one or more inputs to the AD neuron. Encapsulated locations are combined linearly. With this in mind, here is the formal definition of a compartment.

Definition: A *compartment* is defined by the function:

$$v_a = \psi(z), \text{ where } z = \sum_{i=0}^{n} z_i \tag{4.1}$$

v_a is the output of a dendritic compartment. ψ is a compartment's output function and z_i are the inputs terminating in the compartment. ψ is not strictly defined by the basic AD neuron model because it is a placeholder for any activation function.

By this definition, the dendritic neuron compartment is identical to a basic perceptron or point neuron which outputs the sum of its inputs modified by a nonlinear activation function. There is one clear distinction between the perceptron and the dendritic compartment. A perceptron describes a whole neuron, whereas, a compartment is capable

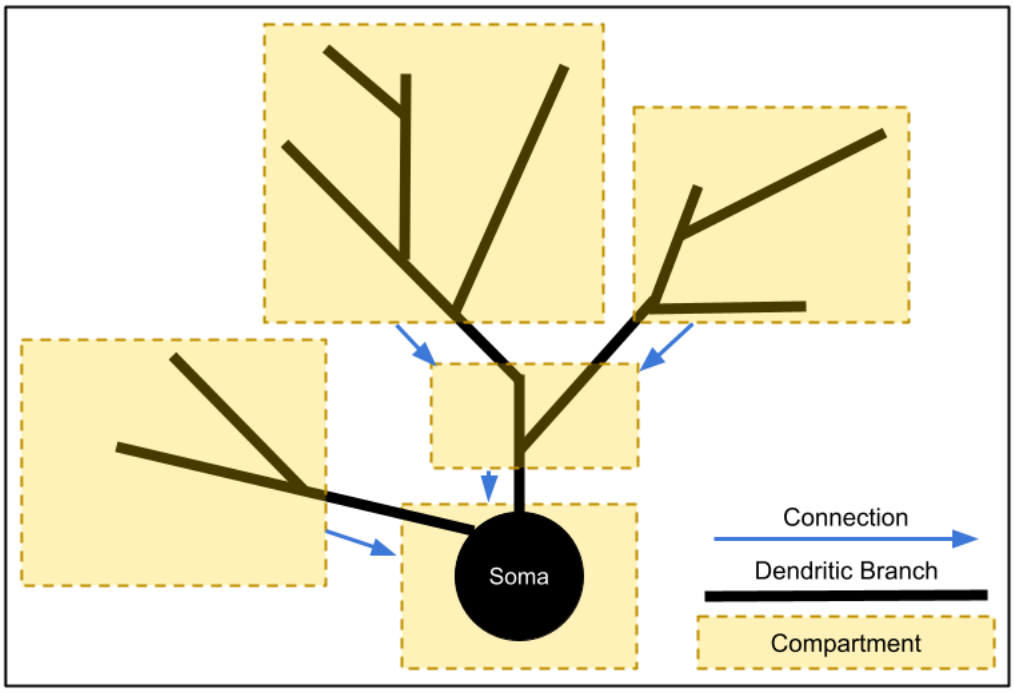


Figure 4.4: Compartmentalization of the dendritic neuron. A dendritic neuron (black) is broken into several compartments (dotted yellow). The underlying shape of the dendritic tree is used to create matching connections (blue arrows) between compartments.

of describing the whole neuron or just a part of it. The effect of this is: whereas a perceptron receives all inputs to its dendritic field, a dendritic compartment can receive just a subset of the inputs. This is an important distinction that will be better understood once we have defined the rest of the dendritic neuron model.

Compartments, or the compartmentalization of a dendritic tree, respect three rules:

No-skip Rule: If a compartment, C, contains two input locations, i and j, then there does not exist a k between i and j along the dendritic branch such that $k \notin C$. In other words, compartments cannot skip over input locations. An addendum to this rule is that if two inputs share the same location, they are also in the same compartment.

Order Rule: If a compartment, C, contains two input locations, i and j, and the distances between them and the soma is d_i and d_j such that $d_i > d_j$, then $\hat{d}_i \geq \hat{d}_j$, where $\hat{d}$ is their distance-to-soma after compartmentalization. Or to put it another way,

compartmentalization is order preserving. The compartmentalized tree forms a non-strict partial order of the AD neuron's inputs.

Intersection Rule: Let i and j be two inputs to the AD neuron located on different dendritic branches. If a compartment, C, contains the two input locations, i and j, such that $i, j \in I_C$, the input vector to C, then I_C must contain all inputs on the dendrite between i and j. To find all inputs between i and j we follow the dendrite from one to the other. All inputs encountered must be in C. This prevents a compartment from including multiple branches of the dendritic tree without also including the intersection of those branches.

These three rules insure that compartmentalization, discussed in Chapter 6, respects the ordering of inputs and shape of the tree.

4.4.2 Connection

Connections link compartments by specifying which compartment's output becomes another compartment's input. Whereas compartments define an input-to-output relationship, connections define the output-to-input relationships between compartments. As this implies, connections are directed. Figure 4.4 shows how connections (blue arrows) link compartments creating an abstract version of a more complex underlying dendritic tree.

Connections are not just dimensionless links between compartments. They store data about the separation between the compartments. In the opening section of this chapter, we stated that the degree of separation between compartments describes the transformation of a signal passing from one compartment to another. Here we clarify how separation transforms signals with respect to the dendritic neuron.

When v_a, compartmental output, moves from one compartment to another, it passes through a connection. If v_a matches the separation stored by a connection, the connection emits a maximum input to the receiving compartment. When there is a significant

mismatch, the connection is silent. In this sense, a connection has an expected value or range of values similar to the combination lock.

Definition: A *connection* is defined by a 4-tuple (c_i, c_j, v_e, ϕ).

c_i and c_j are dendritic compartments which define the start and end of a connection. c_i is the compartment whose output is input to the connection and c_j is the compartment which receives the output of the connection. From this, we say that c_i is adjacent to c_j. v_e is a value in $\mathbb{R}$ that measures the separation between the compartments. For the model, v_e is unit-less. With respect to an implementation, v_e can take on the unit of measure most applicable.[1] One good choice for ϕ is a radial metric function, such as a radial basis function (RBF).

A radial basis function (RBF) maps from $\mathbb{R}$ to $\mathbb{R}$ using a metric or distance between its input, x, or the *actual value* and a fixed value, c, or the *expected value*. RBFs produce a maximum value (typically 1) when $x = c$. As $||x - c|| \to \infty$, the output of an RBF approaches a minimum value (typically 0). There are many examples of RBF functions. Figure 4.5 provides a graph of a simplified version of the function used throughout this research. In the dendritic neuron model, v_e is substituted for c and v_a, as the input to ϕ, is substituted for x.

ϕ defines the effect the connection has on a signal passing from compartments c_i to c_j. We restrict ϕ to be an RBF-like function to keep the door open for ϕ to use difference-measuring functions which are not strictly RBFs. While RBFs have proved useful in this research, we have no reason to claim that RBFs are the best suited for modeling the effects of compartmental separation.

Equation 4.2 shows how the difference between v_e and v_a forms the input to ϕ. This creates a metric for judging how well values passing through the connection match the separation between compartments.

[1]If input to the model is rate-based, v_e would model the rate of compartmental output. If input were spike times, then v_e could be propagation time. In a biological scenario, v_e might be a measure of electrical current.

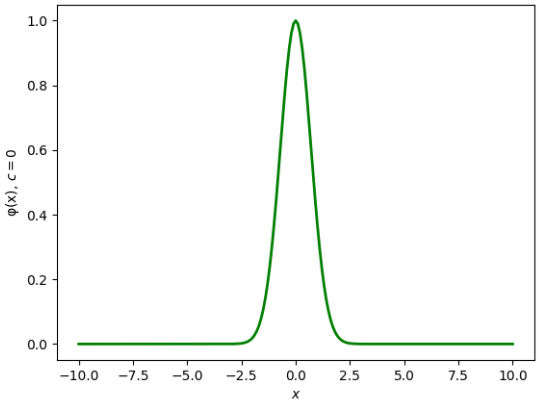


Figure 4.5: Example output of the inverse quadratic radial basis function: $\phi(x) = \frac{1}{1+(x-c)^2}$. The center, c, is at origin.

$$O = \phi(||v_e - v_a||) \tag{4.2}$$

Another way to think about separation and the output of a compartment is to see them as a connection's expected value, v_e, and the actual value, v_a, sent to it. In fact, a common theme in neuroscience research is that, at least in part, neural learning is predictive. Neurons are conditioned to expected certain input and learning (modifications to the neuron) come as a result of comparing actual input to expected input [33, 140, 103].

Throughout the rest of this work, we will refer to a connection's *region of activation*. A region of activation describes the values of v_a for which a connection produces a non-minimal input to the next compartment. In Figure 4.5, we might identify the range $[-2.5, 2.5]$ as the connections region of activation. What constitutes 'non-minimal' is arbitrary and is not intended to suggest that small input values are unimportant.

4.4.3 Metric Spaces

v_e and v_a are defined within the context of one or more metric spaces. A *metric space* is an ordered pair (M, d) where M is a set and d is a metric on M, typically a function where $d : M \times M \to \mathbb{R}$. Three axioms of identity, positivity, symmetry and subadditivity govern the behavior of d for metric spaces [170].

1. $d(x, y) = 0 \quad \Leftrightarrow \quad x = y$

2. $x \neq y \quad \Rightarrow \quad d(x, y) > 0$

3. $d(x, y) = d(y, x)$

4. $d(x, z) \leq d(x, y) + d(y, z)$

where x, y, and z are points in M. d can be single- or multi-valued depending on the utilized space.

The dendritic neuron model uses metric spaces to measure the separation between compartments. Dendritic neurons or networks of dendritic compartments are free to define multiple, independently anchored spaces. Measures of separation between spaces can be irrelevant, arbitrarily determined or utilize a d drawn from some hyperspace. A *hyperspace* is simply a space that contains other spaces with respect to some reference point; however, the local spaces do not need to conform to the containing hyperspace. This allows for the possibility of independent structures within a network, such as neurons, layers or regions. Local- and hyperspaces may require different types of spaces. Figure 4.6 gives an example of multiple spaces. In this example, the use of local and hyperspaces is less necessary since local and hyperspace positions use Cartesian coordinates. Cartesian coordinates were used in this example because they are easier to visualize; however, multiple embedded spaces become useful if d differs between them. For example, a layer-space might assign neuronal positions using some biologically-inspired space (e.g., a manifold representing the cortex), but the neuron-space locates compartments according to a non-Euclidean, spherical geometry.

4.5 The Expanded Model

The expanded dendritic model extends the basic model to constrain compartmental connectivity around a biological model and to more precisely define the role of ϕ. The expanded model fills in the undefined aspects of the base model to facilitate an

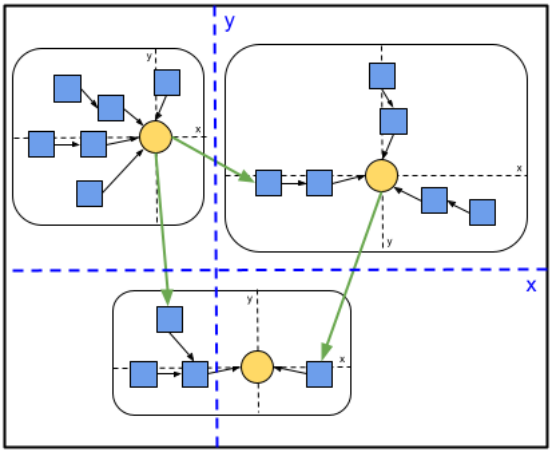


Figure 4.6: An example of the use of multiple spaces within a hyperspace. Compartments are placed within spaces defined by Cartesian x, y-coordinates. Blue squares are branch compartments and yellow circles are somatic compartments. Each somatic compartment exists at the origin within a separate local space and associated child compartments derive their local space values for d with respect to a local origin. Each compartment also maintains a position within the hyperspace coordinate system (blue axes). In this example, black arrows denote connections whose values of d rely on the local space. And green arrows use hyperspace coordinates.

implementation. It also adds several additional parameters to the definition of a connection to better control the effects of compartmental separation.

4.5.1 ϕ, the Output Function

The base model simply defines ϕ as a radial basis-type function (RBF) making use of expected (v_e) and actual (v_a) values--the minimum requirements of an RBF. To create the expanded model, the definition of a connection is increased to a 6-tuple: $(c_i, c_j, v_e, \phi, b, w)$. c_i, c_j and v_e are unchanged. b is a shape parameter that allows ϕ to adjust the width of a connection's *region of activation*. Smaller values of b cause the region of activation to widen and vice versa for large values. The default value used for b across most experiments in this work is 1 as is used in the example figure 4.5.

w is a weight parameter that scales the output of ϕ. However, w is not applied directly to ϕ. w is, in effect, a raw weight that is allowed to vary without bounds. The weight applied to ϕ is an actual weight (w_a) is given by equation 4.3. This sigmoidal function

bounds the actual weight to $(-W_{max}, W_{max})$. A wise choice for W_{max} prevents run-away behavior. Such a value is given in the following chapter on training.

$$w_a = \frac{W_{max} w}{|W_{max}| + |w|} \tag{4.3}$$

Equation 4.4 gives the expanded version of ϕ.

$$\phi(v_a) = \frac{w}{((v_a - v_e)b)^2 + 1} \tag{4.4}$$

The suggested expansion is not the only one possible. Other RBFs could be substituted for Equation 4.4. Two examples are the inverse multiquadratic ($\phi(v_a) = \frac{w}{\sqrt{1+((v_a-v_e)b)^2}}$) and the Gaussian ($\phi(v_a) = we^{-b(v_a-v_e)^2}$). These functions differ in their rates of decay as v_a moves away from v_e. The inverse quadratic (Eq. 4.4) was chosen over the other two because of its middling decay rate. When $b = 1$ and $w = 1$, the Gaussian decays to ~ 0 quickly ($|v_a - v_e| > 2$). And the inverse multiquadratic decays slowly ($\phi(v_a) > 0.1 \quad \forall |v_a - v_e| \in [-10, 10]$). b has a stronger and more linear impact on the shape of the inverse quadratic than the Gaussian.

4.5.2 Two Concerns: Relevance and Importance

Why do connections use an RBF to transform the signals passing from one compartment to another?

An RBF allows the dendritic neuron model to separate two concerns in the point neuron model that are somewhat conflated. These concerns are the *relevance* of a specific value to the receiving compartment and the *importance* of the connection carrying the signal.

Relevance is the meaning assigned by the connection to a specific value of v_a. Just like in the combination lock metaphor, the magnitude of a signal is not its relevance. A connection expects a specific value, v_e. The closer it is, the more relevant it is. The shape parameter b helps to define the range of relevant signals (i.e., the range of activation) [2] .

[2]Another way to think of the effects of b is the capacitance of a connection. Capacitance describes the ability of an object to maintain a value which is different from the contents of its surroundings (space, objects

Importance is the strength of a connection with respect to its sibling connections. Sibling connections are defined as all connections with the same c_j, or end compartment. Importance is controlled by w. Weight ranks sibling connections with respect to A) each other and B) the generation of compartmental output. For example, if w is relatively small, no matter how close v_a is to v_e, a connection's ability to generate compartmental output will be negligible. Inversely, relatively large w allow one connection to dictate the output of an entire compartment.

In the point neuron relevance *is* the magnitude of a signal. The only way for a connection to reduce the relevance of a signal is to minimize its weight. In other words, it is impossible for a point to neuron to make a weak signal relevant without also making a strong signal even more relevant, and vice versa.

The expanded model was designed to allow for a greater control over these concerns.

Figure 4.7 shows the differences between a point neuron's and the dendritic model's separation of relevance and importance. The x-axis gives the output of c_i, or its ψ, and the y-axis is the input to c_j, or φ. Solid lines are dendritic model connections and dashed lines are point neuron connections. The dendritic connection in blue has a lesser importance, i.e. weight, (0.5) compared to the orange (1.0). Although blue's region of activation is centered on 2 and orange's is centered on 7, the orange signal is not 2.5 times as important. Importance is tied to weight. This demonstrates that the region and peak of activation are completely separate from the magnitude of the signal.

The point neuron does not share this separation of concerns. For the point neurons, weights were chosen such that at the peak activation of the dendritic connection, the point neuron's connection produces the same value ($\varphi(2)$ and $\varphi(7)$ have the same result for both neuron models). To match the weaker (less important) dendritic compartment connection (solid blue) requires a stronger weight in the corresponding point neuron (dashed blue) compared to the other (dashed orange). In other words, the roles are reversed; the signal

or connections). b has an inverse relationship to capacitance. For large b, as v_a moves from v_e output drops quickly. When b is small, the drop is slow. Relevance bleeds into more of the input space surrounding v_e.

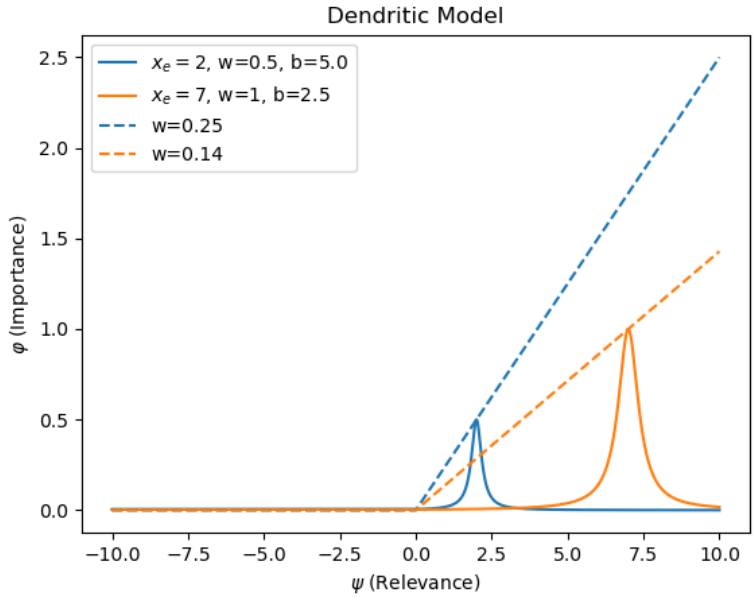


Figure 4.7: Comparison of how one output signal (ψ) is transformed by a connection of two different neuron models: dendritic neuron model and the traditional point neuron model. Both models receive output from an assumed ψ. Solid lines show how dendritic connections transform their input and dashed lines show how point neurons do it. For dendritic connections $\varphi(v_a) = \frac{w}{((v_a - v_e)b)^2+1}$. For the point neuron $\varphi(v_a) = v_a w$. The weights of the point neurons were chosen so that for the same ψ both versions of φ will output the same value for $v_a = 2$ and $v_a = 7$ respectively.

received by the blue point neuron has a greater absolute importance than the orange. Absolute importance means that for all input its output is larger. This is evident in the different slopes of the two dashed lines. This is an example how the point neuron treats smaller inputs as less important and, therefore, must increase the weight to produce greater importance.

4.5.3 Mimicking the Point Neuron

Since the compartmental output function, ψ, can use any activation function, the point neuron and the AD neuron compartment are identical (with respect to output). However, the dendritic connection's transformation of compartmental input is very unlike that of the point neuron's. Connections respond to a specific range of inputs. This precision impacts

the dendritic model's ability to match the types of input passed between point neurons, e.g., a broad range of values or an open-ended range (e.g., $[t, \infty)$, where t is a threshold).

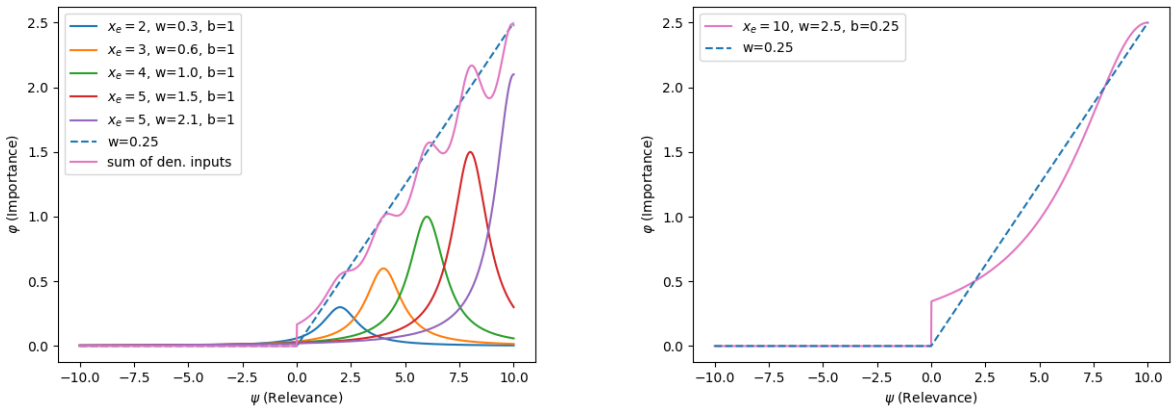
Does the AD neuron's type of input prevent it from mimicking the point neuron? In a larger sense, this is a mathematical question. It involves answering whether all instances of a target class of function can be matched by some combination of instances from another class of function. Since point neuron activation functions are drawn from a variety of function classes, the question actually involves a set of target functions. A precise answer to this question is beyond the scope of this work.

However, we can provide evidence of the ability to mimic behavior qualitatively. The dendritic model can mimic point neuron input through either duplicate connections (Figure 4.8a), multiple v_e per connection (see Figure 5.3), or a single connection with a broad region of activation (Figure 4.8b).

These solutions cannot match the entire range of positive input values exactly. However, in practice, covering from 0 to infinity is usually not necessary. If input to a point neuron, in practice, is constrained to some finite range, either of these dendritic-to-point conversion techniques can approximate it. This comparison assumes that it is desirable in some scenarios for point neurons and compartments to receive unconstrained monotonic input.

So far, we have compared only typical point neuron input (from other point neurons) to the input generated by AD neuron connections. There is no reason a point neuron cannot utilize an RBF as its activation function, or even a function with more complex behavior [26]. Ignoring that a point neuron RBF activation function belongs to the neuron and not a connection, point neuron input can mimic AD connection behavior. Likewise AD compartments can be made to behave like point neurons by selecting different activation functions.

Finally, how does the full AD neuron fare when compared to the point neuron? In an identical network setting both point and AD neuron receive the same number of inputs. Neither has an informational advantage. The AD neuron possesses potentially many more



(a) (b)

Figure 4.8: This pair of graphs shows two ways point neuron behavior can be mimicked by the dendritic model;. In both figures, the dashed blue line depicts the behavior of a point neuron using a ReLU activation function over the input range $[-10, 10]$ (a) The solid lines (blue, orange, green, red and purple) depict output for five dendritic model connections with varying values for v_e and w. The pink line depicts a dendritic compartment's ReLU response to these five connections. The compartment response roughly matches the point neuron behavior over the shown interval. (b) The pink line shows a compartment's response to a single dendritic connection which uses a small value for b which produces a broad range of activation. The broad range of activation allows the dendritic neuron to mimic roughly point neuron behavior.

activation functions in both its connections and compartments, thus it can apply more transformations to the same information. This alone suggests it is more powerful. From the earlier argument, the AD neuron has not lost the ability to be a point neuron because a single compartment is also an AD neuron.

4.5.4 The Benefits of Separation

Relevance and *importance* are a product of modeling the separation of network components and the dendritic model is based on the idea that separation itself possesses computational significance. What computational characteristics do these products of separation bring to the dendritic neuron? The separation of relevance and importance has three primary, intertwined computational effects: democracy of signals, connection determines meaning, and computation through coincidence.

The *democracy of signals* refers to the idea that the magnitude of a signal is irrelevant to its impact on the receiving compartment. A very small (or even negative) value can drive the output of a compartment just as well as a very large one. The democracy of signals is a direct consequence of divorcing the relevance of a signal from its importance.

Second, *connection determines meaning* is the idea that the meaning of a signal is something locally determined by each receiving neuron. Both biological and artificial neuronal connectivity follows a one-to-many pattern. Rather than subject every receiving neuron to the strength of a signal, the AD neuron allows each receiver to determine the meaning of a particular signal by manipulating v_e of the intervening connection.

Third, *computation through coincidence* refers to neuronal behavior that is the product of coinciding activity (or co-activity) at a select set of inputs. Since dendritic model input signals are evaluated individually based on their relevance and collectively based on their importance, the output of a dendritic neuron depends on the right set of co-active inputs.

4.5.5 Connectivity

The AD model compartment, by itself, does not make a dendritic neuron. Alone it is a point neuron. So, to fully realize the AD neuron model, we must define a set of rules governing compartmental connectivity.

The AD neuron model can accommodate any number of biologically plausible and implausible morphologies. The following constraints are not hard requirements. Generally, we propose these rules to provide an example how compartments can create complex dendritic trees through sub-typing. Compartment sub-types are created by selecting different constraints on connectivity (and different activation functions) to limit the possible shapes (and behaviors) of the dendritic tree. For this work, the following set of constraints guided the development of the tree-building algorithms in Chapter 6.

Let K be the set of compartments which comprise the dendritic neuron. K can be divided into two disjoint sets, S and B, such that:

$$K = S \cup B \quad \text{and} \quad S \cap B = \emptyset \tag{4.5}$$

$$|S| = 1 \quad \text{and} \quad |B| = |K| - 1 \tag{4.6}$$

B contains all branch compartments and S contains all soma compartments. Each compartment's set of connections can similarly be partitioned into two sets, A and E, or the afferent (incoming) and efferent (outgoing) connections. Each of these sets can be further partitioned into those either coming from or going to branch or soma compartments. In total, each compartment's connections are partitioned into four sets: A_S, A_B, E_S, and E_B. For example, the set, A_S contains all afferent connections from M-compartments. Partitioning the connections in this way allows us to create a collection of restrictions on each set. Together, these restrictions define the possible shapes of a dendritic tree.

Definition: Soma, or M-compartments, are defined by the following restrictions on A and E:

$$\begin{aligned} A_S &= \{s | s \in S \quad \text{and} \quad |A_S| \in \mathbb{N}_0\} \\ A_B &= \{b | b \in B \quad \text{and} \quad |A_B| \in \mathbb{N}_0\} \\ E_S &= \{s | s \in S \quad \text{and} \quad |E_S| \in \mathbb{N}_0\} \\ E_B &= \{b | b \in B \quad \text{and} \quad |E_B| \in \mathbb{N}_0\} \end{aligned} \tag{4.7}$$

Equations 4.7 state that S-compartments can make afferent and efferent connections to zero or more S- and B-compartments. Allowing M-compartments to receive input and send output to as many (or few) compartments of both types is important because it allows AD neurons to create both dendritic and adendritic subnetworks within a network.[3]

Definition: Branch, or N-compartments are defined by the following restrictions on A and E:

[3]The existence of adendritic neurons in the brain suggests not all neural computation is best served by dendritic neurons.

$$
\begin{aligned}
A_S &= \{s | s \in S \quad \text{and} \quad |A_S| \in \mathbb{N}_0\} \\
A_B &= \{b | b \in B \quad \text{and} \quad |A_B| \in \mathbb{N}_0\} \\
E_S &= \{s | s \in S \quad \text{and} \quad |E_S| + |E_B| = 1\} \\
E_B &= \{b | b \in B \quad \text{and} \quad |E_S| + |E_B| = 1\}
\end{aligned}
\tag{4.8}
$$

Equations 4.8 restrict branch compartments such that they, like the soma compartment, can make unrestricted afferent connections. They differ from the M-compartment's in that they only send output to one and only one compartment. The type of the efferent connection is unrestricted.

Definition: A *dendritic path* is defined as the B-compartments and connections linking two S-compartments. A path, P, is a dendritic path if $P = \{p_0, p_1, p_2, \ldots, p_k\}$ where p_0 and p_k are S-compartments and $p_i, i \in \{1, k-1\}$, are B-compartments. P is also defined by a set of connections, $P_C = \{c_{k,k-1}, c_{k-1,k-2}, \ldots c_{1,0}\}$. All dendritic paths are directed such that there is a connection from p_{i+1} to p_i.[4] To guarantee the dendritic neuron's shape is tree-like, we place a further restriction on dendritic paths. Given the above definition of a path, we add that in a dendritic path P, $\nexists\, i, j \mid p_i = p_j$. In other words, there are no cycles. As a last step, p_k can be removed from P. p_k is removed because it is the S-compartment, or soma, for an upstream neuron. It was included initially to ensure that the last B-compartment receives input from at least one S-compartment.

Two paths are the same if their compartments and connections are identical and in the same order.

Definition: A *dendritic tree* is defined as a set of dendritic paths terminating in the same S-compartment which also share at least one B-compartment. Given an S-compartment, m, a dendritic tree of m is a set of dendritic paths D such that for all

[4]It might seem odd that we are defining direction flowing from p_k to p_0 rather than the reverse. The choice is arbitrary in general, but defining it this way simplifies the definition of dendritic trees which relies on the path definition.

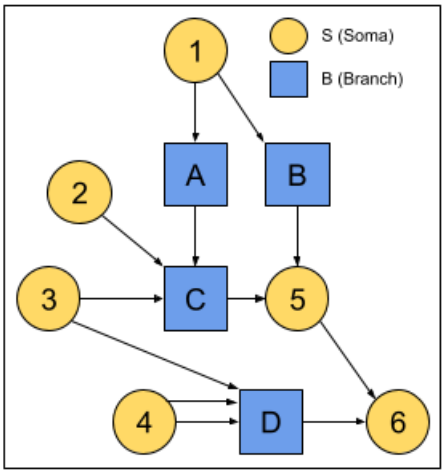


Figure 4.9: Sample network of dendritic compartments using the proposed connection restrictions.

$P \in D$, $p_0 = m$ and there exists an n such that the sub-path $\{p_0, \ldots, p_n\}$ is equivalent (by the above definition of path difference) for all paths in D.

Definition: A *dendritic neuron* is defined as a single M-compartment and *all* of its dendritic trees.

The dendritic neuron model's connectivity constraints allows for the creation of *asymmetric*, *sparse* and *super-dense* networks. An asymmetric network is one in which not all paths from input to output are the same length. A sparse network is one in which connections between layers is something less than all-to-all. And a super-dense network possesses a connectivity greater than all-to-all in that it has duplicate connections between components.

Figure 4.9 provides a sample network using these compartments and possessing these qualities. Somatic compartments 1, 2, 3 and 4 comprise the set of input neurons, where 5 is the only soma in the hidden layer, and 6 is the output compartment. The network exhibits sparsity in that the path from 4 to 6 does not pass through 5. Asymmetry can be seen in that there are two 3-to-6 paths of differing lengths (same with 1-to-6). Super-density is evident in the multiple connections from 4-to-6.

4.6 Combinatorics of the Dendritic Neuron

Before moving on to an examination of the behavior of the dendritic neuron model, let us consider the effects of compartmentalization and separation on the shape of the dendritic neuron. Related research suggests that the shape of a network determines its expressivity [132]. Expressivity describes the range of functions a neural structure can approximate.

If shape impacts expressivity, then increasing the number of different shapes a neuron can take on is beneficial because the dendritic neuron can better control its behavior through its shape. The dendritic neuron model has vastly more unique shapes than the point neuron and is therefore more expressive. It is capable of approximating a wider range of functions because it is better able to control its shape and therefore exhibit more precise behavior. To better understand just how configurable it is by comparison, let us start with a point neuron and from it construct a dendritic neuron. This will allows us to compare the configurability of the two types of neuron models.

A point neuron, e, receives inputs from an input vector, I where $|I| = n$. The first step in constructing a dendritic neuron, f, from e is to partition e's inputs into compartments. The number of ways to partition a set of n things is given by the Bell numbers [14]. For n inputs there are B_n ways to compartmentalize (partition) them, where B_n is the n^{th} Bell number.

Bell numbers have an exponential growth rate in n and are given by the following recurrence relation:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \tag{4.9}$$

The exponential growth of the Bell numbers is very fast. For example, the Bell numbers, B_k, for all $k \in \{0, 1, \ldots, 13\}$ are 1, 1, 2, 5, 15, 52, 203, 877, 4,140, 21,147, 115,975, 678,570, 4,213,597, 27,644,437. An AD neuron with thirteen inputs can partition them in 27,644,437 ways. By comparison, the point neuron is incapable of partitioning its inputs; therefore, regardless of n, it always has just one compartment--all n inputs. The

equations in 4.10 give the number of shapes of the point neuron with n inputs ($S_e(n)$) and the number of shapes of dendritic neuron with n inputs ($S_f(n)$).

$$S_e(n) = 1 \qquad S_f(n) = B_n \tag{4.10}$$

The partitioning of inputs does not give the full account of dendritic model. The above does not take into account the ordering of compartments. If we further consider that compartments can be ordered (connected) hierarchically then the number of different combinations further increases.

For any partitioning of the inputs, p, $|p| = m$, or the number of partitions of p. We label each partition with a label, $l_i, i \in \{1, 2, \ldots, m\}$. To find the total number of unique trees given m labeled partitions, we use Cayley's formula, Equation 4.11. The number of trees, t, with m labeled vertices is:

$$t(m) = m^{m-2} \tag{4.11}$$

Cayley's formula grows even faster than the Bell numbers. For example, $B_{13} = 27,644,437$ and $t(13) = 1,792,160,394,037$. Strictly speaking, Cayley's formula cannot be applied to the point neuron because it has only one partition; however, a tree of one node is a tree.

Combining B_n and $t(n)$ gives a lower bound of the total number of shapes a dendritic neuron with n inputs. We will call this $a(n)$:

$$a(n) = \sum_{i=1}^{B_n} t(|p_i|), \text{where } p_i \text{ is the } i^{th} \text{ partitioning of the set of } n \text{ inputs.} \tag{4.12}$$

The equations in 4.10 can be rewritten to give our final comparison:

$$S_e(n) = 1 \qquad S_f(n) = a(n) \tag{4.13}$$

A lower bound on the growth rate for $a(n)$ is $\Omega(a(n)) = \frac{t(n)t(n+1)}{2}$. In other words, $a(n)$ grows according to Gaussian summation of the result of Cayley's formula from 1 to n, the maximum possible cardinality of all partitionings of n. This is a lower bound on $a(n)$ as it assumes (among other things) that there is only one partition of each size. In fact, the number of ways to partition a set into exactly k elements where k is from 1 to n is given by the Stirling number of the second order, or $S(n, k)$. For all values of k such that $1 < k < n$ and n such that $n > 2$, $S(n, k) > 1$. For our purpose here, it is enough to show a lower bound as all we are concerned with is that $S_f(n)$ is vastly larger than $S_e(n)$.[5]

The above does not take into account that each connection within each arrangement of each partitioning can also assign different values for v_e, w and b which would multiply the number of possibilities by a constant, $n(2^{64})^3$, assuming an implementation on a 64-bit machine.

4.7 Conclusion

The potential benefit of the dendritic neuron is its ability to precisely bound regions of the input space that produce interesting (or meaningful) behavior. To do this, the artificial dendritic neuron model has been constructed to allow for a large number of configurations, or shapes. Certainly, it is reasonable to assume that many shapes of a single dendritic neuron are equivalent in that they produce, at least qualitatively, the same behavior. So it is necessary to show that different configurations produce different behaviors. In the next chapter, we will look at a simple implementation that explores the subtleties of dendritic expressivity through the partitioning of inputs and the ordering of compartments.

[5]For $n = 13$, $a(n)$, by the lower bound, says that $S_f(n)$ is least 29.2 trillion.

CHAPTER 5
BEHAVIOR OF THE AD NEURON MODEL

In this chapter we employ a simple, non-trainable implementation of the extended AD neuron model to demonstrate its basic behavior. The purpose of starting with a simple, non-trainable version is to show the computational nature of compartmental separation and connectivity and the benefits of separating relevance from importance. Examples are limited to a single dendritic neuron with a minimal number of compartments. AD neuron shapes and parameters are set by hand since the goal is to show behavior, not learning.

This implementation adheres to the specifics of the extended model from Chapter 4. Specifically, ϕ is given by Equation 4.4 and the connection scheme for multiple compartments follows those in Section 4.5.5. ψ primarily uses an identity function $\psi(x) = x$ in earlier examples where negative weights are not present. When negative weights are present, $\psi(x) = \text{ReLU}(x)$ to prevent the compartment from producing negative values. Further alterations to ψ in the more complex later examples will be identified.

5.1 Input Relevance

As stated in the previous chapter, dendritic connections use the RBF ϕ to define a discrete range of activation over which input generates a non-minimal signal to the compartment. How close an input is to the center of the range of activation gives its relevance. In the following examples we demonstrate the AD neuron model's ability to precisely define bounds for input relevance, how v_e, b and w can be used to shape a compartment's response and the effects of relevance on compartment behavior. The goal is to show that individual dendritic compartments exhibit a wider range of behaviors over point neurons.

In this first example (Figure 5.1) we present two dendritic compartments, A and B. Compartment A has two connections with differing centers relevance (v_e). The right

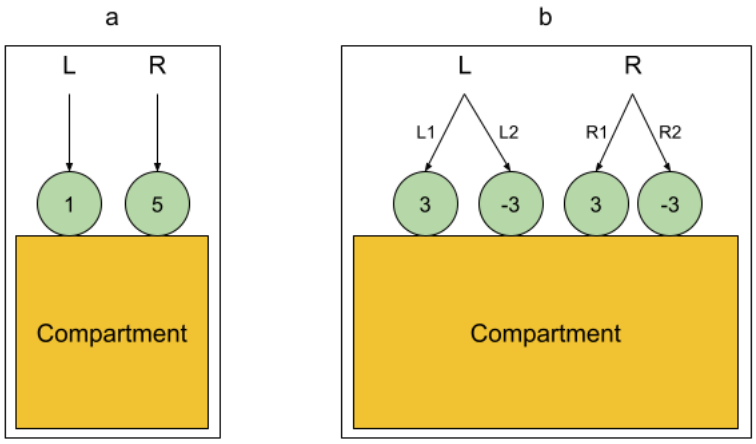


Figure 5.1: Two example AD model compartments. Yellow boxes are compartments. Green circles show each connection's v_e. (a) A single compartment with two inputs: L and R. The L input has $v_e = 1$ and $b = 0.5$. R has $v_e = 5$ and $b = 3$. (b) A single compartment with two duplicated inputs. L1 and L2 both receive the same signal, L, and R1 and R2 both receive R. L1 and R1 both have $v_e = 3$. L2 and R2 both have $v_e = -3$. All four connections have $b = 3$. w is 1 for all connections in (a) and (b).

connection is centered at 5 and the left is at 1. The compartment's maximum response is, then, at the intersection of 5 and 1 (left plot of Figure 5.2). Each connection to A uses a different value for b. b determines the width of the region of activation. The right connection has a larger value ($b = 3$) compared to the left ($b = 0.5$). Larger values tighten a connection's region of activation and higher values widen it. This causes the compartment to receive near-max input for a wider range of input values to L than for those to R. This example demonstrates how the input to compartment A is detached from the strength of the actual value sent to each connection. R is keyed to a stronger value (5 vs 1 for L) but how this is interpreted by the connection for the compartment creates an inherent equality between inputs.

Compartment B has four connections from two input sources (L and R). Each source and compartment B share two connections. Duplicated connections have different values for v_e. The first connection (L1 and R1) from each source have a v_e equal to -3. In second connections $v_e = 3$. For all connections, $b = 3$. Compartment B receives a maximum input from each source at two values; therefore, the combined input from all four connections

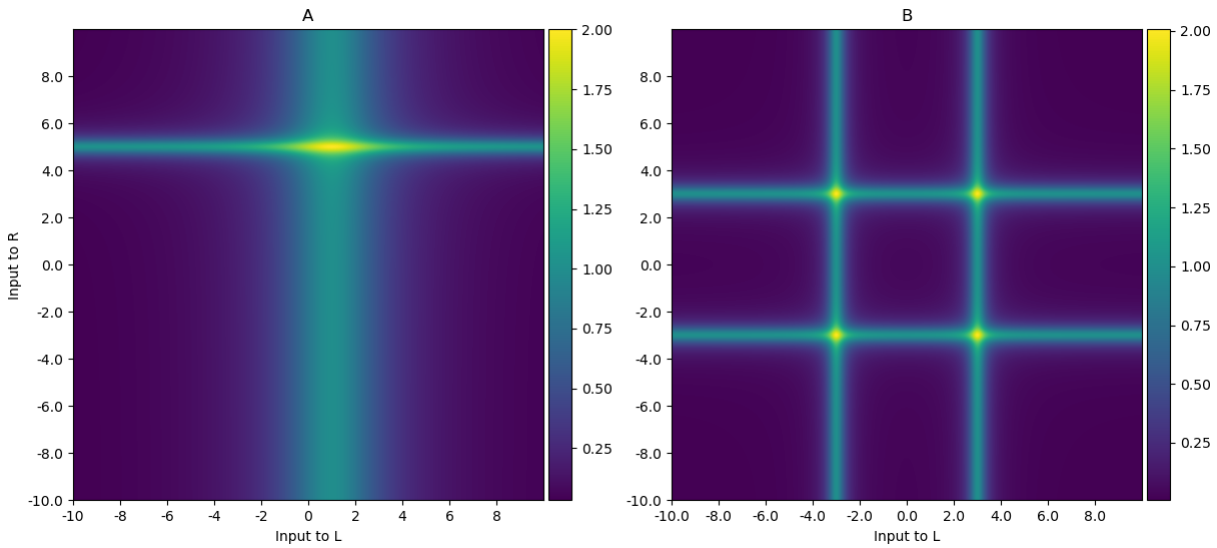


Figure 5.2: The input profiles for compartments A (left) and B (right) from Figure 5.1.

reaches a maximum at four points in the input space. The four ranges of activation are narrow enough that each one does not overly impact the others. But there is some overlap. This can be seen in a maximum slightly above 2 (see B's color bar). At each of the four points of maximum relevance in the compartment's input space, traces of relevance generated by the other connections is felt. How much is felt depends on the proximity in $\mathbb{R}$ of each connection's v_e. Large b result in little or no overlap. By contrast, very small values for b, ($b \ll 1$), could produce ranges of activation so wide that areas of maximum activation would shift from the four intersections toward a central region (as we will see in a later experiment).

The above examples are limited to two input sources for display purposes. More inputs will create regions of input relevance in higher dimensional spaces. If a compartment's set of connections all receive input from unique connections--no two connections share the same source--the compartment's region of maximum input within the input space is around a single fuzzy point whose shape is governed by the values of b across all connections. As shown in Figure 5.2, multiple max regions can be created through duplicate connections from the same source. Input relevance for a single connection, however, is not limited to a fuzzy point in the higher dimensional input space. Each connection in the two dimensional

case produces a fuzzy line of elevated relevance cutting across the input dimensions created by sibling connections from a different source. From this we see that the input relevance of a single connection is a hyperplane of $n-1$ dimensions, where n gives the number of input sources, not connections, to the compartment. This is an important distinction between connections and sources. Multiple connections from the same source do not increase the dimensionality of the activation profile, they increase the regions of activation. Sources on the other hand increase the number of dimensions.

The shape of a compartment's regions of activation are determined by the implementation of ϕ. The AD neuron model does not set any limits on ϕ except that it is a radial basis-type function. Regions of activation can take on different shapes by using different implementations of ϕ. Figure 5.3 gives four variants of ϕ which use an inverse quadratic; however, each is augmented by an additional transformation. The top left adds the update function of a logistic map to the R connection. The result is that the connection has two centers of input relevance at 0 and v_e (which is 5). The top right uses sine and cosine to create a grid of activations. In the bottom left plot, ϕ is negated for the R connection. Both L and R have have an $v_e = 0$ and $\psi = \text{ReLU}$. The negative input silences the region of maximum activation. If the compartment's ψ used the identity function, R's activation line would produce a trough in the input space. Finally, the bottom right shows a connection with multiple centers (2 and -2). R's output is the sum of ϕ calculated at each center. Because the centers have a narrow profile ($b >> 1$) and are well separated, maximum relevance is still 2. In other words, the regions of R's activation (-2 and 2) do not overlap.

The preceding plots show how different implementations of ϕ and connection-level properties impact the shape and regions of activation of compartments. Compartmental behavior depends on ϕ's treatment of input relevance. We can manipulate b to broaden or narrow the region of max relevance. And we can augment the evaluation function itself to

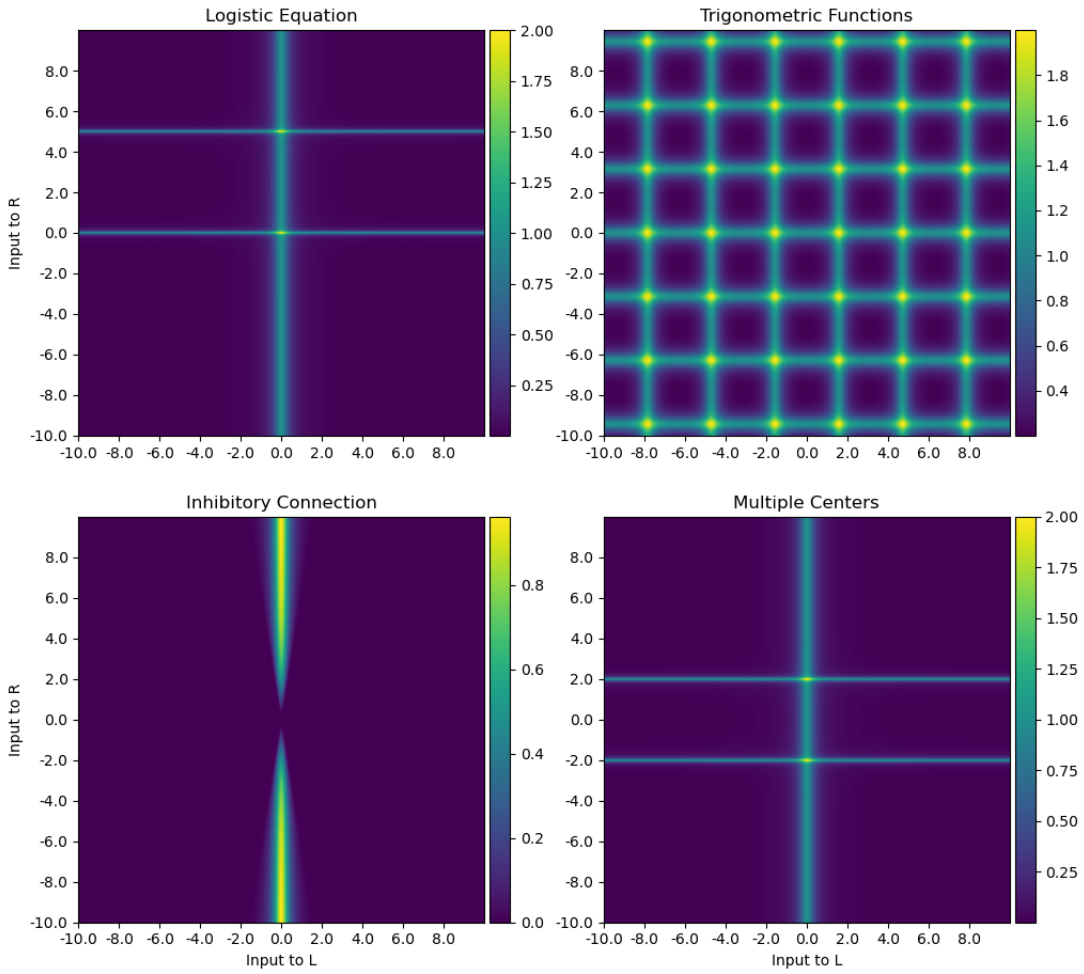


Figure 5.3: Four variants on the basic implementation of ϕ. TOP LEFT: The left connection's ϕ includes the update function for a logistic map $\phi(v_a) = \frac{w}{(bv_a(v_a - v_e))^2+1}$ with $v_e = 5$. TOP RIGHT: The two connections, left and right, include cosine and sine respectively: $\phi(v_a) = \frac{w}{(b \sin v_a - v_e)^2+1}$ and $\phi(v_a) = \frac{w}{(b \cos v_a - v_e)^2+1}$. BOTTOM LEFT: The right connection is negated: $\phi(v_a) = \frac{-w}{(bv_a - v_e)^2+1}$. BOTTOM RIGHT: v_e is not a single value but a list of values (2, -2). $\phi(v_a) = \sum_v^{v_e} (bv_a - v)^2 + 1$

generate different types of activations. Relevance is the first step in showing that by making connections nonlinear, we make them unique in how and where they respond to input.

5.2 Weight, or Connection Importance

In the preceding chapter (Chapter 4) we suggested that connection weights are synonymous with the importance of a connection to the receiving compartment. In this subsection, we briefly demonstrate the way in which connection weights adjust the

contribution each connection makes to the compartment and how this alters compartmental behavior.

A connection's weight determines how strong a role the connection plays in forming the overall input to a compartment. Figure 5.4 show two variants of Compartment A from Figure 5.1 for which connections are identical but the implementations of ψ are not. In both variants, R is four times the weight of L ($w = 4$ versus $w = 1$). Being four times as strong R dictates the extent and strength of the compartment's regions of activation. In the first variant (left plot) L and R, $\psi(x) = x$. The linear integration of L and R allows the strength of R's signal to dominate the input profile around R's center. The output of compartment A depends strongly on R.

In the second variant (right plot), ψ is the hyperbolic tangent function. The nonlinear integration of L and R creates two inputs of relatively equal strength but unequal width. Compartmental output is at maximum whether one or both connection receives a relevant input. The greater importance of R is manifested in this broader range of activation. The region of activation of the second variant is somewhat akin to a very small value for b, except R has a wide range over which actual compartmental input plateaus. Once off this plateau, relevance decays quickly, mimicking a square wave.

Both variants show that a connection's weight determines its importance and how well signals from a single connection determine compartmental output. But importance is also subject to interpretation by the compartmental activation function, ψ.

This is not the only interpretation of the effects of connection weights.

Connection weights and ψ help determine a compartment's number of distinct responses across its entire input space. A distinct response is a local or global minimum or maximum. To find all distinct responses we discretize the output space accordingly. For example, if we discretize the outputs of both variants of Figure 5.4, we see that the first contains more distinct values than the one which uses the hyperbolic tangent (right). Discretized, the left compartment produces 0, 1, 4 and 5. These discrete outputs correspond to the four states:

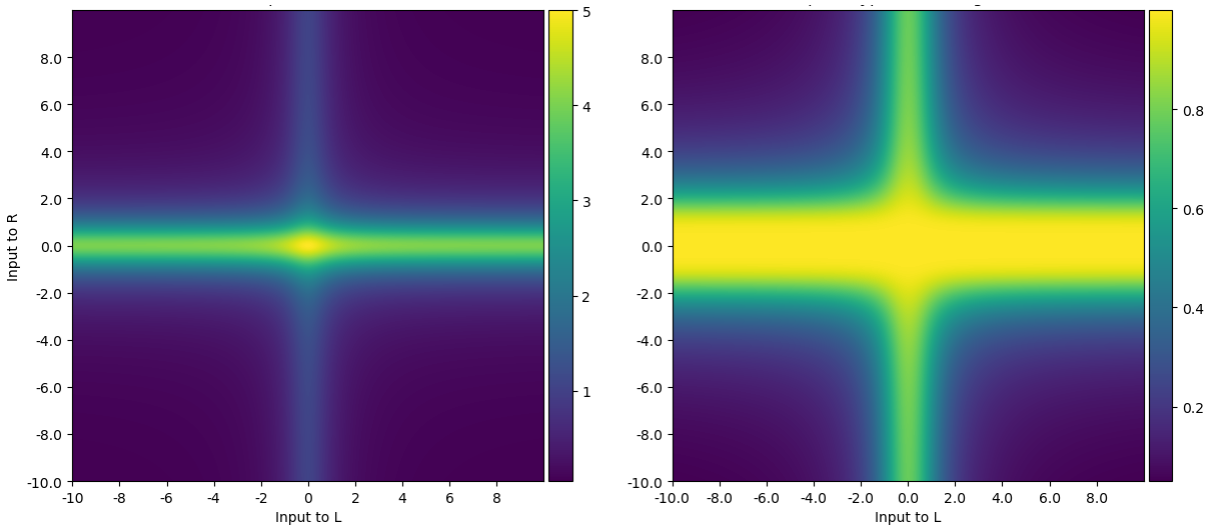


Figure 5.4: The impact of weight on connection importance. Both plots are derived from identical compartments except for the implementation of ψ. LEFT: ψ is the identity function. RIGHT: ψ is the hyperbolic tangent function. $b = 1$ for both.

$\bar{L}\bar{R}$, $L\bar{R}$, $\bar{L}R$ and LR. The right compartment on the other hand only has two distinct states: 0 when no inputs are active ($\bar{L}\bar{R}$) and 1 when any is active ($L\bar{R} = \bar{L}R = LR$).

What does all this mean?

Weight, in the context of compartmental output, can function as a kind of multiplexer. Differing connection weights cast different combinations of active inputs into unique ranges. The left compartment has 2^2 possible output states which is maximal for a compartment with 2 connections from unique sources. The right compartment demonstrates a configuration with only 2 possible output states. The right compartment is limited to reporting only the presence or absence of input. In the next section, we show how this interplay between ϕ and ψ can be used to create Boolean functions.

5.3 Boolean Behaviors

In this section, we will demonstrate that the AD neuron can compute any logical expression by showing that the AD neuron is capable of behaviors that resemble the basic Boolean functions of OR, AND, and NOT. We also demonstrate that it is capable of the additional functions XOR, NOR, NAND and IMPLIES.

Boolean evaluation of input is performed with respect to a connection's v_e. In the unary case, a single input signal is "truer" the closer it is to v_e because only around v_e will the connection produce an above baseline signal. The choice of v_e (or which signal evaluates to true) is arbitrary.

In a Boolean context, the AD neuron's evaluation of truth and falsehood are fuzzy and cover all values. It does not make sense to limit inputs to the interval $[0, 1]$. In fact, the following examples were configured around the value zero. The following results should be interpreted as A=0 OP B=0 rather than A OP B, where OP is one of the Boolean operations. For example, in Figure 5.5 the output of the OR network (d) is closer to 1 (truer) when either or both inputs are or near 0. It is closer to 0, or falser, the farther both inputs are from 0.

In each sub-figure, the green squares indicate connections and the numbers inside are the values for v_e. Arrows indicated the flow of information. Blue circles are compartments. A and B are input signals and C is the output. [1] ϕ utilizes the inverse quadratic RBF and ψ is the identity function. For all connections, $b = 2$ and $w = 1$. All examples receive input from the same sources, A and B, which ranged from -5 to 5.

The simplest logical function is NOT (boxes with dashed lines) requiring a compartment and an output connection. NOT is not shown in a separate diagram but can be seen in the sub-figures d, e and f of figure 5.5. There are two important aspects to NOT that cause it to invert output. First, the connection's v_e equals 0. When the compartment receives a non-zero value, the following connection will output a 1. And when it receives an input of 0, it will output a 1. Second, NOT depends on an earlier evalution of a particular signal. For example, in f (IMPLIES), the initial NOT receives input from an initial connection with a $v_e = 0$. When $A = 0$, the value sent to the NOR compartment will be 0 and not 1 like B. The same is true for the other three examples. They receive as input the

[1] Technically, C would be the signal received by a next, downstream compartment since it is the result of a connection.

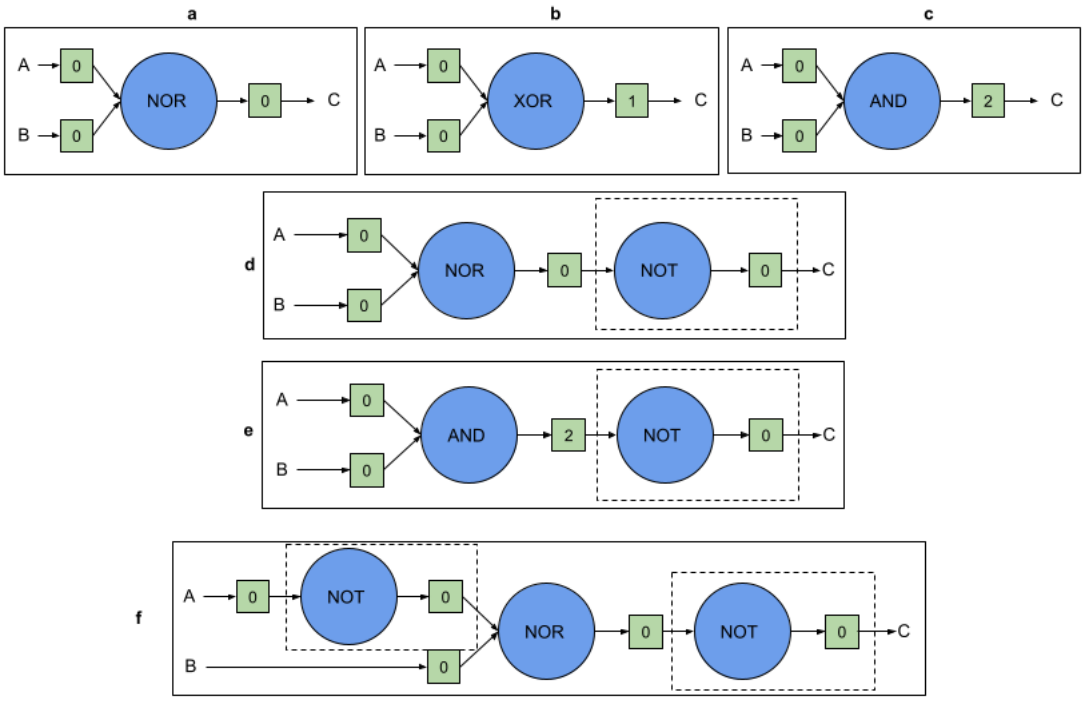


Figure 5.5: Boolean behaviors: architecture. Blue circles are AD neuron compartments labeled with their basic Boolean function. Green squares represent the compartment's ϕ function with the numeric value giving v_e. Arrows show the flow of information. (a) Nor (b) Xor (c) And (d) Or (e) Nand (f) Implication ($A \implies B$) is implemented as $\overline{\overline{A} \vee B}$. Boxes with dashed outlines indicate the architecture for Not.

output of NOR and AND operations. Each will produce 1 when the comparison is true. NOT, expecting 0, will invert the results producing OR and NAND.

There are other ways to create NOT involving more elements. The only element in the NOT structure doing any work is the connection. The additional compartment has been added because the AD neuron does not allow for two connections without an intervening compartment.

For the binary operations, we start by replicating NOR, XOR and AND (top row) because these require the fewest elements (a compartment with preceding and following connections). In each example, the preceding connections have a $v_e = 0$. When input is 0, they will send an input of 1 to the compartment. Therefore, compartmental input will range from 0 to 2. The following compartments of a, b and c have different values for v_e. These values indicate how many inputs must evaluate to true for the total output, C, to be

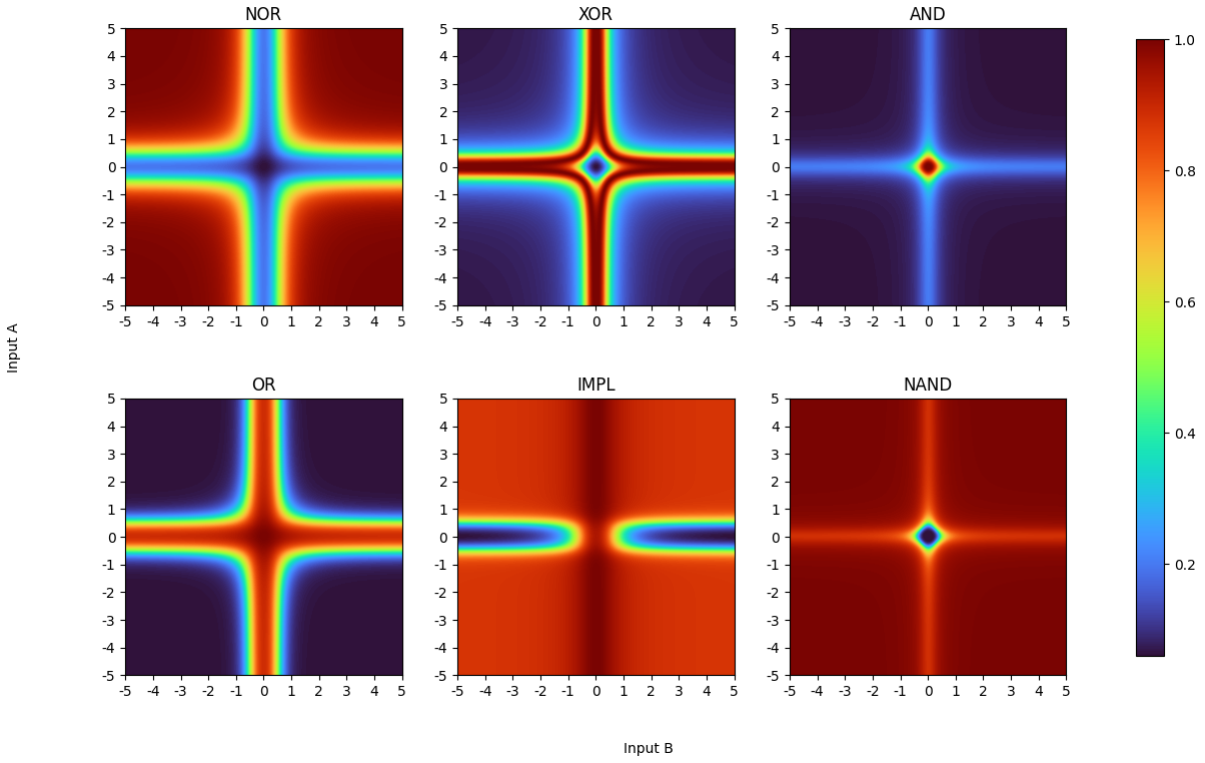


Figure 5.6: Boolean behaviors: results. The behaviors depicted by these images show use the architecture in Figure 5.5. To create these images, a sweep was performed over inputs A and B in the range of [-5,5]. The warmth of the color indicates truth.

true. For example, AND is only true when ψ, the compartmental output, is 2 signalling both inputs evaluated to true (A=0 and B=0).

To produce OR (D) and NAND (E) the output of NOR and AND is fed into a second compartment which functions as a NOT compartment. Logical IMPLIES (F) requires three compartments: two NOTs and a NOR.

Boolean behaviors (Figure 5.6) of the AD neuron have several distinguishing characteristics due to the radial basis nature of ϕ. First, the transition from regions of low activity to regions of high activity (or from false to true), or vice versa, is fuzzy. Fuzziness here is defined as allowing truth values to take on any real number from 0 to 1. Transitions between states pass through a continuous range of values rather than through a set of jumps or steps. The fuzziness of the transition and width of each low/high regions is, as demonstrated in previous sections, determined by the interaction of b, w and ϕ. Truth and

falsehood for some Boolean functions is not equal across all regions where they are typically true and false. For example, OR is typically true when one or both inputs are true; however, the AD neuron produces a higher value when both inputs are true than when only one is. This can also be seen in the AND example's pale blue regions along both x- and y-axes versus the dark blue as both inputs move away from them.

More interesting versions of Boolean behavior can be created by chaining these elementary examples together. Diagram F in Figure 5.5 shows how to construct logical implication, $A \implies B$. The bottom center image shows the results of this construction. Output is false only when A is true (or $A = 0$) and B is false (or, $B \neq 0$). Again, certain values for A and B produce different levels of truth. The AD neuron's version of IMPLIES produces a higher output for $\overline{A} \implies B$ than for $\overline{A} \implies \overline{B}$. In general, it responds more strongly when input B is active. This mirrors the fact that for $A \implies B$, when B is true the output is true regardless of A.

By comparison, four of these Boolean functions can be reproduced by single point neurons. OR and AND are relatively easy for the point neuron because these Boolean functions exhibit threshold-like behavior. Once enough inputs are true (or not true) to cause the output to be true, adding more true inputs does not make them false. Similarly NOR and NAND can be reproduced using point neurons with negative connection weights and a positive threshold signal. A single AD neuron compartment can reproduce XOR, a single point neuron cannot. A minimal point neuron implementation of IMPLIES requires two neurons and the AD neuron requires three compartments.

The AD neuron does face limitations in creating Boolean functions. Functions such as OR are best approximated by a threshold, whereas, the AD neuron is best when approximating functions which are true for unique combinations of input values, such as XOR, NOR or AND. For example, XOR is true when the sum of its inputs is one. For NOR, the sum of its inputs must be zero. For AND, it must be two. It requires more compartments when when approximating functions which are true for multiple

combinations, such as OR, which is true when the sum of its inputs are one or two. This limitation is overcome through the negation of one of the easier functions.

The AD neuron with enough compartments is capable of computing any logical expression. As the number of compartments and the complexity of their connections increases so does the fuzziness of the output space. Increased fuzziness comes from overlapping regions of activation due to wide activation profiles or layering across a chain of compartments. Regions of silence for a connection or compartment are not always completely silent. Depending on b and w and within a range of source values, input or output may never reach zero or its maximum. This can be seen in the output of IMPLIES (F) where the regions over which OR is true leave an imprint on the final result.

We also expect that for very complex Boolean expressions, requiring deep dendrites, early operations, performed in distal branches of the tree, will be subject to later ones. The effects of this are demonstrated in the next section. Constructing very complex Boolean expressions may be challenging and require fine-tuning values for b, w and v_e.

A potential solution to this problem is to use matching values of v_e and w along interior connections (i.e., non-input or non-output) to multiplex signals, casting them into different parts of the real number space. The variant of compartment A in the left image of Figure 5.4 demonstrates how w can be used to separate the signals and combinations of signals a compartment receives from its connections by shifting them away from the unit range $[0, 1]$. This could be used as kind of gain function to reduce fuzziness and create definition between true/false boundaries.

5.4 More Complex Behaviors

Before we move onto multiple compartment examples, single AD neuron compartments are capable of even more complex behaviors by adding compartment- and connection-level parameters. An example of a compartment-level parameter is a threshold, t. In point neurons, thresholds are typically a static negative input that prevents a neuron from

producing an output unless the sum of its inputs exceeds it. Such a threshold models the biological all-or-nothing phenomenon associated with neural activity. Below a certain level of input, the neuron does not alter its baseline activity. Equation 5.1 modifies the basic AD compartment model to include a threshold. The threshold is simply a static negative value applied to the sum of all connection signals.

$$v_a = \psi(z - t), \text{where } z = \sum_{i=0}^{n} z_i \text{ and } \psi = \text{ReLU} \tag{5.1}$$

In the left plot of Figure 5.7, we have given a variant of compartment B (from Figure 5.1) a threshold value of 2. All four connections are given sub-threshold weights. Two of the connections, L2 and R2 (-3, -3), have a weight of 1; whereas L1 and R1 (3, 3) have a weight of 2. This alters compartment B's behavior such that at only one point, when source L and R are both equal to 3, does the compartment reach a maximum output of 2. It has two regions where output is half the maximum, 1. At the intersection of L2 and R2 (-3,-3), where the weighted signals do not breach the threshold, the neuron does not respond at all. This plot shows that weights, in conjunction with a threshold, can elevate some regions of activation while silencing others.

The right plot depicts cluster-like behavior. This plot depicts the output of another variant of compartment B. We have altered weights, thresholds and activation profiles to shift the compartment's region of activation. In this variant, the threshold is set to 7. All connections are given a very broad activation profile ($b = 0.1$) and relatively low (in comparison to the threshold) weights ($w = 2$). On their own, each connection cannot cause compartmental activation. In fact, even two or three connections together cannot exceed the compartment's threshold. Only where all four overlap is this possible. The maximum point of compartmental activation occurs at the centroid of all connections' regions of activation, or, in this case, the origin. The maximum compartmental output is reduced to approximately 0.35 because the point of maximum activation is shifted away from the

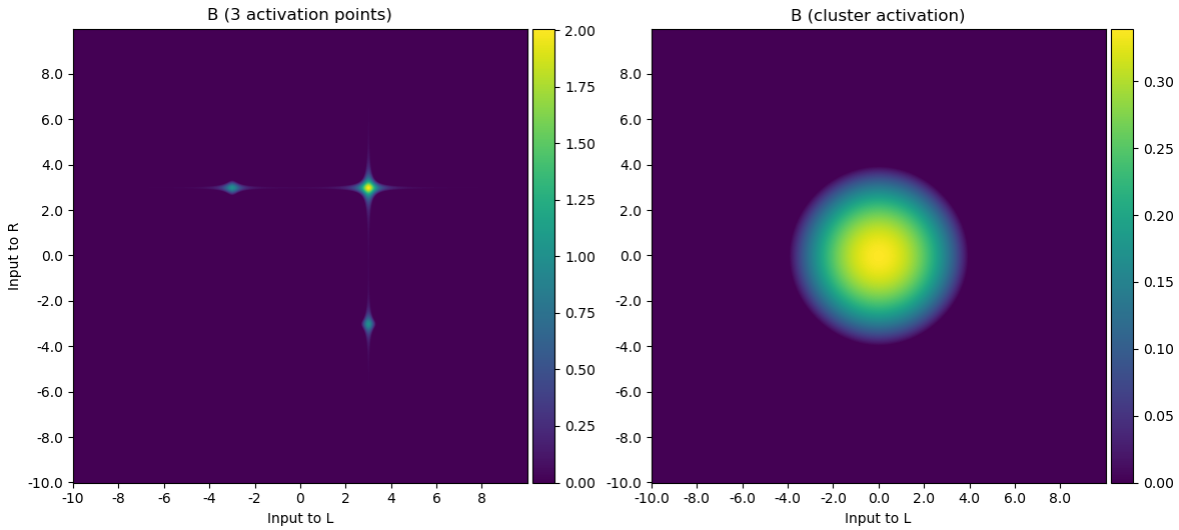


Figure 5.7: Examples of compartment B's possible behaviors using a compartment threshold and adjusting each connection's values for b and w. For this example, we have used $\phi(x) =$ ReLU(x) due to the possibility for negative output. v_e are unchanged from Figure 5.1. LEFT: Compartment threshold, t, equals 2. For all connections, $b = 1$. $w = 2$ for L1 and R1. For L2 and R2, $w = 1$. RIGHT: For the compartment, $t = 7$. For all connections, $b = 0.1$ and $w = 2$.

maximum's for each connection. In this case, the actual max is irrelevant since we are only interested in qualitative behavior.

5.5 Input Complexity

The input profile of an AD neuron model compartment does not require signals from many different sources to produce highly complex shapes. In Figure 5.8, we show that compartmental input, even from a single source, can get very interesting. Multiple connections from a single source can give a compartment the ability to respond in unexpected ways. For example, in the left plot, a compartment receives the same signal four times through different connections. Three of the connections have an identical profile (same b and w) but are centered at different points (-5, -1 and 2). The fourth connection, centered at 0, has a much broader region of relevance ($b < 1$) and higher strength; therefore, it dominates the profile. The second connection's center, $v_e = -1$, is close enough to the fourth's center, $v_e = 0$, to usurp it as the global maximum. Their

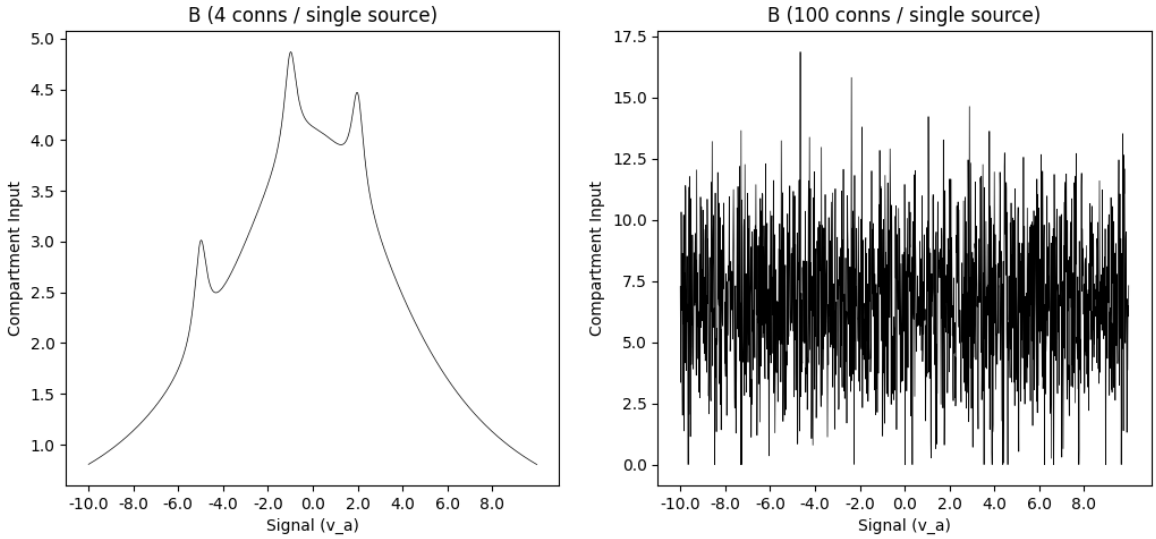


Figure 5.8: Compartments with a single source. LEFT: A single input source makes four unique connections with a compartment. The connections have the following parameters (v_e, b, w): (-5, 3, 1); (-1, 3, 1); (2, 3, 1); (0, 0.2, 4). RIGHT: A compartment has 100 connections from a single source. All connections use the version of ϕ with an added sine in the denominator given in Figure 5.3. Each connection is given random parameters with the following bounds. v_e: $[-\pi, \pi]$, b: $[0.1, 100]$, w: $[0, 1.01]$. These bounds have no significance other than that they produce interesting behavior.

combination shifts the global point of maximum activation left to -1. The other two connections create local maxima at -5 and 2. Alone each point of peak activation would be equal; however, since they overlap with the broadly activating connection at different places they have different peak values.

The right plot shows just how extreme these profiles can get. An AD neuron model compartment is given 100 connections from a single source. Each connection's parameters are randomly chosen (see figure caption for the random ranges) and the connection itself uses the sine-wave variant of ϕ given in Figure 5.3. The result is a visually unpredictable mapping of signal value to compartmental input strength as the input is the sum of 100 different sine waves. For every local peak in the profile, one can look left or right and find a nearby input value that produces a near zero output.

These examples show that multiple and duplicate connections can generate complex input profiles for the compartment.

5.6 Signal Coincidence

As we have shown above, the response of the AD neuron depends on the relevance and importance of an input signal. But it also depends on the amount of coincidence between signals arriving along the dendritic tree. To highlight how signal coincidence is used by the AD neuron, we ran several experiments using a type of signal coincidence that exists in the biological brain: *directional selectivity* [162, 23]. Directional selectivity is ability of some neurons to respond to a series of input signals whose order of arrival matches their arrangement along the dendrite. The correct order will produce a stronger response from the neuron than identical signals arriving in a different order. To show that the AD neuron can mimic directional selectivity, and therefore more complex versions of signal coincidence, ten AD neuron model compartments were connected together to form a single chain (Figure 5.9). Together, the ten compartments form the dendritic neuron, D. Each compartment is ordered from d_1 to d_{10} corresponding to the flow of input. The nine upstream compartments (d_1 to d_9) are B-compartments and d_{10} is an S-compartment (or between branch and soma compartments).

The separation between compartments, v_e, in D models a unit-less propagation time and not a specific value. This is the time it takes for a signal leaving one compartment to arrive at another compartment. Because we are only interested in the timing of input and, in this toy experiment, each compartment only receives one external signal, connections have a weight of 1. v_a, the input to ϕ, is not the signal's magnitude but a temporal difference between a signal's time of arrival, t_a, and the current time, T. Therefore, $v_a = T - t_a$. For all connections, $v_e = 10$. For example, a signal arriving at d_1 takes 10 time steps to reach d_2. ϕ uses the inverse quadratic RBF.

We ran two variants of the task using different compartment activation functions (ψ). In the left figure, $\psi(x) = \tanh(x)$. And in the right, $\psi(x) = \text{ReLU}(x)$. For each variant, five input sequences were tested: forward, forward (fast), forward (slow), backwards and

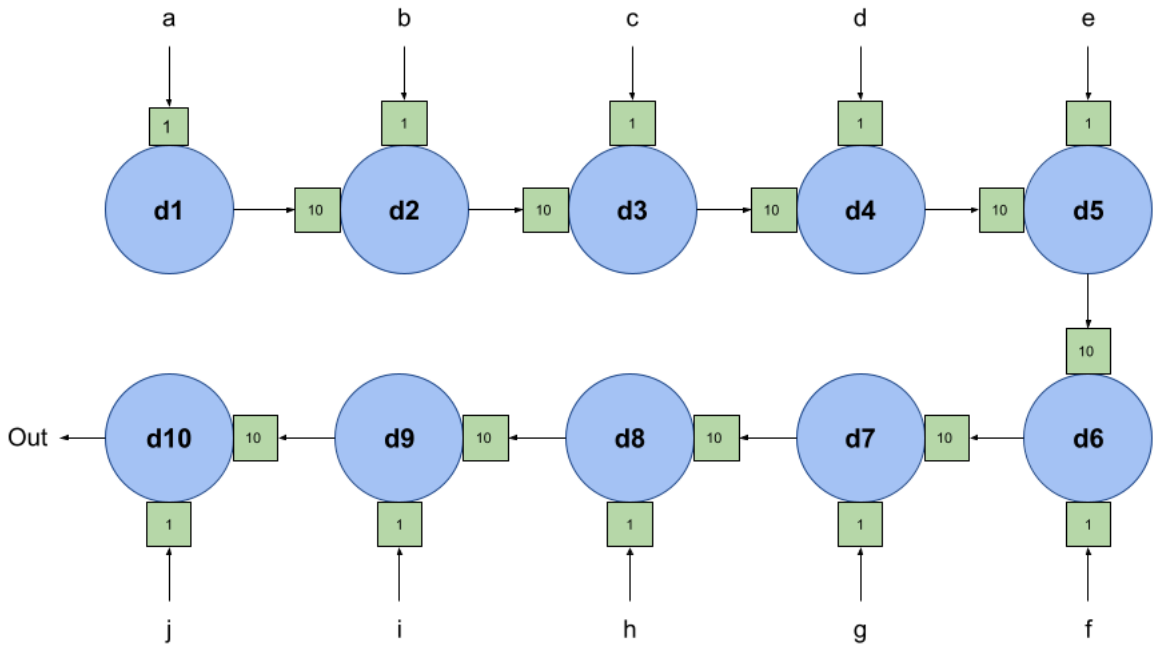


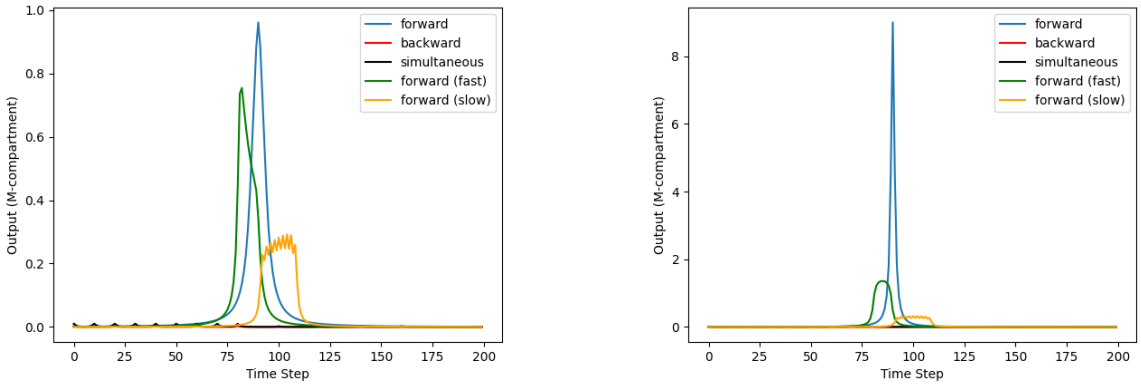
Figure 5.9: Diagram of dendritic neuron D. Letters denote input sources. Each blue circle labeled d_i is the i_{th} dendritic compartment. Green squares are connections and the number in the green square is the value for v_e.

	a	b	c	d	e	f	g	h	i	j
Forward (10 ts)	0	10	20	30	40	50	60	70	80	90
Forward (9 ts)	0	9	18	27	36	45	54	63	72	81
Forward (12 ts)	0	12	24	36	48	60	72	84	96	108
Backward	90	80	70	60	50	40	30	20	10	0
Simultaneous	0	0	0	0	0	0	0	0	0	0

Table 5.1: Times of arrival for inputs during the different sequences. Numbers given are unit-less time steps.

simultaneous. Each sequence specifies when an input signal arrives at each compartment. The timing of each input sequence is given in Table 5.1.

In Figures 5.10, the perfect *forward* example depicts an input sequence that matches both the compartmental order (d_1 to d_{10}) and the separation between compartments. Signals arrive starting at d_1 when $T = 0$ and arrive at adjacent compartments 10 time steps later: $T_{d_i} = T_{d_{i-1}} + 10$ where T_{d_i} gives the time of arrival of a signal at compartment d_i. The *backward* example reverses the input sequence. d_{10} receives a signal at $T = 0$ and d_1 when $T = 90$. The *simultaneous* example depicts AD neural output when all inputs

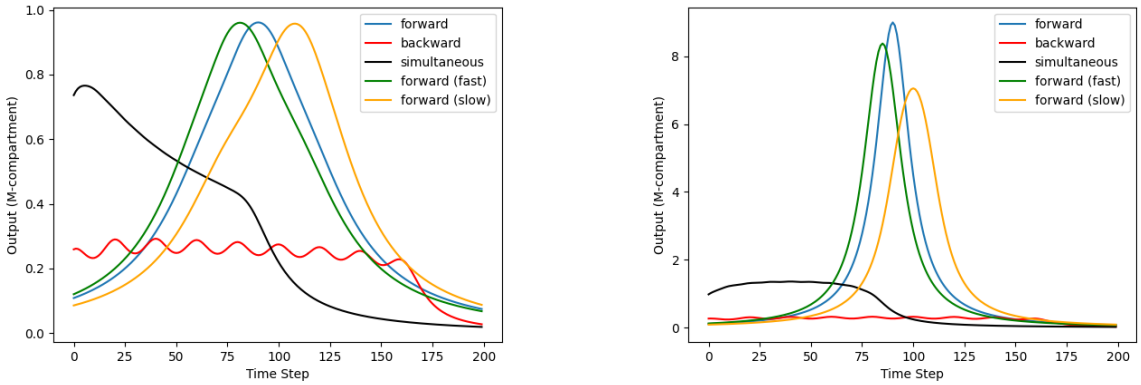


(a) Tanh (b) ReLU

Figure 5.10: Directional selectivity. 10 AD neuron compartments connected as a single chain where $v_e = 10$ and $v_a = T - t_a$. T gives the current simulation time and t_a is the time of input arrival. For all experiments, $b = 1$ and $w = 1$. (a) ψ uses tanh. (b) ψ uses ReLU.

arrive at the same time ($T = 0$). Finally, two variants on the forward test are shown in which the sequencing is the same but the timing is slightly off. In the *fast* variant, input arrives at the next downstream compartment after 9 time steps; whereas, in the *slow* variant, it arrives after 12 time steps.

In both experiments depicted in Figure 5.10, the forward sequence (blue) produces the greatest somatic (S-compartment) response. Since input times match compartmental order, upstream signals collide at downstream compartments producing stronger input to the next compartment. The variants of ψ produce significant differences in the peaks of the forward variants. *tanh* normalizes the final signal. The two slightly out-of-phase sequences produce peaks that are well above base line. The faster sequence (green), which is out of phase by only 1 ts, reaches a peak near the optimal case (blue). The slower sequence (yellow), off by 2, produces a peak only 20-30% of the optimal. The fast sequence produces a peak whereas the slow sequences produces more of a plateau. The simultaneous sequence barely registers as a series of evenly spaced peaks. The backward variant, although obscured in the figure, produces similar responses; however, the timing of somatic responses suffer long delays which extend far off the timeline. This demonstrates that



(a) Tanh (b) ReLU

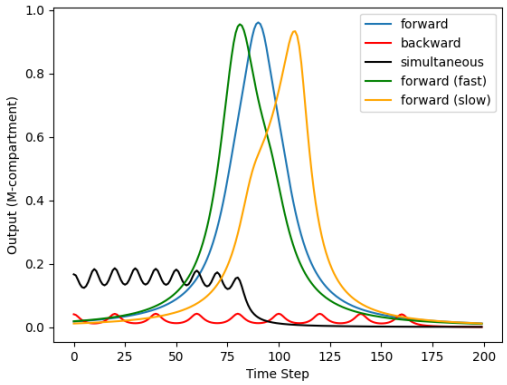
Figure 5.11: Experiment from Figures 5.10 with $b = 0.1$.

subtle differences in signal coincidence are clearly differentiable in the neuron's output. And drastically mismatched sequences produce little to no response in the S-compartment. Taken together, this shows that an input sequence with too much randomness would be unlikely to produce a response from the M-compartment.

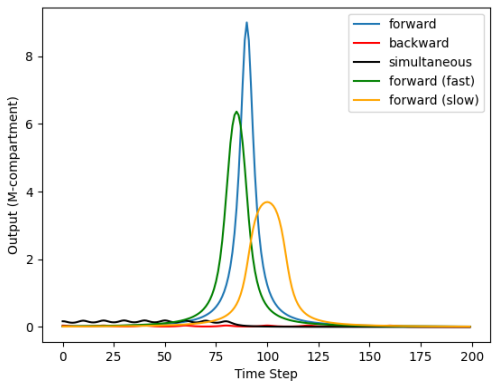
In the ReLU variant, out-of-phase sequences show much less distinction. They are a fraction of the peak of the optimally timed sequence. This reinforces what was stated in a previous section, the effects of signal coincidence do not just depend on the response profile, b, but also on the compartmental activation function, ψ. If the M-compartment normalizes its output, optimal signal coincidence is less important to the neural response.

To highlight the impact of b on neural behavior due to signal coincidence, the experiments were repeated several more times with $b = 0.1$ (Figure 5.11), or a very broad response profile; $b = 0.25$ (Figure 5.12), a somewhat broad response profile; and $b = 4$ (Figure 5.13), a narrow profile. Altering b has interesting effects and interpretations.

When the response profile is broader, slightly out-of-phase forward sequences reach approximately the same peak. Distinguishing between out-of-phase sequences relies more on ψ. For example, Figure 5.11a, in which $\psi = \tanh$, output peaks at approximately the same value for all forward sequences. Changing ψ to ReLU allows the AD neuron to

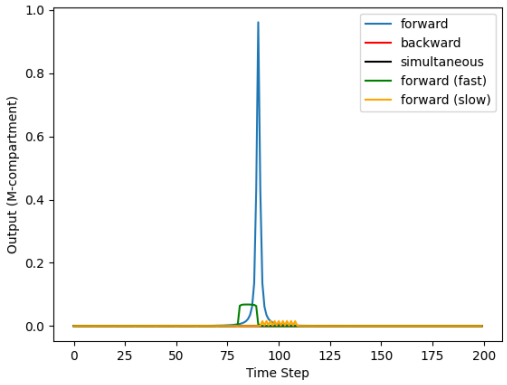


(a) Tanh

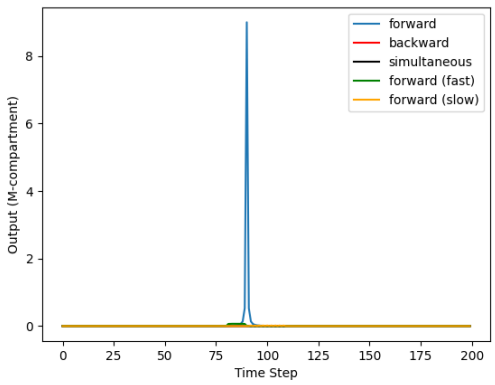


(b) ReLU

Figure 5.12: Experiments from Figures 5.10 with $b = 0.25$.



(a) Tanh



(b) ReLU

Figure 5.13: Experiment from Figures 5.10 with $b = 4$.

differentiate between forward sequences even when $b = 0.1$. The distinction between slightly out-of-phase sequences increases as b increases (Figure 5.12b).

But the reverse is true for the narrow profile (Figure 5.13). Slightly out-of-phase forward sequences are no better at producing a somatic response than very out-of-phase sequences (simultaneous and backward). In general, a narrowing of the response profile decreases the window for signal coincidence within the dendritic tree. It shinks a compartment's region of activation. Anything other than a perfect match is ignored. To compensate, normalization at the S-compartment can scale the different neural responses to improve comparisons but for $b >> 1$ normalization has a limited effect. In general, wider

response profiles give the compartment wiggle room to respond to sequences with a small amount of jitter. On the other hand, narrow profiles make the AD neuron a precise tool. Together these show how the response profile, b, can be used to capture the level of signal coincidence.

The potential benefits of AD neuron behavior is not limited to its ability to separate peaks. Figure 5.11 demonstrates how a broad response profile can be used to allow certain sequences to prime an entire branch of the tree making it more likely to produce output on subsequent input. For example, the simultaneous input sequence (black) produces an elongated signal. A single input pulse at time 0, spread over the length of the dendrite, causes the AD neuron to produce elevated output for nearly 100 time steps. In 5.11a, the signal resembles an exponential decay. In 5.11b, simultaneous input produces a plateau-like signal. This elevated state could be used to prime one dendritic branch over another such that a pulse arriving at a competing branch would be ignored in favor of this elevated one. The pulse would combine with the plateau-like signal, similar to the left example in Figure 5.8, allowing for selection between two identical inputs. The initial signal which primes an entire branch could come from a single source that makes multiple independent connections to the receiving neuron. Or it could come from a single source with a very broad response profile (small b). Since the priming signal persists over time, it allows the dendrite to bridge temporal gaps between pieces of information.

These experiments demonstrate the AD neuron's ability to capture signal coincidence along the dendritic tree. While an AD neuron's parameters and activation functions do have an impact on how it responds to different sequences of inputs, the shape of the tree determines whether the AD neuron responds at all. Shape determines which input sequences receive a response and compartmental level parameters adjust how closely sequences must match the neuron's shape.

The dendritic tree in these examples is a simple non-branching chain of compartments. Next, we examine which effects dendritic branches have on the overall response.

5.7 The Role of Branching

Dendritic branching is the bifurcation of dendritic paths as they extend away from the S-compartment. Specifically, a dendritic branch is a point in a dendritic tree where one compartment (B or S) receives input from two or more B-compartments. We have termed this many-to-one connectivity, *backward branching*. *Forward branching* is the one-to-many connectivity of the S-compartment. This section is only concerned with backward branching (henceforth just called 'branching').

This section explores how branching impacts the behavior of the AD neuron. Branching impacts not just the set of functions a particular AD neuron configuration can approximate, but also how well it mimics each function.

In the previous section, we showed how a single chain of compartments allow signals to reinforce each other. It shows how they combine incrementally. However, certain scenarios cannot be accomplished with single chains. Imagine, for example, a neuron that must respond to both a 'forward' and 'backward' sequence but not a random or simultaneous one. An AD neuron capable of recognizing multiple exclusive input sequences requires some form of selection. The Boolean behaviors presented in Figure 5.6 depict how an AD neuron could be configured to recognize both patterns. Configuration D describes how to create an OR-like combination. If inputs A and B originate from two chains of compartments each matching the forward and the backward sequence, the output, $f(A, B)$, will be high when either sequence is present.

Figure 5.14 gives more a complex example of the role of branching and demonstrates how each branch works as an independent interpreter of upstream information. Fifteen compartments are connected in the shape of a balanced, symmetric and full binary tree to form a single AD neuron. Compartment 0 is an S-compartment and compartments 1 through 14 are B-compartments. The neuron receives eight distinct inputs, A through H.

The AD neuron has two important features. First, $v_e = 2$ for compartments 1 and 2 whereas for all other compartments $v_e = 1$. Second, the S-compartment, 0, defines ψ to be

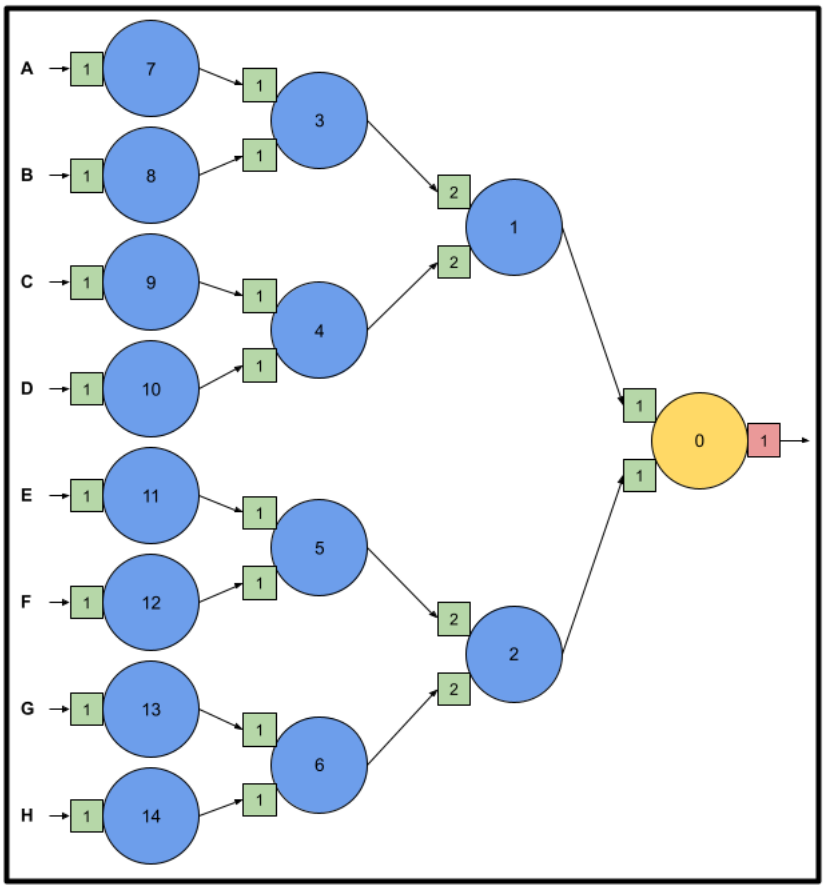


Figure 5.14: Diagram of the AD neuron configuration used for the branching examples. Blue circles are B-compartments (branch) and yellow indicates the S-compartment. Compartment numbers are labels. The green boxes represent ϕ and the number in each green box is the value of v_e. The letters are the input channels. ψ is not shown for compartments 1 through 14 because it is the identity function, $f(x) = x$. ψ in compartment 0 is the inverse quadratic RBF which is also the RBF used by all ϕ of all connections. All connections have a lower threshold of 0.05 under which a signal is set to 0 to handle the zero problem (see Section 9.2.8).

an RBF with $v_e = 1$. This differs from all other B-compartments which use the identity function: $\psi(x) = x$. The leaf compartments exist just to receive input; they do not transform the input. They are included to demonstrate when each input is active. The next level of compartments (3, 4, 5 and 6) combine pairs of leaf compartment outputs. These four compartments connect to either compartment 1 or 2. The connections to 1 and 2 have a $v_e = 2$. This makes compartments 3, 4, 5 and 6 functionally identical to the AND compartment depicted in Figure 5.5. Only when both inputs of these compartments are

active will compartments 1 and 2 receive a signal. Lastly, compartments 0, 1 and 2 together compute XOR. For compartments 1 and 2, their connections to compartment 0 will only activate if only one of their connections was active. For example, if compartments 3 and 4 each send an output of 2 to compartment 1, compartment 1 will also produce a 2 because both of its connections gave an input of 1. This does not match the v_e of the connection between compartments 1 and 0. Compartment 0 functions similarly because its RBF version of ψ has a v_e of 1.

Taken together, the AD neuron in Figure 5.14 recognizes the expression:

$$((A \wedge B) \oplus (C \wedge D)) \oplus ((E \wedge F) \oplus (G \wedge H)) \tag{5.2}$$

Figures 5.15, 5.16 and 5.17 show the AD neuron's behavior for different combinations of inputs. In each figure, sub-plots show the output of each of the neuron's fifteen compartments. Active inputs are given values from a contiguous range of values from 0 to 2 with a step of 0.1. A range of input values are used rather than a single value to show compartment behavior in the input space surrounding v_e.

Figure 5.15 depicts the activation of inputs which would satisfy the expression 5.2. Inputs A and B activate compartments 7 and 8. Compartments 7 and 8 drive a doubly strong response from compartment 3 which is a match for compartment 1's v_e of 2. Compartment 0 responds because ψ requires that one and only one of its inputs is active. The AD neuron's output suggests that the expression 5.2 is satisfied when A and B are approximately 1 and the rest are 0.

Figure 5.16 shows the AD neuron's behavior when inputs A, B, G and H are active. Compartments 7, 8, 13 and 14 activate. These activations propagate into 3 and 6 which also activate, producing an output of 2. Again, the output value of 2 causes both compartments 1 and 2 to activate. However, the M-compartment does not respond. This is due to compartment 0's ψ expecting a value of 1. Because ψ's $v_e = 1$, the B-compartment expects only one sub-tree to be active. In this case, the AD neuron inhibited non-matching

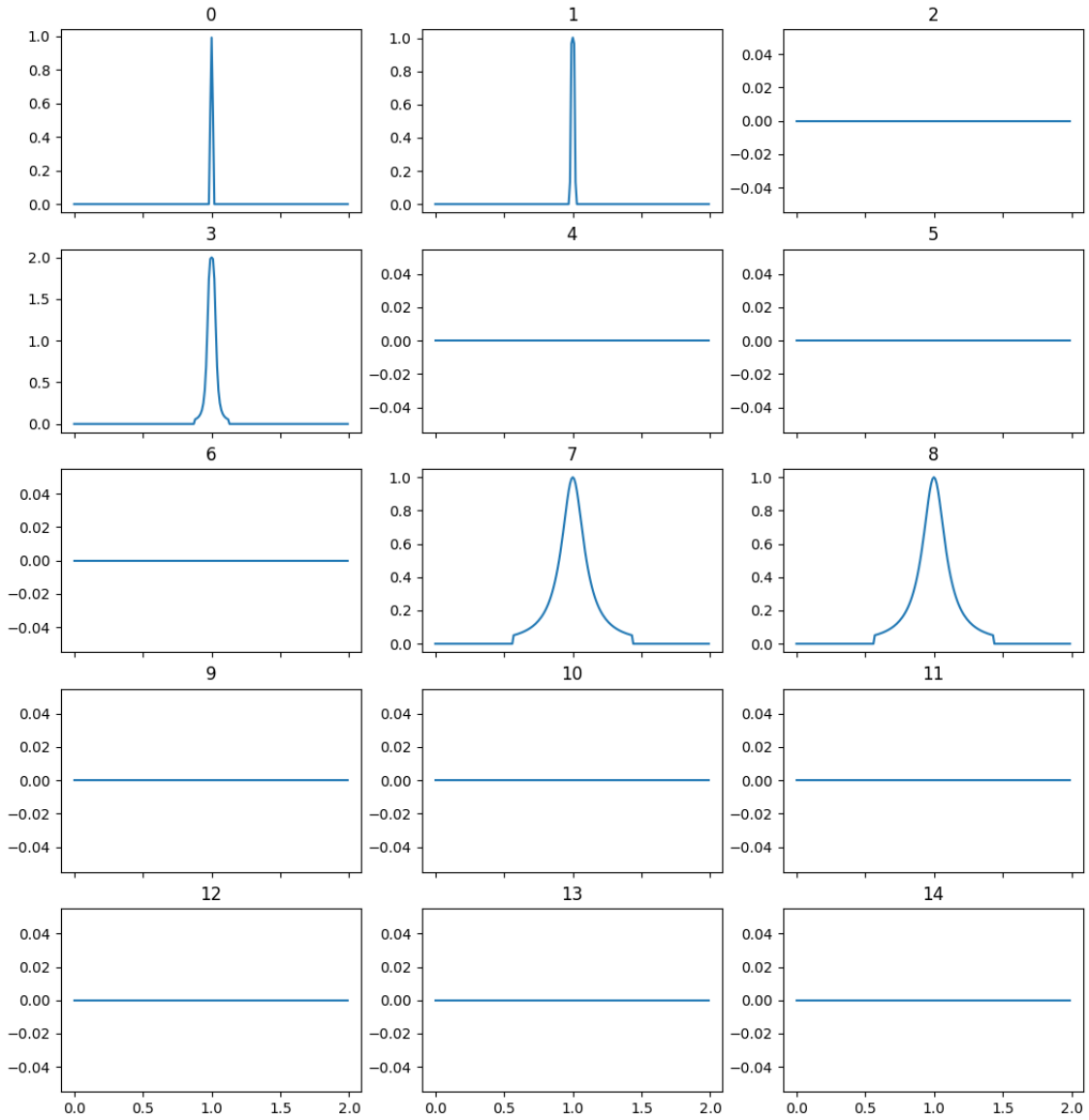


Figure 5.15: Compartment-level output of the AD neuron in Figure 5.14. Compartmental responses are show for a range of values, 0 to 2, for inputs A and B. Input: x-axis. Output: y-axis.

activity late in dendritic tree (at the root). In fact, calling it 'non-matching' is not quite correct. For two sub-trees, 7-8-3-1 and 13-14-6-2, the input did match. It only did not match at the root. This demonstrates the quasi-independence of dendritic branches in that an isolated match between input value and tree shape can be overridden by subsequent processing.

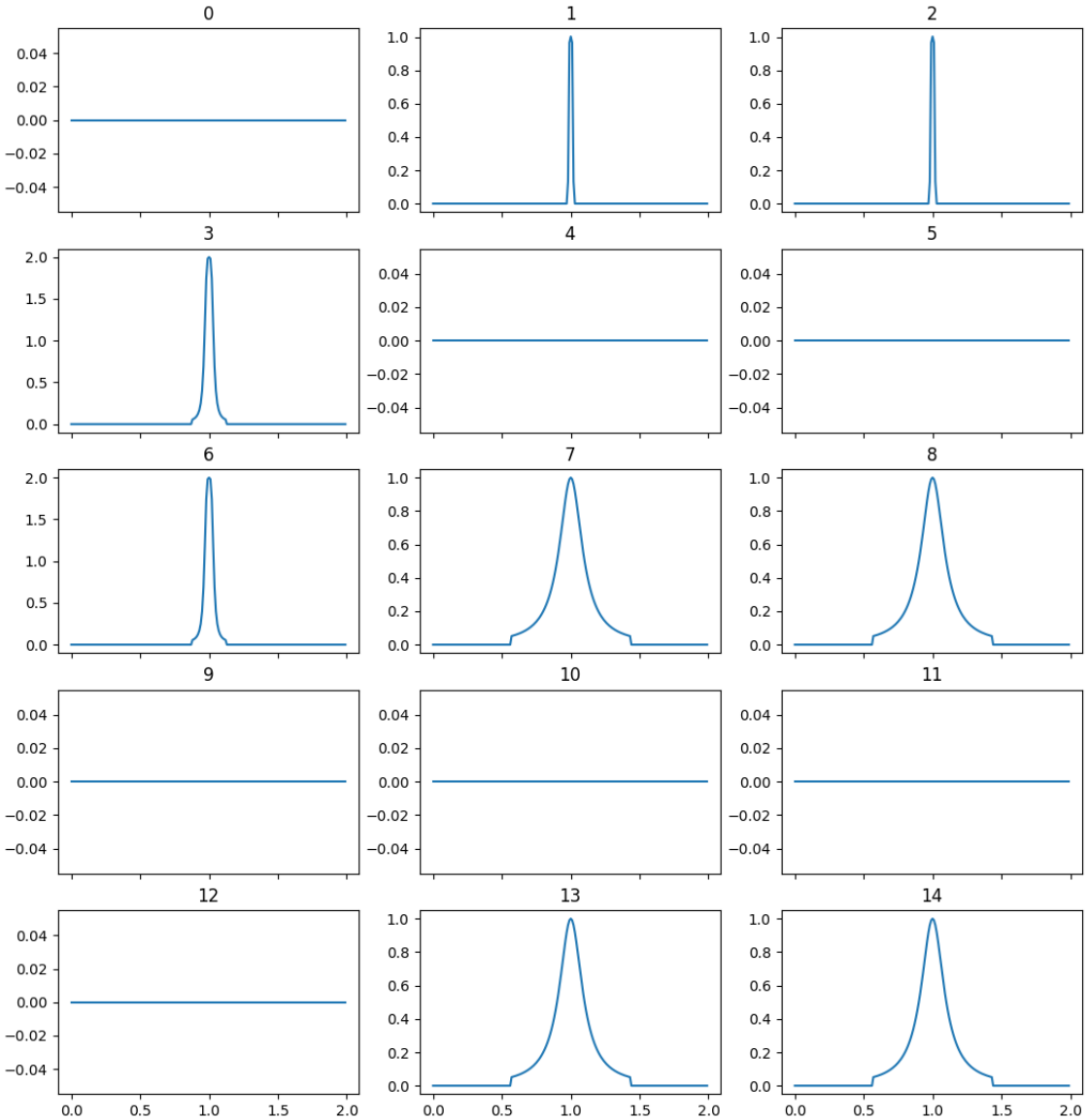


Figure 5.16: Compartment-level output of the AD neuron in Figure 5.14 with inputs A, B, G and H active. Input: x-axis. Output: y-axis.

Figure 5.17 shows another example of the inhibition of non-matching activity but with a positive outcome. Again, A and B are active. But so are C, E, and G. Since compartments 1 and 2 have an expected value of 2 for all inputs, the outputs of 4, 5 and 6, which are only 1, fail to generate a downstream response. Compartment 3's activation causes compartment 1 to activate. And the S-compartment subsequently responds because only one of its sub-trees are active. Here is a case where insufficient input is suppressed within the dendritic tree such that it does not reach the soma (compartment 0).

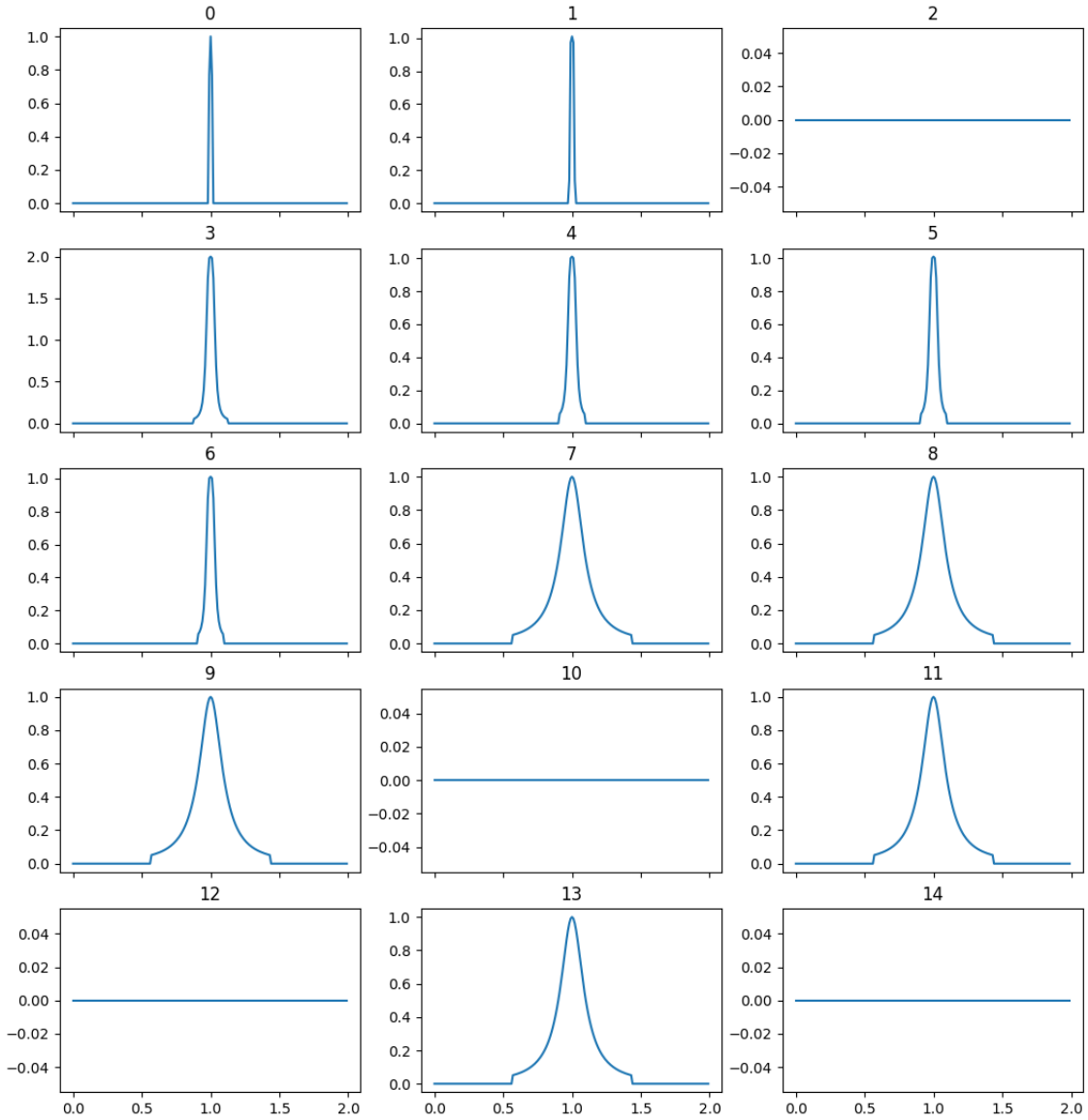


Figure 5.17: Compartment-level output of the AD neuron in Figure 5.14 with inputs A, B, C, E and G active. Input: x-axis. Output: y-axis.

Dendritic morphology impacts Boolean results. Even if two expressions are equivalent when constructed using AD neurons they can produce different results.

Altering morphologies can produce subtle differences in behavior even if the general Boolean behavior matches. For example, if the AD neuron is reconfigured in the shape of an unbalanced tree such that it evaluates the expression in the order of 5.3,

$$(((A \wedge B) \oplus (C \wedge D)) \oplus (E \wedge F)) \oplus (G \wedge H) \tag{5.3}$$

the output profile of compartment 0 when inputs A and B are 1 (same as Figure 5.15) changes. In the original version compartment 0 activates (above minimum threshold) in the range $(0.98, 1.02)$. The alternate, unbalanced configuration halves the range by narrowing it to $(0.99, 1.01)$. Thus, there are two ranges, $(0.98, 0.99)$ and $(1.01, 1.02)$, when inputs A and B are active, over which the balanced tree will satisfy the expression and the unbalanced one will not. By comparison, in the unbalanced tree, inputs G and H are closer to the S-compartment than all other leaf compartments. If they (rather than A and B) are given input, the response range is even wider than the range of the original, balanced configuration, $(0.978, 1.022)$. In other words, an unbalanced tree responds to input over a wider range when it is received closer to the S-compartment. The range of satisfying input narrows the deeper in the tree it is received. While both the balanced and unbalanced AD neurons capture the same Boolean expression generally, they do not have identical behavior over the entire range of input.

Finally, we conclude that a branch is a placeholder for an operation on one or more inputs. The actual operation depends on the parameters and composition of ϕ and ψ of the receiving compartment. The role of AD branching is to create separation within the overall input vector such that parts of it can be integrated incrementally and independently. This allows for AD neurons to form complex functions intentionally and with sparse connectivity.

5.8 Comparison and Summary

This chapter demonstrated a variety of AD neuron behaviors using an implementation of the extended model. The AD neuron model is capable of defining complex regions of activation with a strong degree of precision. It is capable of representing Boolean expressions, albeit fuzzily. We demonstrated in limited way that duplicate connections can be used to create multiple regions of activation.

As stated in previous chapters, the AD neuron differs from the point neuron in two important ways: nonlinear connections and the compartmentalization of sets of inputs. Each characteristic gives the AD neuron behaviors which are largely beyond the behavior of the basic point neuron.

First, the point and AD neuron models each have different strengths and weaknesses. The point neuron excels in situations where the desired region of activation is linearly separable. The Boolean operators of OR and AND fall into this category. The AD neuron can approximate point behavior through multiple connections or small values for b (Figure 4.7). The AD neuron excels at defining bounded regions of activation and so is better at approximating functions like XOR. We demonstrated this in the timing example where compartmental output first increased as the age of an input side approached the propagation time to the next compartment, and then, the output decreased as the age exceeded propagation time. The point neuron is not designed to differentiate between input which reaches its threshold and input which exceeds it.

Compartmentalization allows for situations which depend on a sequential and/or independent evaluation of subsets of the neuron's inputs. This is evident in the signal coincidence experiment and the branching experiment. The branching experiment showed how irrelevant input can be silenced within the dendritic tree such that it never impacts the somatic response. By comparison, the point neuron could silence certain inputs by zeroing their associated weights but in doing so it could not, then, respond to their signals.

In this chapter, we have demonstrated AD neuron behavior. Using different configurations, we showed that it is capable of representing Boolean expressions and other complex behavior. The tasks in this chapter, however, were all constructed by hand. For the model to be of use in truly complex, real-world tasks, it must be able to self-adjust its own parameters and connectivity. In other words, it must be trainable and that is the subject of the next three chapters.

CHAPTER 6
A METHOD FOR TRAINING THE AD NEURON

6.1 Overview

This chapter discusses the training methods and all associated expansions to the AD neuron model. A neuron model is of little use if its defining properties cannot be trained, so we begin by defining a trainable implementation. The defining properties of the AD neuron model are separation and compartmentalization. This chapter presents a set of algorithms that together modify an AD neuron using a combination of supervised and unsupervised learning.

The training process is divided into several modular algorithms. There are in order:

1. **Positioning:** the position of an input within the dendritic field.
   (a) **Angular Positioning:** the angular component of input position given by its latitude and longitude.
   (b) **Radial Positioning:** the radial component of input position given by its radius.
2. **Tree-building:** the defining of connections linking all inputs into one or more trees.
3. **Compartmentalization:** the grouping of inputs into compartments based on dendritic distances.
4. **Weight Modification:** the altering of connection weights.

Each algorithm builds on modifications made by previous ones. First, inputs are (re)positioned within the dendritic field by comparing the timing of input and output activity. Angular positions are the result of input-input coactivity. Radial positions are determined by input-output coactivity. Inputs are connected to each other based on these trained positions. The connected inputs are assigned to compartments based on relative

distances within the dendritic field. Weights are modified based on an input's contribution to compartmental output.

ψ, the compartmental output function, is implemented via a spiking neuron model. A dendritic field is defined which provides a metric to position inputs. The bulk of the chapter describes the algorithms outlined above.

6.2 Creating Separation Between Compartments

Before diving into the algorithms, there are several points of clarification that need to be made about how the AD neuron creates *separation* between compartments during the training process. The following does not alter the definition of the AD neuron model.

Each connection maintains a value (v_e) giving the separation between two compartments. In Chapter 4, separation was not defined differently for connections within an AD neuron or between AD neurons. AD neural learning depends on being able to train the separation between compartments. The following algorithms focus on single layer networks and therefore say nothing about larger, multi-neuron structures. This means that connections between neurons (e.g., between the input layer and output layer) are static. For our purposes, we declare all inter-neural measures of separation to be nil. This means that the metric of separation is only valid inside an AD neuron.

Figure 6.1 gives a visual of the implications of the above. The top diagram describes a full case where the connection C_1 crosses between two AD neurons, i.e., C_1 is inter-neural. C_2 connects a B-compartment (branch) to an S-compartment (soma) within the same neuron. In this case C_1 gives the separation between the S-compartment of one neuron to the B-compartment of another. The bottom diagram describes an alternative connection scheme in which inter-neural connections have no separation. Signals move from one to the other unaltered. Inter-neural signals arrive simultaneously at their destination. So the only separation between one S-compartment and the next are the intra-neural connections of the downstream neuron (C_1 of the bottom diagram).

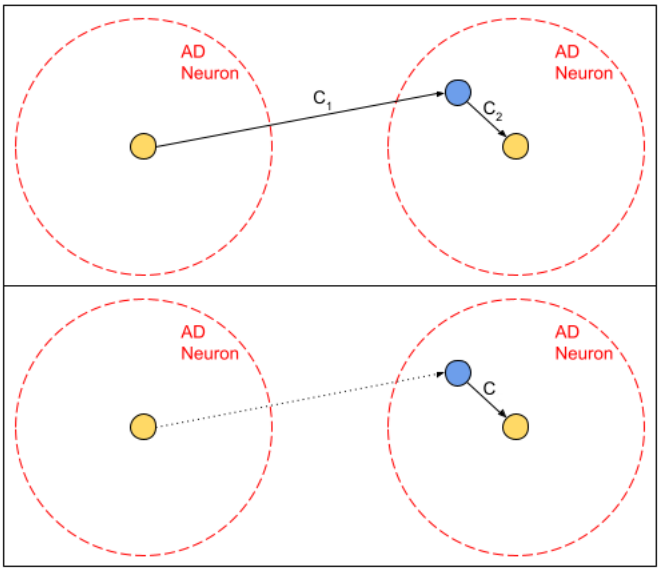


Figure 6.1: Diagram of two AD neurons with example connections. C_1 and C_2 are connections. Blue indicates B-compartments and yellow indicates S-compartments. Red outlines show neural boundaries. (Top) Two connected AD neurons in which inter-neural connections contribute a measure of separation. (Bottom) Inter-neural connections do not contribute to separation.

Next, we need to clarify another term for the output side of inter-neural connections. We will refer to the output end of such connections as *neural inputs* or just *inputs*. Neural inputs represent the points in the dendritic field at which input signals arrive. The training algorithm positions neural inputs within the dendritic field in order to build the dendrite. In this and following chapters, references to an input value received by the AD neuron use the term *signal* to differentiate them from neural inputs.

6.3 The Izhikevich Spiking Neuron Model

At the core of the AD neuron is a spiking neuron model. It forms the basic input-to-output machinery for all AD neural compartments. Spiking neuron models typically transform real-valued input into discrete temporal spikes. This ability is the reason one is used in our AD neuron implementation. The following subsections give a brief description of the Izhikevich neuron model [82].

Our implementation of the AD neuron model uses the Izhikevich spiking neuron model to create ψ, the output of a compartment. A spiking neuron model outputs a pulse, or spike, when input exceeds its threshold. This differs from rate-based models more commonly used in ANNs which produce a value when input exceeds a threshold. Spiking models generate a temporal encoding of their input--a series of spikes. The AD neuron's training process makes extensive use of the relative timing of spikes to modify input positions (see subsection 6.5.3).[1]

The Izhikevich model is a spiking neuron model whose behavior can be tuned to mimic a variety of biological neuron behaviors. It consists of two ordinary differential equations which model the cell's membrane potential, v and recovery, u. Equations 6.1 and 6.2 give the state change of the model.

$$C\dot{v} = k(v - v_r)(v - v_t) - u + I \tag{6.1}$$

$$\dot{u} = a(b(v - v_r) - u) \tag{6.2}$$

In the above equations, I is the total input to the neuron. a, b, c, d, k, v_r, v_t, v_{peak} and C are all fixed parameters that govern the behavior of the neuron. Using different values for these parameters, the Izhikevich neuron can model different classes of neurons (e.g., regular spiking, intrinsic bursting, fast spiking, etc.). All neurons in our model use the parameters for a regular spiking neuron that produces behavior typical of cortical pyramidal neurons. We chose this because, as the name suggests, it produces a regular, evenly spaced spike train for a given input. The parameters are given in Table 6.1.

A neural response, or spike, is produced if the value of v reaches a threshold value, v_{peak}. When this event occurs, v and u are reset according to Equations 6.3 and 6.4. v is the driver of the neural response and u is a dampener which prevents the neuron from producing back-to-back spikes.

[1]Why our implementation chose to use spiking neurons is discussed in a later subsection: 9.2.7.

Parameter	Value	Parameter	Value
a	0.03	b	-2.0
c	-50	d	100
k	0.7	v_r	-60
v_t	-40	v_{peak}	35
C	100	i_b	50

Table 6.1: The Izhikevich model neuron parameters used in the AD neuron.

$$v = c \tag{6.3}$$

$$u = u + d \tag{6.4}$$

We add one additional parameter to the Izhikevich model, i_b. The value of this parameter is sent to the spiking model as a baseline, constant input signal. i_b causes the neuron to "rest" just below its spiking threshold without having to alter the well-established Izhikevich model parameters. This gives the spiking model a hair trigger as depicted in Figure 6.2. The top plot depicts a regular spiking model with no baseline input. The purple area measures the length of time between the onset of an input signal (rise in the red voltage line) and the point the neuron reaches its spiking threshold (dashed black line). A spike is caused not only by the strength of an input signal but also its duration. Even modest input over long periods of time can cause a spike. The bottom plot shows how using i_b causes the neuron to rest just below its threshold. The result is that even subtle input will cause it to spike.

Why do we want this fast-acting behavior? The answer is control. The Izhikevich model was designed to mimic the behavior of a neuron receiving input from other neurons. This is an important point and is one of the stark differences between AI neural work and neuroscience modeling, which require that input likewise mimics neural signals. While we desire to use a spiking model, we did not want to be trapped into casting all AD neuron input into some biologically plausible space. For example, low-level signals to the Izhikevich model have trouble generating spikes. And neural behavior across the transition

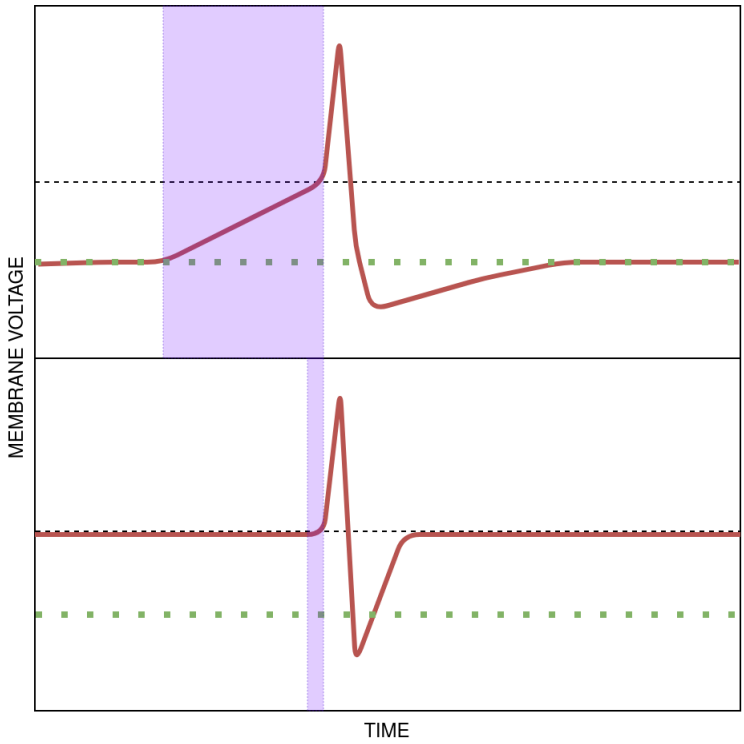


Figure 6.2: Neural behavior with (bottom) and without (top) baseline input. Horizontal black dashed line is the spiking threshold. The purple shaded area highlights the onset and duration of the input signal. The solid red line is the membrane voltage (v in the Izhikevich model equations). The green dotted line is the membrane resting value, u. Upper plot: typical neural response to input without i_b. Lower plot: neural response with i_b just below the neuron's spiking threshold.

from non-spiking to spiking input space can be dramatic. We desired to have a more linear gain on spike rate where small signals produced sparse spike trains and large signals produced dense ones. i_b, as simple a solution as it is, is the result of substantial (failed) experimentation trying to cast a variety of input into a biologically plausible range.

The Izhikevich model is used only to generate spike times, not the signals themselves. Connections receive spike times which they convert to real-valued signals through the RBF function φ. The signals are received by a connection's terminal compartment. The compartment, then, sums all signals it receives in a given time step. This combined input is used as input to the Izhikevich model. With enough input, the model spikes and the time of the spike is added to a queue of spikes which form the output of the compartment. Spikes are removed from the queue when their age reaches a learning window (the learning

Input	Neural Component	Output
Time	Connection	$\mathbb{R}$
$\mathbb{R}$	Compartment	Time

Table 6.2: Input and output types for each AD neuron component.

window was equal to A_{max}, given below in table 6.3). The queue of spike times are sent to the compartment's set of outgoing connections. And the process begins again. Table 6.2 gives a breakdown of each neural component and type of input it expects and output it generates.

As mentioned above, compartmental spike times are queued to form a history of recent output. The queue allows each compartment to store a precise record of recent activity. Without a queue, the most recent spike would erase previous spikes times. Keeping a queue allows all spikes to propagate the length of the dendrite. Also, a spike history speeds up the initial stages of training by allowing spikes to propagate over longer initial connections. The precise record of activity provided by the queue improves training performance in general since all aspects of training use the relative timing of spikes.[2]

6.4 The Dendritic Field

6.4.1 Description

In Chapter 4, a dendritic neuron was defined to be a single S-compartment and its dendritic tree of B-compartments. Since connected compartments maintain a degree of separation, we said that the separation between compartments creates a metric space. This metric space containing the dendritic neuron and all of its compartments is called the *dendritic field*. The origin of this space is the position of the S-compartment which for all experiments is $(0, 0, 0)$. The extent of the dendritic field is theoretically infinite, but it is useful to bound the space such that it contains all meaningful input positions. The

[2]While we do not have specific data or figures to back up this claim, we have tested a queued and queue-less version extensively. The compartmental spike queue was a later addition to our AD neuron implementation. For much of its initial development, we used the queue-less version. When the queue was added, results and training times improved substantially.

definition of a *meaningful position* depends on the hyper-parameters R_{max} and A_{max}. R_{max} and A_{max} define both the learning window used by various aspects of the training process and the radius and circumference of the field in temporal units. In general, a meaningful position is any position from which an input can connect to the dendritic tree and have a chance of impacting the behavior of some part of the AD neuron. In other words, any input outside the dendritic field is unable to connect to the tree and has no impact on neural behavior.

6.4.2 The Spherical Geometry of the Dendritic Field

The AD neuron's dendritic field is implemented as a spherical metric space. Spherical geometry deals with spheres and their surfaces. The geometric constraints of spherical geometry govern how compartments are positioned and move within the field and how separation-based relationships are measured.

A sphere is defined by a central point, C, and a line segment, R, referred to as the radius, with length L_R. The surface of the sphere, or shell, is defined by the set of all points in $\mathbb{R}^3$ which are L_R distance from C. The space, S_{CR}, contained within the shell given by C and R can be defined as all shells whose center is C and whose radius is less than or equal to L_R.

Any point within S_{CR} can be given by two angles and a radius. In spherical geometry it is customary to use ϕ, θ and ρ; however, the mathematical sources used for this project originate from the field of navigation.[3] Therefore, *latitude* and *longitude* have been substituted for ϕ and θ. The alternative labeling was also convenient to prevent the AD neuron from containing two versions of ϕ. This is not simply a relabeling as the systems do

[3]**A short word on sources for spherical geometry:** Unfortunately, mathematical sources and texts on spherical geometry and trigonometry are somewhat hard to find. For instance, the modern mathematical source I used as reference, a textbook once used at naval academies, dated to 1940 [85]. It is not that the subject has rarely been published; it is that the subject has not been published recently [159, 160]. According to Van Brummelen, the subject disappeared from math textbooks in the 1950s. Even more illusive are applied sources, such as finding a new point on a sphere given a heading and distance. It was not until I changed my search criteria to the subject of navigation (aerial and naval) that I found what I was looking for [175, 43]. It is for this reason I have included this, perhaps larger than expected, section on spherical maths.

differ. For example, the angle *latitude* ranges over $[-\frac{\pi}{2}, \frac{\pi}{2}]$. This differs from its counterpart ϕ that ranges over $[0, \pi]$.

From here forward *lat*, *lon* and *rad* will be used to indicate latitude, longitude and radius respectively in the equations. And, the short form of a point in spherical space is a triplet with the order: (lat, lon, rad).

A dendritic field F is defined as a spherical metric space with the S-compartment at the center, C, or $(0, 0, 0)$. The extent of the space is given by a maximum radius, R_{max}. Input position is given by a spherical point whose radius is bounded in [0, R_{max}]. Distances are measured differently between types of components. B-to-S compartmental distances are measured using only their radial differences. Since the S-compartment is always at $(0, 0, 0)$ within its own dendritic field, B-to-S distance is simply given by the B-compartment's radius. Angular offsets are ignored. B-to-B distance is given by the equation, D_{ij} (Equation 6.6). Equation 6.6 calculates the angle, $\angle ICJ$, between two spherical coordinates I and J ignoring any radial differences. [4]

$$C(i,j) = \sin^2\left(\frac{\text{lat}_j - \text{lat}_i}{2}\right) + \cos(lat_i)\cos(lat_j)\sin^2\left(\frac{\text{lon}_j - \text{lon}_i}{2}\right) \tag{6.6}$$

$$D(i,j) = 2\,\text{atan2}(\sqrt{C(i,j)}, \sqrt{1 - C(i,j)}) \tag{6.7}$$

6.4.3 Decoupling of Relationships

The reason the AD neuron uses spherical geometry is that it decouples each relationship from the other. Modification of a B-to-B relationship is purely angular. It

[4]In all equations, atan2(y, x) is defined mathematically by Equation 6.5[168]. The C++ code used to calculate it comes from the standard library function *std::atan2*.

$$\text{atan2}(y,x) = \begin{cases} \arctan(\frac{y}{x}) & \text{if } x > 0 \\ \arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases} \tag{6.5}$$

takes place across a unit shell of the dendritic field and involves no radial displacement. Similarly, a change to a B-to-S relationship involves a radial movement which does not alter the angular distance between B-compartments. Decoupling allows each type of relationship to be trained independently without altering the other. Furthermore, decoupling allows them to be measured independently.

Not only is distance calculated differently between types of compartments, but different parts of the training algorithm use different metrics to calculate the distance between B-compartments. Altering neural input position uses (and affects) only angular distances; whereas, tree-building uses both angular and radial distances. The reason for this is that each B-compartment has two distinct types of relationships within the dendritic field: B-to-B (angular) and B-to-S (radial).

Radial distances are based on R_{max}, which is the maximum radial distance between the S-compartment and a B-compartment. Likewise, angular distances are limited by A_{max}. Distances with the dendritic field use temporal units (e.g., time steps). Decoupling means that hypothetically R_{max} and A_{max} could utilize values independent of each other; however, we do not do this. For the dendritic fields in this work: $A_{max} = R_{max}\pi$

6.5 Training

6.5.1 Inspiration

Training the AD neuron is the process of capturing input-output and input-input relationships to (re)configure the dendritic tree in a way that is beneficial to solving a task. Training the AD neuron rests on several neuroscientific ideas. In general, it is the idea that correlated activity plays a role in determining connectivity. This has been stated as "cells that fire together wire together" [143]. Our approach to training the AD neuron stretches this idea to the sub-neural level, such that synapses that fire together wire together.

We also draw inspiration from research suggesting that clusters of synapses within the dendritic tree are important to neural computation [112, 25], and it comes from theories

about how more proximal inputs drive somatic responses versus distal ones [174]. From these broad-stroke ideas about the effects of correlated neural activity and synaptic position, we derive the idea that *correlated activity creates proximity.*[5]

The more two sources of input are temporally aligned in activity the closer their position should be on the dendritic tree and vice versa. If they are perfectly aligned then it is assumed they carry the same information. Likewise, the more in sync a specific input is with the AD neuron's S-compartment--its soma--the closer its position should be to the root, or terminus, of its dendritic path.

Proximity is solely determined by activity regardless whether activity leads to a correct or incorrect response. In other words, adjustments to an input's position is not affected by error estimates. This allows co-activity potentially to position both excitatory (positive weight) and inhibitory (negative weight) inputs in close proximity such that they are included in the same compartment. The choice not to subject neural input positions to error values made sense for the above reasons and it simplified the effects of training. Like many other aspects of the AD neuron, one could argue that input positioning should be, in some or all cases, subject to error values.

Once neural inputs are repositioned the dendritic tree is created in such a way to capture the relationships encoded in their relative positions. In other words, the dendritic tree captures the spatial relationships created by co-activity.

6.5.2 Initialization

Initialization is the process of placing the components of the AD neuron (e.g. inputs, weights, compartments and connectivity) into a proper starting state from which they can be trained. Placement strategy can affect the relative scale of the separation between

[5]We reiterate here from the neuroscientific background chapter that the precise function of synaptic arrangement is an open question. While dendritic spines and synapses grow and atrophy based on local activity, dendrites, once formed, largely do not. The idea that correlated activity creates proximity mixes neuroscientific research related to the early brain development, when neurons and dendrites are still forming, and the adult brain.

inputs and compartments, the shape and size of compartments and the time required for compartments to form. There are many possible starting configurations and initialization strategies. The version described below favors randomization when possible. When elements are placed or tuned by hand, we perform parameter sweeps, opting for the central values across the ranges which produce desirable behavior.

For the AD training algorithm to work, initialization must generate a preexisting dendritic tree which is capable of receiving and conducting input and, as a whole, is capable of producing output. The training algorithm requires that the AD neuron's initial state is viable. In general, threshold-based spiking neuron models require more tuning than their non-spiking counterparts. There are many configurations which are unable to produce spikes; therefore, a viable state is one from which an AD neuron starts in or can reach a spiking state. For example, as stated in the previous subsection, activity causes compartments to reorganize; therefore, if the initial state prevents output in the S-compartment, radial changes will never take place. To accomplish this, we define an initial set of basic parameters for inputs, compartments and connections which have a very high chance to produce a viable initial state.

The basic set of AD neuron hyper-parameters is given in Table 6.3. The purpose of each parameter is given throughout this text in the context(s) in which they appear. Experiments in the following chapter use this set of parameters and any alterations to these parameters are given alongside the experiment description. In fact, the parameters below were derived from the experiments described in chapter 7. They represent a good starting point from which to establish baseline behavior. Certain parameters (e.g., Bf and b) need not be identical across all layers or neurons, but we have not had grounds to vary them within the same experiment.

The S-compartment of the AD neuron is given the position (0,0,0). Every input to the dendritic field is placed within its own leaf B-compartment. Each of these B-compartments is connected directly to the S-compartment and given a radius of R_{max}. R_{max} is the

A_λ	0.001	A_{max}	$R_{max}\pi$
R_λ	0.001	R_{max}	500.0
W_λ	0.001	Bf	0.5
w	100.0	L_{max}	20
W_{max}	500.0	b	0.5

Table 6.3: Base AD neuron hyper-parameters. The three λ parameters are: (A)ngular learning rate, (R)adial learning rate and (W)eight learning rate. w is a trainable weight parameter. W_{max} is a static strength parameter which bounds the actual weight of both inputs and compartmental connections to $[-W_{max}, W_{max}]$. A_{max} and R_{max} are the angular and radial learning windows. Bf is the branch factor used to build the dendritic tree. L_{max} is the maximum compartment length. b is the RBF shape parameter.

maximum radius a compartment can have. $R_{max} = 500$ allows for long temporal B-to-S relationships. Input positions are initialized with a maximum radius because inactive inputs (inputs which receive no spikes) do not move from their initial positions. Were we to initialize input positions to a radius less than the maximum, inactive inputs would occupy middle parts of the dendritic tree. Doing so, they can cause inactive compartments to be created between two active compartments, creating a dead zone which silences the more distal compartments connecting through it.[6]

Unlike the radius, the latitude and longitude of each N-compartment is randomized using Equations 6.8 and 6.9. The function g returns a pseudo-random number from the uniform distribution over the given range. These equations insure a random distribution of compartment positions over the starting spherical shell.

$$\text{lat} = \arccos(2g(-\pi, \pi) - 1) - \frac{\pi}{2} \tag{6.8}$$

$$\text{lon} = \text{g}(-\pi, \pi) \tag{6.9}$$

Next, a basic tree is created. Connections are generated between every single-input B-compartment and the S-compartment. Each connection's v_e is set to R_{max}--the starting

[6]While we have experimented with less than R_{max} initial radii (and discovered the above mentioned problem), we have not experimented with truly random distributions of radii. It is possible there is some benefit to random placement which overrides this concern.

radius of all B-compartments--because they make direct connections to the S-compartment. Therefore, in its initial state, an AD neuron's set of compartments is equal in size to the number of its neural inputs.

Depending on the experiment, multiple AD neurons can be generated through the above process. Many experiments we ran used an input layer of spiking neurons. Other, simpler experiments, do not contain proper input layers but used an internal input generator to produce required signals. The main difference between these two input generation methods is that the former uses a layer of fully-fledged Izhikevich neurons to generate input spike trains, whereas the former generates spikes through a discrete Poisson process.

A discrete Poisson process subdivides time into small intervals, e.g., 1 ms, and assigns a probability of an event occurring within each interval. For each interval, a random value between 0 and 1 is chosen from a uniform distribution. If the random value is less than the probability, a spike occurs in that interval.

The source of input will be given with each experiment. When Izhikevich neurons are used as an input layer, they are not configured as AD neurons since their only purpose is to convert real-valued signal to spike trains.

6.5.3 Input Position and Distance

As we stated above, input position is separated into two parts: angular and radial. Angular position is defined by an input's latitude and longitude; whereas, radial position is just that, the radius. The angular distance between two inputs is given by d_a in Equation 6.11 which requires Equation 6.10.[7] In Equation 6.10, i and j are neural inputs. d_{lat} and d_{lon} give the differences between the two positions latitude and longitude.

$$a_{ij} = \sin^2(\frac{d_{lat}}{2}) + \cos(\text{lat}_i)\cos(\text{lat}_j)\sin^2(\frac{d_{lon}}{2}) \tag{6.10}$$

[7]They have been separated for clarity.

$$d_a = \text{atan2}(\sqrt{a_{ij}}, \sqrt{1 - a_{ij}}) \tag{6.11}$$

Radial distance calculation is not shown because it is simply the radius of a compartment, in the case of B-to-S compartment comparisons, and the absolute difference in B-to-B comparisons.

Distances between neural inputs and between inputs and the S-compartment encode the timing of signals, as we will demonstrate below. For example, if two inputs are on opposite (polar) sides of the dendritic field, then when one receives a signal at time t, the other on average receives a signal at time $t + A_{max}$. Angular and radial positions are stored with respect to the unit sphere and must be translated into the temporal domain using Equations 6.12 and 6.13.

$$s_a = \frac{A_{max} d_a}{\pi} \tag{6.12}$$

$$s_r = R_{max} d_r \tag{6.13}$$

Given an angular distance on the unit sphere, d_a, s_a gives the distance in time steps. Similarly for radial measurements, given a radial distance, d_r, s_r gives the distance in time steps.

6.5.4 Training Angular Position

In the trained AD neuron, relative positions encode *when* an input is active in relation to its siblings. When a spike arrives at an input, it is compared to all its sibling inputs in two ways. First, we find the temporal difference between the two input's spike times. This gives us their expected separation. We then find the temporal difference encoded in their preexisting separation across the spherical shell of the dendritic field. This gives us their actual separation. The input that has just received a spike moves (orbits) toward the point of expected separation from its current angular position along the line, across the spherical

shell, connecting the two positions. Radial differences between the inputs are ignored. During this process, we assume all inputs fall on a unit shell of the dendritic field as this reduces the number of calculations.

Angular movement only affects the currently active input. No movement takes place if the temporal difference between spike times exceeds A_{max}. We have found that limiting movement to temporal differences less than A_{max} is necessary for stability. A great circle passing through two inputs creates two measurements of separation.[8] The AD neuron always uses the shorter of two for the distance between two inputs. Therefore, activity beyond A_{max} would push an input away along one vector only to making it closer along the opposing vector.

The following series of equations describe how the angular component (latitude and longitude) of neural input positions are trained.

$$s_e = \pi \frac{t_i - t_j}{A_{max}} \tag{6.14}$$

Equation 6.14 gives the difference between spike times at neural inputs i and j encoded as an angular separation. This is the expected angular separation. t_i and t_j are the spike times of inputs i and j. A_{max} is the angular learning window. The range of s_e will always be in $[0, \pi]$.

Next, we find the actual angular separation between i and j using Equation 6.11.

$$s_{p_i p_j} = s_a - s_e \tag{6.15}$$

Equation 6.15 gives the difference between the expected and actual angular distances of p_i and p_j, where p_x denotes the position of input x. If $s_{p_i p_j}$, is positive, neural input i will move toward its sibling j. If it is negative, i will move away from j. In order to accomplish

[8] A *great circle* of a sphere is the circle created by passing a plane through the sphere at its origin. For any two points on the surface of a sphere, there is only one great circle which connects them.

this move, we must first find the heading, or vector, going from i to j along the shortest line between them.

$$\begin{aligned} h_{p_i p_j} = \text{atan2}(& \\ & \sin(\text{lon}_{p_j} - \text{lon}_{p_i}) \, \text{lat}_{p_j}, \\ & \cos(\text{lat}_{p_i}) \sin(\text{lat}_{p_j}) - \sin(\text{lat}_{p_i}) \cos(\text{lat}_{p_j}) \cos(\text{lon}_{p_j} - \text{lon}_{p_i}) \\ &) \end{aligned} \tag{6.16}$$

Equation 6.16 gives the heading from p_i, the currently active input, to p_j.

$$\begin{aligned} \text{lat}_{new} = \arcsin(& \\ & \sin(\text{lat}_{old}) \cos(s_{p_i p_j}) + \\ & \cos(\text{lat}_{old}) \sin(s_{p_i p_j}) \cos(h_{p_i p_j}) \\ &) \end{aligned} \tag{6.17}$$

$$\begin{aligned} \text{lon}_{new} = \text{lon}_{old} + \text{atan2}(& \\ & \sin(h_{p_i p_j}) \sin(s_{p_i p_j}) \cos(\text{lat}_{old}), \\ & \cos(s_{p_i p_j}) - \sin(\text{lat}_{old}) \sin(\text{lat}_{new}) \\ &) \end{aligned} \tag{6.18}$$

Using $s_{p_i p_j}$ and $h_{p_i p_j}$, Equations 6.17 and 6.18 give the new latitude and longitude of the active input i. The *old* and *new* subscripts to latitude (lat) and longitude (lon) specify the previous or new values.

The above comparison and movement process is repeated for all sibling inputs j, where $j \neq i$ and whose most recent signal falls within the angular learning window, A_{max}. To facilitate these movements, they are applied to a copy of input i's position. We distinguish the two positions as the original, p_i, and the modified, $\hat{p}_i$. $\hat{p}_i$ is moved instead of the original because we must first locate the target position before applying a learning rate.

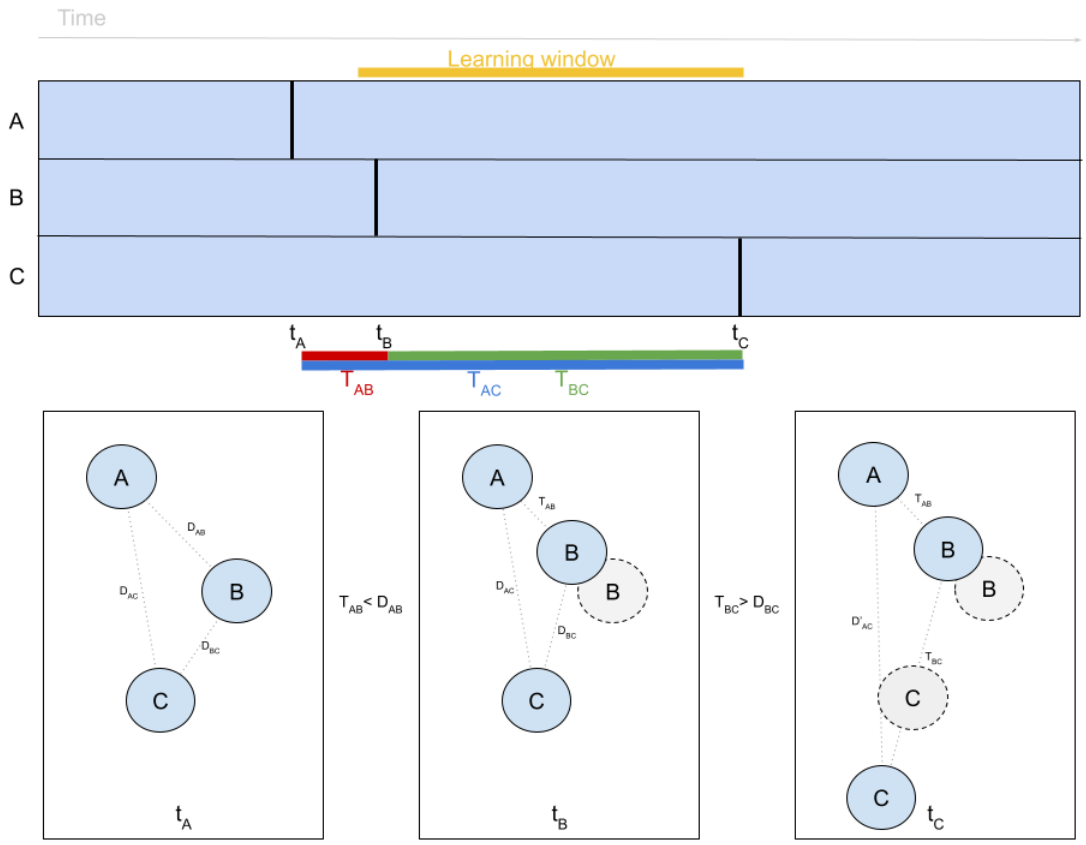


Figure 6.3: An example of how angular movement depends on the relative timing of signals. The time line, in light blue, gives the arrival time of three signals, t_A, t_B and t_C. Yellow bar is the learning window. The bottom three diagrams describe how the arrival of each signal impacts input position. Light blue circles are the current positions (time moves left to right); light gray circles reference previous positions. Dotted lines show the angular separation between positions. Spherical coordinates are flattened and these plots assume that $A_\lambda = 1$.

Once all i, j comparisons have been made and the target location has been located, p_i is moved toward $\hat{p}_i$ using the same set of Equations 6.15, 6.16, 6.17 and 6.18. The one difference is that Equation 6.15 has a learning rate applied to it to make Equation 6.19. A_λ is the angular learning rate.

$$s_{p_i \hat{p}_i} = (s_a - s_e) A_\lambda \tag{6.19}$$

Figure 6.3 presents an example of how signal times alter input positions. At the top, the light blue plot shows the timing of signals for three different inputs (A, B, and C)

within the dendritic field of an AD neuron. Black vertical lines represent the signal times. Each input receives a single signal in the depicted window. For example, input A receives a signal at t_A. The green, red and blue colored bars below the plot give the temporal interval between signals arriving at different inputs. The three diagrams at the bottom depict how the position of the inputs changes with the arrival of each signal.

The arrival of a signal at A, at time t_A, has no impact on input position because no signals to B or C have arrived previously. The arrival of a signal to B, at time t_B, causes B to move toward A. It moves toward because the interval T_{AB} is shorter than their current distance, D_{AB}. Previously, the separation between A and B was D_{AB}. Now it is T_{AB} to reflect the relative interval between input signal times. With B's movement toward A, the distance between B and C also changes. This relative change is not haphazard or necessarily unwanted. B's movement toward A adjusts B's relationship to C *with respect to* A. In this way, B partially adopts A's relationship with C.

Lastly, a signal arrives at C, at time t_C. C's movement is made with respect to its position to B and not A. This is because the interval T_{AC} is longer than the learning window, so A's signal (and position) is ignored. Therefore, C moves only with respect to B. Unlike B to A, C moves away from B because, as the inequality to the left of the diagram shows, $T_{BC} > D_{BC}$. In other words, the actual separation is shorter than the expected. Although, C's movement was made only with respect to B, due to B's proximity to A, C also moves away from it, too.

This small example shows how inputs alter their angular positions due to the relative timing of signals. If the pattern depicted at the top of the plot were typical for some set of input examples, A and B would cluster together away from C.

Neural inputs can be moved by direct comparisons of signal times to relative positions. But there exists a transitivity within the field. The relative positions of two inputs which do not move by direct comparison will move if they have an co-active siblings in common.

This also implies that the angular positions of two neural inputs without any indirect relationships is meaningless.

For a single neural input, training angular position requires linear time in the number of inputs to the dendritic field. Angular movement only occurs when an input receives a signal. Therefore, training time depends strongly on the level of activity received in the dendritic field.

Finally, an input's role in the dendritic field *is* the set of distances to all other inputs, not its actual position. An AD neuron is invariant under angular rotation so long as all neural inputs rotate along the same heading. An angular position alone is meaningless.

6.5.5 Radial Change of Position

Whereas, angular change captures relationships between input-input timing, radial position changes due to input-output timing. When the AD neuron's S-compartment produces output all inputs which have received a spike within the learning window alter their radial position. Radial movement follows the same steps as angular movement. Since radial movement happens along a single axis, there is no need to calculate a heading other than the sign of the displacement.

$$r_e = (t_m - t_i)R_{max} \tag{6.20}$$

Like Equation 6.14, Equation 6.20 uses the temporal separation of signals to calculate the expected radial separation. t_m is the timing of an S-compartment spike and t_i is the most recent signal received by input i. r_e is in the range $[0, R_{max}]$.

$$r_{p_m p_i} = (r_a - r_e)R_\lambda \tag{6.21}$$

Next, we need to calculate how far input i should move toward or away from m. Equation 6.21 gives this displacement as the difference between the actual, r_a, and expected, r_e, radial separation between i and m. This amount is scaled by the radial

learning rate, R_λ. $r_{p_m p_i}$ is then applied to the radial component of p_i, the position of input i. Like the angular training algorithm, this process is repeated for all i in the dendritic field.

In summary, a single S-compartment spike causes all inputs belonging to its dendritic field to reevaluate their radial position with respect to the origin. Proximity to the origin (the S-compartment) encodes how closely signals received at an input mimic the timing of S-compartment output. It is also important to note that radial and angular training do not measure separation along the dendrite but use geodesic lines in the angular case and straight ones in the radial case.

6.5.6 The Units of Temporal and Spatial Separation

The previous sections discussed how the temporal interval between signals is encoded into a spatial separation within the dendritic field. Here we take a short look at the units of measurement of both as they will appear in experimental tasks in the next two chapters.

Input to an AD neuron is given in Hz, or the average number of signals received within a 1,000 time step window. The use of Hz comes from the Izhikevich neuron (and most spiking neuron models in general) which has a temporal resolution of 1 millisecond. Throughout this work we equate 1 time step with 1 millisecond. Therefore, input sources which use a random process are controlled through a signal generation rate given in Hz.

To translate more easily temporal intervals into spatial ones, and vice versa, both systems should share a common unit. The radius and circumference of the dendritic field is therefore directly convertible into Hz. For example, an R_{max} of 500 can encode temporal relationships between a neural input and the S-compartment down to 2 Hz (or 500 time steps apart).

Consider the following example:

One neural input repeatedly receives signals 100 time steps after one of its siblings. If A_{max} is 500, then, given a perfect spatial encoding, the angular distance between the two

should be $\frac{\pi}{5}$ radians. The denominator of 5 comes from the fact that the temporal relationship of their input signals is one-fifth of A_{max}.

The *perfect encoding* mentioned above is when the angular distance between two neural inputs is not disturbed by relationships with other sibling inputs. Neural input positions can suffer when they have multiple, divergent relationships. Distances can be stretched or compressed. This will be highlighted in several tasks in the next chapter.

6.5.7 Building the Dendritic Tree

The creation of the AD neuron's dendritic tree is broken into two steps: tree-building and compartmentalization. Tree-building is the process of linking the set of neural inputs based on their position within the dendritic field. We use the term *link* in this section to differentiate between a connection as defined by the AD neuron model. Links are not part of the model but they are a necessary step to define the dendritic tree's connectivity. A link is a parent-child relationship between inputs where the flow of signals move from the child to the parent. All inputs are linked to form a tree which is used to compartmentalize the structure.

The dendritic tree is built starting at the origin--the position of the S-compartment--and moving outward through the dendritic field. Inputs are added to the tree one at a time using a modified version of the minimum spanning tree algorithm proposed in [39]. Algorithm 1 shows the steps used to build the tree. Initially, only the S-compartment belongs to the dendritic tree. On each iteration, all inputs in the set of unconnected inputs, $U = \{u_i | i \in I\}$ where $I = \{0, 1, \ldots, n\}$, are compared to the set of connected inputs $C = \{c_j | j \in J\}$ where $J = \{0, 1, \ldots, m\}$. A pair of inputs are chosen, $(u, c) \mid u \in U,\ c \in C$ such that $d(u, c) = \min(d(u_i, c_j))\ \forall i \in I, j \in J$ where d gives the distance between two inputs. d uses both the angular and radial distances between input positions and the current path length to c when searching for the nearest pair. In other

words, a nearest pair, one from each set U and C, is chosen and a link is added between them such that u is linked to c.

Tree building creates a minimum spanning tree with one critical addition. When adding an unconnected input to the tree, the length of the dendritic path from the root to the potential link point is taken into account. The path length is scaled by bf, the branch factor. bf controls both the length of each branch when measured from the origin and how frequently existing dendrites bifurcate into child branches.

Algorithm 1 Building the dendritic tree.

Require: Disconnect all inputs
  $bf \leftarrow$ branch factor
  $C \leftarrow$ Set of all connected inputs
  $U \leftarrow$ Set of all unconnected inputs
  $s \leftarrow$ The dendritic neuron's S-compartment
  $C \leftarrow C \bigcup \{s\}$
  $U \leftarrow U - \{s\}$
  while $U \neq \varnothing$ **do**
    $\dot{c} \leftarrow$ any element in C
    $\dot{u} \leftarrow$ any element in U
    $\dot{c}_{bf} \leftarrow bf *$ path length from s to $\dot{c}$
    $\dot{d} \leftarrow \dot{c}_{bf} +$ distance between $\dot{c}$ and $\dot{u}$
    for all $c \in C$ **do**
      $c_{bf} \leftarrow bf *$ path length from s to c
      for all $u \in U$ **do**
        $d \leftarrow c_{bf} +$ distance between c and u
        if $d < \dot{d}$ **then**
          $\dot{d} \leftarrow d$
          $\dot{c} \leftarrow c$
          $\dot{u} \leftarrow u$
        end if
      end for
    end for
    Link $\dot{u}$ *to* $\dot{c}$
    $C \leftarrow C \bigcup \{\dot{u}\}$
    $U \leftarrow U - \{\dot{u}\}$
  end while

The plots in Figure 6.4 show the effects of the tree building algorithm 1 for ninety-nine inputs (blue) and the S-compartment (orange) using different values for bf. Neural input

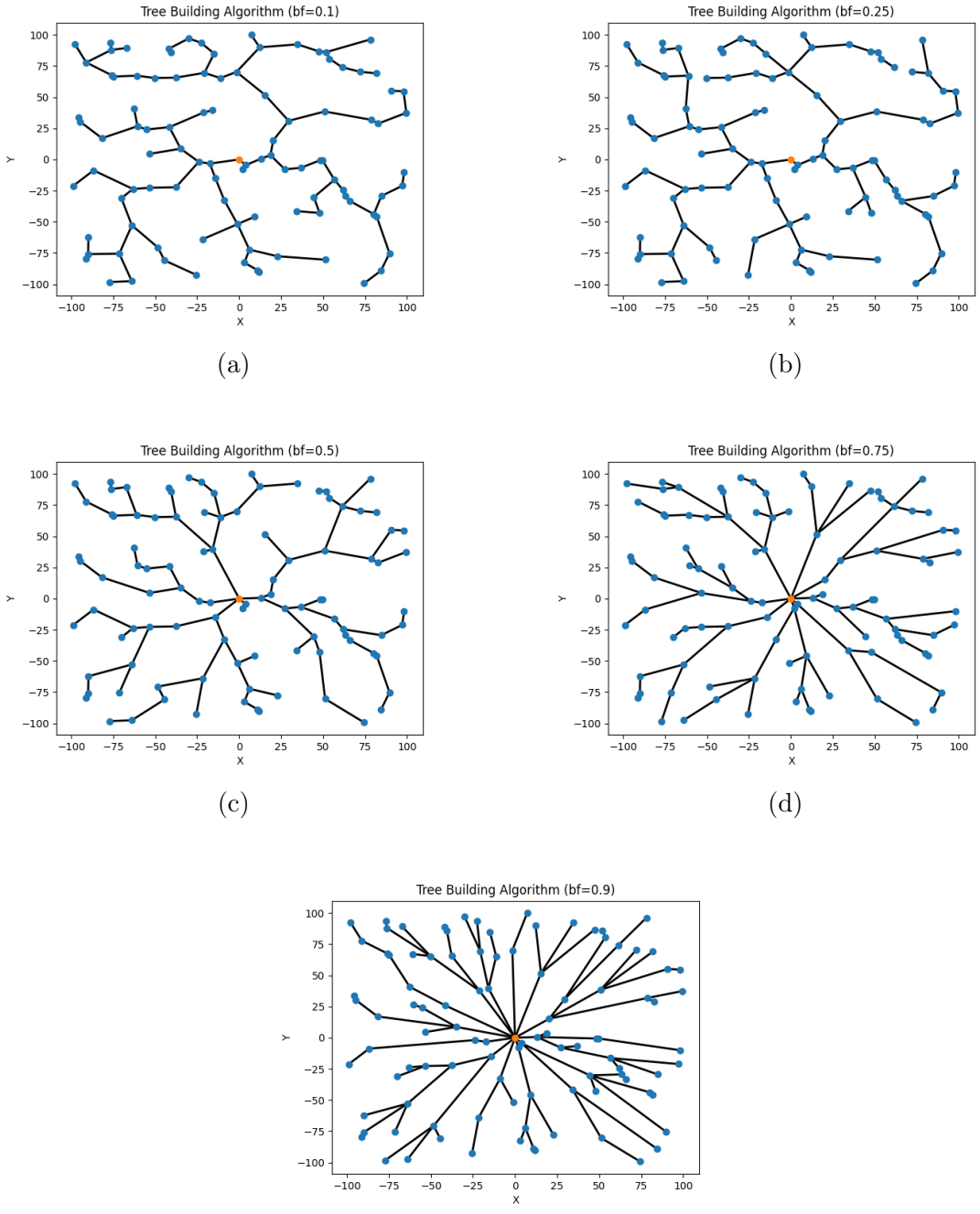
positions are identical for all figures. A lower branching factor produces fewer dendrites rooted at the S-compartment. For bf=0.1 (6.4a) and bf=0.25 (6.4b), there are only two links to the S-compartment. For bf=0.5, (6.4c), there are five. bf=0.75 (6.4d) produces eight S-compartment links. And bf=0.9 (6.4e) produces fourteen. At the extremes (not shown), bf=0.0 can produce as few as one S-compartment link and bf=1.0 will almost always produce as many S-compartment links are there are inputs.

As bf increases, the tree changes shape. Between $bf = 0.1$ and $bf = 0.25$, only a few inputs in the top left and right of the plot alter their linkage. Increasing bf to 0.5 has a drastic effect. It has few sub-trees in common with $bf = 0.25$. As bf approaches 1, there is little lateral movement in the trajectory of subsequent, deeper links.

bf has two major effects on the shape of the tree. First, bf determines the average length of dendritic paths. A low bf produces very long paths with many links. Second, bf limits the angular range within which subsequent links will be made. When $bf = 0.9$, subsequent links are made within a small angular offset from the previous link. In other words, a high bf favors straight dendritic paths. The combined impact of these two effects determines how far inputs can be from one another and still share the same dendritic path. Looking at $bf = 0.1$, we can see that the two S-compartment-rooted trees fan out each covering half of the dendritic field. This is the same for $bf = 0.25$. Using spherical input positions, a low bf can link inputs located on the same shell (close input-output relationships). On the other hand, a high bf favors links between inputs whose angular distance is small (close input-input relationships).

Tree-building guarantees that all neural inputs to an AD neuron are linked to form an acyclic network or tree. Once all inputs are linked, compartmentalization begins.[9]

[9]In our implementation they work in tandem. As new inputs are linked, they are either added to an existing compartment or define the start of a new one.



(a) (b) (c) (d) (e)

Figure 6.4: An example of the tree building algorithm 1. Each figure uses the same set of input positions with a different branching factor (bf) value. (a) $bf = 0.1$ (b) $bf = 0.25$ (c) $bf = 0.5$ (d) $bf = 0.75$ (e) $bf = 0.9$. The blue dots are the positions of neural inputs. The orange dot is the origin (or eventual S-compartment position). Black lines denote links between neural inputs. Note: the input positions in this example use x,y-coordinates rather than 3D spherical ones to simplify the visualization.

6.5.8 Compartmentalization

Compartmentalization uses the acyclic network created by the tree-building algorithm to assign neural inputs to a set of compartments. The number and shape of compartments depends on many factors, such as the maximum compartment length, the position of neural inputs and in which order inputs are linked. Compartmentalization uses both position and connectivity. Algorithm 2 defines the process of compartmentalization.

Compartmentalization begins at the origin of the dendritic field. At the origin the neuron's S-compartment is created. Following the links of the tree, inputs are added to the current compartment until the dendritic path linking them back to the S-compartment exceeds the maximum compartment length. A *compartment root* is the first input added to an empty compartment. The maximum length of a compartment is given by L_{max} in Table 6.3. Distances are measured using the links of the dendrite (the network distance). If the next input selected to add to a compartment would extend the compartment beyond L_{max}, a new compartment is generated with length zero and the next input becomes its root.

Once all inputs have been assigned to a compartment, internal connections (as defined by the AD neuron model) are created between compartments. Links between inputs that span compartments are upgraded to fully-fledged connections. The distance between these inputs becomes v_e for this new connection. For each input within a compartment, its network distance to the compartment root is calculated. This value is used to create v_e for the incoming external connection of which the neural input is the terminal end.

This process of replacing links with connections is depicted in Figure 6.5.

It may not be obvious at first glance why this two step compartmentalization makes sense. For the middle compartment, the external connection from D arrives at the compartment root. During the second stage, v_e for the D external connection is assigned 0 because there is no separation between its position and the root. It *is* the root. Compare this to the external connection B which gets a v_e equal to 18. The separation between B's

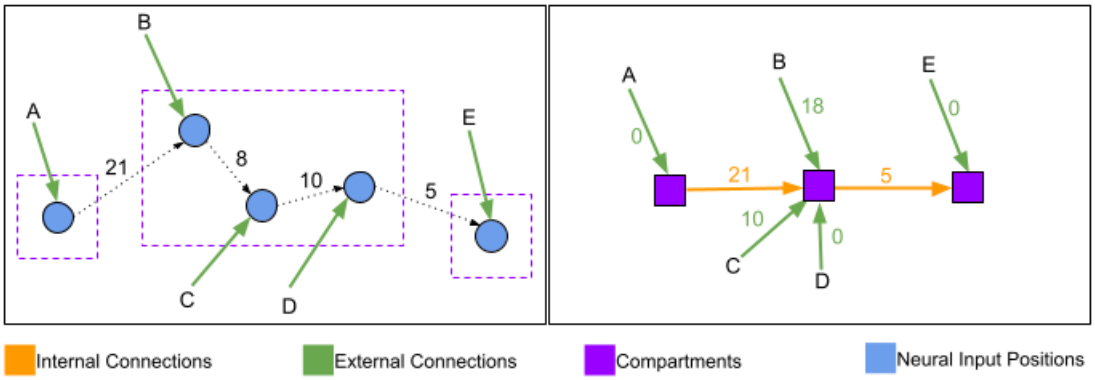


Figure 6.5: A two-step depiction of building compartments from a segment of a dendritic tree. In the left pane, neural inputs (blue) are compartmentalized using an L_{max} of 20. This creates three compartments (purple). Labeled external connections are in green. Dotted black lines show the links created during tree-building. In the right pane, inputs are collapsed into single entity compartments. Links provide distance information for the connections, both internal (orange) and external (green).

point of arrival and the compartment root is 18, given by a link of 8 and one of 10. The internal connections, on the other hand, simply replace links.

The reason external connections inherit measures of separation which appear to be internal to the AD neuron goes back to the discussion around Figure 6.1. External connections do not carry a measure of inter-neural separation. The compartmentalization algorithm must only maintain the distance between its point of arrival in the dendritic field and the root compartment. Again, this design choice was made because we are only focused on single neurons. The alternate would simply add the inter- and intra-neural distances to get the v_e of an external connection.

Operationally, the AD neuron in its final form consists of connections and compartments.

There are several quirks to this algorithm.

First, the degree of branching in a dendritic tree can increase or decrease the size (number of inputs) of compartments. Compartments are created from the root (or a sub-root for subsequent compartments) using the tree's paths. Compartment length is synonymous with tree depth. Since compartment length is measured from the compartment

Algorithm 2 Compartment creation.

Require: No disconnected components in the dendritic field
  $S \leftarrow$ a stack of inputs
  $o \leftarrow$ S-compartment
  $c_o \leftarrow$ new compartment ID
  $K \leftarrow$ set of inputs linked to o
  for all $k \in K$ **do**
    $d_k \leftarrow 0$
    $c_k \leftarrow c_o$
    Push k onto S
  end for
  $d_{max} \leftarrow$ maximum compartment length
  while $S \neq \varnothing$ **do**
    Pop i from S
    $J \leftarrow$ all children of i
    for all $j \in J$ **do**
      $d_j \leftarrow d_i +$ distance from i to j
      if $d_j \geq d_{max}$ **then**
        $c_j \leftarrow$ new compartment ID
        $d_j = 0$
      else
        $c_j \leftarrow c_i$
      end if
      Push j onto S
    end for
  end while

root, branching increases the number of dendritic paths under consideration, potentially decreasing the length from the root to a candidate for inclusion in the compartment.

Second, Algorithm 2 does not isolate the S-compartment but allows neural inputs to be included in it. Therefore, maximum compartment length also defines the distance at which dendrites begin because all inputs whose network distance is less than L_{max} are included in the S-compartment (the soma). If all inputs fall inside this distance, the AD neuron is adendritic (a point neuron). An alternate version of the algorithm alters a single line, changing $c_k \leftarrow c_o$ to $c_k \leftarrow$ new compartment ID. This alternate version forces all inputs to occupy dendrites; the S-compartment contains no neuron inputs (i.e., external connections) but only receives signals from compartments which belong to the AD neuron.

This work uses the alternate version exclusively because it simplifies our implementation and focuses only on signals arriving on dendrites. Otherwise, results have not led us to favor one version over the other. We have experimented with both versions and in the experiments both versions produced qualitatively similar results.

6.5.9 Weight Modification

Angular position encodes input-input relationships. Radial position encodes input-output relationships. The dendritic tree encodes an ordering of the inputs drawn from their positions. Compartmentalization encodes sub-patterns based on clustering caused by the aforementioned encodings.

So, which aspect of neural activity would be captured by weight modification that is not captured by its other properties?

Chapter 4 stated that a connection's weight gives its importance at the compartmental level. Therefore, weight training encodes a connection's contribution to the compartmental response. When a compartment emits a spike, all of its connections are evaluated for weight modification. Connection weight is modified using a method similar to the angular and radial training components. The compartment spike time is compared to the most recent spike received through each of its connections. The closer these two times align the greater the change in weight.

Figure 6.6 shows an example compartment and how the timing of compartment-connection spikes create $\hat{w}$. The top half of the figure shows a diagram of the compartment. Link information is shown to give a spatial dimension to the visual. Compartment 1 contains four connections: A, B, C, and D. D, in green, is the root of the compartment. A compartment root is the final connection, with respect to the forward flow of information, of the compartment. The numbers in black above each link in the dendritic tree indicate distances between connections. For example, the distance between A and D is 15. Five signal times are given for the compartment and the four connections. Connection

signals times are S_A, S_B, S_C and S_D. The compartment's signal time (S_1) is the spike time of the compartment.

The bottom half of figure 6.6 plots the possible values for $\hat{w}$ for each connection (black line). For example, connection D's distance to the compartment root is 0 (because it is the root); therefore, the max $\hat{w}$ for D is when $S_D = S_1$. In this case, the difference in timing of their signals is 5 (80-75) so the change in D's weight is minimal. C, on the other hand, received a signal 20 time steps before S_1. Since the distance between C and the root is 20, $\hat{w}$ is maximal (red line cuts through the peak). Note, the $\hat{w}$ plots extend to time 90 to show the full rise and fall of D's calculation for $\hat{w}$.

Compartmental-connection timing takes the dendritic tree into account. In other words, when the temporal difference between compartmental-connection signals matches the length of the dendritic path between them, weight change is maximal.

$$\hat{w} = \frac{\epsilon}{(((t_c - t_i) - d_i)b_w)^2 + 1} \tag{6.22}$$

Equation 6.22 gives the change in weight due to a compartmental signal. When a compartment emits a signal all connections $i \in C$, the compartment, are modified using Equation 6.22. t_c is the time of compartmental signal. t_i is the time of the most recent signal received by connection i. d_i is the distance as measured by the dendritic path from connection i to the compartment root. b_w is a shape parameter similar to the b used elsewhere[10] . And ϵ is the error value. $\hat{w}$ is then applied to w_c the connection strength.

Input weight consists of two parameters: the maximum weight, w_m and the current weight, w_c. Together these combine to form the actual weight, w_a. Equation 4.3 in chapter 4 showed how w_a is calculated.

This equation was chosen because it avoids the necessity of artificially capping the growth of w_c. As w_c gets very large W approaches the maximum weight, w_m (or $-w_m$ for

[10]In all experiments $b_w = 1$. It could be omitted from this equation for the purpose of this thesis but we include it to show that, like connections, a shape parameter can be used to alter how precise compartment-connection timing must be to modify connection weights.

extreme negative values). This formulation has the added benefit of representing the longevity of an association. When w_c is very large (or small), changes in its value have a small impact on W. This makes an established weight more robust to outliers or other types of irregular input.

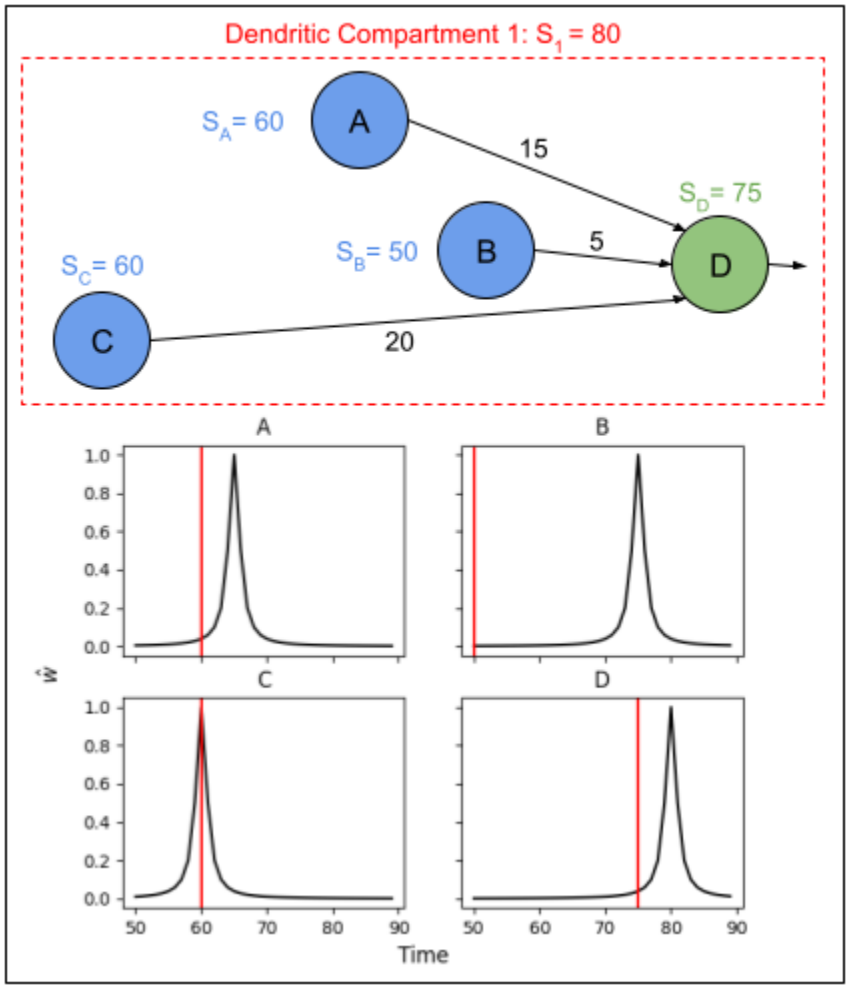


Figure 6.6: (Top) An example compartment with four connections and weight modifications due to a compartment signal. Red box (dashed outline) delineates the dendritic compartment. Circles indicate labeled connections which belong to the compartment. Blue connections belong to the compartment. The green connection is the root connection of the compartment. Numbers (S_x) above each connection and compartment indicate the most spike time. Black arrows indicate links and the flow of information. Black numbers give distances. (Bottom) Weight changes due to a compartment signal at time 80 ($S_1 = 80$). The intersection of red vertical lines and the black plots indicate the value of $\hat{w}$ for each connection. The error value in this plot is 1 for demonstration purposes.

6.6 Miscellaneous Training Aspects

6.6.1 Random Mask

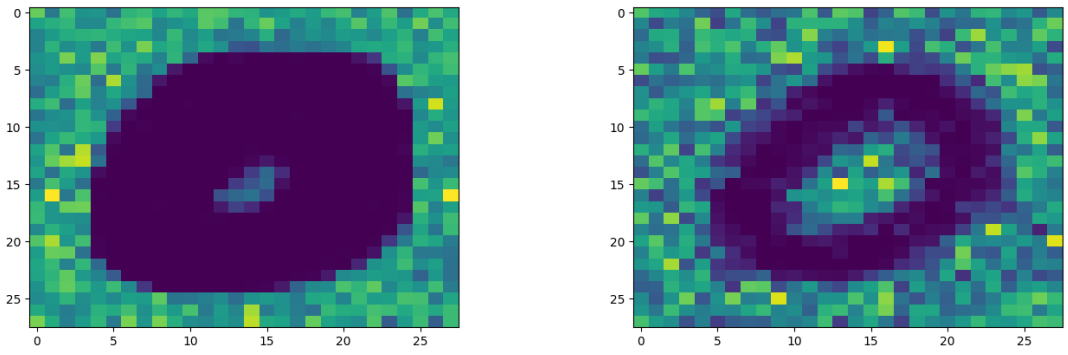
A random mask was used by several experiments to increase the angular separation between unrelated inputs and/or patterns. Unrelated inputs (or patterns) are those whose received signals do not coincide within the learning window. Due to random placement during initialization, unrelated neural inputs can cluster together within the dendritic field. This causes them to be grouped together in the same compartment which can impact performance.

To solve this problem, we added the random mask. The random mask was inspired by visual masks used in neuroscience experiments. A visual mask is typically a neutral example (e.g., white noise or random patterns) shown to the subject in order to disrupt neural activity caused by a previous example.

A random mask consisted of randomly generated spike trains of low activity ($\leq$ 10 Hz) across all input sources. Spike trains were created using a Poisson process. Whenever the mask was presented to the AD neuron, the global error variable was set to a negative value (typically -1) causing coinciding input to drive neural inputs apart. The random mask phase of training took place at the end of an epoch, after training and testing.

The random mask and negative error created a background repulsive force across the dendritic field. This repulsive force was low enough such that genuine attraction between inputs caused by actual data could easily overcome it.

We found that the random mask can be used in lieu of negative feedback in tasks which contain multiple types belonging to a single category. For example, suppose we desired to train a single AD neuron to respond to the letters 'A' and 'B'. Since the two types share some features, without any repulsive force, the active inputs encoding both types would likely cluster into an indistinguishable mass. Using a mask, the cluster of inputs corresponding to non-overlapping features of A will move away from the clusters of non-overlapping features of B. And both non-overlapping clusters will separate from the



(a) (b)

Figure 6.7: Example of the effects of using a random mask. Both images display the sum of angular distances from each neural input to all sibling inputs. Darker colors indicate shorter total distances. Both AD neurons were trained on all images of the digit 0 from the MNIST handwritten digit data set using (a) a mask strength of -1 and (b) of -32. Mask rate was 10 Hz in both.

cluster of overlapping inputs. Cluster-level separation increases the probability that clusters will occupy separate dendrites and/or compartments.

The strength of the mask, i.e., the associate error value, can also sharpen pattern boundaries. A pattern boundary is a consistent difference in activity between two adjacent features with respect to the original data (e.g., two adjacent pixels in an image). The angular distance between inputs which fall on such boundaries increases with the strength of the mask. Very strong mask values tend to exclude all but the most co-active inputs from pattern clusters. On the other hand, a weak mask is overcome by any degree of co-activity between inputs.

Figure 6.7 shows the general effects of the mask strength. Neural inputs are arranged by their source's placement within the image. Darker-colored inputs are closer angularly to all other inputs. The stronger mask, 6.7b, creates a smaller pattern cluster in the number of inputs compared to 6.7a.

Tuning the impact of the random mask was performed by hand. The effect of the mask can be tuned using either the strength of the repulsive force or the spike rate of the random trains. We favored a lower strength and higher spike rate. A higher rate causes more

co-activity between all inputs and a lower strength reduces the variance between input displacements due to a single mask. Such a mask was preferred because it has a uniform impact. However, higher rates do increase training time due to the number of spikes involved.

6.7 Comments on Training

6.7.1 Complexity

The amortized complexity of both input position algorithms--angular and radial--are $\mathcal{O}(n)$ with n being the number of inputs to the AD neuron. How often they contribute to the run-time of the AD neuron's overall update algorithm depends on several factors, such as incoming signal rate and the S-compartment output rate.[11]

The tree building algorithm is $\mathcal{O}(n^3)$. On paper, this is the most computationally expensive aspect of AD training. However, experience tells us that learning does not benefit from rebuilding the tree after every example so its impact can be reduced. Rebuilding after every one hundredth or one thousandth training example (every epoch) seems adequate. More testing is required to know the true of impact of delayed rebuilding.

Compartment creation, in isolation, is $\mathcal{O}(n)$ and like tree building does not need to happen after each example. In fact, since regenerating compartments only makes sense once a new tree has been built, which takes into account new input positions, compartment creation can be woven into the tree building algorithm. Therefore, in practice, it does not add an additional linear time before the beginning of an epoch but simply a constant amount of work to the outer most n of the tree building algorithm.

Weight training is harder to categorize since it is highly dependent on the shape of the tree and how widely active the neuron's set of inputs are. Only active compartments modify connection weights. If only a subset of a neuron's inputs are active and they alone

[11]Since our implementation uses Izhikevich spiking neurons, run-time also depends on the neuron type; some types produce more spikes given the same input.

belong to a single compartment, then the run-time of the weight training algorithm is linear in the size of the active subset. In the worst case, it is $\mathcal{O}(n)$ in the size of all inputs to the AD neuron. This occurs if all inputs belong to one compartment.

Overall, an inactive AD neuron's training regime is $\mathcal{O}(1)$ as nothing moves without activity and if nothing has moved, there is no need to rebuild the tree or regenerate compartments. At its worst, it is $\mathcal{O}(t(n^2 + 2n) + n^3)$ for one example if every input received a signal every time step, the S-compartment emitted a signal every time step and the tree and compartments are regenerated after every example. t is the number of time steps devoted to a single input example.

6.7.2 Approximating a Centroid on the Surface of a Sphere

Each signal that arrives in the dendritic field, the receiving neural input seeks a new position. It is pulled toward each active sibling input based on the recency of the sibling's last received signal. The optimal new position of the receiving input is the centroid of these attractions. In mathematics, this is known as the Fréchet mean on a 2-sphere.

Finding the Fréchet mean is nontrivial [4, 9, 47]. Since it is a closed surface, there can be more than one mean point which minimize the geodesic distance to a set of points.[12] Current methods for approximating the means are computationally expensive, consisting of heuristic methods such as branch and bound, Monte Carlo and gradient descent.

In the AD neuron, the Fréchet means must be approximated for each incoming signal. It does this by moving an input along the geodesics between the input and all other active inputs. Together these moves constitute a single attempt to find the mean. However, we are not as interested in a perfect approximation for a single signal but in how well the approximation converges on the true mean through time and repeated activity. Therefore, it is enough if the set of moves produced by each signal position an input closer to a bounded space around the actual mean. Once the input enters this bounded space, it does

[12] A closed surface is compact and has no boundary; therefore, a great circle which passes through two points defines two geodesic distances between them. For antipodal points, there are infinitely many means.

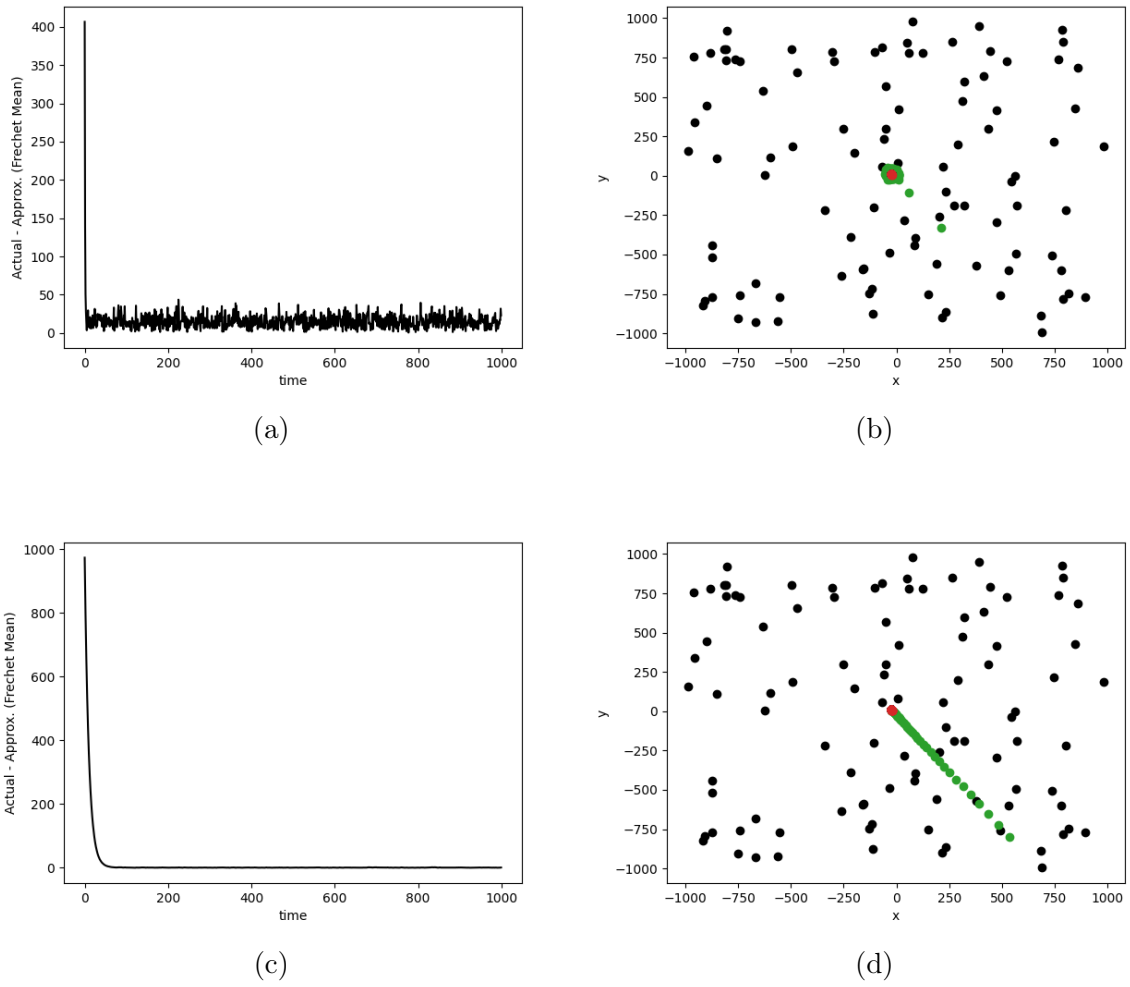


Figure 6.8: Simulation of the angular training algorithm for 1 input and 100 static sibling inputs using Cartesian coordinates. Top plots use a learning rate, $\lambda = 0.01$. Bottom plots, $\lambda = 0.001$. (a) and (c) depict the distance between the approximated mean and the actual mean over the entire run. (b) and (d) show the sibling inputs (black dots), the mean (red) and the path of the approximated mean (green).

not exit it (unless the mean itself moves). Let us define this bounded space as the set of points on the surface of the sphere which are within ϵ of the Fréchet mean.

To discover how well angular training finds and limits this bounded space around the mean, we conducted a reduced simulation. Figures 6.8 depict a simulation of the angular training algorithm for a single input in its search for the mean of activity generated by 100 sibling inputs. In this simulation, the sibling points do not move. Since these examples are just for demonstration of the issue, we used Cartesian rather than spherical coordinates.

The top plots use a learning rate (λ) of 0.01 whereas the bottom plots show the simulation with a learning rate of 0.001. Otherwise, the simulations are identical. Figures 6.8a and 6.8c give the distance between the approximated mean and the actual mean across 1,000 time steps.

The smaller learning rate resulted in a tighter bounded region around the actual mean ($\epsilon \approx 2$ for $\lambda = 0.001$ and $\epsilon \approx 50$ for $\lambda = 0.01$) but a slower convergence (2 ts for $\lambda = 0.01$ and 70 ts for $\lambda = 0.001$). The right side plots show the input's path (green) toward the mean (red).

This reduced simulation shows that the angular training algorithm does approximate the mean given enough time and activity. Angular training is itself a heuristic to approximate the Fréchet means, not for a set of static points but for patterns of activity arriving at a set of points. For a set of input examples, the mean with respect to a single input's position in the dendritic field is not fixed. Input positions change during training therefore the mean changes. With enough training epochs, an input's learned position converges toward the mean of means.

In this reduced simulation, activity is kept to consistent levels and activity at each input varies predictably. If activity varies drastically and the actual mean moves significantly with each example, the smaller learning rate performs less well. The approximated mean (i.e., the position of the input in question) cannot make large enough shifts. For data with a large variance between examples, a larger learning rate allows the approximated mean to keep up with the actual. For such data, the error rates (like those shown in Figures 6.8a and 6.8c) have a wider range.

However, the larger learning rate performs slightly better. It is unclear if this is desirable. Even if the training algorithm matches the actual mean by making large jumps in its approximation, this causes an instability in the dendritic tree. Since connectivity and compartmentalization depend on input position, large changes in input position drastically alter the shape of the neuron.

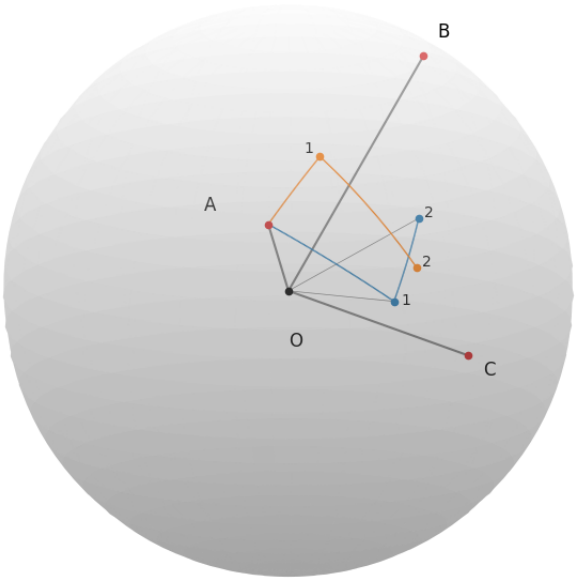


Figure 6.9: Demonstration of the differences caused by altering update order. Red dots (A, B, and C) are input positions. Black dot (O) is the origin. Orange and blue dots and lines show changes to $\hat{p}_i$'s position using two different update orders. Blue: C-B. Orange B-C. Grey lines are a visual aid to help with distance and orientation of the inputs from the origin.

Its approximation of the mean is also dependent on the order of evaluation with respect to sibling positions. Different orderings of the sibling inputs will produce different final positions of $\hat{p}_i$. This is demonstrated in Figure 6.9.

In Figure 6.9 a signal arrives at A. Both B and C have also received signals within the learning window. The orange and blue lines depict the path of $\hat{p}_i$ as it is moved toward both B and C using different orders. For the orange case, $\hat{p}_i$ is moved first toward B and then C. For the blue case, the opposite is true: C and then B. Only the order was altered yet their final positions (positions 2) are different.

6.8 Summary

This chapter described a suite of algorithms by which AD neurons can be trained. Training consists of positioning inputs, connecting them, grouping them into compartments and altering the connection weights. Input position is determined by co-activity. A modified minimum spanning tree algorithm uses the trained input positions to build the

tree. Compartments are grouped using a network distance metric. Connection weights are adjusted based on a comparison between input and compartmental output times. AD training is unique because it creates a network shape informed by the behavior of its input and output.

Dynamic network topology is atypical for ANNs. Connections are usually fixed. Input location is preset. The sets of connections which come together linearly are pre-configured. Training moves through a single mechanism, the connection weight. This simplifies the training process. A neural network models layers and layers of functions composed of other functions. Rearranging the network recomposes these functions potentially disrupting attempts to find a solution through standard learning methods like gradient descent or evolutionary search. Altering the shape of the network during training potentially disrupts many of the high-performance optimizations on which state-of-the-art ANNs depend (e.g. fast matrix operations). This approach stands in contrast to existing AI work on artificial dendrites which favors predefined dendrites or sub-networks so as not to completely wreck access to off-the-shelf, high performance hardware.

But stepping back further, AD neuron training is not generating the entire network. The dynamism of the AD neuron is an internal matter. From a distance, the ADNN (Artificial Dendritic Neural Network) looks like an ANN. Connections between two AD neurons, between the S-compartment of one and the dendritic field of another, are not in question. The AD neuron only concerns itself with *where* in the field it connects. This suggests that the AD neuron training algorithm is amenable to current optimizations.[13]

The preceding approach depends on the assumption that clustering based on co-activity is computationally beneficial. This assumption is undergirded by neuroscience research: fire together, wire together[143]. However, this assumption is not a universal dictum for all neural circuitry. It is likely a variety of rules will be necessary to make the AD neuron

[13]To accommodate the mutating tree, perhaps all that is needed is a neuron-level lookup table. This is precisely how connections are handled in this work's implementation of the training algorithm; however, it was not done in the context of a larger network of AD neurons or pushed onto specialized hardware, e.g. GPU.

more of a general tool. With this in mind, this approach to training was designed to be modifiable to allow for different rules and training behaviors. Our goals are simple: provide an extensible starting point. Throughout we have attempted to provide a general approach to training which supports modification.

The preceding approach does have several issues which are highlighted in the following chapters. The following chapter delves into a series of experiments which helped to refine and test the individual algorithms of AD neuron training. The final discussion chapter walks through many of the issues encountered during this work and the ways in which it can be extended.

CHAPTER 7

TRAINING CALIBRATION

This chapter describes how we calibrated the AD neuron's training algorithm. We train it on a collection of tasks to gain a better understanding of the algorithm in order to determine an appropriate set of hyperparameters (Table 6.3). In the next chapter, we test the AD neuron model on several categorization models to examine its performance on more complex tasks.

Training an AD neuron has five parts, corresponding to angular positioning, radial positioning, dendritic tree-building, compartmentalization, and weight modification. Since each of theses has a degree of independence from the others, we trained each in isolation as well as together with the others. This allowed us to understand how each works by itself and as part of a team.

7.1 Angular Position Training

We first trained angular position alone to see if it would produce stable and explainable patterns in the dendritic field. The goal of these tasks is to see if the angular training algorithm correctly encodes the temporal relationships between average spike times received by neural inputs into matching positions within the dendritic field. For example, if one input repeatedly received a spike 50 time steps after another, the angular distance between them should encode this temporal relationship (see Section 6.5.6). We will also show that angular distances can be warped by multiple, complex relationships.

If the post-training separation between input positions is not representative of patterns within the set of input examples, subsequent steps of the training process could be in vain. Compartmentalization and tree-building are designed to take advantage of a meaningful spatial encoding of inputs within the dendritic field.

To answer this question, we presented an AD neuron with simple input patterns that should be easily recognizable in post-training input positions. The AD neuron was initialized with a small set of afferent connections. Each connection received input in the form of signals (i.e., spikes) at a specific rate from an input source using a Poisson process. The number of connections and spike rates vary between tasks. All training processes other than angular positioning were disabled. For all tasks AD neurons were initialized using the initialization procedure from 6.5.2.

Specific input rates were selected because they produced recognizable differences in angular (and later radial) positions. Generally, 100 Hz was used as an upper bound on rates because rates above 100 Hz produce tight clusters in input positions which impact visualization when displayed alongside clusters produced by lower rates. Input rates also affect other AD neuron parameters, such as the dimensions of the dendritic field, therefore, rates are generally kept within 20 to 100 Hz.

7.1.1 Task 1

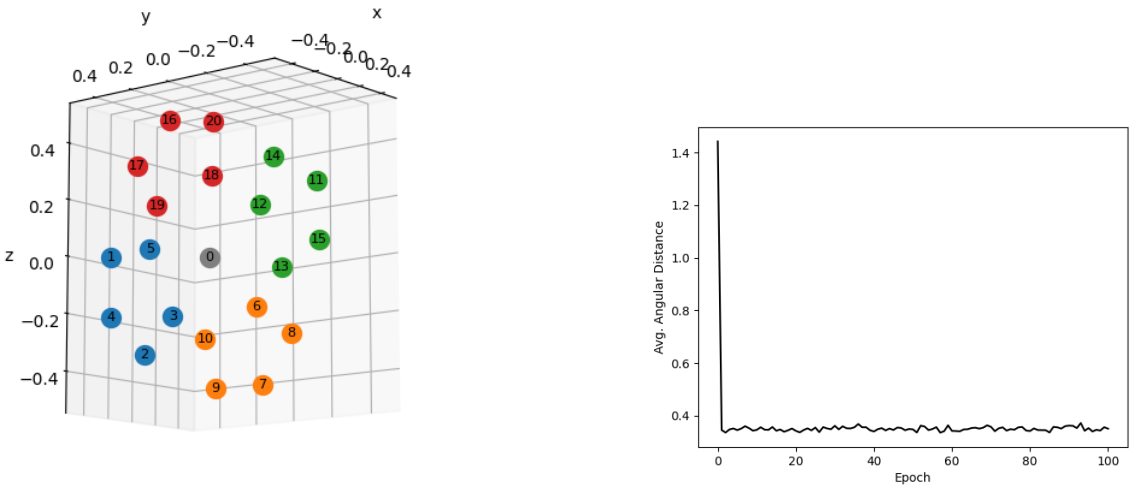
Task 1 consisted of 21 input sources which were grouped into four patterns. Patterns are shown in Table 7.1. Each pattern activated 5 unique sources and all patterns activated, or shared, one source: 0. When a pattern was presented to the network, sources that belonged to it produced signals at a rate of 100 Hz. Sources that did not belong to it were silent (0 Hz).

A 21-input neuron was initialized using the method described in 6.5.2 with parameters from Table 6.3 in general, although A_{max} was set to 100 to make input distance calculations more intuitive.

An AD neuron was instantiated with 21 neural inputs receiving signals from the 21 sources. AD neuron parameters were drawn from those given in Table 6.3 with one exception. Each input was connected to a source of spikes. The rate of spikes was the same for all sources for each iteration of this learning task, e.g., 100 Hz, but was changed

Pattern 0	0	1	2	3	4	5
Pattern 1	0	6	7	8	9	10
Pattern 2	0	11	12	13	14	15
Pattern 3	0	16	17	18	19	20

Table 7.1: The four patterns, color coded, of Task 1. Gray designates shared connections.



(a) (b)

Figure 7.1: Task 1: Four patterns, 6 sources per pattern, 1 source shared by all patterns. Pattern colors correspond to those given in Table 7.1. The gray-colored neural input (0) is shared by all patterns. (a) Angular positions after 100 epochs. (b) The average angular distance within patterns over 100 epochs. Distances are angular measured with respect to the center of a unit sphere.

between iterations to test how input rates form different spatial relationships between input locations in the dendritic field. The timing of the spikes was controlled by a Poisson process for each input source.

Different patterns to learn were created by activating different sets of input sources. The different patterns are shown in Table 7.1. Each iteration of the learning task consisted of 100 epochs of 100 examples drawn randomly from the four patterns. We made no attempt to ensure the patterns would be equally represented in each epoch or overall.

The AD neuron was exposed to each example for 1,000 time steps during which the angular training algorithm was active. A random mask was used with a rate of 10 Hz, a weight of -1 and an exposure time of 10,000 time steps per epoch.

In the base variant of this task (Figure 7.1), all input sources spiked at the same rate (100 Hz.) creating an equal positive relationship between all inputs within the same pattern. The task was run for 10, 100 and 1000 epochs. Only the 100 epoch run is shown. Input positions clustered corresponding to input patterns. Pattern clusters encircled the shared neural input, 0.

A shared input was included to test the angular training algorithm's ability to form meta-patterns (or, patterns composed of patterns).

Inputs within each pattern form rough circles because each input receives signals at the same rate. A slight compression of the cluster is visible. This is not an artefact of the spherical field but due to the peripherally positioned inputs being pulled inward by both centrally located inputs and input 0.

Relative input positions do not visibly improve beyond the first few epochs. This is due to the relatively strong attraction between inputs (e.g., 100 Hz spike rate) in relation to both the size of the dendritic field (i.e, A_{max}) and the strength of the random mask (10 Hz). If the spike rate or learning rate is reduced, clustering takes longer.

The order of pattern clusters around the shared input varies each time the task is run. In the run shown in Figure 7.1a, starting with pattern 0 and working clockwise, the order is 0 (blue), 3 (red), 2 (green), 1 (orange). Once pattern clusters coalesce their order does not change over subsequent epochs. The countering forces of attraction (pattern-based activity) and repulsion (mask activity) are balanced thus keeping patterns in place. It is possible to force a possible reordering of the pattern clusters by exposing the AD neuron to a single instance of a strong or unequal mask--scattering the positions--however, this is not guaranteed to cause a reordering of the clusters or neural input positions within a cluster. Our method for finding a clustering-scattering balance was to vary the mask strength or duration while leaving inputs fixed.

Figure 7.1b shows that average angular distance falls to approximately 0.36 radians. The standard deviation of the average angular distance after the second epoch is 0.007.

The SD shows that the spatial encoding is stable. Using the fact that the maximum possible angular distance between two positions is π, we can calculate how well an average angular distance of 0.36 encodes the relationship between spike rates: $\frac{0.36}{\pi} \approx 0.11$. Given that $A_{max} = 100$, 0.11 translates to an average time between spikes of 11 time steps. This is close to what we would expect with a rate of 100 Hz. A perfect encoding would be $\frac{10\pi}{100} \approx 0.31$. With 1,000 epochs, the average input distance within patterns improves to 0.34. If input 0 is omitted from the distance calculations, the average distance is 0.32.

What these distances encode is not the spike rate of one input source but the average interval between input signals. Neuroscience refers to this as the *interspike interval* (ISI). When all sources spike at the rate of 100 Hz, which is well below A_{max}, the maximum ISI capable of affecting input positions, the average ISI of all sources contributing to a single pattern approaches 10 milliseconds. However, when rates vary and some of them are near or slower than A_{max}, calculating the average ISI of a pattern or a subset of its sources, is more difficult. To estimate complex ISIs (and the simple one above) we used the Python3 script given in B.1. All target rates are found using this script.[1]

What effect does the random mask have on neural input positions? Figure 7.2 shows input positions after 1,000 training epochs without a random mask. The inputs belonging to each pattern cluster in a circular arrangement similar to the baseline example. However, without a mask, the clusters overlap. Without the mask, clusters are guaranteed to overlap due to the shared input source 0.

Removing the random mask, after 100 epochs, the average angular distance falls to 0.326 or an ISI of 10.3. The mask, then, disrupts the spatial encoding of input positions. For this task, it translates to a 10 Hz disruption, which, again, matches the 10 Hz spike rate of the mask. In theory, one could, once input positions are established, readjust

[1]The Python3 script B.1 actually gives an average ISI slightly under the expected value. For N input sources with a spike rate of 100 Hz, as the length of the simulation goes to infinity, the average ISI goes to 9.999. We are unsure whether this approach from the smaller side to the expected value is a problem with the script or the true result.

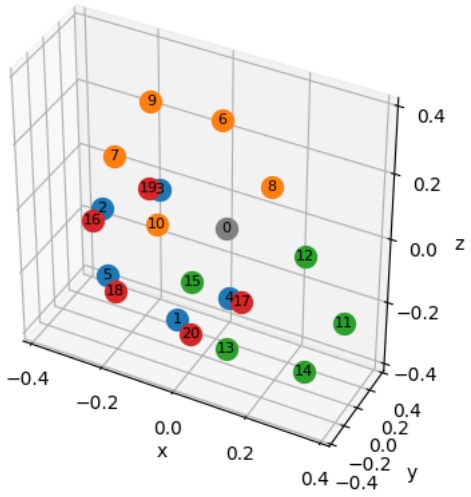


Figure 7.2: Task 1 without a random mask. Training lasted for 100 epochs.

positions to remove the mask's effect. We suppose that the vector of such corrections would be toward the cluster's centroid. [2]

Zooming in on each pattern, non-shared inputs form a rough circular shape. It is impossible for more than three points on a 2-dimensional surface to be equidistant; therefore, some input positions must be closer than others. With each signal, inputs jostle for position, and, due to initial positions, certain inputs belonging to the same pattern become immediate neighbors within the dendritic field. In pattern 3 (red) in figure 7.1a, for example, the pairs of connections (16,20), (20,18), (18,19), (19,17), and (17,16) are closer than any other pairings of inputs of the pattern. In fact, the average angular distance of similar pairings across all patterns is 0.29 which is less than the perfect encoding given above.

Pattern size also has an impact on spatial encodings. Figure 7.3 depicts the results of a tangent task which examined spatial encoding in situations where many input sources with correlated spike rates (100 Hz), i.e., belong to the same pattern.

In this task, the input data set consists of one pattern which includes all sources. Each input source spikes at a rate of 100 Hz. No mask was used to mimic the mask-less variant

[2]We do not make this correction in any of our tasks. Due to the immaturity of the training algorithm we are uncertain whether such small, post-training adjustments will have a meaningful impact.

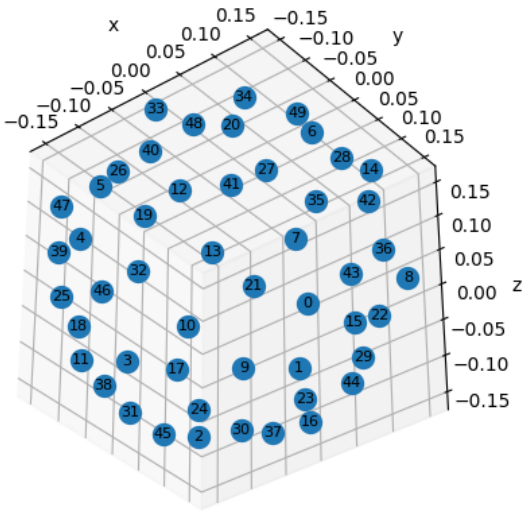
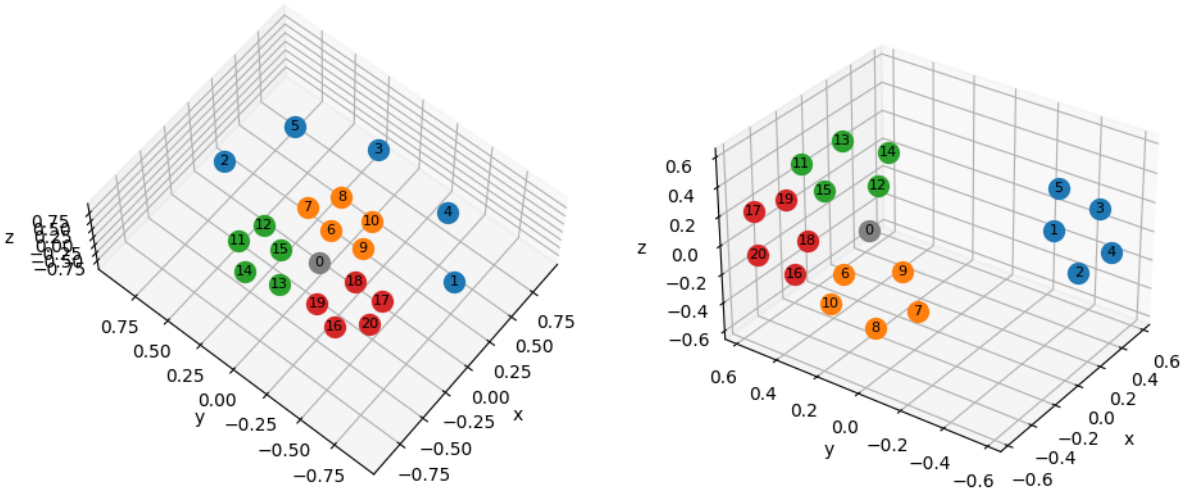


Figure 7.3: An AD neuron with 50 neural inputs. The sources to all inputs belong to the same pattern. All input sources spike at a rate of 100 Hz.

of the base task (figure 7.1a). In the figure, input positions on the periphery of the cluster form a tightly packed ring, whereas those on the inside of the cluster seem less compact. For example, connections 6 and 49 have a final angular distance of 0.04 and an average distance over 100 epochs of 0.18. Interior neighbors 10 and 13, by comparison, have a final angular distance of 0.11 and an average distance of 0.42. The average distance of all 50 input positions over 100 epochs is 0.3. The average distance of 50 positions is smaller than that of size 6 clusters when no mask is used. The reason for this is that each input is pulled into the cluster by its distant siblings and repulsed by those to whom it is too close. For large patterns there will be far more distant sibling inputs than near ones, causing a compaction of the cluster.

We ran two variations of Task 1 to test qualitatively whether clustering is sensitive to input rates or if angular positions are the result of a type of saturation due to repeated attraction. To test this we varied the spiking rates within patterns. If clustering is not due to saturation, different rates should produce different degrees of clustering.



(a) (b)

Figure 7.4: Variants 1 and 2 of Task 1. Dots are the final input positions after 100 epochs of training. Colors designate the pattern membership of the input source. The gray dot is the connection shared by all four patterns. (a) The input sources of pattern 0, inputs 0, 1, 2, 3, 4 and 5, spiked at a rate of 1 Hz. The spike rates of the other patterns remains unchanged at 100 Hz. (b) Input 0 of pattern 0 is set to spike at 1 Hz. The rates of inputs 1, 2, 3, 4 and 5 were set to 100 Hz. Again, the rate of the other patterns remained at 100 Hz.

For the first variation (Figure 7.4a), all sources of pattern 0 (0, 1, 2, 3, 4, 5) were set to a 1 Hz spike rate. The spike rate of the other patterns remained unchanged (100 Hz). The low spike rate of pattern 0 represents a scenario where its input sources are weakly correlated compared to the other patterns. In fact, the correlation between inputs of pattern 0 are, by rate, weaker than the random mask (10 Hz).

Neural inputs 1, 2, 3, 4 and 5 form a semi-circle centered around input 0 but outside the other more strongly correlated patterns 1 (orange), 2 (green) and 3 (red). Attractive forces in a complex scenario do follow paths of least resistance in the dendritic field. There are field dynamics at work that, for example, cause neural inputs 1, 2, 3, 4, and 5 to form a semi-circle. As the clusters coalesce during the initial epoch, these inputs are the last to arrive at the meta-cluster. They then migrate along the periphery toward each other. Their presence on one side of the meta-cluster causes the inputs of pattern 1 to be compressed slightly compared to the other pattern clusters (angular distance of pattern 1 was ≈ 29 compared to ≈ 31 for patterns 2 and 3).

Pattern 0	0	1	2	3	4	5	21	24
Pattern 1	0	6	7	8	9	10	21	22
Pattern 2	0	11	12	13	14	15	22	23
Pattern 3	0	16	17	18	19	20	23	24

Table 7.2: The four patterns, color coded, of task 1, variant 3. Gray designates the shared-by-all connection. Purple designates connections shared by two patterns.

This variant demonstrates that weakly correlated input rates manifest as loose clusters. And it suggests that pattern shapes (circle vs semi-circle) are the result of field dynamics created by the input rates of all sources.

For the second variation (Figure 7.4b), all input sources in all patterns were set to 100 Hz. The exception was input 0 of pattern 0, which was set to 1 Hz. The previous variant examined the effects of different input rates between patterns. This variant examines the effects of different rates within a pattern.

The five connections of pattern 0 receiving spikes at a rate of 100 Hz (inputs 1, 2, 3, 4 and 5) cluster at angular distances identical to the other patterns. But pattern 0's weak association with the shared input 0 causes it to stand away from the meta-cluster of patterns 1, 2 and 3. Although, pattern 0's non-shared inputs are twice as far from input 0 as those of the other three patterns, its presence still creates a void in the space near input 0. This is due to the vector, or direction, of the repulsive forces during the random mask caused by the tight cluster. The vector of the repulsive forces of the mask depend on input positions. Since inputs 1, 2, 3, 4, and 5 are tightly clustered, as are the inputs of the other patterns, the repulsive forces they emit during the random mask all have a similar vector. The consistent direction of repulsion causes the void in meta-cluster around 0 where pattern 0's inputs would occupy if all sources for all patterns shared the same input rate.

Variant 3 adds more shared inputs to create more complex relationships between clusters. Input 0 continues to be shared by all four patterns. Each pattern is paired with two other patterns through two shared inputs. For example, pattern 0 shares input 21 with pattern 1 and input 24 with pattern 3. In total, each pattern contains three inputs which

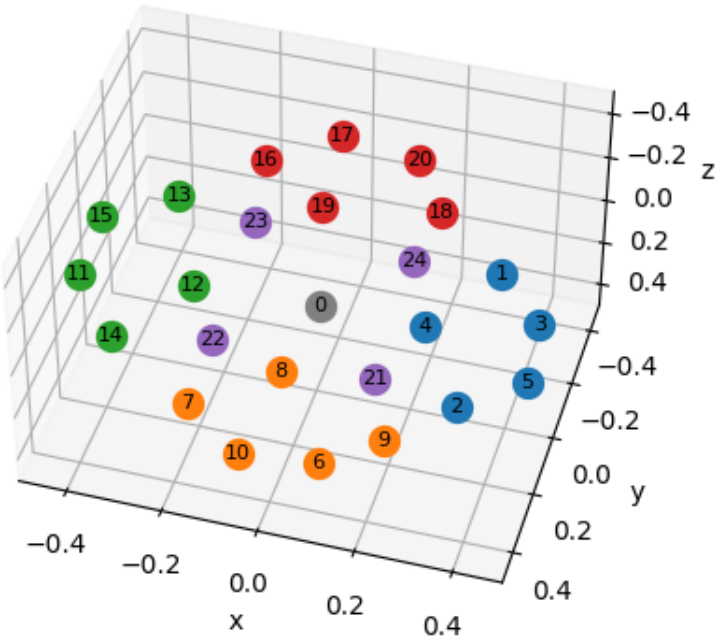


Figure 7.5: Variant 3 of Task 1. Dots are the final input positions after 100 epochs. Colors designate pattern membership. The gray dot is the neural input shared by all four patterns. Purple dots are the inputs shared by two patterns. All input sources of all inputs spiked at a rate of 100 Hz.

are shared with other patterns (Table 7.2). Figure 7.5 shows the results of training input positions after 100 epochs. As in the base version of task 1, the four patterns form a ring around input 0. Patterns with a mutual input source settle into neighboring spaces. Mutual inputs (purple dots) settle into the space between the patterns that share them. Unlike the base variant, patterns coalesce in a fixed order around input 0. They will always have the same neighbors. We did run a version of variant 3 (not shown) without the random mask to test whether added mutual inputs themselves provided more structure to the relationship between pattern clusters. The mutual inputs did increase separation between patterns by pulling the set of unique inputs of a pattern in opposite directions. There was still some degree of cluster overlap within the dendritic field but less than in the base variant.

Task 1 and its variants were designed to test how well the training of neural input positions mimicked relationships embedded in a set of inputs. Input size and pattern complexity were kept small and simple to allow for visual qualitative assessment. As a baseline measurement we expected that tightly correlated inputs (a high rate of input) would move closer together in the dendritic field than loosely correlated inputs. We also demonstrated that the AD neuron is capable of representing multiple patterns through separate clusters and relationships between clusters through their relative arrangement as a meta-cluster.

Figure 7.2 showed the result of Task 1 without the effects of the random mask. In fact, this was the first version of this task. Through these tasks we discovered that pattern clusters could occupy the same space within the dendritic field. Observation of the overlapping positions led to the creation of the random mask (see Section 6.6.1).

7.1.2 Task 2

Task 1 raised several concerns. Were angular positions stable when patterns were less clearly defined in the input data set? Or did wide variations in relationships cause instability? Secondly, how did the training algorithm behave when tasked with a larger complex pattern. To answer these questions, tasks 2 and 3 used two real-world data sets of increasing size and complexity. The two tasks we selected were drawn from the UMI Machine Learning Database. Both consist of noisy time-series data collected in real-world scenarios. For both tasks, AD neuron parameters were set to the base parameters given in Table 6.3.

The BLE RSSI Dataset for Indoor localization and Navigation is a time-series data set of RSSI (or, Received Signal Strength Indication) readings taken from "an array of 13 ibeacons in the first floor of Waldo Library, Western Michigan University" [118]. ibeacons are BLE (Bluetooth Low Energy) devices which transmit a unique identifier to nearby mobile devices [169]. RSSI is a measure of the "power present in a received radio signal"

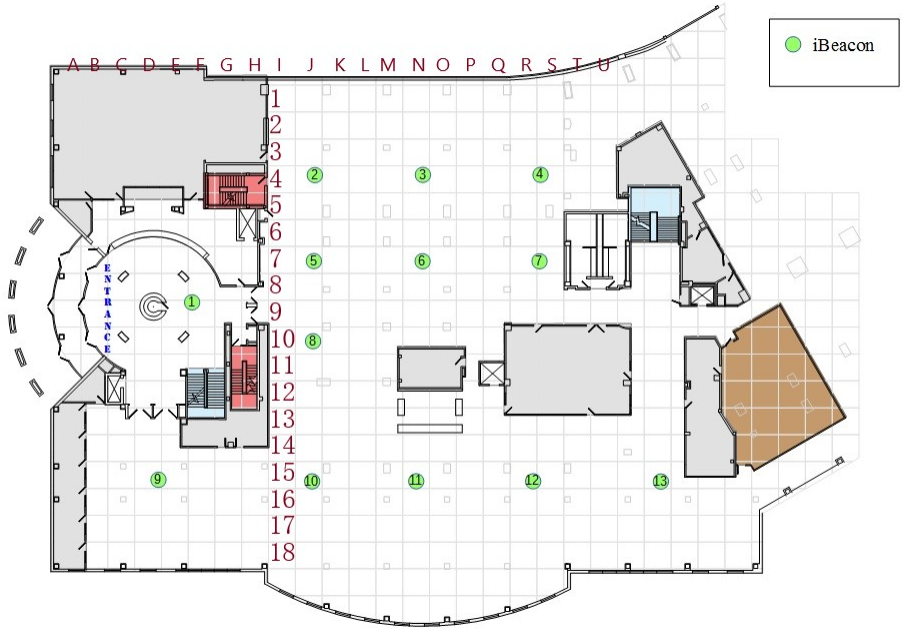


Figure 7.6: iBeacon layout. Labeled green points signify iBeacon locations. The library floor was divided up into a grid (letters for the x-axis and numbers for the y-axis) which provided the location labels for the labeled training data. Note: this plan diagram was included in the data set and was created by the authors of the data set.

[171]. The data was collected over several days for several minutes a day using an iPhone. RSSI readings were logged from the 13 ibeacons every several seconds as the researcher walked around the first floor of the library. RSSI readings range from -100 to 0. A value of -200 is assigned to ibeacons which cannot be heard. Figure 7.6 gives a map of the library and the ibeacon locations (green points).

We selected this data set for reasons of predictability and size. It contains relatively few features (thirteen) and a relatively small number of data points. Normally, a small sample size prohibits the generalizability of a model, but we are not interested in a wider problem than presented by this data set. Also, even in busy scenarios (a university library), fluctuations in RSSI readings are somewhat predictable: they increase in strength as one approaches a source and fall off as one moves away. The data is also sparse in that ibeacons were placed far enough apart such that for any given location only a few are heard at any one time. This means that a location on the library floor will have a unique subset

of RSSI readings. Our expectation was for neural input positions to cluster based on which ibeacons were frequently heard at the same place and time. From the input positions we expected to reproduce relative ibeacon locations.

An AD neuron was initialized with 13 inputs matching the number of ibeacons. Each input received one of the RSSI signals. An input layer of Izhikevitch neurons was instantiated to providing spiking input to the AD neuron. The Izhikevich neurons of the input layer used the same parameters (including the baseline input) as the AD neuron.

To use RSSI readings as input to the network, the readings were altered by adding 100 and multiplying by 4. In doing so, input covers a wider range of values ($[0, 400]$ versus $[0, 100]$) which creates a wider range of input spike rates. Readings of -200 were set to 0 to avoid them becoming inhibitory input. The data set consists of 1,420 labeled and 5,191 unlabeled examples. Labels indicate marked locations in the library.[3] Since we are only interested in training input positions based on ibeacon signals and are not attempting to locate unlabeled readings on the library floor, we have combined the training and testing sets. Furthermore, we scrubbed the data to remove any entries in which zero or one ibeacons are heard. Entries with less than two signals will have no effect on angular positions (they would be useful for radial or strength training). Scrubbing reduced the total entries from 6,611 to 4,069. During training, readings were randomized. An epoch consisted of 10% (or 407) of the total readings. Each of the 407 runs lasted for 1,000 time steps. The dendritic tree was rebuilt after each epoch. A random mask was used to separate connections which are never coactive. The random mask generated randomly timed spikes using a Poisson process at a rate of 10 Hz. The network was exposed to the random mask for 10,000 time steps after each epoch.

Figure 7.7 shows the results after 1,000 training epochs. Input positions are robust. Repeated runs of task 2 produce qualitatively similar input positions. They only variation we have seen in the placement of inputs 3 to 7. Some swap positions, e.g., 3 and 4.

[3]The data set was originally designed to be used for simultaneous location and mapping (SLAM) learning tasks.

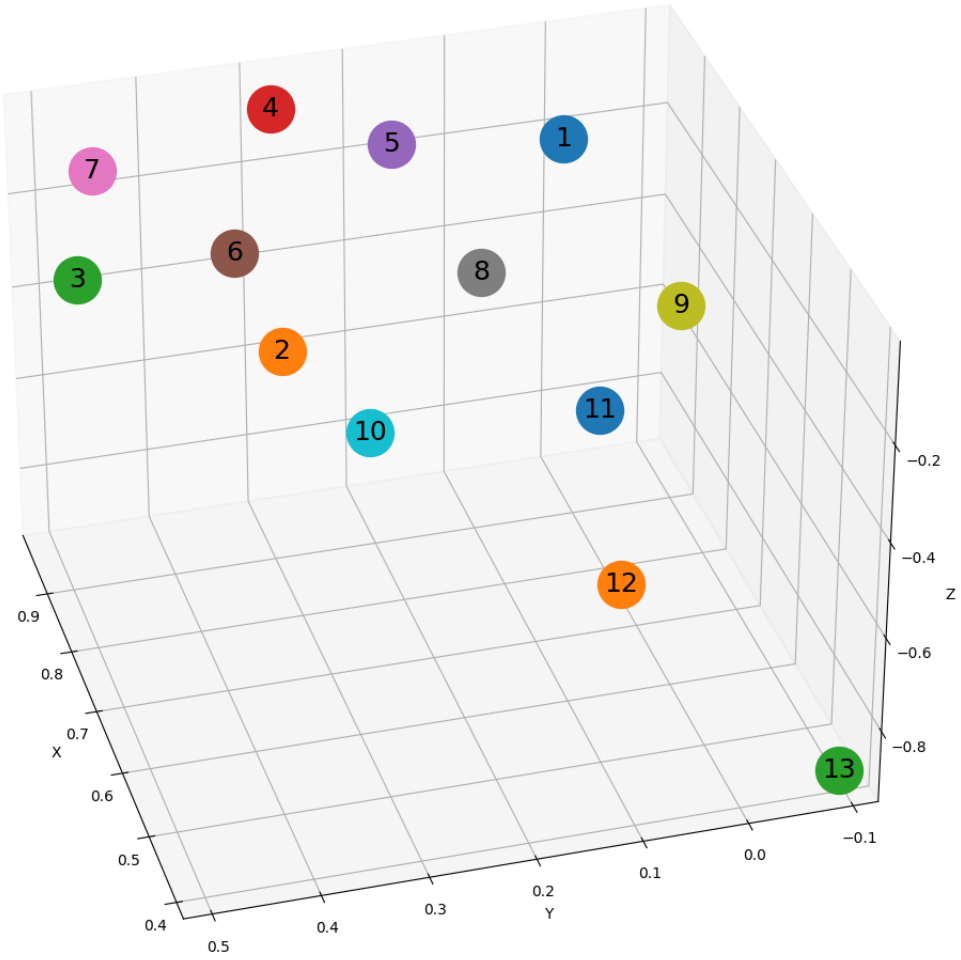


Figure 7.7: Connection positions after 1,000 training epochs.

Referencing the placements given in Figure 7.6, there appear to be some similarities between final input positions and ibeacon locations. For example, input 1 is flanked by inputs 5, 8 and 9. And inputs 10, 11, 12 and 13 show some of their original order. Inputs 2 through 8 are clustered together. There are other similarities but overall the placements deviate significantly. For example, beacon 13 is closer to 9 than 7.

To aid in the interpretations of these results, we calculated the average RSSI strength of each ibeacon when another ibeacon's signal is also heard (Figure 7.8 left). Dark blue (average strength of -100) squares indicate two beacons were never heard simultaneously (e.g., B4 and B9 or B1 and B13). Warm colors indicate a stronger average signal. The average RSSI readings are not symmetric. For example, when B9 and B13 are active, B9's

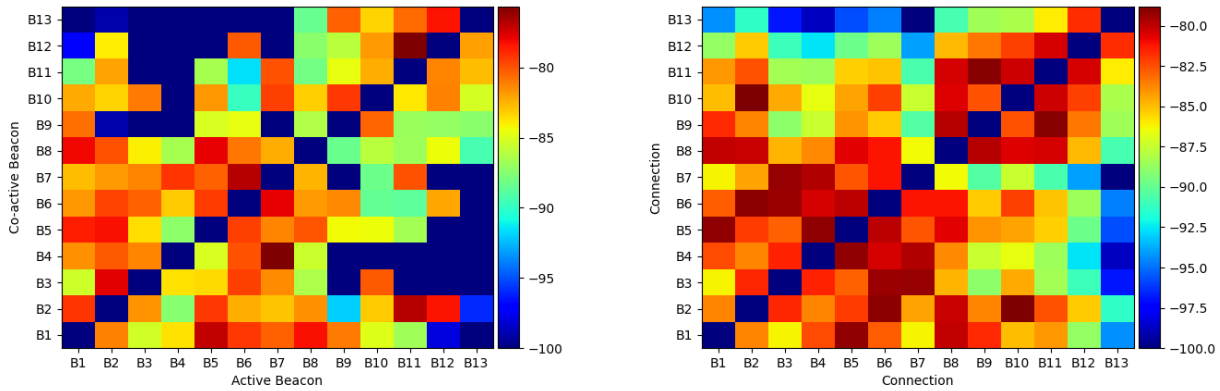
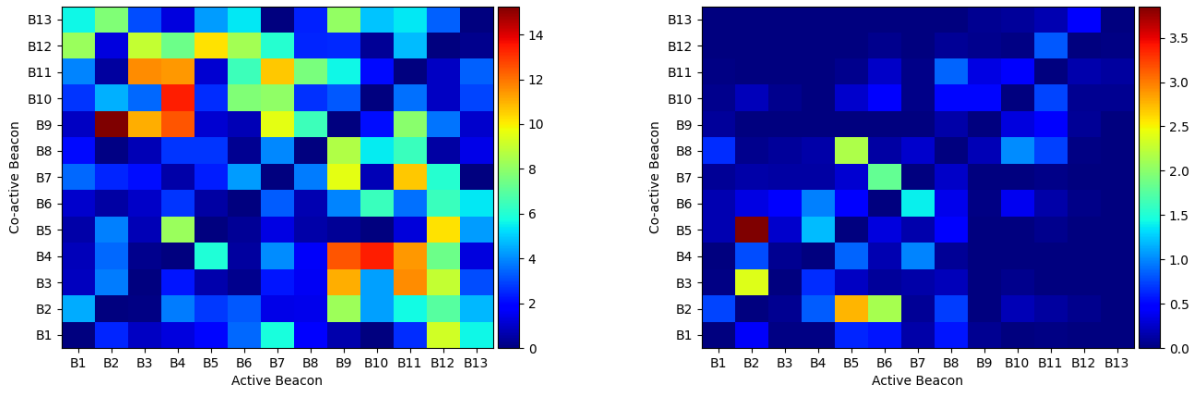


Figure 7.8: Left: The average RSSI strength of each ibeacon (x-axis) in entries where more than one ibeacon signal can be heard. Strengths are for ibeacons along the x-axis when the ibeacon on the y-axis is active. The self-reference diagonal was intentionally omitted (set to -100). Right: Normalized angular input distances rescaled to match the range of average RSSI readings in the left figure. Note: the color bars do not have the same maximum because no connections have an angular distance of zero.

RSSI is approximately -80 whereas B13 is -87. The right figure of 7.8 is a heat map of normalized distances between input positions. They have been rescaled to fit the average RSSI range of the left figure.

There is a qualitative match between the two maps with two groups related by RSSI strength and angular distance: B1-B7 and B8-B12. B8 belongs to both groups. This is evident on the floor map in that B8 lies centrally near the main doors. Similarly, input 8 lies at the center of the input cluster in the dendritic field.

How well do input positions represent the layout? In the previous task co-active sources produced circular or ring clusters. This was caused by sets of related input sources. Limiting the size of the ring clusters to 3 or 4 inputs exposes the second-tier relationships in the data set. First-tier relations are those between two sources. For example, the positions of inputs 10, 11 and 12 form such a cluster. The cluster does not match the linear positions of their corresponding ibeacons. This is likely due to all three being commonly heard. The shorter distances between 10-11 and between 11-12 show that these are the neighboring pairs. The longer distance between 10-12 suggests they are heard together less



(a) (b)

Figure 7.9: Left: Absolute value of the element-wise difference of the two figures of 7.8. The color bar's unit is RSSI. Right: The left figure scaled by the normalized count of entries in which a pair of ibeacon readings are heard together. Counts were normalized by dividing by the maximum count of all pairs.

often. On the other hand, the positions of inputs 11, 12 and 13 suggest the corresponding ibeacons are not heard together as strongly. However, figure 7.8 suggests that the RSSI readings for the latter group are somewhat stronger. An alternative explanation why they differ might not have to do with the 10-11-12 cluster but with the 8-10-11 cluster. ibeacons 8 and 11 are heard frequently heard together, so it likely the shape differences are due to 11 being pulled toward 8 rather than 12 to 10.

The left figure of 7.9 shows the element-wise difference of the two previous figures. Angular distances closely match the average RSSI. The mean and median of all differences are 3.84 and 2.75 respectively. The two large clusters of beacons mentioned above are evident in the dark blue blocks on the bottom left and top right. The stronger relationships between these ibeacons signals aids in positioning inputs to match. The larger differences (top left and bottom right) occur between beacons which do not share strong average RSSI.

Figure 7.9b scales the left by a normalized count of the number of times pairs of ibeacons are heard together. In other words, we are emphasizing the differences between pairs which are frequently heard together and deemphasizing those with few. Doing so

eliminates much of the differences between average RSSI and angular distance (mean: 0.26, median: 0.05).

Interestingly, the input positions also encode, albeit indirectly, obstacles within the space. Similar to the mask-less variant of task 1, relative connection positions are less meaningful between input sources with infrequent co-activity. Certain spatial relationships in the real-world placement of ibeacons do not manifest in the dendritic field. For example, B12 and B13 neighbor B7. However, their signals, (B7-B12) or (B7-B13), are never heard together. It is probable this is due to the large room and elevator separating them thus limiting the paths between them. The same applies to B6 as well as to B1-B10. The stairwell limits paths between them to also passing through B8 or B9. This is also evident in the dendritic field with the position of input 8 between 1 and 10.

The angular training algorithm encodes in the dendritic field paths through a complex space. This presents an interesting application to this specific part of the AD neuron training algorithm. We believe the encoding could be improved with many more examples. The data set contains an uneven distribution of readings over the ibeacons. It is unclear from the data set whether the paths are specific to the task, random or represent some sort of daily activity in the library. All three present interesting scenarios for training.

From task 2 we learned that input positions as a result of real-world data encode features external to the feature set. Input positions are determined by direct and indirect relationships between features. There can be value in examining the differences between the real-world and encoded spatial positions.

In the next task we expand the number of input sources to test whether patterns in the dendritic field which form through angular position training are both robust (i.e., repeatable) and stable.

7.1.3 Task 3

Task 3 builds on task 2 by increasing the complexity of the problem domain--correlated brain activity--and the number of features in the data set from 13 to 64. The data set used in task 3 comes

> "from a large study to examine EEG correlates of genetic predisposition to alcoholism. It contains measurements from 64 electrodes placed on subject's scalps which were sampled at 256 Hz (3.9-msec epoch) for 1 second. There were two groups of subjects: alcoholic and control. Each subject was exposed to either a single stimulus (S1) or to two stimuli (S1 and S2) which were pictures of objects chosen from the 1980 Snodgrass and Vanderwart picture set. When two stimuli were shown, they were presented in either a matched condition where S1 was identical to S2 or in a non-matched condition where S1 differed from S2." [46].

We used the large data set consisting of 10 alcoholic and 10 control subjects.[4] Each subject performed 10 trials of each of the three tasks (S1, S2 match and S2 non-matched). The data recorded from each trial consists of 256 readings from 64 different electrodes. Electrodes are placed at specific positions on the scalp. The positioning used in this task follows the 10/20 standard (Figure 7.10). The data collection is described in Zhang 1995 [182]. For our purposes, we were not concerned with determining whether a reading came from an alcoholic or control individual or identifying differences between the types of tasks. Therefore, we used only S1 trials from control subjects.

The EEG data set was selected as a step up in size and complexity compared to the ibeacon data set. It contains four times the features and many thousands more data points. However, it is also similar to the ibeacon set in two important ways. First, it is time-series

[4]The EEG database contains three slices of the data set: a small data set (2 subjects per type and 3 trials per trial paradigm), a large data set (10 subjects per type and 10 trials per paradigm) and a full data set (122 subjects and 120 trials).

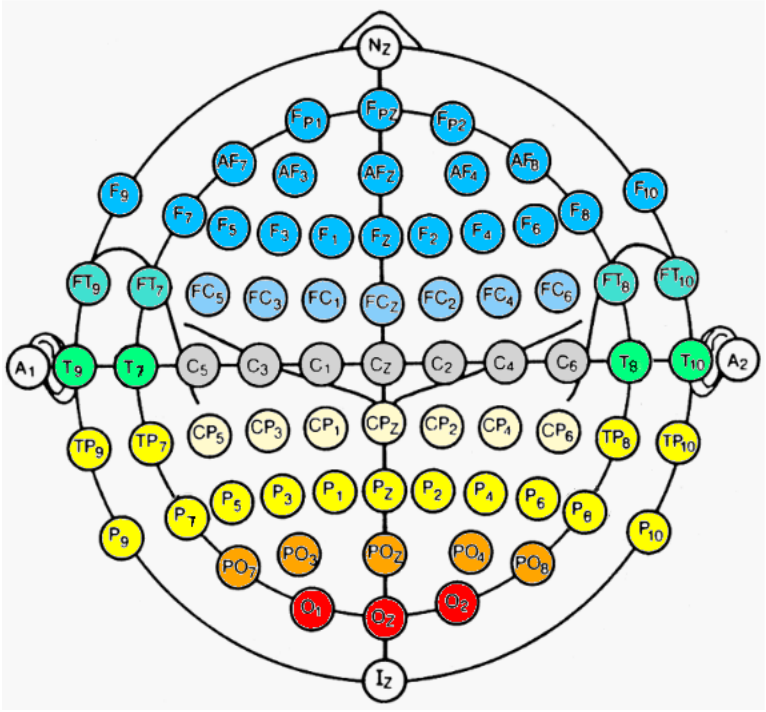


Figure 7.10: The EEG position standards. [3] We colored the electrodes to match the coloring scheme used by our results. Not all electrodes pictured in this standard diagram were used in the original task. We provide it to aid in the assessment of trained input positions. To aid with orientation, N_z is the nose. F_p: Frontal polar. AF: Anterior Frontal. F: Frontal. FC: Frontocentral. FT: Frontotemporal. CP: Centroparietal. T: Temporal. TP: Temporal Posterior. P: Posterior. PO: Parieto-occipital. O: Occipital. The positions in white do not appear in the task.

data. Second, multi-site EEG readings have poor spatial resolution and therefore, brain activity is often read by multiple adjacent electrodes [29] but not all, which is analogous to the way some but not all ibeacons can be heard from one location in the library. We hypothesized that angular position training could use this fact to spatially associate co-active electrodes. Similar to the ibeacon task, the input positions would mimic electrode placement. The increase in size and complexity of this task also aided in refining result data collected during the training process.

An AD neuron with 64 inputs was initialized using the base parameters in Table 6.3. Input positions were randomized. Although the neuron was trained on all 64 inputs, readings from 3 electrodes (labeled X, Y and nd) were excluded from the results because

they do not appear on the electrode standard position chart (Figure 7.10). The neuron was trained using the P1 trial readings from all control subjects in the large data set. The neuron was trained for 5,000 epochs. An epoch consisted of the entire data set. The dendritic tree was rebuilt after each epoch. In each epoch, trials were randomized, but readings within each trial were not randomized.

In the original task, readings were sampled at 256 Hz for 1 second, resulting in 256 readings per trial. The majority of tasks in this work expose AD neurons to each example for 1,000 time steps. To match this training time, each reading was used as input for 4 time steps before being swapped for the next in the series. This resulted in a training time per trial of 1,024 time steps.[5]

EEG readings required some manipulation to convert them into a useful range of values appropriate for spiking neuron models. The vast majority ($\geq 99\%$) of individual EEG voltage readings in this data set fall in the range [-30,30] μV. A few are so large and small that rescaling with these outliers would place 99% of the data in a small range.[6] So we opted to set any reading outside the desired range to the minimum or maximum of the desired range, -30 and 30. We then added 30 to all readings, shifting them into the range [0,60] and then scaled them by a 10, for a final range of [0,600].

We tested several other transformations of the EEG readings into input (e.g., [-20,20] bounds, no bounding, no shifting from the negative domain and different scalars). For those alterations which produced spikes, the results were qualitatively similar. Different spike rates produced a denser or looser cluster. No bounding of EEG readings resulted in the majority of readings falling within a narrow band which made them less distinguishable from each other (depending on scaling). Less distinction between input values has two effects. It causes more variability in the results since similarities in input patterns between inputs are less unique. And more epochs are required to reach a stable pattern.

[5]An alternative way to expand the number of readings to match the 1,000 time step target would be to run each trial four times back-to-back. This alternate expansion does not appear to change the results.

[6]The maxmimum reading in the data set is 453.847. The minimum is -213.491.

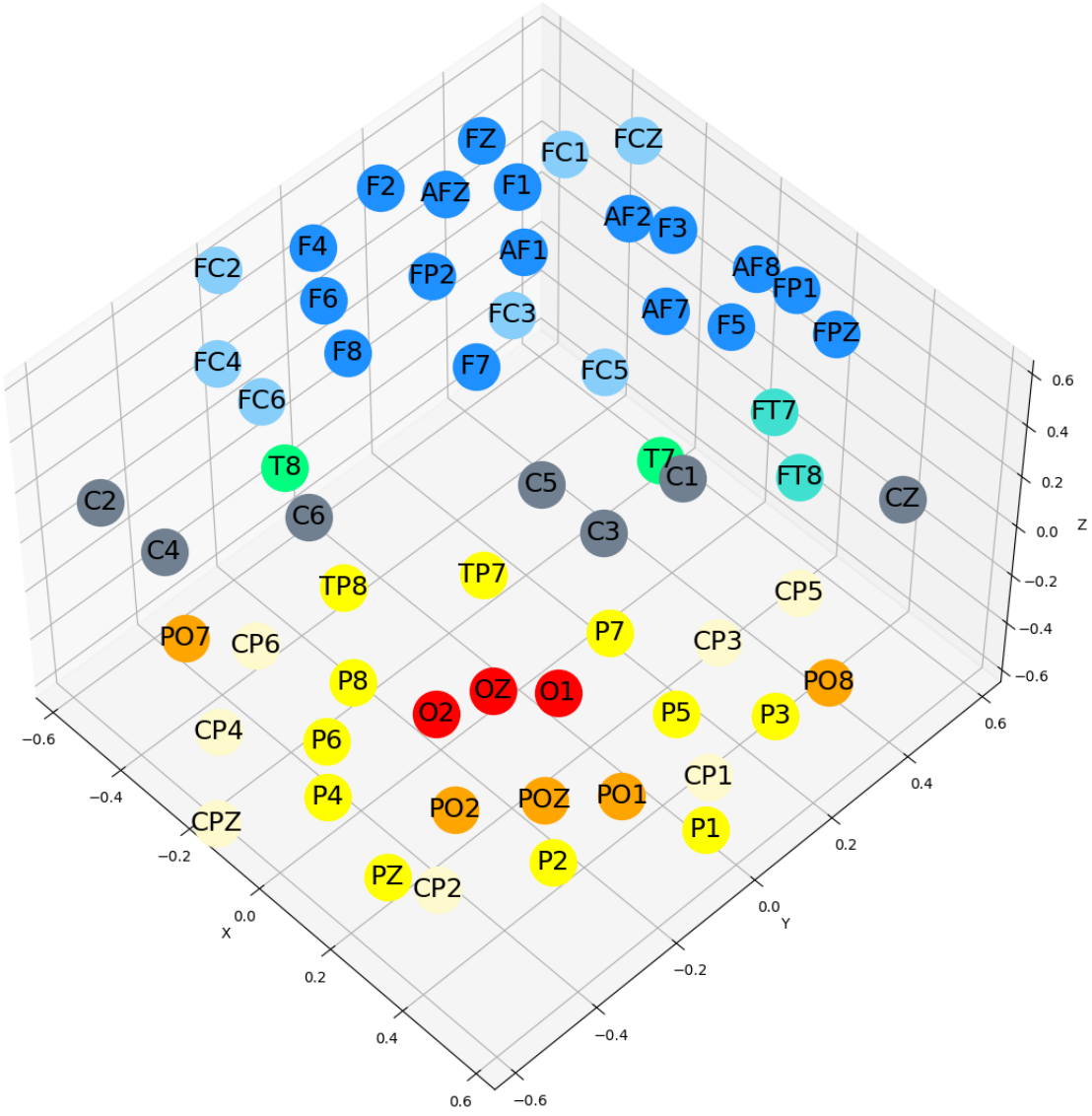


Figure 7.11: Final connection positions after 5,000 epochs. Colors and labels match those in Figure 7.10. Not all electrodes in the standard placement figure appeared in this task, e.g. IDs 9 and 10.

Figure 7.11 shows the final input positions after training. Trained input positions approximately match electrode placements on the skull. Inputs receiving frontal readings (blue) are clustered at the top. Neural inputs receiving parietal and occipital readings (yellow, orange and red) cluster together at the bottom. Central (gray) inputs separate rostral (forward) from caudal (rearward) positions. There are curious anomalies, however.

O2, OZ, and O1 are more forward than PO2, POZ and PO1. And PZ, P2 and P1 are even more rearward. Although out of order, the O's and PO's are aligned by ID number.

Electrode IDs are such that even numbers fall on the right side of the brain and odd numbers on the left. While this looks to be reversed in Figure 7.11, it is not. The orientation of the figure is from the point of view of the base of the skull looking up.[7] Not all inputs follow this left-right dichotomy; PO7 and PO8 (parietal-occipital) are reversed. The majority, however, are separated by the center line.

There are several other anomalies. Frontocentral (FC) and parietal (P) are out of place locally and to some degree enclose frontal (F) and occipital (O) inputs. Temporal (T) inputs are closer to the medial line than their skull placement. Some of the medial inputs (Z) are off to one side.

We hypothesized that the poor spatial resolution of EEG readings would manifest a general cranial map in input positions to demonstrate the angular training algorithm's use with a larger and more complex set of inputs. Each time the task is run, placements are slightly different. To demonstrate this, Figure 7.12 shows four additional runs with the same parameters. Input positions are not identical but do reach a final position which is generally consistent with the front-to-back and side-to-side placement of electrodes on the cranium.

In the repeated runs (Figure 7.12), we demonstrated that training does not always result in the same final state. On the one hand, the final state depends on the initial state--the initial input positions. As we demonstrated in the previous task, clusters themselves can inhibit movement of individual inputs. However, differences in relative input positions between runs may not be caused directly by a combination of initial state and cluster inhibition. We wondered whether they could be caused by moment-to-moment changes in position. The larger concern is that high variability in the relationship between features of examples of the same pattern or category could cause instability in input

[7]Dendritic field figures are oriented this way because, in general, the concave view offers a better view of the positions than the convex view. Only in this example does this orientation result in an awkward view.

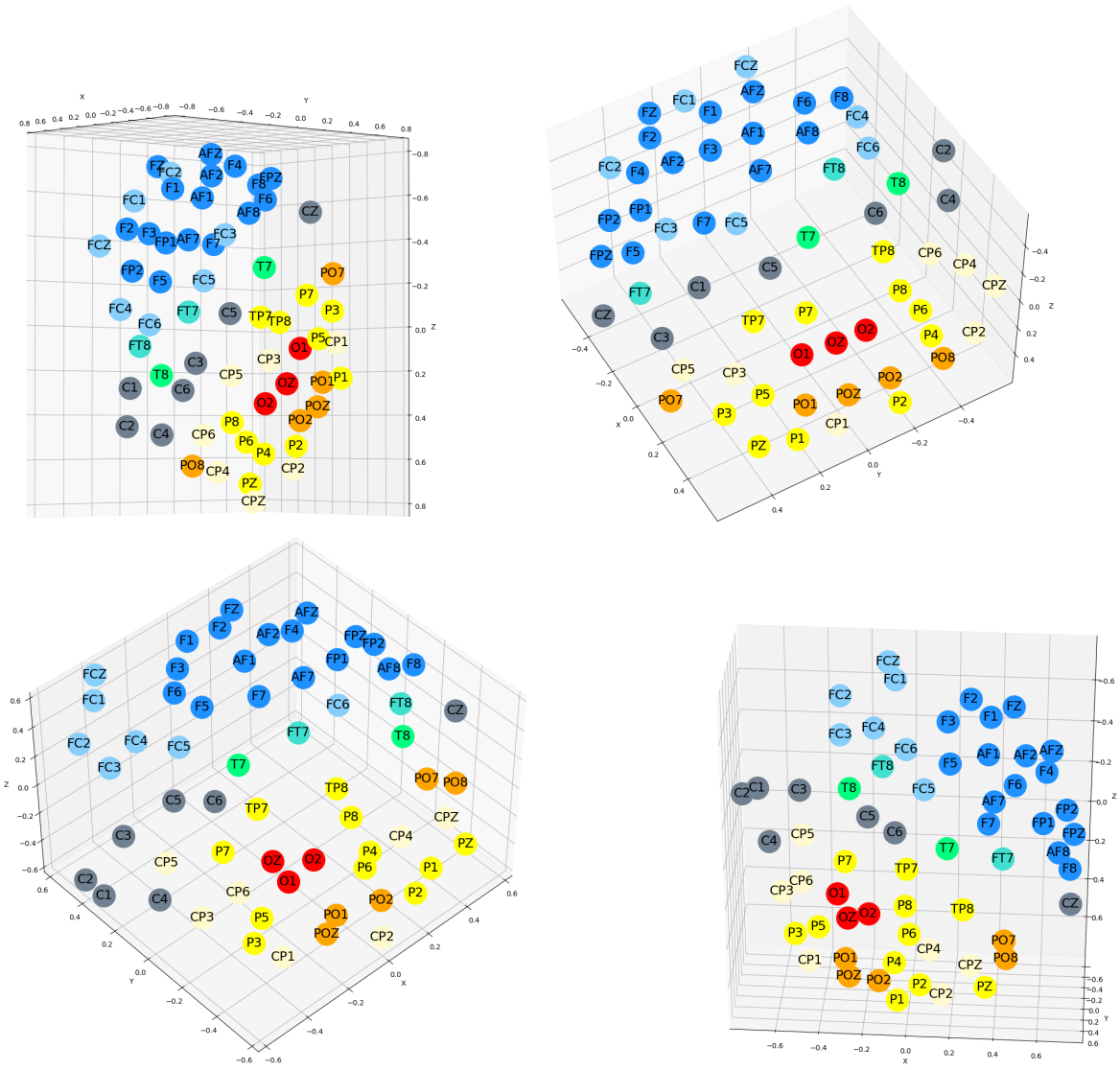
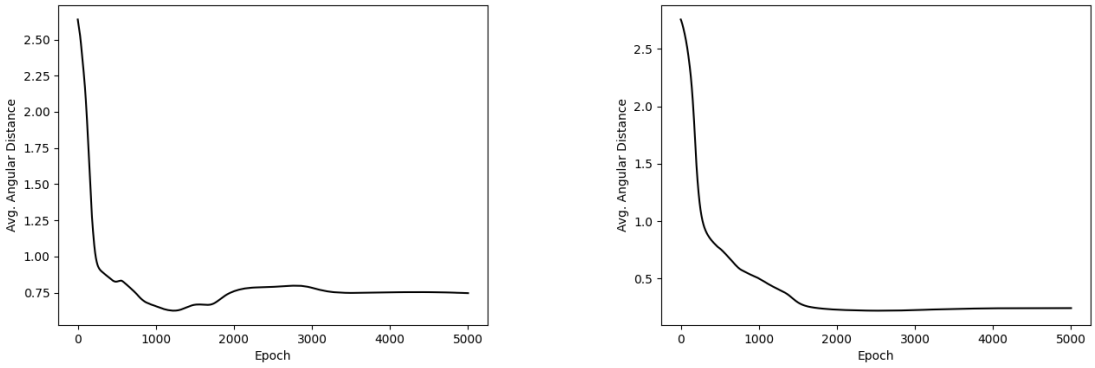


Figure 7.12: Four dendritic fields trained on the same set of EEG data as that used to make Figure 7.11. All parameters are identical. Training lasted for 5,000 epochs.

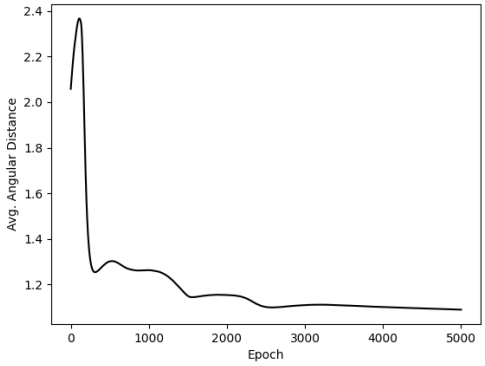
positions encoding the pattern. This could result in a input having a bounded orbit or jitter around an optimal position. For example, wide swings in input values might pull an input in opposing directions.

The figures of 7.13 show that input positions do stabilize after several thousand epochs and that moment-to-moment changes are not the cause of these variations in input position. The top two figures (7.13a and 7.13b) show the distance over time of two inputs whose input source are electrodes adjacent on the cranium. The bottom figures (7.13c and

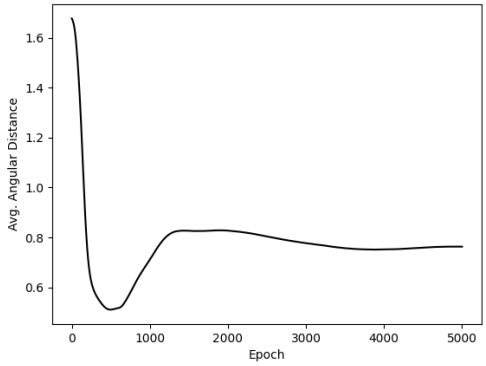


(a) FP1-FP2

(b) O1-O2



(c) FPZ-OZ



(d) T7-T8

Figure 7.13: Four examples of the change in angular distance between pairs of connections over 5,000 epochs. Data is from the same run as 7.11.

7.13d) show the distance over time of two inputs whose electrodes are on opposite sides of the cranium (front-to-back and side-to-side). In all four cases, distances stabilize anywhere between 2,000 to 3,500 epochs. This time frame is the same for all pairs we tested across all runs. A small learning rate is key to reducing instability in angular positions. Although somewhat larger learning rates (0.01) will cause jitter in angular positions, relative distances still stabilize within some bounded range.

We do not try to draw any neuroscientific conclusions from these input placements. In spite of the variations between runs there are curious sub-patterns which recur. Given the training examples of previous tasks, we can conclude that such repeated patterns are not random but are the product of correlated activity. The trained positions shown above are

the result of readings from 10 different individuals. We have also run this task using readings from a single subject. Training on single subject readings results in input positions that deviate from electrode placements. This suggests that our multi-subject task creates in the input positions an average of the differences in readings between subjects, giving us a rough reproduction of electrode placements. We have not followed up on these results since our goal at this stage is to show qualitative, predictable behavior in the training of input positions. Investigating this line would require the use of the full data set of 122 subjects and 120 trials. Identifying individuals by their EEG readings or differentiating between the EEG readings of alcoholic versus control subjects using input positions in the dendritic field is left for future work.

7.1.4 Summary of Angular Training

The EEG task shows that the angular training algorithm produces consistent and stable input positions for larger, more complex data sets. Stability depends strongly on the learning rate. Final input positions depend on initial positions and the order of the training data examples. The angular training algorithm is deterministic in that the same initial positions, input update order and training data order will produce the same final positions. All tests demonstrated that the angular training algorithm encodes input data patterns in the relative angular input positions. They capture relationships between individual features of the data set as well as meta-features, such as the relationships between patterns which share features.

There are limitations to the angular training algorithm. The primary limitation stems from encoding feature relationships in a reduced dimensional space. The angular component of input position is a 2-dimensional encoding (i.e., the shell of a sphere). For data sets with more than 3 features, a 2-dimensional space may not be able to capture all relationships perfectly. Clusters of inputs suffer from position lock, the inability to change position due to being surrounded by sibling inputs, and compression (the discussion section

(9.3.4) proposes a potential alteration which would alleviate some problems caused by dimension reductions).

The circumference of the dendritic field has a strong impact on how patterns cluster. The circumference of the dendritic field is measured in temporal units. The default value used in these tasks is 100 time steps. Input positions on opposite sides of the dendritic field imply a difference of 100 time steps between the arrival time of signals. Therefore, the circumference should be large enough to encode feature relationships with the longest (desired) interval. If the dendritic field is too small temporally long relationships will never become stable since they will always be too close along one trajectory. Even if the field is wide enough for all relationships between individual features, it must be wide enough to separate meta-patterns (e.g., clusters of inputs). Finally, the circumference of the dendritic field must be tailored in tandem with the radius. During tree-building distances between inputs depend on both angular and radial distances. A wide discrepancy between the two can result in a tree which predominantly favors either angular or radial inputs.[8] Our method is not to use independent values for A_{max} and R_{max} but to let one determine the other using the geometric properties of a sphere (circumference from radius, or vice versa).

Angular positions are further analyzed in later tasks when all training algorithms are active. In the next section, we present initial tasks examining the behavior of the radial position training algorithm.

7.2 Radial Position Training

Trained radial positions encode the temporal interval between input received by a specific input and neural output, i.e., output generated by the AD neuron's S-compartment. *Neural output* pertains only to S-compartment output; whereas, *compartmental output* is a more general term which includes neural output. The radial component of the dendritic field, like the angular component, is measured in temporal

[8]We are not suggesting this is a negative to be avoided. It is possible certain scenarios would benefit from favoring one type of input over the other. Further testing is needed.

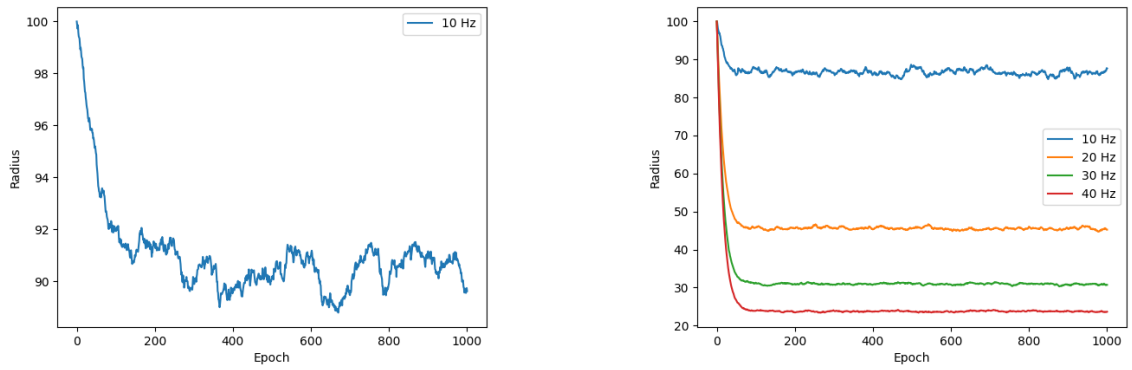
units. An input which receives a spike on average 20 time steps before an S-compartment spike should have a radial position of 20. To test whether the radial training algorithm achieves this we tested it in isolation from the angular, tree-building and strength modification algorithms.

Radial training is more dependent on the AD neuron as a whole than is angular training.[9] It requires that input is capable of generating neural output. This implies that a dendritic tree links inputs to the S-compartment. It also requires that certain parameters, e.g., input and compartmental weights, are large enough to generate output. And that there is sufficient input to the AD neuron. To put it another way, in a silent AD neuron there is no radial change.

For these tasks we made as few assumptions about the other aspects of the AD neuron as possible without relying on degenerate forms. The initial state of the AD neuron proved sufficient. Each input formed its own B-compartment and each single-connection compartment was linked directly to the S-compartment. Neural inputs were placed at a radius of 100, the maximum extent of the dendritic field. Angular positions were randomized. The base parameters given in Table 6.3 were used.

Our approach was to test the radial training algorithm under two conditions. First, we wanted to test its purest form: the input-output timing relationship in the absence of complex dendritic trees caused by angular clustering. Our goal was to discover whether individual radial positions accurately encode spike rates. To discover this we tested AD neurons whose inputs received signals at varying rates. For these simple tests (Task 4), angular and strength training were disabled. Second, we wanted to test what impact, if any, intervening inputs and compartments have on radial positions (Task 5). Once the dendritic tree becomes complex and signals received by individual inputs must pass through multiple branches, does its shape distort input-output encodings?

[9]In fact, angular training (i.e., the clustering of points on a spherical shell [80]) can be done without the rest of the AD neuron.



(a) (b)

Figure 7.14: Radial position tasks. Left: 1 input receiving signals at a rate of 10 Hz. Right: 4 inputs receiving signals at different rates: blue 10 Hz, orange 20 Hz, green 30 Hz, and red 40 Hz.

7.2.1 Task 4

We initialized an AD neuron with 1 neural input. The source to the single input generated spikes using a Poisson process with a mean of 10 Hz. The single input was placed in a B-compartment and connected directly to the S-compartment. Training consisted of 1,000 epochs with 100 runs per epoch. Each run lasted for 1,000 time steps. No random mask was used and the dendritic tree was not rebuilt after each epoch. The neural input was given a weight of 500 which is sufficient for a signal to a single input to generate S-compartment output.

From an input rate of 10 Hz we expected the radial position to remain near its initial value of 100 as this corresponds to the average interval between spikes generated at this rate. Figure 7.14a shows the result for a single 10 Hz input. The input's radial position decreased from its initial position of 100 to approximately 90 over the first 200 epochs and remained there. From the 200^{th} epoch to the end, the mean radius was 90.4; the SD was 0.66; and the minimum and maximum were 88.7 and 91.7. While the final radial position is stable within a range around 90 time steps (± 2), the position fluctuates.

We expanded on the one input task to test whether multiple connections affected radial position and whether increasing the spike rate reduced fluctuations. The next variant, the

Rate	10 Hz	20 Hz	30 Hz	40 Hz
Mean Radius	86.5	45.6	30.9	23.7
SD Radius	0.69	0.32	0.19	0.13
Min Radius	84.8	44.7	30.9	23.3
Max Radius	88.6	46.6	31.4	24.1

Table 7.3

results of which are shown in Figure 7.14b, consisted of 4 inputs with input rates ranging from 10 Hz to 40 Hz, spaced at 10 Hz intervals. The resulting radial positions for each input are given in Table 7.3.

The results of these two simple tasks have several curious aspects. Training produces mean radial positions which are approximately 10% less than expected. We are not entirely certain of the cause of this discrepancy but we suspect it is caused by the parameter selection that governs the impact of the dendritic tree on signal strength. Under the base parameters, not all input spikes produce an S-compartment response. Instead, multiple spikes within some interval are required to generate an S-compartment spike. Input patterns with shorter intervals are favored, causing the imbalance.

Neural inputs with faster input rates will cause a slight decrease in the radius of inputs with slower rates. And the size of the decrease depends on relative input rates. For example, the 10 Hz input's mean radius consistently decreased in the four input example to approximately 86.5 from 90. Duplicating the single input task (Figure 7.14a) with a 40 Hz rate instead of a 10 Hz shows no increase from the radius it achieves when three additional slower inputs are present. The mean radius for a single 40 Hz input is 23.7. However, a variant of the four input task in which the other input rates are faster than 40 Hz (e.g., 50, 60 and 70 Hz) results in a slight decrease to 23.5. Although this is a decrease in mean radius of only 0.2, the results are consistent.

Another interesting feature of these results is that inputs with higher input rates have stabler radial positions. This is likely due to the shorter intervals between input spikes. Shorter intervals result in smaller radial changes once a mean position is reached because

Pattern 0	0 (100 Hz)	1 (80 Hz)	2 (60 Hz)	3 (40 Hz)	4 (20 Hz)
Pattern 1	5 (100 Hz)	6 (80 Hz)	7 (60 Hz)	8 (40 Hz)	9 (20 Hz)
Pattern 2	10 (100 Hz)	11 (80 Hz)	12 (60 Hz)	13 (40 Hz)	14 (20 Hz)
Pattern 3	15 (100 Hz)	16 (80 Hz)	17 (60 Hz)	18 (40 Hz)	19 (20 Hz)

Table 7.4: The four patterns, color coded, of Task 5. Gray designates shared neural inputs. Hz in parenthesis give the spike rate of each input source.

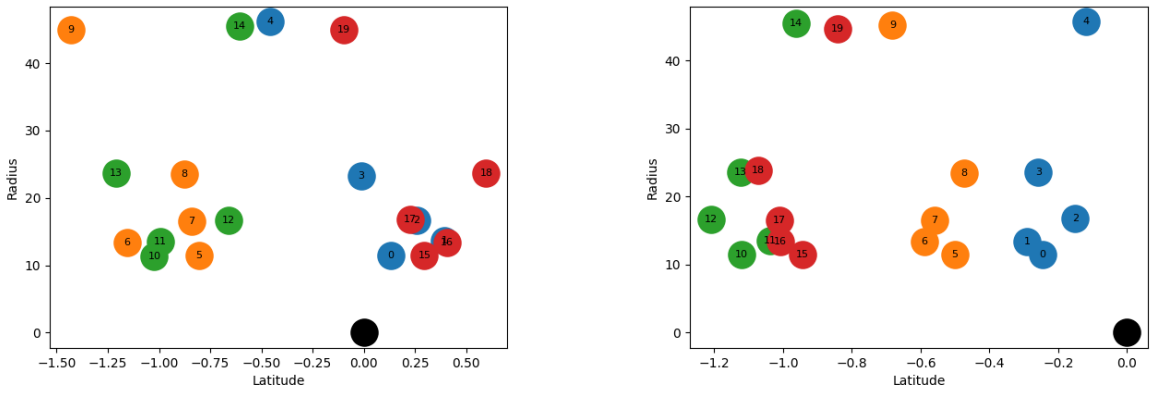
the probability for large discrepancies between input-output timing and radial position is smaller. This suggests that tying radial learning rates inversely to the magnitude of the radial position could improve stability.

Overall, the behavior of the radial training algorithm performed as expected. The radial position of a neural input is determined by how closely the signals it receives are correlated with S-compartment output. Even if rates and radius do not match, their relationship is predictable. Higher input rates result in more proximal radial positions and vice versa. The exact relationship between rates and radial position is likely to be something to be tuned on a per task basis.

7.2.2 Task 5

Task 5 builds on task 4 by testing the relationship between input positions when both angular and radial training are active. Do patterns with varying input rates cause input positions to cluster? With radial training active, cluster shapes become 3-dimensional. What shapes do these clusters form? Does the combination of radial and angular training require alterations to the AD neuron's parameters?

Input sources were grouped into the 4 patterns given in Table 7.4. The patterns and task parameters are similar to task 1's base variant. Tasks consisted of 1,000 epochs of 100 runs. Each epoch was followed by a 10,000 time step random mask. The random mask was kept at 10 Hz. The input rate of the random mask must be less than the minimum spike rate or it will disrupt clustering of weakly correlated inputs. Input rates varied within patterns. Input rates within each pattern ranged from 100 Hz to 20 Hz at increments of 20 Hz. Each pattern contained one neural input for each rate.



(a) (b)

Figure 7.15: Trained input positions for task 5. Colors indicate pattern membership (see Table 7.4). The black circle is the S-compartment position. The graph uses each input latitude for the x-axis. Left: Max angular distance (π) represents 100 time steps. Right: Max angular distance (π) represents 500 time steps.

Rate	20 Hz	40 Hz	60 Hz	80 Hz	100 Hz
Mean Radius	45.3	23.5	16.6	13.4	11.4
SD Radius	0.36	0.15	0.10	0.08	0.07

Table 7.5: Average radial positions of Task 5 taken from the run depicted in Figure 7.15a.

In Figure 7.15a, trained radial positions are arranged within the dendritic field according to their input rates. Inputs with high rates of input (100 Hz) are closer to the center of the dendritic field than those with slower rates. The 20 Hz and 40 Hz radial positions (Table 7.5) match those from Task 3. Again, sharing the dendritic field with inputs receiving faster rates of input causes a slight decrease in radius of the less active inputs. Since the decreases are identical to the ones from task 3 (23.7 to 23.5 in the case of the 40 Hz input for example) and this task contains four times the number of inputs with higher rates, there appears to be a saturation effect on the ability of sibling inputs rates to influence the radial positions of less active inputs.

This task also revealed how pattern clusters with varying rates of input arrange themselves within the dendritic field. Figure 7.15 uses latitude to display radial positions. In doing so, it also demonstrates that in spite of the differing radii, inputs still cluster

angularly (horizontally in the graph). Since figure 7.15 is limited to latitude, not all angular displacements are visible. In the left figure pattern clusters overlap in some cases (blue-red and orange-green). This is not due to graphing input positions along one angular dimension (latitude). A longitudinal graph would show something similar (see Figure 7.16 for a 3D display). Pattern overlap is caused by two aspects of the task: loose clusters and a narrow dendritic field. Loose clustering is caused by slow input rates to some of the inputs. Slow rates are encoded in larger angular distances between inputs. Loose pattern clusters can be seen in the latitudinal positions. For example, inputs 10 and 11 in Figure 7.15a are close latitudinally. Connections with high spike rates belonging to the same pattern tend to train closer together and vice versa. Input 14 which spikes at one-fifth and one-fourth the rates of inputs 10 and 11 is 0.5 radians distant. All less active inputs (4, 9, 14 and 19) in Figure 7.15a are outside the latitudinal bounds defined by the other four belonging to the same pattern. This leads into the second cause: the dendritic field's temporal size.

Second, pattern overlap is caused by a relatively narrow (100 time steps) dendritic field in comparison to the input rates. The mean width of the latitudinal bounds of all patterns in 7.15a is 0.69 radians which when translated to the temporal size of the dendritic field is 21.7 time steps. All four patterns together do fit in a 100 time step field but only just. Increasing the circumference of the dendritic field to 500 time steps encodes temporal differences for pattern clusters in a wider space allowing them to separate. In Figure 7.15b the mean latitudinal bound drops to 0.21 which translates to 33.4 time steps. The latitudinal range occupied by each pattern is broader but the wider field aids in cluster separation. Clusters show much less overlap and overlap disappears when viewed in 3-dimensions (Figure 7.16). These ranges are robust across repeated test of the two variants of this task. As we will demonstrate in the next section through a repeat of these tasks, the ability for an AD neuron to separate patterns within the dendritic field can be important for tree-building.

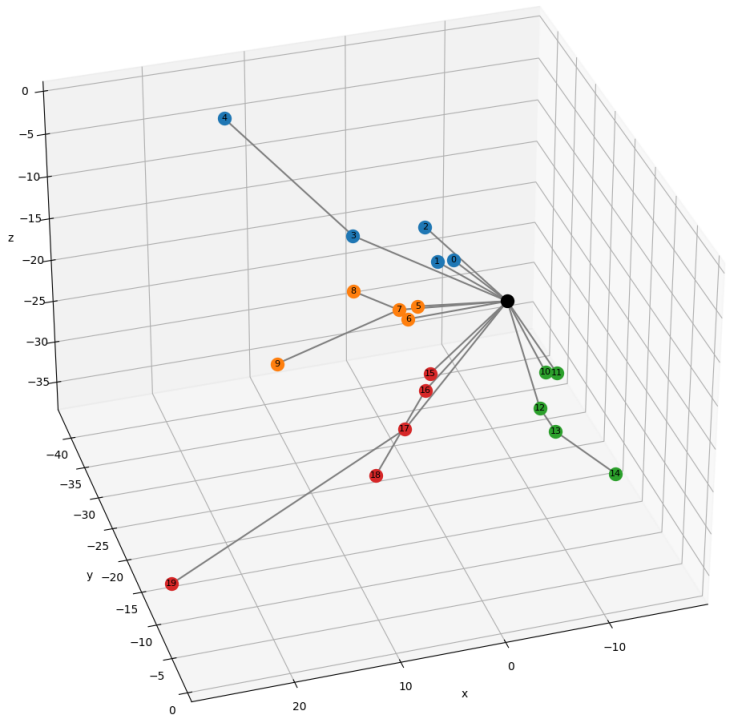


Figure 7.16: Connection positions of the same run depicted in Figure 7.15b. The dendritic tree (gray lines) are included to show depth.

7.2.3 Summary of Radial Training

Radial training positions neural inputs based on the timing of their contribution to neural output. Short intervals between neural input and output move an input closer to the center of the dendritic field. Long intervals push inputs to the dendritic periphery. In these simple examples, input rates largely determined radial position, but this might not always be true. We can imagine a dendrite containing a mix of inhibitory and excitatory inputs based on connection weight. Inhibitory inputs could silence or delay neural output causing a discrepancy between input rate and radius.

It was important to demonstrate the above correlation between timing and radial position. This correlation is one of the effects of training captured by our tree-building algorithm. Signals from more active and participatory inputs have shorter propagation times and are less impacted by compartmentalization.

7.3 Tree-building

Tree-building is the process of linking neural inputs based on their radial and angular position within the dendritic field. The process used to build the tree is given in Section 6.5.7. The following tasks focus on three goals. First, how well does tree-building capture relative input positions. Does it take advantage of the encodings and thereby reinforce them, or does it disrupt the spatial encodings? Second, what are the effects of using different metrics to measure distance across the dendritic field. Different metrics can minimize or favor radial or angular distances. Third, tree-building is affected by the branching factor, bf. The branching factor weights the current extent of a dendrite when considering the next input to link. How does bf impact the above capability?

7.3.1 Task 6

In task 6 we sought to test the tree-building algorithm's behavior on input positions created by angular and radial training. Also, we sought to test whether calculating distances between input positions using a straight-line method is preferable over a Manhattan method. The former uses the Law of Cosines to calculate the distance between two inputs; the latter simply sums the radial and angular distances.[10]

In Section 6.5.7 the distance metric d is described as using "both the angular and radial distances between input positions", but the method for calculating d is not precisely defined. Since angular and radial positions are determined independently we supposed it possible that tree-building would benefit from treating angular and radial distances as non-commensurate. Task 6 was designed to settle this question.

Task 6 reused the basic parameters of task 5 (20 input across 4 disjoint patterns) and $R_{max} = 500$ and $A_{max} = \pi R_{max}$. Under these settings the dendritic field was large enough to capture both radial and angular distinction between input positions within and between

[10]For the AD neuron, the Law of Cosines is written for input positions p_1 and p_2: $d(p_1, p_2) = \sqrt{r(p_1)^2 + r(p_2)^2 - 2r(p_1)r(p_2)\cos(a(p_1, p_2))}$ where r gives the radial position of the input and a gives the angular distance between them

pattern clusters. The AD neuron was trained once for 1,000 epochs. The tree was then built four different times using four different values for bf.

The figures of 7.17 show the results using the Law of Cosines method for calculating input distances during the tree-building algorithm. In all four builds, the algorithm captures angular and radial clustering by linking inputs on common branches. When $bf = 0.2$, green and red pattern inputs are linked to the neuron through orange's 5 input. Similar to the test examples in Section 6.5.7, a low bf causes an increase in angular inputs that stretch laterally across wide angular distances. The jump from $bf = 0.2$ to $bf = 0.4$ and from $bf = 0.4$ to $bf = 0.5$ cause two significant changes to the tree. The increase from $bf = 0.2$ to $bf = 0.4$ causes the red pattern to link directly to the S-compartment. The second increase causes the same for the green pattern. Patterns occupy completely separate dendritic trees. As bf increases less dramatic angular links, such as 2-to-3, 12-to-13 or 15-to-16, are replaced by radial links. This demonstrates how the cumulative dendritic path length at higher bf causes distal inputs to form straighter dendritic branches by making longer links with less angular displacement.

The Manhattan variant (Figure 7.18) rebuilt the dendritic tree four more times using the same values for bf as variant 1. The Manhattan method for calculating distances between positions in the dendritic field is more sensitive to cumulative path lengths. For example, the Manhattan method using $bf = 0.2$ (Figure 7.18a) resembles the Law of Cosines method using $bf = 0.4$ (Figure 7.17b). A $bf > 0.2$ with the Manhattan method produces far more S-compartment-rooted dendrites than the Law of Cosines method with the largest value for bf. Dendrites transition away from complex branches to single connection fan-like branches at lower values for bf. The degree of fanning out at each dendritic connection depends on the tightness of more radially distant connections belonging to a cluster.

From Task 6 we drew several conclusions. The Law of Cosines method seemed superior to the Manhattan method. It generated predictable variations in the dendritic tree across a

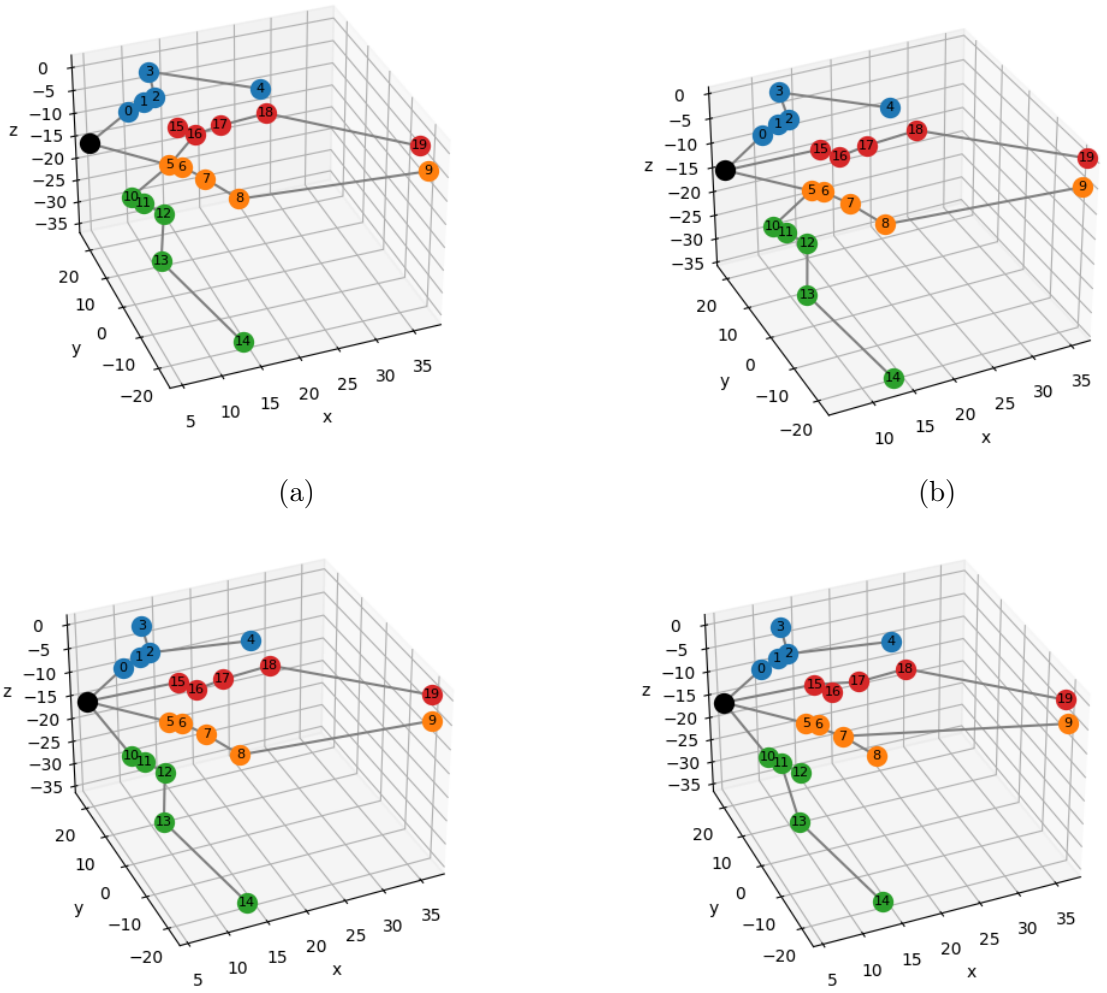


Figure 7.17: Results of the Law of Cosines variant of task 6 using (a) bf=0.2 (b) bf=0.4 (c) bf=0.6 (d) bf=0.8. The AD neuron was trained for 1,000 epochs. Numbered circles represent neural inputs. The black circle is the S-compartment. Gray lines are dendritic (internal) connections. Colors signify pattern membership.

wider of the range of values for bf. Increasing bf caused dendritic trees to favor more direct paths from the S-compartment to distant inputs. Lowering the branch factor created longer, contorted shapes. The Manhattan method, on the other hand, shifted toward an extreme state (one input per dendrite) at lower values. Second, we determined that 0.5 was a good initial choice for bf because it seemed to be a middle ground. The middle ground is characterized by dendritic trees which show signs of extension, or linking an input to the

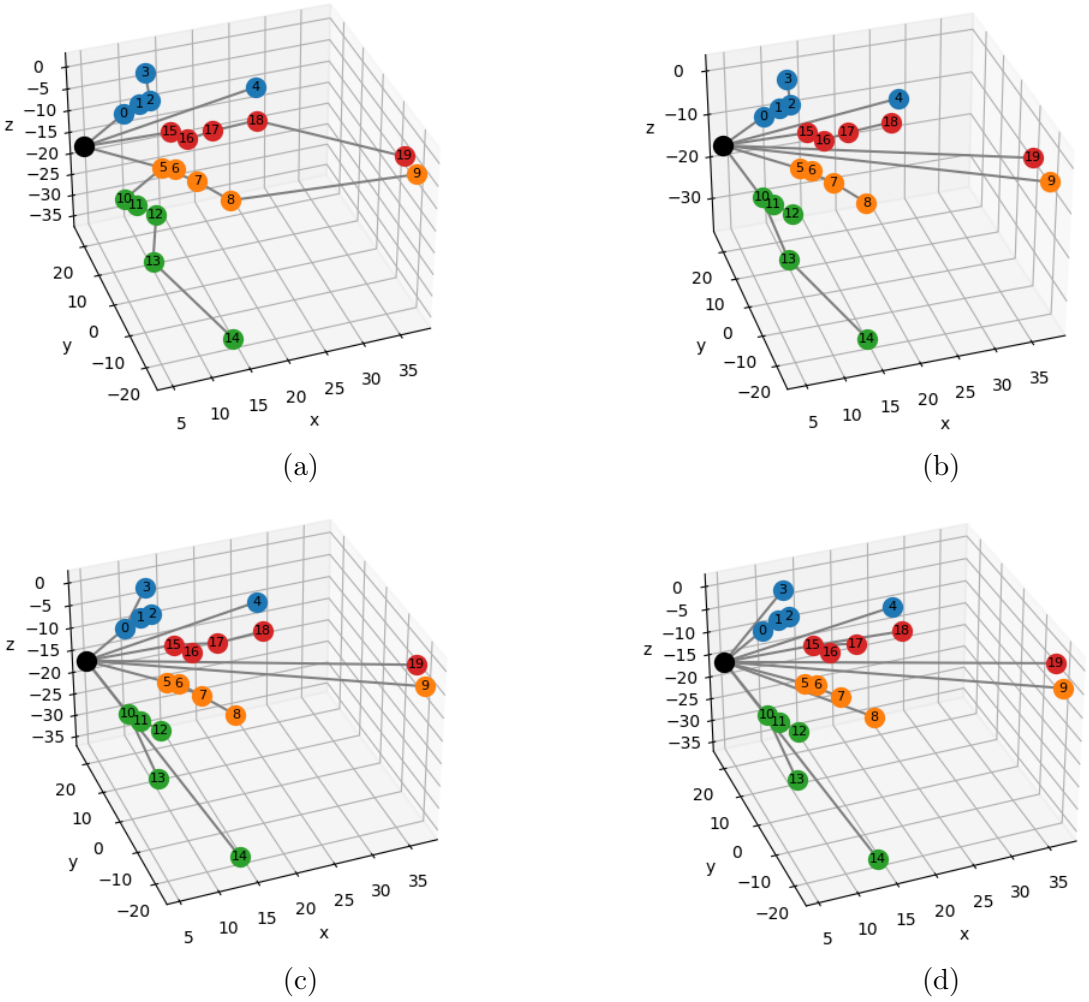


Figure 7.18: Results of the Manhattan variant of task 6 using (a) bf=0.2 (b) bf=0.4 (c) bf=0.6 (d) bf=0.8. The AD neuron was trained for 1,000 epochs. Numbered circles represent inputs. The black circle is the S-compartment. Gray lines are dendritic (internal) connections. Colors signify pattern membership.

end of an existing tree, and branching, or linking an input by forking an existing branch at a non-leaf input.

Task 6 used patterns with varying rates of input. This produced pattern clusters easily captured by the tree-building algorithm. Inputs were spaced out radially with no pattern's trained input positions vastly different from the others. How well does the tree-building algorithm perform in less than optimal conditions? The following task attempts to answer

this question and to demonstrate how dendritic trees contribute to the spike rate of the S-compartment.

7.3.2 Task 7

Task 7 examines how well tree-building performs when all inputs receive input at the same rate. As demonstrated in earlier examples, inputs that receive signals at the same rate will have roughly the same radius. We wished to know whether the tree-building algorithm would group clusters of inputs with near identical radii onto one dendritic tree or across several due angular distance. More important, task 7 was the first task in which enough of the training algorithm was active to examine the impact of the trained dendritic tree on the behavior of the AD neuron (i.e., neural output rate).

An AD neuron was initialized with 40 inputs to input sources with identical spike rates (50 hz). The input sources were grouped into 4 disjoint sets. Each set formed a pattern such that only inputs belonging to the same pattern were active on a given run of the task. Task 7 consisted of 1,000 epochs of 100 runs. Each run lasted for 1,000 time steps. A random mask with a spike rate of 10 Hz was used with a strength of -0.1 (see Section 6.6.1 for the impact of a random mask's strength). The mask was active for 10,000 time steps after each epoch. The dendritic tree was rebuilt before each epoch. The maximum compartmental length was set equal to 20. Using previous data on radial positions for a given input rate, we calculated that a compartment length of 20 would be large enough to compartmentalize all the inputs belonging to a single pattern so long as they all belong to the same dendritic tree; however, a compartment length of 20 is too small to contain all the inputs of multiple patterns. From previous tests (see Section 5.7), we expected that the AD neuron could best learn to differentiate between patterns if patterns did not share compartments and compartments occupied different branches. The task was repeated 3 times using different values for bf.

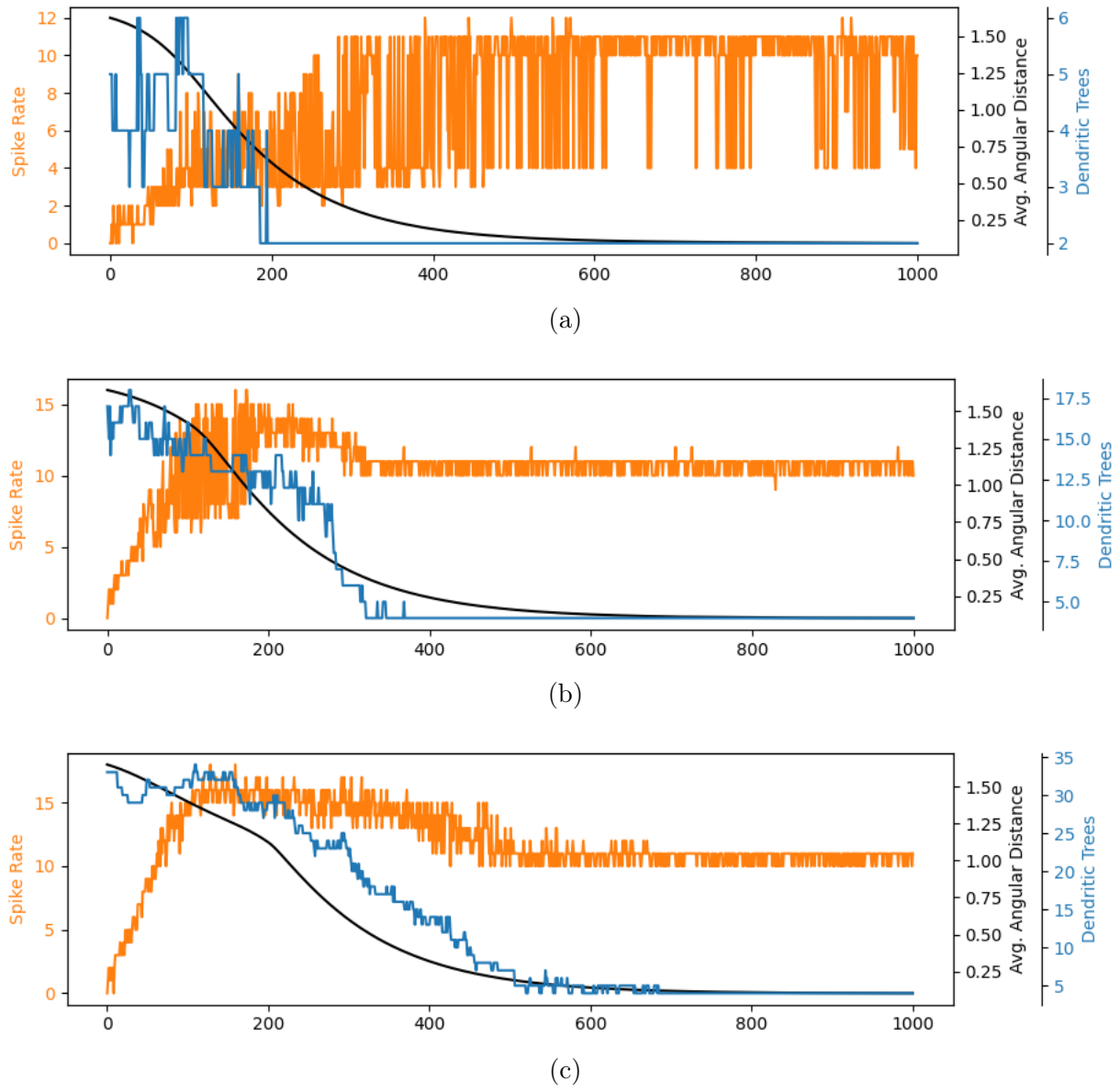


Figure 7.19: Task 7: how the number of dendritic trees impacts the AD neuron's output spike rate using three different values for the branching factor, bf. Y-axis is in epochs. (a) bf=0.2. (b) bf=0.5. (c) bf=0.8. Black lines show the average angular distance between inputs belonging to a pattern. Blue lines show the number of dendritic trees rooted at the S-compartment. Orange line shows the output spike rate of the S-compartment.

Figure 7.19 shows the typical change in the AD neuron's shape and output over the course of 1,000 epochs for three different branching factors. The clustering of inputs along pattern lines (black lines) is similar for all three. Angular positions fall at roughly the same rate to a stable average distance of approximately 0.2 radians. With respect to the number of dendritic trees (blue lines), there is likewise a reduction over time, but there are important differences. Smaller branching factors cause the AD neuron's initial state to start with fewer dendrites. Dendrites are allowed to be longer and link more inputs. This also results in fewer final dendrites: just 2. Increasing the branching factor to 0.5 (figure 7.19b) increases the number of starting and ending dendrites. Also, it takes nearly twice the number of epochs for the final number to be reached (200 vs 400). Increasing it further to 0.8 (figure 7.19c) causes the initial AD neuron to have almost as many dendrites as inputs, but, it does not raise the number of final dendrites. Again, the training required to reach a final, steady number of dendrites is lengthened to around 700 epochs.

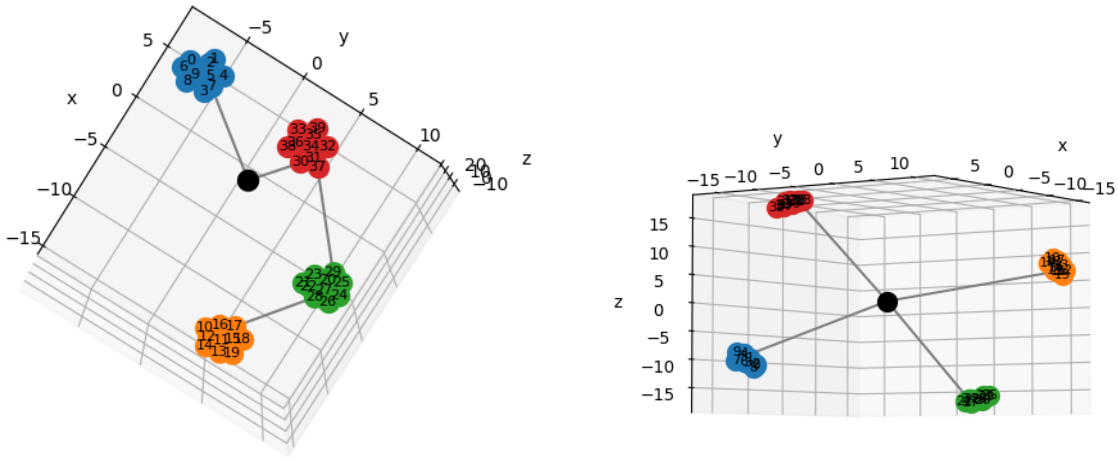
Not only does the branching factor impact the speed and result of forming dendrites, it determines the volatility of the process. Lower values for bf cause larger jumps in the number of dendrites between each epoch. For example, in the first 100 epochs, the number of dendrites in 7.19a range from 3 to 12. Compared to 7.19c, the population of dendrites is between 30 and 35.

Higher values for bf are more sensitive to angular distance. This can be seen in how closely the number of dendrites shadows the average angular distance of pattern clusters. Specifically, for larger branching factors, the number of dendrites follows the decrease in angular distance (blue above black). For smaller branching values, the fall in the number of dendrites precedes the fall in angular distance (blue below black). In other words, bf strongly determines when angular distances between input positions contain enough information about their final state for the tree-building algorithm to produce the final number of dendrites.

The AD neuron's spike rate is strongly affected by the number of dendrites. Output rates begin near zero. This is due to inputs being scattered across the entire dendritic field. With a high bf, each input is its own compartment. With a low bf, patterns are distributed across across multiple dendrites and compartments. These initial conditions result in compartments which are too small or weak to generate output of their own (and thereby generate output in the S-compartment). As inputs cluster according to pattern membership, the output rate rises. It stabilizes when the dendritic trees are comprised of inputs from a single pattern. When bf is in the range 0.5 to 0.8, the AD neuron's output is stable at 10 or 11 spikes per run. It is stable because, regardless which inputs activate for a given pattern, the distance to the S-compartment is the same.

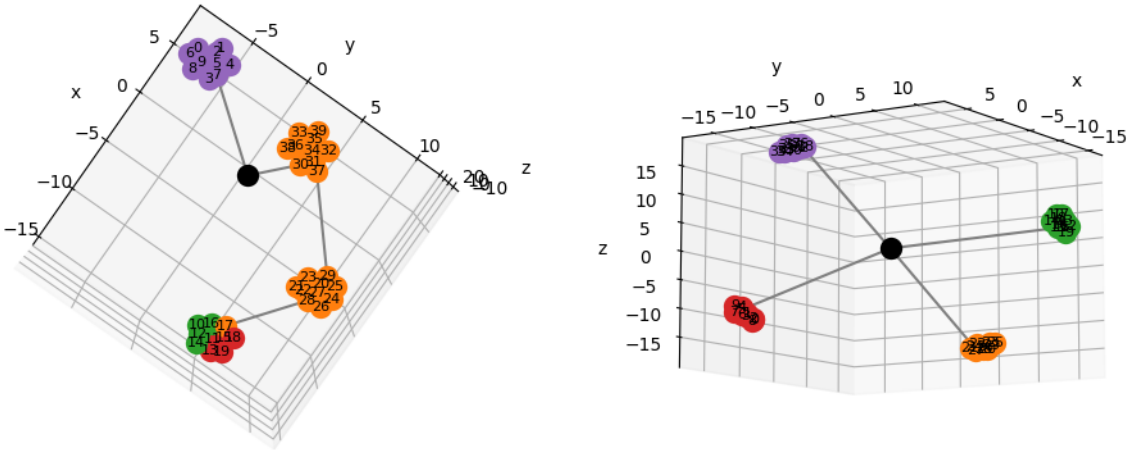
On the other hand, a low bf causes a bimodal rate of 3 or 10 spikes per run. Bimodal output is caused by dendrites linking inputs from more than one pattern in more than one compartment. Figure 7.20 shows the final shape of the AD neuron for task 6 when bf was set to 0.2 and 0.5. Colors highlight which pattern each input belongs to. A bf of 0.2 (Figure 7.20a) causes one dendritic tree to link the inputs of three different patterns (red, green and orange) to the tree. A larger bf increases the impact of overall dendrite length when linking new inputs which causes (Figure 7.20b) each pattern's inputs to be linked to a different tree.

Long dendritic trees are more likely to consist of multiple compartments. Multiple compartments as we will demonstrate in a later task impacts an input signal's chance of generating neural output. Figure 7.21 shows the same AD neurons as figure 7.20 except that colors show compartment membership. In 7.20a, the orange compartment holds inputs which belong to 3 different patterns. The red and green compartments divide the pattern consisting of inputs 10 through 19. Activity caused by an example of this pattern will be much weaker than the others. Input signals arrive in three different compartments. To spur output in the S-compartment, it must generate activity in three different compartments. Compare this with the compartmentalization of the AD neuron created by



(a) (b)

Figure 7.20: The final input positions and dendritic tree of the AD neuron for task 6 when bf=0.2 (left) and bf=0.5 (right). Colors depict which pattern each inputs's source belongs to.



(a) (b)

Figure 7.21: The final input positions and dendritic tree of the AD neuron for task 6 when bf=0.2 (left) and bf=0.5 (right). Colors depict which compartment each input belongs to.

a larger branching factor (7.21b). Each tree is its own compartment. And each compartment contains the inputs of one pattern.

The results of task 7 helped us understand how the branching factor impacts tree building and compartmentalization. It helped us to further hone how we choose a proper branching factor. Figure 7.19c we believe holds the key. We suspect a near optimal bf will cause the rate at which the number of dendritic trees approaches a stable range (blue lines)

to match the clustering of neural inputs (black lines). To put it another way, blue shadows black. For task 7, we knew in advance the optimal number of dendrites was four and therefore were biased to accept bf=0.5 as a good choice. It reaches 4 dendrites ahead of the angular clustering.

Task 7 suggests that once patterns are encoded in the dendritic tree they produce reliable output rates. If so, reliable output rates would help to identify whether an example belongs to a pattern, category or class. In fact, this is the method we use in later tasks to categorize examples. Even in the mismatched tree created by a branching factor of 0.2, it is still somewhat possible to distinguish between active patterns. Although we maligned it in our previous discussion, the modal neural output of the neuron depicted in figure 7.20a could be beneficial. The differing input rates could aid in determining which pattern is active. From the singular output of the one-pattern-per-tree neuron we can only detect the presence of a pattern, but not which one.

Although we have characterized the dendritic trees created by larger bf as being optimal, optimal is, of course, task dependent. This demonstrates that the branching factor can be used to optimize an AD neuron for a specific task.

We tested task 6 many times with many different values for bf. The above supposition about how to choose bf seems to be a good rule of thumb but does not guarantee the algorithm will create an AD neuron with the desired number of dendrites. Generally, bf=0.6 caused the decline in the number of trees to match the input clustering. Above 0.6, there was a chance the tree-building algorithm waffles between 4 and 5 dendritic trees. Below 0.6, there was a chance the algorithm creates less than 4 trees. For any given test, the number of trees seems dependent on initial conditions. The initial distribution of connections in the dendritic field determines where clusters form initially. Some form in close proximity to each other. Usually clusters shift away from each other because of the repelling force created by the random mask. Sometimes they do not and appear fixed in spite of the mask. We believe this is caused by countervailing forces; the sum of repelling

forces generated by the mask create a field with dead zones or sinks. It is unknown if given enough epochs such sinks will dissipate due to small shifts in input positions due to the random spike times.

Finally, in working through task 7 we realized that the random mask works best with smaller negative values (-0.1 vs. -1.0) over longer runs (10,000 vs 1,000).[11]

7.3.3 Summary of Tree-building

The tree-building part of the AD neuron training algorithm creates a minimum spanning tree rooted at the S-compartment. It uses a branching factor, bf, to weight each dendritic path when evaluating the next neural input to link to the tree.

The two tasks above demonstrate the impact tree-building has on the capability of an AD neuron. A suboptimal tree can cause the neuron to perform poorly. An optimal tree depends on the number of dendritic trees, the paths created by linking neural inputs together and compartmentalization (which is the subject of the next section).

7.4 Compartmentalization

7.4.1 Task 8

Task 7 showed that compartmentalization along with the general shape of the tree has an impact on the output of the AD neuron. To further explore this impact, we devised task 8 to test how compartment length alters AD neuron output. Task 8 consisted of a single AD neuron which received input from 100 sources through 100 neural inputs. Input sources were given label IDs ranging from 0 to 99. Input rates are shown in Figure 7.22 by ID. Input rates range from 100 Hz to 3.84 Hz. The AD neuron was trained for 500 epochs of 100 runs each. Each run lasted for 1,000 time steps. A random mask followed each epoch which lasted for 10,000 time steps and used an error value of -0.1. The AD neuron was trained with bf=0.5 and compartment length equal to 20.

[11]Here is the origin of the general rule stated back in Section 6.6.1.

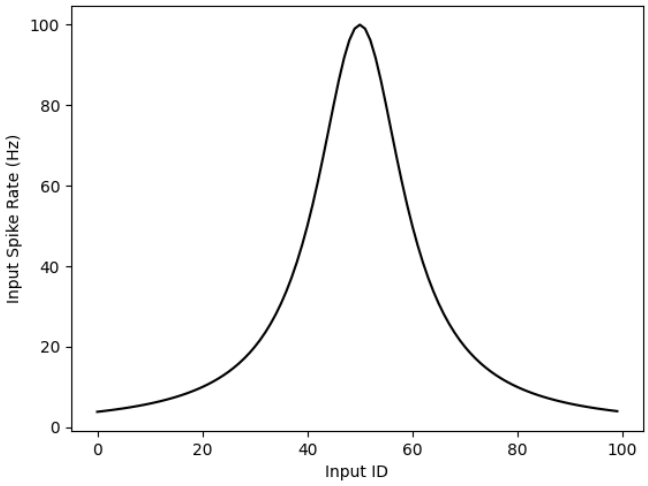


Figure 7.22: Spike rates by input source ID. Rates were produced using the function: $R_i = \frac{0.1}{(0.1(49.5-i))^2+1}$. R_i is the input rate for neuron with ID i.

The dendritic tree produced for task 8 is shown in Figure 7.23. A bf of 0.5 links all neural inputs to the same dendritic tree.[12] The dendritic tree itself has a hyperbolic shape (i.e., crescent shape) which fans out along one dimension as it extends into the field. The curved fan-shape is curious. It appears that inputs at the same radial position separate along one vector which causes the fanning out (see Figure 7.23), whereas, inputs at different radial positions separate along a vector orthogonal to the first which causes the tree to bend (see Figure 7.24). Fanning out causes distal compartments to contain fewer inputs. Proximal inputs are compact which creates fewer compartments. A compartment length of 20 creates a variety of dendritic path lengths measured by number of compartments. The longest path contains 9 compartments and the shortest just 1.

For testing, inputs were grouped into sets of 10 in order from 0 to 99, e.g., group 0 contained inputs 0 to 9, group 1 contained 10 to 19, etc. This produced 10 test example patterns. Input sources belonging to a pattern spiked at a rate of 50 Hz. Sources that did not belong to the pattern were silent (0 Hz). The neuron was exposed to each pattern, presented in random order, for 1,000 time steps. 100 tests were run using each

[12]A high bf (> 0.8) will cause distant inputs to link directly to the S-compartment.

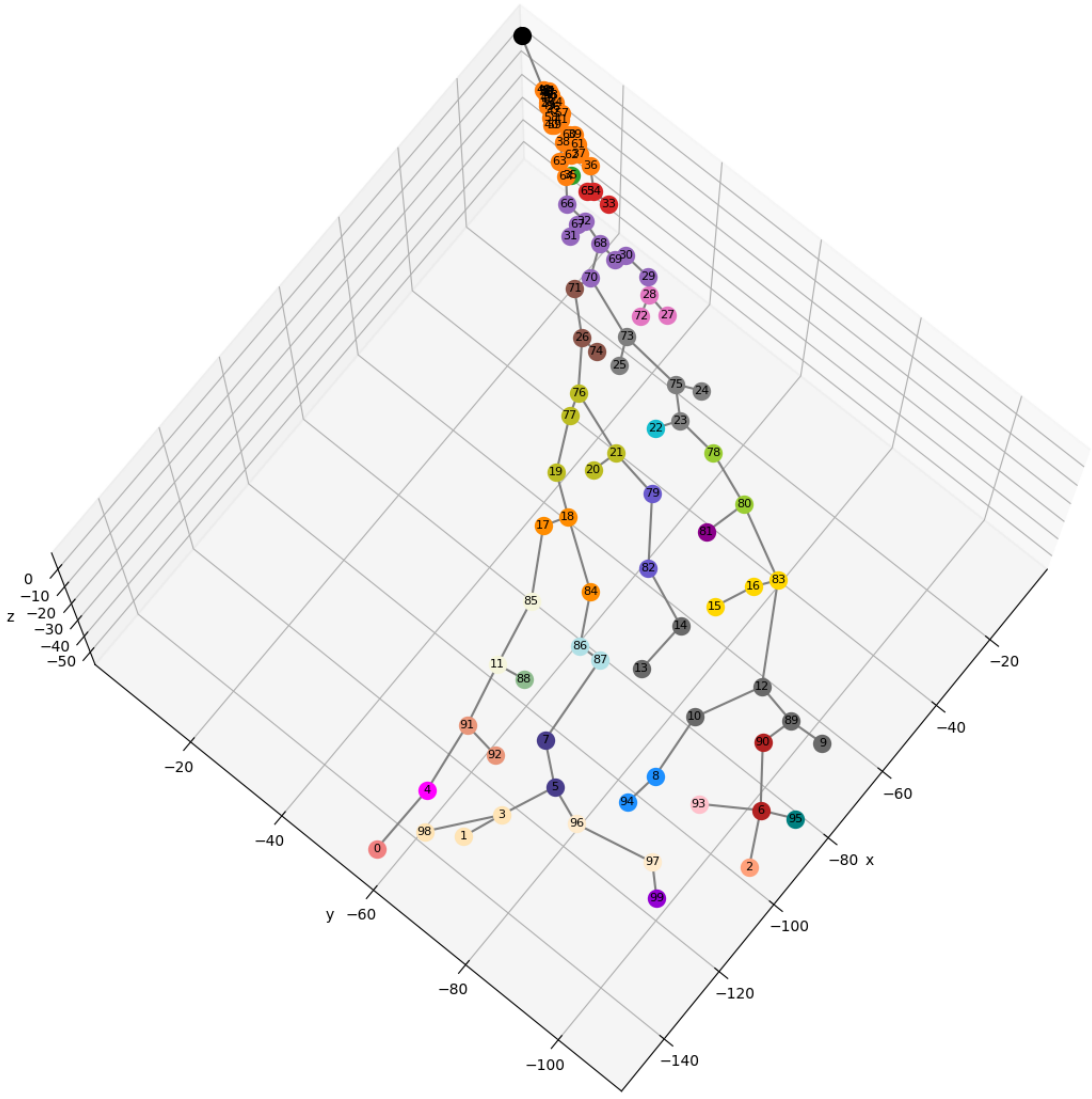


Figure 7.23: Task 8 dendritic tree. Circles are inputs. Gray lines show the shape of the dendritic tree. Colors denote compartment membership. Numbers correspond to input IDs. Compartment length: 20. bf: 0.5.

compartment length. Each pattern targeted a different depth of the tree. Using one trained AD neuron, we ran several variants which rebuilt the same AD neuron using different compartment lengths. Input positions and other parameters remained unchanged.

Using the trained neuron depicted in Figure 7.23 the tree was built using three different compartment lengths: 5, 20 and 100. The six figures of 7.25 show the resulting compartmentalization and S-compartment output. Left-hand figures depict how

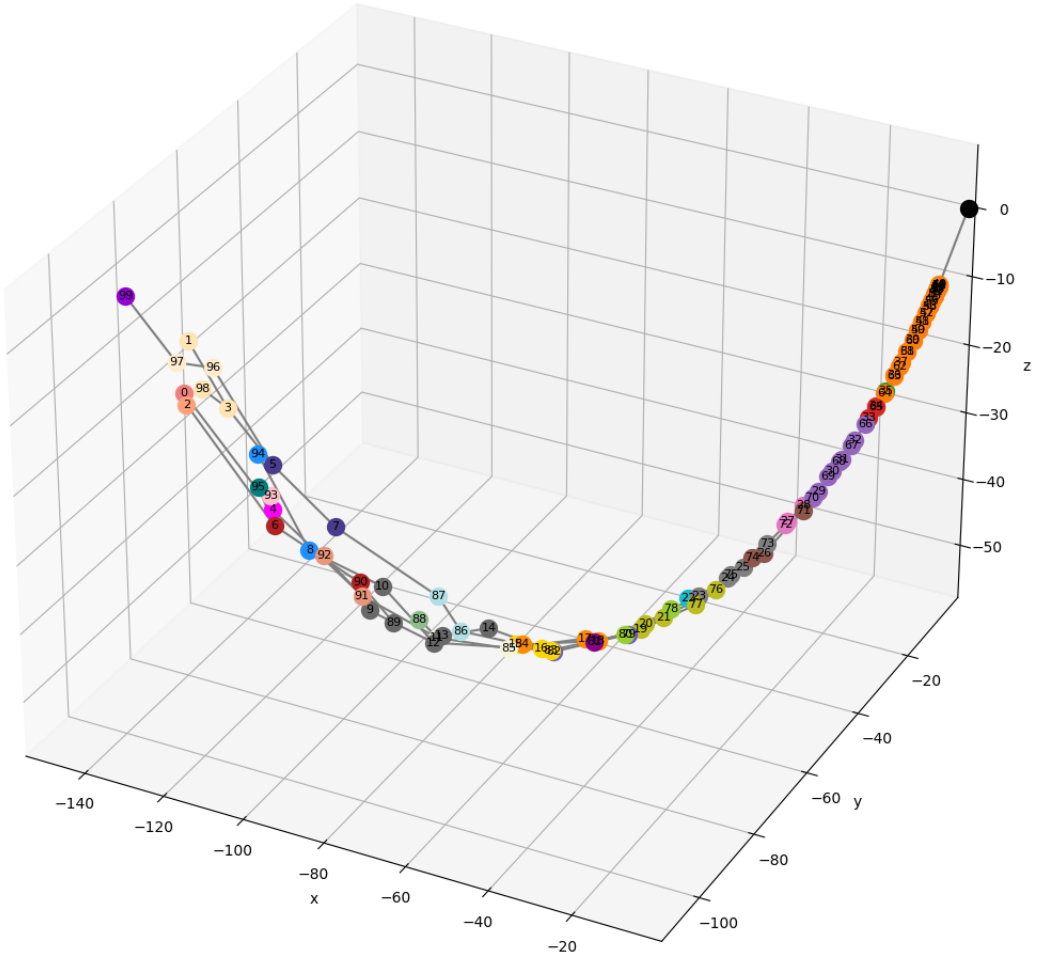
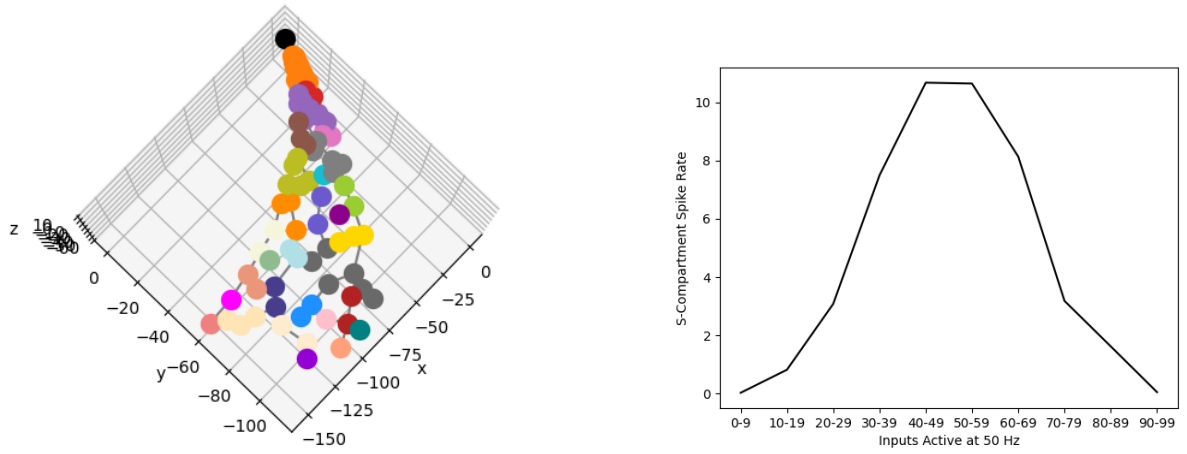


Figure 7.24: Same AD neuron as 7.23 viewed from the side.

compartment length impacts compartmentalization of the entire dendrite. The right figure shows the average neural output rate during for each pattern.

A compartment length of 5 produces S-compartment output that qualitatively best matches the input profile in Figure 7.22. All three lengths produce a similar peak output (from 9 to 11 spikes) when inputs close to the S-compartment are active. There is a subtle positive correlation between compartment length and maximum neural output. However, we have not seen a correlation between neural output from distal inputs and compartment length.

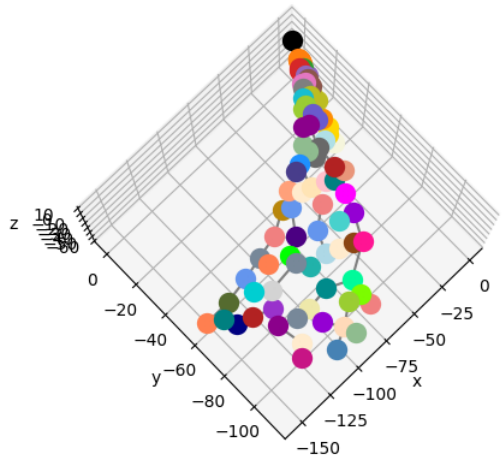
In fact, changes to compartment length have a nonlinear effect on neural output, especially when neural input arrives deeper in the tree. Smaller compartments (i.e., those



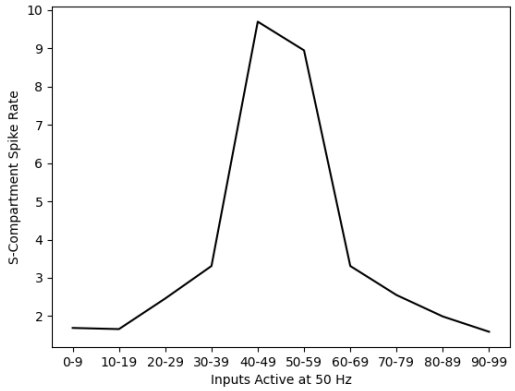
(a) (b)

Figure 7.25: Dendritic trees with compartments and S-compartment output rates for three compartment lengths. (a)(b) Compartment length: 20. Left figures: an AD neuron with the dendritic tree built using three different compartment lengths. Colored dots are inputs. Colors indicate compartment membership (note: colors are drawn from a list so there are some repeating colors). Right figures: S-compartment output rates using the correspond dendritic tree. Input rates: 50 Hz.

with fewer inputs) require fewer active member inputs to generate a compartmental spike. The need for fewer active inputs owes to input scaling due to compartment size (see Section 9.2.12 and the final paragraph of Section 6.3). Therefore, a path of small compartments is more likely to experience a chain of spikes than a chain of larger ones. At the other end of the scale, very large compartments reduce the number of compartments distal signals must pass through to reach the S-compartment. In Figure 7.25f, inputs belonging to test patterns 0 and 9 are spread across the pink, brown, purple and gray compartments; however, the compartments themselves are small. Although patterns 0 and 9 are scattered, they can generate compartmental output. Patterns 1 and 8 are also scattered; however, a few of their inputs belong to the massive orange compartment, so their signals impinge the dendrite close to the S-compartment. Compartment membership closer to the S-compartment increases the pattern's ability to generate neural output. All inputs belonging to patterns 2 through 7 belong to the orange compartment. These patterns are all equally capable of generating maximal output.

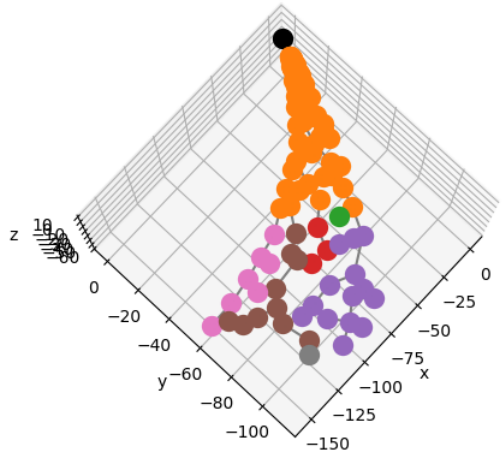


(c)

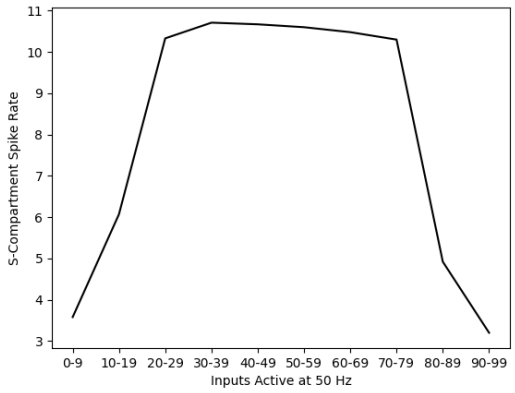


(d)

Figure 7.25: Dendritic trees with compartments and S-compartment output rates for three compartment lengths. (c)(d) Compartment length: 5. Left figures: an AD neuron with the dendritic tree built using three different compartment lengths. Colored dots are inputs. Colors indicate compartment membership (note: colors are drawn from a list so there are some repeating colors). Right figures: S-compartment output rates using the correspond dendritic tree. Input rates: 50 Hz.



(e)



(f)

Figure 7.25: Dendritic trees with compartments and S-compartment output rates for three compartment lengths. (e)(f) Compartment length: 100. Left figures: an AD neuron with the dendritic tree built using three different compartment lengths. Colored dots are inputs. Colors indicate compartment membership (note: colors are drawn from a list so there are some repeating colors). Right figures: S-compartment output rates using the correspond dendritic tree. Input rates: 50 Hz.

In figure 7.25b we see that moderately long compartments (20) prevent distally arriving input from generating S-compartment output. Patterns 0 and 9 produce no output and patterns 1 and 8 do not rise to the minimal levels produced by small or larger compartment lengths. Medium-sized compartments are large enough such that they do not spike as readily from a small number of active inputs. However, they are also small enough that the pattern size (10 inputs) we selected for this task causes inputs belonging to the same pattern to be distributed across many compartments at a certain depth in the dendritic tree. For example, in Figure 7.23, the inputs belonging to pattern 0 (inputs labeled 0 to 9) belong to 8 different compartments.

Lowering the branching factor causes a decrease in output spike rate generally. The decrease is more evident the more distally input is received in the tree. For example, lowering the branch factor to 0.2 in conjunction with a compartment length of 5 results in patterns 0, 1, 2, 7, 8 and 9 producing zero output. For longer compartments, even smaller branch factors are required to silence distal input. Small branch factors have a less pronounced effect on proximal input, reducing rates by 1 or 2 Hz. At the other end of the spectrum, large branch factors (> 0.5) can create significantly different trees, making comparisons more difficult. For example, when $bf = 0.8$ distal inputs (0-9 and 90-99) occupy their own dendritic trees. These distant inputs link directly to the S-compartment, creating short trees. Short trees have a stronger impact on the S-compartment. The result is that patterns 0 and 9 generate more neural output than patterns whose inputs have a smaller radial position.

In summary, compartment length and the compartmentalization process determine the functional length of the dendritic tree. Compartmentalization abstracts the dendritic tree and, although the underlying structure is not removed, it is de-emphasized by longer compartmental lengths. In the examples above, we demonstrated that neural output depends as much on which compartment receives input as where in the tree it is received. This implies that compartment length is important to AD neuron behavior. Currently, we

do not have a method to find an optimal length for a given problem so it falls in that category of hyper-parameters which must be found by hand. Finding compartment length and other such hyper-parameters are further discussed in Section 9.3.10.

Before moving on, we pause to note that the preceding tasks show that the first four properties of AD neuron training--angular position, radial position, tree-building and compartmentalization--together create a neuron capable of recognizing input without altering weights. This is demonstrated in Figure 7.25d in an AD neuron capable of reproducing, albeit qualitatively, the pattern embedded in its input. In other words, the shape of the AD neuron on its own contributes to learning. Normally when we think of learning in neural networks, we think of weights, and weight modification is the subject of the next section. However, shape can support learning as well. When we look at the shape of a neuron, we are looking at more than just the form of a thing. We are looking at one part of a complex function. We are looking at one part of a computational unit. This is the crux of this work and will be discussed in the final chapter.

7.5 Weight Modification

7.5.1 Task 9

Connection weights specify the importance of a connection within a compartment. Task 9 tested whether our approach to weight modification is capable of shifting S-compartment output rate toward a desired goal rate. We also wanted to test the stability of learned output rates. To answer these questions we tasked a single AD neuron to respond differently to two patterns which were equal in strength (equal in number of inputs and input rate). As a control test, we first trained an AD neuron using the two patterns without weight modification to demonstrate that without it the AD neuron does not favor one pattern or the other. Next, we ran the same task with weight training active to demonstrate its ability to bias output for one pattern.

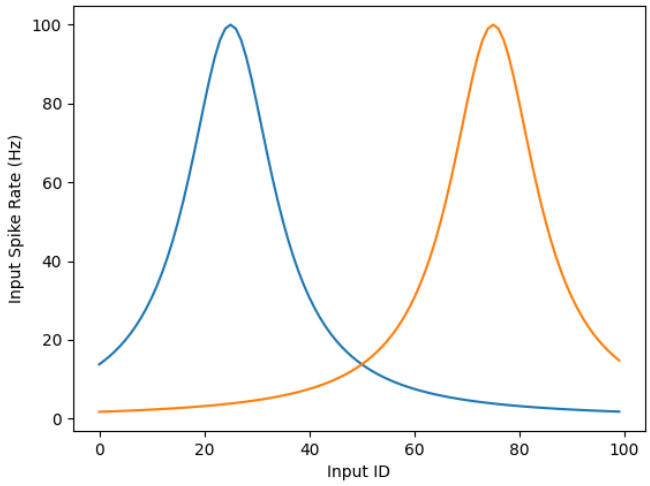


Figure 7.26: The two input patterns used in task 8. Blue: pattern A. Orange: pattern B.

Task 9 is similar to task 8. The AD neuron has 100 inputs. The compartment length is 20 and the branch factor is 0.5. Learning rates followed those given in the base parameters.

We created two patterns, A and B, using the input sources. The method for creating each input pattern is the same as in task 8 (see the caption of Figure 7.22). Both patterns had a peak rate of 100 Hz but their peaks within the input space differed. Pattern A peaked at input 25 and B at 75. Figure 7.26 shows input spike rates per input source for both patterns. Random spikes for each input were created using Poisson process input sources.

The AD neuron was trained for 1,000 epochs. Each epoch consisted of 100 training examples, 50 of each pattern. Each example ran for 1,000 time steps. The tree was rebuilt after each epoch. All aspects of the AD neuron training algorithm were enabled. When weight training was active, an error value was calculated based on the S-compartment spike rate of the previous run of the same pattern. The error rate was calculated using: $E = 1 - \frac{O_r}{T_r}$ where E is the error value, O_r is the previous output rate for the current pattern and T_r is the target rate for the current pattern. Target rates were 20 Hz for pattern A and 5 Hz for pattern B. Error values were constant across all connections for the duration of the run. The target rates used are not special. Relatively low rates were selected to avoid pushing connections toward very large weights which might have made it

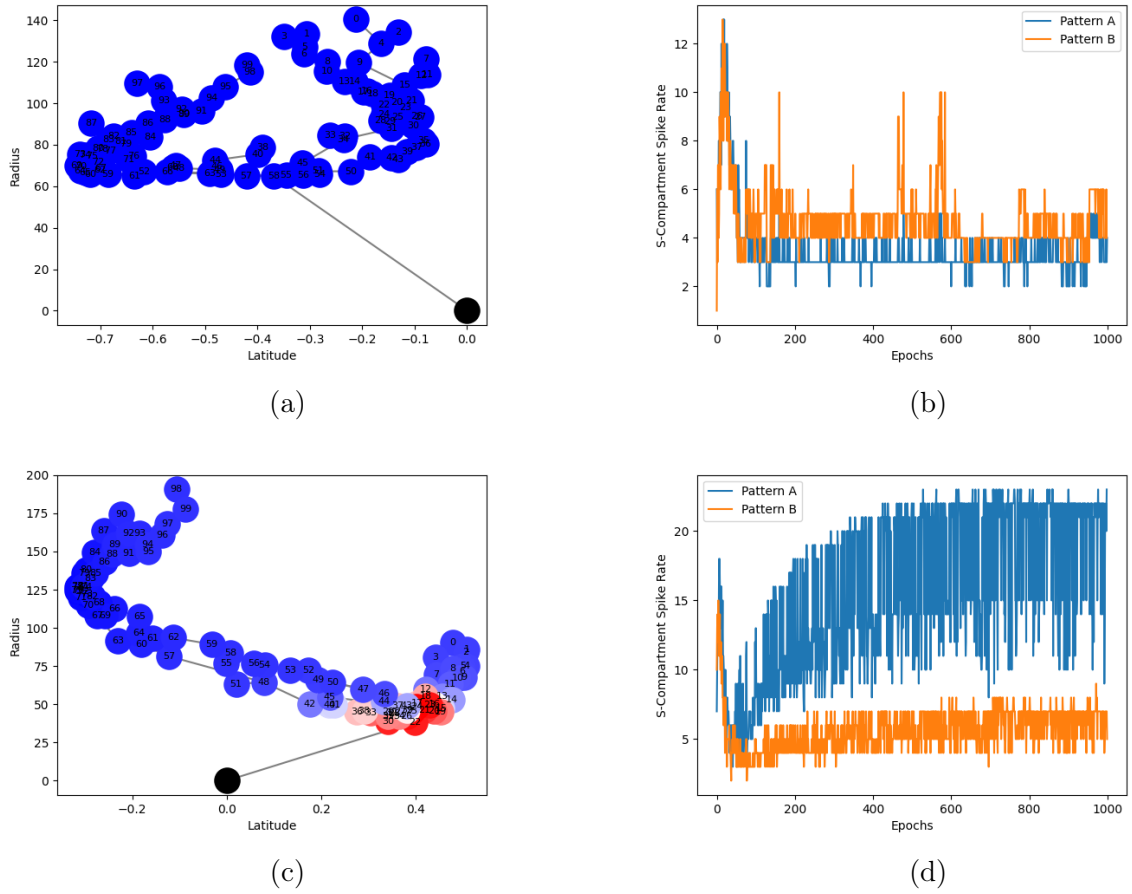


Figure 7.27: Task 9 results. Left: the dendritic trees after 1,000 epochs of training. Circles are neural input positions plotted by latitude and radius. Gray lines indicate connectivity. Warmer colors indicate stronger weights. The black circle is the S-compartment. Right: S-compartment spike rates by pattern. Spike rates are taken from a single run of each example after each epoch, i.e., the rates are not averages. Top: the AD neuron trained without weight modification. Bottom: the AD neuron trained with weight modification.

difficult to differentiate between the target rates and output saturation. And the two target rates are far enough apart to differentiate between the two.

Figure 7.27 shows the results of task 9 both without (top) and with (bottom) weight modification active. In both variants, the dendritic tree is similar. Training results in a single tree with two large arms. Inputs are ordered from one arm to the another in approximate ID order. In the variant without weight modification, the dendritic tree links input 58 to the S-compartment. Connection 58 lies roughly between the most active inputs of each pattern (i.e., inputs 25 and 75 in figure 7.26).

The tree, due to initial conditions and randomization during the training process, is slightly biased to inputs of pattern B. This is evident in Figure 7.27b in which the spike rate of pattern B (orange) is consistently 1 or 2 Hz above the output rate caused by pattern A (blue). Subsequent tests showed that training typically causes a slight bias toward one pattern or the other. This bias emerges early in training and causes the inputs of one pattern to have a shorter path to the S-compartment. While the bias does not cause runaway behavior in favor of itself, without some sort of annealing or randomization during the training process, it persists. Without knowledge of the bias, the two patterns are not distinguishable.

The variant in the bottom two figures uses weight modification. Again, the neuron has only one dendritic tree. However, the connection through which the rest of the tree is linked, #22, creates a complete bias toward pattern A.

The inputs strongly associated with pattern A have a shorter dendritic path. The increased spike rate due to weight modification causes these inputs to have a lower radial position. The tree-building algorithm then favors them with shorter paths. While overall the tree maintains the same shape, it is pulled inward on one side. The stronger weights (red circles) on the right side of Figure 7.27c cause signals more often associated with pattern A to generate more S-compartment output.

The rates in Figure 7.27d show the resulting bias toward pattern A. Quickly the output rates generated by both patterns separate and move toward the target rates (blue: 20, orange: 5). Output rates of both patterns show a higher degree of variance. Pattern A's output rate is likely more varied due to the larger weights scaling its most active inputs. The large weights amplify subtle variations in the timing of the randomly generated input spikes.

Figure 7.28 shows the final distribution of weights across all inputs. The raw weights of all inputs are initialized to 100. After 1,000 epochs of training, the stronger weights belong to inputs associated with pattern A. Those depressed below the starting raw weight are

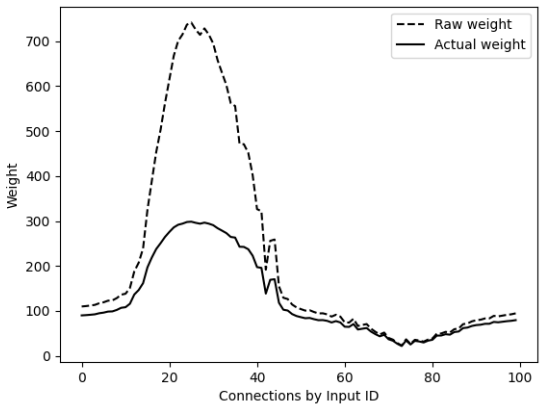


Figure 7.28: Task 9 final connection weights for the test depicted in figures 7.27c and 7.27d. The conversion from raw weight to actual weight is given in 6.5.9.

associated with pattern B. The two patterns overlap (i.e., both patterns generate activity in all input sources). Overlap causes contention between weights of different inputs. For example, a raw weight of 100 causes both patterns to generate an output spike rate near 5 Hz. This can be seen in the baseline run of task 9 in which weight training was inactive (Figure 7.27b). Once input positions stabilize, the rates of both patterns hover around 4 Hz. For the AD neuron to output a rate of 20 Hz when pattern A is active, weights, specifically those receiving spikes from inputs 10 to 40, must increase. However, increasing weights of these inputs also increases the output rate of pattern B. So pattern B's are subsequently depressed. This cycle continues until both reach an average rate equal to the target rates for each pattern (A:20 Hz, B:5 Hz).

Figure 7.28 suggests that the selected target rates for each pattern were too different. The difference in targets created two divergent groups of inputs which drove connection weights toward extremes. To test whether the target rates were causing output instability and the general inability to reach the target,[13] we lowered pattern A's target to 10 Hz. Lowering the target rate had several expected and unexpected effects.

[13]Extended runs of task 9 (5,000 epochs) did not reduce instability.

Reducing the differences in target rates improved output rate instability while maintaining separation between the rates caused by examples of each pattern. We believe the primary reason for this is a reduction in the range of weights across all inputs. When the target rates were 20 and 5, the range of actual weights in the final neuron spanned from 20 to 300. Reducing pattern A's target to 10 reduced the range to 60 to 155. This confirmed our suspicion that selecting achievable target rates is important.

Unexpectedly, the inputs with the highest weights changed. Figure 7.29a shows these differences. Unlike the initial run, the inputs with the highest weights in this variant are more active during pattern B examples. The tree shape and connectivity of the neuron's single dendritic tree is similar. Inputs active during pattern A examples are nearer the S-compartment in both their dendritic path lengths and their radial positions. However, the weights of these inputs are weaker than those activated by pattern B. What caused this change and what does this say about the weight training algorithm?

Connection weights are sensitive to input position and compartmentalization. Task 8 demonstrated the effect tree location and compartmentalization have on input signals. The inputs which receive more input during examples of pattern B are deep in the tree and their signals must pass through multiple compartments before reaching the S-compartment. In a sense, they must speak louder to be heard.

Figure 7.29d compares each input's power, or ability to be heard at the S-compartment, by subtracting normalized radii from normalized weights.[14] This is a rough adjustment since it does not use dendritic path length and ignores the contribution of angular positions to distances. The $\hat{w} - \hat{r}$-measure of input power suggests that the ability to be heard by the S-compartment for inputs 10 to 80 is not as different as weights or position alone might suggest. For example connection 65's weight is 150% the weight of input 38; however their measure of power is equal. This suggests that weights compensate, to some degree, for an input's position within the tree.

[14]Normalization is calculated by the method: $\hat{\mathbf{x}} = \frac{x}{\max(\mathbf{x}) - \min(\mathbf{x})}$.

To test whether this power calculation makes sense, we examined neural output rate when only these inputs were active. They ability to generate similar output rates suggest they have the same power. Using the same AD neuron, each in turn was given an input spike rate of 100 Hz. Input 38 consistently produced S-compartment output at 3-4 Hz while input 65 did so at 2-3 Hz. As a control we also tested input 0 whose connection weight is four-fifths the weight of input 38. It was unable to generate S-compartment output.

Testing individual pairs of inputs is by no means definitive. We examine the output generation ability of many more inputs between 10 and 80 and found a third aspect of the tree which should be included: compartmental distance, or the number of compartments on the path between it and the root. For example, the jump in weights between inputs 63 and 64 coincides with a compartment boundary. Straight-line distances and dendritic path lengths impact an input's power less than its compartmental distance.

The results of figure 7.27c suggested another variant for task 9 which might reduce variance in the output rate. For this variant, the original target rates (A:20 and B:5) were used but the branching factor was increased from 0.5 to 0.8. We theorized that an increase in the branching factor might cause the tree-building algorithm to create multiple dendritic trees which would partition the set of inputs along pattern lines. We wondered whether multiple trees would alleviate the contention created by divergent targets rates.

Increasing the branching factor did cause the dendritic neuron to consistently develop multiple trees, each roughly taking half the inputs, with the split falling somewhere between inputs 45 to 55. Increasing the branching factor did reduce the overall volatility of output spike rates caused by pattern A (pattern B's remained the same). However, actual weights were more extreme (ranging from 0 to 330) and the overall output spike rate generated by pattern A was lower. Pattern A's output rate was also slower to rise above pattern B's.

Using the branch factor to place patterns on distinct dendritic trees did not reduce weight contention. In fact multiple trees consistently made the problem worse. Separate paths allow the most active inputs of each pattern to generate neural output in isolation.

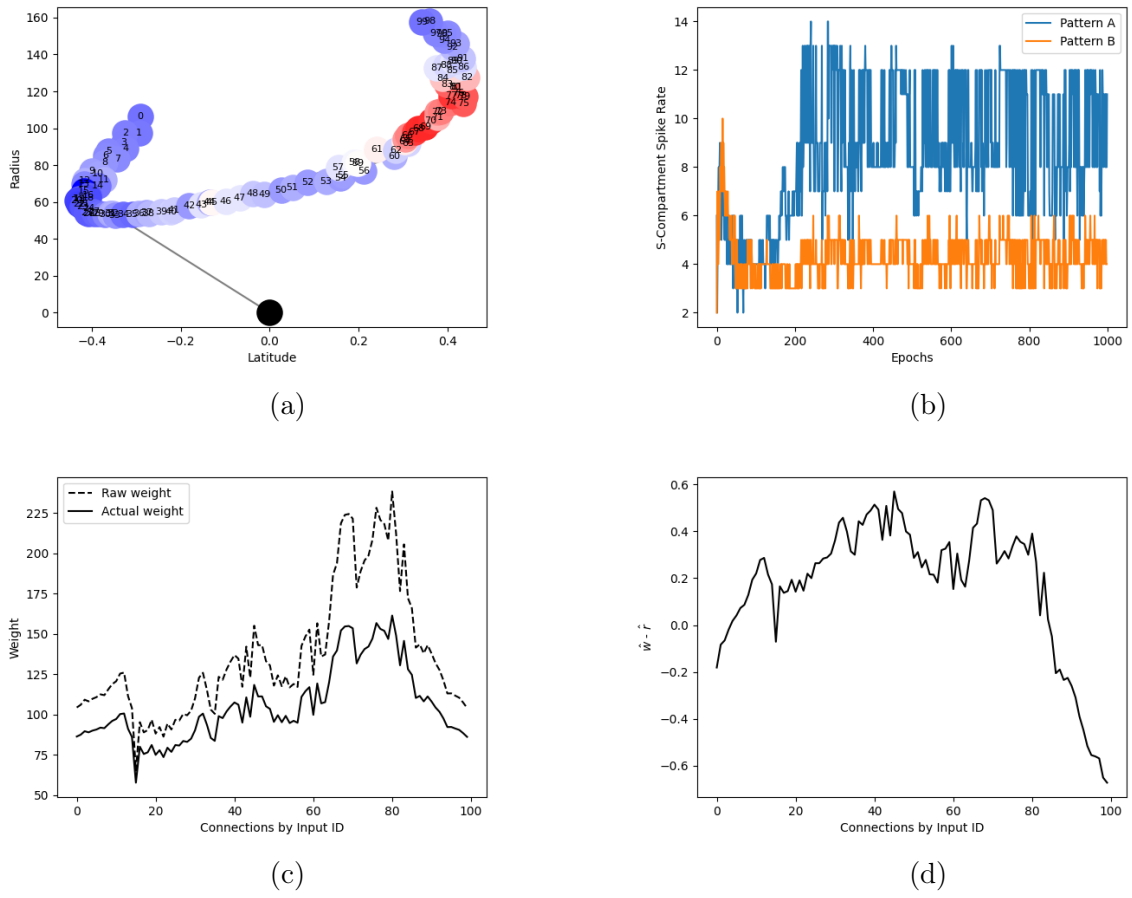


Figure 7.29: Results of task 9 variant where pattern A's target rate is 10 Hz. All other parameters are unchanged. (a) dendritic tree: warm colors are stronger weights. (b) output rate generated by both patterns. (c) raw and actual weights. (d) Normalized connection weights, $\hat{w}$ minus normalized connection radii, $\hat{r}$.

Each pattern triggers fewer spikes in compartments which contain inputs whose rates are biased toward the other pattern. Without this mixing in the middle (inputs 40-50), the transition from strong to weak connection weights is even more extreme that in Figure 7.28.

Task 9 showed that the AD neuron training is capable of matching S-compartment output to multiple target rates. Different rates signal the presence of different patterns encoded in the dendritic tree. Increased weights can empower distal connections allowing their signals to be heard far from their position in the dendritic tree. Weights, however, can mitigate the effects of compartmentalization, equalizing the power individual connections have over neural output. Matching output rates is sensitive to parameter selection and the

type of input received by the neuron. In the end, we felt that task 9 was too complex to further explore the effects of parameter selection on weight training. Task 10 uses a simpler variant to test weight modification using different compartment lengths, maximum weights and branching factors which seem to have a strong effect on weight modification.

7.5.2 Task 10

Task 9 raised questions about how the maximum weight parameter and compartment length affected weight modification and the ability to match the output rate to a target rate. Task 10 was designed to test how sensitive output rates were to poorly chosen maximum weights and to explore how compartment length impacted both weights and output rates.

Task 10 reduced the input count to 20 to create a similar but simpler version of task 9. Input was also simplified to a single pattern of varying rates shown in figure 7.30. A single pattern removed the contention in weight modification caused by multiple patterns. The simpler task also reduced the effects of a complex tree on neural output.

Task 10 used a baseline, best parameter set garnered from previous tasks. The target output rate used to create error values was 10 Hz. Maximum weight was set to 500. Compartment length was set to 20. The branch factor was set to 0.5. The task lasted for 1,000 epochs or 100 runs each. Each run lasted for 1,000 time steps. The dendritic tree was rebuilt after each epoch. All aspects of the AD neuron training algorithm were enabled.

Figure 7.31 demonstrates typical results of task 10 using the baseline configuration. Figure 7.31c shows that neural output reached its target around epoch 600. The rate is mostly stable with occasional $\pm 1Hz$ fluctuations. Again, rates climb from zero as neural inputs encode the pattern through positioning and compartmentalization. The final state of the tree is shown in Figure 7.31a. The most active inputs have clustered into two compartments (orange and green). The dendrite consists of a single tree with two branches at the midpoint (input 12). Figure 7.31b shows the final raw and actual weights. The bulk

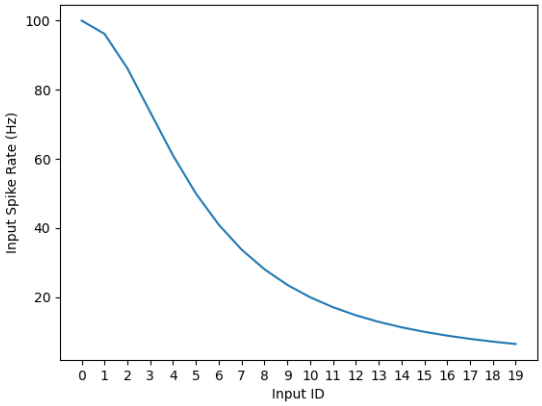


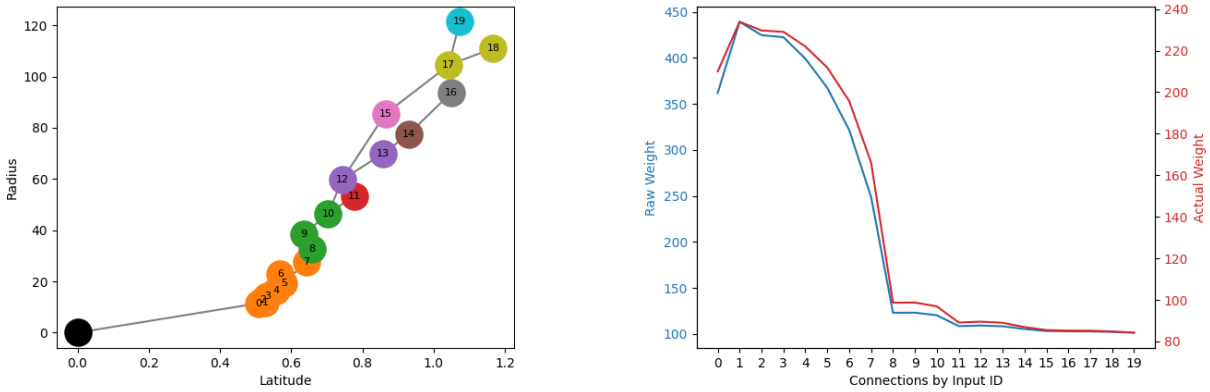
Figure 7.30: Input rates by input ID

of elevated weights all belonged to the proximal orange compartment (input IDs 0-7). This suggested that distal compartments, due to the low rate of input received by their connections, played a small role in achieving the target rate. Compartmentalization effectively silenced these inputs.

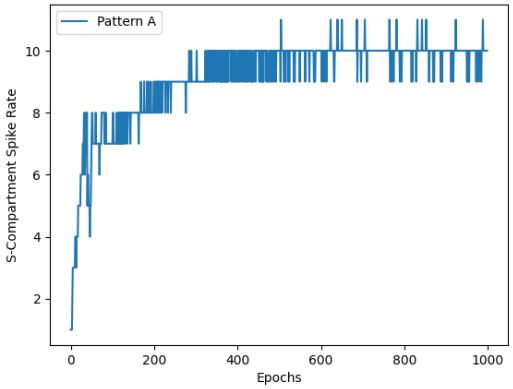
Task 10 was repeated with different compartment lengths. Figures 7.32c show results typical when compartment length is extended to 40. The size of the proximal compartment (orange) increases by 2 to 4 neural inputs. The change had a negligible positive effect on the neuron to reach its target rate. On average, rates stabilize 50 epochs earlier.[15] More significantly, the additional inputs in the proximal compartment resulted in more connections with elevated weights (Figure 7.32b). Expectedly, having more connections with weights caused a reduction in weight values for individual connections. The peak raw weight fell from approximately 440 to 350.

Figure 7.33 shows the results using a compartment length of 5. Neural output was not able to reach the target output after 1,000 epochs. Actual and raw weights are significantly higher than those of the neuron with a longer compartment length. More training does result in the target being reached, but to do so neural weights must reach extreme values.

[15]Our method for calculating output rate stability is simply to log when 50% of the runs within a window of runs (e.g., 20) match the target.



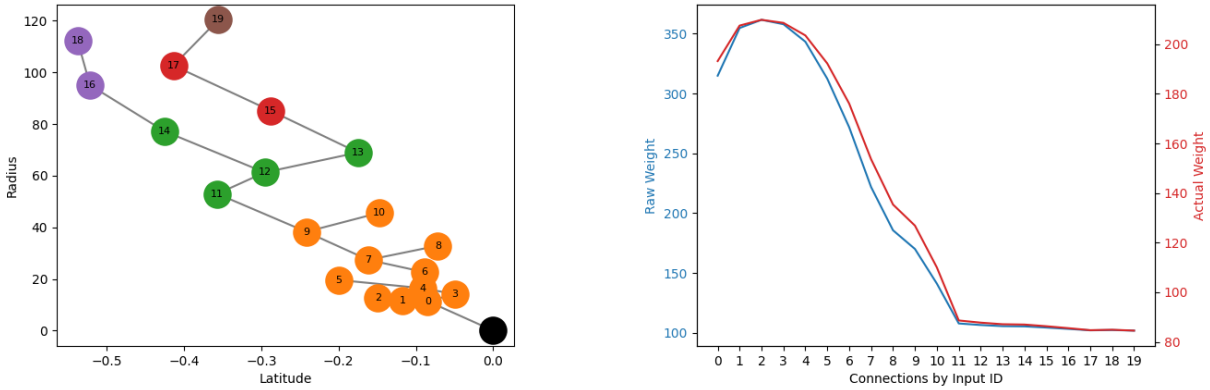
(a) (b)



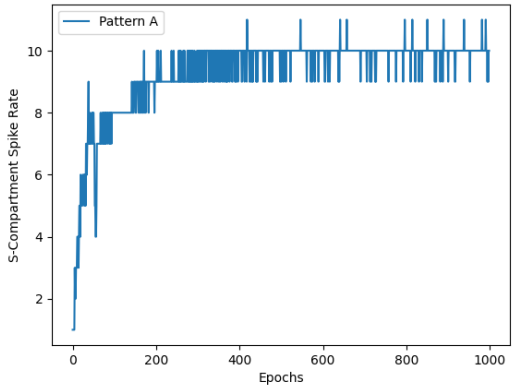
(c)

Figure 7.31: Results of task 10 using a maximum weight of 500 and a compartment length of 20. (a) The dendritic tree with input positions graphed by latitude and radius. Colors indicate compartment membership. (b) The final raw and actual weights of each input. (c) The output rate of the S-compartment.

The next variant increased compartment length to 100. Figure 7.34 shows the results. All inputs but three are included in the proximal compartment. Trained weights mimic the input rates from figure 7.30. Curiously, across all variants of task 10, the connections bound to inputs 0 and 1 finish with smaller weights than those bound to inputs 2 and 3. While we have not investigated the phenomenon for this task, we suspect the lower weights are due to the higher rate of input. A higher input rate (100 Hz) means they can generate compartmental output with small weights. This suspicion is based on our notion of input power ($\hat{r} - \hat{w}$) in the previous task.



(a) (b)

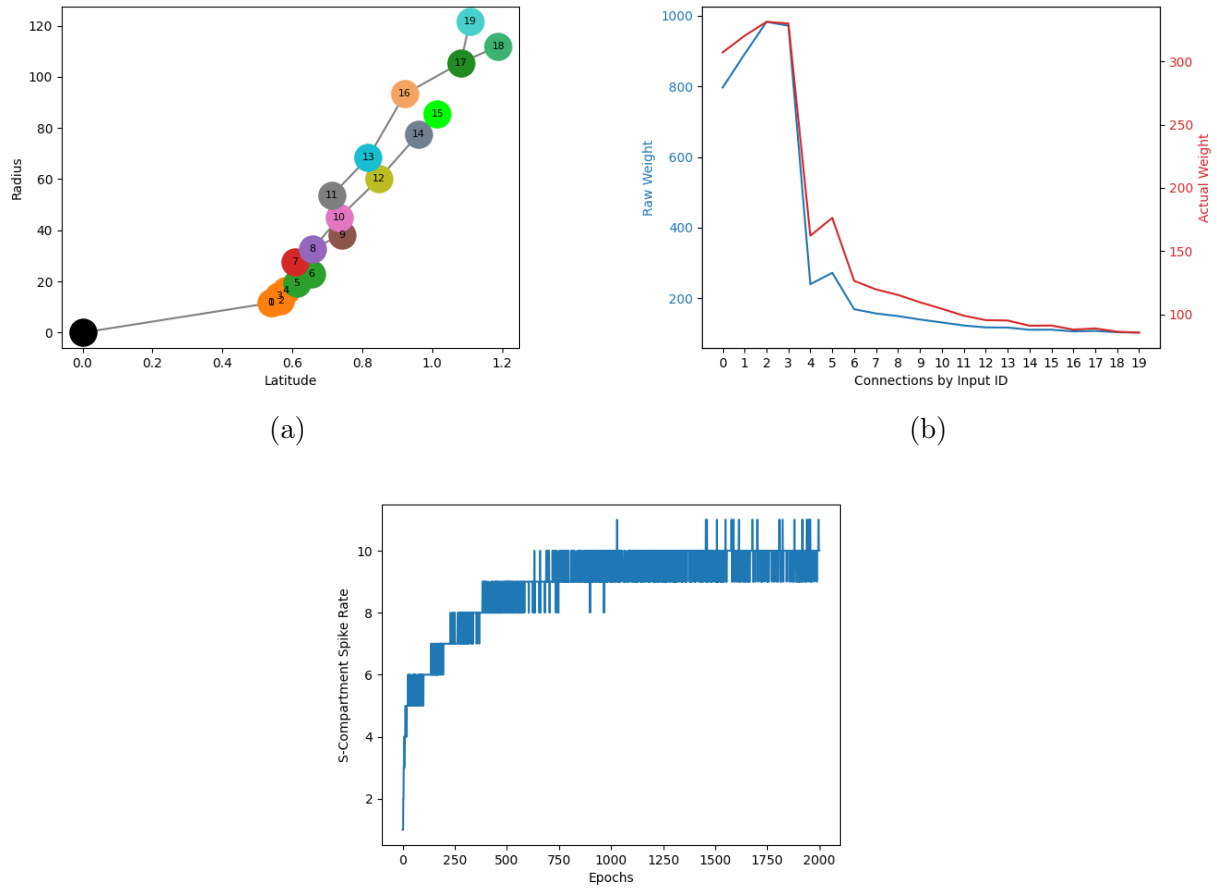


(c)

Figure 7.32: Results of task 10 using a maximum weight of 500 and a compartment length of 40. (a) The dendritic tree with input positions graphed by latitude and radius. Colors indicate compartment membership. (b) The final raw and actual weights of each input. (c) The output rate of the S-compartment.

The longer compartment, however, does not improve the speed at which the neuron trains to its target output rate. In fact, the ±1 Hz fluctuations slightly increased.

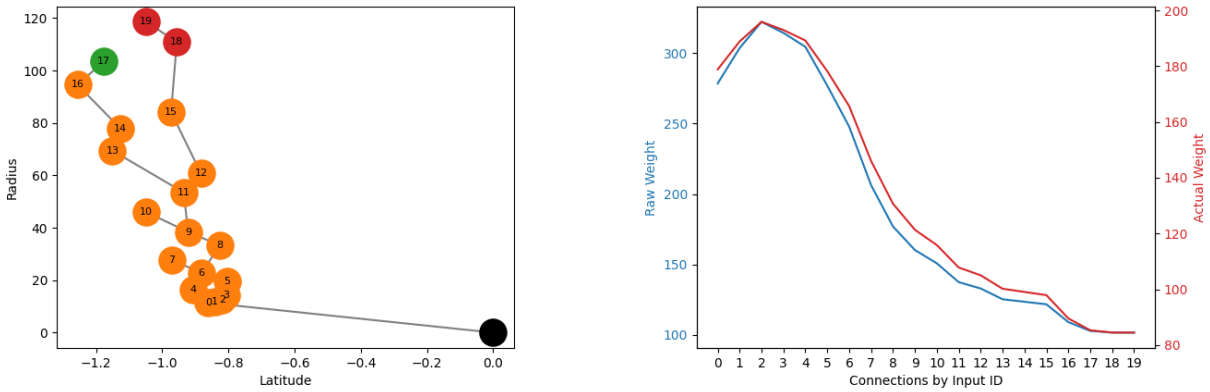
We repeated task 10 several more times using a compartment length of 40 but varying the maximum actual weight. We tried a maximum weight of 100 and 250. 100 was insufficient. Output rates did not exceed 7 Hz and the raw weight reached 1,500. A max weight of 250 did allow the neuron to reach an output rate of 10 Hz; however it struggled to maintain it. Results looked similar to figure 7.33c.



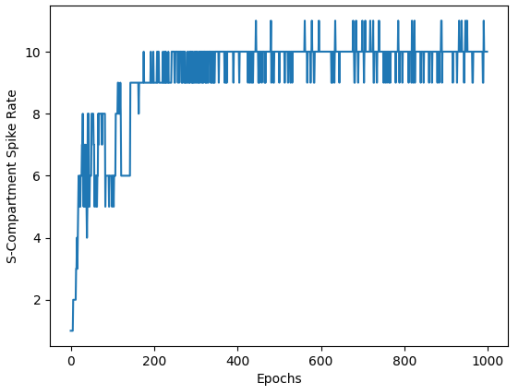
(a) (b) (c)

Figure 7.33: Results of task 10 using a maximum weight of 500 and a compartment length of 5. (a) The dendritic tree with input positions graphed by latitude and radius. Colors indicate compartment membership. (b) The final raw and actual weights of each input. (c) The output rate of the S-compartment.

Task 10 again demonstrated the AD neuron's ability to train for a specific output rate. The maximum weight parameter plays a major role in limiting the range of target output rates a neuron can effectively achieve. The maximum weight parameter must be large enough for the neuron to easily reach the target rate. Because of this, it is better to overestimate the maximum weight needed than underestimate it. A maximum weight which is too small will cause the raw weight to grow endlessly. The raw weight can also signal if the difference between multiple target weights is too large (task 9). The contention they generate in the weight training algorithm will cause weights to continually diverge.



(a) (b)



(c)

Figure 7.34: Results of task 10 using a maximum weight of 500 and a compartment length of 100. (a) The dendritic tree with input positions graphed by latitude and radius. Colors indicate compartment membership. (b) The final raw and actual weights of each input. (c) The output rate of the S-compartment.

Compartment size also plays a role by maximizing the ability of a set of connections to generate local compartment output. Oversized compartments cause fluctuations in output rate. Compartment boundaries do not distinguish between more and less active inputs. Small compartments can be beneficial because the connections in each compartment will tend to receive similar input rates. Smaller compartments can stabilize neural output rates by reducing the number of connections with short paths to the S-compartment. However, small compartments also draw many more boundaries in the dendritic tree decreasing the

ability of distal connections to be heard at the neural level. This can increase the training time to reach a target output or prevent it altogether.

Lastly, task 10 helped us to finalize the set of base parameters given in the previous chapter.

7.5.3 Summary of Weight Training

Tasks 9 and 10 demonstrated the AD neuron's ability to match neural output to different target output rates depending on that pattern was presented. Weight training is the primary mechanism for raising or lowering neural output rate from the rate to which it is predisposed by its initial conditions. Input positions, tree-building and compartmentalization do play a role but since they are not subject to an error signal they encode input without regard for its correctness.

The ability of the AD neuron to signal the presence of a specific pattern by an output rate depends on several factors. In task 9, the two patterns overlapped in the inputs they activated. If the target rates for each pattern were too different, connections contended with each other. One set pushed weights up. Increased weights drove up the output rate of the pattern with the lower target since it activated the same inputs, albeit less often. So, when active, the pattern with the lower target rate drove weights back down. While divergent weights can reach target rates, they create instability. Weight values become so extreme that subtle variations in input are amplified. In the worst case, target rates are impossible to reach causing an unending increase or decrease (or both).

Another important factor for output stability is the maximum weight. From Section 6.5.9, the purpose of the maximum weight is to prevent runaway behavior by providing an upper and lower bound on the actual weight. If the maximum weight is too low for a selected target rate, the raw weight will continue to increase but the actual weight will not. The S-compartment's output rate (barring other changes to the neuron) will never reach the target. On the other hand, if the maximum weight is too high, the output rate might

not stabilize due to the impact of small raw weight changes having too large an impact on the actual weight.

Task 9 demonstrated that target rates alter the shape of the dendritic tree. Inputs which were more frequently active during pattern A were closer to the S-compartment, both radially and along the dendritic path.

7.6 Training Summary

This chapter discussed ten tasks with many different variants which together demonstrate how the AD neuron training algorithms were tested. The tasks of this chapter represent just a few of the many tasks run to bring the work this far. Tasks were excluded primarily because they pertained to earlier, incomplete versions of the AD neuron. Others were excluded because they did not contribute any new information. The set of tasks included here demonstrate our approach to testing and together describe the viability of our approach to trainable artificial dendrites.

Tasks were chosen or designed to test the different mechanism in as much isolation as possible. Four of the five algorithms--angular position, radial position, tree-building and compartmentalization--were developed and tested initially in isolation from one another. Weight modification, because it depended on a complete dendritic neuron, came last and is the least tested of the five. Integrating the algorithms prompted modifications or additions to some algorithms. Integration was time consuming in that it required repeated passes back through earlier tasks to test whether newly added bits conflicted (or enhanced) previously added ones.

In overview, the AD neuron builds its dendrite by positioning neural inputs. These incoming connection ends are themselves connected together to form the dendritic tree. The tree is divided into compartments. Connection weights are modified based on their contribution to compartmental activity.

For each aspect of the neuron, development and testing suggested a myriad of alternatives. Many of these are discussed in the final chapter. Choosing between alternatives often required not choosing but implementing, in a limited way, several possibilities. For example, multiple weight modification algorithms were considered until very late in the project. We chose the current one because it learned from behavior (i.e., compartment-level response to input) not captured by the other learning algorithms. At each crossroad we endeavored to select the simplest version of each method and only moved to more complex forms when the simple version was inadequate.

We do not possess a complete understanding of AD neuron behavior. The different training algorithms themselves interact nonlinearly in that changes to one produce unexpected behavior in the others. Balancing them was a difficult task. However, in their current form, they are robust in that they do not require a perfect set of parameters, as we demonstrated. The only parameter-related requirement is that at some point the S-compartment must begin emitting spikes or else radial positions will not change. Often, tasks begin without neural output. This silence persists for the first few epochs. But as the angular training algorithm begins to cluster connections and compartmentalization and tree-building begin to add them to the same dendritic tree, the S-compartment begins to spike.

Is this approach viable? We believe the contents of this chapter answer affirmatively. Training modifications capture patterns embedded in the input space. The state of the AD neuron encodes these patterns in a retrievable manner. This encoding changes neural output such that input patterns can be distinguished in it. While viable, much more work is required to make it useful. For this reason, we separated the AD neuron model from its trainable form.

Although the approach is viable, unanswered questions and issues abound. For example, tasks 9 and 10 suggest that hard boundaries for compartments are inadequate. For example, in task 9, better results can be achieved if the dendritic tree consists of two

compartments. A proximal one containing 7 or 8 connections and another containing all the rest. As to how or why compartments should expand or contract, we cannot say. Our inclination is that compartment length should be trainable. How this should be accomplished and whether it is beneficial is left for future work.

Another problem area is distal compartments. As configured, it has been difficult to see a specific role played by distal inputs. More often than not they are silenced by the chain of compartments their signals must propagate through. We suspect that our treatment of signal integration within compartments could be improved.

Also, angular position training would benefit from some manner of annealing[16] . Annealing might help prevent angular position lock by reducing final position's dependence on initial ones.

Various aspects of training, which have proven to be important to ANN training, have not been investigated, such as batch size (i.e., the number of training runs in an epoch). For our approach, batch size determined how often the dendritic tree was rebuilt. Most of our tasks used a batch size of no less than 100 and no more than 1,000. The dendritic tree and its compartments do not change until the end of an epoch. This means that by the end, the tree does not match the current state and position of the connections. Similarly, we are uncertain whether neural input positions should be altered during runs (as we have done) or only once per epoch, like tree-building and compartmentalization. Again, we leave such questions to future work.

The tasks do not end here. The next chapter contains several larger, more complex tasks. These tasks test the classification performance of AD neurons.

[16]Annealing in heuristic algorithms introduces small randomizations in the state of the entity to avoid local minima.

CHAPTER 8

TRIALS

This chapter presents the results of several larger tasks which benchmark the entire suite of training algorithms working in concert. While the goal of these tasks is to benchmark performance (i.e., accuracy), they also allow us to stress test the conclusions drawn in the previous chapter concerning parameters, initialization, run-time and the feasibility of the algorithms themselves. The trainable AD neuron is complex. While parts of the combined training algorithm capture patterns embedded in the data, this does not guarantee that together they produce something useful. It is important to acknowledge that the behavior seen in simpler tasks by no means translates to successful performance in more complex tasks.

To understand the differences between the AD neuron and the point neuron we have selected two tasks. The first task is a time-series task which we expect the AD neuron to perform better than the point neuron. The second tasks involves recognizing categories from the Fashion MNIST data set. Typically, categorization is a task which point neurons (specifically large networks of point neurons) handle well.

8.1 Movement Recognition Trial

In this task, the AD neuron implementation is presented with a time series classification (TSC) problem. Time series data is data that has been ordered temporally. It gives the change of state of a system, object or process through the progression of some finite amount of time. TSC involves correctly identifying the source of a specific time series from among a set of possible origins. TSC is a difficult problem for machine learning because potentially it requires a model to identify patterns by connecting data points that are separated by a possibly varying or even undetermined span of time. We chose this trial because it represents the type of trial to which the AD neuron may be better suited.

The goal of this task was to test whether an AD neuron configuration recognizes specific types of movement and whether the training algorithm can produce such a configuration. This task tests whether AD neurons can learn to discriminate between directions of movement within a visual field. This task was inspired by neuroscience research which shows that certain neurons, due to dendritic properties, fire only or at an increased rate when input represents movement in a particular direction [121, 110].[1]

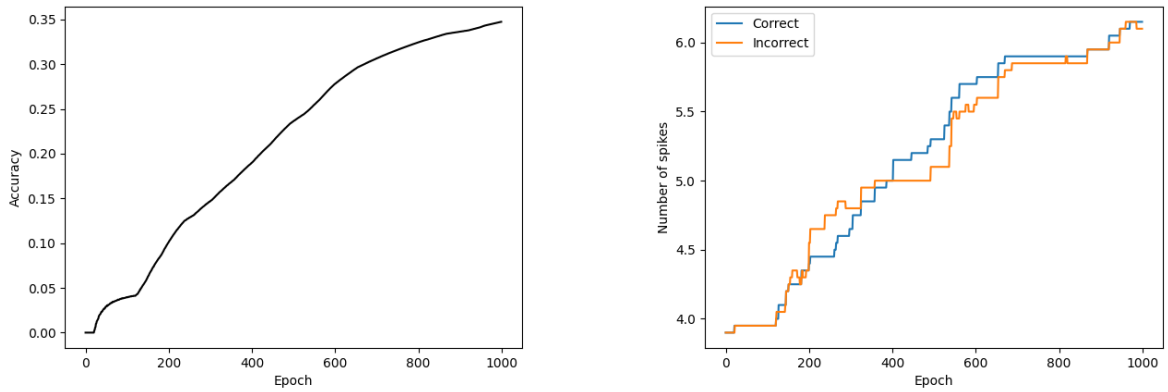
A 2D-space was divided into a 10×10 grid. A 10×10 grid was used because it posed a moderately complex problem without requiring excessive training times. The moderate size of the problem kept results interpretable.

Each input layer neuron was assigned to a different cell in the grid. Simulated objects passing through the grid caused the neurons associated with nearby cells to spike at a rate based on an activation function. The activation function is given by Equation 8.1. Figure 8.4 shows an example of the grid and two example activations.

$$\frac{0.1}{(db)^2 + 1} \tag{8.1}$$

The maximum rate (when an object is directly in the center of a cell) was 100 Hz. d is the distance between the object's location and an input neuron's focal point, and b controls the radius of the activation caused by the object. The initial position of a horizontally moving object was given with a random y-value and an x-value of 0 (i.e., they begin at a random location on the left side of the grid). The initial position a vertically moving object was given a random x-value and a y-value of 0 (top of the grid).

[1]The neurological basis for this appears to be quite complex and a complete understanding of its supporting neural architecture is lacking. That said, there are two competing ideas to explain this phenomenon: sequential synaptic activation and the interplay of excitatory and inhibitory neurons. These are not mutually exclusive since each could be relevant to one type of neuron or region. For this task, the neurological basis is mentioned here because the AD neuron is capable of ordering synapses along a common dendrite. If such ordering is sufficient, then sequential activation should produce stronger output than other forms of activation. The AD neuron can encode temporal delays in the lengths of its dendrite. With the appropriate delay between synapses, movement generated input, which would normally arrive at different times, can be made coincidental at the soma.



(a) (b)

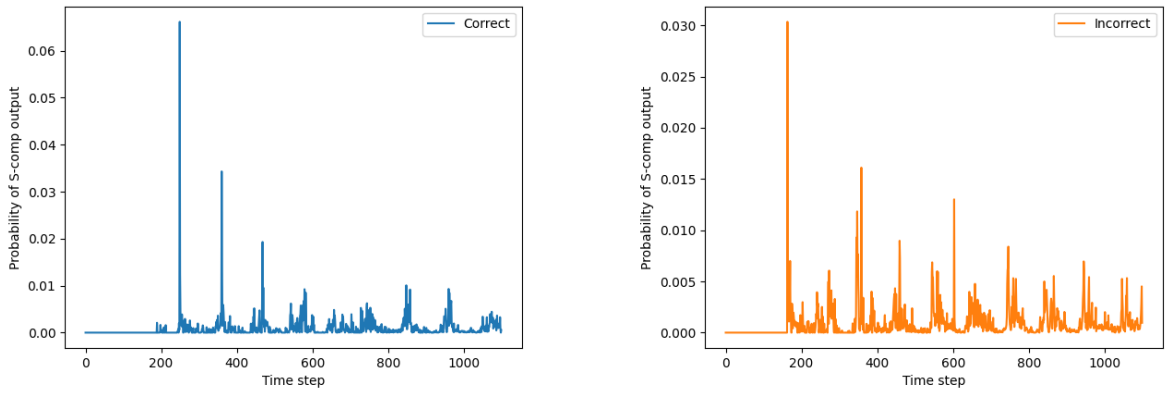
Figure 8.1: (a) The accuracy rate for the movement recognition task using a spiking point neuron. (b) S-compartment output (number of spikes) for both output neurons. Blue: output caused by movement in the correct direction. Orange: Incorrect direction.

Two variants of the trial were conducted: one using AD neurons and one using point neurons.

8.1.1 Point Neuron

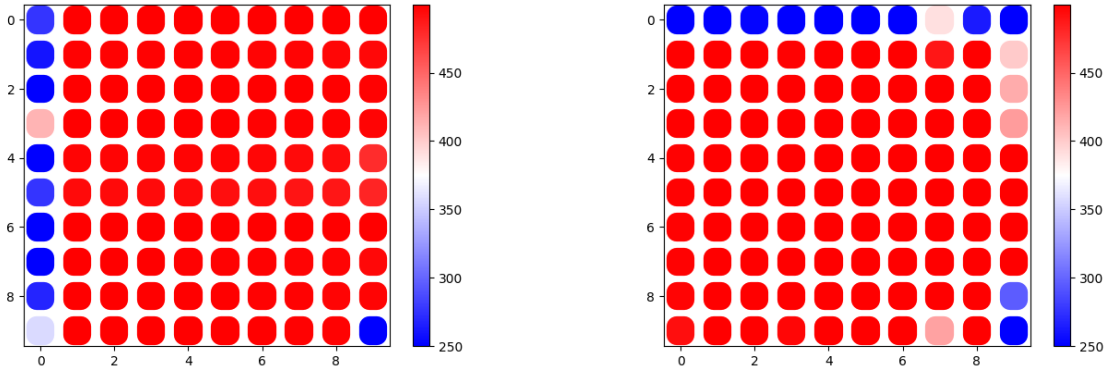
Point neurons were created from AD neurons by adding all neural inputs to the S-compartment and disabling all training except weight modification. Neural input initial positions were identical to the S-compartment, $(0, 0, 0)$. Otherwise, all parameters were unchanged. The trial itself was run under the same conditions. The figures in 8.1 and 8.2 display the same information shown above for the AD neuron. The accuracy of the point neuron variant (8.1a) is low. The direct reason for low accuracy can be seen in the similar responses output neurons have to correct and incorrect input examples (figure 8.1b). Both output neurons spike at the same rate for both types of movement.

Unlike the AD neuron, the behavior of the point neuron is the same for both types of movement. The timing of neural output for the point neuron is does not vary between movement types (figures 8.2). Neural output rises and falls as the simulated object passes near (or through) the center of each cell. Output for a correct response (blue) initiates somewhat later than the incorrect response, or slightly after the 200^{th} time step versus



(a) (b)

Figure 8.2: Probability of an S-compartment output spike per time step. Blue: Correct movement. Orange: Incorrect movement.



(a) (b)

Figure 8.3: The actual weight of each connection arranged according to their selectivity within the grid. (a) Point neuron tasked with recognizing horizontal movement, and (b) vertical movement. Actual weight is calculated according to the formula given in equation 4.3.

slightly before. The cause of this later response can be seen in the actual weights in figures 8.3.

Connection weights place both output neurons in a competition neither can win. In effect, each is playing for the other team.

The output neuron recognizing horizontal movement elevates all connections from each pattern instance (row) except for the first. The opposite is true for vertical movement. The

reason for this is that signals generated by the object passing through the cell arrive at a "cold" neuron, meaning it has not experienced any previous input and is therefore in its resting state. Signals generated by later movement reach a "warm/hot" neuron and more easily produce neural output. This causes a rise in the weights of these connections. Since in this point neuron all neural inputs belong to the same compartment, neural output drives weight modification. The side effect is that 9 out of 10 opposing movement patterns send signals through connections with elevated weights (columns in the left hand figure or rows in the right hand figure). This is the reason why incorrect input examples cause earlier output. Earlier output results in more overall output. Figure 8.1b demonstrates this. After the 1,000 epoch, incorrect output rates exceed correct ones. This trend continues.

8.1.2 AD Neuron

Each run consisted of 1,100 time steps in which simulated objects moved from either the top of the grid to the bottom or from the left side to the right. For vertical movement, the simulated object was placed outside the grid at a y-value of -200. A random x-value was selected from the range $[0, 1100]$. For the horizontal case, the simulated object was placed to the right of the grid at $x = -200$. A random y-value was chosen from $[0, 1100]$. Movement was either perfectly vertical or horizontal. In both cases, the object passed through all cells in a given column or row. Trials consisted of 1,000 epochs of 100 runs per epoch. Each run lasted for 1,100 time steps which allows the simulated object to move through the entire visual field. The dendritic tree is rebuilt and re-compartmentalized between epochs. The parameters used are given in the base parameters (Table 6.3) with several exceptions. The branching factor (bf) was set to 0.7. A higher branching factor was used for the initial trial to bias the tree-building algorithm toward straighter trees. The radius of the dendritic field was increased from 500 to 1,000 to allow for longer relationships between input and output spikes. And the maximum compartment length was set to 100. We discuss below the effects of different compartment lengths and why 100

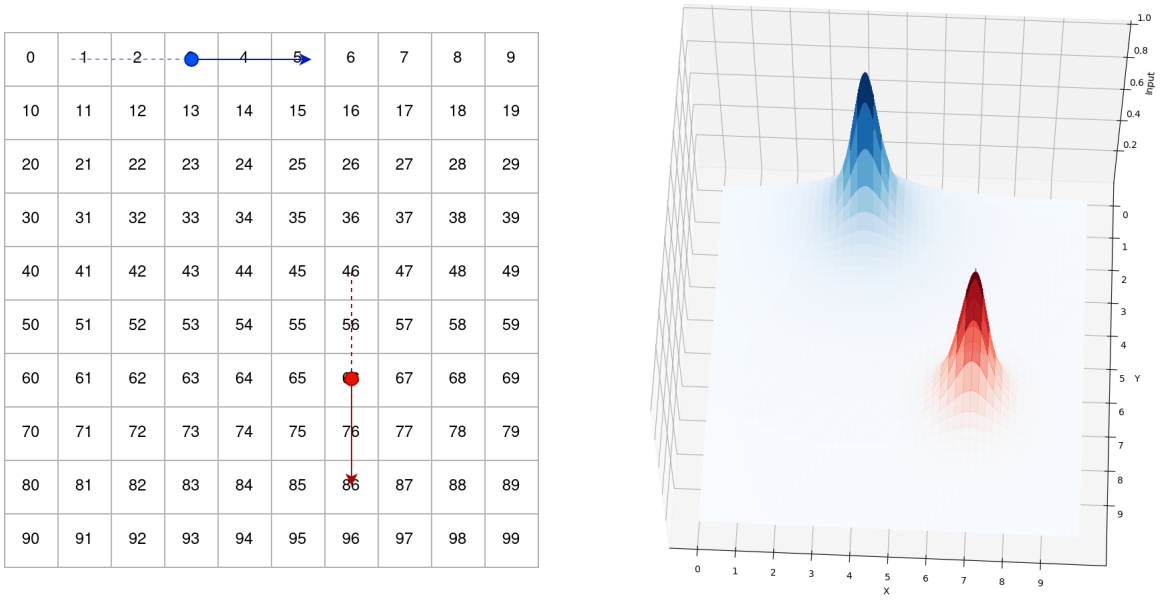


Figure 8.4: LEFT: A simulated visual field divided into a 10 grid. The field is continuous. An input layer neuron was assigned to the center of each cell. The closer an object moves to the center of a cell the stronger the input received by the associated neuron. The red and blue dot and attached arrows demonstrate how objects move across the field. The step-by-step activation caused by these two points is displayed in the right figure. RIGHT: The input from the visual field caused by the two points shown in the left figure. While this field is continuous, it is only sampled at fixed positions: the center of each cell.

was selected. Target output rates were 5 Hz for examples of an output neuron's matching category and 1 Hz for non-matching examples.

A random mask was used to separate patterns. The random mask used a Poisson process to generate input spikes at a rate of 10 Hz.

Evaluation of the AD neuron's ability to learn was based on whether there was a clear distinction between the output levels corresponding to a learned versus unlearned (or opposing) category. Although the target rate for a learned category was 5 Hz, a correct response did not require that a neuron reach this goal, only that its output rate be strictly greater than the rate of the other output neuron. Error values were calculated based on a running average of a neuron's past output rate for a given category. The error calculation used equation 8.2.

$$e_t^{ij} = \begin{cases} 1 - \frac{r_{t-1}^{ij}}{R_C} & \text{if } i = j \\ \frac{R_I}{r_{t-1}^{ij}} - 1 & \text{if } i \neq j \text{ and } r_{t-1}^{ij} \geq R_I \\ R_I - r_{t-1}^{ij} & \text{if } i \neq j \text{ and } r_{t-1}^{ij} < R_I \end{cases} \tag{8.2}$$

e_t^{ij} is the error rate of the i^{th} output neuron for input example category j at time t. r^{ij} is the output rate of the same neuron for the previous presentation of category j. R_C and R_I are the correct category and incorrect category target rates. The piece-wise equation causes examples which match a neuron's category ($i = j$) to increase its output rate toward the correct rate, R_C, and decrease it toward the incorrect, R_I, when they do not match. Additionally, if output rates exceed the correct rate (or dip below the incorrect rates), each part will flip its sign. This prevents runaway behavior in either direction. A special case (the third) is needed when an output neuron's rate is less than the incorrect target rate. In this case, $e_t^{ij} = R_I - r_{t-1}^{ij}$. This prevents divide-by-zero and prevents e_t^{ij} becoming very large when r_{t-1}^{ij} is very small.

Figures 8.5 and 8.6 show the dendritic trees of the two output AD neurons recognizing horizontal and vertical movement respectively. The grid is represented in the tree such that following the tree from a distal input to the S-compartment reproduces the simulated object's trajectory through the grid. The distance between neural inputs in the dendritic field directly corresponds to the speed at which the simulated object moves through the visual field.

The resulting input positions and trees show a distinct bias toward the type of movement each was tasked with encoding. For example, in 8.5, the neuron recognizing horizontal movement, input positions show a distal-to-proximal order corresponding to left-to-right movement through the grid. Neural inputs selective to the left side of the grid (e.g., 0, 10, 20, ...) are most distant from the S-compartment (-1), whereas, right side selective inputs (e.g., 9, 19, 29, ...) are closest to the S-compartment. Since all movement is perfectly vertical or horizontal, movement by the simulated object produces sequences of

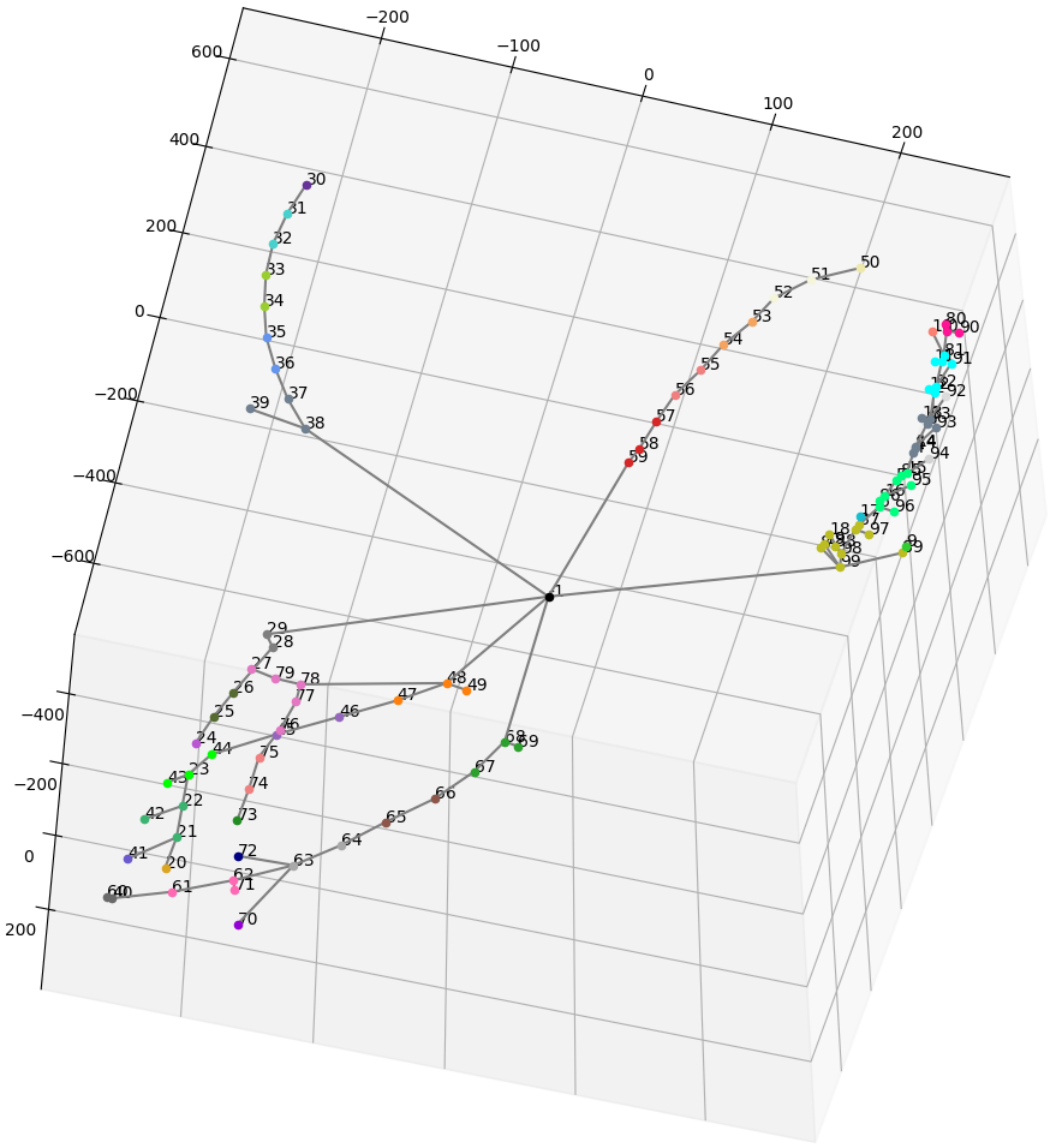


Figure 8.5: Dendritic tree of the AD neuron recognizing horizontal movement after 1,000 training epochs. Colors denote compartment membership. Numeric labels indicate the grid cell that sends signals to each neural input. -1 is the S-compartment (or soma).

input layer activations. The neural inputs receiving signals from these sequences of activations cluster to form dendritic branches. Activations do not only occur along the direction of movement. Since the receptive fields of the input neurons overlap, movement can cause lateral activations in neurons receptive to adjacent cells. However, these lateral activations occur less frequently allowing the random mask to push the branches apart.

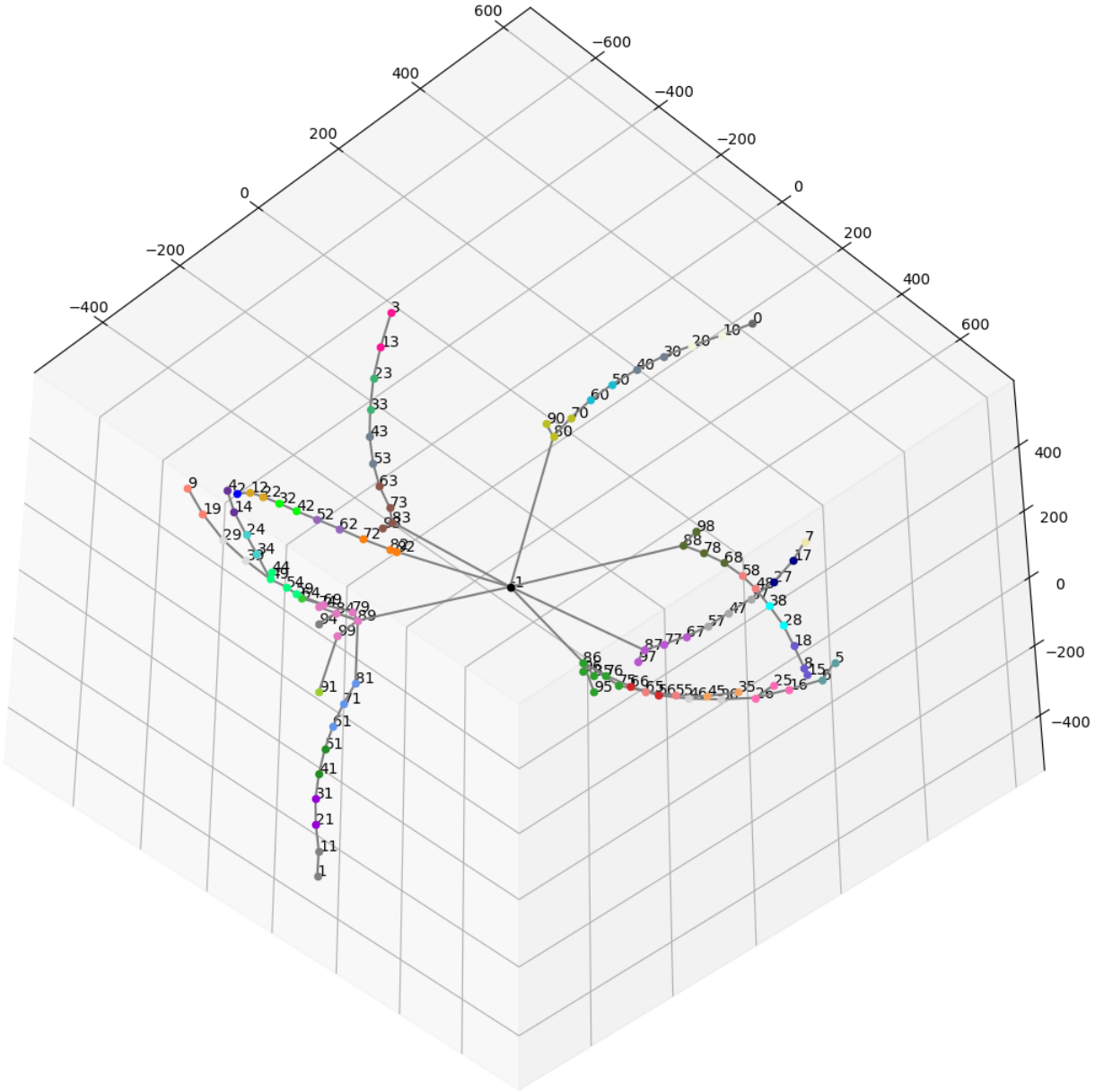
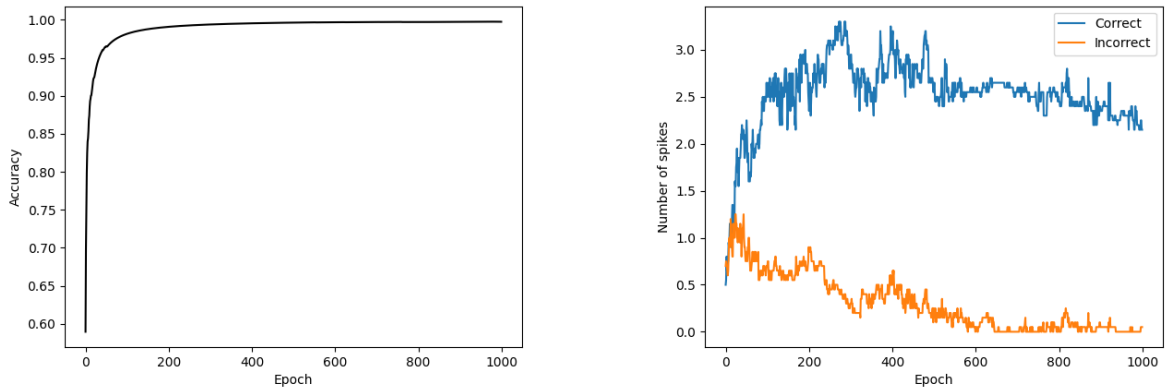


Figure 8.6: Dendritic tree of the AD neuron recognizing vertical movement after 1,000 training epochs. Colors denote compartment membership. Numeric labels indicate the grid cell that sends signals to each neural input. -1 is the S-compartment (or soma).

Each AD neuron was able to learn to discriminate one direction of movement from the other with a high degree of accuracy. Figure 8.7a shows the accuracy rate averaged across 10 trials. The final rate is $> 99\%$. After 100 epochs, learning produces rates of more than 95%.

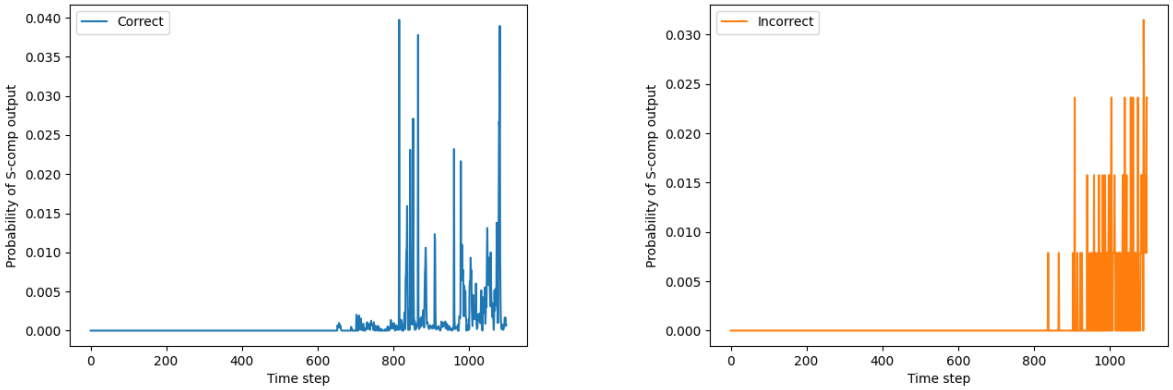
The discrimination between types of movement does not require extremely different output rates. The AD neuron learns to tell one from the other when the rates differ by just 2 Hz. Figure 8.7b shows average output rates per epoch for correct (or matching) and incorrect (or non-matching) responses. In their initial state, they both begin producing, on average, approximately 1 spike per run. After training, movement in the correct direction for each neuron results in 2 spikes, whereas, the incorrect results in 0 for the incorrect direction. To ensure that these rates were stable, the trial was run out to 10,000 epochs.

S-compartment output requires multiple signals reaching it at roughly the same time. Solitary signals do not generate activity in the two output neurons. Combined signals cause the S-compartment to spike. The timing need not be perfect. A signal arriving from distal parts of the tree may prime the S-compartment's spiking neuron model by elevating its membrane potential (v from the Izhikevich neuron model discussion). A following signal



(a) (b)

Figure 8.7: (a) The accuracy rate for the movement recognition task using AD neurons. (b) S-compartment output (number of spikes) for both output neurons. Blue: output caused by movement in the correct direction. Orange: Incorrect direction.



(a) (b)

Figure 8.8: Probability of an S-compartment output spike per time step. Blue: Correct movement. Orange: Incorrect movement.

Input Groups	**0-9**	**10-19**	**20-29**	**30-39**	**40-49**	**50-59**	**60-69**	**70-79**	**80-89**	**90-99**
Path length to S-Comp.	804	742	678	616	557	497	441	387	344	371
Average radial position	705	657	608	560	511	463	414	366	325	351

Table 8.1: Average dendritic path lengths from groups of inputs to the S-compartment of the AD neuron recognizing vertical movement (figure 8.6).

from the subsequent activation of another input neuron would push the model beyond its threshold, generating output. This one-two-punch of distal and proximal input is caused by (in the vertical case) inputs selective to the bottom row of the grid (90 to 99) being closest to the S-compartment.

The plots in figure 8.8 depict when in the course of a run S-compartment output is most likely to occur. When we compare plot 8.8a to the distances given in table 8.1, we see that an increase in the likelihood of output corresponds to the dendritic path length of the most distal neural inputs. Table 8.1 only represents input positions for the neuron recognizing vertical input, but a similar grouping can be made for the horizontal one as well. Inputs are grouped by row. The leftmost column gives the dendritic and radial distances for the neuron inputs receptive to cells forming the top row of the grid.

The high accuracy in this trial does not require a perfect spatial or connective encoding. The separation of branches is not perfect in either final example. Many of the branches

encoding rows or columns share an S-compartment connection. Or, some patterns are interspersed along a common branch. Limited tests show that these imperfect encodings do not impact performance (see hand-crafted variant below). In fact, a decent degree of accuracy can be achieved without arranging neural inputs such that branches resemble an object's path through the visual field. 80% accuracy can be achieved if from the initial state of one input per tree, inputs are given the radial positions listed in table 8.6.

We expected that neural inputs receiving signals from inputs corresponding to the final row (in the horizontal case) or column (vertical case) would be nearest the S-compartment. However, the AD neuron's recognition of a movement type does not require signals from all input neurons. Signals arriving at the S-compartment from the first seven inputs are more likely to coincide with the eighth's arrival (or next to last) input than with the ninth's.

We ran the movement recognition trial using several different configurations. In the parameter space around the configuration described above, the results are qualitatively similar: accuracy above 95%. If the maximum compartment length is significantly larger (> 200), input signals are unable to generate S-compartment output. Large compartment lengths cause the majority of the inputs belonging to a single branch to fall into one compartment. Temporally sparse input is not enough to cause such large B-compartments to produce output. On the other hand, a small compartment length (< 50) places each neural input into its own compartment. The result is that all input (correct or incorrect) is more able to generate B- and S-compartment output, degrading accuracy.

An optimal compartment length divides branches up into compartments containing 2 to 4 neural inputs. Compartments of this size can be activated by two coinciding signals. This provides just enough of a threshold to limit single signals from propagating to the S-compartment.

As stated above, the training algorithm does not produce a set of ideal dendritic trees, i.e., one pattern per dendrite. This raises the question of whether the imperfections of the trained AD neurons are responsible for errors. To answer this question we reran the trial

using a "perfect", hand-crafted pair of output AD neurons. Hand-crafted dendrites rewired the connections of trained AD neurons such that each grid row (horizontal case) or column (vertical case) occupied a separate dendrite. Input positions themselves were not altered or allowed to reposition. The AD neurons were allowed to re-compartmentalize neural inputs based on the hand-crafted tree.

The hand-crafted AD neurons performed similarly to the trained ones. Errors appeared to be caused (for both hand-crafted and trained neurons) by weak input signals. Weak signals were caused by the randomly selected axis of movement falling between two rows (horizontal movement) or columns (vertical movement). Such a path can fail to generate enough input to activate the S-compartment.

8.1.3 Discussion

The AD neuron is capable of distinguishing between different types of movement, albeit simple, disjoint categories. The AD neuron learns this task through clustering neural inputs that receive related signals along common dendrites. Since dendrites (i.e. dendritic compartments) have a squashing effect on signals exiting them, dendrites whose connections work in tandem have a far greater impact on the S-compartment than dendrites which are sparsely active. This is the strength of the AD neuron, but this task highlighted several weaknesses as well.

The training algorithm does not produce ideal dendrites. Inputs belonging to different patterns intermingle or share a main branch. However, this trial suggests that ideal shapes are not required for all tasks. In fact, it is not obvious what an ideal shape is. For example, we later found that a single dendritic tree containing all neural inputs in the distal-to-proximal ordering found in the examples above is sufficient to get a very high accuracy ($> 95\%$). This obviates the need to use the random mask and suggests that, for movement tasks, individual dendritic paths for each pattern are not necessary if they are all identical.

By comparison, it is obvious the point neuron would not be able to learn a solution to this problem because this problem is a form of exclusive-or. Weight modification alone cannot bias a neuron toward one sequence of signals and away from another. The AD neuron is successful because it encodes the timing of the sequence into the structure which is responsible for carrying the signals. Sequences matching the encoding fare better than those which do not.

8.2 FMNIST Trial

For the second trial, we chose a problem to which the point neuron may be better or equally suited, the Fashion-MNIST (or FMNIST) data set [178]. FMNIST presents a classification problem of bi-tone images of clothing which belong to one of ten categories (Figure 8.9). Deep networks of point neurons have proven to be very adept at solving this problem; however, this data set is one of the simpler tasks used to benchmark new algorithms. For example, images of the FMNIST data set can be categorized to a high accuracy using single-pixel analysis [56]. It is for this reason we suggest that a single point neuron may be as equally suited to recognize clothing images as the AD neuron.

The FMNIST data set is a benchmark data set for neural networks. The FMNIST database is a drop-in replacement for the MNIST handwritten image database due to image size (28×28 pixels) and the range of pixel intensity (0-255). It was created because state-of-the-art neural networks made solving the problem posed by MNIST trivial. As a benchmark for new algorithms, MNIST is therefore not a good predictor of performance in real-world tasks. We selected FMNIST over MNIST simply because it is a slightly more difficult task.[2] Shapes and pixel intensities vary significantly more than those of MNIST. Certain images are ambiguous enough to pose a challenge to a human. Because the challenge is greater, we anticipate differences in capabilities between neuron models to be exaggerated.

[2]We did extensive testing using the MNIST data set earlier in the life-cycle of this work but switched for the reasons given above. Some MNIST results do appear in Chapter 6 concerning the random mask.

Figure 8.9: Sample of the Fashion MNIST images and categories.

Our goal for this trial was again to compare the AD neuron to the point neuron across several tasks of increasing difficulty. Although the previous chapter described certain behaviors and contributions of each part of the AD neuron training algorithm, they were examined across different tasks. Here, we wanted to examine their contributions on the same task.

The level of difficulty was increased across tasks in two ways. We deceased the number of output neurons from one-per-category to just one for all categories. And we increased the number of categories we expected the neurons to learn. For the latter, neurons were trained to respond to examples from different categories with different output rates.

For all trials, a one-layer network of AD neurons was initialized. The network consisted of an input layer of 784 Izhikevich spiking neurons, one for each pixel of the 28×28 images. The input layer was used to encode image pixel intensities into spike trains. The input layer was connected to an output layer consisting of one or more AD neurons. Image pixel values are in the range [0,255] and were rescaled to [0,1000] to generate more activity in the input layer. Training lasted for 500 or 1,000 epochs. For some runs, we lowered the epochs to reduce training time. Each epoch consisted of 100 training examples of each category selected at random. The network was exposed to each example for 1,000 time steps.

Following each training epoch, the network was shown 10 testing examples of each category selected at random. A small number of testing examples were used per epoch (instead of the entire data set) because we wanted to show that accuracy rates were not dependent on category averages and that models were consistent across any set of bagged

examples. Spike rates for each output neuron were accumulated during this testing phase and used to calculate the error value for the subsequent training phase. Error values were calculated using the previously mentioned equation (Equation 8.2). Target output rates were set on a trial-by-trial basis. For tasks which used a one-to-one ratio of output neurons to categories, target rates were set to 10 Hz for examples matching a neuron's preferred category and 1 Hz for non-matching examples. For later tasks, target rates were staggered.

We increased the matching target rate compared to the movement trial above because we wanted to ensure a clear distinction between a correct and incorrect response.[3] All other parameters, including those of the random mask, were identical to those used in the time series experiment above.

Point neurons were created by placing all neural inputs at position $(0, 0, 0)$, adding them to the S-compartment and disabling all training algorithms except weight modification.

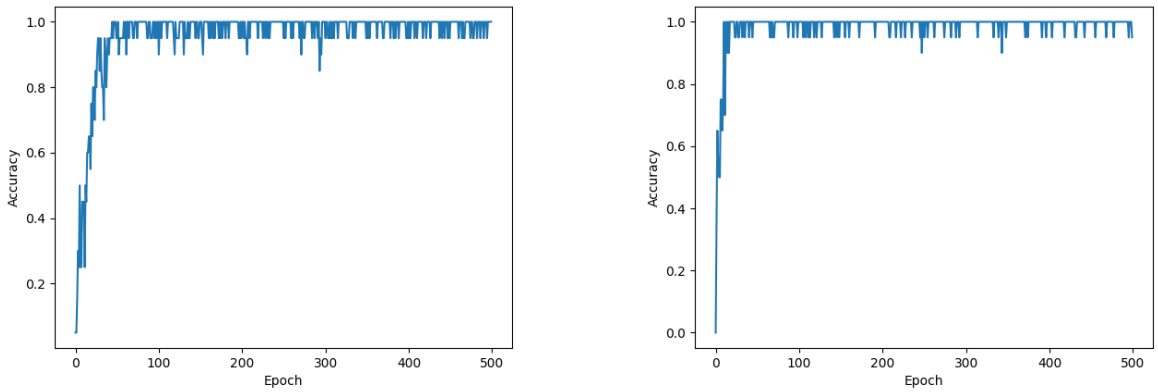
Figures in all variants depict single runs which are representative of either general behavior or some specific quirk. Trial variants were run multiple times to verify general behavior and many times to verify behavior which did not always occur (e.g., Figure 8.13).

8.2.1 Two Categories - Two Output Neurons

We ran several preliminary tests (not included in this work) which showed that both the AD and point neurons could learn a single category. Therefore, for our first significant trial, we tested whether single-layer networks of each type of neuron could learn to distinguish between two image categories: 1 (T-shirt/top) and 5 (Sandal).

We selected these two categories because, as a baseline, they are relatively dissimilar. The articles of clothing in these categories have drastically different shapes and therefore pixel overlap between examples is sparser. Also, examples in category 0 generally occupy more of the image than those of category 5.

[3]In following trials using just one output neuron, we show that using such different target output rates is probably unnecessary since a single AD neuron is capable of responding to different categories using output rates separated by just 2.5 Hz.



(a) (b)

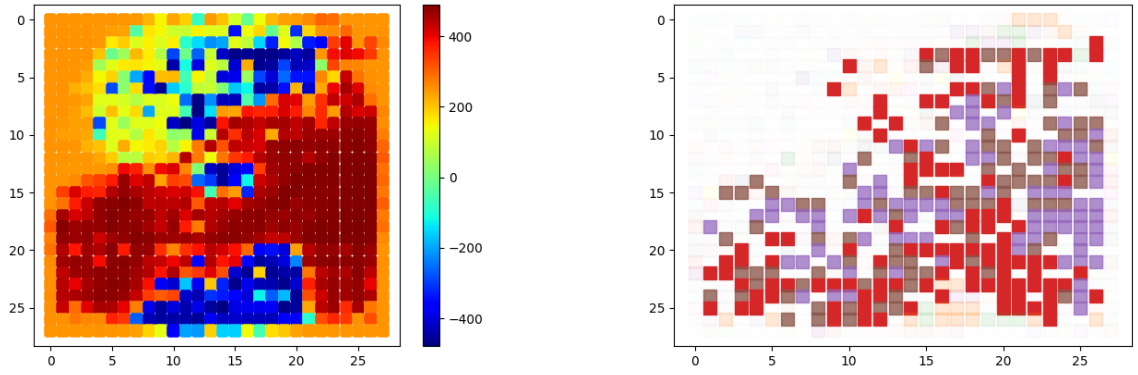
Figure 8.10: Accuracy per epoch of the (a) AD neural network and (b) point neural network for categories 0 and 5 of the FMNIST data set.

Networks of each neuron type were initialized as described above. The output layer contained two neurons, one for each category. Each output neuron would be trained to respond to one of the two categories.

Figure 8.10 shows the AD versus point results for a single test. Both networks performed well on this initial task (above 95%). The network of point neurons slightly outperforms the AD neuron and it reaches peak performance in fewer epochs. Both encodings are stable within the time frames tested (up to 1,000 epochs).[4]

The AD neuron compartmentalizes the connections frequently activated together. Figure 8.11 shows connections by actual weight (left) and compartment membership (right) for the largest compartments of the AD neuron tasked with learning category 5 (sandals). The general shape of category 5 examples can be seen in the connections with large weights. The connections with the strongest weights belong to one of several (in this case, three) compartments. While these compartments do not exclusively contain strong connections, the same general category shape is present in the combined membership of these three largest compartments. The number of compartments depends on the branching factor and

[4]We lowered the trial duration to 500 epochs for most of these tests to increase the number of tests we could perform. AD neurons with 784 neural inputs take a long time to train.



(a) (b)

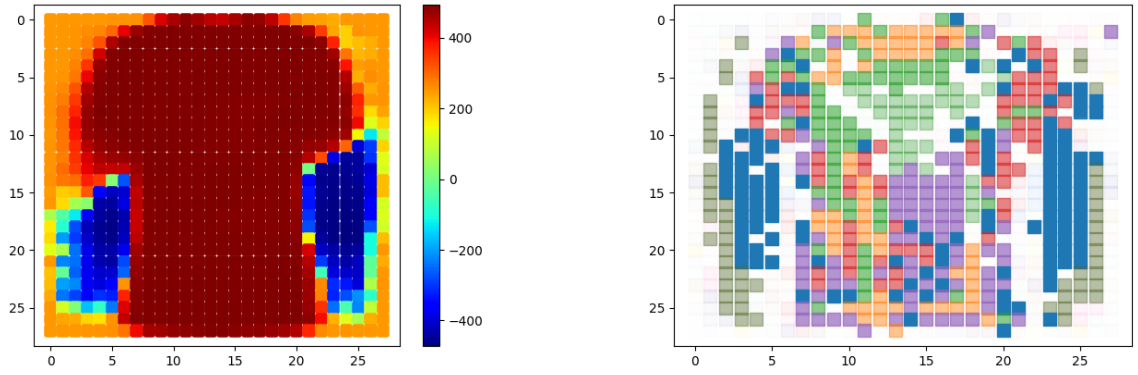
Figure 8.11: (a) The actual weight for all 784 neural inputs to the AD neuron tasked with learning category 5 (sandal). (b) Compartment membership for each neural input. Alpha channel uses compartment size to highlight only the connections belonging to the largest compartments. Colors denote compartments. Inputs in both images are positioned according to the pixel location from which they receive input.

the compartment size. A part of future work is to examine whether spatial locality within the source image (or the lack thereof) is important to compartment membership.

Connections with negative weights receive input from pixels predominantly associated with the other category (blue squares in Figure 8.11a). The range of connection weights depends largely on target output rates. Our use of 10 Hz and 1 Hz caused an extreme difference in the two (approximately $[-450, 450]$ which, being close to the maximum possible range of actual weights, $[-500, 500]$, required very large raw weights, $[-2000, 2500]$).

Figure 8.12 shows actual weights and larger compartments for category 0 (T-shirt). Interestingly, the strongly weighted connections form a well-defined map of a T-shirt; however, the category consists of examples which include both long- and no-sleeved shirts. Whereas, the former example type is not visible in the weights, it is visible in Figure 8.12 (navy and olive colored compartments).

The above suggests that certain compartments appear to contain feature information, e.g., sleeves or stomach. This is also visible in Figure 8.11b. Category 5 examples typically



(a) (b)

Figure 8.12: (a) The actual weight for all 784 neural inputs to the AD neuron tasked with learning category 0 (T-shirt). (b) Compartment membership for each neural input. Alpha channel uses compartment size to highlight only the connections belonging to the largest compartments. Colors denote compartments. Inputs in both images are positioned according to the pixel location from which they receive input.

fall into two sub-categories: flats and heeled (see Figure 8.9). Heeled sandals are represented by the red compartment, whereas, pixels indicative of flats send input to connections belonging to the purple compartment.

Altering compartment sizes does not appear to enhance this feature-compartment correlation. Very large compartments blob all connections associated with category features into one giant compartment. Smaller compartments do not appear to further isolate features. For example, the use of smaller compartments does not, say, divide the navy-colored compartment into a right and left sleeve but includes connections from both sides.

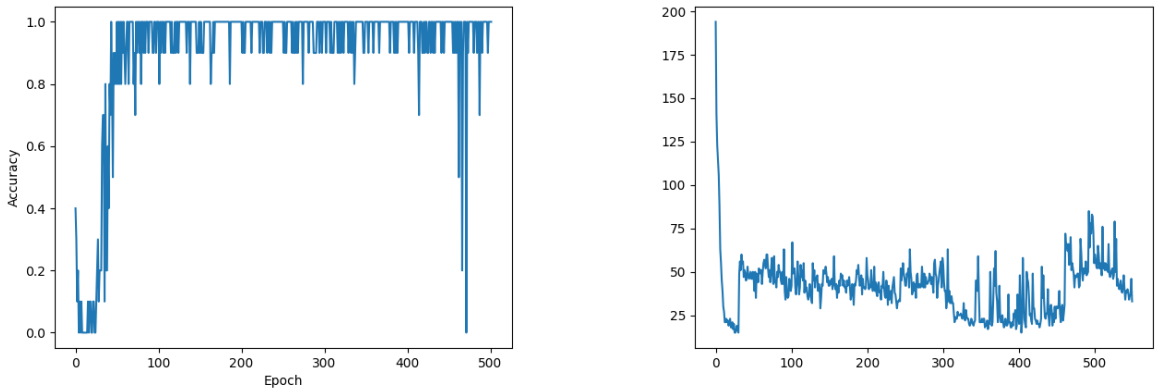
Reducing compartment size seems to have another impact on AD neuron output; it introduces more variance and instability in the results. Behavior is more easily impacted by outlier examples because smaller compartments allow individual connections to have potentially a stronger voice. Connection weights become less uniformly spread over the total range. A handful of connections will have weights at the extreme end of the upper or

lower quartile of the range. This suggests that the neuron's output has learned to recognize categories based on a small percentage of the pixels.

The main source of instability caused by small compartments is the dendritic tree's susceptibility to drastic reordering. On the other hand, larger compartments cause a general decrease in accuracy but seem to grant more stability. This could be because AD neurons with larger compartments better absorb small changes to input positions and connectivity. In larger compartments, changes to input positions appear less likely to cause splits or combinations of existing trees or compartments. With smaller compartments, sometimes the shift of a single neural input can cause compartments to divide.

Figure 8.13 shows an example of an AD neuron with smaller compartments experiencing a sudden drop in accuracy. Smaller compartments were created by increasing the branching factor to 0.8. This created smaller compartments by reducing the branching of dendritic trees and forcing more trees to originate at the root. At epoch 460, accuracy of the output neuron associated with category 0 (T-shirt) drops to zero after a long period of high accuracy. Within a few epochs, the accuracy rate recovers. Each time this has occurred, it seems to signal a fundamental reordering of compartments, either in the number of compartments or their composition. In this case, the range of the number of compartments shifts upward, doubling. However, the timeline for compartment population (Figure 8.13b) is rife with changes, shifts and sudden spikes. Such changes also cause a change in average compartment size. We are reasonably certain that the catastrophic drop in accuracy is due to changes in the AD neuron and not, for example, a bag of outlier examples since the drop persists over several epochs. However, we are uncertain why some changes affect accuracy and some do not.

We believe that the difference in impact on accuracy depends on whether compartment population changes involve proximal neural inputs. Divisions between compartments depend heavily on the angular position of proximal inputs with respect to larger clusters of more distal inputs. These proximal inputs typically form the root of a dendritic tree. If a



(a) (b)

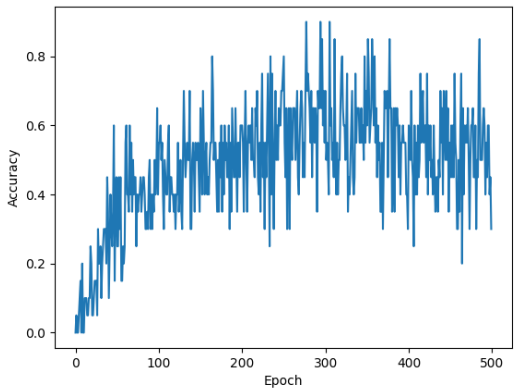
Figure 8.13: Results for a variant of the two category-two output neuron task with the branching factor set to 0.8. (a) Accuracy rates for the output neuron associated with category 0 (T-shirt). (b) The number of compartments in the AD neuron per epoch with size (i.e., number of connections) greater than 1.

proximal input's position relative to a larger distal cluster changes, a large number of inputs can change compartment affiliation.

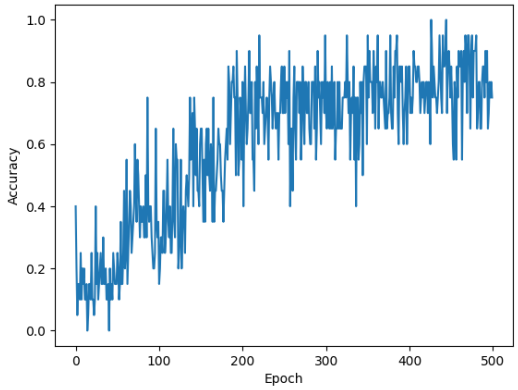
In a larger sense, impact is determined by the "importance" of the neural inputs involved in the change. A more definitive answer to this question is left for future work primarily because it requires a method for defining and tracking neural input and compartmental importance over the training life cycle. Defining importance potentially involves combining input position, connectivity, compartmentalization and weights, relative to all other inputs, into some kind of metric. We feel this is an area of artificial dendritic research beyond the scope of this work.

These comments about accuracy, stability and compartment size are anecdotal. Whether sudden drops in accuracy are due to compartment reordering or to some other phenomenon is an open question.

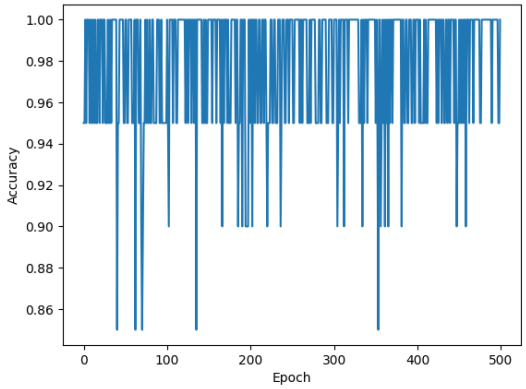
Since the AD neuron performed well distinguishing these two categories, we selected this trial to perform ablative tests. In each test, we turned off various parts of the training algorithm in order to determine their contribution. We assumed that by disabling part of



(a)



(b)



(c)

Figure 8.14: Ablative tests of the AD neural network. (a) No weight modification. (b) No radial modification. (c) No angular modification.

the overall algorithm accuracy would fall by varying degrees demonstrating the contribution of each.

Figure 8.14 shows the same trial in three different configurations. In each configuration, we turned off either weight modification, angular position training or radial position training. Removing weight modification (Figure 8.14a) had the most pronounced effect on accuracy, reducing it from the high 90s to approximately 50%. Omitting radial training (Figure 8.14b), such that all neural inputs remained in their initial positions at the outer edge of the dendritic field, had a less severe effect on accuracy, dropping it to approximately 80%. Lastly, removing angular training (Figure 8.14c), improved the

learning speed of the network. Without angular training, the network typically reached greater than 90% accuracy after just one epoch.

Angular movement is disruptive as, during initial epochs, neural inputs move across the dendritic field. This causes the creation and destruction of vastly different dendrites each epoch until the input positions settle. This is evident in the linear learning rate across the first 50 to 200 epochs. Radial movement decreases learning time by reducing the separation between a neuron's most active inputs and the S-compartment. This allows the S-compartment to be more responsive to input due to a shortening of the time between a signal's arrival at the point of input and its arrival at the S-compartment. This increases the output rate of both the S-compartment and B-compartments, which generates more change in connection weights. This reinforcement of weight modification by radial repositioning is the cause for the quick learning seen in the no-angular test in Figure 8.14c.

What, then, does the angular modification provide in the context of this trial? It slows the the rate at which the network encodes a category. However, angular position training contributes stability to the model. In our trials, disabling angular training lowered the overall accuracy of the trained model by approximately 2%. We believe this is caused by compartmentalization. While compartmentalization was not disabled in any of the tests using AD neurons, without angular position training, the vast majority of neural inputs belong to a compartment all their own, making it a close cousin to the point neuron. With angular training, compartment sizes grow. Compartments dilute the individual impact of the neural inputs contained within them. Each input on its own is less important. A compartment's output is nonlinear with respect to the amount of input. Above a certain point, more input does not increase a compartment's output.

These properties of compartments cause the AD neuron to be less susceptible to outlier input examples, i.e., examples with different shapes or pixel intensities. This seems also the reason why the rate of learning decreases as it takes time for compartments to form and connection weights to adjust.

The AD neural network's performance was on par with the network of point neurons. This led us to wonder whether, for certain tasks, there is any benefit to using AD neurons.[5] Previous tasks suggest that AD neurons are capable of recognizing different patterns through isolating sets of neural inputs on different dendritic trees. We hypothesized that the AD neuron might outperform the point neuron if the number of output neurons was reduced. In other words, can one neuron learn to differentiate between categories? This question formed the basis of the next variant of the FMNIST trial.

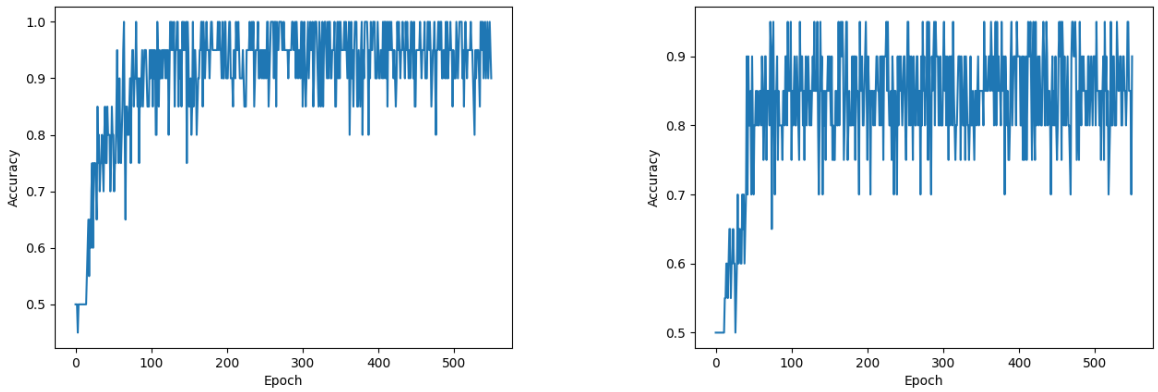
8.2.2 Two Categories - One Output Neuron

Networks of each neuron type were initialized as described above. The output layer contained one neuron. This single output neuron was trained to recognize different categories by responding with a different spike rate for each category. The target rate for examples belonging to category 0 (T-shirts) was 10 Hz and the target rate for category 5 was 5 Hz. These two rates were chosen based on previous experience with setting target rates. Each rate is both achievable by the default AD neuron configuration, different enough to allow for identification but not so different as to prevent the reaching of either target.

The output rate was matched to a target rate by selecting the minimum of $|r - t_i|$ where r is the spike rate for the output neuron, and t_i is the target rate for category i. Only the final 200 epochs were used for accuracy calculations to omit rates from earlier in the training process.

Figure 8.15 shows an example of the accuracy rates for this variant (two categories and one output neuron). Expectedly, the accuracy of both networks is diminished. The AD neural network accuracy rate dropped to 93% and the network of point neuron fell to 84% across five runs. This is a reversal from the two output neuron variant above in which the network with point neurons was more accurate.

[5]After all, some neurons in the brain are adendritic.



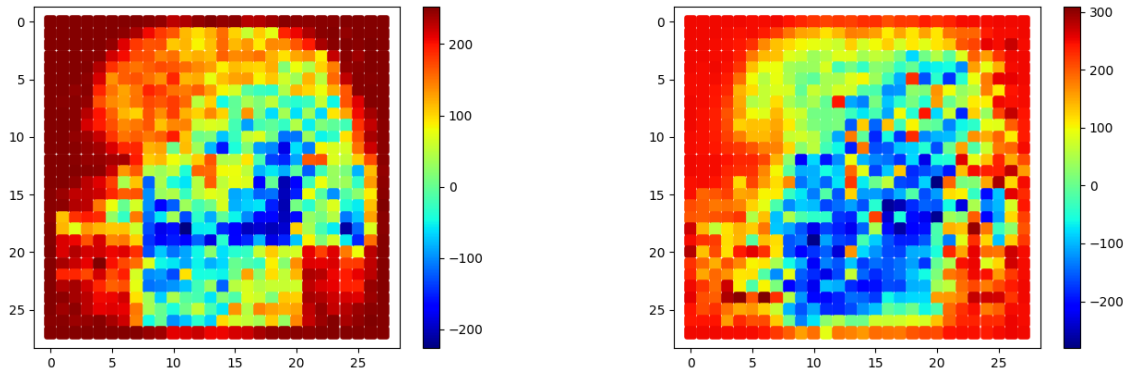
(a) (b)

Figure 8.15: Accuracy per epoch of the (a) AD neural network and (b) point neural network for categories 0 and 5 of the FMNIST data set.

To understand what causes this difference in diminished performance, we must look into the interaction of neural position, compartmentalization and weights. Figure 8.16 shows the actual connection weights for all incoming connections to the output neuron in each type of network.[6] While the connection weights of both neurons vary significantly, the range of the point neuron's are typically greater. Second, the rough shape of each category (T-shirt and sandal) is more pronounced in the former than the latter. Category 0 (T-shirt) is roughly visible in the point neuron's pixel-by-pixel weight map, category 5 is not. Connections corresponding to pixels with defining intensities for both categories (i.e., where the two categories overlap in the image) have a strong negative weight in the AD neuron. In the point neuron, more connections have extremely negative actual weights.

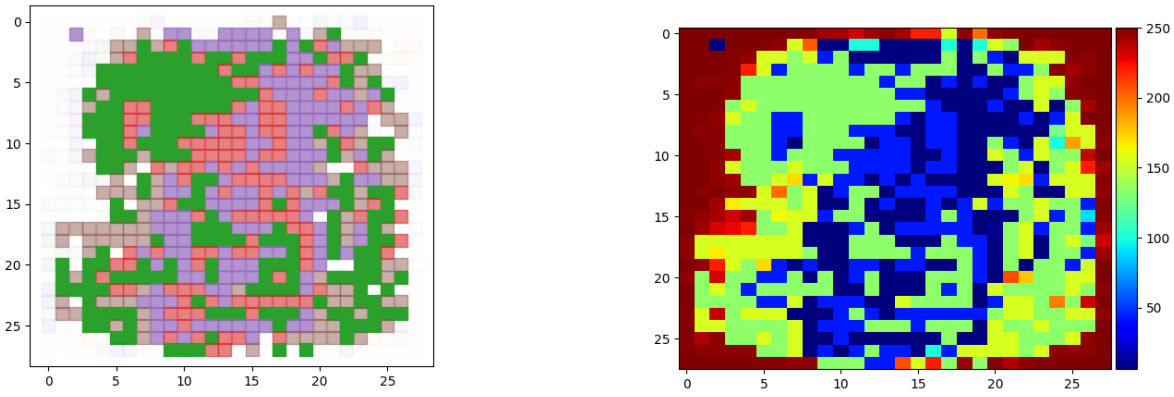
In the point neuron, since all connections belong to the S-compartment, input signals directly impact the state of the S-compartment. Individual connections are more capable of producing neural output. Similarly, neural output modifies each connection directly. Thus, when activity at an individual connection causes S-compartment output, it causes modifications to itself. This allows very active connections to push themselves toward one

[6]Recall, actual weights are bounded between a minimum and maximum using the equation: $w_a = \frac{w_r w_b}{|w_r|+w_b}$, where w_r is the raw weight and w_b is the bound. w_a is in $[-w_b, w_b]$.



(a) (b)

Figure 8.16: Weights of all afferent, external connections (i.e., connections originating in the input layer). (a) Network with an AD neuron. (b) With a point neuron.



(a) (b)

Figure 8.17: (a) Compartment membership for each connection arranged by pixel location. Different colors represent different compartments. Color transparency was used to highlight only the largest compartments. (b) Actual connection weights averaged by compartment.

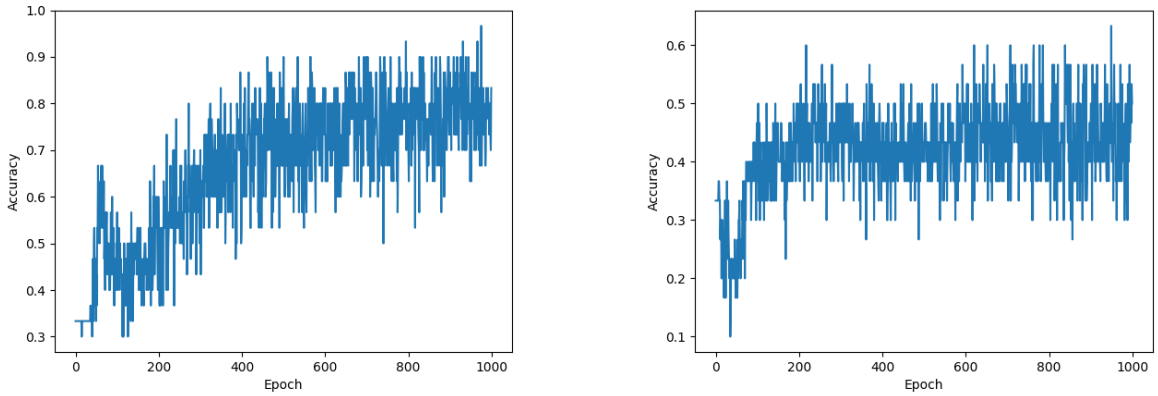
extreme or the other in terms of connection weights. By comparison to the AD neuron, when a B-compartment causes S-compartment output, the B-compartment's outgoing connection is affected not the connections belonging to the B-compartment. This puts a barrier between S-compartment activity and low, connection-level changes.

Compartments are also distinguishable by weight. Figure 8.17 shows membership of the largest compartments and the average connection weight of each compartment. While

compartments contain a mix of different connection weights, the mix does not appear to be uniform. The purple and red compartments cover the torso and right shoulder area of category 0 (T-shirt). The green colored compartment contains almost all connections activated by pixels of the left shoulder. Many of green compartment connections belong to category 5 which lie outside the area typically occupied by the shirt. In fact, the red and purple compartments contain connections frequently activated by both categories; whereas the green compartment contains connections activated by just one. The green compartment contains more connections with strong weights and vice versa for red and purple.

The above arrangement of connections, weights and compartments is typical of overlapping patterns. Connections frequently used by both patterns often have weak or negative weights. These inhibit activity at the compartment level. If we take the mean actual connection weight of a compartment as an indicator of how likely its connections contribute to accurate output, we can begin to understand the role of each. This assumes that all connections in a compartment are equally likely to receive a signal on a given time step. Since co-activity forms the basis of compartment creation (via input position), this assumption makes sense as a general rule.

Compartments with many low or negatively weighted connections absorb input. These compartments usually contain connections receiving input from ambiguous sources, i.e., pixels whose values are often indicative of both categories. In the example above, the purple compartment appears to have this function as its mean connection weight is just 6. The red compartment has a mean actual weight of 42. At the other end of the spectrum, connection's which receive less ambiguous input about an example's category belong to compartments with a stronger mean actual weight. The green and brown compartments contain many connections which are only typically activated by one or the other category. Green's mean actual weight is 133, and brown's is 155.



(a) (b)

Figure 8.18: Accuracy for the three category - one output neuron variant of the FMNIST trial. (a) The output neuron consists of a single AD neuron. (b) The output neuron is a single point neuron.

This reinforces the conclusion that the separation of connections by compartments is a large part of what enables an AD neuron to learn multiple categories. To explore this further, we ran another variant which increased the number of categories to three.

8.2.3 Three Categories - One Output Neuron

The previous experiment was repeated with the addition of a third clothing image category: category 1 (trousers). Category 1 was given a target rate of 7.5 Hz, which is between the two other categories' rates.[7] Adding a third category increased the difficulty of the task in two ways. First, the single output neuron was tasked with learning a third type of item. And second, the separation between target output rates was cut in half (from 5 Hz to 2.5 Hz). Also, we increased the number of epochs for the experiment to 1,000 because we found that output rates took longer to stabilize.

Figure 8.18 compares the accuracy and accuracy variance of a typical run for both the AD neuron (a) and the point neuron (b). Table 8.2 shows the mean accuracy and standard

[7]In a single test, it is impossible for a neuron to spike seven and a half times in 1,000 time steps. However, error rates are calculated based on the mean rate from one epoch, so a target rate of 7.5 Hz is achievable. However, concerned that this might inadvertently cause instability, we ran a later test using rates of 12 Hz, 9 Hz and 6 Hz. Results did not change.

	AD	POINT
SD	0.076	0.064
MEAN	0.77	0.44

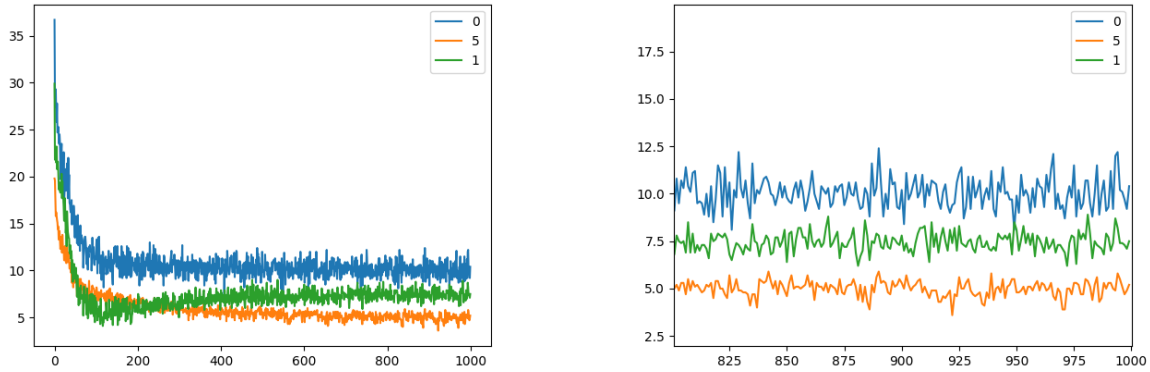
Table 8.2: Mean accuracy and SD for five runs of the three category, one output neuron trial variant. These numbers were compiled from the accuracy rates of the second half of five runs (i.e., the final 200 epochs).

deviation for five runs of each. Adding a third category lowers the accuracy rates of both. For the point neuron the accuracy is roughly half of what it was (a 40 point reduction) from the two category, one output neuron variant. The AD neuron also showed diminished accuracy, dropping from 93% to 77%. However, results for the AD neuron varied substantially more than in the previous task; the standard deviation increased from 0.049 to 0.076. The point neuron's results did not vary significantly more (SD change: 0.061 to 0.064).

To understand the difference in the AD and point neuron results, we examined how well each network matched the target output rates for each category. Figure 8.19 show the mean testing phase output rates by category for the AD neuron, and Figure 8.20 shows the same for the point neuron. The plots on the right of each figure provide a closer look at the final 200 epochs of a typical run. Both neurons successfully matched output rates to the target rates.

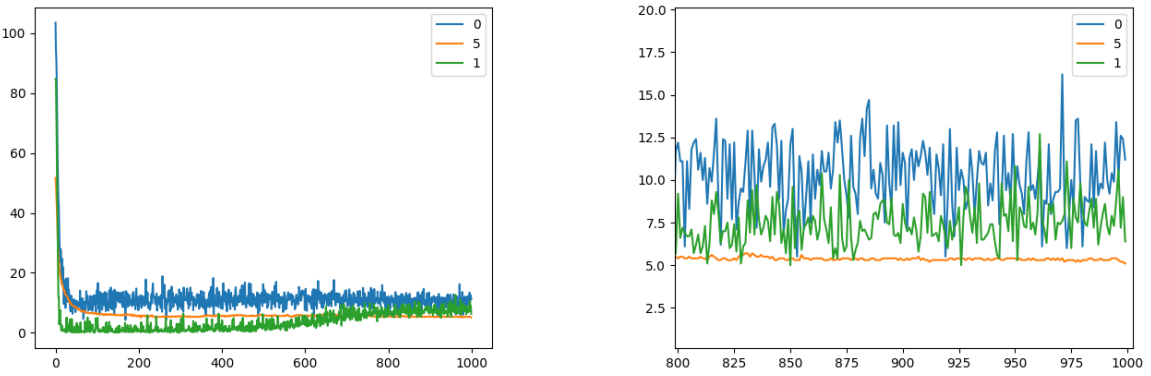
The reason why the AD neuron outperforms the point neuron can be seen in the closer treatment of the output rates. For the AD neuron, the output rate generated by each category varies similarly within each epoch. Rate changes caused by the previous training phase have a similar impact across all categories. A relatively constant separation is maintained between output rates. This is not true for the point neuron. Category output rates do not always train in the same direction. This causes the epoch-by-epoch separation between output rates to vary, sometimes allowing rates to cross.

Figures 8.19 and 8.20 show several additional quirks of the training process. The output rate generated by category 1 examples falls initially below the rates of other categories for



(a) (b)

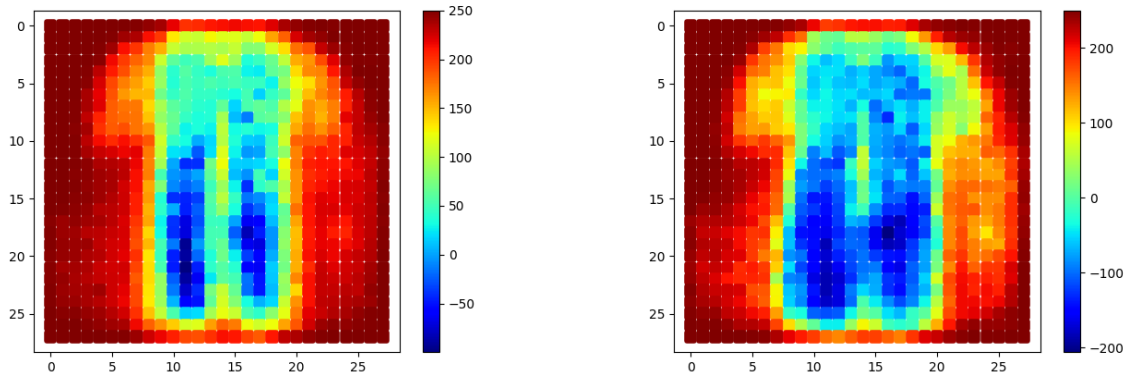
Figure 8.19: Mean testing phase output rates by category for the AD neuron. Target rates were: category 0, 10; category 1, 7.5; category 5, 5. (a) Output for an entire run. (b) Zoomed in view of (a) of the last 200 epochs.



(a) (b)

Figure 8.20: Mean testing phase output rates by category for the point neuron. Target rates were: category 0, 10; category 1, 7.5; category 5, 5. (a) Output for an entire run. (b) Zoomed in view of (a) of the last 200 epochs.

the first part of training. The target rate for category 1 examples is between the other two. This behavior is evident in the training of both neuron types. For the AD neuron, the target rate is typically found around epoch 200, whereas, the point neuron does not find it until epoch 600 or 700. This behavior in the point neuron is particularly baffling because this change in rates takes place after the output rates have been stable for approximately 500 epochs. This phenomenon poses two questions. First, why does the category 1 rate



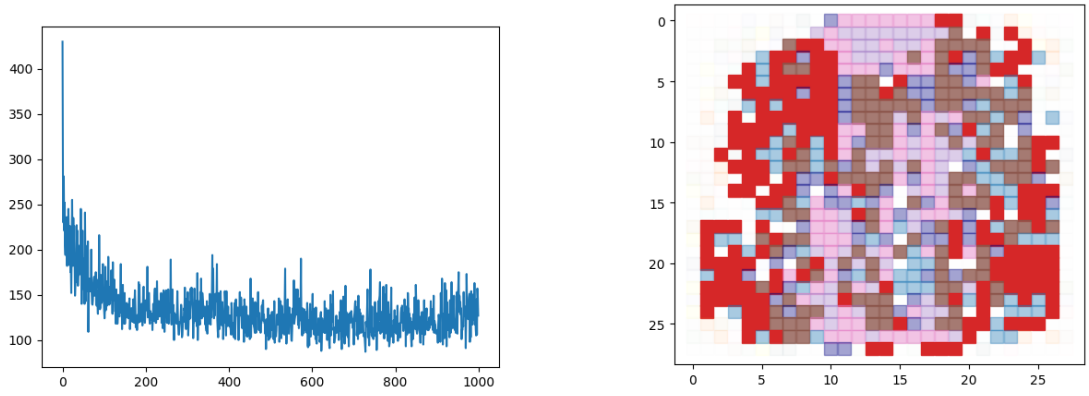
(a) (b)

Figure 8.21: Actual weights for the point neuron arranged by input pixel location within the FMNIST images. (a) Actual weights at epoch 200. (b) Actual weights at epoch 900.

drop below the others during the first part of training? Two, what change in neural state causes it to recover? We also note that the drop in category 1 rates happen regardless which target rate is given. We also tried switching rates with category 0 and giving category 1 a target rate of 15 Hz. We reiterate that for all trials, the order of categories per epoch was randomized.

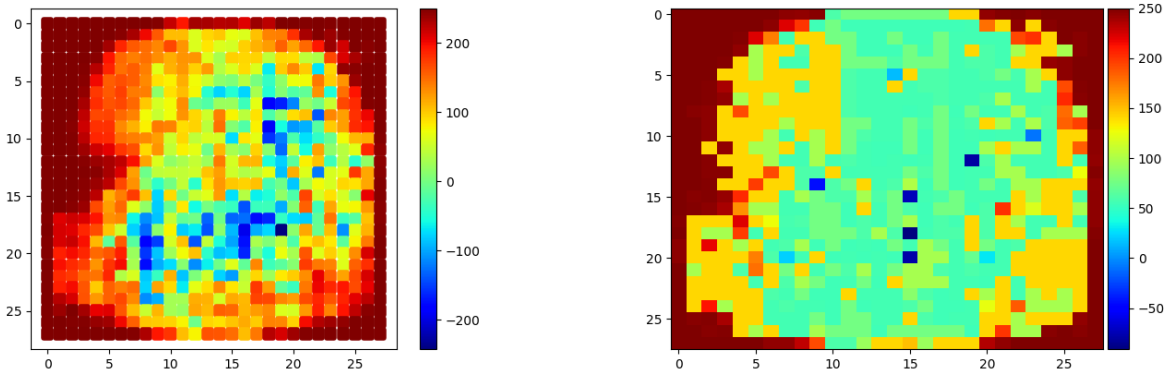
The reason why category 1 output rates bottom out is due to the nature of the category itself. Category 1 (trousers) examples significantly overlap category 0 (T-shirt) in terms of pixel intensities. The average category 1 example uses a subset of the pixels used by category 0 to represent the article of clothing. This can be seen in the examples in Figure 8.9. Because output rates of both neuron types in their initial states are very high relative to the target rates, training rapidly reduces the weights of frequently active inputs over the first few epochs. Reducing category 0 output rates has a strong impact on the rates of category 1. This initial reduction in weights affects a small set of inputs activated by all categories.

For the point neuron, weight readjustment causes the recovery of category 1 output rates. Figure 8.21 shows an example of weights for the point neuron at epoch 200 and at epoch 900. Initially, low or negative weights are concentrated at connections associated



(a) (b)

Figure 8.22: (a) Number of compartments by epoch of the AD neuron for the run depicted in Figure 8.18a. (b) Connections arranged by input pixel location. Colors denote compartment membership. Color transparency was used to show only the largest connections.



(a) (b)

Figure 8.23: Final actual weights for the run depicted in Figure 8.18a. (a) Per connection (b) Per compartment (mean).

strongly with category 1 (trouser legs). After 700 more epochs of training, low or negative weights are spread over more connections. This is particularly true of the connections associated with the sleeve area of category 0 (T-shirt).

The AD neuron recovers the output rate of category 1 examples much more quickly. Compartmentalization allows the output rate to recover. The number of compartments stabilizes near the time when category 1's output rate begins to rise (approximately epoch

200). Once the training algorithm depresses the weights of connections frequently activated by category 1 examples to a point below which example input cannot reach the target output rate, category 0 and 1 examples each try to push a similar set of connection weights in opposite directions. In other words, category 1 examples will try to lift weights while category 0 examples will try to keep them depressed.

Compartmentalization separates certain connections (or groups of connections) that are salient to each category. Separating connections causes their signals to combine indirectly and only after each has been transformed by their respective compartment's nonlinear activation function (ψ). Signals salient to different categories are no longer in direct competition--reinforcing or cancelling each other. More importantly, since weight modification is caused by the local compartment spike rate, connections in different compartments cannot directly modify each other's weights.

Another curiosity of training regards to the point neuron, which is evident in Figure 8.20 (and later in Figure 8.25). The output rate generated by examples belonging to category 5 are flat compared to the other two. Since this does not concern the AD neuron, we were only interested in testing whether this artefact was due to our implementation. To understand whether this phenomenon was caused by the choice of rates or by the category itself, we performed several runs in which rates were changed or simply swapped. Pairing category 5 with different categories also did not change this behavior. Regardless the selection of target rates, the output rate associated with category 5 examples behaved the same (the same goes for category 5 in the two category variant and category 8 in the final five category variant). We are uncertain of the cause of this phenomenon.

Figure 8.23 shows the actual weights of each connection and the mean actual weight for each compartment. Even with this simple example of an AD neuron trained on just three categories, expressing why specific connections belong to specific compartments is difficult. However, we do see some similarities with the two category variant (Figure 8.17). Compartments with high mean weights tend to have connections which receive input from

pixels indicative of just one category, for example, the left sleeve area of category 0 (T-shirt) or the heel and toe areas of category 5 (sandal). Compartments with lower mean weights tend to receive input from pixels which overlap one or more categories, for example, the knee area of category 1 (trouser) or the right sleeve of category 0 which often overlaps with high-heeled sandals of category 5.

Ultimately, the training algorithm tends to place connections in different compartments whose inputs are activated by examples of more than one category and those which are not.

This trial indicates that the AD neuron is capable of separating, via compartmentalization, information associated with different categories. This added capacity gives it an edge over the point neuron. However, there must be an upper limit to the number of categories one neuron can encode. This formed the question for the final experiment.

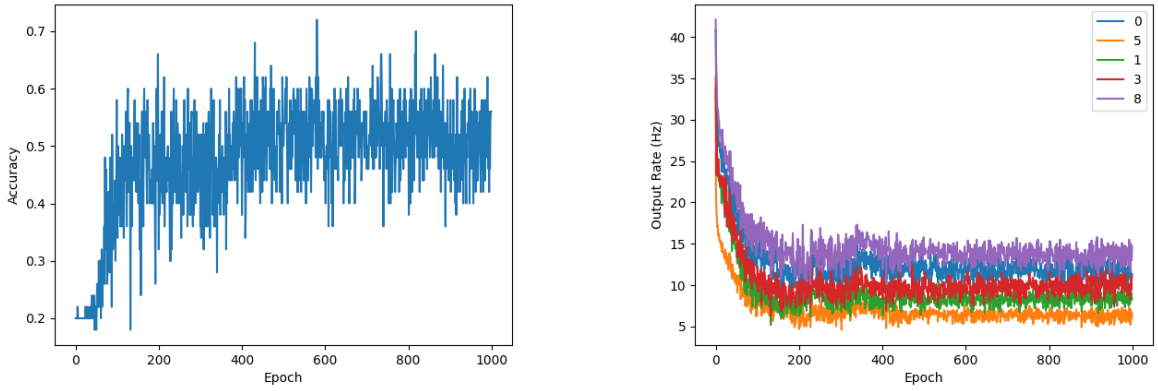
8.2.4 Five Categories - One Output Neuron

For our final variant of the FMNIST trial, we again increased the number of categories to be learned by a one layer network with a single output neuron. All parameters and categories from the previous variant were retained. Examples from two new categories were added to the training and testing sets: category 3 (dress) and category 8 (bag). We selected these categories because they are distinct types from the other three categories. No quantitative measure of distinctiveness was made. We simply wanted categories that contain different types of clothing. The unused categories--pullover, coat, shirt, sneaker and ankle boot--are variations of the five used categories.

The two new categories, dress (3) and bag (8), were given target output rates of 12.5 Hz and 15 Hz respectively. Table 8.3 gives the rates by category. Rates of previously used categories were kept the same because tests from the previous variant suggests that matching categories to rates does not significantly affect accuracy. Furthermore, for these

	Category 5 (sandal)	Category 1 (trouser)	Category 3 (T-shirt)	Category 0 (dress)	Category 8 (bag)
Target Rate	5 Hz	7.5 Hz	10 Hz	12.5 Hz	15 Hz

Table 8.3: Target output rates for the five category variant of the FMNIST trial.



(a) (b)

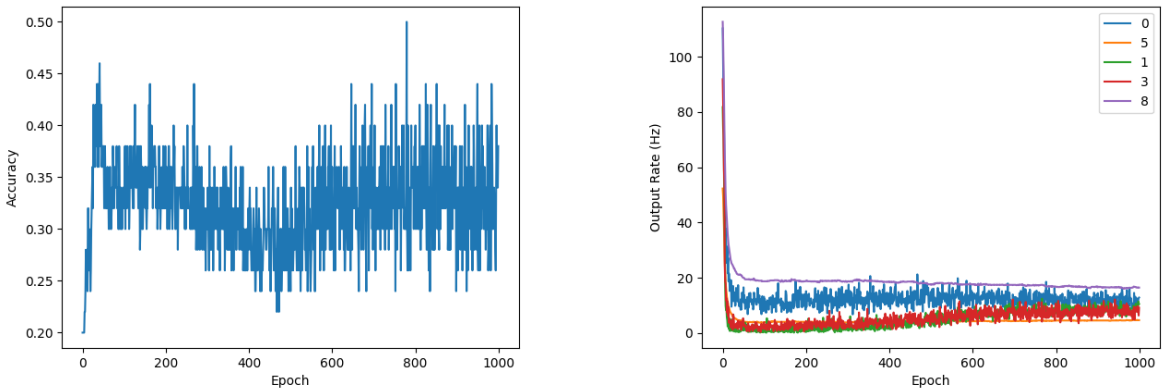
Figure 8.24: AD neuron performance in the five category, one output neuron variant. (a) Accuracy rate. (b) Testing phase output rates by category.

variants, we are only interested in changes to AD neuron learning caused by more categories than a study on the affects of rate choice on accuracy.[8]

The AD neuron outperformed the point neuron in this variant; however, the difference between their accuracy narrowed compared to the three category variant. The AD neuron was 51% accurate across three runs and the point neuron was 32% accurate. Figures 8.24 and 8.25 show the accuracy and output rates of a typical run for both networks. Can we identify the cause for this increased drop in accuracy?

An increase in the number of categories causes a shift in the AD neuron's ability to match output rate to target rate. Figure 8.26 shows the output rates for the AD neuron for the final 100 epochs of a run. Output rates are pulled toward the mean of all target rates.

[8]Limited tests were run by swapping rates between categories and by assigning different rates. Swapping rates did not affect accuracy; however, separating rates did have an effect. For example, in the previous variant (Section 8.2.3), we tested category 1 (trouser) with 15 Hz instead of 7.5 Hz and saw a small gain.
We suspect that increasing the separation between rates also causes instability as the neuron tries to accommodate a wider range of output rates.
More study in this area is needed.



(a) (b)

Figure 8.25: Point neuron performance in the five category, one output neuron variant. (a) Accuracy rate. (b) Testing phase output rates by category.

For example, the output rate generated by category 8 examples is 13.6 Hz, a 1.4 Hz drop from the target of 15 Hz. Category 5, which should be 5 Hz is 6.3 Hz, a 1.3 Hz rise. The category with the middle rate (category 1 with a target of 10 Hz) produces an output rate of 9.9 Hz, which is almost exact.

This phenomenon is caused by training as it adjusts the neuron's state to match five different output rates. It is a direct cause of the further reduction in accuracy of the AD neuron compared to the three-category trial. Effective target output rates are closer together. While this decreases the variance in rates compared to the three-category trial (SD 0.059), the decrease is not sufficient to maintain accuracy. This pull-to-the-center is completely absent from the previous three-category variant. The point neuron shows an opposite, albeit inconsistent, behavior where rates are pushed away from the center. In the AD neuron, this could be another indicator (along with accuracy) of the encoding capacity for a particular configuration.

Trained connection weights between the two neuron types differ significantly (Figure 8.27). For example, the left side of the weight map is extremely positive in the AD neuron's case and very negative for the point neuron. In general, the point neuron utilizes more negatively weighted connections than the AD neuron and its very positively weighted

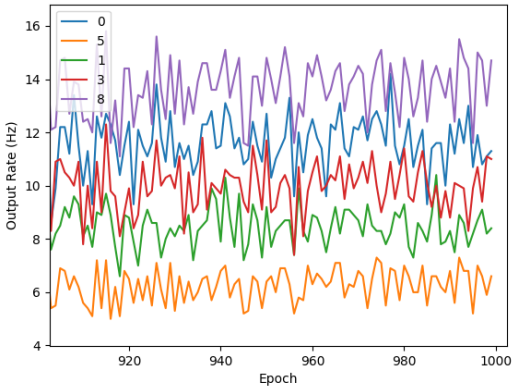
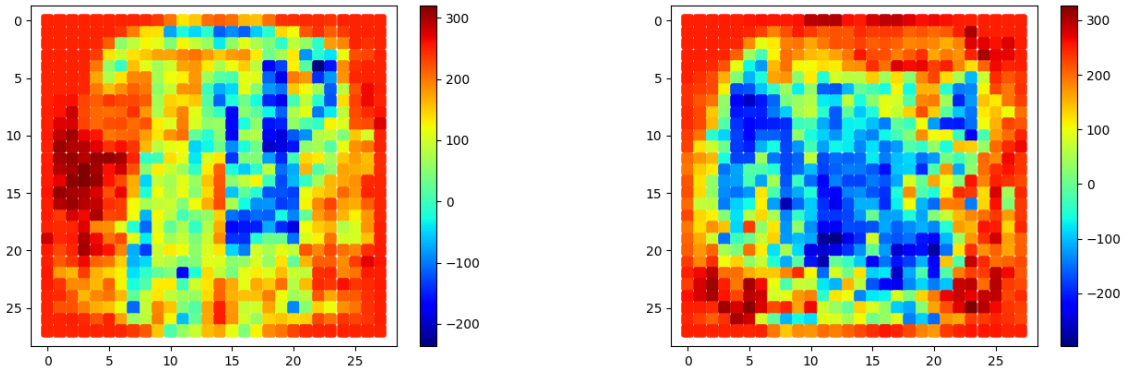


Figure 8.26: A closer look at the final 100 epochs from Figure 8.24b.

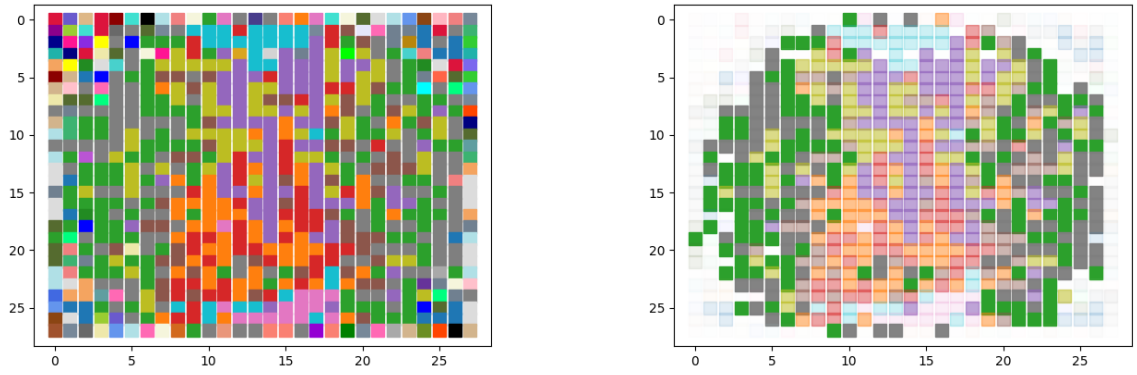


(a) (b)

Figure 8.27: Actual connection weights arranged by input pixel location within the source images. (a) AD neuron. (b) Point neuron.

connections lie predominantly along the periphery. This is likely due to the moderating effects of compartmentalization which diminish the impact of individual signals. If we compare the actual AD neuron connection weights in Figure 8.27a to compartment membership in Figure 8.28, we again see a positive-negative weight divide between compartments. Green and gray compartments hold more of the positive weights while purple, red, orange and yellow hold more of the negatively weighted connections.

To get a fuller picture of connection weight-compartment relationships, we can examine if the dendritic field contains any visible meta-patterns. Figure 8.29 compares neuron input

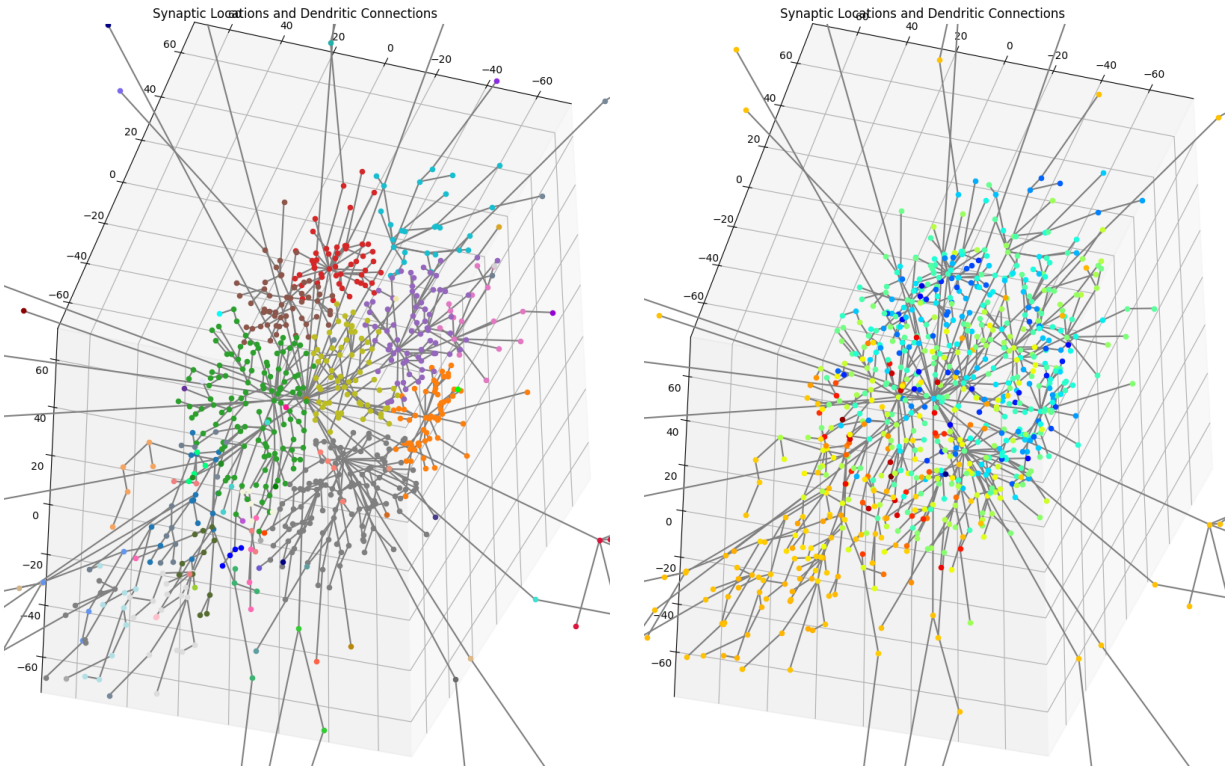


(a) (b)

Figure 8.28: Compartment membership of AD neuron connection arranged by input pixel location within the source images. Different colors denote different compartments. Same neuron is depicted in both images. (a) All compartments are visible. (b) Only the largest compartments are visible.

position, compartment membership and actual weights within the dendritic field. The compartment colors match those of Figure 8.28 (i.e., same neuron). First, the compartments mentioned above with many negatively weighted connections cluster toward the top right of the figure. Those with more positive connections are in the bottom left. The positive-negative division between compartments is a result of input position within the dendritic field.

Do compartment positions within the dendritic field mimic the location of input pixels within the images? The gray and green compartments occupy the image's left and right flanks; however, their associated input pixel locations are thoroughly interspersed with each other. An examination of Figure 8.30 suggests that within each compartment field location has little correlation to pixel location. Considering what we know about the behavior of the input position algorithms from the previous chapter (see Task 1), it is possible that all neural inputs in both compartments are drawn together. The tree-building and compartmentalization algorithms are unable to compensate for such a large cluster of inputs with an irregular (i.e., non-circular) shape. Therefore, they are split into two or more compartments. Similarly, it is likely some connections belonging to the yellow



(a) (b)

Figure 8.29: A 3D-view of the trained AD neuron. Circles show the neural input positions associated with each incoming connection. Gray lines show the links between them created by the tree-building algorithm after the final training epoch. The same neuron is depicted in both sub-figures. (a) Neural inputs are colored by compartment membership. (b) Neural inputs are colored by the associated connection weight.

compartment, by the location of their neural inputs in both the dendritic field and image, belong to either the green or gray compartments. Obviously, this is not the only interpretation.

An alternative interpretation can be argued from the red and orange compartments. If we examine just their associated pixel locations, by the logic above, we might say their connections should all belong to the same compartment. However, if we examine their relative locations within the dendritic field, we find the orange and red compartments on opposite sides of the main cluster, completely separated by the yellow and purple compartments. The red and orange compartments suggest pixel locality is not (or less) important. In fact, the green and gray compartment connections show locality in the

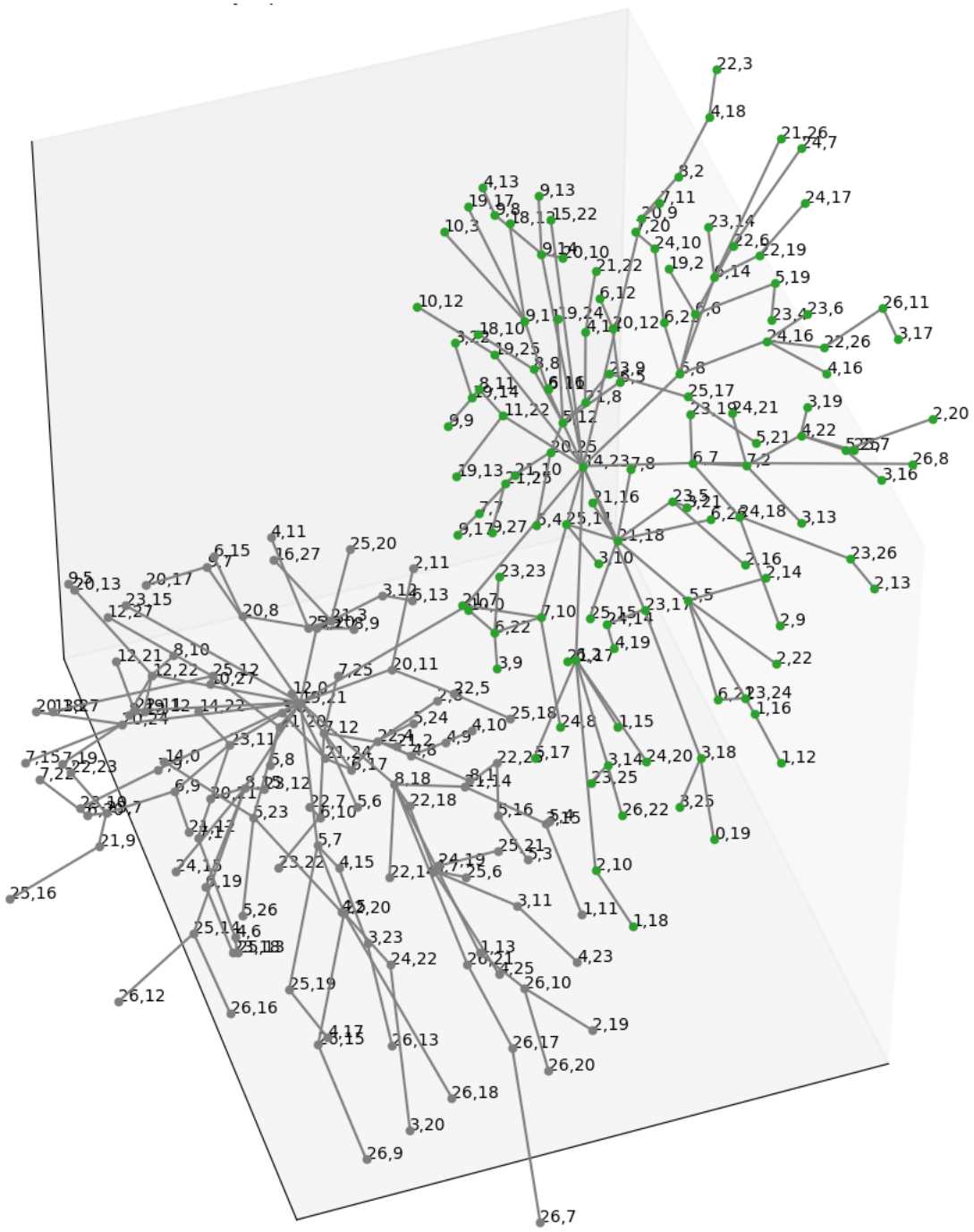


Figure 8.30: Closer view of the dendritic field of Figure 8.29. Compartments 3 (green) and 8 (gray) visible. Labels are the x,y input pixel locations within example images.

dendritic field and in the image due to a distinct feature in one of the categories, for example, the sleeves of a shirt.

Lastly, the pink and light blue compartments contain connections whose associated pixels are exclusively at the top and bottom of the image. Within the dendritic field, the associated input positions of both compartments are close together. Figure 8.31 depicts neural input location within the dendritic field (points) and within the image (labels). Connections of the light blue compartment are predominantly associated with input pixels at the top of the image. However, some are associated with pixels at the bottom. If these latter connections were on the border between the pink and light blue compartments, we might assume that the tree-building and compartmentalization algorithms disrupted their inclusion in the pink compartment. However, they are not, e.g., the neural input labeled 10,26 in the upper right hand corner of Figure 8.31. They are scattered throughout the compartment's cluster.

Calibration tasks (Figure 7.3) of the last chapter showed that relationships between inputs can restrict angular movement, especially if the number of related inputs is large. It is possible that this prevents their associated connections from joining compartments. Although the final chapter talks about how this effect might be reduced (see 9.3.8), we are unsure whether (a) the general problem strongly affects performance, (b) any given input suffered such restriction, (c) a perfect spatial encoding exists and (d) assuming a perfect encoding exists, the tree-building and compartmentalization algorithms can take advantage of it.

The five category trial suggests that AD neurons have a limit to the number of categories they can encode. This limit is likely a function of the size of the neurons input and the similarity of the categories.

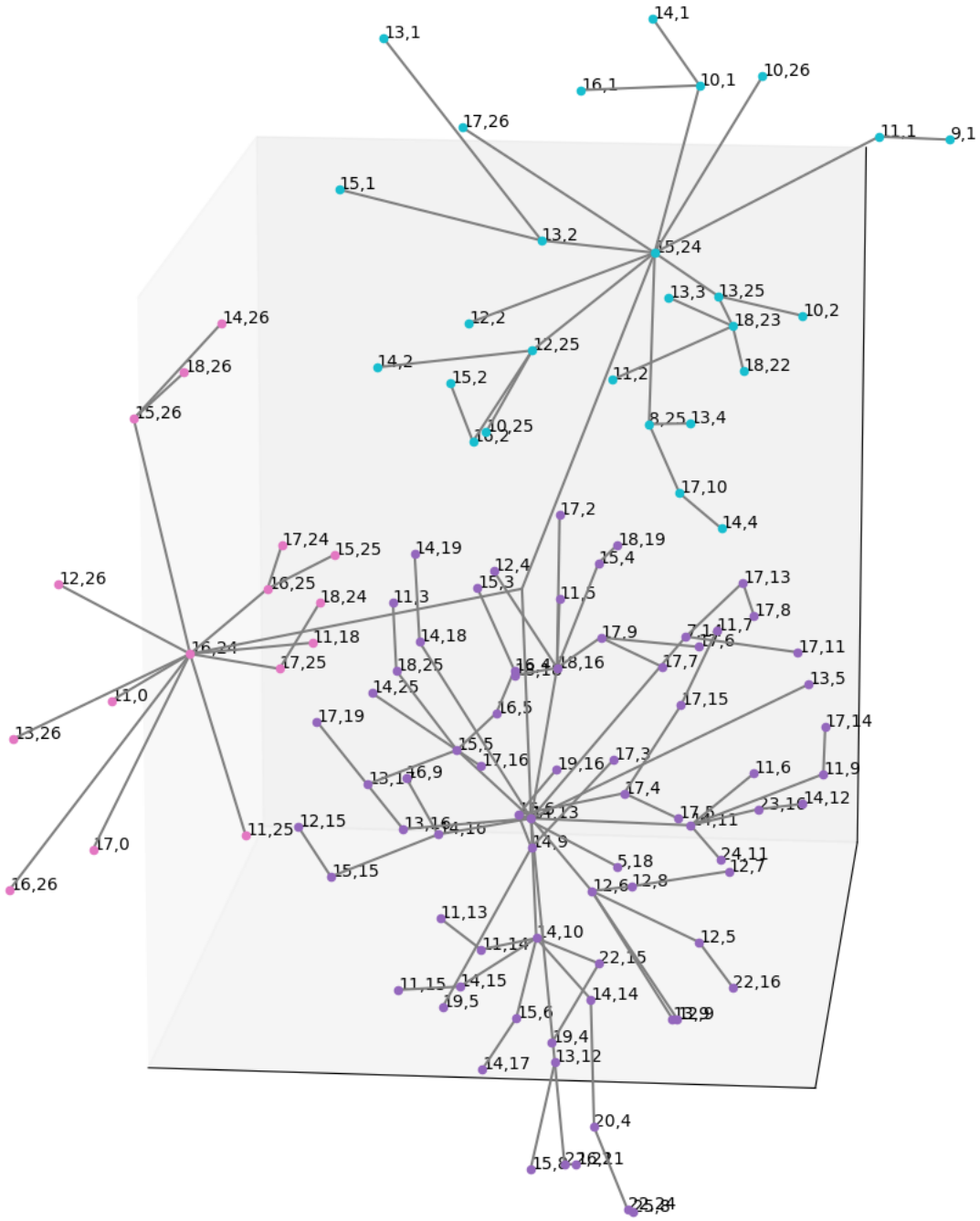


Figure 8.31: Closer view of the dendritic field of Figure 8.29. Compartments 5 (pink), 7 (light blue) and 10 (purple) visible. Labels are the x,y input pixel locations within example images.

8.2.5 Discussion

The FMNIST trials show that a single AD neuron is capable of learning multiple categories better than a single point neuron. This, we believe, is the strongest argument for continued work on dendritic neurons. A single point neuron, on the other hand, excels at encoding a single category. The AD neuron benefits in more complex scenarios from its ability to adjust which connections belong to which compartments. The FMNIST trial made use of completed different categories, but we suggest (for future work) examining how well it performs on variations of one category, where differences are subtle. Using FMNIST as an example, train an AD neuron to recognize just shirt sleeves or the heel of a shoe. We suggest this because asking one neuron to encode an entire category, even one as simple as a shoe, could be asking too much. An advanced use of AD neurons, we predict, would be connecting neurons trained to recognize independent features together to recognize different, say, clothing types.

Our goal for this trial and its variants was to show the performance of the AD neuron compared to the point neuron. Obviously, focusing on accuracy leaves many important questions about AD neuron performance unexplored. Such questions include the impact of the branching factor, compartment size, the size of the dendritic field, learning rates, target rates and error rates are calculated.

We selected FMNIST because it is a standard benchmark for machine learning algorithms. However, in hindsight, it might not be the best benchmark for such single-neuron scenarios. It is our opinion that an understanding of single-neuron learning behavior would be better formed at this early stage by using more basic categories. In general, this reevaluation of benchmarks is part of a larger question about the maximum computational capacity of an AD neuron of a certain size. Therefore, testing a particular configuration of an AD neuron would be best served by data sets which fall within a neuron's computational range. We suspect that future AD neuron benchmarking will

require a general reassessment of which data sets are best suitable to study which parts, e.g., compartmentalization, tree-building, etc.

8.3 Trials Summary

This chapter described two extended trials of AD neuron performance compared to a comparable point neuron. In the first trial, an AD neuron was tasked with learning to differentiate vertical and horizontal movement through a simulated visual field. The AD neuron was able to learn the task by separating and ordering neural inputs along different dendrites. In the second trial, we showed that a single AD neuron is able to learn multiple FMNIST categories better than the point neuron. The results of both trials suggest that there are benefits to the AD neuron. These benefits seem to stem from the AD neuron's ability to divide its set of inputs into different compartments. This gives the neuron quasi-independent places of computation.

The next and final chapter will look back and give an assessment of our work and the AD neuron. More importantly, it will lay out our vision of future work and ideas for continuing this research.

CHAPTER 9

ASSESSMENT AND FUTURE WORK

This work proposed an artificial dendritic neuron model, an implementation, and a method to train the implementation.

Inspiration for the AD neuron came from the characterization of the dendrite by the neuroscience community as a computational workhorse. The impetus for this undertaking was the dearth of dendritic work by the AI community despite the successes of neural network-based solutions. By contributing a dendritic model and a way to train it we hope to create new interest in more complex, dynamic artificial neuron models.

The AD neuron model consists of two fundamental components: compartments and connections. Compartments are similar to traditional point neurons. They combine a subset of neural inputs and transform the combined signal using an activation function. Compartments are linked together into dendrites using connections. Connections use radial basis functions to model the effect of the dendrite on signals moving between compartments. Model behavior was characterized through a series of static demonstrations using a simplified implementation. These demonstrations highlighted the effects of different AD neuron configurations. We showed that AD neurons are capable of replicating any Boolean expression.

A training method was proposed for the AD neuron. The training method consisted of a suite of algorithms. The training algorithm first positions neural inputs within the dendritic field. Using these positions, neural inputs are linked together by connections to form a dendritic tree. The dendritic tree is then compartmentalized. Lastly, weights are modified based on the activity of each compartment. The training method was tested using tasks which targeted each individual algorithm. Finally, the trainable AD neuron was given two trials to test its classification capability versus the point neuron.

In this chapter, we assess the work and discuss several challenges, and we conclude by discussing alternate implementations and future research.

9.1 Contributions

The central contribution of this work is a complex neuron model with fully trainable dendrites. To our knowledge this is the first such neuron model to come out of either AI or neuroscience. We demonstrated that training dendrites is possible using only task data. Work using our model showed that trained dendrites allow single AD neurons to perform tasks beyond the capability of a point neuron. It also showed that trained dendrites increase the number of categories a single neuron can learn.

More generally, we set out to convince the community that AI should earnestly take up research into complex neuron models by showing a glimpse of their potential computational benefits. Research into complex neuron models is inevitable if we are to sidestep the current limitations of ANN solutions.

In the short term, we do not expect that the AD neuron model will revolutionize neural network-based machine learning. There are many hurdles to overcome to make dynamic dendrites feasible in real-world use cases given available tools. First among the hurdles is the additional complexity which prevents the use of current neural network-related tools and hardware. Reliance on CPU-based computation slows down learning dramatically. AD neurons require more information than point neurons. This reduces any potential parallelization achievable on hardware such as GPUs that have per core memory constraints [163]. Even without memory concerns, the dynamic connectivity of the AD neuron poses a challenge. At a minimum, each AD neuron would require its own dynamic connection and weight matrices. We believe these are not insurmountable issues but they will require changes to how we effect training.

Next, it is likely that efficient, optimal solutions will require multiple dendritic and adendritic models. Heterogeneous networks will require a knowledge of best uses for each

which will increase the difficulty of network design. AD neurons will not make our jobs easier. To fully realize complex neuron models will require a refocusing on simple tasks more characteristic of ANN's early days than the current ones being tackled by mature ANN research.

However, parts of the AD neuron model could be useful in the short term. The AD neuron model itself is a composite of several algorithms and design choices. Interest in dendritic neuron models is growing. Specifically, related work has repeatedly signalled that dendrites are a potential source for additional network depth. The AD neuron demonstrates several properties of this additional depth which are distinct from the depth provided by layers. *Shape matters.* Over the short term, this work might inspire related work to experiment with a greater variety of micro-topologies even if the shapes themselves are static.

It is also possible that the neural input positioning algorithm could be useful as a machine learning tool. For example, it could be used to help with dimensionality reduction. Dimensionality reduction is the process of excluding entire features of the input data during training [77]. The input positioning algorithm encodes the n-dimensional relationships in the input data into the reduced dimensions of a dendritic field. As part of a preprocessing stage, this could help to expose irrelevant features or other feature relationships that might inform network construction.

Our work contributes to the need for an expansion of ANN terminology with respect to network components (see Section 9.2.5). Currently, *neuron* denotes the smallest component of a neural network. The AD neuron successfully uses point neurons to model segments of the dendrite. We believe that point neurons should be renamed *compartments* and that the term *neuron* should refer to a set of connected compartments. We believe this change is more than superficial. It allows us to think of layers as deep networks.

The dendritic model provides a way to inject network complexity in the form of sparsely connected trees at a micro level. Adding complexity at a micro level means we can

still treat networks on a macro scale as static entities. We can initialize a network as a collection of layers and each layer can be instantiated as a set of homogeneous neurons. The bounds of the dendritic field encapsulate network heterogeneity and the ability to train it.

For the long term, this work has the potential to direct the development of tools and hardware to facilitate future research into complex neurons. The point neuron model, the Perceptron, existed for many decades before the current revolution. This change was largely brought on by tools and hardware capable of generating and processing large networks. During the life-cycle of this work, neuromorphic chips--a hardware embodiment of spiking neurons and local memory--have been developed. While the AD neuron model and neuromorphic chips have several common themes, such as local memory, neuromorphic chips rely on point neuron models for computation. Our hope is that research into dendritic neuron models can aid in the development of a second generation of neuromorphic technology which incorporates more of what we know about the substrate for biological intelligence.

Finally, explainability in neural networks, for both AI and neuroscience, is a problem that stems from the complex topologies required to perform human-level tasks. A potential long term benefit of work on trainable artificial dendrites is an improved understanding of what dendritic shape contributes to neural computation.

9.2 Assessment

This section assesses each part of the AD neuron. We summarize the capabilities and limitations. Also, potential usefulness to neuroscience will be touched upon. This section focuses also on development because the usefulness of this work is not only in the final tool. Tool building is as important as the tool. We will walk through the more important design choices that went into making the AD neuron in the hopes that it will aid in understanding the many choices that went into the model as well as suggest alternate configurations to be explored.

9.2.1 Compartmentalization

The AD neuron demonstrates an ability to learn tasks which are beyond the capability of a similar (e.g., spiking) point neuron. The AD neuron does not require more information than the point neuron; they both receive the same number of inputs. The AD neuron raises its computational ability by combining subsets of inputs prior to making a final, somatic response, i.e., by compartmentalization. Since the AD neuron uses a multi-dimensional space to situate and link compartments, it is important to look at patterns of compartmentalization and ask which, if any, are beneficial.

More specifically, the AD neuron derives its increased computational range from independent or parallel compartmentalization. This form of compartmentalization forms separate dendritic trees, branches or paths (see tasks in Sections 8.1 and 7.3.2). Success in these types of tasks rely on the useful positioning of neural inputs within the dendritic field such that they form multiple, independent sets of inputs. Our work suggests that independent compartmentalization gives the AD neuron the ability to respond in different ways to multiple patterns (Section 8.2). This idea mirrors the theory that the width of an ANN allows the network to create, through training, independent subnetworks to perform isolated transformations. But these same tasks (Section 8.1 for example) suggest that a perfect positioning or compartmentalization of neural inputs is not always necessary. If the patterns of a task are identical, one branch might suffice. Where and the degree to which branching occurs could be taken as an indication of pattern complexity.

Our research suggests that dependent, or repeated, compartmentalization benefits tasks of a compositional nature. Dependent compartments are chained together to form a single dendritic path. In the movement trial, the order of compartments along a dendritic tree capture the order of cell activation. Without the ability to train such orderings, the trial would fail. Calibration tasks (Section 7.4.1) suggest paths composed of multiple compartments alter signals arriving at different compartments in different ways.

Ultimately, the computational role of compartmentalization is likely task dependent. Two tasks might require that the same information is filtered in different ways. How to train different filters for the same information is not addressed directly by our research.

Compartmentalization has another potential benefit. Compartments divide the input space into quasi-independent sets of input dimensions. An input dimension is just a single input to the neuron. We have added the word 'dimension' here to connect compartmentalization to a reduction of the 'curse of dimensionality' [15, 20], which describes the fact that the number of input examples needed to adequately describe an input space grows exponentially with the number of dimensions (i.e. inputs). An adequate description of an input space is not defined here as it is determined by the problem and measures of acceptability of any given solution. How do compartments reduce the curse of dimensionality?

Creating separate input spaces means that some input dimensions do not directly interact with each other. For example, if a point neuron has ten inputs and each input can take on 2^{64} values, then the size of the input space is $(2^{64})^{10}$ or 2^{640} possible input examples. Now let us assume that the same neuron partitions these ten inputs into two compartments of five inputs each. The input space is now $2[(2^{64})^5]$ or 2^{321}. This off-the-cuff estimation does not take into account the additional complexities of the AD neuron, such as the shape of the tree. If we assume that the two compartments connect in a chain, then the input space grows to $(2^{64})^5$ for the distal compartment and $(2^{64})^6$ for the proximal one since it also receives the output of the former. In the worst case, each input is a compartment and all ten compartments impinge directly on the S-compartment which puts the input space back in the realm of $(2^{64})^{10}$. The impetus for our work does not depend on this argument and we have done no work in this area. It is mentioned here because it might be an area of interest.

From the start of this work, we envisioned compartments as point neurons in order to reduce the complexity of the artificial dendritic neuron. This choice seemed like a natural step beyond the point neuron model.

However, the process of creating compartments was not clear from the beginning. It required dividing the input space into discrete subsets. The evolution of two aspects of compartmentalization during the course of this work show how we worked through this problem.

First, the process of creating compartments from subsets of neural inputs was not immediately clear. We considered two approaches which established opposing dependencies in the training algorithm: tree-first or compartment-first approach. In the compartment-first approach, compartments are created based solely on input positions within the dendritic field. This approach mirrors traditional methods of clustering, such as finding centroids or training weights in a self-organizing map, since it requires identifying sets of related points (or features) based on their location in some N-dimensional space. The compartment-first approach potentially could experiment with the myriad of clustering algorithms and therefore was initially favored. In the tree-first approach, inputs themselves would be used to build the dendritic tree based on their positions. Compartmentalization would follow using as its metric, not unfettered distances across the dendritic field, but distances based on connections and paths in the tree. Compartmentalization, then, would amount to a reduction in the size of the tree by combining nodes while maintaining some properties of the underlying connectivity.

Only once we considered both approaches did we realize that tree-building and compartmentalization together created a form of hierarchical clustering. Hierarchical clustering is the process of clustering in which new clusters are created by dividing or combining clusters identified in a previous iteration. By treating tree-building and compartmentalization as two parts of one larger process (the creation of the dendritic tree), we could tighten their relationship. For example, the metric used to connect neural inputs

during tree-building would be the same metric used to measure an input's relationship to its compartment. Relationships between compartments would use relationships between inputs as connected by the tree.

Another serious consideration was that compartmentalization was not a one time process. An AD neuron would need to be re-compartmentalized many times during the training process. Each re-compartmentalization would require a new tree. Clustering algorithms themselves require some form of training or repetition. The danger of drastically increasing AD neuron training time steered us away from this approach. We were also determined to minimize which aspects of the AD neuron were trained and which were created deterministically based on trained aspects.

We chose the tree-first approach because it allowed us to integrate the two algorithms such that compartmentalization is a constant amount of additional work in each step of tree-building. Input positioning makes certain assumptions about the nearness of neural inputs and we did not want to layer on more assumptions through clustering. Also, we already had determined to use the tree-building algorithm provided by previous neuroscience research [39]. Altogether, the tree-first approach seemed to make fewer assumptions about how inputs should be combined and connected along the dendrite.

Next, we needed to determine how to divide the tree into different compartments. The most obvious way was to start at the S-compartment since it, as the source of neural output, must be a compartment. Working outward from the S-compartment, our algorithm could choose to add an input to the current compartment or create a new one. Without hints that a more complex heuristic would improve the AD neuron's performance, we strove to keep the algorithm as simple, time-wise, as possible. Compartment lengths were measured from the first input added to a new compartment. This prevented the tree-building and compartmentalization algorithms from having to reconsider previously linked inputs under a more complex compartmentalization metric. One compartment ended when its length reached some predetermined maximum.

The second aspect of compartments that evolved during this work was the compartment-level response. Although we determined to represent compartments as point neurons, we were uncertain whether branch compartments should behave like somatic compartments. If they spiked, was that their only output, or did some B-compartment input "leak" into subsequent B-compartments without a spike?

Early on we tested an alternative design which added a decay rate to connections. Even if a compartment did not spike, some amount of its input would reach the next compartment. The rate of decay was a function of the separation between compartments. Since AD neuron compartments are primarily based on spiking point neurons, the forward leak of compartmental input would be an addition specific to the AD neuron. Having no quantitative or background reason to include or exclude it from our initial model, we opted to exclude it. Once this simpler version was tested and better understood, as part of future work, we could revisit the idea.

As to whether B-compartments behaved like S-compartments, we took the middle road. First, we added to the model the ability to create different types of compartments which could be distinguished by their implementation of ψ, the output function. This allowed us to diversify compartment responses if we felt a scenario required it without revising the model. Second, for the initial implementation, we did not vary ψ. All compartments, somatic or branch, behaved the same. This, again, seemed to make the fewest assumptions.

9.2.2 Tree-Building

The tree-building algorithm is built around a modified minimum spanning tree that includes a weighted overall path length when adding new neural inputs to the tree. It was chosen for its simplicity and because neuroscience research identified it as being capable of generating many types of dendrites. Throughout this research we did not consider additional methods; therefore, it is difficult to assess the algorithm in the light of alternative ideas. In some scenarios (Section 8.1) where patterns created isolated clusters

of inputs, it perform well in connecting inputs such that compartmentalization captured the patterns. However, for denser clusters, trees did not always create natural bounds between compartments.

The only dial in the tree-building algorithm is the branching factor, which indirectly affects the depth of the dendritic tree. If it is low, dendrites become long, winding from input-to-input, with relatively few branches. The depth of the neuron increases as there are fewer independent, parallel compartments. Radial position becomes very important for one or two inputs and irrelevant to the rest. Only one or two inputs will connect directly to the S-compartment. The rest will be linked by connections that wrap around the dendritic field. Most inputs will be too far from the S-compartment to have any impact. If the branching factor is too high, every input connects directly to the S-compartment. Depending on the compartment length, such neurons can devolve into a point neuron (all inputs belong to one compartment).

However, in between extremes, sensitivity to the branching factor, or its ability to impact performance, is task dependent. Simple patterns in the input space can be captured by a wide range of branching factors. All values above the range produce similar results. All values below the range also produce similar results. It is the size of this Goldilocks range which is task dependent. Also, based on our experience, extreme precision is not necessary. We did not need to select values of greater precision than a tenth to see a useful result, and often shifting the branching factor up or down by a tenth did not catastrophically alter results.

In future work we make several suggestions for alternate tree-building algorithms (Section 9.3.4). We also suggest that tree-building and compartmentalization could have their associated parameters trained as part of a second stage of training (Section 9.3.3).

9.2.3 Input Positioning

Angular and radial position together give the position of a neural input within the dendritic field. The relative timing of input signals determines the angular distance between two neural inputs. And the relative timing of an input signal and S-compartment output determines the radial position of a neural input. These two components form an input's position and are used to inform the tree-build and compartmentalization algorithms.

Using a Hebbian-like learning algorithm to arrange neural inputs in order to construct dendrites is novel. Generally, Hebbian learning uses the relative timing of events (e.g., spikes) to increase or decrease the connection between neurons. For example, Hebbian learning is typically used to implement synaptic learning. From a neuroscience perspective, realizing dendritic shapes through Hebbian learning seemingly ignores genetic contributions during neurogenesis. However, our algorithm, even conceptually, is not growing dendrites in the sense that it is picking and choosing which neural spaces to invade through dendritic growth. The inputs to the AD neuron are set. At the micro-level, dendritic spines, small extensions from the dendrite proper, which house the synapses, are thought to grow and shrink due to activity. When we consider that the AD neuron's set of inputs is fixed, our algorithm is not determining which neurons among all to make connections to. It is determining how and where the connections are made. One could argue that the AD neuron is not growing dendrites but shaping them: a task somewhere between dendritic morphology and spinal growth.

In simple scenarios where the number of neural inputs are few and the variance of the input signals between examples is minimal, the positioning algorithm performs well in capturing their relationships. Patterns in the input space form recognizable input clusters in the dendritic field. Multiple patterns are more easily separated by either negative error values or a random mask. Such clusters are easily captured and separated by the tree-building algorithm. In these cases, the AD neuron can learn to recognize different patterns without having to adjust weights.

However, in more complex scenarios, where the number of neural inputs is large and examples vary significantly, the position algorithm tends to cluster all active inputs into one large group. While there does appear to be meaning in the location of individual inputs within such large clusters, the tree-building algorithm struggles to capture it (see Section 8.2). This assessment is not definitive since identifying meaning in large clusters of trained inputs is difficult. This suggests to us that immediate future work should limit the number of inputs or spread inputs across several AD neurons. Our reasoning for this comes from our belief that explainability in these early stages of research is important if we are to characterize the contributions of dendrites.

Another reason is based on what could be a limitation of our current algorithm. As we demonstrated in early calibration tasks, the larger a cluster of neural inputs becomes, the more imperfect each input's angular position is likely to be. This is due to the fact that each input in the cluster has a relationship to all inputs in the cluster. Angular positions are 2-dimensional, and a 2-dimensional encoding of N inputs, when N is larger than 2, is potentially impossible. Smaller numbers of inputs allow for more movement within the dendritic field and reduce the chance that an input becomes blocked by or locked within a cluster.

Of course, limiting the AD neuron to a maximum number of inputs because of training limitations is an unsatisfying suggestion, especially when we consider that biological neurons can have tens of thousands of inputs. What can be done to reduce or eliminate the underlying problem? An N+1-dimensional dendritic field (N neural inputs plus the soma) would solve the problem but, again, this is unsatisfying since the biological brain exists in only 3-dimensions.

As we previously stated, compartmentalization itself reduces dimensions. If we assume there is some perfect N-dimensional encoding of a data set in the dendritic field which is used to produce k compartments, so long as the 2-dimensional encoding produces the same k compartments and connects them in the same order, we theorize that the two will be

functionally equivalent within some margin e. The margin would be based on the input-input differences in distance between the perfect and imperfect encoding as a function of the frequency with which each participates in the generation of output, or their radial position. If this measurement of imperfection is correct, then we must only reduce e until the AD neuron becomes useful for a given scenario. In a following section, we discuss alternate designs for the dendritic field and angular positions which might alleviate this issue.

Training and encoding input positions went through several major revisions early in this work. Positioning is the oldest part of the work. Initially, input positions were encoded and trained using Cartesian coordinates in $\mathbb{R}^3$. Changes to input positions used straight-line displacements. Several scenarios were identified in which this method inadvertently created spatial relationships not present in the input data. For example, imagine two neural inputs whose initial positions are on opposite poles of the dendritic field. Any temporally correlated input would cause them to move toward each other and the center of the dendritic field, altering their relationship to the S-compartment in the process.

At the time, the positioning algorithm was less sophisticated. If two inputs received signals within the learning window, they moved toward each other as a function of how temporally close their signals were. While the current algorithm shifts input positions toward a spatial encoding of signal temporal proximity, the older version simply moved one input toward another. The result was that if two inputs frequently received signals at or near the same time, they moved ever closer together. This runaway behavior caused inputs to have only two types of positional relationships to each other in the dendritic field: far away or tightly packed.

Because of these two issues, we decided to decouple input-input and input-output relationships by converting the dendritic field to spherical geometry. This allowed us to use angular displacements to model the former and radial position to model the latter.

Positioning was changed such that neural inputs moved toward a spatial distance which encoded the temporal distance between received signals.

While these changes improved the results of the AD neuron, we do not suggest that the earlier implementation has no merit. In fact, as future work, we believe a return to the Cartesian system might be useful. Ultimately, the spherical dendritic field was selected to disassociate input-input relationships from affecting input-output relationships (and vice versa); however, we suspect that the muddling of relationships were, in part, exacerbated by the earlier positioning algorithm. Under the new algorithm, disruptions to relationships caused by input or output related movement should be eventually corrected. For example, even if two inputs on opposite sides of the field initially move toward the soma due to input, any correlation of either's input to output would cause both to move to a proper distance from the soma over time. It is possible this aberrant intermediate stage could slow or disrupt learning; however, it might be made moot by a decrease in training time overall. Also, movement and distance calculations using spherical geometry are computationally expensive compared to the Cartesian system. A profile of our implementation suggests that the training algorithm spends a significant amount of time calculating the angular distance between neural inputs. While switching could increase the number of epochs needed, this might be offset by a decrease in the CPU time needed for each epoch.

9.2.4 Complexity

The AD neuron is more spatially and temporally complex than the perceptron. This could hamper its use in large networks. In this section we estimate the AD neuron's complexity and discuss its impact.

9.2.4.1 Space

The number of connections in an AD neuron is twice the size of its inputs, or $2N$. There are N incoming connections (one per input) and potentially N connections to build the dendritic tree. Each connection contains b, w, v_a and v_e. The trainable implementation

of the model also assigns a three-dimensional position to each input and a compartmental identification number. The tree requires another N variables if it is encoded using an adjacency list. If each of these variables is stored using 64bits, then for an AD neuron with N inputs, it requires $104N$ bytes. Without considering additional information required by the training algorithms or the memory requirements of the Izhikevich neuron model, an AD neuron with 100 inputs uses more than 10 KB. Compare this to a perceptron which uses 1.6 KB for 100 inputs, or 8 bytes per weight and input value.

The AD neuron's memory footprint is six and half times that of the perceptron (at the very least). To justify this increase in size, future work will need to compare the memory requirements of an ANN with the same capabilities for a particular task. We predict that for some tasks, the overhead will be too much under the current configuration. There are several ways to reduce its size. The model uses input position to calculate v_e and build the tree but once training is complete positions are no longer necessary. Removing them would make the trained model $24N$ bytes smaller. Also, the training algorithm does not include a method for altering b. As in our implementation, a single constant suffices, reducing the model by $16N$. These optimizations reduce the size of the trained model but they still far exceed current network memory requirements and they do nothing for the space requirements during training.

9.2.4.2 Time

The added temporal complexity of the AD neuron is likely more impactful than its additional spatial complexity. The feed-forward pass through the network is uneven, in the sense that certain paths through a network of AD neurons can be much shorter than other paths due to the dynamic nature of the dendritic tree. Uneven paths within a layer of AD neurons could cause delays in updating the next layer. Parallelized versions of the AD neural network could struggle to saturate all processing units due to these delays and the need to wait for all inputs to the next layer. Furthermore, the degree of unevenness is not

static during training. Each AD neuron rebuilds its dendritic tree after each epoch at the very least. This implies the source of delay would be difficult to predict from one pass or epoch to another.

During its learning phase, the AD neuron update method is more complicated than the perceptron's. In addition to weight modification, it must reposition its inputs and, potentially, rebuild its tree. Rebuilding the tree is $\mathcal{O}(N^3)$. Updating input position is $\mathcal{O}(N)$. The calculations involved in our implementation are inherently more complex than those of the perceptron. However, the complexity does depend on the design of the implementation which we noted above.

Time complexity also depends on the somatic model. We used the Izhikevich model which requires updating two state variables each time step to generate spiking behavior. There is one beneficial aspect to spiking models. They greatly reduce the amount of information which needs to move from one AD neuron to another. When the neuron spikes all downstream neurons connected to it receive an updated spike time. When it does not emit a spike, no communication is needed since neurons keep a history of received spikes. Very active neurons (100 Hz) only spike 10% of the time.

9.2.4.3 Detail

Is the AD neuron overly complex? We pause to ask whether our model has advanced beyond a more utilitarian fully trainable model. For example, the AD neuron uses a continuous space to track neural inputs within the dendritic field. Perhaps this level of precision with respect to input position is unnecessary to prove that artificial dendrites are a useful addition to AI's neural toolkit. Simpler methods, such as discrete "partitions" of the dendritic field, might suffice (some are discussed below). However, starting with a more detailed model has aided our ability to consider the necessary level of detail required to create fully trainable dendrites. A detailed first attempt will help make future iterations more efficient.

9.2.4.4 Hyper-parameters

A question arises as to whether the AD neuron is hyperparameter heavy? There are more hyperparameters for AD neurons as we have described them than for point neuron-based ANNs. This is due to the relative complexity of the AD neuron and the larger set of parameters needing to be trained than just weights. Depending on the type of experiment, the neuron requires defining, largely by hand, the branching factor, compartment size, the size of the dendritic field, target spike rates, and learning rates. While this is far from ideal, as discussed in Section 9.3.3, we see a fruitful avenue of future research addressing these also with training, similar to some ANN work [137, 11].

9.2.5 The AD Neuron: Neuron or Network?

The AD Neuron, defined as a connected tree of compartments, blurs the definition of an artificial neuron. What is the line between a neural network and a dendritic neuron? A point neuron is a self-contained, monadic entity. Whereas, the plurality and shape of a dendritic neuron resembles a network. A dendritic neuron with only one compartment is a neuron, but is an AD neuron, which could have hundreds of compartments and an elaborate topology, still a neuron? Assuming a future ability to train elaborate dendrites, it might be hard to draw a circle around a set of compartments and call it a neuron. If we were to zoom out from a network of elaborate AD neurons, we would see a jumble of connected compartments.

The AD neuron challenges the definition of neuron and neural network in that a case can be made that it is one or the other. For example, the AD neuron partitions its set of incoming connections which is not typical of point neurons. On the network side of the argument, its topology as we have described it is limited and its connectivity is sparse. However, the dendritic neuron's computational abilities are somewhere between the point neuron and the ANN in general.

Rather than argue how many independent or quasi-independent compartments a neuron can have before it becomes a network, the dependencies of a training regime are better able to draw the line between neuron and network. Neuronal bounds are given by the limits of its training algorithm. *What learns together belongs together.* No matter how many elements comprise a neuron model, we can circumscribe those compartments whose activity impacts the group, excluding those that do not. Of course, any complex entity, like the AD neuron, can consist of sub-elements which have aspects that train independently, but they must also share a coordinated change. This definition allows single compartments neurons to be still considered neurons.

Although the computational ability of the AD neuron goes beyond the point neuron, the AD neuron is not an ANN because, as we have defined it, its topology is limited. It lacks expandable layers, multiple output channels or the ability to make recurrent connections. All of these have proven to be important to different ANN solutions. Due to these structural limitations, the computational ability of any AD neuron can be bounded by some ANN. Crossover research between neuroscience and AI has suggested that the dendritic neuron has a computational bound similar to a five-to-seven layer ANN [141].

The dendritic neuron, then, should be viewed as containing an intermediate complexity that lies between the point neuron and the network. Rather than come up with a new term, this intermediate complexity should retain the name 'neuron' and demote the point neuron to a 'compartment'.

9.2.6 Dendritic Paths

If compartments recognize sub-patterns within the input space, what, if anything, do dendritic paths contribute? For the AD neuron model, dendritic paths complete our use of compartmentalization by defining relationships between compartments which are constrained by the rules given in Chapter 4.

Each dendritic path forms a unique path through the AD neuron and a chain of dendrites form a path through a network of AD neurons. Recent research suggests that neural networks can be evaluated by the number and nature of the paths through them [138, 42]. Although we have only examined dendritic paths on a micro scale, AD neuron results reinforce the idea that the shape and composition of a path strongly impact learning. An AD neural network diversifies the types of paths (e.g., different lengths) which can exist in a network even beyond those achieved by most regularization methods. Deep networks of multilayer perceptrons are universal function approximators so, certainly, more types of paths do not improve theoretical performance. But, in our estimation, the main current goals of ANN research is (1) to do more with less (reduce network size for a given task) and (2) explain how trained networks make decisions. Therefore, it is likely that path-based analysis can benefit from what is the path-based learning of the AD neuron.

9.2.7 Why Spiking Neurons?

Our work uses a spiking neuron, the Izhikevich neuron, to model compartmental dynamics. We did not set out to use spiking neurons. In fact, our first, static prototypes of dendrites used activation functions common to ANNs, such as hyperbolic tangent. Spiking neurons are less common in AI research due to their discontinuous update function which impedes the use of backpropagation learning. We desired, if possible, to build our training algorithms on research into learning in artificial or biological neural networks. During the design of the input position training algorithm, we were inspired by two descriptions of dendrite modification given by neuroscience.

The process of neurogenesis and dendrite growth certainly depends on genetic information; however, research indicates that activity plays an important role in shaping dendrites [111, 176]. Specifically, activity is thought to play a large role in encouraging the growth of dendritic branches.

On the other hand, dendrites typically do not undergo continual regrowth or massive reshaping during their life cycle. They do, however, make and retract connections to nearby neurons through the growth and atrophy of dendritic spines. Spines are small protrusions from the dendrite which house synapses. Changes to dendritic spines are due to correlated activity in different, nearby neurons.

These two descriptions of dendritic and spinal growth gave us the idea to use presynaptic and postsynaptic activity to determine the location of neural inputs. Steeped in the descriptions of neural activity provided by neuroscience, we first developed a learning algorithm based on the temporal proximity of input and output spikes. This led us to the idea to encode the temporal proximity of activity into a spatial proximity within the dendritic field.

We were concerned with using temporal encoding as it is not common in AI research. However, a rate-based approach appeared to pose several questions absent from the spike-based version. For example, if two neurons providing input to a third each produce an output of 1 (which we take to be the equivalent of 1 Hz for the sake of this argument), should their neural inputs be close together or far apart in the dendritic field? Should they be deep in the dendritic field or near the periphery? Such an approach seemed to have only one interpretation: low with low and high with high. At the time, this seemed like a limited learning rule, so we opted for the spike-based approach, which necessitated the use of spiking neurons.

We selected the Izhikevich neuron because we were familiar with it. To a lesser extent, the Izhikevich neuron is configurable via its parameters, making it capable of producing different spiking behavior. We doubted our research would use multiple configurations but it seemed wise to use a model that had different options.

The use of a spiking neuron model precluded the use of backpropagation to train weights. During initial development, we were not certain we would be able to use backpropagation due to some other dendritic property; therefore, preventing its use did not

alter our choice. Ultimately, the question whether or not a rate-based neuron model could be used to model compartmental dynamics should be revisited as part of future work.

9.2.8 The Zero Problem

Complex systems are often comprised of multiple functions where the output of one is the input of another. Predicting where in the input space such systems exhibit undefined or undesirable behavior can be difficult. It can be equally difficult to put in place systems to avoid such behavior.

Zero output, in a neural network setting, typically means that a neuron is silent. However, in many cases, for the AD neuron, zero output is capable of generating a response in a subsequent compartment due to the way connections transform signals. The zero problem is due to the use of RBFs to model the separation between compartments. Since our work uses spiking neuron models, this is not applicable to our work. However, in a potential future rate-based version of the AD neuron, the zero problem will become a factor in the behavior of compartments. Because of this, we include the following discussion to make future researchers aware of this issue.

The zero problem is caused by the fact that ϕ does not equal zero when its input is zero. For example, two AD neuron compartments, A and B, are connected together such that A's output is B's input (Figure 9.1). Both compartments respond maximally when input equals 1. Looking at the output of A (blue plot), as the input moves away from 1, A's output drops toward zero. A is inactive when its output is zero. The output of A is confined to $[0, 1]$ for all input values.

B's only input comes from A. Therefore, we only need to inspect B's output (green plot) for inputs in $[0, 1]$. Here is the problem. B's output range is limited. B's output will always be in the range $[0.8, 1]$. In other words, due to A's output range and B's expected input of 1, B will never be inactive. In fact, B's output will always be within 20% of its maximum.

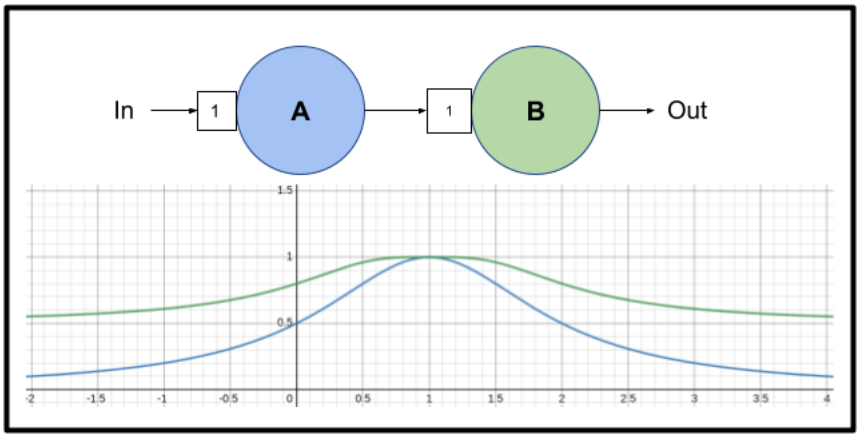


Figure 9.1: An example of the zero problem. Colored circles A and B are AD neuron compartments. Boxes with values are connections with associated values of v_e. The graph shows the output of each compartment. Plot colors match compartment colors.

Why is this a problem? It depends on whether an input of zero should cause a compartment to activate. The scenario depicted here depends on the relationship between the range of ψ and the expected value of ϕ (its v_e). But we can also imagine a desirable scenario in which B should activate only when A is silent (i.e. B's $v_e = 0$).

It is likely that there are two general types of connections: those that carry a positive response to activity and those that carry a negative response. Using compartments A and B, a *positive response* connection activates B when A is active. A *negative response* connection activates B when A is inactive. The zero problem exists only in positive response connections which should respond to minimal input but should be silent on no input.

In general, AD neuron compartments have two states--inactive and active--which correspond to two disjoint unit ranges: 0 and $(0, 1]$.[1] The range $(0, 1]$ contains all active states and 0 is the only inactive state.

The question becomes, how does an AD neuron connection differentiate between very low activity and no activity?

[1]The unit ranges disregard connection modifiers such as weights to simplify our expression of the problem. A version which uses the entire real number line would use three ranges. Active: $(-\infty, 0)$ and $(0, \infty)$. Inactive: 0.

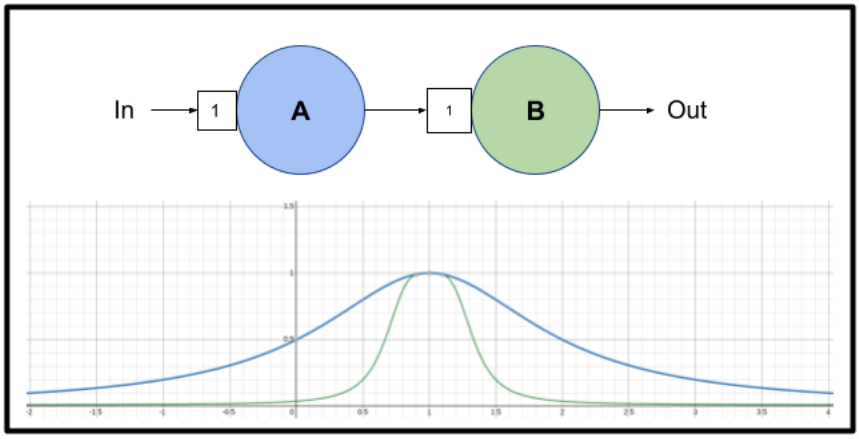


Figure 9.2: The experiment of Figure 9.1 repeated except ϕ in the connection from A-to-B uses $b = 5$.

One solution would be to embrace the discontinuity of a connection's output. We can acknowledge that ϕ, while a continuous function, should follow the discontinuous spirit of two states. The input space is, then, divided into active and inactive sections. Inactivity could be defined as all output equal to zero within some ϵ. And active is all output outside $[0, \pm\epsilon]$.[2] Using this definition of inactivity, ϕ would become a piece-wise function. For example, the inverse quadratic would become:

$$\varphi(x) = \begin{cases} \frac{w}{((v_a - v_e)b)^2+1} & \text{if } \frac{w}{((v_a - v_e)b)^2+1} > \epsilon \\ \infty & \text{if } \frac{w}{((v_a - v_e)b)^2+1} \leq \epsilon \end{cases} \tag{9.1}$$

Another possible solution is to utilize b which controls a connection's activation profile. Liberal use of large values for b would shrink most activation profiles such that only those very close to zero would include zero. Figure 9.2 shows that if the connection between A and B uses a larger b then the output range of B spans most of $[0, 1]$.

A third solution is to scale compartment and connection values away from zero. By multiplying the output of ψ and v_e by a large enough scalar, we can map ranges of

[2]This is the technique used in the experiments of Section 5.7.

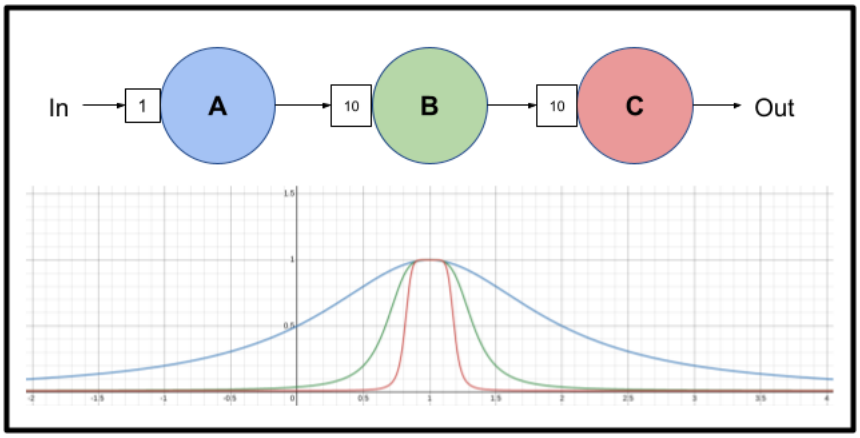


Figure 9.3: An extension of the previous two experiments. An additional compartment, C has been added. The output of ψ and v_e is scaled by the amount seen in the box between connections. The A-to-B scalar is 10. The B-to-C scalar is 10.

activation such that they are less likely to contain zero.[3] This is not a complete solution. Depending on the scalar, very small v_e still include zero.

Figure 9.3 demonstrates the effects of this scalar when $v_e = 1$. The number of compartments has been extended to three to show the cumulative effects of scaling over repeated connections. Scaling causes a narrowing of the range of activation for subsequent compartments. C responds only when A's response is close to its maximum.

While we have called this *the zero problem*, it is actually a problem of thresholds. In the extreme case, a chain of compartments can elevate a near total mismatch to a near total match. Partial matches suffer the same issue. Depending on how the chain of compartments is configured with respect to its parameters x_e, b and w certain partials might be pushed to zero or elevated to the maximum.

Finally, whether or not zero is a problem also depends on the meaning of compartment separation within the context of the task. For example, if separation between compartments signifies the time it takes for a signal to propagate from one compartment to another, then zero is not a problem at all (as in our implementation). In this context, a v_e close to zero means that a signal arrives at downstream compartments with near

[3]An implementation of the AD neuron containing this scalar is not included in this work. The experiment in Figure 9.3 uses a global constant which scales the result of ψ and v_e.

simultaneity because the distance between the compartments is very small. If v_e represents a spike rate, as it does in the above experiments, then perhaps the researcher can estimate its likely range and choose an appropriate scalar.

Again, we have included this discussion to inform future research on a rate-based version.

9.2.9 Calculating the Error

The use of a spiking neuron model complicated the calculation of error values in scenarios where the AD neuron was tasked with determining the category of an input example. One way is to count the number of output spikes within a window of time and use this information to interpret neural behavior. Another method is is to judge output based on the timing of the first spike generated by an input example. We used the former method because some of our trials used time-series. Spikes generated by time series are not guaranteed to fall at the beginning of the series. For example, one example might produce a single, early spike followed by silence; whereas, another might produce a later, dense spike train. We wanted to use roughly the same error calculation throughout.

The number of output spikes generated by an example was stored based on the category of the example, and for the next example of the category, this previous rate was used to construct an error value. An initial version of this project ran each example twice: once to get an output rate and a second time to train. The above method of using the rate of the previous to train the next cut training time in half. With regards to learning, neither our tests nor related literature suggest one method is better than the other. Training time is an issue with the AD neuron and reducing it was our main concern.

In all scenarios, error values were based on comparing the output rate to a target rate. One, this allowed us to task single neurons with learning multiple categories since they could be encoded using different target rates. Two, having a target prevented runaway behavior by bounding changes to the neuron regardless whether its performance was

correct or not. The biggest challenge was in selecting rates which due to input values and parameters the neuron could achieve. If rates were too high, neural properties never settled. If rates were too low, there was the danger of silencing the neuron. Selecting optimal target rates is an avenue of future research.

9.2.10 Order of Update

As noted in the chapter on training, the order of update of neural inputs must be randomized at regular intervals. Without randomization, clusters of neural inputs orbit the dendritic field. This phenomenon is not always present and is most evident in data sets with regular categories, i.e., composed of very similar examples. Without randomization inputs change position in the same order, later moves always take into account the same earlier moves made by other inputs.

The problem with orbiting behavior is that this orbital force is a strong force within the dendritic field because it is a product of regular co-activity of inputs. It is strong in that it can cause a group of inputs to ignore weaker correlations. In other words, intra-cluster correlations in activity disrupt inter-cluster correlations. It cannot be said for certain at this stage whether this orbiting behavior is completely undesirable, but, on the surface, it seems to be disruptive.

Early experiments training input position struggled to produce AD neurons that could solve very simple tasks. We believe this orbiting behavior was the cause because as soon as update order was randomized, results improved dramatically. Learning stabilized.

It is difficult to track the incremental movement of even a few inputs over a long enough span to deduce precisely the cause of one specific type of orbiting behavior. In general, orbiting seemed to be caused by ordered movement creating a leader-follower scenario. Imagine two inputs where one input always moves first and the other, being related in activity, follows it. This leader-follower dynamic is exacerbated when some inputs within the group receive input in a predictable time series.

9.2.11 Stability and Compartments

The previous section discussed one quirk of input position training which affected the stability of learned patterns. Learning in the AD neuron is stable, as we demonstrated in several tasks. Subsequent training does not appear to disrupt learned patterns. However, we are less confident about epoch-to-epoch improvement for complex tasks (Section 8.2). In some tasks, the accuracy rate of the AD neuron jumped significantly. In the later FMNIST trials, these jumps resembled a seesaw effect. Although each positive or negative jump is roughly bounded, this behavior suggests that the neuron is switching between several states.

Across several tasks we tested smaller learning rates, thinking smaller changes might reduce the effect; however, the swings in accuracy were still present in the trained neuron. In fact, for the FMNIST trial, lower rates did not alter the swings.

Cross-referencing examinations of two aspects of the AD neuron--input position (Figure 7.13) and compartmentalization (Figure 8.22a)--we believe compartmentalization is the culprit. Although input positions do not change substantially between epochs, the number of compartments do. The majority of these changes impact small compartments, but larger compartments often break, combine and drift even after many epochs of training.

We believe a prime cause of compartment jostling is the combination of many neural inputs in an oversized dendritic field. In the situation, many inputs cluster in a small angular segment of the field. The angular displacement between inputs is small causing more inputs to be added to fewer clusters. A smaller dendritic field would spread out the same inputs over the entire field. This would potentially create more, smaller compartments whose members are more spread out and less susceptible to jumping compartments due to small positional changes.

Perhaps a completely different approach to compartmentalization (Section 9.3.4) could reduce jostling compartments. Future work should concentrate on stabilizing learning with respect to compartments.

9.2.12 Compartment as Pattern Identifier

In the current model, compartments can contain a single neural input, all inputs to the AD neuron or anything in between. Ideally, connection weights control a compartment's response such that it is in proportion to the presence of some learned pattern. A compartment containing many inputs is far more likely to produce output than one with a few. Does this make sense? In other words, conceptually, should we equate compartment size (i.e., number of neural inputs) with number of patterns or with the number of features in a pattern? Our intuition tells us the latter. Alternatively, the number of patterns recognized by a single compartment could be a function of its size--somewhere between 1 and N, the number of inputs.

We tested several approaches. We averaged input by the number of connections, by a percentage of the number of connections (e.g., 0.5), and by not dividing input at all. The notable change we observed was that scaling input by compartment size had a tendency to create AD neurons which could become silent when large numbers of active inputs were all placed in one compartment. These tests did not convince us that scaling compartmental input produced better results.

Ideally, either a connection's weight, w, or b, the shape property, both proposed by the extended model, could be trained to solve this problem. While we have examined the contribution of connection weights to AD neuron behavior, we have not explored how they contribute to compartment dynamics deeper in the dendritic tree. Future works should explore this question to determine whether connection weights are sufficient to adjust input with respect to a pattern-to-compartment ratio.

9.2.13 Connection Shape Parameter

The extended model contains a shape parameter, b, which allows for the narrowing or widening of the result of ϕ, the connection's output function. This idea was drawn from the different time scales of potentiation in the dendrite (i.e., long plateau potentials). It

was included because we assumed controlling the range of activation around v_e would be useful. However, this work made very little use of b. The complexity of the AD neuron was such that using a single value for many of the tasks and tests sufficed.

Despite its lack of use, it is likely a trainable version of b would prove useful, particularly in a spiking model. b allows the AD neuron to bridge or separate input signals offset in time. It determines how precise compartmental output must be to generate input to the next compartment. It is likely, like in biology [147], different neurons operate at different time scales. A trainable b would allow each AD neuron (or even each connection) in a network to learn its own optimal value.

9.3 Future Work

9.3.1 Next Steps

As we stated earlier, the AD neuron model and training algorithm are substantially more complex than the point neuron and its training algorithm. The dendritic neurons produced by training and their behavior we found difficult to interpret as input complexity increased. To remedy this, we intend to investigate the AD neuron in a simpler setting. The goal of working within a simpler setting is further understand how the AD neuron encodes information and learns to recognize patterns so that the AD neuron itself can be refined.

For example, this work would benefit from more testing using predetermined dendrites to establish a broader understanding of the role of general shapes and configurations. Also, training should focus on continuing with simple tasks like those in Chapter 7. These two steps will be linked by training AD neurons to match the results of the predetermined dendrites similar to the signal coincidence behavior in Section 5.6 and the movement trial in Section 8.1.

We do not intend to abandon the complexity of the current model. A parallel track we intend to take is to break down the AD training algorithm and reevaluate its different components. It is likely there are better solutions to some of the problems. In particular,

we would like to find a better way for the positioning parts of the algorithm and the tree-building parts to interact. The best way to do this is to take nothing for granted and return to the calibration tasks. As part this, we intend to explore a way to expand training to one or more of the hyperparameters.

It is tempting to push forward into more complex scenarios, specifically those involving networks of AD neurons. It is our belief that such an inevitable push will benefit from spending more time trying to understand the AD neuron in simpler, more isolated tasks.

9.3.2 Triplet Learning

Neuroscience modeling research has suggested that synaptic learning is more complex than the simple pre-before-post, or cause-and-effect Hebbian model would indicate [34, 35]. In fact, learning is the result of three coinciding spikes. Using this idea, we could alter the input position training algorithm to combine both angular and radial positions into one update. Currently, coinciding signals arriving at two different neural inputs causes them to adjust their relative angular positions with respect to the temporal interval between the two signals. And coinciding input and output signals do the same for the radial position. An alternative would be to only evaluate angular positions after an S-compartment, or output, spike. In other words, angular position training would require two input signals followed by an output signal.

The reason this change might be helpful is that it would provide some local supervision to angular training. Currently, angular training is the most unsupervised part of the algorithm. All pairs of input signals arriving within the learning window of each other cause a change in angular position. Requiring S-compartment output would allow the algorithm to limit the amount of angular change. It would also allow the algorithm to base angular change on some measure of correctness with respect to current output. While we do not have any hard evidence to support this change, out intuition concerning angular movement is that there is too much of it involving too many neural inputs.

Alternatively, angular change could depend not on S-compartment output but on the output of a local B-compartment. The local B-compartment could be either the compartment to which either is a member or a compartment through which both connect to the S-compartment. This idea suggests that learning depends on local, not global, conditions. However, it presents several complexity issues regarding identifying which compartment activity affects which neural inputs. This is a topic that should be followed up in future research.

9.3.3 Multi-stage Training

The AD neuron trains all aspects of itself--input position, the dendritic tree, compartmentalization and weights--simultaneously. Future work will explore multi-stage training. There are several reasons for this.

In several scenarios, the S-compartment is inactive until angular input positions have been significantly trained. Until neural inputs begin to cluster within the dendritic field, building the dendrite or compartmentalizing inputs can be unnecessary. Also, since weight modification is based on compartment membership, it might make sense to pause weight modification until more stable compartments form. Early, unnecessary changes to weights might have to be undone, increasing training time.

Breaking the training algorithm into multiple steps might allow more of the AD neuron to be trained. For example, in the current version the branching factor and compartment size are selected by hand. If the tree-building algorithm ran offline with respect to input position training, we could incorporate an heuristic that tries multiple values for the branching factor in an attempt to improve results. The same process could be used for compartment size. We did not attempt this due to the increase in training times and because we could find through hand-selection a value that worked.

9.3.4 Discrete Dendritic Field

We are uncertain how much detail is necessary to create beneficial dendritic computation. It is possible that a better first step in expanding our neuron models would be to use a discrete dendritic field. A discrete dendritic field is divided up into predetermined compartments (Figure 9.4 left column). Compartments can form slices or/and layers. Neural input positions could be trained as per the algorithm described in this work. During tree-building, connectivity is determined by radial position only and angular positions, once compartment membership is determined, is ignored (top right). Another possibility (bottom right) is to collapse both angular and radial positions during tree building such that all inputs falling within a compartment connect at the same location. In the examples, precise input positions need to be maintained but play a reduced role in calculating separation.

Using discrete dendritic fields could simplify tree-building and compartmentalization without losing the basic additions of dendritic computation. The tree-building and compartmentalization phases of the training algorithm are the most computationally expensive: $\mathcal{O}(n^3)$, where n is the number of neural inputs. Roughly, a discrete model of the dendritic field would require n operations to determine which compartment an input is current in. Depending on the connectivity scheme--order preserving (top right) or order collapsing (bottom right)--the algorithm would add $n \log n$ (in the worst case, all in one compartment) operations to order the inputs in each compartment.[4] This reduces the complexity to $\mathcal{O}(n \log n + n)$.

Any new method of compartmentalization would need to be tested extensively against the results of this work.

[4]It is best to use the worst case here. Our experiments show that input position training causes many inputs to cluster in small segments of the dendritic field. In the way we have designed the AD neuron, a uniform distribution of neural inputs across the dendritic field and therefore across predetermined compartments only happens at initialization.

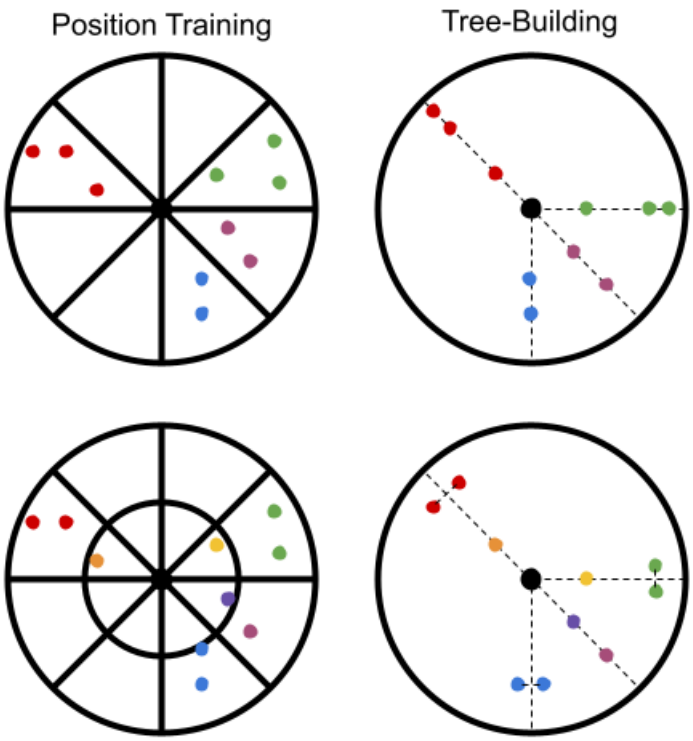


Figure 9.4: Two versions of discrete dendritic fields. Top row: the dendritic field is sliced into eight compartments which extend from the center to the periphery. Bottom row: the dendritic field is sliced into sixteen compartments with an inner and outer set of eight. Left column: neural inputs after positional training but prior to tree-building. Right column: neural inputs after tree-building. Trees are indicated by dotted lines. Colors denote compartment membership.

9.3.5 Alternate Dendritic Field Geometry

Spherical geometry is not the only possible way to define the dendritic field and neural input positions. Earlier versions of the AD neuron of this work used a 2− and 3−dimensional Cartesian coordinate representation of space to track synapse location. Future work will investigate the use of 4-dimensional quaternions which use a single real and three complex values. Quaternions are efficient at modeling rotations of an object across three dimensions. When we consider each neural input within the dendritic field, its angular position is simply a rotation around the field's center. It will be necessary to compare the requirements of the two approaches. It is possible that switching to quaternions could make several aspects of the algorithm more efficient.

Another avenue of future is the use of different field shapes to limit the types of dendrites that can form. For example, the dendrites of some neurons appear limited to a conical field. Others fan out in a 2-dimensional plane. It is likely such recurring shapes have a computational purpose.

9.3.6 Limited Sibling Influence

Currently, an incoming signal to an AD neuron triggers the receiving neural input's position to be compared to the position of *all* sibling input positions with respect to their recent activity. In theory, this method to train input positions is costly; however, if incoming signals are sparse enough, the number of calculations are minimized. A potential way to reduce the number of calculations is to compare each input only to a subset of its siblings. Our present implementation excludes sibling inputs if their most recent activity is beyond the learning window. This limits the number of complex calculations associated with finding distance, heading and movement in a spherical space. Regardless, the number of checks for an input signal is still n, the number of sibling inputs.

Potentially, a more robust way to reduce the computational complexity of the AD neuron is to set a hard limit on the size of influencing sibling inputs. The potential problem in this is identifying (and storing), for each input, the set of influencing siblings without trading one form of complexity for another.

A possible method is to use distance. We could limit influence to all siblings less than d distance from the receiving input. Already, the tree-building algorithm is (almost) computing the all-to-all distances between all neural inputs. This information could be stored with each input to create a list of influencing siblings for use during the next epoch. Another possibility, if discrete dendritic fields were adopted, would be to use only those inputs in one's own or adjacent compartments. Compartmentalization would naturally facilitate a limited number of sibling checks.

Assuming a process of limiting sibling checks is computationally beneficial, such a change could impact the dendrite. During the early stages of development, we omitted siblings during position updates for spatial and temporal reasons. For example, if the distance to another sibling was beyond a spatial encoding of the temporal learning window, it was not checked. It came to light that this caused initial positions to play an oversized role in creating clusters of neural inputs, especially if the learning window was small compared to the extent of the dendritic field. Despite correlated activity in two inputs, they might not cluster unless one or more related siblings occupied the intervening space in the dendritic field. The spatial reason for limiting sibling checks was then abandoned. A potential solution is to include one or more randomly selected siblings in each set of influencing siblings.

Future work on this issue should also investigate whether sparse influence between siblings is beneficial for training as a whole. Perhaps it would be enough to predetermine, for each input, a set of influencing siblings.

9.3.7 Duplicate Connections

A single cortical axon can make many thousands of synapses, but only a handful (<10 for spiny stellate cells) seem to be onto any one neuron [135]. Multiple connections between neurons can take place along the same or different dendrites. It is theorized that duplicate connections strengthen the communication, learning and/or memory between neurons, but their precise function remains unknown [53]. Certainly, a functional analysis of duplicate connections depends on the context. Clustered connections might have a different purpose than disparate ones. Duplicate connections could allow a neuron to integrate the same information, the same variable, in duplicate parts of the overall expression. Under the current training implementation, duplicate connections from the same source would have a very strong tendency to cluster in the same position.

Despite the above, we theorize that in dense dendritic fields, duplicate connections from the same source might help the AD neuron to include the same information in multiple contexts. We are not exactly sure how to achieve this. We spent a significant amount of time on a further extended implementation which added *associativity* to each input (see Section 9.3.8). Input associativity was simply a small static floating point value which caused each input to interact with siblings differently based on how close their associative values were. Experimentation suggested that associativity was capable of causing duplicate connections to belong to different clusters creating different roles for each duplicate.

Duplicate connections increase the complexity of the AD neuron. A simple method to identify unnecessary duplicates could be based on input position. Any duplicates within some range of each other could be declared unnecessary and the number reduced to one. To keep the scope of the project contained, we abandoned this thread of investigation. However, we believe there is potential in duplicate connections.

9.3.8 Input Associativity

Input associativity is the idea that sibling influence during angular position training is not uniform. Some neural inputs exert a stronger influence than others. This idea grew out of tangent research into duplicate connections between AD neurons. Duplicate connections need a mechanism to prevent them from mirroring each other during the training process. Getting duplicates to associate with different groups of inputs has the potential increase the computational complexity of a neuron by allowing the same signal to contribute to different contexts.

We implemented associativity as a randomly selected floating point value from a set of predefined values, e.g., one to ten. Inputs were assigned one of the predefined values using a uniform distribution. Associativity was calculated based on the minimum difference between two values with wrap around. The minimum difference was given as input to a radial basis function with a center of zero. The result was used to scale angular movement.

Correlated activity between less associative inputs formed a weaker bond and vice versa. The effect moderately diversified input clusters. In one learning task (not discussed in this work), we used associativity and fixed weights to train an AD neuron to learn XOR. Associativity was subsequently abandoned due to added complexity. It required additional computational work, a method (and potentially a heuristic) for assigning associations and the selection of additional hyper-parameters, e.g., the number of associative values. More important, without first testing a simpler model, we could not judge its contribution. We believe this is an interesting tangent that should be explored in future work.

9.3.9 Fuzzy Branching Factor

Our work uses a one-size-fits-all approach to the branching factor. In other words, the branching factor is the same throughout the dendritic field. Considering that biological neurons exhibit different degrees of branching at different radii, it is natural to wonder whether the branching factor should be constant throughout the field.

Certain parts of this work have hinted that a given trained AD neuron could be improved if the earlier stages of tree-building used a high bf and the later stages used a low bf. This would encourage a few number of S-compartment rooted trees which on the periphery would expand into highly branching tufts that receive many inputs. Our reason for thinking this is based on qualitative assumptions about neural input clustering. Results might improve if large clusters connect through one S-compartment rooted tree. Different regions of the cluster would form tuft compartments recognizing different subpatterns or alternatives. This is something that should be investigated in future work.

9.3.10 Training Hyper-parameters

We discussed in Section 9.2.4.4 that a potential problem in the model is the number of hyperparameters needed. Future work should investigate whether some or all of these are themselves amenable to training.

A potential method for training the branching factor and compartment size is to run multiple testing phases, each using different values. The best performing value would then be used in the next epoch. A hill-climbing algorithm would be used to select candidate values. On the surface this seems like it would increase the duration of the testing phase by the number of candidate values; however, each value could be tested independently and therefore in parallel. Candidate branching factor and compartment size could be selected and tested simultaneously or they could be staggered across epochs, e.g., even-od epochs.

Finally, we did not present a method to train b, the connection's shape parameter. A potential idea is to change b with respect to a correct or incorrect compartment response. Increasing b has the potential to disassociate incoming spikes which do not arrive simultaneously but at a small offset. Decreasing b has the opposite effect. It would cause more input to a compartment to overlap. It would seem natural to increase b for incorrect responses and decrease b for correct ones. However, a case can be made for increasing b due to correct responses as a way to encourage the AD neuron to further temporally align useful input signals. The answer to this is left for future work.

9.3.11 When to Rebuild?

Our implementation rebuilds the dendritic tree and creates a new set of compartments after each training epoch. While we have tested epochs of different lengths, we have not directly experimented with altering the frequency at which the tree is rebuilt. During subsequent training phases, input positions move away from their previous, end-of-epoch positions used to build the last tree. Many calculations depend on these positions. By the end of a training epoch, the input positions could be radically different from the shape of the tree.

The question that future work needs to answer is: how often should the tree be updated? We have kept updates sparse due to the time-complexity of the algorithm. Perhaps more frequent full, or even partial, rebuilds would be beneficial.

9.3.12 Neural Input Importance

In a previous chapter, we mention a possible role for the *importance* of a neural input. An input's importance could be defined around how vital its information is to creating correct or preventing incorrect neural output. Theoretically, if we could track an input's importance, we could under certain conditions delete it from the AD neuron's set of inputs. Currently, we equate importance with radial position. All neural inputs are initialized to the outermost shell of the dendritic field and they are prevented from moving outside this shell. Therefore, any input which never moves from the outer shell is irrelevant. Otherwise, it has some measure of importance.

Neural importance is key to understanding changes in AD neuron behavior during training. For example, changes to radially-short inputs can have a greater impact than changes to those far from the center of the dendritic field.

Future work should focus on developing one or more heuristics for calculating the importance of a input. An initial version could use the measure described above. An online heuristic could allow for pruning during training and even contributing to other parts of the training algorithm by altering or scaling changes to (un)important neural inputs.

9.4 Final Thoughts

Throughout this work, we have been well aware that each choice we made had many alternatives. Many of those alternatives have been realized only now, at the end. They are part of the contributions of this work. The output of exploration is often learning the questions we need to ask. This chapter was written largely to expose these questions and suggest next steps.

What we label as 'intelligent' is itself a controversial subject. It is the opinion of the author that, despite recent advances, intelligence still belongs only to biology. Mimicking intelligence even with our best hardware is expensive [157]. The 100s of megawatt hours required to train (not to mention run for millions of users) are unsettling at a time when

human activity is driving the world's climate toward extremes. This is just the beginning. If the technology proves useful and not immediately destructive, in the near future there will be thousands of such expensive models being trained and retrained, and their results will be consumed by billions.

We need to do more with less. More than anything, we hope this work encourages the ANN community to work the field as both AI researchers and neuroscientists toward this end. Creating an intelligence outside the natural process will prove to be humanity's worst idea. Let us use these tools primarily to understand ourselves.

Finally, a lesson from my father has been helpful in keeping morale up during this work. And I think it serves as a final contribution. He was a carpenter and liked to say, "If you have to build fifty of something, fifty chairs for example, you will be ready to start building the fifty chairs when you finish the last one." Following this thought, the forward-looking discussion in this chapter might be the biggest contribution of this work. Hopefully others will see the potential in the ideas of this work and where we have made mistakes and take inspiration from their synthesis.

We will only know how to build artificial dendritic neurons after we have designed several dozen. Well, here's one more.

BIBLIOGRAPHY

[1] Larry F Abbott and Frances S Chance. "Drivers and modulators from push-pull and balanced synaptic input". In: *Progress in brain research* 149 (2005), pp. 147–155.

[2] Larry F Abbott and Sacha B Nelson. "Synaptic plasticity: taming the beast". In: *Nature neuroscience* 3.11 (2000), pp. 1178–1183.

[3] Jayant N Acharya et al. "American clinical neurophysiology society guideline 2: guidelines for standard electrode position nomenclature". In: *The Neurodiagnostic Journal* 56.4 (2016), pp. 245–252.

[4] Bijan Afsari, Roberto Tron, and René Vidal. "On the convergence of gradient descent for finding the Riemannian center of mass". In: *SIAM Journal on Control and Optimization* 51.3 (2013), pp. 2230–2260.

[5] Subutai Ahmad and Luiz Scheinkman. "How can we be so dense? The benefits of using highly sparse representations". In: *arXiv preprint arXiv:1903.11257* (2019).

[6] Francisco J Alvarez et al. "Cell-type specific organization of glycine receptor clusters in the mammalian spinal cord". In: *Journal of Comparative Neurology* 379.1 (1997), pp. 150–169.

[7] P Andersen et al. "A comparison of distal and proximal dendritic synapses on CA1 pyramids in guinea-pig hippocampal slices in vitro". In: *The Journal of physiology* 307.1 (1980), pp. 273–299.

[8] Srdjan D Antic et al. "The decade of the dendritic NMDA spike". In: *Journal of neuroscience research* 88.14 (2010), pp. 2991–3001.

[9] Marc Arnaudon and Laurent Miclo. "A stochastic algorithm finding p-means on the circle". In: *Bernoulli* 22.4 (2016), pp. 2237–2300.

[10] Giorgio A Ascoli, Duncan E Donohue, and Maryam Halavi. "NeuroMorpho. Org: a central resource for neuronal morphologies". In: *Journal of Neuroscience* 27.35 (2007), pp. 9247–9251.

[11] Nurshazlyn Mohd Aszemi and PDD Dominic. "Hyperparameter optimization in convolutional neural network using genetic algorithms". In: *International Journal of Advanced Computer Science and Applications* 10.6 (2019).

[12] Lou Beaulieu-Laroche et al. "Enhanced dendritic compartmentalization in human cortical neurons". In: *Cell* 175.3 (2018), pp. 643–651.

[13] Bardia F Behabadi et al. "Location-dependent excitatory synaptic interactions in pyramidal neuron dendrites". In: *PLoS Comput Biol* 8.7 (2012), e1002599.

[14] Eric Temple Bell. "Exponential polynomials". In: *Annals of Mathematics* (1934), pp. 258–277.

[15] 1920-1984 Bellman Richard. *Adaptive control processes: a guided tour. R-350.* English. California: Rand Corp., 1961, 1961.

[16] Yoshua Bengio et al. "Towards biologically plausible deep learning". In: *arXiv preprint arXiv:1502.04156* (2015).

[17] David Beniaguev, Idan Segev, and Michael London. "Single cortical neurons as deep artificial neural networks". In: *Neuron* 109.17 (2021), pp. 2727–2739.

[18] D Harshad Bhatt, Shengxiang Zhang, and Wen-Biao Gan. "Dendritic spine dynamics". In: *Annual review of physiology* 71 (2009), pp. 261–282.

[19] Elie L Bienenstock, Leon N Cooper, and Paul W Munro. "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex". In: *Journal of Neuroscience* 2.1 (1982), pp. 32–48.

[20] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[21] Katie C Bittner et al. "Behavioral time scale synaptic plasticity underlies CA1 place fields". In: *Science* 357.6355 (2017), pp. 1033–1036.

[22] Stephen Blomfield. "Arithmetical operations performed by nerve cells". In: *Brain research* 69.1 (1974), pp. 115–124.

[23] Tiago Branco, Beverley A Clark, and Michael Häusser. "Dendritic discrimination of temporal input sequences in cortical neurons". In: *Science* 329.5999 (2010), pp. 1671–1675.

[24] Tiago Branco and Michael Häusser. "Synaptic integration gradients in single cortical pyramidal cell dendrites". In: *Neuron* 69.5 (2011), pp. 885–892.

[25] Tiago Branco et al. "Local dendritic activity sets release probability at hippocampal synapses". In: *Neuron* 59.3 (2008), pp. 475–485.

[26] David S Broomhead and David Lowe. *Radial basis functions, multi-variable functional interpolation and adaptive networks*. Tech. rep. Royal Signals and Radar Establishment Malvern (United Kingdom), 1988.

[27] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets". In: *Pattern recognition* 36.6 (2003), pp. 1291–1302.

[28] RE Burke, WB Marks, and B Ulfhake. "A parsimonious description of motoneuron dendritic morphology using computer simulation". In: *Journal of Neuroscience* 12.6 (1992), pp. 2403–2416.

[29] Boris Burle et al. "Spatial and temporal resolutions of EEG: Is it really black and white? A scalp current density view". In: *International Journal of Psychophysiology* 97.3 (2015), pp. 210–220.

[30] S Ramon Cajal et al. "Histology of the nervous system of man and vertebrates". In: *History of Neuroscience (Oxford Univ Press, New York)* 6 (1995).

[31] Shih-Kang Chao et al. "Directional pruning of deep neural networks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 13986–13998.

[32] Dmitri B Chklovskii. "Synaptic connectivity and neuronal morphology: two sides of the same coin". In: *Neuron* 43.5 (2004), pp. 609–617.

[33] Andy Clark. "Whatever next? Predictive brains, situated agents, and the future of cognitive science". In: *Behavioral and brain sciences* 36.3 (2013), pp. 181–204.

[34] Claudia Clopath and Wulfram Gerstner. "Voltage and spike timing interact in STDP–a unified model". In: *Frontiers in synaptic neuroscience* 2 (2010), p. 25.

[35] Claudia Clopath et al. "Connectivity reflects coding: a model of voltage-based STDP with homeostasis". In: *Nature neuroscience* 13.3 (2010), p. 344.

[36] M. Cremer. "Zum Kernleiterproblem". In: *Zeitschrift für Biologie* 37 (1899).

[37] Hermann Cuntz, Alexander Borst, and Idan Segev. "Optimization principles of dendritic structure". In: *Theoretical Biology and Medical Modelling* 4.1 (2007), pp. 1–8.

[38] Hermann Cuntz, Michiel W. H. Remme, and Benjamin Torben-Nielsen. *The computing dendrite: from structure to function*. English. Vol. 11. 2014.

[39] Hermann Cuntz et al. "One rule to grow them all: a general theory of neuronal branching and its practical application". In: *PLoS Comput Biol* 6.8 (2010), e1000877.

[40] Hermann Cuntz et al. "The morphological identity of insect dendrites". In: *PLoS Comput Biol* 4.12 (2008), e1000251.

[41] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[42] Dawei Dai et al. "Understanding a deep neural network based on neural-path coding". In: *IEEE Access* 8 (2020), pp. 174495–174506.

[43] Nihad E Daidzic. "Long and short-range air navigation on spherical Earth". In: *International Journal of Aviation, Aeronautics, and Aerospace* 4.1 (2017), p. 2.

[44] Giseli De Sousa et al. "Dendritic morphology predicts pattern recognition performance in multi-compartmental model neurons with and without active conductances". In: *Journal of computational neuroscience* 38.2 (2015), pp. 221–234.

[45] Dominik Dold et al. "Lagrangian dynamics of dendritic microcircuits enables real-time backpropagation of errors". In: *target* 100.1 (2019), p. 2.

[46] *EEG Database*. UCI Machine Learning Repository. 1999.

[47] Gabriele Eichfelder, Thomas Hotz, and Johannes Wieditz. "An algorithm for computing Fréchet means on the sphere". In: *Optimization Letters* 13.7 (2019), pp. 1523–1533.

[48] Ronald AJ van Elburg and Arjen van Ooyen. "Impact of dendritic size and dendritic topology on burst firing in pyramidal cells". In: *PLoS Comput Biol* 6.5 (2010), e1000781.

[49] Ronen Eldan and Ohad Shamir. "The power of depth for feedforward neural networks". In: *Conference on learning theory*. PMLR. 2016, pp. 907–940.

[50] John G Elias. "Spatial-temporal properties of artificial dendritic trees". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 2. IEEE. 1992, pp. 19–26.

[51] John G Elias and Ben Chang. "A generic algorithm for training networks with artificial dendritic trees". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 1. IEEE. 1992, pp. 652–657.

[52] John G Elias, H-H Chu, and Samer M Meshreki. "Silicon implementation of an artificial dendritic tree". In: *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*. Vol. 1. IEEE. 1992, pp. 154–159.

[53] Michael Fauth, Florentin Wörgötter, and Christian Tetzlaff. "The formation of multi-synaptic connections by the interaction of synaptic and structural plasticity and their functional consequences". In: *PLoS computational biology* 11.1 (2015), e1004031.

[54] Lukas Fischer et al. "Dendritic Mechanisms for In Vivo Neural Computations and Behavior". In: *Journal of Neuroscience* 42.45 (2022), pp. 8460–8467.

[55] Clement A Fox and John W Barnard. "A quantitative study of the Purkinje cell dendritic branchlets and their relationship to afferent fibres". In: *Journal of Anatomy* 91.Pt 3 (1957), p. 299.

[56] Luca Franceschini. *Luca Franceschini Blog*. Apr. 2021. URL: `https://lucaf.eu/2021/04/05/mnist-pairwise-one-pixel.html`.

[57] Valerio Francioni and Mark T Harnett. "Rethinking single neuron electrical compartmentalization: dendritic contributions to network computation in vivo". In: *Neuroscience* 489 (2022), pp. 185–199.

[58] Jonathan Frankle and Michael Carbin. "The lottery ticket hypothesis: Finding sparse, trainable neural networks". In: *arXiv preprint arXiv:1803.03635* (2018).

[59] Jonathan Frankle et al. "Linear mode connectivity and the lottery ticket hypothesis". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 3259–3269.

[60] Sonia Gasparini, Michele Migliore, and Jeffrey C Magee. "On the initiation and propagation of dendritic spikes in CA1 pyramidal neurons". In: *Journal of Neuroscience* 24.49 (2004), pp. 11046–11056.

[61] Zong Woo Geem, Joong Hoon Kim, and Gobichettipalayam Vasudevan Loganathan. "A new heuristic optimization algorithm: harmony search". In: *simulation* 76.2 (2001), pp. 60–68.

[62] Oguzhan Gencoglu et al. "HARK Side of Deep Learning–From Grad Student Descent to Automated Machine Learning". In: *arXiv preprint arXiv:1904.07633* (2019).

[63] Marc-Oliver Gewaltig and Markus Diesmann. "NEST (NEural Simulation Tool)". In: *Scholarpedia* 2.4 (2007), p. 1430.

[64] Jordan Guerguiev, Timothy P Lillicrap, and Blake A Richards. "Towards deep learning with segregated dendrites". In: *Elife* 6 (2017), e22901.

[65] Allan T Gulledge, Björn M Kampa, and Greg J Stuart. "Synaptic integration in dendritic trees". In: *Journal of neurobiology* 64.1 (2005), pp. 75–90.

[66] Constance Hammond. *Cellular and molecular neurophysiology*. Academic Press, 2014.

[67] Song Han, Huizi Mao, and William J Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding". In: *arXiv preprint arXiv:1510.00149* (2015).

[68] Kimberly J Harms and Anna Dunaevsky. "Dendritic spine plasticity: looking beyond development". In: *Brain research* 1184 (2007), pp. 65–71.

[69] Demis Hassabis et al. "Neuroscience-inspired artificial intelligence". In: *Neuron* 95.2 (2017), pp. 245–258.

[70] Michael Häusser and Bartlett Mel. "Dendrites: bug or feature?" In: *Current opinion in neurobiology* 13.3 (2003), pp. 372–383.

[71] Michael Häusser, Nelson Spruston, and Greg J Stuart. "Diversity and dynamics of dendritic signaling". In: *Science* 290.5492 (2000), pp. 739–744.

[72] Jeff Hawkins and Subutai Ahmad. "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex". In: *Frontiers in Neural Circuits* 10 (2016), p. 23. ISSN: 1662-5110. DOI: 10.3389/fncir.2016.00023. URL: https://www.frontiersin.org/article/10.3389/fncir.2016.00023.

[73] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[74] Donald Olding Hebb. *The organization of behavior: a neuropsychological theory*. J. Wiley; Chapman & Hall, 1949.

[75] L. Hermann. "Beiträge zur Physiologie und Physik des Nerven". In: *Archiv für die gesamte Physiologie des Menschen und der Tiere* (Aug. 1905).

[76] Bertil Hille. *Ion Channels of Excitable Membranes*. Vol. 18. Jan. 2001, pp. 1–814. ISBN: 0878933212.

[77] Xuan Huang, Lei Wu, and Yinsong Ye. "A review on dimensionality reduction techniques". In: *International Journal of Pattern Recognition and Artificial Intelligence* 33.10 (2019), p. 1950017.

[78] Kevin Lee Hunter, Lawrence Spracklen, and Subutai Ahmad. "Two Sparsities Are Better Than One: Unlocking the Performance Benefits of Sparse-Sparse Networks". In: *arXiv preprint arXiv:2112.13896* (2021).

[79] Shaista Hussain, Shih-Chii Liu, and Arindam Basu. "Improved margin multi-class classification using dendritic neurons with morphological learning". In: *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2014, pp. 2640–2643.

[80] Zachary S Hutchinson. "Spike-time Dependent Feature Clustering." In: *ICAART (3)*. 2022, pp. 188–194.

[81] R Iansek and SJ Redman. "The amplitude, time course and charge of unitary excitatory post-synaptic potentials evoked in spinal motoneurone dendrites". In: *The Journal of physiology* 234.3 (1973), pp. 665–688.

[82] E Izhikevich. *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting*. 2007.

[83] Monika Jadi et al. “Location-dependent effects of inhibition on local spiking in pyramidal neuron dendrites”. In: *PLoS Comput Biol* 8.6 (2012), e1002550.

[84] Ilenna Simone Jones and Konrad Paul Kording. “Can Single Neurons Solve MNIST? The Computational Power of Biological Dendritic Trees”. In: *arXiv preprint arXiv:2009.01269* (2020).

[85] LM Kells, WF Kern, and JR Bland. *Plane and Spherical Trigonometry*. NY: McGraw Hill Book Company Inc., 1940.

[86] Christof Koch and Allan Jones. “Big science, team science, and open science for neuroscience”. In: *Neuron* 92.3 (2016), pp. 612–616.

[87] Christof Koch, Tomaso Poggio, and Vincent Torre. “Nonlinear interactions in a dendritic tree: localization, timing, and role in information processing”. In: *Proceedings of the National Academy of Sciences* 80.9 (1983), pp. 2799–2802.

[88] Christof Koch and Idan Segev. “The role of single neurons in information processing”. In: *Nature neuroscience* 3.11 (2000), pp. 1171–1177.

[89] Matthew E Larkum, J Julius Zhu, and Bert Sakmann. “A new cellular mechanism for coupling inputs arriving at different cortical layers”. In: *Nature* 398.6725 (1999), pp. 338–341.

[90] Yann LeCun, John S Denker, and Sara A Solla. “Optimal brain damage”. In: *Advances in neural information processing systems*. 1990, pp. 598–605.

[91] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[92] Robert Legenstein and Wolfgang Maass. “Branch-specific plasticity enables self-organization of nonlinear computation in single neurons”. In: *Journal of Neuroscience* 31.30 (2011), pp. 10787–10802.

[93] Johannes J Letzkus, Björn M Kampa, and Greg J Stuart. “Learning rules for spike timing-dependent plasticity depend on dendritic synapse location”. In: *Journal of Neuroscience* 26.41 (2006), pp. 10420–10429.

[94] Henry W Lin, Max Tegmark, and David Rolnick. “Why does deep and cheap learning work so well?” In: *Journal of Statistical Physics* 168.6 (2017), pp. 1223–1247.

[95] Bo Liu, Zhengtao Ding, and Chen Lv. “Distributed training for multi-layer neural networks by consensus”. In: *IEEE transactions on neural networks and learning systems* 31.5 (2019), pp. 1771–1778.

[96] Shiwei Liu et al. “Topological insights into sparse neural networks”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2020, pp. 279–294.

[97] Vincent Liu et al. “The utility of sparse representations for control in reinforcement learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 4384–4391.

[98] Rodolfo Llinás et al. “Dendritic spikes and their inhibition in alligator Purkinje cells”. In: *Science* 160.3832 (1968), pp. 1132–1135.

[99] Michael London and Michael Häusser. “Dendritic computation”. In: *Annu. Rev. Neurosci.* 28 (2005), pp. 503–532.

[100] Attila Losonczy, Judit K Makara, and Jeffrey C Magee. “Compartmentalized dendritic plasticity and input feature storage in neurons”. In: *Nature* 452.7186 (2008), pp. 436–441.

[101] Christos Louizos, Max Welling, and Diederik P Kingma. “Learning sparse neural networks through L_0 regularization”. In: *arXiv preprint arXiv:1712.01312* (2017).

[102] Zhou Lu et al. “The expressive power of neural networks: A view from the width”. In: *Advances in neural information processing systems* 30 (2017).

[103] Artur Luczak, Bruce L McNaughton, and Yoshimasa Kubo. “Neurons learn by predicting future activity”. In: *Nature Machine Intelligence* 4.1 (2022), pp. 62–72.

[104] Jeffrey C Magee. “Dendritic integration of excitatory synaptic input”. In: *Nature Reviews Neuroscience* 1.3 (2000), pp. 181–190.

[105] Guy Major, Matthew E. Larkum, and Jackie Schiller. “Active Properties of Neocortical Pyramidal Neuron Dendrites”. In: *Annual Review of Neuroscience* 36.1 (2013). PMID: 23841837, pp. 1–24. DOI: 10.1146/annurev-neuro-062111-150343. URL: https://doi.org/10.1146/annurev-neuro-062111-150343.

[106] Judit K Makara et al. “Experience-dependent compartmentalized dendritic plasticity in rat hippocampal CA1 pyramidal neurons”. In: *Nature neuroscience* 12.12 (2009), p. 1485.

[107] Rahim Mammadli, Felix Wolf, and Ali Jannesari. “The art of getting deep neural networks in shape”. In: *ACM Transactions on Architecture and Code Optimization (TACO)* 15.4 (2019), pp. 1–21.

[108] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. “A history of spike-timing-dependent plasticity”. In: *Frontiers in synaptic neuroscience* 3 (2011), p. 4.

[109] Natalie Matosin et al. *Negativity towards negative results: a discussion of the disconnect between scientific worth and scientific culture*. 2014.

[110] Alex S Mauss et al. “Visual circuits for direction selectivity”. In: *Annual review of neuroscience* 40 (2017), pp. 211–230.

[111] A Kimberley McAllister. “Cellular and molecular mechanisms of dendrite growth”. In: *Cerebral cortex* 10.10 (2000), pp. 963–973.

[112] Bartlett Mel. “The clusteron: toward a simple abstraction for a complex neuron”. In: *Advances in neural information processing systems* 4 (1991).

[113] Bartlett W Mel. “Information processing in dendritic trees”. In: *Neural computation* 6.6 (1994), pp. 1031–1085.

[114] Hrushikesh N Mhaskar and Tomaso Poggio. "Deep vs. shallow networks: An approximation theory perspective". In: *Analysis and Applications* 14.06 (2016), pp. 829–848.

[115] Risto Miikkulainen et al. "Evolving deep neural networks". In: *Artificial intelligence in the age of neural networks and brain computing*. Elsevier, 2019, pp. 293–312.

[116] Kenneth D Miller and David JC MacKay. "The role of constraints in Hebbian learning". In: *Neural computation* 6.1 (1994), pp. 100–126.

[117] Marvin Minsky. *1. Introduction to 'The Society of Mind'*. Youtube. 2011. URL: https://youtu.be/-pb3z2w9gDg?t=4570.

[118] Mehdi Mohammadi and Ala Al-Fuqaha. *BLE RSSI Dataset for Indoor localization and Navigation*. UCI Machine Learning Repository. 2018.

[119] Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. "Topology of Deep Neural Networks." In: *J. Mach. Learn. Res.* 21.184 (2020), pp. 1–40.

[120] Thao Nguyen, Maithra Raghu, and Simon Kornblith. "Do Wide and Deep Networks Learn the Same Things? Uncovering How Neural Network Representations Vary with Width and Depth". In: *CoRR* abs/2010.15327 (2020). arXiv: 2010.15327. URL: https://arxiv.org/abs/2010.15327.

[121] Nicholas Oesch, Thomas Euler, and W Rowland Taylor. "Direction-selective dendritic action potentials in rabbit retina". In: *Neuron* 47.5 (2005), pp. 739–750.

[122] Hysell Oviedo. *Contribution of active dendrites to the transformation of input to output*. New York University, 2004.

[123] Lucy M Palmer et al. "NMDA spikes enhance action potential generation during sensory input". In: *Nature neuroscience* 17.3 (2014), pp. 383–390.

[124] Athanasia Papoutsi et al. "Coding and decoding with dendrites". In: *Journal of Physiology-Paris* 108.1 (2014), pp. 18–27.

[125] Alexandre Payeur, Jean-Claude Béïque, and Richard Naud. "Classes of dendritic information processing". In: *Current Opinion in Neurobiology* 58 (2019). Computational Neuroscience, pp. 78–85. ISSN: 0959-4388.

[126] Panayiota Poirazi and Bartlett W Mel. "Impact of active dendrites and structural plasticity on the memory capacity of neural tissue". In: *Neuron* 29.3 (2001), pp. 779–796.

[127] Alon Polsky, Bartlett W Mel, and Jackie Schiller. "Computational subunits in thin dendrites of pyramidal cells". In: *Nature neuroscience* 7.6 (2004), pp. 621–627.

[128] Ben Poole et al. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems* 29 (2016).

[129] Robert Clay Prim. "Shortest connection networks and some generalizations". In: *The Bell System Technical Journal* 36.6 (1957), pp. 1389–1401.

[130] Xiaoxiao Qian et al. "Evolutionary Dendritic Neural Model for Classification Problems". In: *Complexity* 2020 (2020).

[131] Xiaoxiao Qian et al. “MrDNM: A Novel Mutual Information-Based Dendritic Neuron Model”. In: *Computational Intelligence and Neuroscience* 2019 (2019).

[132] Maithra Raghu et al. “On the expressive power of deep neural networks”. In: *international conference on machine learning*. 2017, pp. 2847–2854.

[133] W. Rall. “Distinguishing theoretical synaptic potentials computed for different soma-dendritic distributions of synaptic input”. English. In: *Journal of neurophysiology* 30.5 (1967), p. 1138. ISSN: 0022-3077.

[134] Blake A Richards and Timothy P Lillicrap. “Dendritic solutions to the credit assignment problem”. In: *Current opinion in neurobiology* 54 (2019), pp. 28–36.

[135] Edmund T Rolls. *Brain computations: what and how*. Oxford University Press, USA, 2021.

[136] João Sacramento et al. *Dendritic cortical microcircuits approximate the backpropagation algorithm*. 2018. arXiv: 1810.11393 [q-bio.NC].

[137] Jakub Safarik et al. “Genetic algorithm for automatic tuning of neural network hyperparameters”. In: *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*. Vol. 10643. SPIE. 2018, pp. 168–174.

[138] Michele Sasdelli, Ian Reid, and Gustavo Carneiro. “Counting the Paths in Deep Neural Networks as a Performance Predictor”. In: (2019).

[139] Jackie Schiller et al. “NMDA spikes in basal dendrites of cortical pyramidal neurons”. In: *Nature* 404.6775 (2000), pp. 285–289.

[140] Philipp Schwartenbeck et al. “Evidence for surprise minimization over value maximization in choice behavior”. In: *Scientific reports* 5.1 (2015), pp. 1–14.

[141] Idan Segev and Egidio D’Angelo. *Brain in the computer: what did I learn from simulating the brain*. 2019. URL: https://www.youtube.com/watch?v=sEiDxti0opE.

[142] Idan Segev and Wilfrid Rall. “Excitable dendrites and spines: earlier theoretical insights elucidate recent direct observations”. In: *Trends in neurosciences* 21.11 (1998), pp. 453–460.

[143] Carla J. Shatz. “The Developing Brain”. In: *Scientific American* 267.3 (1992), pp. 60–67. ISSN: 00368733, 19467087. URL: http://www.jstor.org/stable/24939213.

[144] S Murray Sherman and RW Guillery. “On the actions that one nerve cell can have on another: distinguishing “drivers” from “modulators””. In: *Proceedings of the National Academy of Sciences* 95.12 (1998), pp. 7121–7126.

[145] P Jesper Sjostrom et al. “Dendritic excitability and synaptic plasticity”. In: *Physiological reviews* 88.2 (2008), pp. 769–840.

[146] James Smith. “Space-time algebra: A model for neocortical computation”. In: *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE. 2018, pp. 289–300.

[147] Mehran Spitmaan et al. “Multiple timescales of neural dynamics and integration of task-relevant signals across cortex”. In: *Proceedings of the National Academy of Sciences* 117.36 (2020), pp. 22522–22531.

[148] Emma Strubell, Ananya Ganesh, and Andrew McCallum. “Energy and policy considerations for deep learning in NLP”. In: *arXiv preprint arXiv:1906.02243* (2019).

[149] Greg Stuart, Nelson Spruston, and Michael Häusser. *Dendrites*. Oxford University Press, 2016.

[150] Zheng Tang et al. “A model of the neuron based on dendrite mechanisms”. In: *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)* 84.8 (2001), pp. 11–24.

[151] Matus Telgarsky. “Benefits of depth in neural networks”. In: *Conference on learning theory*. PMLR. 2016, pp. 1517–1539.

[152] Fei Teng and Yuki Todo. “Dendritic Neuron Model and Its Capability of Approximation”. In: *2019 6th International Conference on Systems and Informatics (ICSAI)*. IEEE. 2019, pp. 542–546.

[153] Darcy Wentworth Thompson and D’Arcy W Thompson. *On growth and form*. Vol. 2. Cambridge university press Cambridge, 1942.

[154] Neil C Thompson et al. “Deep Learning’s Diminishing Returns: The Cost of Improvement is Becoming Unsustainable”. In: *IEEE Spectrum* 58.10 (2021), pp. 50–55.

[155] Yuki Todo et al. “Unsupervised learnable neuron model with nonlinear interaction on dendrites”. In: *Neural Networks* 60 (2014), pp. 96–103.

[156] Benjamin Torben-Nielsen and Klaus Stiefel. “An inverse approach for elucidating dendritic function”. In: *Frontiers in computational neuroscience* 4 (2010), p. 128.

[157] Hugo Touvron et al. “Llama: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971* (2023).

[158] Alexandra Tran-Van-Minh et al. “Contribution of sublinear and supralinear dendritic integration to neuronal computations”. In: *Frontiers in cellular neuroscience* 9 (2015), p. 67.

[159] Glen Van Brummelen. *Heavenly mathematics*. Princeton University Press, 2012.

[160] Glen Van Brummelen. *The Mathematics of the Heavens and the Earth*. Princeton University Press, 2021.

[161] Arjen Van Ooyen et al. “The effect of dendritic topology on firing patterns in model neurons”. In: *Network: Computation in neural systems* 13.3 (2002), pp. 311–325.

[162] Rall W. “Theoretical significance of dendritic trees for neuronal input-output relations”. English. In: *Neural theory and modeling: proceedings of the 1962 Ojai symposium*. Ed. by Richard F. Reiss,
United States. Air Force.Office of Scientific Research, and Inc General Precision.
Stanford, Calif: Stanford University Press, 1964.

[163] Linnan Wang et al. "Superneurons: Dynamic GPU memory management for training deep neural networks". In: *Proceedings of the 23rd ACM SIGPLAN symposium on principles and practice of parallel programming*. 2018, pp. 41–53.

[164] Zhe Wang et al. "A Dendritic Neuron Model with Adaptive Synapses Trained by Differential Evolution Algorithm". In: *Computational Intelligence and Neuroscience* 2020 (2020).

[165] Jens P Weber et al. "Location-dependent synaptic plasticity rules by dendritic spine cooperativity". In: *Nature communications* 7.1 (2016), pp. 1–14.

[166] Quan Wen and Dmitri B Chklovskii. "A cost–benefit analysis of neuronal morphology". In: *Journal of neurophysiology* 99.5 (2008), pp. 2320–2328.

[167] Quan Wen et al. "Maximization of the connectivity repertoire as a statistical principle governing the shapes of dendritic arbors". In: *Proceedings of the National Academy of Sciences* 106.30 (2009), pp. 12536–12541.

[168] Wikipedia contributors. *Atan2 — Wikipedia, The Free Encyclopedia*. [Online; accessed 31-December-2021]. 2021. URL: https://en.wikipedia.org/w/index.php?title=Atan2&oldid=1061049709.

[169] Wikipedia contributors. *IBeacon — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-December-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=IBeacon&oldid=1120081090.

[170] Wikipedia contributors. *Metric space — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-June-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Metric_space&oldid=1092197273.

[171] Wikipedia contributors. *Received signal strength indication — Wikipedia, The Free Encyclopedia*. [Online; accessed 14-December-2022]. 2022. URL: https://en.wikipedia.org/w/index.php?title=Received_signal_strength_indication&oldid=1102960892.

[172] Stephen R Williams and Greg J Stuart. "Dependence of EPSP efficacy on synapse location in neocortical pyramidal neurons". In: *Science* 295.5561 (2002), pp. 1907–1910.

[173] Stephen R Williams and Greg J Stuart. "Site independence of EPSP time course is mediated by dendritic I h in neocortical pyramidal neurons". In: *Journal of neurophysiology* 83.5 (2000), pp. 3177–3182.

[174] Stephen R. Williams and Greg J. Stuart. "Role of dendritic synapse location in the control of action potential output". In: *Trends in Neurosciences* 26.3 (2003), pp. 147–154. ISSN: 0166-2236. DOI: https://doi.org/10.1016/S0166-2236(03)00035-3. URL: http://www.sciencedirect.com/science/article/pii/S0166223603000353.

[175] James Wolper. *Understanding mathematics for aircraft navigation*. McGraw Hill Professional, 2001.

[176] Rachel OL Wong and Anirvan Ghosh. "Activity-dependent regulation of dendritic growth and patterning". In: *Nature reviews neuroscience* 3.10 (2002), pp. 803–812.

[177] Xundong Wu et al. “Improved expressivity through dendritic neural networks”. In: *Advances in neural information processing systems*. 2018, pp. 8057–8068.

[178] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[179] Neha Yadav et al. “History of neural networks”. In: *An Introduction to Neural Network Methods for Differential Equations* (2015), pp. 13–15.

[180] Esin Yavuz, James Turner, and Thomas Nowotny. “GeNN: a code generation framework for accelerated brain simulations”. In: *Scientific reports* 6.1 (2016), pp. 1–14.

[181] Friedemann Zenke and Wulfram Gerstner. “Limits to high-speed simulations of spiking neural networks using general-purpose computers”. In: *Front Neuroinform* 8 (2014). DOI: 10.3389/fninf.2014.00076. URL: http://journal.frontiersin.org/Journal/10.3389/fninf.2014.00076/abstract.

[182] Xiao Lei Zhang et al. “Event related potentials during object recognition tasks”. In: *Brain research bulletin* 38.6 (1995), pp. 531–538.

[183] Wenda Zhou et al. “Compressibility and generalization in large-scale deep learning”. In: (2018).

APPENDICES

APPENDIX A

ARCHITECTURE

The software used in this work was largely coded from scratch. This was necessitated by our goal to implement fully trainable dendrites. Prior to coding everything ourselves, we examined several computational neuroscience libraries and tools to see if they would work. We looked to neuroscience rather than AI because computational neuroscience makes frequent use of complex neuron models. None of the tools we investigated supported working with dynamic dendrites. The tools we looked at included Brian2, NEST [63], Spike, GeNN [180], and Auryn [181]. Although no documentation stated that these tools supported dynamic dendrites, it is possible that the tools could be hacked or modded to complete this work. However, in making our first steps, we were not entirely sure what the final implementation would require and were therefore hesitant to invest time into something that would prove insufficient.

Our implementation was coded in Modern C++ (the current build uses C++20). Current version of the code can be found here: Project Radius. Future versions of this project will be available on my Github repository. Project Radius depends on the JSON for Modern C++ to read JSON config files and the C++ implementation of OpenMP for paralellization. Hardware used during the life cycle of this project varied from an older AMD Phenom II 955 to a modern i7-9700. All hardware used was off-the-shelf. Visualization used Python3's matplotlib module. Development was initially on Ubuntu and later on Manjaro (but likely any linux distro will run the code). Windows and Mac versions are not available.

Next, we will briefly describe the implementation and its design. Component names used in the software do not necessarily correspond to those in this text.

At the heart of the simulation is the *Neuron*. Neurons model a single AD neuron and therefore are responsible for keeping track of all neural inputs. During the Neuron's update process each *Neural Input* is checked whether it received a spike. If a Neural Input receives a spike, the angular training algorithm is called on that Neural Input. Next, the neuron queries its dendrites for input. This recursive query is passed back down the dendrite through each compartment until the complete input is calculated. The input is then applied to the S-compartment. If the S-compartment emits a spike, the radial training algorithm is called. Radial training was paralellized using OpenMP. Weight modification can take place during the input query and as a result of an S-compartment spike. Each Neural Input keeps track of its own weight, compartment membership and position within the dendritic field. Compartment membership is stored as an integer identifier. Connectivity between Neural Inputs is stored as a parent and vector of integer identifiers.

Neurons are stored in *Layers*. Layers function exactly like they do in typical ANNs such that the output of one Layer is the input to the next. Apart from defining the flow of information by defining the order of updates, they are simple containers for Neurons. Neuron updates within layers was paralellized using OpenMP. A Connection matrix is used to send spike times from one layer to the next to decouple communication between neurons. Neuron and Input addresses into the matrix are given by three integer identifiers: Layer, Neuron and Input.

Layers are all stored in the *Network*. A Network encapsulates the inner workings of the Layers and Neurons such that Simulation code is similar regardless of the composition of the Network or any requirements of the task itself. Networks are given by a JSON object which defines its composition: Layers, Neurons, connectivity, and certain initial parameters, such as initial radius and weights. Only those parameters required during the Network building phase are in the JSON Network definition.

Task code is broken into an initialization section which loads all necessary resources and generates the Network. Once initialization is finished, the task section runs the required

number of epochs. Each epoch consists of a training, testing and possibly random mask section in this order. The architecture provides for a mechanism to load pre-trained models for further or alternative training sessions; however, this was not utilized extensively. On several occasions trained models were loaded and further trained to verify long term stability.

Serialization is handled by the Writer. Each component of the Network has a data-only structure specific to itself which it populates with information and sends to the Writer. This allows the Writer and Network to function agnostically of each other. While Network state and activity can be serialized as often as required, the work presented in this document serialized the network only during the testing phase. This vastly reduced the data created by the Network, which, in turn, reduced the time it took to generate visualizations for analysis. Very early work, specifically during the creation of the angular training algorithm, reversed this policy and serialized the Network during the training phase. This allowed us to visualize the movement of neural inputs through the dendritic field. However, this produced very large data sets. For example, 1,000 epochs of 100 examples per epoch of a single AD neuron training on the MNIST data set produced 60 GB of data. While this data was useful to prove to ourselves that neural inputs were behaving as expected, it was not feasible for us to store such quantities of data for 100s of runs. Focusing on just the testing phase data, we reduced model data for the same task to just 4-6 GB.

The implementation contains several miscellaneous modules. Several modules were written to handle specific tasks, such as FMNIST. These include code to load the binary data and present it to the Network in several different ways. Input Generators are capable of generating different simple spike patterns using different processes, such as the Poisson process.

APPENDIX B

MISC. CODE

This section contains miscellaneous code used to complete this dissertation.

B.1 Average Interspike Interval Simulator

The following Python3 script was used to simulate the average interspike interval of a given number of inputs each spiking at a given rate. Output of this script was used in Chapter 7.

```
import random
#########################################################
# Finds the average interspike interval for a set of inputs
# with given spike rates. Any ISI greater than the window
# maxT is ignored. Based on the Law of Large Numbers.
#
# Code: zsh
#
# Parameters:
#
# NUM_INPUTS: number of inputs per pattern
# I: dict of spike times per input
# R: spike rates per input
# flags: tracks whether input i has spiked
# dI: interspike intervals
# T: make sim time
# maxT: max ISI. ISI beyond max will not be counted.
#########################################################
NUM_INPUTS = 6
I = {}
R = [0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
flags = []
dI = {}
T = 100000
maxT = 100

# Initialization
for i in range(NUM_INPUTS):
    I[i] = []
    dI[i] = []
```

```
    flags.append(False)

# Run sim
for t in range(T):

    # Reset all flags
    for f in range(len(flags)):
        flags[f] = False

    for i in range(NUM_INPUTS):

        # if there's a spike
        if random.random() < R[i]:
            flags[i] = True
            dAll = 0
            dCount = 0

            # calculate the average of all ISIs less than maxT
            for k in range(NUM_INPUTS):
                if k==i: continue
                if len(I[k]) > 0:
                    d = t-I[k][-1]
                    if d <= maxT:
                        dAll += d
                        dCount += 1
            if dCount > 0:
                dI[i].append(dAll/dCount)


    # If input f had a spike, append the current sim time.
    for f in range(len(flags)):
        if flags[f]:
            I[f].append(t)

# Output
for d in range(len(dI)):
    print(f"{d} average: {sum(dI[d])/len(dI[d])}")
```

BIOGRAPHY OF THE AUTHOR

Zachary Hutchinson was born in Houston, Texas on April 11, 1977 and raised in Wharton County, Texas. He attended Boling Independent School District and, during his senior year, he was a foreign exchange student with the American Field Service to Russia. He attended the University of Houston and graduated in 1999 with a Bachelor of Arts in English Literature and German Language. Later, he attended Hunter College in New York City and graduated in 2014 with a Bachelor of Arts in Computer Science.

Zachary Hutchinson is a candidate for the Doctor of Philosophy degree in Computer Science from the University of Maine in May 2023.