From Random to Organized: The Architecture of Neural Networks During Development

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Avery Morris

Annandale-on-Hudson, New York
December, 2017

Abstract

The brain is constantly changing during development because of various stimuli: memories, language, visual patterns and other sensory information. As a result, networks need to have specific learning rules to function being both plastic and stable. In this project, I've constructed a mathematical model based on a biological neural network. I've written differential equations to describe: a. these learning rules and b. methods of visual input to the network. Using Euler's method, I've created a discrete-time version of this biological phenomenon to implement on the computer. I've successfully coded this, using difference equations in MATLAB to simulate developmental neurons in the retina. This project is at the cutting edge of the computational neuroscience field, particularly, because it remains unknown exactly how the topology of a neural network changes in development and how neurons self-organize from a previously random network of connections. Ultimately, I've found specific ways that synaptic connections evolve over time and my results illustrate the maturation of a previously unrefined network. This gives us insights into how learning takes place during an organism's critical period of development.

Contents

Dedication

I'd like to thank my entire support system of friends, family, advisors and teachers. These include: My grandma Ama for being my biggest role model; My grandfather Papa, who sparked my interest in Mathematics from an early age; My mother who took me to my first ever exhibit at the California Science Center when I was 10 to see real human brains behind a glass screen; My father for always inspiring me with the passion he has for what he does for a living; My sister Alexzandra for being such an amazing listener and for constantly bringing an excitement and unique curiosity to the world and all she does.

To Peter, for always making me laugh and for helping me with Latex bugs. To Aunt Frannie and Uncle Richard for being so generous and for reminding me how important family is. To Georgina for teaching me how brains really work :). To Emily Mackevicius and everyone from the Fee Lab at MIT (Internship 2015) for sparking my interest in computational neuroscience early on. To Andrea Henle and Amy Savage for believing in me as a scientist. To my inspiring violin teachers: Sam, Mr. Treger and Weigang for teaching me how working hard and loving what you're doing pays off. To my Conservatory advisor Frank Corliss for your kindness and meaningful advice these last 5 years. To my math advisor, Lauren Rose for being so supportive of me these last 5 years and for always cheering me up with a Suzuki song!

And finally, to my Senior Project advisors Jim Belk and Arseny Khakhalin. Jim, I am so grateful for all of the hours you have put into meeting with me about this project, as well as for all of the other classes I've taken with you during my 5 years at Bard. I really appreciate how clearly, patiently and thoughtfully you explain everything, and am just in awe of how quickly you think and how creative you are with math. Thank you for being such an inspiration! Arseny, I am so thankful for you for introducing me to this senior project topic as well as helping me develop a background knowledge of the field. Thank you also for answering my questions and for meeting with me even when you were on sabbatical!

Acknowledgments

I would like to acknowledge Stefan for meeting with me regularly to help me learn MATLAB. I'd also like to acknowledge Peter for answering my Latex questions and Brigitte for proofreading. Emma, for constantly cheering me up with music and stories. Labeeby, for listening to me give a run-through of my prospectus talk and always asking for my latest pictures of neural networks. Olivia, for agreeing to work on a Leaky integrate and fire simulation project with me for MATH 314 (Mathematical Modeling) to help me prepare for my senior project. Finally, I'd like to acknowledge my sister Alexzandra for our senior project study sessions and for being my best friend for 23 years.

Introduction

A biological **neural network** is a group of two or more neurons that are connected in a specified fashion. For example, if neuron i is connected to neuron j, then neuron i sends information to neuron j and neuron j sends information to neuron i in the form of electrical charge. A **spike** is when a neuron receives so much information that its voltage exceeds a set **threshold** and immediately resets back down to a resting potential. We can visualize a neural network as a **graph**, in which every pair of neurons is connected to each other, though no neuron is connected to itself.

Neural networks are governed by a set of biological **learning rules**, which keep the network stable, though also allow for the connections between neurons to change. If we expose our network to external **stimuli**, whether it be visual, auditory, tactile, etc., the topology of the network changes.

In the last twenty years in the field of computational neuroscience, researchers found that in **development**, networks use these learning rules to converge from a relatively unrefined state to something more organized, a stable state. These learning rules include: a voltage-based model called **leaky integrate and fire**; a current-based model; a rule called **spike-time-dependent plasticity** in which neurons are rewarded based on how much information they send to their neighbors; and finally **homeostatic plasticity**, which is an umbrella term for the following network normalization techniques: **synaptic scaling** and **changing threshold**, tools to readjust the parameters so that they are receptive to how frequently a neuron is spiking.

Modeling such a network requires converting what we already know about biological neural networks, particularly those networks located in the retina, into a simplified model of **differential equations**. The difficulty with this lies in the fact that our time differential equations only describe continuous network behavior in the absence of spiking. Since spiking is an instantaneous change to an assumed continuous system, we must convert these differential equations to **difference equations** using **Euler's method** to create an accurate simulation of a developing network.

A **dynamical system** is a system whose state, an element of a state space, changes over time and is governed by a particular fixed rule, in our case, the voltage of neurons [1]. In our simplest model, we are looking at a 100-dimensional dynamical system, comprising of the voltages of 100 neurons. In particular our dynamical system is interesting because as mentioned, it exhibits characteristics of both continuous and discrete systems, making it a **hybrid system**.

My goals in this project were to create a **mathematical model** of differential equations describing a spiking neural network that receives visual input, as well as to successfully create a computer simulation of this model in MATLAB, implementing the learning rules above. While it was time my first time using this program, MATLAB turned out to be very beneficial in working with the large matrices and graphs we were using. Beginning with random connections between neurons, my goal was to see how the strengths of connections in the network would change in response to a visual stimulus, after running my simulation for a few days.

Previous research has been done in this area with Xenopus tadpoles, mice, Zebra fish and Zebra finches, particularly observing and modeling neuron behavior during development [2]. Researchers have shown using MATLAB, that with these learning rules, networks can maintain homeostasis and become more refined even to the extent of repairing the loss of ion channels [3].

In my simulation, I found that the connections between neurons changed over time and the majority of them converged to zero. This gave reason to believe that neurons initially reacted much stronger to input as opposed to later on in the simulation. My resulting graphs of connections illustrate that a patterned visual stimulus is enough information for the neurons to in a sense, “figure out” that they are organized in a grid and eventually use this point of reference to organize themselves into a grid. This shows the maturation of a previously random, chaotic network.

In this project, the chapter structure is arranged as follows: In Chapter 1, I will describe the biological aspects of my project such as how neurons send information to each other in the form of **action potential** and I will show differential equations that model this behavior. The particular learning rules I will discuss in this chapter mostly occur in the absence of spiking. These include leaky integrate and fire, a basic way to model the neurons’ behavior; a current-based model, which is a reasonably good model for describing how neurons“communicate” with each other; spike time dependent plasticity, the reward system I spoke about previously; synaptic scaling, a normalization technique used for scaling the sum of inputs to a particular neuron; visual input to the network, in the form of an approaching stimulus or expanding disc; and finally changing thresholds, another technique that normalizes the network so that all neuron spiking approaches a target rate.

In Chapter 2, I will convert all of my differential equations into difference equations as well as explain more of my parameters and the particular functions of my code in MATLAB. In Chapter 3, I will describe the results I mentioned above as well as the final graphs of my simulation, and goals for future work in this field. In the Appendix, I will show my final MATLAB simulation code, as well as that for my input function.

1
Chapter 1: My Model

1.1 Leaky Integrate and Fire

As previously stated, we define a **neural network** as two or more neurons that are connected in a graph, where a neuron represents an **node** and its connection to another neuron represents an **edge**. Assume the following information: neuron i is connected to neuron j and neuron i sends information to neuron j in the form of an electrical charge. This means that neuron i is called the **presynaptic** neuron and neuron j is called the **postsynaptic** neuron. A **synapse** as in the words pre and postsynaptic refers to the connection between both neurons, biologically, in the form of axons and dendrites. In this project, we will casually refer to neurons i and j as "neighbors" in a neural network.

But how exactly do neurons send these signals to each other? **Action potential** describes the process by which, biologically, cells send and receive input, and chemical signals turn into electrical signals. Here's how the process works: Every neuron has a **resting potential**; this is its equilibrium. This value is on average equal to -70 mV meaning that neurons are negatively charged at rest [5]. However, when a stimulus is presented, whether its an electrode, a visual stimulus, a smell, etc., the neurons' sodium channels open and let sodium ions (Na+) from the exterior enter the cell causing it to become more positive. This process is called **depolarization**. The membrane potential increases and the sodium channels inactivate (roughly causing the membrane potential to jump up to +30 mV). This causes the cell's potassium channels to open and positively charged potassium ions (K+) to leave the cell. Next the cell **repolarizes**, as in the voltage lowers and finally **hyperpolarizes**, meaning that the membrane potential decreases so much that it becomes lower than the original resting potential. This causes sodium channels to de-inactivate over time and ionic pumps to restore equilibrium in the interior and exterior of the cell.

But how do we model this mathematically and create a basic "toy model" of signals between neurons: For example, those signals between photoreceptors and bipolar cells and even those between horizontal cells and bipolar cells in the retina. The first step requires constructing a **Leaky integrate and fire model**.

After Louis Lapicque's neuroscience discoveries in the early 20th century in frog nerve stimulation and also later, Bruce Knight and Rick Purple's discoveries, the phrase "leaky integrate and fire" was developed and the model, originally something which was purely circuit-based, was conceived [7]. The meaning of the phrase "leaky integrate and fire" refers to the fact that sometimes a neuron fires and sometimes it "leaks," meaning after the neuron reaches a set **threshold** (i.e. its maximum voltage), its voltage decays and is reset back to the resting potential.

In the field of computational neuroscience, leaky integrate and fire is one of the most common models of an individual spiking neuron. There are various types of leaky integrate and fire models, some very simple and others, more complicated. These include exponential integrate and fire in which voltage decays with the addition of an exponential term, linear integrate and fire and non-linear integrate and fire [8]. In this project, we are using a version of the leaky integrate and fire model, though with the addition of current-based synaptic transmission. However, let's first start describing the standard leaky integrate and fire model.

Here's how the model works: Each neuron, j, has a voltage as well as a resting potential (mV). In my model, the initial voltage value is equal to zero, slightly differing from the biology, where the resting potential is -70 mV. This voltage value decays **exponentially** by a factor of e in

10 ms in the absence of spiking. Broadly speaking, this is similar to how a real neuron might repolarize when potassium channels open. Let's call this decay factor $T_v = .01$ seconds.

Taking account of spiking: when a neighboring neuron, i, spikes, j receives a small amount of **input**: a boost in a neuron's voltage from either its neighboring neuron or an external stimulus. When neuron j receives enough voltage from all neighboring neurons (i included), it surpasses its threshold (which we set as $= 1$), spikes and is reset back to the resting potential. No hyperpolarization occurs in this model, as it does in biology because our goal is to use the simplest model possible and still get optimal results. This process continues and eventually, j's neighboring neurons receive enough voltage to spike.

Our model consists of a network of 100 neurons. The way in which the neurons are connected resembles a **complete graph**: the neurons are assigned to a point on a grid and they are all connected. In biology however, networks are much larger than 100 neurons, as there are roughly 103 million photoreceptors in the human retina [9]. In addition in our model, we are assuming a fact that is not necessarily true in biology. We are assuming that every synapse between neurons is **excitatory**: the signal is probabilistically going to be transferred from postsynaptic neuron to presynaptic neuron successfully, differing from **inhibitory synapses** [10].

Let $\mathbf{v}_{\max}$ be our threshold. When the voltage of a neuron reaches its threshold, $\mathbf{v}_{\max}$, it spikes. When it spikes, it sends signals to neighboring neurons and its voltage resets to 0.

In figure 1.1.1, when the voltage increases, we know that the neuron has received a signal. When the voltage decreases that is due to the nature of exponential decay. The graph represents a neuron that receives multiple signals from neighboring neurons until it eventually gathers so much voltage that it spikes, and then resets to 0.

In our Leaky integrate and fire model, we model purely voltage. We write our exponential decay equation as the following differential equation, in terms of neuron i:

$$\frac{dv_i}{dt} = \frac{-v_i}{\tau_v}$$

Using separation of variables, we see that the solution to our differential equation is:

$$v_i(t) = v_i(0)e^{-t/\tau_v}$$

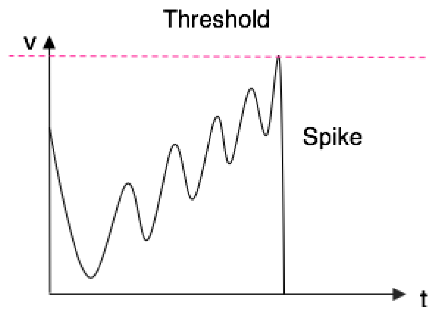

Figure 1.1.1: Neuron j receives small degrees of input from neighboring neurons while simultaneously exponentially decaying. Once the sum of this input exceeds the threshold level, Neuron j spikes, sends input to its neighbors and its voltage is reset back to 0 mV.

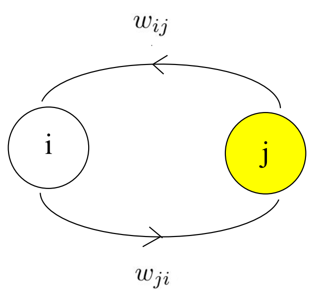

Figure 1.1.2: Neuron j receives small degrees of input from neighboring neurons while simultaneously exponentially decaying. Once the sum of this input exceeds the threshold level, Neuron j spikes.

where v_i represents the voltage of neuron i and τ_v represents the decay factor of e for each single neuron (in units of ms). We place a negative sign to show proper exponential decay. We see this differential equation is what we want because it decays by a factor of $\frac{1}{\tau_\text{v}}$, however it does not take into account the instantaneous voltage neuron i receives from it's neighbors.

In our model all of the neurons are connected by a **weight**, w_{ij}, which indicates the strength of the signal between neuron j and neuron i (Figure 1.1.2). When neuron j spikes, its neighboring neurons receive a small amount of voltage. We can write the instantaneous change in voltage for neuron i as:

$$\Delta v_i = w_{ij} \quad \text{when } j \text{ spikes}$$

Therefore, this is what we want because every time j spikes, its neighboring neurons receive a small amount of voltage.

1.2 Current-Based Synaptic Transmission

As we discussed, action potential is a necessary process transferring information between neurons, and we wouldn't exist without it! Voltage-gated sodium and potassium channels control the timing and release of sodium and potassium ions. This allows for a gradual increase and subsequent decrease of voltage, where graphs resemble continuous parabolic curves with an identifiable peak voltage (i.e., the threshold). In addition, neurons and synapses resemble the structure of circuits and they depend on **current** (amps), conductance and capacitance. Though in our model, we are just going to focus on current.

If current didn't exist in a biological neural network, this network would be hyperactive; some might call this epileptic. Incorporating current improves our model, and delays the speed in which the input travels from neuron j, which just spiked, to j's neighboring neurons. Similarly, a strictly voltage-based model would not realistic at all because the voltage would travel instantaneously, causing all of neuron j's neighbors to spike immediately after j spiked. Therefore, presenting voltage and current together, makes our model more realistic to a biological human neural network.

In our model, voltage also exponentially decays, though it increases with the addition of current from neighboring neurons instead of just purely synaptic input from spiking neurons, as we saw in our leaky integrate and fire model. Current, scaled by a constant, $v_r \approx 271.8282$, is being added to voltage. We will explain the meaning behind this value in the next subsection. This exponentially decaying variable, current, decreases the power of the voltage and guarantees that our network will not explode, but will instead converge to fixed values over time. We can write a new differential equation combining current and voltage:

$$\frac{dv_i}{dt} = \frac{-v_i}{\tau_v} + v_r c_i$$

in which v_i represents neuron i's voltage, $v_r \approx 271.8282$ controls how much current is released, and $\tau_{\text{v}} = .01\,\text{sec}$ implies that it takes neuron i 10 ms or .01 sec for its voltage to decay by a factor of e. Also, c_i indicates the current for neuron i. Therefore, when a neuron spikes current makes the voltage increase at a rate of v_r.

Yet how does the current alone work, and when does it increase? Let's examine our current differential equation in the absence of spiking:

$$\frac{dc_i}{dt} = -\frac{c_i}{\tau_{\text{c}}}$$

where current decays by $\frac{1}{\tau_{\text{c}}}$. τ_{c} implies that it takes neuron i $\tau_{\text{c}} = 10\,\text{ms}$ or $\tau_{\text{c}} = .01\,\text{sec}$ for its current to decay by a factor of e. Solving for c_i, we have that $c_i(t) = e^{-t/\tau_{\text{c}}}$, using separation of variables.

We see that the differential equation above accurately models the decaying current which we wanted to decrease the speed that the voltage spreads, yet how do we explain for what happens when instantaneous spiking occurs?

As discussed in last chapter, we know already that every pair of neurons, i, j, is connected. Therefore, we define the weight, w_{ij}, as something which measures the strength of the connection from j to i. We can write this instantaneous change in current for neuron i as:

$$\Delta c_i = w_{ij} \quad \text{when } j \text{ spikes}$$

Therefore in this case, when j spikes it releases synaptic input to all of its neighbors i in the form of current, and this current gets added to neuron i's voltage (also called, membrane potential).

1.2.1 Parameter Values

The parameter, v_r appears quite often in our current model and is crucial to our model, as it controls how much current is released. Yet, how do we solve for its precise value?

More specifically, we want to solve for v_r in the instance where voltage starts at zero, and current starts at 1, $c(0) = 1$. In addition, we would like to normalize this parameter so in circumstances when the voltage goes up to one, the current goes back down to zero. This makes voltage and current inversely related, which is the entire purpose of our model.

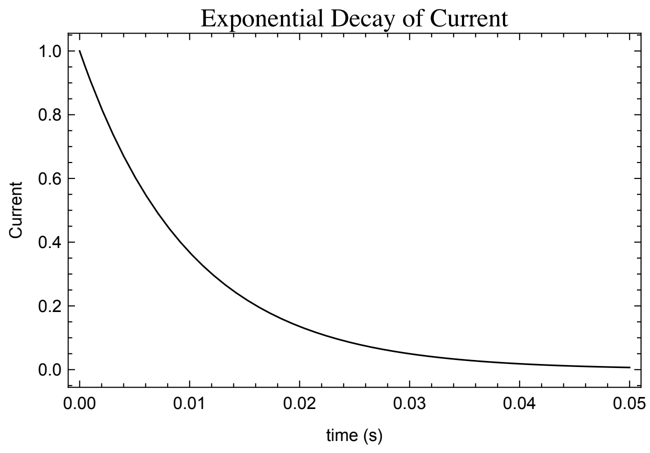

Figure 1.2.1: Graph showing Current Exponential Decay

We found the solution to $\frac{dc}{dt}$ in the previous section:

$$c(t) = e^{-t/\tau_c}$$

Let's take the integral of $c(t)$ from 0 to ∞, to find its average value on this interval. We have:

$$\int_0^\infty e^{-t/\tau_c} dt$$

We have:

$$= \left[-\tau_c e^{-t/\tau_c}\right]_0^\infty$$

$$= \tau_c$$

where as we saw previously, $\frac{1}{tau_c}$ is the rate in which the current is decaying. We can use substitution in our differential equation for $\frac{dv}{dt}$, replacing c with our solution to $\frac{dc}{dt}$. We have:

$$\frac{dv}{dt} = \frac{-1}{\tau_v} v + v_r c$$

$$\frac{dv}{dt} = \frac{-1}{\tau_v} v + v_r e^{-t/\tau_c}$$

Let's assume that v_r is the maximum point on the voltage curve (and is the solution to the $\frac{dv}{dt}$ differential equation). We realize that if a neuron spikes and current gets an input of 1, our graph looks something like this:

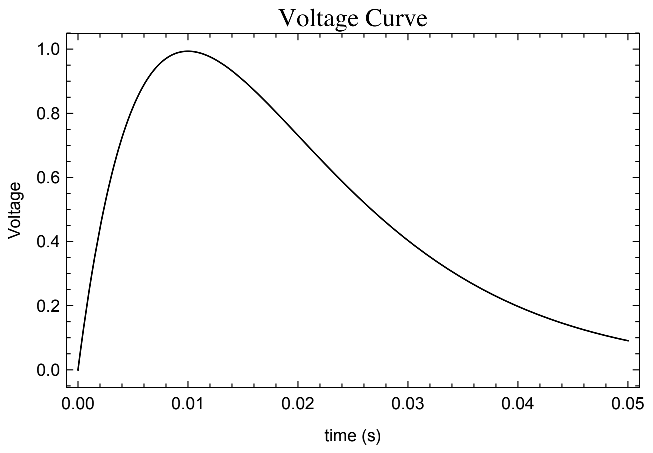

Figure 1.2.2: Graph showing voltage curve, where the maximum point on the curve is t_m

Using the `Dsolve` command in Mathematica, let's solve for the solution to our differential equation for voltage to best determine v_r. This will give us the maximum point of the voltage curve, which we can use to determine how much current is released. We start with the initial condition that $v(0) = 1$. We have the following graph:

Let t_m be the maximum point on our voltage curve. Therefore, at this maximum point:

$$v'(t_m) = 0$$

and

$$v(t_m) = v_r$$

Using Mathematica and these initial conditions (above), we find that the solution to the differential equation is:

$$v(t) = v_r t e^{-t/\tau_v}$$

Plugging in t_m in Mathematica, we see that when $\tau_c = \tau_v$, $t_m = \tau_v$. So we can plug our solution into the differential equation:

$$v(t_m) = v_r t_m e^{-t_m/\tau_v}$$

$$v(t_m) = v_r \tau_v e^{-\tau_v/\tau_v}$$

$$v(t_m) = \frac{v_r \tau_v}{e}$$

Assuming that $v(t_m) = 1$ because this is the maximum of our voltage range, we can use substitution:

$$1 = \frac{v_r \tau_v}{e}$$

$$\frac{1}{v_r} = \frac{v_r \tau_v}{v_r e}$$

$$\frac{1}{v_r} = \frac{\tau_v}{e}$$

$$v_r = \frac{e}{\tau_v}$$

$$v_r = \frac{e}{.01}$$

$$v_r \approx 271.8282$$

Therefore, we can also conclude that $\frac{-1}{\tau_v} v_r$ is the initial slope of the Voltage curve because

$$v(t) = v_r t e^{-t} / \tau_v$$

$$v'(t) = \frac{-1}{\tau_v} v_r e^{-t} / \tau_v$$

$$v'(0) = \frac{-1}{\tau_v} v_r$$

1.3 Spike Time Dependent Plasticity

When the brain matures from a disorganized spiking network into something more organized, synapses are changing, more specifically, the types of synapses, the number of connections as well as the strengths of the connections (i.e. the **synaptic weights**). As mentioned previously, we define every pair of connected neurons as i, j. Therefore, we define the weight, w_{ij}, as something which measures the strength of the connection from j to i, and w_{ji}, as a value which measures the strength of the connection from i to j. Summing the inputs to neuron i represents the aggregate synaptic drive coming to this neuron from neighboring neurons.

We define Hebbian plasticity as “changes in the connection strength between two neurons as a result of correlated firing” [4]. Spike time dependent plasticity (STDP) is a generalization of Hebbian plasticity. We can think of this as a type of reward system for spiking neurons; it

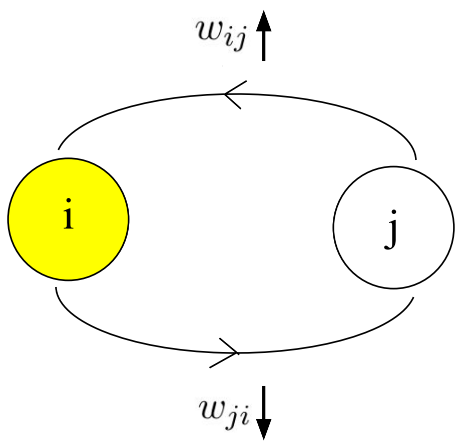

Figure 1.3.1: Neuron i (yellow) spikes, and this causes w_{ij}, the connection from neuron j to neuron i to increase. This causes w_{ji}, the connection from neuron i to neuron j to decrease.

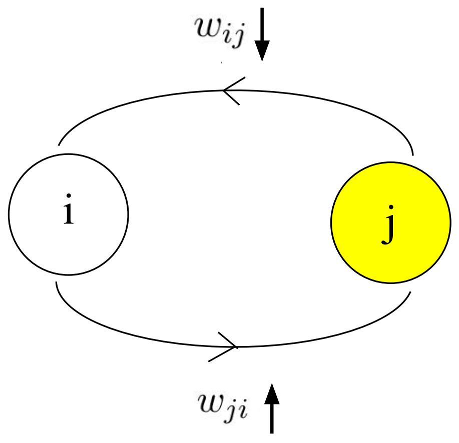

Figure 1.3.2: Neuron j (yellow) spikes, and this causes w_{ij}, the connection from neuron j to neuron i to decrease. This causes w_{ji}, the connection from neuron i to neuron j to increase.

adjusts the strengths of connections and increases those connected to neurons who are spiking often.

On a neurological level, we can break STDP down into two categories: Long Term Potentiation (LTP) and Long Term Depression (LTD). LTP refers to the rule that when a neuron spikes, the already-existing strong inputs to that neuron become stronger. This is a strong demonstration of Hebbian plasticity. LTD refers to the rule that if a neuron did not spike, then the already-existing weak inputs from that neuron get weaker.

Therefore, looking at our neurons i and j, we can conclude the following. First scenario: If i spikes, w_{ji} (the connection from neuron i to j) increases in strength. Assuming j did not spike recently, then w_{ij} (the connection from j to i) decreases in strength. Second scenario: If j spikes, w_{ij} (the connection from j to i) increases in strength. Assuming i did not spike recently, then w_{ji} (the connection from i to j) decreases in strength.

Similarly to voltage and current, we can describe our STDP model with a differential equation. Let p be an exponentially decaying trace for spiking, which controls how much synaptic input from neighboring neurons is released, similarly to current, which controls how much voltage is released. Let's look at the differential equation for p_i, corresponding to the decaying trace for neuron i:

$$\frac{dp_i}{dt} = \frac{-p_i}{\tau_p}$$

We can solve for our solution:

$$p_i(t) = p_i(0)e^{-t/\tau_p}$$

where $p_i(0)$ is our initial value and t is time. Let's solve for τ_p, also know as our e-folding time. We would like τ_p to make p decay about 10 times in about 50 ms. Therefore, we can write this as:

$$p_i(.05) = \frac{1}{10}p_i(0)$$

This means that:

$$p_i(0)e^{-.05/\tau_p} = \frac{1}{10}p_i(0)$$

Therefore, we can rewrite this as:

$$e^{-.05/\tau_p} = \frac{1}{10}$$

Taking the natural log of both sides we have:

$$\log e^{\frac{-.05}{\tau_p}} = \log\frac{1}{10}$$

$$\frac{-.05}{\tau_p} = \log(\frac{1}{10})$$

We see that τ_p is roughly equal to .0217146, our e-folding time.

However, we are just considering our model in the absence of spiking. What happens when we do have spiking? When neuron j spikes, something instantaneous happens:

- w_{ji} immediately increases by $w_{\text{spike}}p_i$
- w_{ij} immediately decreases by $w_{\text{spike}}p_j$

each describing LTP and LTD, respectively. w_{spike} is a constant roughly equal to .00046052. We will explain how this is calculated when we discuss changing thresholds. w_{spike} controls how much the weights change.

Next, we see that synaptic weight (w_{ij}) has an effect on how much current is released. Therefore, when neuron j spikes, c_i increases by:

$$\begin{cases} 0 & \text{if } w_{ij} \leq 0 \\ w_{ij} & \text{if } 0 < w_{ij} < 1 \\ 1 & \text{if } w_{ij} \geq 1 \end{cases}$$

We see that this model works exactly as we like. When w_{ij} or w_{ji} changes, this has an effect on the current and thus the voltage in the entire system (as we can infer from our Current-Based Synaptic Transmission model)

1.4 Synaptic Scaling

In biology, there exist various negative feedback mechanisms that add stability to the system. **Homeostatic Synaptic Plasticity** (i.e. **synaptic scaling**) is considered a facet of the main negative feedback mechanism in cells, **Homeostatic Plasticity**. This is a normalization procedure in biologically which in our model, should make all of our randomized connections converge faster.

We know already w_{ij} means that every pair of neurons i, j, is connected. Summing the inputs to neuron i represents the aggregate synaptic drive coming to this neuron from neighboring neurons.

To control the speed in which the synaptic drive increases and the neuron spikes [11], let's normalize all of the inputs to neuron i, so that they don't exceed a particular value, which we set as being equal to 1. All the inputs to i will be scaled by $\frac{1}{\text{sum of inputs to } i}$. In the case that there exist zero inputs to i, we set the sum of all inputs to be equal to one, guaranteeing that eventually the sum of all inputs to i does not change as a result of synaptic scaling.

More motivations behind synaptic scaling include the fact that: if these inputs become too large, our network will be physically unrealistic. Similarly, to current, we see that if synaptic scaling didn't exist, biologically, a neural network would be hyperactive; or epileptic. Incorporating synaptic scaling improves our model and delays the speed in which the input travels from the neuron j that just spiked, to other neighboring neurons.

In addition as we saw with STDP, when weights change, w_{ij}, for instance, will become stronger, and by default w_{ji} will become weaker. However, if we did not have synaptic scaling, then we would always end up with w_{ij} as being really large and w_{ji} as being really small. Thus, synaptic scaling makes our synaptic connections more realistic.

For a particular pair of neurons, neuron i and j, we can define the following to be the sum of all inputs to neuron i.

$$m = \sum_{j=1}^{100} w_{ij}$$

where m = the sum of inputs to neuron i. We assume that $m_i \neq 0$, because in that case we would be dividing by zero. In this case that $m_i = 0$, we change m_i to equal 1. Finally we define our new synaptic weight as:

$$w_{ij}{}^{\text{new}} = \frac{w_{ij}}{m_i}$$

where w_{ij} = the new strength of the connection from j and i. We see this technique is exactly what we want because it normalizes the connections between neurons by making sure the inputs to i don't exceed 1, and it makes our model more biologically realistic.

1.5 Input

You see an object in the distance. It's getting bigger as its gradually approaching your face. The speed in which the object is approaching you seems to be getting faster and faster. It's a ball and it's being thrown at you, at full force. How do you react?

Neurologically speaking, this object is a stimulus. The human brain is constantly changing during development as a result of many different types of stimuli such as sights, sounds and tastes. In this project we will primarily focus on visual stimuli, also called input.

Human vision starts with incoming light traveling through all of the layers in the retina: the Ganglion Cell Layer, the Inner Plexiform Layer, the Inner Nuclear layer, the Outer Nuclear Layer, the Layer of Photoreceptor Outer segments and finally, the Pigmented epithelium [6]. We are mostly focused on the layer of photoreceptors for this project. Rods and cones are

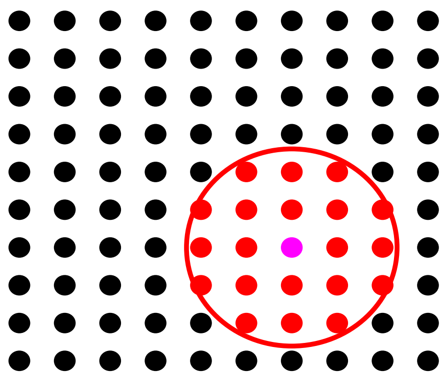

Figure 1.5.1: Input Disc Expansion, the magenta point indicates (x_c, y_c) our center of the expanding disc

photoreceptors which are necessary for vision, though these cells are also neurons. Our project is mainly (though not specifically) focused on modeling an 100-neuron network of photoreceptors.

Lets start by constructing a model of the colliding object on a 10×10 grid. This "object" could represent a ball getting thrown at an infant or possibly a predator approaching the infant in the wild. In our network, the object, also called a patterned visual stimulus structure on a grid, should be enough information for the neurons to, in a sense, figure out that they are organized in a grid. Eventually, the goal is for the neurons to use this point of reference to organize themselves back into a grid.

Our expanding disc will be overlaid on top of a 10×10 grid of all the neurons in our network, where each vertex represents a neuron. We see that the point $(10, 1)$ on the grid corresponds to neuron 1 in our network, $(9, 1)$ for neuron 2, $(8, 3)$ for neuron 3, etc. For the next ten neurons, we have that $(10, 2)$ corresponds to neuron 11, $(10, 3)$ for neuron 12, $(11, 3)$ for neuron 22, etc.

The disc will be constructed as a matrix, which we will discuss in the next chapter. The model works in such a way so that the disc starts from a random center point, such that $1 < (x_c, y_c) < 10$, where $x_c, y_c \in \mathbb{R}$. As the disc expands at a rate of $\tau_{\text{m}} = 0.001$ (taking 20 seconds of simulation time to expand, equal to only 5 seconds of real time) over the grid, it delivers input to a neuron when the rim of the disc hits the vertex corresponding to that neuron. Thus the neurons positioned furthest from the center, if they get any input, will get it much later than those located closest to the center of the disc. We set our τ_{m} to this value, as it creates the most biologically realistic model of an object approaching.

Therefore we see our equation for the radius of the disc expanding is:

$$r(t) = \tau_{\mathrm{m}}(t - t_0)$$

where τ_{m} = the rate in which the circle is expanding and in which the center of the disc is randomly selected. Once the disc expands beyond the dimensions of the grid, it starts over at a different random point on the grid, not necessarily an integer value, and begins to expand and deliver input to the neurons.

1.6 Changing the Thresholds

As described previously, every neuron has a resting potential which is roughly equal to -70 mV. When it receives enough input, it depolarizes and eventually reaches the target threshold which is often -30 mV in a cell (or $= 1$ in our model). Once the voltage reaches this value, it will reset back to the resting potential (zero in our model).

Biologically, there exist slight fluctuations in threshold from neuron to neuron, though it's still unclear whether this is mere noise or a more complicated mechanism, one that depends on how frequently the neuron is firing and specific membrane potentials [12]. This process is called Homeostatic Intrinsic Plasticity. We see in some studies, neuronal threshold values decrease after high-frequency synaptic stimulation (i.e. current) is introduced in rat hippocampus pyramidal neurons [13].

Threshold changing makes our model more accurate to a real brain, and regulates neuronal activity, causing neurons that are too excited (high spiking rate) to spike less, and neurons that are not spiking enough to spike more.

In our current-based model, the rate in which neurons spike changes from that of a purely voltage-based model. While current adjusts the rate in which voltage approaches the threshold, this model alone doesn't adjust the constant-valued threshold.

Why do we want to incorporate a changing threshold? We see that when thresholds are too high for a particular neuron, the network is dead, and cell input isn't even sufficient to result in an eventual spike. In the rare case that one neuron spikes, this neuron doesn't give enough voltage to neighbors to make them spike. Contrarily, when the thresholds are too low, we see that the network becomes epileptic (hyperactive); when one neuron spikes, it gives input to all of its neighbors and since the neighbors' threshold values are much lower, they spike instantaneously.

The learning rule we're implementing is similar to synaptic scaling, in that it adjusts neuron's average spiking so that every neuron spikes roughly 10 times a second, further normalizing the network. While, biologically speaking this target spiking value ($S_{\mathrm{t}} = 10$ spikes per second) is relatively high, we set it to this value to make our network converge faster. Furthermore, if a neuron is spiking too much (as we see looking at its average spiking history), its threshold increases, making frequent spiking less frequent. And if a neuron isn't spiking enough relative to the target spiking rate, its spiking threshold will be lowered, making it easier for the neuron to spike.

Each neuron has a spike history, which measures how many times recently the neuron has spiked. In addition, there exists a target value for how often we choose a neuron to spike. When a neuron's recent history shows more spiking than the set target value, we want the threshold to go up. When a neuron's recent history shows less spiking than the set target value, we want the threshold to go down.

Here's how the spike history, $S_{\mathrm{av}i}$ for neuron i, works mathematically. $S_{\mathrm{av}i}$ decays exponentially in the absence of spiking, and we see that the units for $S_{\mathrm{av}i}$ are: number of spikes per second. In particular $S_{\mathrm{av}i}$ obeys the following differential equation:

$$\frac{dS_{\mathrm{av}i}}{dt} = -\frac{S_{\mathrm{av}i}}{\tau_{\mathrm{Sav}}}$$

However, we see that the model also depends on the instantaneous change in $S_{\mathrm{av}i}$ for neuron i when there is spiking:

$$\Delta S_{\mathrm{av}i} = \frac{1}{\tau_{\mathrm{Sav}}} \quad \text{when } i \text{ spikes}$$

Therefore $S_{\text{av}i}$ will increase by $\frac{1}{\tau_{\text{Sav}}}$ when i spikes and when i does not spike $S_{\text{av}i}$ will continue exponentially decaying. τ_{Sav} is a parameter that describes the amount of time (sec) it takes for a neuron to realize it is not spiking. So we see that the units of $\frac{1}{\tau_{\text{Sav}}}$ are $\frac{1}{\text{sec}}$.

In addition to the average spiking vector, there exists a target value indicating how often we choose for a neuron to spike. Let's call this $S_{\text{t}} = 10$ spikes/sec. Therefore, we choose that every neuron spikes roughly 10 times a second, which seems very high though reasonable for our model because it will result in faster convergence. From this, the thresholds should go up or down depending on the difference between the target spiking, S_{t} and the average spiking history, S_{av}. Let $v_{\text{max}} =$ be i's threshold, indicating i's maximum voltage (mV). We begin with the following equation:

$$\frac{dv_{\text{max}}}{dt} = \frac{S_{\text{av}i} - S_{\text{t}}}{\tau_{\text{th}}}$$

Note that this differential equation includes the difference $(S_{\text{av}} - S_{\text{t}})$. We see this is exactly what we want: If $S_{\text{av}} > S_{\text{t}}$, then v_{max} increases, and neuron i spikes less. If $S_{\text{av}} < S_{\text{t}}$, then v_{max} decreases, and neuron i spikes more. To clarify, we have the following cases:

- Case 1: $S_{\text{av}} > S_{\text{t}} \Rightarrow$ Threshold increases and as a result, neurons have less likelihood to spike.
- Case 2: $S_{\text{av}} < S_{\text{t}} \Rightarrow$ Threshold decreases and as a result, neurons have more likelihood to spike.
- Case 3: $S_{\text{av}} = S_{\text{t}} \Rightarrow$ Threshold remains the same and activity stays constant.

How responsive is i's threshold to recent spiking? Let τ_{th} be a parameter that is equal to the threshold's response time, where if $\tau_{\text{th}} = 100$, in a completely dead network, i's threshold will go to zero in about 100 seconds. However, this value is slow and might take the network an unnecessary amount of time to converge. The difficulty in decreasing this time of responsiveness is that if it is too fast (i.e. τ_{th} is too low), then the network will quickly start producing seizures. Thus, this parameter needs to change over time. Let τ_{th} start at 1 sec and converge to $\tau_{\text{th}} =$

100 sec. We see our differential equation for τ_{th} can be shown as:

$$\frac{d\tau_{\text{th}}}{dt} = \frac{100 - \tau_{\text{th}}}{\tau_{\text{relax}}}$$

This does what we want it to do because we see that when $\tau_{\text{th}} = 100$, τ_{th} will stop changing, and if $\tau_{\text{th}} > 100$, τ_{th} will continue to approach 100. In addition, since our parameter $\tau_{\text{relax}} = 10,000$, τ_{th} will continue to increase until it asymptotically approaches 100. We see that the solution to the differential equation is:

$$\tau_{\text{th}}(t) = 100 - Ce^{\frac{-t}{\tau_{\text{relax}}}}$$

where C is a constant of integration. Ultimately, in our model we want connections between neurons to change over time, not the individual spiking rates of neurons, because in biology, every excitatory neuron spikes at a similar rate. We want to see initial chaos in the network and then convergence. The learning rule we've described is necessary in achieving that.

2

Converting Differential Equations to Discrete Time

2.1 Our Approach

In this section, we will convert our continuous model into a discrete-time simulation in MATLAB. Since MATLAB does not do symbolic computation, using difference equations (also called recurrence relations) is most efficient [14]. In addition, differential equations are not preferred in implementing this model because in actuality, they are **average time differential equations**, which are continuous until biologically-speaking, an instantaneous event occurs. For instance when a neuron spikes, we use a discrete rule for that instant.

One major difference between the simulation and the biological model is in the fact that in the model, it is not possible for two neurons to spike at the same time because the spiking differs by a fraction of a second, but in the simulation this is possible. Ultimately our differential equations model only really governs behavior between spikes, therefore it is necessary to implement our discrete model for the time when a neurons spikes.

In order to convert our continuous differential equations into difference equations, we use a technique called **Euler's Method** (to be discussed later). Another difference between these two approaches is that our discrete model uses vectors and matrices to keep track of values such as weights, voltage, spiking history, and threshold values of every neuron in the network, as we are modeling a network consisting of 100 neurons. Finally, additional parameters will be introduced using those from the previous chapter. However these will be taking our Δt step size into account. We must remember that ultimately, these discrete time equations are really just an aspect of our simulation of the biological model itself, which in actuality, is purely a differential equations model.

2.2 Weight Matrix, Spike Vector and Voltage Vector

We start by introducing our method for organizing the synaptic weights between neurons. Let's define W as our 100×100 weight matrix of synaptic connections. Entry w_{ij} of W implies that neuron j is connected to neuron i. Almost each synaptic weight is a randomly generated real number between 0 and 1, implying that all neurons in the network are connected and for all 100 neurons, we have $10,000$ connections. Another way of stating this is that every neuron represents a **node** of a graph and the synaptic connection between neurons represents **edges** of a directed graph.

W has a diagonal line of all zeros, implying that no neuron is connected to itself. The line of zeros (where $i = j$) indicates that there exist no neuron that is connected to itself, as we see below:

$$W = \begin{bmatrix} 0 & w_{1,2} & w_{1,3} & \cdots & w_{1,100} \\ w_{2,1} & 0 & w_{2,3} & \cdots & w_{2,100} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ w_{100,1} & w_{100,2} & w_{100,3} & \cdots & 0 \end{bmatrix}$$

We see that while $w_{1,2}$ and $w_{2,1}$, for example, represent the same connection between two neurons, $w_{1,2} \neq w_{2,1}$. Let's define: $\mathbf{s}$ as our 100×1 spike vector, which consists of only zeros and ones. At a particular time step value k, if neuron i spikes, then $\mathbf{s}_i$ becomes a 1 at the $k+1$ time step. If not, $\mathbf{s}_i$ stays as 0. To simplify this, we see that:

$$s_i^{k+1} = \begin{cases} 1 & \mathbf{v}_i^{k} > \mathbf{v}_{\text{max}} \\ 0 & \mathbf{v}_i^{k} < \mathbf{v}_{\text{max}} \end{cases}$$

where $\mathbf{v}$ is our 100×1 voltage vector, which consists of the voltage values (mV) for every neuron in the network. $\mathbf{v}_i$ represents neuron i's voltage. As we saw in our modeling chapter, when the voltage in a particular neuron, say $\mathbf{v}_i^{k}$, exceeds the $\mathbf{v}_{\text{max}}$ threshold value, $\mathbf{v}_i^{k+1}$ is reset to 0.

Let's multiply W by $\mathbf{s}$, our spike vector:

$$\mathbf{s}^{k+1} = W\mathbf{s}^k$$

This quantity s^{k+1} gives us the spike record for each neuron in the $k+1$ time step, because of synaptic scaling which we will speak about later. We see this in matrix form:

$$s^{k+1} = W \cdot s^k = \begin{bmatrix} 0 & w_{12} & w_{13} & \dots & w_{1j} \\ w_{21} & 0 & w_{23} & \dots & w_{2j} \\ w_{31} & w_{32} & 0 & \dots & w_{3j} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{i1} & w_{i2} & w_{i3} & \dots & 0 \end{bmatrix} \cdot \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_i \end{bmatrix}$$

were $i = j = 100$. Adding the spike record at step $k+1$ to the vector of voltages at k gives us the voltage vector for the $k+1$ time step. We prove this more extensively in the next chapter, converting our differential equations into difference equations using Euler's method.

$$v^{k+1} = r_{\text{v}} v^k + W \cdot s^k$$

where $W \cdot s^k$ indicates the synaptic input the spiking neuron releases to neighboring neurons.

We see that this difference equation does what we want it to do because in the absence of spiking, the voltage decreases exponential, though as soon as a neuron spikes and an entry in the $\mathbf{s}$ vector $= 1$, the voltage for that particular neuron increases.

2.2.1 Implementation in Code

- `tmax`: runs program for a specified number of seconds of simulation.
- `delta_t`: Indicates the step size.
- `maxstep=tmax/delta_t`: Number of iterations program runs for.
- `size`: The size of the network is 100 neurons.
- `W`: Our weight matrix. Weights along the diagonal equal zero, while the rest of the entries initially are a random rational real number in between zero and one.
- `Vreset`: Resets voltage back to 0 after neuron has spiked.

- `Wscale`: Scales the weight matrix: $W_{\text{scale}} = \frac{2}{size-1} = 0.0202$
- `Vscale`: Indicates the maximum initial voltage for each neuron, and works as a coefficient for our initial voltage vector. In our program, $V_{\text{scale}} = 0$.
- `V`: Indicates voltage. This is specifically, a 100×1 vector, which contains the initial voltages of the neurons, $\mathbf{v}$ and then gets updated as the program runs.
- `S`: The initial 100×1 spike vector, which is essentially input in this model. As explained previously, $\mathbf{s}$ consists of zeros and ones. This indicates the spiking history where a zero indicates there was no spike and a one indicates there was a spike. $\mathbf{s}$ initially contains all zeros.

2.3 Leaky Integrate and Fire

As stated earlier, using Euler's method, we see that the discrete time model is really just an aspect of the simulation of the model itself. Our mathematical model mostly comprises of just differential equations. We start with our differential equation from Chapter 1.

$$\frac{dv_i}{dt} = -\frac{v_i}{\tau_v}$$

where v_i represents the voltage of neuron i and τ_{v} implies that neuron i takes 10 ms to decay by a factor of e. In our model, $\tau_{\text{v}} = .01$. This equation, above is very simplified because it purely describes this decay in the absence of spiking or between spikes, hence the fact that it is a **average time differential equation**. It does not take into account the instantaneous input neuron i receives from it's neighbors.

We can rewrite the above differential equation in vector form, where now instead, we are looking at the voltages of all 100 neurons in our network. We have:

$$\frac{d\mathbf{v}}{dt} = -\frac{\mathbf{v}}{\tau_{\text{v}}}$$

Now, let's convert this into a difference equation using Euler's method. We can write our change in voltage ($d\mathbf{v}$) as the difference between two subsequent time steps determined by the step size

Δt. We write this as: $(\mathbf{v}^{k+1} - \mathbf{v}^k)$. In addition we can write our dt as a change in time: simply, Δt. Therefore, we have:

$$\frac{\mathbf{v}^{k+1} - \mathbf{v}^k}{\Delta t} = -\frac{\mathbf{v}^k}{\tau_{\text{v}}}$$

We can simplify this to:

$$\mathbf{v}^{k+1} = (1 - \frac{\Delta t}{\tau_{\text{v}}})\mathbf{v}^k$$

Let $r_{\text{v}} = 1 - \frac{\Delta t}{\tau_{\text{v}}}$. Therefore, we have:

$$\mathbf{v}^{k+1} = r_{\text{v}}\mathbf{v}^k$$

We see that r_{v} is just an aspect of our simulation because it depends on Δt, whereas τ_{v} represents the fact that it takes neuron i 10 ms for its voltage to decay by a factor of e. Therefore, we set $\tau_v = 0.01$ sec and $\Delta t = .001$:

$$r_{\text{v}} = 1 - \frac{.001}{.01} = 0.9$$

This serves as the perfect decay constant for $\mathbf{v}^k$.

Now, what does our discrete model look like where neurons are spiking? Again, let's look at the rules for our spike vector, where i indicates any given neuron:

$$s_i = \begin{cases} 1 & \text{if neuron i just spiked} \\ 0 & \text{otherwise} \end{cases}$$

We can now implement, $W \cdot s^k$, the sum of synaptic input released by spiking neurons to neighboring neurons, into our difference equation. To simplify this, let's look at it in component form, where $\mathbf{v}_i{}^k$ = neuron i's voltage at the k^{th} time step, and $w_{ij}\mathbf{s}_j$ indicates the sum of the synaptic input from neuron j to neuron i in the k^{th} time step.

$$v_i{}^{k+1} = r_{\text{v}}v_i{}^k + \sum_{j=1}^{100} w_{ij}s_j{}^k$$

Rewriting our component equation into vector form, we have:

$$\mathbf{v}^{k+1} = r_v\mathbf{v}^k + W\mathbf{s}^k$$

We see that, like our differential equation, this difference equation does exactly what we wanted it to do because every time step $\mathbf{v}$ decays by a rate of r_{v}. This means that if neuron j

spikes, then $vecs_j$ will be equal to 1 and w_{ij} will be added to the voltage vector of every neuron in the network (i.e. every neuron j is connected to). In the case that j doesn't spike, the entries in voltage vector will decrease because they haven't gotten any input.

2.3.1 Implementation in Code

Many of these variables follow from the last section.

Old Variables (from last section):

- `tmax`: runs program for a specified number of seconds of simulation.
- `delta_t`: Indicates the step size.
- `maxstep=tmax/delta_t`: Number of iterations program runs for.
- `size`: The size of the network is 100 neurons.
- `W`: Our weight matrix. Weights along the diagonal equal zero, while the rest of the entries initially are a random rational real number in between zero and one.
- `Vreset`: Resets voltage back to 0 after neuron has spiked.
- `Wscale`: Scales the weight matrix: $W_{\text{scale}} = \frac{2}{size-1} = 0.0202$
- `Vscale`: Indicates the maximum initial voltage for each neuron, and works as a coefficient for our initial voltage vector. In our program, $V_{\text{scale}} = 0$.
- `V`: Indicates voltage. This is specifically, a 100×1 vector, which contains the initial voltages of the neurons, $\mathbf{v}$ and then gets updated as the program runs.
- `S`: The initial 100×1 spike vector, which is essentially input in this model. As explained previously, $\mathbf{s}$ consists of zeros and ones. This indicates the spiking history where a zero indicates there was no spike and a one indicates there was a spike. $\mathbf{s}$ initially contains all zeros.

New variables:

- τ_{v}: Time it takes neuron i for its voltage to decay by a factor of e. We set $\tau_v = 0.01$ sec

- r_V: the decay factor of voltage for each neuron at each time step. $r_{\text{v}} = 1 - \frac{\Delta t}{\tau_{\text{v}}} \Rightarrow r_{\text{v}} = 1 - \frac{.001}{.01} = 0.9$

- V=r_V*V+W*S: The recurrence relation in our for-loop. WS creates a 100×1 vector, which we add accordingly to our voltage ($\mathbf{v}$) vector.

2.4 Current-Based Synaptic Transmission

Using Euler's method, we can convert the following differential equation describing current and voltage into another recurrence relation. From Chapter 1, we have:

$$\frac{dv_i}{dt} = \frac{-v_i}{\tau_v} + v_r c_i$$

in which v_{res}, which stands for voltage response, a parameter which controls how much current is released, and τ_v is the time constant for the decay. We rewrite our differential equation as:

$$\frac{{v_i}^{k+1} - {v_i}^k}{\Delta t} = \frac{-{v_i}^k}{\tau_{\text{v}}} + v_r {c_i}^k$$

$${v_i}^{k+1} - {v_i}^k = -{v_i}^k \cdot \frac{\Delta t}{\tau_{\text{v}}} + (v_r \cdot \Delta t) c_i$$

$${v_i}^{k+1} = {v_i}^k (1 - \frac{\Delta t}{\tau_{\text{v}}}) + (v_r \Delta t) c_i$$

Let $r_{\text{v}} = 1 - \frac{\Delta t}{\tau_{\text{v}}}$ and $v_{\text{res}} = v_r \Delta t$. We can write our final difference equation as:

$${v_i}^{k+1} = r_{\text{v}} {v_i}^k + v_{\text{res}} {c_i}^k$$

We can write this in vector form as:

$$\mathbf{v}^{k+1} = r_{\text{v}} \mathbf{v}^k + v_{\text{res}} \mathbf{c}^k$$

We can also use Euler's method to convert our current differential equations into recurrence relations, however this gets slightly more complex. We have the current differential equation:

$$\frac{dc_i}{dt} = \frac{-c_i}{\tau_{\text{c}}}$$

where τ_c is the time constant for the current decay, and the current decays by $\frac{1}{\tau_c}$.

However, this is considered an average time differential equation because the model also depends on the instantaneous change in current for neuron i, which we explained previously was:

$$\Delta c_i = w_{ij} \quad \text{when } j \text{ spikes}$$

where τ_c is the time constant for the decay. After this instantaneous change, voltage resumes changing continuously.

So how do we incorporate both the differential equation and this condition into our difference equation? Using Euler's method, we can write the current equation describing a particular neuron i as:

$$\begin{aligned} \frac{{c_i}^{k+1} - {c_i}^k}{\Delta t} &= \frac{-{c_i}^k}{\tau_c} \\ {c_i}^{k+1} - {c_i}^k &= -(\frac{\Delta t}{\tau_c}){c_i}^k \\ {c_i}^{k+1} &= {c_i}^k - (\frac{\Delta t}{\tau_c}){c_i}^k \\ {c_i}^{k+1} &= (1 - \frac{\Delta t}{\tau_c}){c_i}^k \end{aligned}$$

Let's add the instantaneous change of current to this difference equation. This guarantees that when neuron j spikes, each c_i immediately increases by w_{ij}, or the sum of synaptic inputs from neighboring neurons. We have:

$${c_i}^{k+1} = (1 - \frac{\Delta t}{\tau_c}){c_i}^k + \sum_{j=1}^{100} w_{ij} s_j$$

where as previously mentioned, τ_c is the time constant for the decay. Let $1 - \frac{\Delta t}{\tau_c} = r_c$. By substitution, we have:

$${c_i}^{k+1} = r_c {c_i}^k + \sum_{j=1}^{100} w_{ij} s_j$$

We see this works because when neuron j spikes, each c_i immediately increases by w_{ij}. Therefore, we see that r_v is necessary for the releasing of current. When there is no current, we see that when neuron j spikes, each v_i immediately increases by w_{ij}, as mentioned in the previous section. We can rewrite our current recurrence relation in vector form as:

$$\mathbf{c}^{k+1} = r_c \mathbf{c}^k + W\mathbf{s}$$

2.4.1 Implementation in Code

- `r_C = 1-delta_t/T_c`: the decay factor for the current for each neuron at each time step. The decay factor changes depending on our Δt value. $r_c = \frac{1-.001}{.01} = .9$
- `V_r=exp(1)/T_v`: v_r is the max point of the V curve (solution to the $\frac{dv}{dt}$differential equation).
- `Vresponse = V_r*delta_t`: Controls how much current is added to the voltage in a given time step.
- `C`: Our initial vector of currents consisting of all zeros.
- `C=r_C C +(1/size)*W*S`: Our recurrence relation for current. This formula does not include input, as we will see later in this chapter.
- `V=r_V*V+Vresponse*C`: Our recurrence relation including current.

2.5 Spike Time Dependent Plasticity

As mentioned in the first chapter, spike time dependent plasticity (STDP) is a generalization of Hebbian plasticity. We can think of this as a type of reward system for spiking neurons; it adjusts the strengths of connections and increases those connected to neurons who are spiking often.

We start with our differential equation, showing the decaying trace for spiking over time:

$$\frac{dp_i}{dt} = \frac{-p_i}{\tau_\mathrm{p}}$$

We can now rewrite our differential equation as a recurrence relation, where $\mathbf{p}$ is a vector. Every entry in the vector corresponds to a particular neuron, i's decaying trace for spiking:

$$\frac{p_i^{k+1} - p_i^k}{\Delta t} = -\frac{{p_i}^k}{\tau_\mathrm{p}}$$

$$p_i^{k+1} - p_i^k = -\frac{\Delta t}{\tau_\mathrm{p}} p_i^k$$

$$p_i^{k+1} = p_i^k - \frac{\Delta t}{\tau_{\mathrm{p}}} p_i^k$$

$$p_i^{k+1} = (1 - \frac{\Delta t}{\tau_{\mathrm{p}}}) {p_i}^k$$

However, as we say in our model, there is an instantaneous component to our decaying trace. When a neuron spikes, the spiking trace increases by 1, corresponding to the entry in our **s** spike vector. If not, **p** continues to exponentially decrease. We can write the instantaneous change in spiking trace for neuron i as:

$$\Delta p_i = 1 \quad \text{when } j \text{ spikes}$$

Therefore, we need to show this instantaneous change of the spiking trace in our difference equation, so we add an additional term: the corresponding entry of our **s** vector (consisting of only zeros and ones)

$$p_i^{k+1} = (1 - \frac{\Delta t}{\tau_p}) {p_i}^n + s_i$$

We see that this does what we want because, naturally **p** decays on its own, though when i receives input from neighboring neurons, p_i increases by 1.

Let $(1 - \frac{\Delta t}{\tau_p}) = r_p$. By substitution, we have:

$$p_i^{k+1} = r_p {p_i}^k + s_i$$

We can write this in vector form:

$$\mathbf{p}^{k+1} = r_p \mathbf{p}^k + \mathbf{s}^k$$

How big should p be? Let's add S_{t}, our target spiking rate from Chapter 1, to our decaying trace differential equation to best approximate spiking in the network:

$$\frac{d\mathbf{p}}{dt} = \frac{-1}{\tau_p}\mathbf{p} + S_{\mathrm{t}}$$

Therefore, we see that we can find the equilibrium by setting the differential equation equal to zero. We have:

$$\frac{-1}{-\tau_p}\mathbf{p} + S_{\mathrm{t}} = 0$$

$$\mathbf{p} = \tau_p \mathbf{s}_t \approx (21.7)(10) \approx 217$$

so this is the average $\mathbf{p}$ value. Let's make a new variable $W_{\text{change}} = .0001$, where W_{change} represents the amount we expect a weight to change with a spike. Meaning, when there is a spike, the average weight change is .0001. Since the weights start out at .01 (for 100 neurons), this means it takes about 100 bad spikes for one neuron to start ignoring input from its neighbor. Therefore, we define:

$$W_{\text{spike}} = \frac{W_{\text{change}}}{\tau_p \mathbf{s}_t}$$

where τ_p represents the e-folding time in STDP, we have it decaying by a factor of 10 in about 50 milliseconds. As we know the weights are starting at roughly .01 because of synaptic scaling, which we will observe later.

Now, how do we connect these two equations to our matrix of weights, W in a way that properly simulates our model? Remember, we can break STDP down into two categories: Long Term Potentiation (LTP) and Long Term Depression (LTD). LTP refers to the rule that if a neuron spiked in the previous time step, then the already-existing strong inputs to that neuron become stronger. LTD refers to the rule that if a neuron didn't spike in the previous time step, then the already-existing weak inputs from that neuron get weaker.

Therefore, we can deduce that when neuron j spikes, something instantaneous happens:

- w_{ji} immediately increases by $w_{\text{spike}} p_i$
- w_{ij} immediately decreases by $w_{\text{spike}} p_i$

And when neuron i spikes,

- w_{ij} immediately increases by $w_{\text{spike}} p_j$
- w_{ji} immediately decreases by $w_{\text{spike}} p_j$

Examine the expression below:

$$w_{\text{spike}}({s_i}^k {p_j}^k - {s_j}^k {p_i}^k)$$

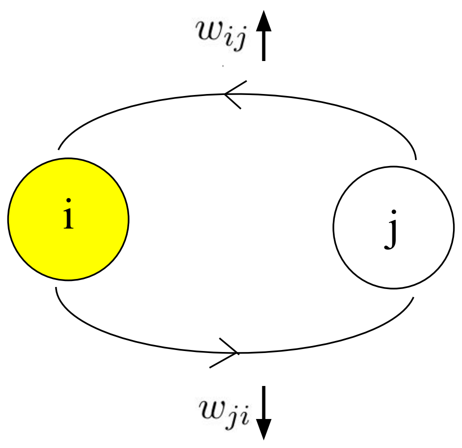

Figure 2.5.1: Neuron i (yellow) spikes, and this causes w_{ij}, the connection from neuron j to neuron i to increase. This causes w_{ji}, the connection from neuron i to neuron j to decrease.

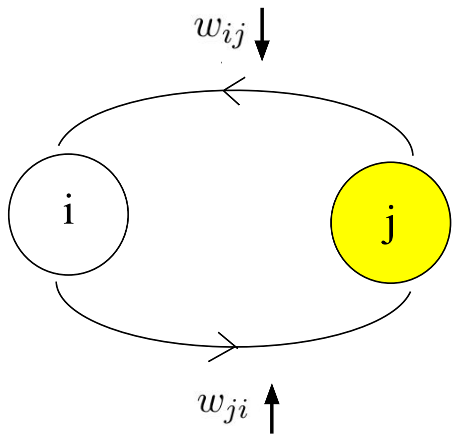

Figure 2.5.2: Neuron j (yellow) spikes, and this causes w_{ij}, the connection from neuron j to neuron i to decrease. This causes w_{ji}, the connection from neuron i to neuron j to increase.

We see that if j spiked, meaning $s_j = 1$ and $s_i = 0$, then the expression above follows as:

$$= w_{\text{spike}}(0 \cdot {p_j}^k - 1 \cdot {p_i}^k)$$

$$= -w_{\text{spike}}{p_i}^k$$

So for a spiking j, we've shown some sort of decrease of $w_{\text{spike}}{p_i}^k$. Consider the case when i spikes, meaning $s_i = 1$ and $s_j = 0$, then the expression follows as:

$$w_{\text{spike}}({s_i}^k{p_j}^k - {s_j}^k{p_i}^k)$$

$$= w_{\text{spike}}(1{p_j}^k - 0 \cdot {p_i}^k)$$

$$= w_{\text{spike}}{p_j}^k$$

So for a spiking i neuron, we've also shown some sort of increase of $w_{\text{spike}}{p_j}^k$. Adding this expression to our weight, w_{ij} will give us a difference equation for all inputs from neuron j to neuron i.

$${w_{ij}}^{k+1} = {w_{ij}}^k + w_{\text{spike}}({s_i}^k{p_j}^k - {s_j}^k{p_i}^k)$$

In our model,

$$w_{\text{spike}} = \frac{w_{\text{change}}}{\tau_{\text{p}} \cdot S_{\text{t}}}$$

$$w_{\text{spike}} = \frac{.0001}{0.0217(10)}$$

$$w_{\text{spike}} = .00046083$$

We see that synaptic weight (w_{ij}) has an effect on how much current is released. When neuron j spikes, for example, c_i increases by:

$$\begin{cases} 0 & \text{if } w_{ij} \leq 0 \\ c_{\text{spike}} w_{ij} & \text{if } 0 < w_{ij} < 1 \\ c_{\text{spike}} & \text{if } w_{ij} \geq 1 \end{cases}$$

We can rewrite our difference equation in vector form as:

$$W^{k+1} = W^k + w_{\text{spike}}(\mathbf{s}^k(\mathbf{p}^k)^\top - \mathbf{p}^k(\mathbf{s}^k)^\top)$$

The transpose allows us to multiply our two existing column vectors so that we can determine whether we add or subtract a value from W^k.

2.5.1 Implementation in Code

- `Wchange=.0001`: Used for computing.
- `Wspike=Wchange/(T_p*St)`: Used for change in synaptic weight if a neuron spikes. $w_{\text{spike}} = .00046083$
- `r_p=1-(delta_t/T_p)` Spike Decaying Trace for STDP. $r_p = 0.9539$.
- `W(W<0)=0`: Guarantees that all weights will be positive.

2.6 Synaptic Scaling

Again, to control the speed in which the synaptic drive increases and the neuron spikes [11], let's normalize all of the inputs to neuron i, so that they don't exceed a particular value, which we set as being equal to one. So all the inputs to i are scaled by $\frac{1}{\text{sum of inputs to } i}$. In the case that there exist a sum of zero inputs to i, we set the sum of all inputs to be equal to one, guaranteeing that eventually the sum of all inputs to i, does not change as a result of synaptic scaling.

This involves us constantly scaling our W matrix every time-step of our simulation. This differs from our model where synaptic scaling occurs after every spike. The first step in synaptic scaling is summing all of the inputs to i. To do this, we sum all of the rows in W. That looks something like this:

Let $\mathbf{a}$ be a vector of all ones:

$$\mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Let's define:

$$\mathbf{m} = W\mathbf{a}$$

This equals the sum of all of the rows of W, representing the inputs to i. Therefore, we have:

$$\mathbf{m} = W\mathbf{a} = \begin{bmatrix} \sum_{i=1}^{100} \\ \sum_{i=2}^{100} \\ \vdots \\ \sum_{i=100}^{100} \end{bmatrix}$$

And then, component-wise, we see that

$$m_i = \sum_{j=1}^{100} w_{ij}$$

We set $m_i \neq 0$, because in that case we would be dividing by zero. In the case that $m_i = 0$, we change m_i to equal 1.

Next, dividing every entry in W by the sum of its particular row, will guarantee that the sums of the rows will add up to 1.

$$w_{ij}{}^{\mathrm{k+1}} = \frac{w_{ij}{}^{k}}{m_i}$$

This means that the program is constantly readjusting so that $\sum_{j=1}^{100} w_{ij} = 1$. We can convert this into vector form:

$$W^{\mathrm{k+1}} = \frac{W^k}{\mathbf{m}}$$

2.6.1 Implementation in Code

- `v=ones(size,1)`: Creates an initial 100×1 column vector of ones that is used to sum the rows of W. v' indicates the transpose of $\mathbf{v}$ in MATLAB.

- `m=W*v`: Multiplying W by our vector of ones gives us a 100×1 column vector, m, of the sums of the rows in W.
- `m(m==0) = 1`: Guarantees that no entry in m is equal to zero because in this case we would get a divide by zero error in the code.
- `m=1./m`: Takes the reciprocals of all of the elements in m.
- `W=W.*(m*v')`: Since MATLAB does not execute component-wise division, here we are using component-wise multiplication to turn our new m vector of reciprocals into a matrix, in which the column vector gets repeated 10 times horizontally. This operation makes it possible for us to multiply every entry in our W matrix by this new scaling value in our `m*v'` matrix.

2.7 Input

As we already discussed in Chapter 1, our expanding disc will be overlaid on top of a 10×10 graph of neurons, where each vertex represents a neuron. We see that the point $(10, 1)$ on the grid corresponds to neuron 1 in our network, $(9, 1)$ for neuron 2, $(8, 3)$ for neuron 3, etc. For the next ten neurons, we have that $(10, 2)$ corresponds to neuron 11, $(10, 3)$ for neuron 12, etc.

We define our **input** vector as a vector that gives us the times in which a neuron gets input. **input** is a vector of all zeros, initially. In our simulation it is a $20,000 \times 1$ column vector which results in our disc expanding in roughly 5 seconds of real time, though running for 20 seconds of simulation time. We see this because the dimension of **input** is determined by:

$$= \left\lfloor \frac{20}{\tau_{\mathrm{m}}} \right\rfloor = \left\lfloor \frac{20}{.001} \right\rfloor = 20,000$$

where τ_{m} is our rate of disc expansion. We are taking the floor function of the above value so that we have an integer value corresponding to the length of the **input** vector.

The program works as such: Let (x_c, y_c) be the randomly chosen center of a disc, where $1 < x_c, y_c < 10$ so that the center remains within our pixelated 10×10 grid, though never touches the edges.

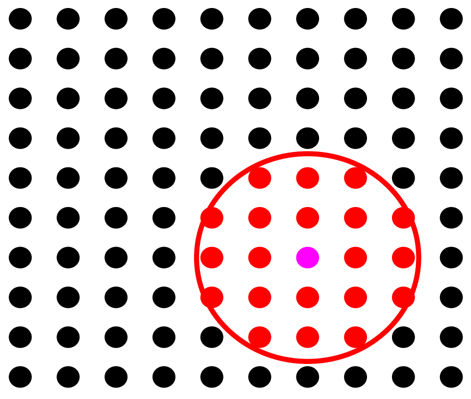
Figure 2.7.1: The magenta point indicates (x_c, y_c) our randomly selected center of the expanding disc. Red dots show neurons that have already received input. Black dots show neurons that haven't already received input.

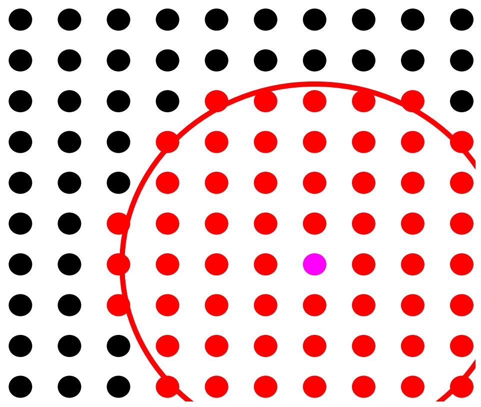
Figure 2.7.2: The caption of the figure to the left applies here. In this figure, we see the disc has now expanded since the last iteration, so that the radius is bigger implying that more neurons have received input.

Let's call our graph, G. Therefore, for every vertex on our grid $(x, y) \in G$ we are finding its distance from the center point (x_c, y_c). Using the distance formula, we have that: $a = (x_c - x)^2$, the horizontal distance and $b = (y_c - y)^2$, the vertical distance from the center point. Therefore, using substitution we have:

$$d = \sqrt{a + b}$$

Then using our scaling factor, to control the rate in which the disc expands, we modify accordingly:

$$d = \frac{\sqrt{a + b}}{T_m}$$

Next, we find use the floor function again to find the value of the distance, to guarantee that this will be an integer.

$$d = \left\lfloor \frac{\sqrt{a + b}}{T_m} \right\rfloor$$

Since d is an integer, this will help us determine which entry to change in our **input** vector:

$$\mathbf{input}(d) = 10(x - 1) + y$$

If $(x, y) = (9, 2)$, this means that $d = 379$ using the equations for a and b we previously mentioned. It follows that the 379th entry of **input** will become the value: $10(9-1)+2 \Rightarrow 82$, corresponding to the 82th neuron in our network.

Simply speaking, this value will tell us that after roughly 379 iterations, neuron 82 will get input. In other words:

$$\mathbf{c}^{379}(82) = \mathbf{c}^{378}(82) + 1$$

where **c** represents our current vector. When the disc has filled the whole input field, the program starts over.

2.7.1 Implementation in Code

- `global delta_t T_m size n k kreset maxZ input`: Keeps these listed variables as global variables in both this function and our original program.
- `kreset = k-1`: creates new system for iterating through the disc function, such that $1 \leq k - k_{\text{reset}} \leq 20,000$.
- `maxZ = 0`: An initial condition.
- `input = zeros(floor(20/Tm),1)`: initial input vector which keeps track of the times in which the neurons receive input.
- `d = floor(sqrt(a+b)/Tm)`: Computes the distance of all (x, y) points from the random center.
- `input(d,:) = (x-1)*10 + y`: Reorganizes entries of input vector so that they specify which neuron should get input for a given time step (d).
- `kreset=1`: so that $k - \text{kreset} > 0$
- `disc1()`: the name of our disc function. We call this function in the main program under the condition that `k-kreset==maxZ`.

- `C(input(k-kreset)) = C(input(k-kreset))+1`: Here is where the input gets added to the current vector, **c** under the condition that `input(k-kreset) = 0` . The current increases by 1 every time a neuron receives input.

2.8 Changing the Thresholds

As we saw in Chapter 1, we can write our differential equation for average spike history as:

$$\frac{dS_{\text{av}i}}{dt} = -\frac{S_{\text{av}i}}{\tau_{\text{Sav}}}$$

$S_{\text{av}i}$ decays exponentially but increases every time neuron i spikes. We can notate this instantaneous change in the average spike history for neuron i as:

$$\Delta S_{\text{av}i} = \frac{1}{\tau_{\text{Sav}}} \quad \text{when } i \text{ spikes}$$

Let's first begin by using Euler's method to convert our differential equation into a recurrence relation. We can represent the change in time, dt as Δt and S_{av} as the difference between two subsequent iterations of the program, k and $k+1$. Therefore, we have:

$$\frac{{S_{\text{av}i}}^{k+1} - {S_{\text{av}i}}^{k}}{\Delta t} = -\frac{{S_{\text{av}i}}^{k}}{\tau_{\text{Sav}}}$$

$${S_{\text{av}i}}^{k+1} - {S_{\text{av}i}}^{k} = -\frac{\Delta t}{\tau_{\text{Sav}}}{S_{\text{av}i}}^{k}$$

$${S_{\text{av}i}}^{k+1} = {S_{\text{av}i}}^{k} - \frac{\Delta t}{\tau_{\text{Sav}}}{S_{\text{av}i}}^{k}$$

$${S_{\text{av}i}}^{k+1} = (1 - \frac{\Delta t}{\tau_{\text{Sav}}}){S_{\text{av}i}}^{k}$$

Now, we can implement the instantaneous component to our difference equation. For every time neuron i spikes, our S_{av} vector increases by $\frac{1}{\tau_{\text{Sav}}}$. Therefore, it suffices to add an additional term to our difference equation. We have:

$${S_{\text{av}i}}^{k+1} = (1 - \frac{\Delta t}{\tau_{\text{Sav}}}){S_{\text{av}i}}^{k} + \frac{{s_i}^k}{\tau_{\text{Sav}}}$$

where s_i represents an element in our spike vector, equal to one in the case that neuron i did spike, and zero in the case that neuron i did not spike. To simplify this, let $r_{\text{Sav}} = 1 - \frac{\Delta t}{\tau_{\text{Sav}}}$. By substitution, we have:

$$S_{\mathrm{av}i}{}^{k+1} = r_{\mathrm{Sav}} S_{\mathrm{av}i}{}^{k}$$

where in our program,

$$r_{\mathrm{Sav}} = \frac{1 - .001}{1}$$
$$= 0.9990$$

We can write the above equation in vector form as:

$$S_{\mathrm{av}}{}^{k+1} = r_{\mathrm{Sav}} S_{\mathrm{av}}{}^{k} \frac{\mathbf{s}}{\tau_{\mathrm{Sav}}}$$

Now let's examine our threshold differential equation:

$$\frac{d\mathbf{v}_{\mathrm{max}i}}{dt} = \frac{1}{\tau_{\mathrm{th}} S_{\mathrm{t}}} (S_{\mathrm{av}i} - S_{\mathrm{t}})$$

where $\mathbf{v}_{\mathrm{max}}$ is our threshold for neuron i, $\mathrm{th}_{\mathrm{res}}$ measures how long it takes for threshold to go from 1 to 0 in the absence of any spiking and S_{t} is our target spiking rate = 10 spikes per second. We use Euler's method to convert this into a difference equation. We have:

$$\frac{\mathbf{v}_{\mathrm{max}}{}^{k+1} - \mathbf{v}_{\mathrm{max}}{}^{k}}{\Delta t} = \frac{1}{\tau_{\mathrm{th}} S_{\mathrm{t}}} (S_{\mathrm{av}} - S_{\mathrm{t}})$$
$$\mathbf{v}_{\mathrm{max}}{}^{k+1} - \mathbf{v}_{\mathrm{max}}{}^{k} = \frac{\Delta t}{\tau_{\mathrm{th}} S_{\mathrm{t}}} (S_{\mathrm{av}} - S_{\mathrm{t}})$$
$$\mathbf{v}_{\mathrm{max}}{}^{k+1} = \mathbf{v}_{\mathrm{max}}{}^{k} + \frac{\Delta t}{\tau_{\mathrm{th}} S_{\mathrm{t}}} (S_{\mathrm{av}} - S_{\mathrm{t}})$$

Let $\frac{\Delta t}{\tau_{\mathrm{th}} S_{\mathrm{t}}} = \mathrm{th}_{\mathrm{res}}$. We have:

$$\frac{d\mathbf{v}_{\mathrm{max}i}}{dt} = \frac{1}{\tau_{\mathrm{th}} S_{\mathrm{t}}} (S_{\mathrm{av}i} - S_{\mathrm{t}})$$

This equation works the way we want because when the average spiking rate of a neuron is greater than the target spiking rate, $\mathbf{v}_{\mathrm{max}}$ will increase by $\mathrm{th}_{\mathrm{res}}$, resulting in the following:

Case 1: $S_{\mathrm{av}} > S_t \Rightarrow$ Threshold increases and as a result, neurons have less likelihood to spike.

Case 2: $S_{\mathrm{av}} < S_t \Rightarrow$ Threshold decreases and as a result, neurons have more likelihood to spike.

Case 3: $S_{\mathrm{av}} = S_t \Rightarrow$ Threshold remains the same and activity stays constant.

2.8.1 Threshold Response Time

We create an exponential function for τ_{th}, our response time for the thresholds which will begin at 1 and exponentially increase to 100 throughout the simulation. 100 seconds means that in a completely dead network, the thresholds will go to zero in about 100 seconds. We define the following, τ_{relax} as how long it takes for τ_{th} to get to 100, where $\tau_{\text{relax}} = 10,000$ seconds. Let's begin by defining that when $t = 0$, a constant $C = 99$, which guarantees that:

$$\frac{d\tau_{th}}{dt} = \frac{C}{dt}$$

And since, we define: $\tau th = 100 - C \Rightarrow C = 100 - \tau_{th}$

Therefore, we can rewrite our differential equation as:

$$\frac{d\tau_{th}}{dt} = \frac{100 - \tau_{th}}{dt}$$

We see that the solution to the differential equation is:

$$\tau_{th}(t) = 100 - Ce^{\frac{-t}{\tau_{\text{relax}}}}$$

And

$$100 - \tau_{th}(t) = Ce^{\frac{-t}{\tau_{\text{relax}}}}$$

Converting this to a difference equation, we have:

$$\frac{d\tau_{th}}{dt} = \frac{100 - \tau_{th}}{dt}$$

$$\frac{\tau_{\text{th}}^{k+1} - \tau_{\text{th}}^{k}}{\Delta t} = \frac{100 - \tau_{\text{th}}^{k}}{\tau_{\text{relax}}}$$

$$\tau_{\text{th}}^{k+1} - \tau_{\text{th}}^{k} = \frac{100\Delta t}{\tau_{\text{relax}}} - \frac{\tau_{\text{th}}^{k}\Delta t}{\tau_{\text{relax}}}$$

$$\tau_{\text{th}}^{k+1} = \frac{100\Delta t}{\tau_{\text{relax}}} - \frac{\tau_{\text{th}}^{k}\Delta t}{\tau_{\text{relax}}} + \tau_{\text{th}}^{k}$$

$$\tau_{\text{th}}^{k+1} = \frac{100\Delta t}{\tau_{\text{relax}}} + \tau_{\text{th}}^{k}(1 - \frac{\Delta t}{\tau_{\text{relax}}})$$

$$\tau_{\text{th}}^{k+1} = 100 - (1 - \frac{\Delta t}{\tau_{\text{relax}}})(100 - \tau_{\text{th}}^{k})$$

Let $r_{\text{relax}} = 1 - \frac{\Delta t}{\tau_{\text{relax}}}$. We have:

$$\tau_{\text{th}}^{k+1} = 100 - r_{\text{relax}}(100 - \tau_{\text{th}}^{k})$$

We see this exponential decay with:

$$100 - \tau_{\text{th}}^{k+1} = r_{\text{relax}}(100 - \tau_{\text{th}}^{k})$$

This guarantees that the threshold response time is faster initially (1 second) and then relaxes to 100 seconds over the course of the simulation. This assures that the network isn't hyperactive.

2.8.2 Implementation in Code

- `T_th=1`: The threshold response time begins at 1 second and then relaxes to 100 seconds.
- `T_relax=10000`: A parameter of the model, τ_{relax} indicates how long it takes for τ_{th} to get to 100
- `r_relax=1-delta_t/T_relax`: A parameter of the simulation, r_{relax} is the "decay" factor for the threshold response time.
- `T_sav=1`: For every time neuron i spikes, our S_{av} vector increases by $\frac{1}{\tau_{\text{Sav}}}$.
- `r_sav=1-delta_t/T_sav`: A parameter of the simulation, where r_{Sav} is the decay factor for the average spiking vector.
- `St=10`: target spiking rate, which is a comparison value for the spiking average (S_{av}) vector.
- `Sav=St*ones(size,1)`: The initial spiking average vector, in which every neuron's averages spiking value begins at the target spiking rate ($S_{\text{t}} = 10$).
- `thresponse=delta_t/(T_th*St)`: Controls the responsiveness of the changing threshold.
- `threshold`: Refers to the initial threshold vector (of size 100×1), also called $\mathbf{v}_{\text{max}}$ in our model. Each element corresponds to the threshold value for each neuron. Initially starts at `threshold=1`.

- `T_th=100-r_relax*(100-T_th)`: The threshold response time eventually relaxes from 1 to 100 seconds.

- `threshold=threshold+thresponse*(Sav-St)`: This recurrence relation guarantees that if $S_{av} > S_t$ then the threshold will increase for a particular neuron. If $S_{av} < S_t$, then the threshold will decrease for that neuron.

- `Sav=r_sav*Sav+S/T_sav`: If a neuron didn't spike, since S_{av} is exponentially decaying, S_{av} will continue to decrease. If a neuron did spike, then S_{av} will increase. If $S_{av} = S_t$ then the threshold level will remain constant. If S_{av}, S_t, then the threshold will decrease causing the neuron to 'spike' more.

- `Sav(Sav>2*St)=2*St`: guarantees that guarantees that S_{av} will never be greater than 20 spikes per second.

3
Results

After running my simulation multiple times, slightly adjusting parameters, I found some noticeable differences in my network over time. While I didn't see any clear convergence to a strongly identifiable structure, I did notice that the majority of the synaptic weights converged to zero. For instance, running the program for $400,000$ seconds (approximately 111 hours of simulation, though taking only 19.6017 hours in real time) and setting the initial $\tau_{\text{th}} = 1$, I found that $1,898$ of the weights converged to precisely zero, comprising about 18.98% of the networks's connections. In addition, the majority of the connections that didn't die got stronger. We saw this in each of our four final simulations despite different τ_{th} values and durations of simulation.

In creating the below graphs, I designed a filter which would sort through the W matrix of synaptic weights depending on a threshold value I specified. All weights equal to the threshold value or higher than it, would get rounded to 1 and remain on the graph. All other weights would be rounded to 0 and would be omitted from the graph. The specific thresholds for each graph are specified in the figure comments below.

These graphs look interesting even after 10 seconds of simulation, when the network has just received two rounds of visual input. We see very clearly, that the neurons who just received input and spiked have a significantly large number of connections to other neurons. In particular, these neurons which we will call "popular" neurons, are spiking a lot and then sending large amounts

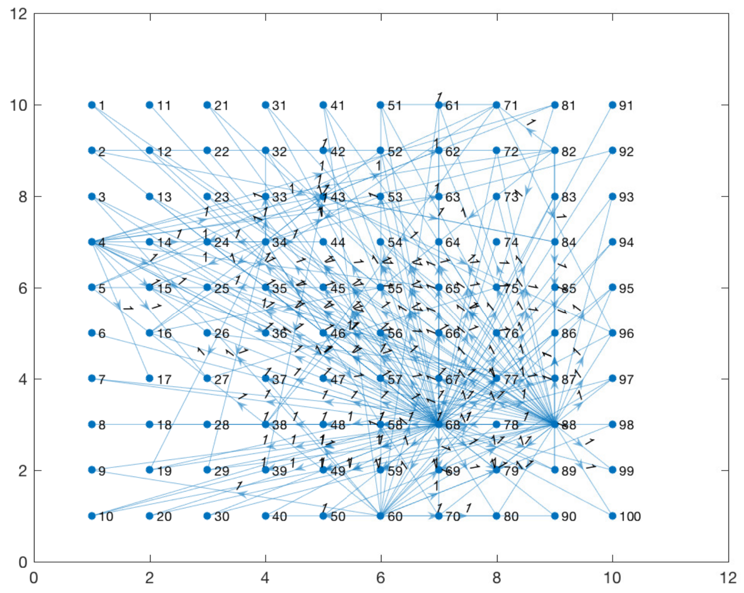

Figure 3.0.1: $\tau_{\text{th}} = 1$, $200,000$ sec simulation, threshold for figure $= .072$

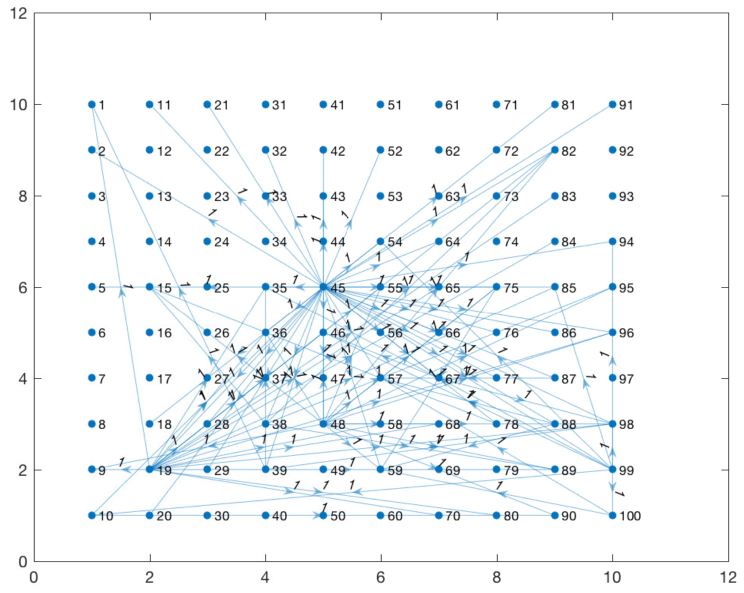

Figure 3.0.2: $\tau_{\text{th}} = 10$, $200,000$ sec simulation, threshold for figure $= .05$

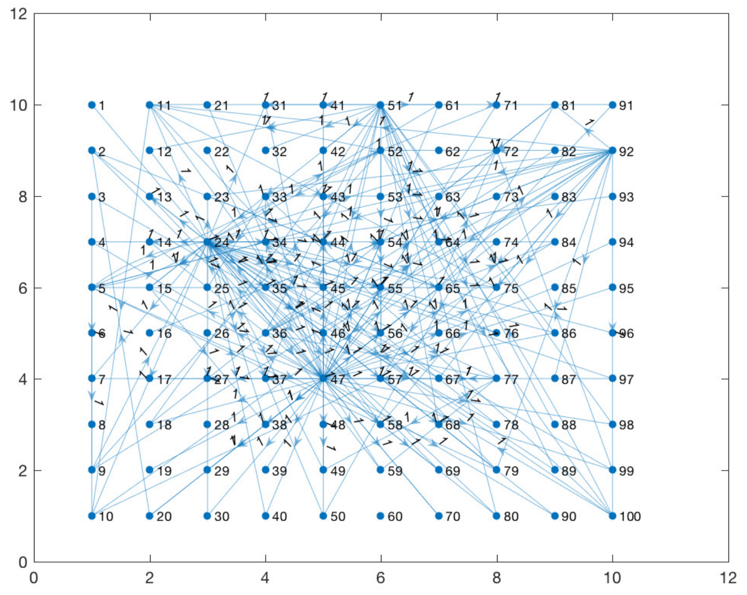

Figure 3.0.3: $\tau_{\text{th}} = 1$, $400,000$ sec simulation, threshold for figure = .082

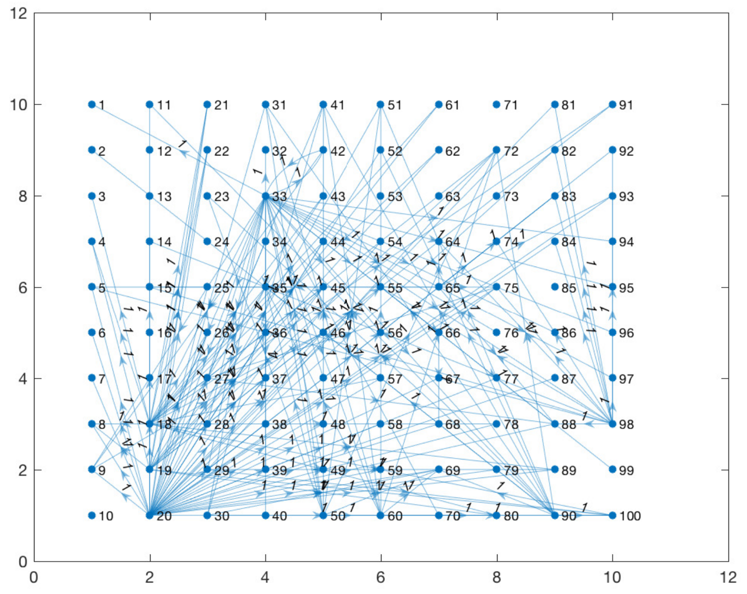

Figure 3.0.4: $\tau_{\text{th}} = 10$, $400,000$ sec simulation, threshold for figure = .07

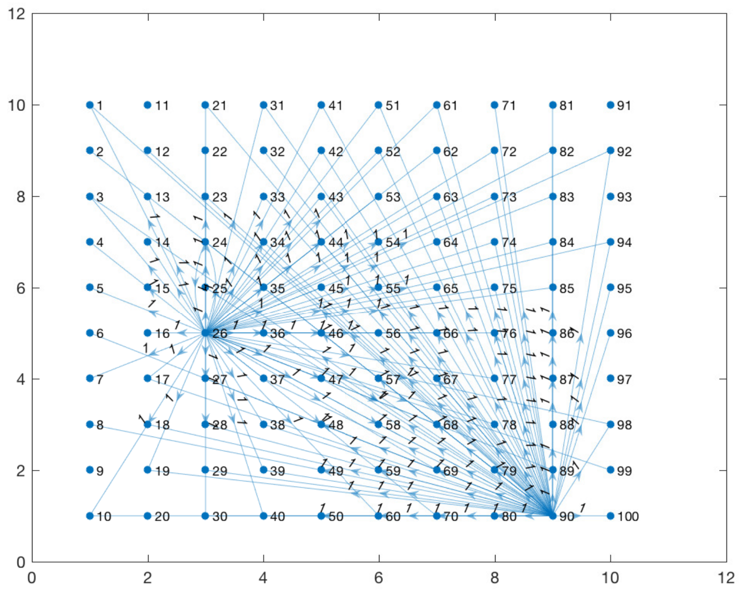

Figure 3.0.5: $\tau_{\text{th}} = 1$, $200,000$ sec simulation, threshold for figure = .055

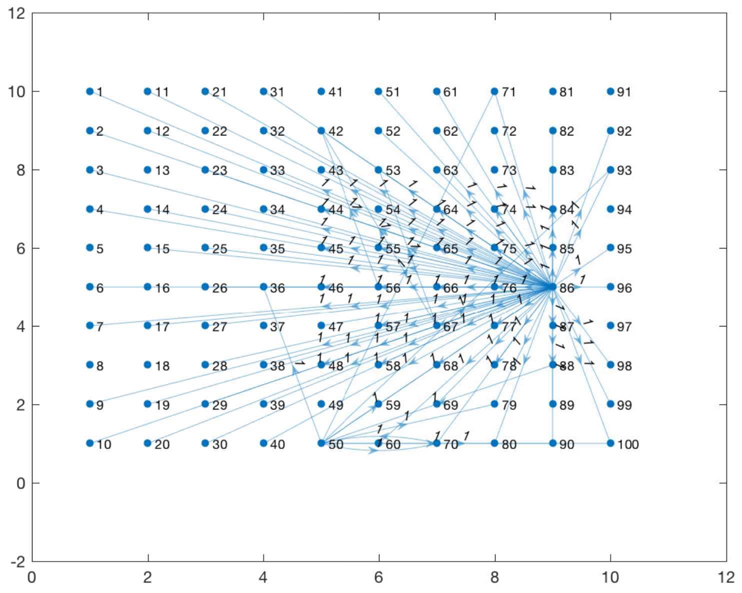

Figure 3.0.6: $\tau_{\text{th}} = 10$, $200,000$ sec simulation, threshold for figure = .055

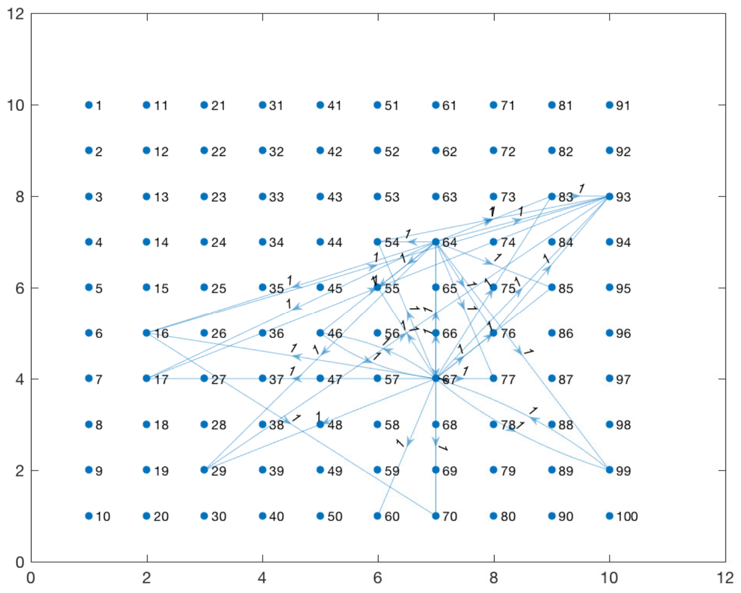

Figure 3.0.7: $\tau_{\text{th}} = 1$, $400,000$ sec simulation, threshold for figure = .05

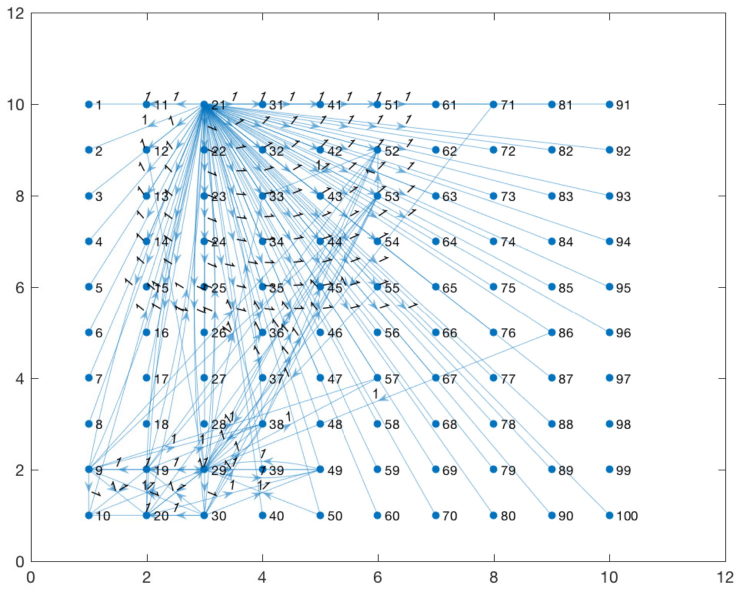

Figure 3.0.8: $\tau_{\text{th}} = 10$, $400,000$ sec simulation, threshold for figure = .05

of input to their neighbors. This is exactly how we intended our model to work. When a neuron spikes, it sends input to its neighbors. Interestingly enough, when the simulation is run for a longer duration, we see that neurons react less to incoming input, as the structure of the graph seems to resemble more of a grid-like pattern. Therefore, there is reason to suspect that the nodes of these graphs with significantly more edges than others become “popular” neurons at the very beginning of the simulation. Over time, neurons react less strongly to their spiking neighbors and the “learning” process plateaus. We can view this as a type of desensitizing.

However, this is just an observation and has not been entirely confirmed yet. In order to do this, we would need to record the weight matrices throughout the entire simulation to see the exact changes.

My resulting graphs of connections illustrate that a patterned visual stimulus is enough information for the neurons to, in a sense, “figure out” that they are organized in a grid and eventually use this point of reference to organize themselves into a grid. While my graphs don’t resemble precise grid-like structures, they appear to be much more grid-like than in the beginning of the simulation where every neuron was connected, showing that the existing edges are definitely converging to some stable point.

We implemented two different τ_{th} constant values: $\tau_{\text{th}} = 1$ and $\tau_{\text{th}} = 10$, to see if the later would help our network converge faster. From the graphs below, this hypothesis seems like an accurate one, as these are more sparse.

If I had more time, I would consider playing around with the parameter values to investigate to what extent I could get my graphs to look precisely like grids. In addition, I’d like to keep a record of synaptic connections from the beginning to the end of the simulation to track exactly how the network self-organizes. While I initially tried this, it gave such a considerable lag to the program, that I unfortunately decided to disband it. In addition, I’d like to continue testing which learning rules are most important in the convergence towards this stable state, by omitting one at a time and seeing if the graphs converge to the same topological structure.

Regardless, our graphs illustrate the maturation of a previously unrefined network and give us insights into how neural networks learn during their critical period of development. Our graphs give even more reason to believe that brains are plastic and ever-changing. We can determine that it is more likely than not that an infant's brain comprises of random connections that change as a response to external stimuli, rather than a prefixed architecture of connections.

Appendix

3.1 Disc Input Function

```
function [ ] = disc1()

global delta_t T_m size n k kreset maxZ input


        kreset = k-1;

        x_c=rand*9+1;

        y_c=rand*9+1;

        maxZ = 0;

        input = zeros(floor(20/T_m),1);

        for x=1:n

            for y=1:n

                a = (x_c-x)^2;

                b = (y_c-y)^2;

                d = floor(sqrt(a+b)/T_m);

                input(d,:) = (x-1)*10 + y;

                if d > maxZ

                    maxZ = d;
                end

            end

        end

    end
```

3.2 Final Simulation Program

```
clear all


%profile ON


% This program simulates a network of a specified number of neurons using

% a Current-Based LeaKy Integrate and Fire Model. Each neuron is connected

% to all neurons in the networK, except itself via randomly generated

% synaptic weights. Input in our difference equation is negligible.


changingthreshold=1; %1 implies that threshold is changing, 0 implies that threshold

is constant.

currentmodel=1; %1 runs Leaky Integrate with Current model, 0 runs simple Leaky Integrate

and Fire model

saverandomness=0; %1 saves random variables from the last run, 0 generates new

random variables.


global delta_t T_m size n k kreset maxZ input

delta_t = 0.001; %time step in seconds, delta_t is equal one millisecond

T_m = delta_t; %speed of disc expansion

size = 100; % number of neurons in networK

n = sqrt(size);
```

```
kreset=1;

maxZ=0;

input=zeros(floor(20/T_m),1);

x_c=rand*9+1;

y_c=rand*9+1;

Z=zeros(sqrt(size));

tmax = 200000; %runs program for 200,000 seconds

maxstep=tmax/delta_t; % maximum length of iterations


% neuron setup

T_v=.01; %the Voltage for a single neuron decays by a factor of e in 100ms.

T_c=.01; %the Current for a single neuron decays by a factor of e in 100ms.

r_v = 1 - delta_t/T_v; % decay factor of voltage for each neuron at each step

Vreset = 0; % resetting value of voltage after spiking

r_c = 1 - delta_t/T_c; % decay factor of current for each neuron at each step

V_r=exp(1)/T_v; %V_r is the max point of the V curve (solution to dV/dt).

Vresponse=V_r*delta_t; % regulates how much current is released


% random weighted adjacency matrix

if saverandomness==1  %1 saves random variables from the last run, 0 generates new random
variables.

    s = rng;

end

Wscale=2/(size-1); % scaling value of W adjacency matrix

W=Wscale*rand(size); % creates an adjacency matrix of randomized synaptic weights

for i=1:size

    W(i,i)=0;    % weights along the diagonal should be zero
```

```
end


% initial conditions

Vscale=0; % maximum initial voltage for each neuron

V = Vscale*rand(size,1); % initial vector of voltages

S=zeros(size,1); % initial S vector (essentially input)

C=zeros(size,1); % initial vector of currents (start with 0 current)


% parameters for threshold change

if changingthreshold==1 %1 implies that threshold is changing, 0 implies that threshold

is constant.

    T_th=1; %Threshold response time, begins equal to 1

    T_relax=10000;

    r_relax=1-delta_t/T_relax;

    T_sav=1; %in seconds

    r_sav=1-delta_t/T_sav;

    St=10;  % S target: the comparison value for Sav vector

    Sav=St*ones(size,1); % S average: initial Sav starts at St

    thresponse=delta_t/(T_th*St);   % responsivity of the threshold

    threshold = ones(size,1); % initial thresholds start at 1

else

    threshold=1; % in the case where changingthreshold=0, the threshold is set to a

    constant value for all neurons.

end


%STDP

p=zeros(size,1);
```

```
%precord = ones(size,maxstep);

v=ones(size,1); %used for synaptic scaling

T_p=-.05/log(1/10); %e-folding time

r_p=1-(delta_t/T_p);

Wchange=.0001;

Wspike=Wchange/(T_p*St);


% record keeping, for debugging and inspection only

Srecord=zeros(size,1);

Vrecord=[zeros(size,1),V];

numgraphs=4;

Vgraph=[zeros(numgraphs,1),V(1:4)];

Crecord=ones(size,1);

Savrecord=zeros(size,1);

T_threcord=zeros(maxstep,1);

threcord=zeros(size,1);

W1=zeros(size);

W2=zeros(size);

W3=zeros(size);

W4=zeros(size);

W5=zeros(size);

record1= maxstep/20000;

record2= maxstep/2000;

record3= maxstep/200;

record4= maxstep/20;

record5= maxstep/2;
```

```
%Input
%I=zeros(size,1);
N=zeros(size,1);

if saverandomness==1 % holds random variables from previous run.
    rng(s);
end


for k = 1:maxstep
        if k-kreset==maxZ
            disc1();
        end

         if currentmodel==1
            %if input(k-kreset) ~= 0
             %   I(input(k-kreset)) = I(input(k-kreset))+1;
                %N(input(k-kreset)) = N(input(k-kreset))+1;
            %end;
            %reshape(N,[10,10]) %for viewing input/debugging
          % N = N*0.99;
            C= r_c*C+W*S;
            if input(k-kreset) ~= 0
                C(input(k-kreset)) = C(input(k-kreset))+1;
               % N(input(k-kreset)) = N(input(k-kreset))+1;
                %reshape(N,[10,10])
            end
    %Crecord=[Crecord,C];  %for debugging

     V=r_v*V+Vresponse*C;

 else

     V=r_v*V+W*S;

 end


 %Vrecord(:,k)=V; %Vrecord=[Vrecord,V]; % for debugging

 %Vgraph=[Vgraph,V(1:4)];


 if changingthreshold==1

     T_th=100-r_relax*(100-T_th);

     threshold=threshold + thresponse*(Sav-St);

     %threcord(:,k)=threshold; %for debugging

 end


 S(1:size) = 0;

 S(V>threshold) = 1;

 %Srecord=[Srecord,S]; % for debugging


if changingthreshold==1

    Sav=r_sav*Sav+S/T_sav;

    Sav(Sav>2*St)=2*St; %guarantees that Sav will never be more than 20

    %T_threcord(k)=T_th;

    %Savrecord=[Savrecord,Sav];

end
```

```
        %STDP

        p=r_p.*p + S;

        W=W+Wspike*(S*p'-p*S');


        %Weight Cut off, gurantees that weights are between 0 and 1

        W(W<0)=0;


        %Synaptic Scaling

        m=W*v; %sums the columns of W

        m(m==0) = 1;

        m=1./m;

        W=W.*(m*v');


        %for b = 1:size

         %   W(b,:) = W(b,:) / m(b);

        %end


        V(V>threshold)=Vreset;


        %record keeping of W matrix after select time intervals

%         if k==record1;

%             W1=W;

%         end

%

%         if k==record2;

%             W2=W;

%         end
```

```
%
%          if k==record3;
%              W3=W;
%          end
%
%          if k==record4;
%              W4=W;
%          end
%
%          if k==record5;
%              W5=W;
%          end

end
```

Bibliography

[1] James Meiss, *Dynamical system*, `http://www.scholarpedia.org/article/Dynamical_systems`.

[2] Ciarleglio, Khakhalin, Wang, Constantino, Yip, and Aizenman, *Multivariate analysis of electrophysiological diversity of visual neurons during development and plasticity*, Elife **4** (2015), e11351.

[3] OLeary, Williams, Franci, and Marder, *Cell types, network homeostasis, and pathological compensation from a biologically plausible ion channel expression model*, Neuron **82** (2014), no. 4, 809–821.

[4] Gina Turrigiano and Sacha Nelson, *Homeostatic plasticity in the developing nervous system*, Nature Reviews Neuroscience **5** (2004), no. 2, 97–107.

[5] John A. White, *Action Potential*, Encyclopedia of the Human Brain, 2002, pp. 1 - 12, DOI https://doi.org/10.1016/B0-12-227210-2/00004-2.

[6] Helga Kolb, *Simple Anatomy of the Retina*, `http://webvision.med.utah.edu/book/part-i-foundations/simple-anatomy-of-the-retina/`.

[7] Brunel and Van Rossum, *Lapicques 1907 paper: from frogs to integrate-and-fire*, Biological cybernetics **97** (2007), no. 5-6, 337–339.

[8] Fourcaud-Trocmé, Hansel, Van Vreeswijk, and Brunel, *How spike generation mechanisms determine the neuronal response to fluctuating inputs*, Journal of Neuroscience **23** (2003), no. 37, 11628–11640.

[9] JM Berg, JL Tymoczko, and L Stryer, *Section 32.3, Photoreceptor Molecules in the Eye Detect Visible Light*, W H Freeman, New York, 2002.

[10] D Purves, GJ Augustine, and D Fitzpatrick, *Excitatory and Inhibitory Postsynaptic Potentials.*, Sinauer Associates, MA, 2001.

[11] Turrigiano and Nelson, *Hebb and homeostasis in neuronal plasticity*, Current opinion in neurobiology **10** (2000), no. 3, 358–364.

[12] Fontaine, Peña, and Brette, *Spike-threshold adaptation predicted by membrane potential dynamics in vivo*, PLoS computational biology **10** (2014), no. 4, e1003560.

[13] Chavez-Noriega, Halliwell, and Bliss, *A decrease in firing threshold observed after induction of the EPSP-spike (ES) component of long-term potentiation in rat hippocampal slices*, Experimental brain research **79** (1990), no. 3, 633–641.

[14] Tung, *Topics in Mathematical Modeling*, W H Freeman, 2007.