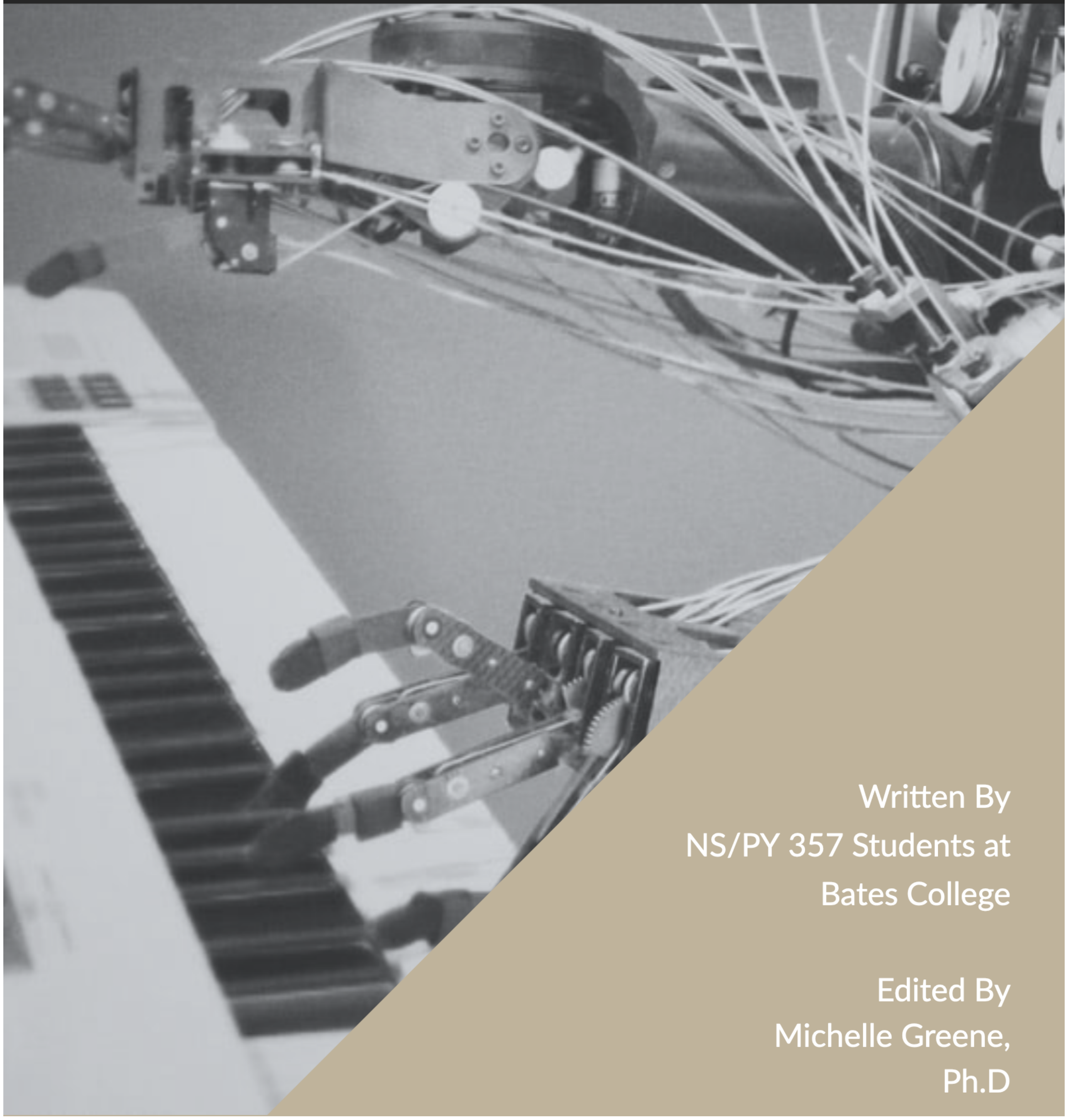
12-2019

Computational Neuroscience

Michelle Greene
Bates College, mgreene2@bates.edu

NS/PY 357 Students
Bates College

Computational Neuroscience



Written By
NS/PY 357 Students at
Bates College

Edited By
Michelle Greene,
Ph.D

Computational Neuroscience

Students of NS/PY 357 Bates College

2019-12-20

Contents

Chapter 1

Preface

What you are about to read is an open textbook written for (and by) students of Computational Neuroscience at Bates College. This is version 0.0 of a living document that will be revised, reused, and appended over the course of many generations of this course. As the instructor of this course, I want to briefly outline *my* motivations for undertaking this project in hopes that this idea may spread.

1.1 This book is free (as in pizza)

It is broadly accepted that college affordability is a key challenge for the U.S. in the 21st century. One of the drivers of increased college cost is the increased cost of course textbooks. Over the past 40 years, textbook prices have risen over 1200% over the last 40 years – much higher than the rate of inflation, and higher even than housing or healthcare! By creating a free textbook, we are broadening the participation of students in computational neuroscience.

1.2 This book is free (as in speech)

As important as cost-free textbooks is, equally important are the freedoms that openness provides. We are opening this resource for reuse, revision, and redistribution. We welcome others to remix into other works. It is my belief that the availability of high-quality resources allows for creativity and innovation to spring up in others. My teaching and scholarship has benefitted greatly from openly available sources, and I feel that my success as an academic is to pay this forward.

1.3 This book can be revised and disseminated more rapidly than traditional textbooks

Part of the impetus of this book came from a frustration in finding a traditional textbook that was appropriate for my undergraduate, 300-level course in computational neuroscience. Many of the books, though excellent, assumed a graduate-level sophistication in mathematics. Nearly all were missing some of the most modern topics. Computational neuroscience is a rapidly-evolving field, so an open textbook allows for more rapid editing, addition, and dissemination than is afforded by a traditional publishing model.

1.4 This book creates a public record of learning that exists after the semester ends

Part of the educational journey is making the leap between being a *consumer* of knowledge to being a *generator* of knowledge. It is oft-said but nonetheless true statement that one truly learns by teaching. This assignment places students in the role of teacher, making the content come alive by explaining it in their own words. All too often, the writing that we do in college is in the form of the "disposable assignment" - one that students will spend a few hours working on, that I will spend a few hours reading and grading, and then is thrown away. Writing an open textbook is more of a renewable assignment - one that will have value in the world long after the semester is over.

We hope that you enjoy this book. Please feel free to reach out to me if you have any questions or comments about our work: mgreene2@bates.edu.

1.5 Authors

Juliet Bockhorst (2022)
Abraham Brownell (2020)
Paloma Noriega Burrill (2021)
Catherine Crossin (2020)
Leo Crossman (2020)
Logan Douglas (2020)
Nick Antonellis (2021)
Robin Kass (2020)
Sasha Cadariu (2021)
Wuyue Zhou (2021)

Chapter 2

What is Computational Neuroscience?

2.1 Vocabulary

- Algorithm
- Bottom-up processing
- Computational neuroscience
- Computational theory
- Emergent phenomena
- Hardware and implementation
- Hebbian learning
- Reconstruction
- Reductionism
- Reductionism
- Representation
- Top-down processing
- Turing machine

2.2 Introduction

Look at this picture of a desk on the page in front of you. As you look at it, your brain is somehow able to turn the raw sense data coming from your eyes into a judgment about the identity of the object in the image. If we want to build a machine that can demonstrate a similar capacity to judge — then we need to be able to model and understand the mechanisms that are at work when we see the desk. We are now left with at least two questions: *(1) How might we create a computer program that performs the same cognitive tasks as us humans?* And (2) *What can such as program tell us about the mechanisms at work in our own human brains?* While these questions may not be exhaustive of the concerns of computational neuroscience, they should at least give you a taste of some of the issues the discipline grapples with on a regular basis.
For more information please explore.

2.3 What is computational neuroscience?

Computational neuroscience is an interdisciplinary field that applies the principles of mathematics, philosophy, and computer science to study the inner workings of the brain. Computer models are critical to computational neuroscience, because they allow experiments to be conducted in a highly controlled and replicable fashion. In this context, a "model" is a simplified and simulated version of a system that tries to guess how the actual (simulated) system would behave in the real world.

For example, suppose a computational neuroscientist wants to understand how the human brain begins to make sense of sounds. A computer model could be constructed for this purpose, because many disparate aspects of the hearing parts of the brain have been measured. Such measurements would make constructing a useful computer model possible, because they would constrain the design model. In other words, our researcher could design a model where the simulated features match the measurements of the corresponding real features of the brain. This model could be useful, because our researcher has access to all the features of the computer model—including those that could not be easily and ethically measured in the actual human brain. This utility would be borne out in plausible inferences about currently unmeasured properties and behaviors of the brain.

While it is true that the inferences we've just discussed can not be made with complete certainty, they can be instrumental in guiding future research as new technology (and sources of funding) become available. Even if a given model ends up not holding up to future data, the model could still prove useful for developing artificial intelligences. With the potential of computer models of the brain in mind, it may be tempting to think you could build a model that truly "understands" in the same way that a person does. The question of whether or not such a model is possible is a matter of much debate, so suffice it to say we will only briefly survey the issues here.

On the side arguing that a computer model could never truly understand something (e.g. the Chinese language) the way a human does is the philosopher John Searle. Searle makes use of his famous "Chinese Room Argument" to suggest that merely following a set of rules to produce a desired result from a given input is not enough to count as true understanding. For example, Searle would argue that using a big book of rules for writing Chinese answers that respond to Chinese questions is not the same thing as having a natural conversation in Chinese.[1]

This may seem intuitively true, but many of Searle's opponents[2] argue that Searle's alleged argument is only intuitively true. That is to say that Searle is merely provoking intuitions rather than supplying premises or facts that lead to his desired conclusion.

The motivations for this debate could be explained in terms of the level of organization at which Searle and his opponents appear to be thinking.

While Searle published his paper on the "Chinese Room Argument" in 1980, the conversation of whether computers would be able to fully understand humans had been ongoing for many decades prior. In 1936, Alan Turing, created the theory of the Turing test to find when there was an equivalence between artificial intelligence and humans. Turing found the limitations of computation by understanding the limitations of humanity, and thus he created the **Turing Machine.** This machine paved way for the turing test, that tested whether a human could determine whether they were interacting with another machine or another human. Hence, the question of how to tell when artificial intelligence will be comparable to human intelligence is an ongoing problem today.

[1]See https://plato.stanford.edu/entries/chinese-room/ for more on this thought experiment.

[2]E.g. Dennett, Thagard, and Pinker to name a few.

Figure 2.1: Example of Conway's Game of Life

Exercise: Try out Conway's Game of Life to explore more into the implications of the Turing Machine and computer simulations.

2.4 Levels of organization

In 1982, David Marr, introduced a new approach to analysis. He believed that there were three levels in the model of the brain. The first level is the **computational theory**, , which is a description of the information going into the system, and the corresponding output desired from the system. An example of this is addition. The input is two numbers, and the desired output is the sum of those numbers. The second level consists of the representational scheme and the algorithm. The **representational scheme** is the description of the functional elements that are used in the computation, while the **algorithm** is the set of operations that are performed with or by those elements in order to carry out the transformation specified by the computational theory. One example is a cookbook recipe; this will define a step by step process (an algorithm) for how to produce a product given a set of clearly defined ingredients (a representational scheme). The third level is **hardware implementation**, which refers to the physical machinery that realizes the algorithm.

Exercise: Are you smarter than a computer?
Let's think about simple addition using two different implementation levels.

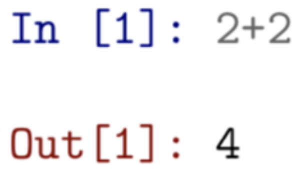
- Take a piece of paper and use it to add 5 to 5.

- Now, instead of using paper, add 3 to 7 with your fingers.

  Now consider how you solved these two problems. The method in which you performed the computation differed (on an implementation level), but the mental math and process of addition is the same. You took the input (the two numbers), processed them through an algorithm (symbolized by the addition symbol), and your output, hopefully, was 10.

The goal of computational neuroscience is to be able to replicate the functions of the brain, such as the one you just performed, in a non-organic setting. One of the ways to do this is through computer programming software, such as the application Python.

Consider the following basic code:



```
In [1]: 2+2

Out[1]: 4
```

Here is an example of an addition problem coded in Python. We can see the input, the algorithm which is transforming the input, and then the correct output of 4. Given the properties of coding software such as Python, which takes input, and runs it through a set algorithm. Is this similar to the way we humans do it? If so, how? If not, why not?
Applications of computational neuroscience
As we delve deeper into Computational Neuroscience, we will find that the field has a variety of potential applications when it comes to understanding how cognition happens. Computational Neuroscience, perhaps most importantly, allows us to create models of our cognitive processes, such that they are able to capture the basis of complex phenomena in a simple way. The brain is an extremely complex organ, and while we may not always understand all of its architecture and functionality, by using methods of Computational Neuroscience we are able to abstract certain notions to the extent that they become comprehensible. In doing so, we develop the ability to understand interactions between neurons in the brain and begin seeing the nature of certain causal relationships. Furthermore, we can begin to predict how complex systems in the brain will behave when presented with particular stimuli. This may all seem rather abstract, so let's think of an example we often take for granted: Vision. How is it that we are capable of recognizing a variety of highly specific things and distinguishing them from one another? What constitutes recognition and the neural processes behind it? How do we decide what deserves recognition and what doesn't? Why is it so difficult to replicate these seemingly innate processes in a robot? All these questions can be addressed using Computational Neuroscience. We must, however, keep one thing in mind. When creating models, the correct level of simplicity is difficult if not impossible to discern. So, it is important to engage in criticism and scrutiny when developing the simplest and most efficient model one can think of.

You may be saying to yourself, "Ok, well all this is great in theory, but where do we begin?". We start by trying to establish **Emergent Phenomena.** Emer-

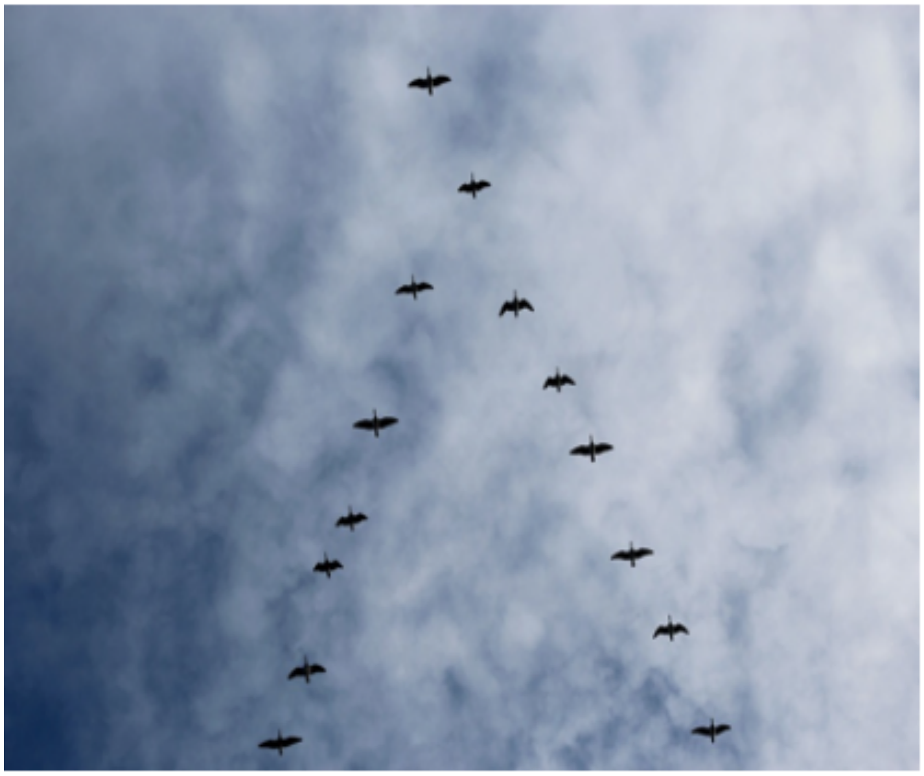


Figure 2.2: One example of emergent phenomena are the flight patterns of geese. One goose alone flies as it wishes but a collection of geese come together to form a v-shaped pattern that affects the overall movement of the geese.

gent Phenomena allow us to reframe composite systems such that we are able to understand their underlying mechanisms in simpler terms. To clarify, an Emergent Phenomenon can highlight both the mechanics and the nature of a particular system. For instance, a normal car cannot function properly unless it has four wheels. However, the car also derives part of its “car-ness” from the fact that it has four wheels. There are two primary schools of thought in this domain: **Reductionism** and **Reconstructionism.** Reductionism is the idea that in order to understand a given complex system, we must be able to reduce it to its simplest form, while Reconstructionism claims that we go in the opposite direction and reconstruct the system such that we are able to capture its complexity. After all, when we create models of the brain, it is not sufficient to explain their architecture. We must also show how the architecture gives rise to certain relationships and interactions. To do this, we can design our models using one of the following approaches: **Top-down processing** or **Bottom-up processing.** Top-down processing makes us design our models with a certain purpose, or goal in mind. Bottom-up processing leads us to establish a base of information or data before creating the model, and then creating the model and its purpose off of what we have collected.

Exercise: Explain the difference between the top-down and bottom-up philosophy.

2.5 The future of computational neuroscience

Given all this information, what is the future of Computational Neuroscience? Interest in the field is increasing steadily and everyday the range of its possible applications grows. In 2013, the Obama administration began the BRAIN initiative, a program designed to facilitate the development of innovative technologies that allow for a well-rounded and versatile understanding of brain function. In 2005, a Swiss team of scientists began another initiative called the Blue Brain project whereby they reconstructed the mammalian brain using simulations in order to generate a comprehension of the basic underlying principles of brain function and architecture. Computational Neuroscience becomes more relevant everyday and allows us to tackle difficult issues like the complexities of **Hebbian Learning** and Neural Networks. The implications of the field are perhaps unparalleled by any other fields of scientific inquiry, in that they may hold the answers to the creation of true Artificial Intelligence and the understanding of Consciousness and perception in the human brain.

2.6 Summary

Every person has a brain, but that doesn’t mean we understand what the brain is, or what it does. The brain is a complex organism with many facets to un-

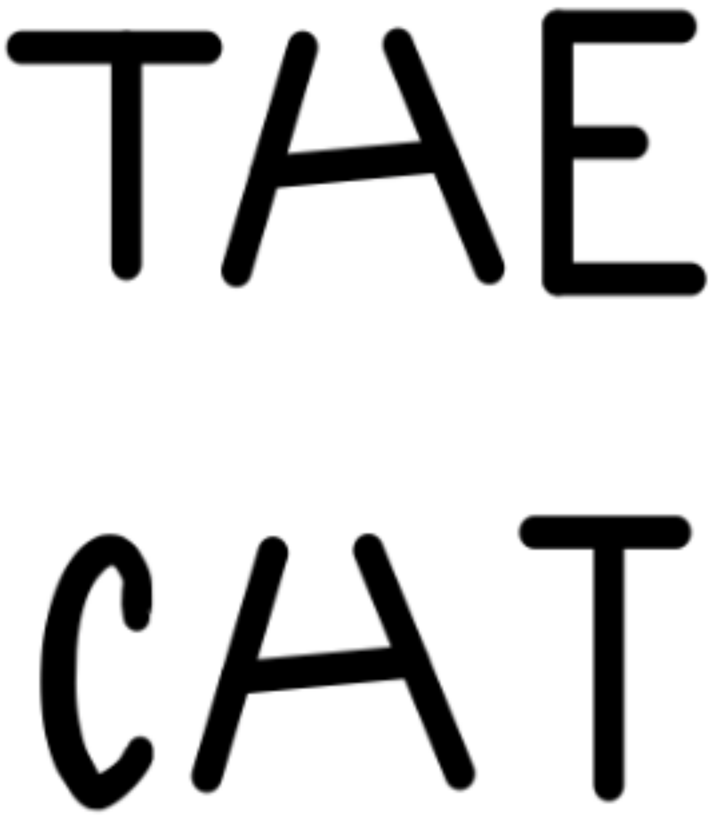


Figure 2.3: This image can be approached in two different ways. For bottom-up processing, we can see the letters first and then figure out the words. For top-down processing, we can see the words and decide what each letter is.

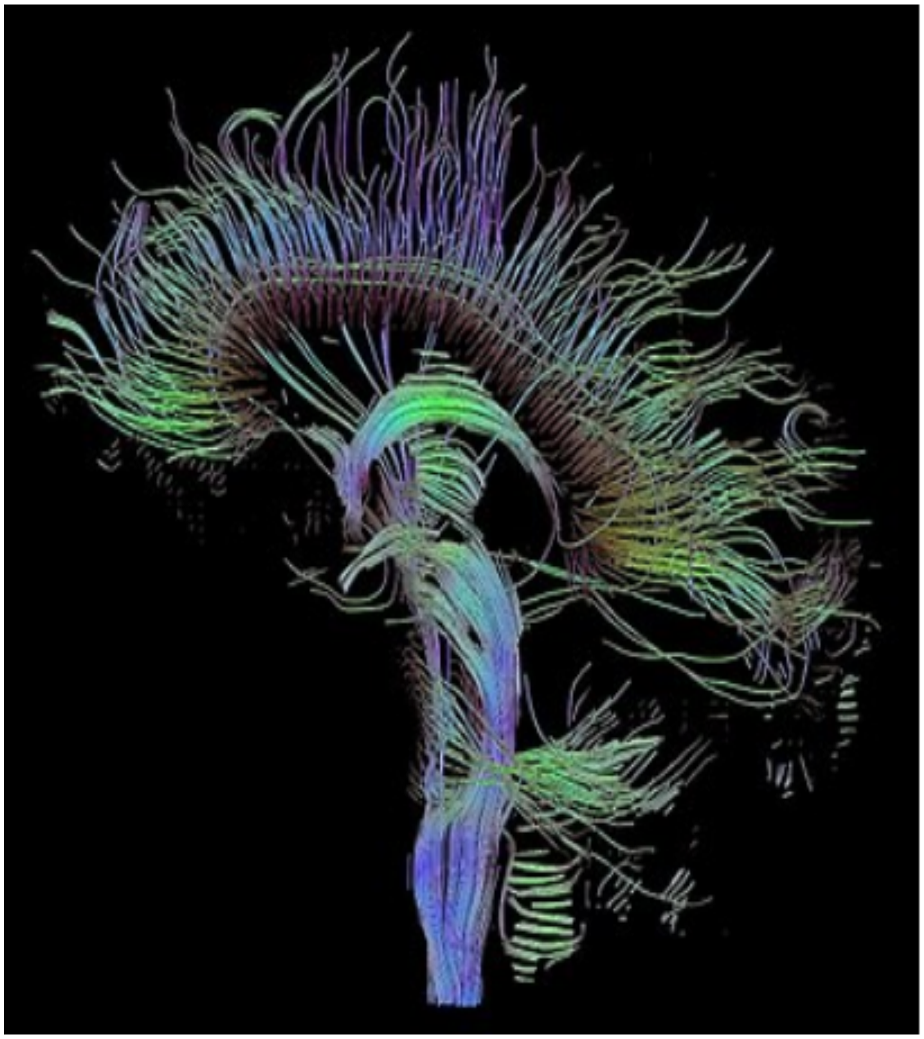


Figure 2.4: Applying Hebbian theory to computational neuroscience allows us to envision the connections between various neurons in the brain, displayed here is what is called a ‘connectome’.

derstand, and to fully comprehend; one must understand the mechanics and the reasoning behind each component. Whether studying in a top-down process, or a bottom-up process, one must keep in mind Marr's three levels of investigation: computational theory, representation and algorithm, and hardware and implementation. Computers and brains are different organisms, but by studying them alongside one another, a deeper understanding can be elicited. Turing introduced the world to the concept of using humans as a test of how to understand artificial intelligence, but we can now use artificial intelligence to understand the brain as well.

2.7 Exercises:

1. Which better explains and represents cognitive phenomena: the top-down approach, or the bottom-up approach?

2. Characterize the concept of emergent phenomena. Is human sentience that kind of phenomenon?

3. Does artificial intelligence have to have emotions to be sentient?

Chapter 3

Hodgkin and Huxley Model

3.1 Vocabulary

- Depolarization
- Positive Feedback
- Hyperpolarization
- Negative Feedback
- Membrane Potential
- Sodium-Potassium Pump
- Nernst Potential
- Reversal Potential
- Equilibrium Potential
- Driving Force
- Conductance
- Leak Current
- Leaky Integrate and Fire Model
- Absolute Refractory Period
- Gating variable

3.2 Introduction

Before you read this chapter, we would like to draw your attention to this video. We call this a Zombie Squid because the squid is in fact dead; however, it is recently deceased. Since the squid passed shortly before, Adenosine triphosphate (ATP) energy stores are still available to the squid's muscles. When the soy sauce, which has a lot of sodium chloride (salt) in it, is poured onto the squid, the salt in the soy sauce causes a voltage change which causes the squid's muscles to contract. Thus, we have a Zombie Squid.

So why is the Zombie Squid important? The Hodgkin-Huxley Model, said to have started the field of computational neuroscience, all hinges on the giant axons of squid. In the 1950s Alan Hodgkin and Andrew Huxley built a model that shows us how computers can successfully predict certain aspects of the brain that cannot be directly studied. The two even won a Nobel Prize in Physiology or Medicine in 1963 with Sir John Carew Eccles for their model. The Hodgkin-Huxley Model is now the basis of all conductance-based models. As a result, we can now understand how an action potential works, and why it is an all-or-none event.

While Hodgkin and Huxley created their model in the 1950s, the first recording of an action potential was done by Edgar Adrian in the 1920s. However, the first person to realize that neurons communicate via electrical signals came much earlier in 1791, when Luigi Galvani found that electricity from lightning or primitive batteries can cause a dead frog's leg muscle to contract. This led to a good amount of Frankenstein-like science with interested parties running electricity through dead bodies in an attempt to bring them back to life. However, the next truly scientific discovery came from Hermann Helmholtz in the 19th century. Helmholtz found that he could measure the speed of muscles contracting when he stimulated the nerve linked to that specific muscle. The Hodgkin-Huxley Model was then created once Adrian noted that not only were action potentials discrete, but the firing rate (spike per second) increased as stimulation to the nerve increased.

Exercise: Briefly explain how the Zombie Squid moves. Why is this important to understand?

3.3 What is an action potential?

Before going into ways of modelling action potentials, let's further explore what an action potential is. Within a cell, there are more sodium ions outside the membrane, which have a positive charge. Because there are less positive ions inside the cell compared to the outside, the inside of the cell has a negative resting potential. When there is a spike of voltage, that causes both voltage-gated sodium and potassium channels–meaning that these channels activate and inactivate at certain voltages–to activate on a neuron's membrane. However, the sodium channels open much faster than the potassium channels. The flow of sodium ions into the cell causes the membrane potential to become more positive. This process of positive ions flowing into the cell is called **depolarization**. When depolarization happens, it causes additional sodium channels to open, which causes further depolarization—this phenomenon is called **positive feedback**, and the positive feedback starts if the depolarization hits a set threshold. In other words, a positive feedback loop is a process that perpetuates itself. The positive feedback loop ends once voltage-gated potassium channels also open at the peak of the action potential and potassium begins flowing out of the cell

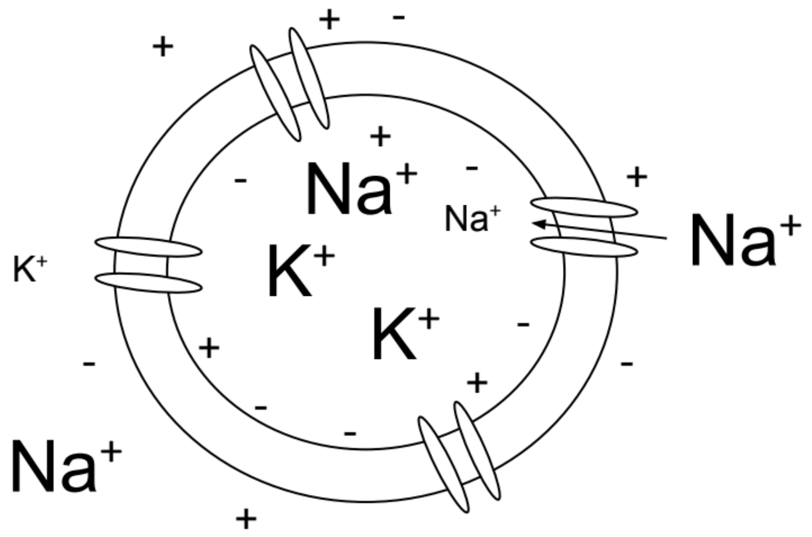


Figure 3.1: A cell at rest has more potassium ions intracellular than extracellular and more sodium ions extracellular than intracellular. There is a negative net charge within the cell being maintained by the voltage gradient.

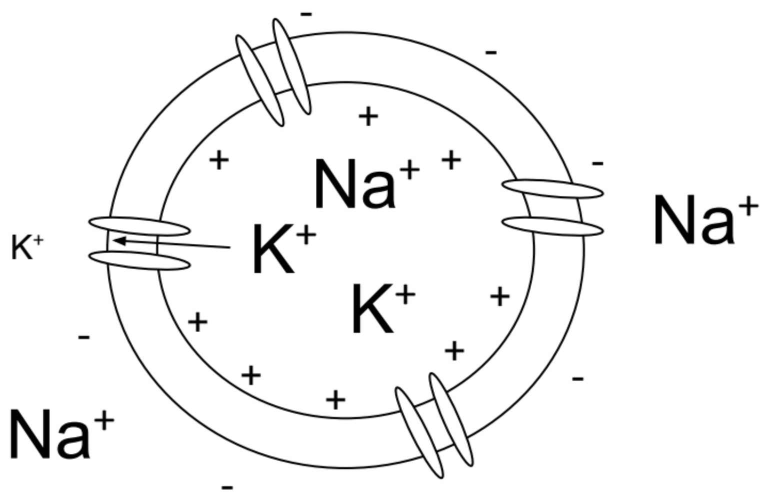


Figure 3.2: When the cell becomes depolarized sodium ions enter the cell. The charge within the cell becomes more positive.

and ending depolarization, we call this **repolarization**. When the voltage is below the resting potential, we call this undershoot **hyperpolarization**, which is when the membrane potential decreases towards a move negative value via potassium ions flowing out. As the voltage-gated potassium channels open and the voltage-gated sodium channels to close or become inactivated, there is now **negative feedback**. Negative feedback is a process by which an initial change is opposed by a force caused by the initial change. In this situation, the positive feedback causes a spike in membrane potential and the negative feedback stabilizes.

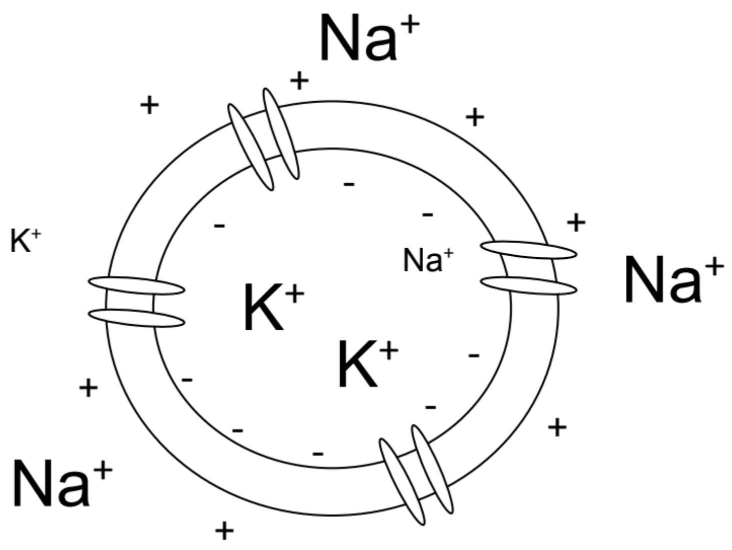
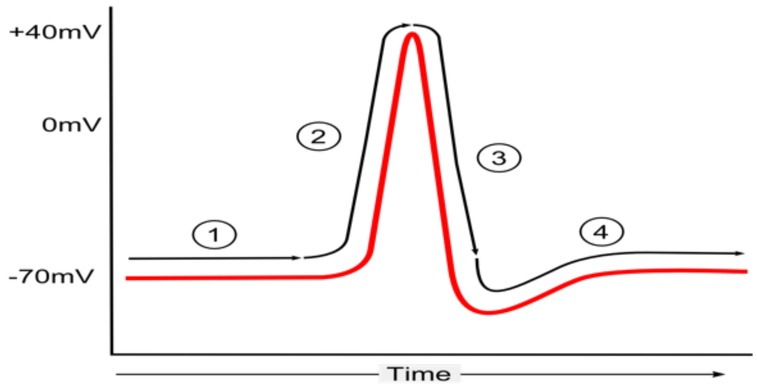


Figure 3.3: When the cell repolarizes potassium ions leave the cell. The charge within the cell go from positive to negative as it goes back to the resting state.



Exercise:

Exercise: Explain how depolarization and hyperpolarization relate to positive/negative feedback.

3.4 Nernst equilibrium potential

The electrical activity generated in a neuron is a result of ions flowing across the neuron's membrane which is caused by the following two principles: opposite

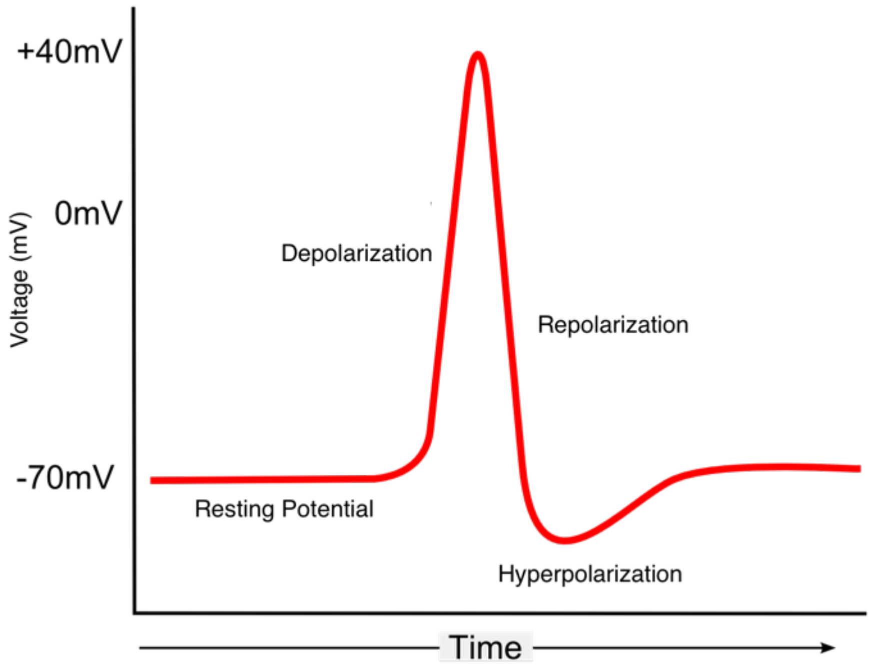


Figure 3.4: Example of an action potential.

charges attract, and concentration gradients seek to equalize. This potential difference is referred to as the **membrane potential**. In order for ions to flow, a concentration gradient must be established because the difference in concentration across the membrane leads it to pass either into the neuron or out of the neuron. This is accomplished by the **sodium-potassium pump**, which uses just below 10% of your body's daily energy to pump three sodium ions out of the neuron for every two potassium ions pumped in, thus forming two respective concentration gradients.

With the concentration gradient established, the sodium and potassium ions will flow down the concentration gradients when their respective channels open, generating an electrical current that propagates down the axon. We must also take into account the fact that each ion possesses a charge–or charges in the case of Ca++ and Mg++–and that as this charge is built up on one side of the cell, this will generate an electrical force that will begin to repel ions with similar charge as they try to flow down their concentration gradient. When the force of the concentration gradient matches the electrical force attracting or repelling the ion, this is known as the **Nernst potential** for that ion, also referred to as the **reversal potential**. This means that means that both sodium and potassium possess their own respective Nernst potentials. Nernst potentials are especially important because they allow us to calculate the membrane voltage when a particular ion is in equilibrium, which helps to define the role it plays in an action potential.

The Nernst potential for an ion can be derived from the following equation:

$$E_{ion} = \frac{RT}{zF} ln(\frac{[out]}{[in]})$$

Expression	Meaning
E_ion	Nernst Potential
R	Gas constant: 8.314 J/mol*K
ln()	Natural log
z	Valance
T	Temperature in Kelvin
F	Faraday's constant: 96485.336512 C/mol
[out]	Concentration of ions outside the membrane
[in]	Concentration of ions inside the membrane

While the Nernst potential will give the equilibrium point for a single ion, it also has a relation to the **equilibrium potential** or the resting potential the membrane, which is potential at which there is no net flow of ions, leading to a halt in the flow of electric current. The equilibrium potential is really a weighted average of all of the Nernst potentials and is modeled by the Goldman-Hodgkin-

Katz equation which is shown below:

$$V_m = \frac{RT}{F} ln(\frac{P_K[K+]_{out} + P_{Na}[Na+]_{out} + P_{Cl}[Cl-]_{in}}{P_K[K+]_{in} + P_{Na}[Na+]_{in} + P_{Cl}[Cl-]_{out}})$$

This equation utilizes the membrane permeability, P, in conjunction with the concentration of each ion inside and outside of the cell to produce the equilibrium potential of a membrane. Using this equation alongside the Nernst potential, the **driving force**, which is a representation of the pressure for an ion to move in or out of the cell, can be calculated using the following equation:

$$DF = V_m - E_{ion}$$

The Nernst Potential, the Goldman-Hodgkin-Katz equation, and the driving force present necessary calculations that allow for better understanding of the flow of ions in relation to an action potential.

Exercise: Why does depolarization not continue indefinitely once voltage-gated Na+ channels have opened?

Exercise: Briefly explain the concept of driving force.

Exercise: Explain the difference between the membrane potential, Nernst potential and equilibrium potential.

3.5 The Hodgkin and Huxley model

Alan Hodgkin (pictured left) and Andrew Huxley (pictured right) were two Cambridge University undergraduates who eventually found themselves working in a marine biology laboratory with the axon of a giant squid. The two men were able to derive the necessary information for their influential model of an action potential using the massive axon of the giant squid.

Hodgkin and Huxley developed a series of equations that could accurately predict and depict action potentials. Their work is a cornerstone for computational modeling as computer modelling can now be used to mimic the biological properties of a neuron that we are unable to directly observe.

Really the Hodgkin-Huxley Model is just an elaboration on the Integrate and Fire Model. The Integrate and Fire model was generated by French neuroscientist Louis Lapicque, who in 1907 sought to generate a mathematical model that could be used to predict and graph an action potential. In his efforts to understand action potentials, Lapicque chose to model the flow of ions as a single **leak current**.

Hodgkin and Huxley took the single conductance term from the Integrate and Fire Model is broken up into three separate conductance terms, each relating to a

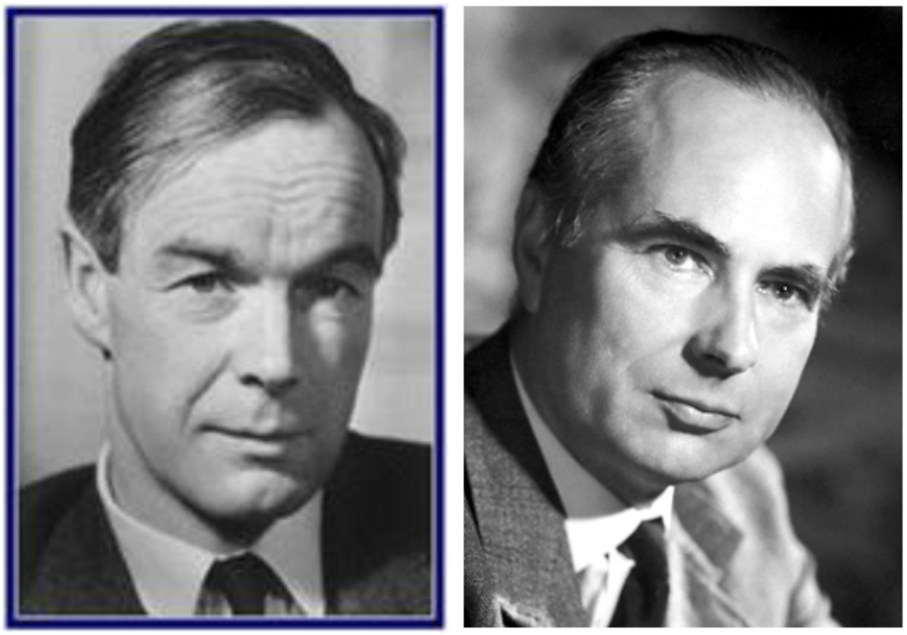


Figure 3.5: Alan Hodgkin (left) and Andrew Huxley (right).

different ion channel. These conductance terms are known as **gating variables** and are labeled *m*, *n*, and *h*. Voltage-gated sodium channel activation is modeled by the letter *ms*. Voltage-gated sodium channels have three subunits, as these three subunits are involved in the channels activation, *m* is raised to the third power. Voltage-gated sodium channel also inactivate at the peak of the action potential and this variable is modeled by the letter *h*. The combination of *m* and *h* gives rise to the conductance of Voltage-gated sodium channel which is modeled below:

$$\bar{g}_{Na} m^3 h(V(t) - E_{Na})$$

Voltage-gated potassium channels are modeled by the letter *n*. Voltage-gated potassium channels have four subunits, and thus the gating variable, *n*, is raised to the fourth power. The conductance of Voltage-gated potassium channels is modeled below:

$$\bar{g}_K n^4(V(t) - E_K$$

The final conductance taken into account by Hodgkin and Huxley is the leak potential of all the ions. The Leak conductance is taken into account for the instance when all ion channels are open. This conductance is represented below:

$$\bar{g}_L(V(t) - E_L)$$

These three conductance variables are combined together to form the Hodgkin-Huxley equation which is written as follows:

$$C\frac{dV}{dt} = I_e(t) - [(\bar{g}_{Na} m^3 h(V(t) - E_{Na})) + (\bar{g}_K n^4(V(t) - E_K)) + (\bar{g}_L(V(t) - E_L))]$$

Expression	Meaning
n	Activation of potassium channels
m	Activation of sodium channels
h	Inactivation of sodium channels
C	Capacitance
$I_{injected}$	Injected current
$\bar{g}_{Na}$	Maximum sodium conductance
$\bar{g}_{K}$	Maximum potassium conductance
$\bar{g}_{L}$	Maximum leak conductance
V	Membrane voltage
E_{Na}	Sodium Nernst potential
E_{K}	Potassium Nernst potential
E_{L}	Leak Nernst potential

Additionally, we can calculate the value of each gating variable over different voltages and times:

$$m\frac{dV}{dt} = \alpha_m(V)(1 - m) - \beta_m(V)m$$

Both n and h can be substituted for m in the above equation in order to calculate values for each gating variable. Additionally, note the α and β in the equation are rate constants that govern the opening and closing (respectively), of their channels. Here are their values:

$$\alpha_n(V_m) = \frac{0.01(V_m + 55)}{1 - exp(-0.1(V_m + 55))}$$

$$\alpha_m(V_m) = \frac{0.1(V_m + 40)}{1 - exp(-0.1(V_m + 40))}$$

$$\alpha_h(V_m) = 0.07exp(-0.05(V_m + 65))$$

$$\beta_n(V_m) = 0.125exp(-0.0125(V_m + 65))$$

$$\beta_m(V_m) = 4exp(-0.0556(V_m + 65))$$

$$\beta_h(V_m) = \frac{1}{1 + exp(-0.1(V_m + 35))}$$

Expression	Meaning
α_{n}	Rate constant for potassium channel activation (open)
α_{m}	Rate constant for sodium channel activation (open)
α_{h}	Rate constant for sodium channel inactivation (open)
β_{n}	Rate constant for potassium channel activation (close)
β_{m}	Rate constant for sodium channel activation (close)
β_{h}	Rate constant for sodium channel inactivation (close)

3.6 Summary

An action potential is the electro-chemical signal that propagates down a neuron. Action potentials are facilitated by the electrochemical gradient that is maintained through the action of the Sodium-Potassium Pump. The role that each ion plays within the action potential can be determined through the use of the Nernst Equation, which allows us to understand the movement of a specific ion at a specific membrane voltage. Understanding the role of ions, is important, but this is not how we actually graph an action potential. One of the first modules developed to graph an action potential was the integrate and fire module. This equation disregarded all of the biomechanical features of an action potential and focused on the subthreshold membrane dynamics of a neuron. This model was relatively effective, until Hodgkin and Huxley reassessed it and changed the single leak resistance in integrate and fire to three separate resistance terms. Hodgkin and Huxley in doing this created an equation that more thoroughly analyzed and depicted the very action potential that is displayed to this day.

##Exercises:## Here is a code that models the Hodgkin-Huxley model. Fill in the blanks and try to understand what each line of code does. Why does the code calculate α and β values first?

```
import numpy as np
import matplotlib.pyplot as plt

# Define model parameters
dt           = 0.1 # time step (ms)
tFinal       = 1000 # total time of run (ms)
tStimStart   = 250 # time to start injecting current (ms)
tStimEnd     = 750 # time to end injecting current (ms)
c            = 10 # capacitance per unit area (nF/mm^2)
gMaxL        = 3 # leak maximal conductance per unit area (mS/mm^2)
EL           = -54.387 # leak conductance reversal potential (mV)
gMaxK        = 360 # maximal K conductance per unit area (mS/mm^2)
EK           = -77 # K conductance reversal potential (mV)
gMaxNa       = 1200 # maximal Na conductance per unit area (mS/mm^2)
ENa          = 50 # Na conductance reversal potential (mV)

# set up data structures to hold relevant variable vectors
timeVec = np.zeros(0, tFinal, dt)
voltageVec = np.zeros(len(timeVec))
Ivector = np.zeros(len(timeVec))

# Fill in the BLANK!
mVec = np.zeros(     )
hVec = np.zeros(     )
nVec = np.zeros(     )

# assign the initial value of each variable
# For initial voltage, we set the resting potential to -65 mV
voltageVec[0] = -65
Ie = 200
mVec[0] = 0.0529
mVec[0] = 0.5961
nVec[0] = 0.3177

# For-loop to integrate equations into model
# Can you see why we are subtracting 1 here?
for i in range(    ):
  # Calculate alpha values for m, h, and n
  # These functions are fit from empirical data
  alpha_m = 0.1 * (voltageVec[i]+40)/(1-np.exp(-0.1*(voltageVec[i]+40)))
  alpha_h = 0.07 * np.exp(-0.05*(voltageVec[i]+65))
  alpha_n = 0.01 * (voltageVec[i]+55) / (1-np.exp(-0.1*(voltageVec[i]+55)))
```

```
# Calculate beta values
beta_m = 4*np.exp(-0.05556*(voltageVec[i]+65))
beta_h = 1 / (1+np.exp(-0.1*(voltageVec[i]+35)))
beta_n = 0.125 * np.exp(-0.01125*(voltageVec[i]+55))

# Calculate tau values for m, h, and n
tau_m = 1 / (alpha_m + beta_m)
tau_h = 1 / (alpha_h + beta_h)
tau_n = 1 / (alpha_n + beta_n)

# Calculate inf values for m, h, and n
infM = alpha_m / (alpha_m + beta_m)
infH = alpha_h / (alpha_h + beta_h)
infN = alpha_n / (alpha_n + beta_n)

# Calculate and store values in m, h, and n vectors
mVec[i+1] = infM + (mVec[i] - infM) * np.exp(-dt/tau_m)
hVec[i+1] = infH + (hVec[i] - infH) * np.exp(-dt/tau_h)
nVec[i+1] = infN + (nVec[i] - infN) * np.exp(-dt/tau_n)

# Calculate and store values in the tau voltage vector
tauVec = c/(gMaxK*nVec[i]**4+gMaxNa*mVec[i]**3 * hVec[i]+gMaxL)
vInf = (gMaxK*nVec[i]**4 * EK + gMaxNa*mVec[i]**3 * hVec[i]*ENa + gMaxL*EL + IVector[i])
/ (gMaxK*nVec[i]**4+gMaxNa*mVec[i]**3 * hVec[i]+gMaxL)
voltageVec[i+1] = vInf + (voltageVec[i]-vInf) * np.exp(-dt/tauVec[i])
```

Chapter 4

Reverse Correlation and Receptive Field Mapping

4.1 Vocabulary

- Poisson process
- Spike train
- Peri-stimulus time histogram
- Spike count rate
- Interspike interval
- Fano factor
- Coefficient of variation
- Spike-triggered average
- White noise
- Reverse correlation

4.2 Introduction

Throughout our everyday lives, we receive an enormous amount of sensory inputs from our surrounding environment: the color of the clouds before sunset,

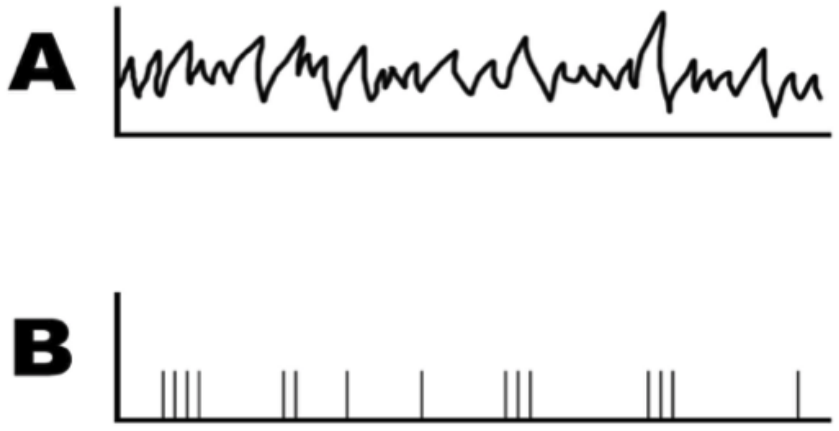


Figure 4.1: Example of a spike train. Graph A shows the recorded stimulus and graph B shows the recorded actions potentials during the stimulus.

the melody played by an old record player, the smell of apple pie, or the taste of your favorite dish. Our brain, with its incredible computational capacity, successfully encodes all these sensory inputs from different modalities to something we can perceive and understand, in the language of neurons. Our discussion from previous chapters noted that the language of neurons–or the neural code–consists of action potentials that are all-or-none events, and we learned how neurons fire an action potential. In this chapter, we are going to talk about why neuron fires and how to characterize and analyze these action potentials using spike trains. Based on this, we are going to discuss ways to study the relationship between outside stimuli and neural responses.

4.3 Spike Trains

Assume that we measured a neuron firing in response to a sensory stimulus, and we recorded its voltage changes and displayed the signal in an oscilloscope. How should we analyze the information encoded in these action potentials? As we mentioned before, the action potential is an all-or-none event. This binary characteristic gives us a way to simplify the complicated voltage response curve: for every time point in our measurement, if there is a spike firing, denote its value as 1; if not, denote it as 0. After the recording, we get a sequence of 0s and 1s in a time-dependent order. We commonly refer to this sequence as a spike train, shown in Figure 1, part B.

Since all action potentials fire to the same voltage level, there is no difference in their intensities. Thus, in order to have action potentials that convey meaningful information, neurons can only vary on timing of firing, including varying firing rates or varying the time intervals between each firing. Although this seems super-simplified, our spike train contains mostly the information we need to analyze if we want to know what causes the original neuron to fire. In order to systematically analyze these data in spike trains, we first need to define some statistics.

4.4 Spike Statistics

Now we have a sequence of 0s and 1s which represent neural firing, the next step is to calculate the **spike count rate**, which is the number of spikes divided over a given time interval. This parameter directly shows the firing rate, but it cannot reflect variation. Assume that we have two neurons that have the same spike count rate. One is a regular-firing neuron and the spikes are evenly distributed along the time axis, while the other is a bursting neuron that fires sets of spikes with longer intervals between individual sets. How can we distinguish between these two spike trains? Here, we want to introduce another parameter called the **interspike interval (ISI)** or in other words, the time interval between every pair of spikes. In Figure 2, there is a histogram that shows the distribution of ISIs from an artificial spike train. The ISI histogram can be characterized by **coefficient of variation (CV)**, which is the standard deviation of ISIs divided by the mean of it. Apart from that, we can also use the **Fano factor** to measure the spike variability. It is calculated by the variance of the number of spikes divided by the mean number of spikes in a given time interval. Compare to CV, Fano factor is less dependent on the intervals between spikes but more on the number of spikes in a given time bin. If the underlying firing rate varies or the spike firing in irregular time points, both CV and Fano factor increase. Thus, CV and Fano factors are useful secondary statistics that helps to measure variability in spike trains.

We can do a lot with a single spike train. However, in vivo, neural responses are highly variable and the response of a neuron to the same stimulus may even vary from trial to trial. Effectively, to account for the cross-trial variability, we need to analyze results from multiple trials to get a better estimate of the average neural response. A **peristimulus time histogram (PSTH)** can be generated by averaging across trials. The most direct method is to put trials in small time windows that correspond to each time point and calculate the spike count rate in window for each time point. By implementing this process, each time point is assigned an average rate, and by plotting we get a continuous frequency curve. One example is shown in the Exercise 1, in the "spike density" section, in which spikes are averaged in time windows, and then averaged across trials. In this case, all trials are run on the same neuron, which reduces the cross-trial variability and increases the time resolution. The response, however,

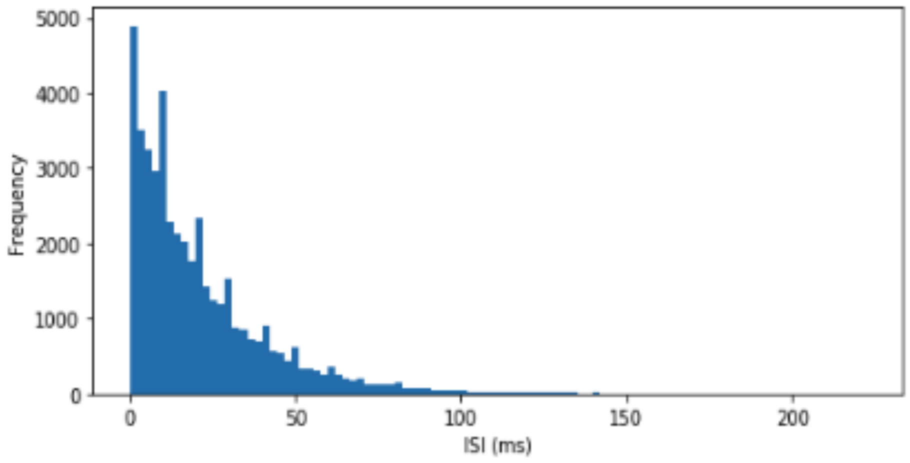
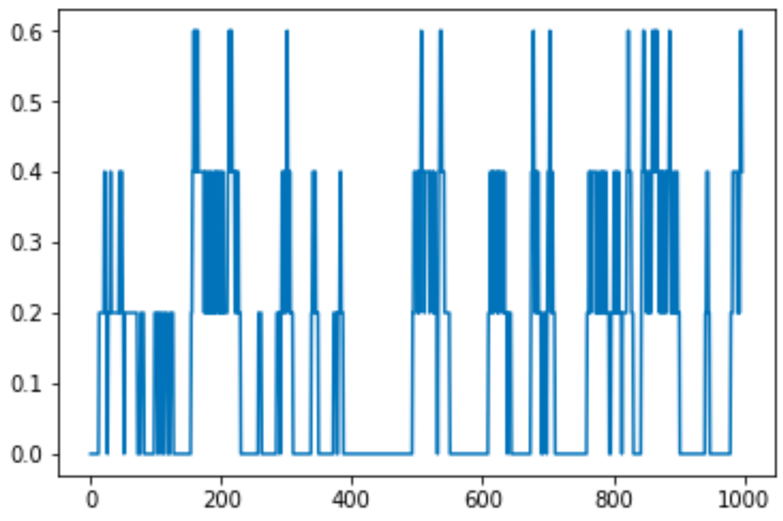


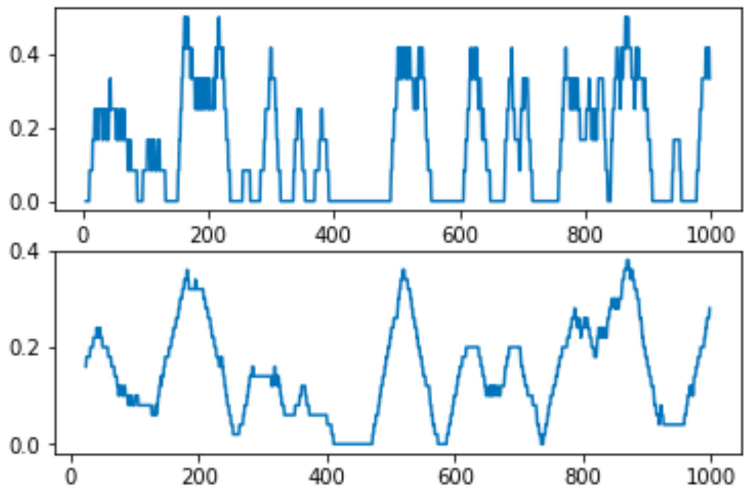
Figure 4.2: Distribution of ISIs from a randomly firing artificial neuron.

may be affected by adaptation. Furthermore, we can also assess the firing pattern of a population of neurons by calculating the average spike counting rate for all the time points, and then averaging across trials. At this point, different trials are run on different neurons, and the generated response curve accounts for the response pattern of a population of neurons in the area that was tested. This method, while it generates better time resolution, omits the possible variability across neurons in the population. Choosing different time windows influences the characteristics of the frequency curve, as shown in Figure 3. In general, PSTH is essential because it transforms a discontinuous spike train into continuous response curves, which allows us to calculate the correlation between stimulus and response. This will be further elaborated upon in the later sections.

Exercise 1: What are the pros and cons for each type of histogram?

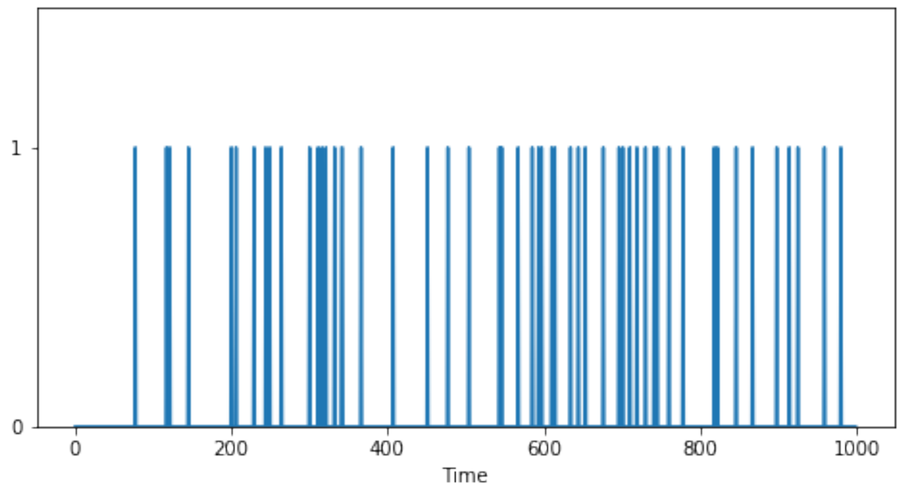
Name	Algorithm	Input
Spike Count Rate	Calculate the number of spikes per time interval	![](~/Dropbox/teaching/comp
Spike Density	Use a single neuron to compute different trials	![](~/Dropbox/teaching/comp
Population Density	Use different neurons during a single trial	![](~/Dropbox/teaching/comp

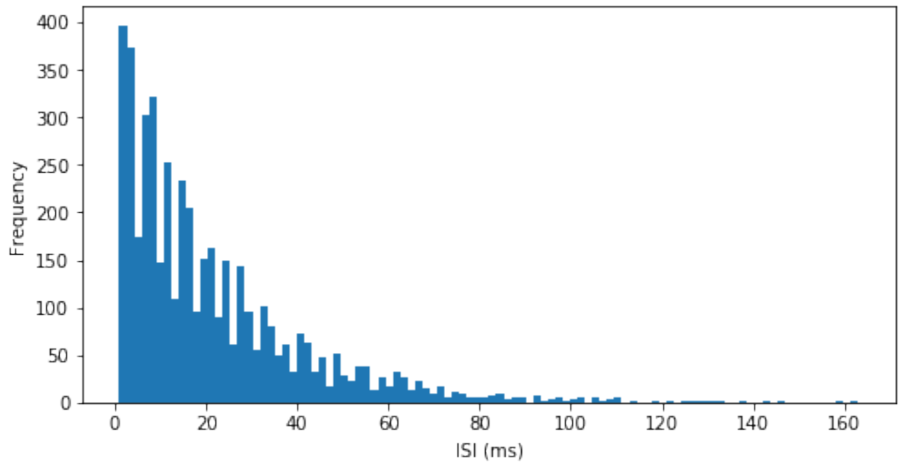




Sometimes, instead of collecting data from real neurons, we need to simulate spike trains from given statistics. The artificial spike trains can be compared with real data, or used to reconstruct possible firing patterns with a given stimulus. Here, we will discuss the homogeneous Poisson process, or the simplest way of generating artificial spike train. The homogeneous **Poisson process** entails that for every small interval on the timeline, the probability of an event happening (in our case, action potential) will be proportional to the length of the time interval, while the proportionality constant r is fixed. To understand this abstract definition, think about a timeline whereby at each time point, we throw a coin and record the head as 1. Try to visualize that timeline. We agree that all 1s will be randomly spread along the timeline, and the interval between when we get a pair of heads, varies along the timeline. This will look very similar to an artificial spike train generated by the Poisson process, whose distribution can be expressed by the following formula:

$$p(q\ spikes\ in\ \Delta t) = e^{-\lambda}\frac{\lambda^n}{n!}$$





Exercise 2: Explain the qualifications for a poisson spike train?

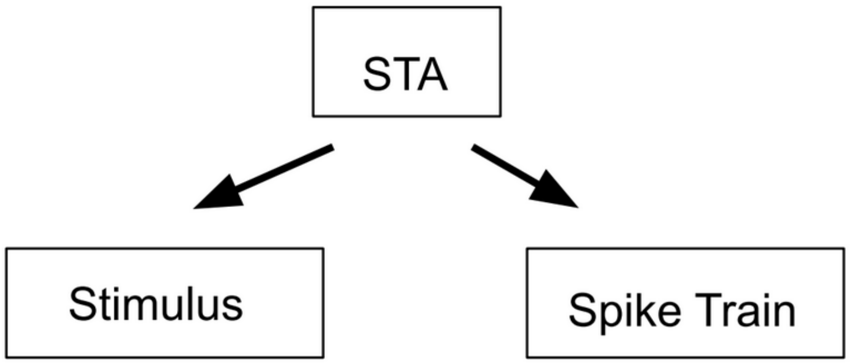


Figure 4.3: The spike triggered average can be used to calculate both the stimulus and the spike train.

4.5 Spike-triggered Average

An essential tool for describing neurons, and how they respond to certain stimuli, is the **spike-triggered average (STA)**. The STA is the average value of the stimulus during some time interval before a spike occurs. Researchers record a neuron's activity as it responds to various stimuli. First, the researchers must determine the amount of time before a spike they want to analyze. Once the data has been obtained and the time step determined, the value of the one-time step before a spike is recorded, and averaged across trials. This value ultimately characterizes the level of stimulus necessary for the neuron to fire. It is important to note that the spike-triggered average is measuring the average level of the stimulus, not of the neuron. This calculation is based on the probability of a neuron spiking due to stimuli activity to occur in the recent past.

The spike-triggered average can be utilized to determine the receptive fields of individual neurons. However, when using the STA to determine receptive fields, the stimulus presented to the recording neuron must be sufficiently random. If it isn't, any correlation in stimuli will be presented in the generated receptive field, thus skewing the result. A **white noise**, is a type of stimulus with random variation where the value at each time point is independent of all other points. White noise can be employed in these instances to provide a receptive field without bias. White noise stimuli can look different based on the neuron and the system being observed, from a series of random auditory frequencies to randomly generated pixels stimulating the visual cortex. For each set of stimuli, the value at each time point does not correlate with the values around it. For more

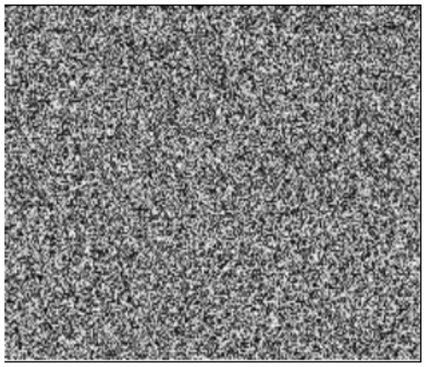
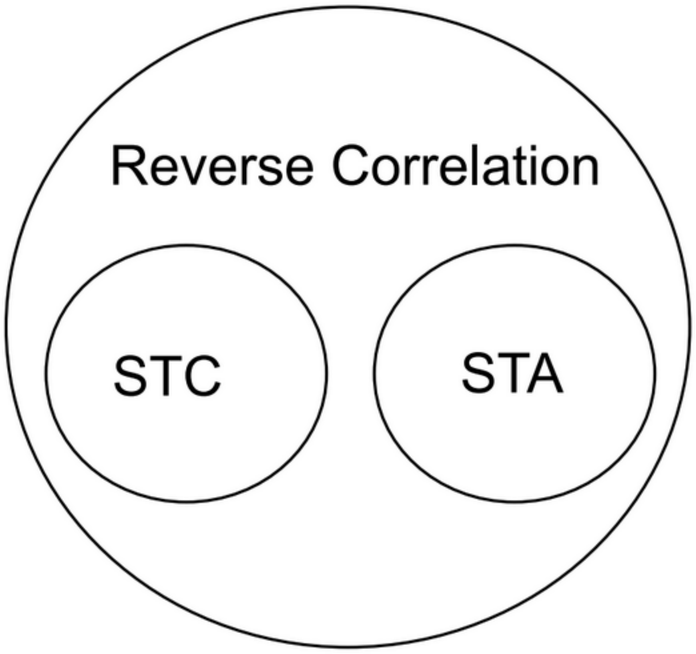


Figure 4.4: Example of a white noise stimulus using gray scale.

information on spike-triggered average analyses, especially concerning receptive fields,read this experiment.

4.6 Reverse Correlation



When analyzing neurons and neuronal responses, there are two main factors: the inputs (controlled by researchers), and the outputs (measured by researchers). The process of **reverse correlation** implements the analysis of outputs to determine the inputs that the neuron will respond to with a spike. The spike-triggered average is an application of this process as it looks back at the stimuli preceding a spike to determine information about the sensitivity and response of the neuron. In addition to spike-triggered average, a calculation known as spike-triggered covariance can be used to analyze neuronal responses. Spike-triggered covariance (STC) can be used to identify multi-dimensional inputs to a neuron and is especially useful in linear-nonlinear Poisson models that will be discussed later in this section. STC uses the covariance, variability between two factors, of stimuli that trigger spikes in a neuron to determine a neuron's response characteristics to multi-dimensional stimuli.

The basic model for reverse correlation is a Linear Single Input Single Output system (LSISOS). Such linear systems assume the two principles of homogeneity. First, the neuron will not undergo processes such as habituation and will always respond in the same way to the stimulus. Second, superposition: the response from multiple stimuli will be equal to the sum of the individual stimuli. In this model, the LSISOS response is the sum of the spikes scaled to time. Similar to spike-triggered averages, it is best to input a white noise stimulus and then cross-correlate it with its output, which will give you the spike-triggered average. In cross-correlation, the higher the similarity between the two values (the stimulus value and the output value), the greater the correlation value (the spike-triggered average).

The LSISOS model assumes a linear activity of neurons that is not entirely accurate due to neuron characteristics such as a spike threshold and a refractory period. A new model, known as the linear-nonlinear-Poisson model takes these factors into account. There are three stages to this model: linear stage, nonlinear stage, and the Poisson spike generator stage. The linear stage considers how a neuron responds to a specific feature in a spatio-temporal linear sense. The second stage takes the linear output and input through a nonlinear function to give a neuron's instantaneous spike rate. The nonlinear function can either be a logistic curve or a rectified linear (ReLU) function. The final step translates the output of the initial steps into spikes using an inhomogeneous Poisson process. The final result from the Poisson generator demonstrates the areas of the stimulus where a spike is more likely to occur.
Reverse correlation is a technique used for understanding what neurons are responding to, and the spike-triggered averages discussed earlier are one example of how reverse correlation is implemented.

4.7 Summary

Reverse correlation and all the concepts that play a role in this widely implemented technique, from different modes of spike statistics to various types of

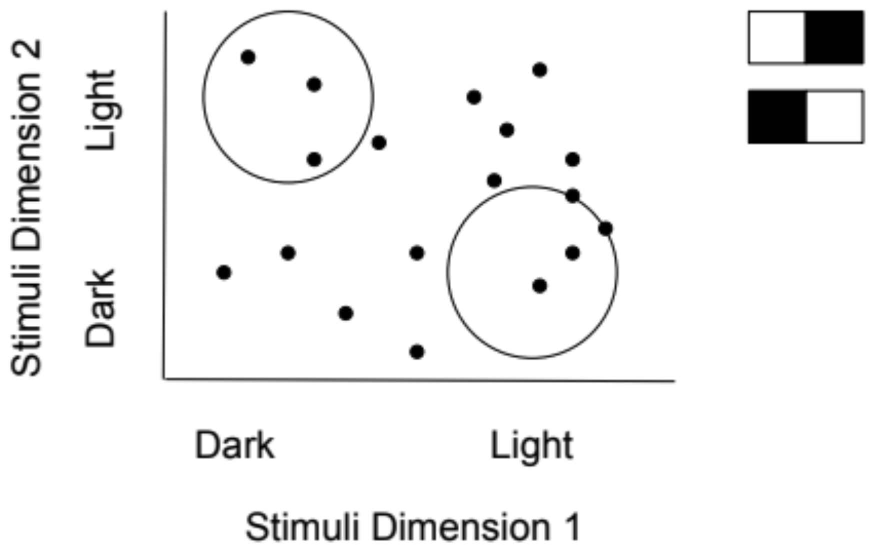


Figure 4.5: Spike-triggered covariance shows how two different stimuli dimensions can be calculated together.

stimuli, can be an intimidating topic in computational neuroscience. Analyzing the relationship between inputs and outputs to understand the effect each has on the activity of a neuron is the root of this topic. These analyses can move in both directions: input to output, or output to input. One can manipulate the stimulus and measure the resulting spike train through various statistical methods, or one could use reverse correlation by utilizing the known output of a spike to look back and understand the input necessary to create such a response. These analyses are working on grasping what stimuli the neuron does, or does not, “like”. In other words, they help us predict the neuron’s responses.

Exercise 3 (*Challenge!*) Match each concept to the Python function.

Concepts:

- Fano Factor
- Spike Count Rate
- Interspike Interval

```python
# Function 1
prob = 45/1000
spikeMatrix = np.zeros((1000,1000))
for i in range(1000):
  for j in range(1000):
    if np.random.rand() < prob:
      spikeMatrix[i,j] = 1
spikeCountRates = np.sum(spikeMatrix, 1)
plt.figure(figsize=(8,8))
plt.subplot(2,1,1)
plt.hist(spikeCountRates, 40, edgecolor='black')
plt.xlabel("Average firing rate (Hz)")
plt.ylabel("Frequency")
```

```python
# Function 2
np.var(spikeCountRates) / np.mean(spikeCountRates)
```

```python
# Function 3
spikeCountRates = np.sum(spikeMatrix, 1)
totalSpikes = int(np.sum(spikeMatrix, axis=None))
isi = np.zeros(totalSpikes - 1)
count = -1
for i in range(1000):
  spikes = np.nonzero(spikeMatrix[i,:])[0]
  for j in range(len(spikes)-1):
    count += 1
    isi[count] = spikes[j+1] - spikes[j]
plt.subplot(2,1,2)
plt.hist(isi, 100)
plt.xlabel("ISI (ms)")
plt.ylabel("Frequency")
```

Chapter 5

Neural Networks

5.1 Vocabulary List:

- Neural network

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning
- McCulloch-Pitt (MCP) neuron
- Perceptron
- Step function
- Linearly separable
- Activation function
- Sigmoid activation function
- Hidden layers
- Back propagation
- Cost

5.2 Introduction/Background

Imagine for a moment that you wake up, groggy and not looking forward to your commute, when you ecstatically remember that you don't have just any car, you have a self-driving car. You slide into the driver's seat of your Tesla and faintly pay attention as your car does the heavy lifting and drives you to work. When one considers the concept of a neural network, a biological definition may first come to mind in the sense of the neuronal connections in the brain; however, this chapter delves into how we can recreate the learning apparent in our biology through computational models. Although this may sound like a slightly intimidating goal, neural networks have become a commonly used method. They are found in a wide variety of technologies from Tesla's self-driving cars to Go playing robots. Overall, the goal of a neural network is to identify existing patterns in stimuli or inputs and produce an output that would mirror the output of our own brain through a set of determined algorithms. This allows us to create complex neural networks that can allow algorithms with the ability to learn. In this chapter, we aren't going to delve into the deep complexities of neural networks required to fully understand how a self-driving car works, but we will outline the basics of how machines learn through neural networks.

Neural networks identify existing patterns in stimuli. This means that based on a series of inputs, a neural network identifies whether or not the input conforms to a specific group or definition. In other words: a computer learns to perform a particular task by analyzing sets of examples. Take for example a technology that recognizes whether or not there is a face in a photograph. The neural network may ask if a stimulus has eyes, a nose, and a mouth. If the answer is yes, it will recognize the input as a face. If the stimulus lacks these features, the model will give an output to convey that there is no face. The output in this situation is the binary answer of whether or not a face exists. The more questions asked and the more layers in the neural network, the more complex stimuli and patterns we can look for. The initial example given here is a highly simplified idea of a neural network. In this chapter, we will start with the most simple building block of a neural network and build up to a more complex network.

5.3 Different Types of Learning

Before defining a **neural network**, first, let's take a moment to consider the concept of learning. Learning is something that can be defined in a variety of ways. One definition is the acquisition or modification of knowledge, behavior, skills, values, or preferences. What does this mean in the context of deep learning and neural networks? It may be difficult to use just one definition of learning to understand neural networks, so instead let's consider three different types of learning: **supervised**, **unsupervised**, and **reinforcement learning**. Supervised learning is where a teacher provides input and the expected outputs to a student for the student to better predict future problems. This is a system of learning which you may be familiar with, one example is when a teacher gives you both the problem and the answer for you to be able to do future problems. Another example is computer vision learning. Giving a computer examples of different visual stimuli, such as handwriting, it can learn to distinguish between different letters using a neural network. Unsupervised learning is learning that occurs in the absence of a teacher. A student simply looks at patterns and tries to maximize correlations or find a basic understanding. **Hebbian learning** is an example of unsupervised learning. Finally, reinforcement learning is the shaping of behavior through reward and punishment. It is learning shaped through interactions with the environment. An example of this would be the robot AlphaGo which was taught how to beat humans at the game of Go. The neural networks we will be discussing in this chapter primarily use supervised or unsupervised deep learning, but if you are interested in reinforcement learning, this video on AlphaGo is a great resource. As you continue to read through this chapter, keep the goal of neural networks in mind as well as the various types of learning which can be used to achieve this goal.

Exercise 1: Briefly describe the different kinds of learning. Can you provide a real-world example for each? (one not mentioned in the reading.)

Exercise 2: Are certain kinds of learning more capable of tackling complex issues, why or why not? What kind of questions can be addressed by individual kinds of learning?

5.4 McCulloch-Pitt (MCP) Neurons

MCP neurons were some of the first examples of artificial neurons that can be used to build networks. MCP neurons are named after Warren McCullough and Walter Pitts, who together proposed the model in 1943. Pitts self-taught logic and mathematics. He eventually ended up doing research at the University of Chicago, despite adverse conditions growing up. When he met Warren McCullough, a professor at the university, McCullough suggested that Pitts come to live with him and the two began a research partnership through which they produced their concept of the MCP neuron. The MCP neuron is a simple analog of its biological counterpart. The neuron receives one or multiple inputs which are then summed up to produce an output. These summed inputs essentially tell the neuron whether or not to fire.

It is, however, slightly more complicated than a yes or no question as to whether the neuron fires. Each input is multiplied by an assigned weight. These resulting values are then added up. The model then compares the actual summation to an already existing threshold value. If the sum of the various inputs multiplied by the weights is greater than the threshold, the neuron is considered to be firing. If the sum is less than the threshold, the neuron does not fire. The equation is shown below:

$$Output = 1\ if\ \sum x_i w_i < threshold$$

$$Output = 0\ if\ \sum x_i w_i \geq threshold$$

Exercise 3: How does an MCP neuron work? How similar is it to a real-life neuron?

Exercise 4: Before we delve into further details about neural networks, what do you think could be some of the potential limitations of MCP neurons?

MCP neurons function as effective and simple building blocks but they do have certain limitations. Let's consider a neural network designed to recognize a human face again. MCP neurons can ask certain types of questions to answer the question of whether or not something has a human face, such as:

- Does this stimulus have eyes and a mouth?
- Does this stimulus have eyes or a mouth?
- Does this stimulus not have fur covering the entirety of its skin?

These AND, OR, and NOT questions can be answered in a binary (yes or no) manner and thus can be modeled by an MCP neuron. MCP neurons cannot, however, answer what are called exclusionary, also known as XOR questions. Let's consider the example of a neural network that suggests movies. You have two hours to watch a movie and cannot decide between a romance or horror film. You can't watch both due to your time constraints so you need a neural network that will suggest either a romance movie or a horror movie but not both movies. This is an example of a situation in which asking an XOR question is necessary. In this case the question specifically is:

Romance	Horror	Possible.Picks
1	1	0
1	0	1
0	1	1
0	0	0
.	.	.

The MCP neuron lacks the ability to ask an XOR question due to the nonlinear nature of its question. Earlier it was mentioned in this chapter that MCP neurons are based upon binary inputs and outputs. In the case of the XOR question, the complexity of the input-output relationships, due to the nonlinearity apparent in the question, prevents the neural networks from being able to produce an answer. Despite this drawback, there are ways to produce more complex neural networks which we will elaborate on later in this chapter.

Coding exercise: Fill in the input and output for an AND gate in the following code, implement it in a Jupyter Notebook.

```
# Import useful packages
import numpy as np
import matplotlib.pyplot as plt

# Define Output and Input
```

```
X1 = np.array([ [ , ], [ , ],[ , ],[ , ] ])
Y1 = np.array([ [], [], [], [] ])
```

5.5 Perceptron

Before we dive into neural networks and discuss how they work and what they do, we will introduce the concept of a **perceptron** and discuss its relevance. A perceptron is an algorithm for performing binary classification based on a step function. A trivial perceptron functions as follows:

1. Consider a set of inputs and a corresponding set of weights:

$$Output = f(x) = \begin{cases} 1, & \text{if } \sum w_i x_i + b \geq 0 \\ 0, & \text{if } \sum w_i x_i + b < 0 \end{cases}$$

2. Take the dot product of these two sets.

3. Add another predetermined number called bias.

4. 4. If the result is greater than or equal to 0, the output is 1; otherwise, the output is 0. This is the step function:

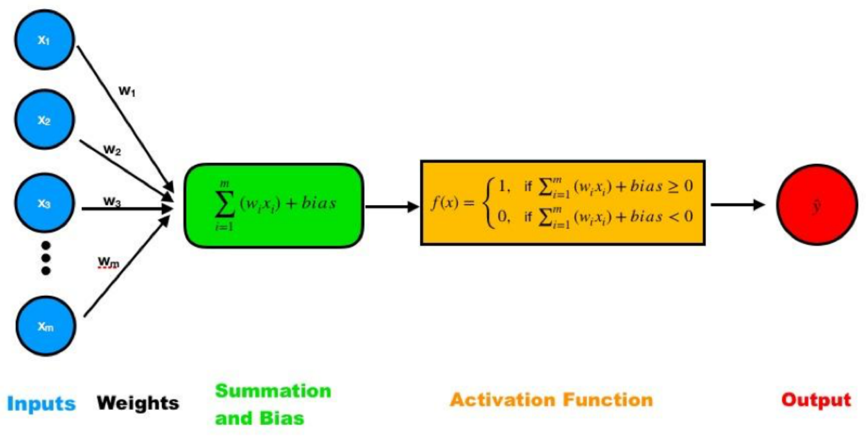
$$Output = f(x) = \begin{cases} 1, & \text{if } \sum w_i x_i \geq threshold \\ 0, & \text{if } \sum w_i x_i < threshold \end{cases}$$

Note that in this **step function** , we compare the result of part 3 to 0. Alternatively, we can subtract b from both sides so that -b is a threshold value to compare with instead of adding bias and comparing to 0.

Code Exercise (continued): Assume that bias equals to -1. Define the initial weights randomly by sampling from a uniform distribution between -1 and 1. Use the code below to get started.

```
# Initialize parameters
# eta is the learning rate for your model
eta = 0.01
output = np.zeros(4)
bias =
weights =

# Initialize data structure to hold the accuracy of your model's prediction
accuracy = np.zeros(500)
```



x_1
x_2
x_3
x_m
w_1
w_2
w_3
w_m
$\sum_{i=1}^{m}(w_i x_i) + bias$
$f(x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{m}(w_i x_i) + bias \geq 0 \\ 0, & \text{if } \sum_{i=1}^{m}(w_i x_i) + bias < 0 \end{cases}$
$\hat{y}$
Inputs
Weights
Summation and Bias
Activation Function
Output

Figure 5.1: Scheme of a single-layer perceptron. Inputs are multiplied with their corresponding weight and the products are summed up plus the bias. The result is then entered in the activation function, which generates the output.

Follow the summation and bias function above in Figure 1, fill in the blanks inside the for loop:

```
# Calculate the output for the summation and bias function
output =           + np.matmul(   ,   )
# Input the output in the activation function,
# the result will be the prediction of this model based on the
# given inputs.
output =

# Denote the prediction as yHat
yHat =

yHat = np.sign(output)/2 +0.5
```

We just described what is known as a single-layer perceptron. This type of perceptron produces outputs that are **linearly separable**. This means that there exists some line that can be drawn between our two output sets. Note that this is only the case if there are two input dimensions. With three input dimensions, the data “live” in a 3D scatterplot, and it requires a plane to separate the classes. In more than three dimensions, a hyperplane is necessary. Single-layer perceptrons can solve simple problems like representing logic operators AND, OR, and NOT. However, these single-layer perceptrons are incapable of solving

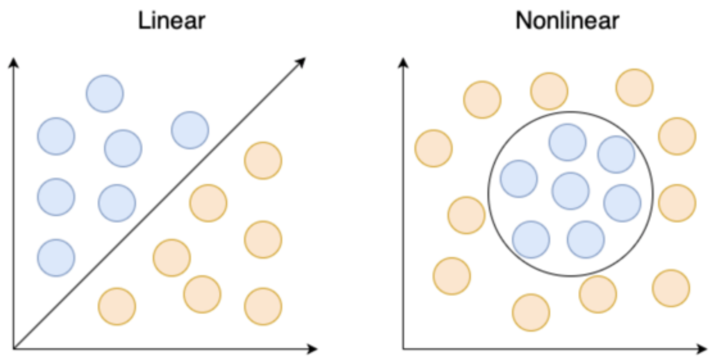


Figure 5.2: A visual demonstrating the distinction between linearly and nonlinearly separable problems. The left image shows a linearly separable problem that can be solved with a single line. The right image shows a nonlinear problem that cannot be solved with a single line.

a more complex problem such as XOR because the outputs cannot be produced from a linear combination of inputs (the outputs are not linearly separable).

Exercise 5: What is the difference between a single layer perceptron and an MCP neuron?

Exercise 6: Explain how a perceptron functions. Which step contributes most to the "learning" process?

One of the major shortcomings of the single-layer perceptron is its **activation function**. We already mentioned the activation function of the single-layered perceptron as being a step function:

The main fault of the step function, however, is that it cannot represent small changes in the weights to reduce error and approach the optimal solution. This is because the activation function is not differentiable. Therefore, if there is an error in the output, changes made to the weights are constant and are not dependent on the change in input.

Now consider a multilayer perceptron with a differentiable nonlinear activation function. If the activation function is differentiable, then you can make gradual changes to the weights and bias so as not to overshoot the optimal solution. Consider an alternative activation function:

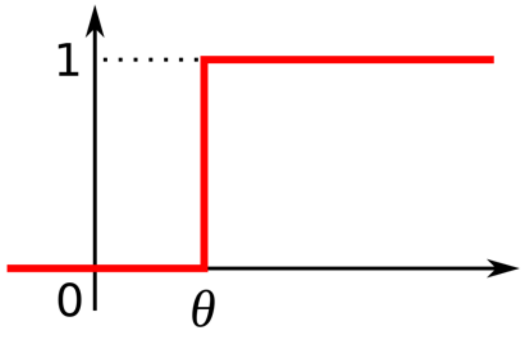
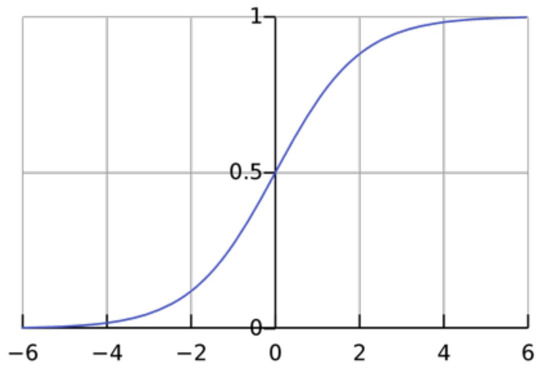


Figure 5.3: Graph of a step function that can only oscillate between 0 and 1. The function begins at 0 and then at some time theta directly rises to 1.



We call this the **sigmoid activation function**. While the step function has an output of either 0 or 1, the output of the sigmoid function is continuous between 0 and 1. There also exists some threshold for decision making in both functions. Another distinction between sigmoid and the step function is that sigmoid is a smooth curve that is differentiable everywhere. This allows us to be able to make slight adjustments to our weights. Thus, the process of tuning weights and bias is gradual and leads to better learning in these networks.

Earlier we explained a single-layer perceptron as having a step function, which is just one of many possible activation functions. When our output is not linearly separable, which is the case in most real-world problems, we chain layers of neurons together and use multiple nonlinearities from the various units to solve the problem–these added layers are known as **hidden layers** . Each of these hidden layers–as well as the output layer–will have its own activation function, and will make a decision based on some input (neural feature). The output of this function is mapped between 0 and 1 where 0 means the feature is not present and 1 means it is present, given a differentiable activation function. Non-linearity

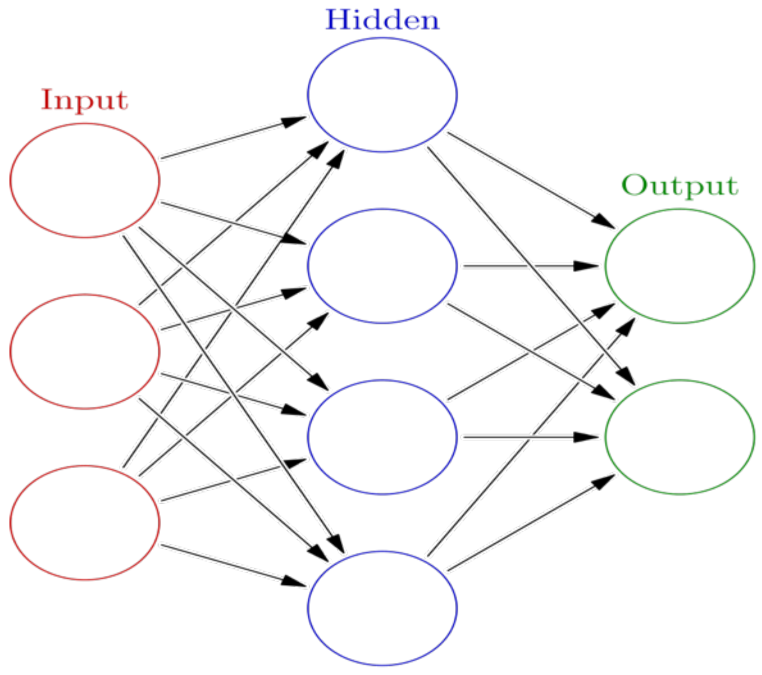


Figure 5.4: Example of a multilayer network. (Glosser.ca, 2013)

is required as we are attempting to produce a non-linear decision boundary between two sets (see **??**). Furthermore, if we only use linear activation functions, we can add any number of additional layers to our network and the final output is simply a linear combination of the initial input data. In other words, if we had no activation functions and we were to merely pass the weights from layer to layer, the output would be a linear combination of the inputs.

Exercise 7: Why is it that the activation function produces linear data? Why is linear data a problem with respect to real-world situations?

In 1986, Hinton et al published "Learning representations by back-propagating errors." This paper introduced two concepts that allow for these non-linear activation functions to learn more complicated features. The first being the addition of hidden layers, as mentioned earlier, to the perceptron. These are nodes representing neurons in the network between the input and output. It is these hidden layers explicitly that allow for multilayer perceptrons i.e. neural networks, to learn more complicated features (like XOR). The second of these concepts is **backpropagation**, a procedure to repeatedly adjust the weights to decrease the difference between the network output and the desired output.

Backpropagation is performed by calculating the **cost** of an output neuron. One example of a cost function is Mean Squared Error, which allows us to calculate cost via:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Yhat_i)^2$$

In the cost function, Y_i is the output of our activation function and $\hat{Y}_i$ is our desired output. We take the output of our cost function and use it to make changes in our weights with the goal of minimizing cost for the next iteration through our network. Note that in order for backpropagation to be possible, our activation function must be differentiable between 0 and 1 e.g. the sigmoid function, but the step function is not. For a detailed explanation of how backpropagation works, check out this video.

Exercise 8: Explain why backpropagation and hidden layers are necessary for the perceptron to optimize its learning capabilities.

Exercise 9: Why is the sigmoid function a better alternative than the activation function for multilayered perceptrons?

Exercise 10: Think back to real neurons. How does a multilayered perceptron compare to an actual neural network? What strengths does it have, and what weaknesses?

Code exercise (continued):
Below is the completed for loop following from the previous code snippet. Note the manner in which we updated the weights.

```
    # Calculate the error between predicted y and your expected Y1
    error = (Y1 - yHat)

    # Compute delta weights. This step is where the
    # model starts to 'learn' based on the difference
    # between expected output and the actual output.
    # Delta weight will be the change of weight. It is
    # based on input value and error size.

    deltaW = eta * np.matmul(X1.T, error)

    # Update weights
    weights = weights + deltaW

    # Store accuracy for the prediction along iterations
    accuracy[i] = len(error[error==0]) / 4

# Plotting
plt.figure()
plt.plot(accuracy)
plt.xlabel('iterations')
plt.ylabel('percent accuracy')
```

5.6 Summary

Neural Networks have come quite far in the last half-century. What was initially thought to exist exclusively in the brain can now be replicated in an algorithm. It was not until the creation of the McCulloch-Pitts neuron, followed by the extension into the perceptron before people were convinced that a machine could learn. People initially believed that the perceptron is useless as they could not represent logic as simple as XOR with their single layer. It was not until Geoffery Hinton pioneered the creation of the multilayer perceptron that artificial neural networks were respected as a strong representation of how a computer can learn. Hinton and his collaborators are responsible for many of the significant leaps in programming computers to learn. The introduction of backpropagation and hidden layers to neural networks allowed for substantial progress to be made in the field of Artificial Intelligence.
Neural networks are perhaps the most hyped topic in the tech world today. As we journey further into the 21st century, we are constantly bombarded with headlines and news feeds of self-driving cars, agents teaching themselves to play hide and seek, and the greatest chess engine the world has seen that taught itself to play in a few hours. There is no doubt that it is an exciting time to be a neuroscientist; we are at the forefront of considerable innovation and it all emerges from the material discussed in this chapter.

Coding Challenge Following the code given in this chapter, try to make a model for an XOR gate. Does the accuracy change when you do more iterations? Explain your result. Hint: consider the term “linearly separable”.

Chapter 6

Decoding

6.1 Vocabulary

- Imaging techniques
  - EEG
  - MEG
  - fMRI
  - ECOG
- Multivariate Pattern Analysis (MVPA)
- Decoding
- Classifier
  - Correlation classifier
  - Distance-based classifier
  - Boundary-based classifier
- Curse of dimensionality
- Linear discriminant analysis
- Linear support vector machine (LSVM)
- Cross-validation
- Rank measure

6.2 Introduction

In our previous chapter on reverse correlation, we discussed how we may utilize spike trains to look back and understand the input necessary to create such a response in a neuron. We discussed the ability to not only correlate an output with a specific stimulus but additionally the capability to decode raw brain data

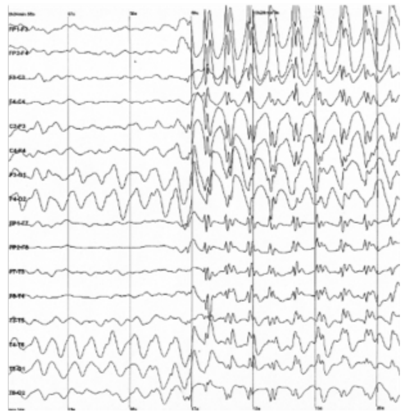


Figure 6.1: Raw output of EEG recording

measured via EEG, MEG, fMRI, and ECOG. In other words, for some given neuronal signals, what stimulus was provided to that neuron which caused it to fire? In this chapter, we will introduce several methods of neural decoding. In particular, we will detail the specifics of several classifiers pertinent to decoding, as well as their various benefits and constraints.

6.3 Imaging Techniques

For decoding problems, there are numerous different types of data that could be analyzed. The simplest recording of a single neuron will give a spike train that contains 0s and 1s, or we can average the rate across time bins to generate a continuous curve to study. Although it might be easier to just focus on one neuron, usually data are drawn from a population of neurons. In some cases, a pseudo population of data is constructed from a single-neuron recording in order to reduce noise or assess group activity. As a type of supervised learning, in decoding problems, both the input and output are given, but this input can vary in its appearance. There are multiple types of imaging techniques commonly employed by researchers to obtain data and recordings from participants. We will discuss four of the main techniques here: EEG, MEG, fMRI, and ECOG.

6.3.1 EEG

Electroencephalography (EEG) is a method by which the electrical signals produced by action potentials across a large population of neurons are recorded to distinguish areas of activation in the brain. In an EEG setup, electrodes are placed around the scalp in a non-invasive manner to record voltage fluctuations.

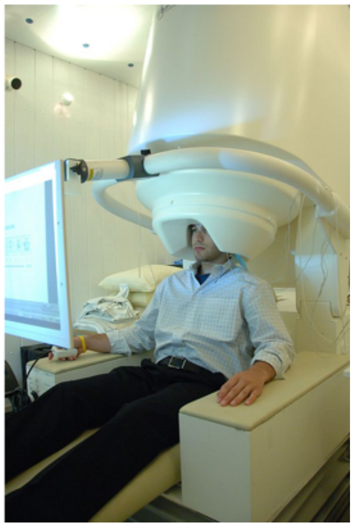


Figure 6.2: Image of a participant in a MEG scanner.

EEGs often record such fluctuation every millisecond, allowing for strong temporal resolution. However, since the electrodes are placed on the outside of the scalp there is difficulty with the spatial resolution of the recording. The actual output will be a sequence of voltage values over time from each electrode. This output can be decoded with multivariate pattern analysis (MVPA) where all the relationships between time points can be factored in the analysis. **MVPA** is able to be utilized for all the imaging techniques discussed in this chapter as a broad form of decoding that factors in the relationship between variables so they are not treated as independent variables. Overall, EEG is a cheap, non-invasive imaging method that is implemented in many laboratories.

6.3.2 MEG

Magnetoencephalography (MEG) is a brain recording technique similar to EEG. MEG measures the small magnetic fields produced by a population of neurons being activated together, demonstrating areas of the brain that are being highly stimulated. For a MEG scan, the machine encompasses the exterior of the participant's head and must be completed in a magnetically shielded room as the magnetic fields produced by the brain are quite small. Similar to EEG, the output will be a time-series data that is conducive to strong temporal resolution as data is being recorded every millisecond. In this case, instead of voltage over time as in EEG, MEG will give a recording of magnetic flux over time. Since the machine is recording from outside the skull the spatial resolution and only providing data on the activation at each location the spatial resolution is weaker. MEG is more expensive than EEG but provides similar, temporally accurate,

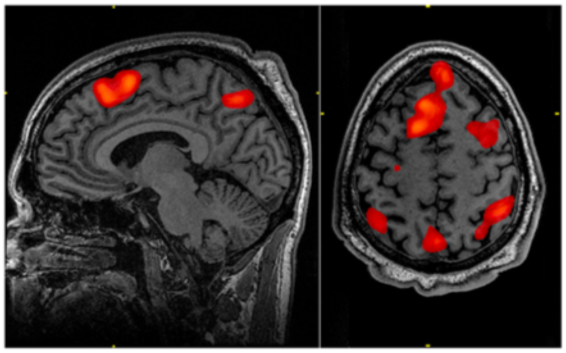


Figure 6.3: Image of an fMRI scan with red sections indicating areas of the brain that are more active than the control condition.

data on the activation of regions of the brain.

6.3.3 fMRI

Functional magnetic resonance imaging (fMRI) is a system that shows where oxygenated blood is focused in the brain, indicating which regions of the brain are most active. Oxygenated and deoxygenated blood have different magnetic properties allowing researchers to differentiate the two in an MRI scanner. Participants are placed in a machine that encompasses their head for the procedure to take place. fMRI scans, while external, are still able to achieve strong spatial resolution through the use of voxels. Voxels are small three dimensional sections that the brain is divided up into, and the color assigned to each voxel represents its level of activation as compared to a standard baseline. All voxels are active the majority of the time, so the output of the scan is focused on which voxels are more activated during a specific task of interest compared to this control activation. The product of an fMRI scan is an image of the participant's brain with regions of higher relative activation indicated with color. These colored regions indicate the presence of more oxygenated blood, as greater activation requires more oxygenated blood to sustain it. Despite its successes in spatial resolution, fMRI scans have poor temporal resolution as they take six to ten seconds for blood-oxygen-level-dependent (BOLD) contrast to show changes after something happens. fMRI provides descriptive, easily interpretable imaging, but requires expensive machinery with poor temporal resolution.

6.3.4 ECOG

Electrocorticography (ECOG) is a brain imaging technique that is not utilized often as it requires electrodes to be placed on the exposed surface of the cortex of a participant's brain. The electrodes record electrical activity in the brain,

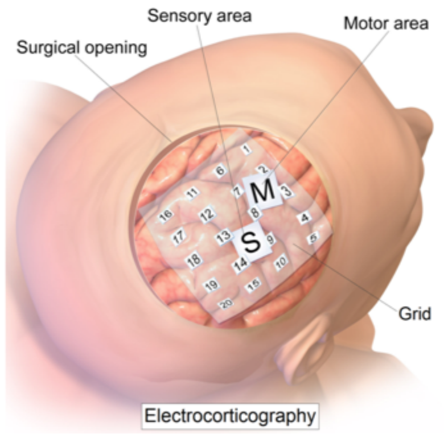


Figure 6.4: Image depicting electrodes placed on cortex for ECOG imaging.

similar to EEG. However, due to the proximal location of the electrodes, ECOG is able to record with both extremely accurate spatial and temporal resolution. ECOG introduces questions regarding ethics and treatment on humans as it requires exposing the surface of the brain. Therefore, ECOG is typically only implemented on participants with epilepsy who require the placement of electrodes on their brain to record from the region where their seizures are centralized. Due to its similarity to EEG, the ECOG output would also be tracking voltage over time from each electrode. ECOG is able to provide critical data but is a difficult and extreme procedure.

Features Exercise: What features can be used in decoding different types of data? Fill in the chart below with corresponding features for each type of data. The first one has been done for you.

Data.Type	Features
Single neurons	Firing rate, spike counts
Population of neurons	.
EEG signals	.
fMRI	.
.	.

6.4 Introduction to Decoding

With all these data from the various brain imaging techniques in hand, we want to **decode** the data–in other words, we want to detect the activity pattern buried in the random, noisy neural firing data and analyze if different stimuli lead to different activity patterns. If by analyzing activity patterns we can "predict" the stimulus that causes the activation and if the accuracy of "prediction" is higher than just random guessing, we can say that we successfully decoded the information in our brain activity recording. The word "prediction" is in quotation marks because we are not foretelling future events, but instead checking the answer key to see if our solution is correct.

Depending on the imaging techniques we used in the data acquisition phase, there are multiple ways to present and analyze data. Popular brain imaging techniques like EEG, MEG, fMRI, and ECOG all give recordings for a population of neurons, and it is the experimenter's choice to focus more on the group level or on local variance. It might be the case that whole-brain EEG data was implemented to analyze event-related brain activation, or it could be that occipital and frontal electrodes were selected to measure their correlation. Another example comes from the field of fMRI, where current interests shifted from the classical approach that explores the involvement of brain regions in certain activities to revealing the representational pattern within functional regions. Depending on the purpose of the study, different classifiers can be selected to categorize the data. **Classifiers** are algorithms that make predictions for test data based on a "learned" pattern from the training set. Here are some types of classifiers generally used in decoding.

6.5 What is a classifier?

In order to decode brain data, we will need a classifier, which allows us to learn from labeled data and make predictions on test data from its experience with the labeled data. We will discuss three types of classifiers which each have their own strengths and weaknesses. Note that all of the classifiers we will talk about fall under the category of supervised learning, as they all make decisions based on labeled training data.

6.5.1 Correlation classifiers

The first classifier we will examine is the simple, yet powerful correlation classifier, which works well for multivariate classifications. In a correlation classifier, the mean of each class is observed and correlated with each input feature. The class most correlated with the test item is the classifier's predicted output. James Haxby does a great job explaining some of the more intricate details to be aware of when implementing a correlation classifier. In practice, we consider each neuron individually as a test point and split the rest into training data. Next, we find all of the training trials for each class and calculate its respective mean. We then find the correlation coefficients between the training data and our test point and assign our predicted class to be the category with the highest correlation. We can then determine the accuracy of this classifier by evaluating the percentage of correctly classified test points.

6.5.2 Distance-based classifiers

The second classifier we will discuss is the distance-based classifier, which bases its decision making on the distance calculated from the test point to the training point. One popular example of a distance-based classifier is the k-nearest neighbors (KNN) algorithm, where k is the number of training points that are closest to our test point. Each training point yields one vote to decide the class for each test point. The class with the most votes is then the resulting classification of that test point. Note that k is conventionally odd to ensure a majority class. To perform the algorithm, we first sample each of our training points. For each of these points, we calculate the distance to the test point. These distances are then sorted to ensure picking the k-smallest distances i.e. nearest neighbors. Finally, we take the most frequent class to be the predicted class of our test point. Before we move on to our next classifier, we must touch on one of the major pitfalls of distance-based classifiers: the curse of dimensionality. Distance-based algorithms work great in low dimensions where it is computationally inexpensive to calculate distances. The **curse of dimensionality** pertains to the fact that, as we move up in dimensions, the distances between all points to one another grow very large and the variability among distances becomes small. Therefore, your data must grow exponentially as dimensions are increased in order to sample enough of the space for these dimensions to be meaningful. It is for this reason that when working with high dimensional data, a distance-based classifier may not be the best choice.

6.5.3 Boundary-based classifiers

The last of the three primary classifiers we will discuss are boundary-based classifiers, which produce some line or curve separating classes in two dimensions, a 2D plane to separate classes in three dimensions, or an abstract boundary (hyperplane) in higher dimensions. One example of a decision-based classifier is the Linear Support Vector Machine, which functions similarly to Linear Discriminant Analysis. **Linear Discriminant Analysis** works by asking what is the best line/hyperplane we can draw to separate the centroid means of our classes, whereas a **Linear Support Vector Machine (LSVM)** instead draws a boundary between the hard examples in the training set. We mentioned that distance-based classifiers are not a good choice of a classifier for higher-dimensional data. Conversely, LSVMs work well with high dimensional data; instead of calculating large distances between points, we only require a single hyperplane separating classes. Data points falling on either side of the hyperplane can be used to predict test point classes. In an LSVM algorithm, we aim to maximize the margin between the training points and the hyperplane via a cost function. Similarly to our neural network weight updating, when our LSVM misclassifies a data point, the weights determining our hyperplane are adjusted by this cost function.

One question you may be asking yourself based on the last chapter, “Wait–what about neural networks? We just learned that they can make decisions based on labeled input data!” This is a natural question to ask and the answer is in fact, yes–neural networks can function as a classification algorithm. However, when creating a classifier, the setup of a neural network is a significant hassle in terms of parameter tuning (layer size, layer count, learning rate, etc). When possible, it makes more sense to work with one of the previously discussed trivial classifiers, which are more lightweight than a neural network and work “out of the box.”

6.6 Cross validation

We talked about that decoding means to make predictions on the test set based on the pattern from the training set. It is worth mentioning that both the test set and the training set, although all from experimental data, should be *clearly separated* during learning and predicting. Otherwise, you will fall into the circular logical fallacy. Formally this is done by **cross-validation**, a scheme that partitions the data set into the test set and the remaining data into the training set, and then predictions can be made on the test set. The training and predicting processes repeat several times to obtain a prediction for every data point. This may sound paradoxical since we mentioned that it is problematic to use A to predict A, but here since the test and training sets are clearly separated for every round of prediction, and the predicted results are *never used* for training, we are not falling into this logical trap.

Cross-validation is a decoding strategy that includes some subtypes like leave-one-out cross-validation, which is an extreme case in the cross-validation family. This method leaves exactly one sample to be the testing set and uses all other samples as the training set. It is worth noticing that when data are classified into multiple groups, one sample must be left out for every group. This is to prevent unbalancing in the training set, as an unbalanced training set may lead to a biased model. This method, while having the best chance to generalize to new examples, also requires very high computational power. A less demanding method is the k-fold cross-validation where *k* splits are made and *(k-1)* groups are assigned to the training set. The remaining group serves as the test set. The training and predicting repeats k times, and it is generally faster than leave-one-out-cross-validation in terms of computing power. The effectiveness of the classifier is evaluated by accuracy, which is usually computed as the average correctness across all samples. Lastly, another method is the **rank measure**, which ranks the probability for all labels and measures the distance between the predicted label and the top. Sometimes it is worth noticing that higher accuracy does not mean better classification. Assume that based on some brain data we want to classify patients into two groups: “children who have ADHD” and “children who do not have ADHD”. It is better to misclassify children without

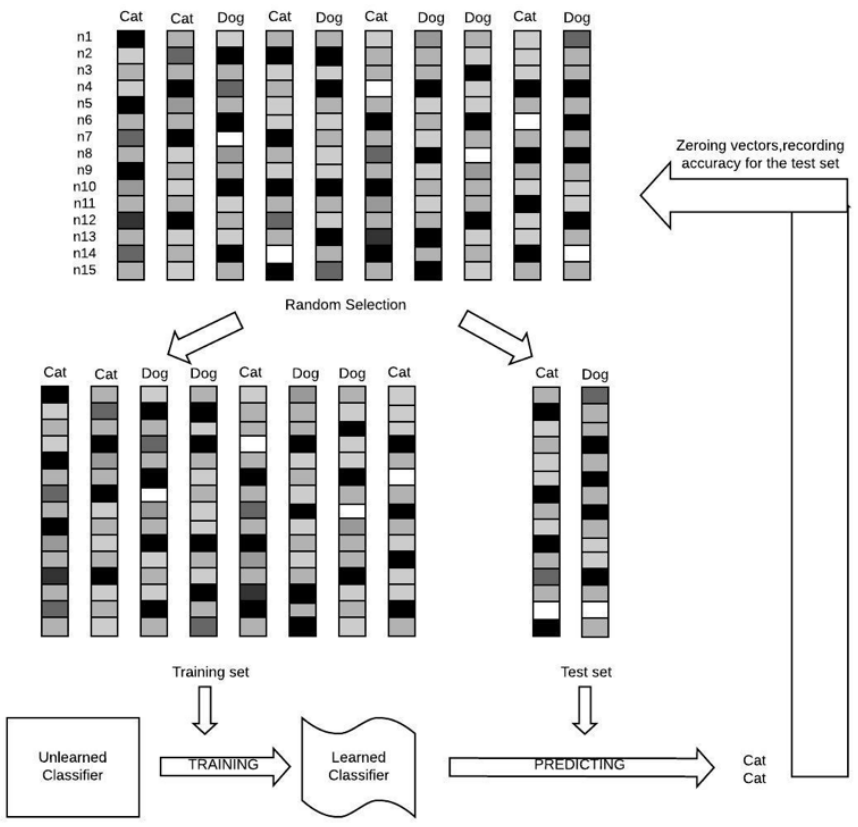


Figure 6.5: The Logical Flow of Cross Validation. Notice that the loop continues until every group of data served as the test group.

ADHD into the ADHD group, compared to classifying children with ADHD to the control group, as the latter one may be more deleterious.

Case study exercise: A group of researchers is studying object recognition in the inferior temporal cortex using ECOG and decoding. The neurons are not all equally responsive because an electrode array was not making full contact with the participant. What issues might this cause in decoding data? Discuss how you might fix this problem.

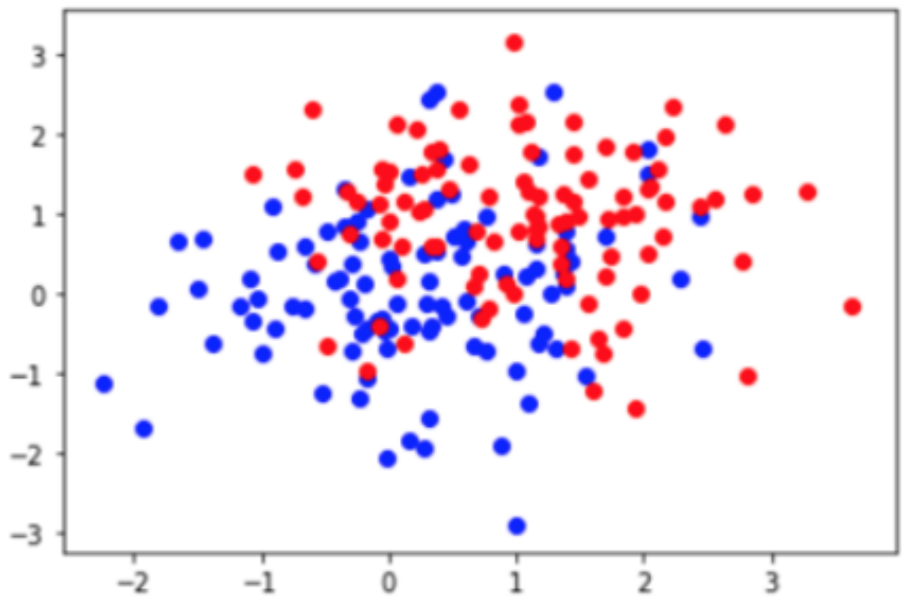
6.7 Conclusion

When performing neural decoding, we are aiming to determine what information about the stimulus is available in the electrical activity of neurons. There are many areas in which decoding can be applied and used to make predictions about what people are thinking, dreaming, seeing, or hearing. In the fast-moving field of Computational Neuroscience, we will likely see great strides made in decoding in all of these areas. It is paramount to our interests as Neuroscientists to be mindful of certain ethical questions as this progress is made. In China, we already see the persecution of both the Uyghur and Hong Kong populations through government-sponsored facial recognition. This abuse of facial recognition technology is a strong indicator that these decoding techniques may similarly be used in the future by malevolent actors. One potential dystopian scenario based on these methods might include crime prediction and imprisonment solely based on brain activity. It is our responsibility to be mindful of how this technology is introduced to the field, as well as the world.

Ethics exercise: Some say that decoding could allow scientists to “read minds”. Here is one article that refers to decoding as “mind-reading”. In light of this, what might be some ethical issues regarding the various applications of decoding? How could we potentially protect against these issues?

Coding exercise: The code below codes for two separate clusters of data with means at 0.05 (blue) and 0.95 (red) which is plotted in a scatter plot.

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(10)
m1 = np.array([0.05, 0.05])
m2 = np.array([0.95, 0.95])
sigma = np.eye(2)
data1 = np.random.multivariate_normal(m1, sigma, 100)
data2 = np.random.multivariate_normal(m2, sigma, 100)
plt.figure()
plt.scatter(data1[:,0], data1[:,1], c='blue')
plt.scatter(data2[:,0], data2[:,1], c='red')
```



Based on this scatter plot and the code above, fill in the following code. The code should generate a test point from either group m1 (the blue scatter plot) or m2 (the red scatter plot). Then the distance between the generated test points and the two means should be found using the pdist() function. You can look up the pdist() function to learn more about it and its arguments. This should be done in a for loop with 100 steps. The code additionally computes the accuracy of the decoding. This part is mostly written for you but remember to initialize the data structure and write the conditional to help determine accuracy.

```
# Importing pdist function
from scipy.spatial.distance import pdist

# Initiate data structure for accuracy
accuracy = np.zeros(   )
mat1 = np.zeros(   )
mat2 = np.zeros(   )

# Loop to classify which group the test point is in
for i in range(   ):
  # Conditional to randomly pick a point from m1 or m2
  if np.random.rand()      :
    myMean =
    realClass =
  else:
    myMean =
    realClass =

  # Define test point
  myPoint = np.random.multivariate_normal(   ,   ,   )

# Calculate the distance to m1 and m2 (row 0: myPoint; row 1: m1 or m2)
mat1[0,:] =
mat1[1,:] =
mat2[0,:] =
mat2[1,:] =

dist1 = pdist(   )
dist2 = pdist(   )

# Conditional to assign predicted class to be the class w/ smallest distance
if dist1   dist2:

else:

# Conditional to determine trial accuracy
if           :
  accuracy[i] = 1

# Calculate mean accuracy
meanAcc = np.sum(accuracy)/100
print(meanAcc)
```

Chapter 7

References

*** Denotes references of particular interest.

7.1 Chapter 1

Anderson, B. (2014). *Computational Neuroscience and Cognitive Modelling: A Student's Introduction to Methods and Procedures*: SAGE Publications.

Hodges, A. (2009). Alan Turing and the Turing Test. Epstein, R., Roberts G., & Beber, G. (Ed.) *Parsing the Turing Test: Philosophical and Methodological issues in the Quest for the Thinking Computer.* (pp. 13-22). Springer.

Lytton, W. W. (2002). *From Computer to Brain: Foundations of Computational Neuroscience*: Springer.

Markram, H. (2006). The Blue Brain Project. *Nature Reviews Neuroscience*, 7(2), 153-160. doi:10.1038/nrn1848

*** Marr, D. (1982). The Philosophy and the Approach. *Vision.* San Francisco: Freeman.

O'Reilly, R., & Munakata, Y. (2000). *Computational Explorations in Cognitive Neuroscience Understanding the Mind by Simulating the Brain.* MIT Press.

P. Trappenberg, T. (2002). *Fundamentals of Computational Neuroscience*: Oxford University Press UK.

*** Pessoa, L. (2017). Do Intelligent Robots Need Emotion? *Trends in Cognitive Sciences*, 21(11), 817-819. doi:https://doi.org/10.1016/j.tics.2017.06.010

Selfridge, O. G. (1955, March). Pattern recognition and modern computers. In *Proceedings of the March 1-3, 1955, Western Joint Computer Conference* (pp. 91-93). ACM.

Studios, BBC, director. The Chinese Room Experiment - The Hunt for AI. YouTube, YouTube, 17 Sept. 2015, www.youtube.com/watch?v=D0MD4sRHj1M.

7.2 Chapter 2

Anderson, B. (2014). *Computational Neuroscience and Cognitive Modelling: A Student's Introduction to Methods and Procedures*: SAGE Publications.

Dayan, P. A., L. F. (2005). *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems* (T. M. Press Ed.).

Discovery. "Dancing Zombie Squid Explained." YouTube, YouTube, 10 Aug. 2011, www.youtube.com/watch?v=JGPfSSUlReM.

Lytton, W. W. (2002). *From Computer to Brain: Foundations of Computational Neuroscience*: Springer.

Mallot, H. A. (2013). *Computational Neuroscience* (S. International Ed.). Switzerland.

P. Trappenberg, T. (2002). *Fundamentals of Computational Neuroscience*: Oxford University Press UK.

Sterratt, D. G., Bruce; Gillies, Andrew; Willshaw, David. (2011). *Principles of Computational Modelling in Neuroscience* (Cambridge University Press ed.).

7.3 Chapter 3

Background: Spike Trains as Point Processes. (n.d.). Retrieved October 10, 2019, from http://www.stat.cmu.edu/~kass/contrib.html#background.

Jaeger, D., Jung, R., & Springer. (2015). "Spike Train." *Encyclopedia of Computational Neuroscience*: Springer.

Mallot, H. A. (2015). "Chapter 2 Receptive Fields and the Specificity of Neuronal Firing." *Computational Neuroscience: A First Course.* Berlin: Springer.

Dayan, P., & Abbott, L. F. (2001). "1.3 What Makes a Neuron Fire?" *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*: MIT Press.

Ringach, D., & Shapley, R. (2004). Reverse correlation in neurophysiology. *Cognitive Science*, 28(2), 147–166. doi: 10.1207/s15516709cog2802_2

Schwartz, O., Pillow, J. W., Rust, N. C., & Simoncelli, E. P. (2006). Spike-triggered neural characterization. *Journal of Vision*, 6(4), 13. doi: 10.1167/6.4.13

Rieke, F. (1999). *Spikes: exploring the neural code.* Cambridge, MA: MIT Press.

7.4 Chapter 4

3Blue1Brown. “Backpropagation Calculus | Deep Learning, Chapter 4.” YouTube,YouTube, Nov. 2017.

Anderson, B. (2014). *Computational Neuroscience and Cognitive Modelling: A Student’s Introduction to Methods and Procedures*: SAGE Publications.

Baker, Bowen. “Emergent Tool Use from Multi-Agent Interaction.” OpenAI, OpenAI, 29 Oct. 2019, openai.com/blog/emergent-tool-use/.

Glosser.ca. (2013). Colored neural network. Wikimedia.

Kang, N. (2017). Introducing Deep Learning and Neural Networks — Deep Learning for Rookies. Towards Data Science.

Kang, N. (2017). Multi-Layer Neural Networks with Sigmoid Function— Deep Learning for Rookies. Towards Data Science.

*** Lettvin, J. Y., Maturana, H. R., McCulloch, W. S., & Pitts, W. H. (1959). What the Frog’s Eye Tells the Frog’s Brain. *Proceedings of the IRE*, 47(11), 1940-1951. doi:10.1109/JRPROC.1959.287207

Lytton, W. W. (2002). *From Computer to Brain: Foundations of Computational Neuroscience*: Springer.

Mallot, H. A. (2013). *Computational Neuroscience* (S. International Ed.). Switzerland.

Murphy, K. P. (2012). *Introduction Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)* (1st ed.).

P. Trappenberg, T. (2002). *Fundamentals of Computational Neuroscience*: Oxford University Press UK.

Silver, David, et al. “AlphaZero: Shedding New Light on the Grand Games of Chess, Shogi and Go.” Deepmind, Dec. 2018, deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go.

Tesla. “Full Self-Driving.” YouTube,YouTube,www.youtube.com/watch?v=tlThdr3O5Qo.

7.5 Chapter 5

Glover, G. H. (2011). Overview of Functional Magnetic Resonance Imaging. *Neurosurgery Clinics of North America*, 22(2), 133–139. doi: 10.1016/j.nec.2010.11.001

Grootswagers, T., Wardle, S. G., & Carlson, T. A. (2017). Decoding Dynamic Brain Patterns from Evoked Responses: A Tutorial on Multivariate Pattern Analysis Applied to Time Series Neuroimaging Data. *Journal of Cognitive Neuroscience*, 29(4), 677–697. doi: 10.1162/jocn_a_01068

Haxby, J. V. (2012). Multivariate pattern analysis of fMRI: The early beginnings. *NeuroImage*, 62(2), 852–855. doi: 10.1016/j.neuroimage.2012.03.016

Lecture 2: k-nearest neighbors. (n.d.). Retrieved from http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote02_kNN.html.

Kriegeskorte, N., & Kreiman, G. (2012). *Visual population codes: toward a common multivariate framework for cell recording and functional imaging.* Cambridge, MA: MIT Press.

Singh, S. (2014). Magnetoencephalography: Basic principles. *Annals of Indian Academy of Neurology*, 17(5), 107. doi: 10.4103/0972-2327.128676

Chapter 8

Glossary

- Absolute Refractory Period: The point from the beginning of the action potential to the peak. In this time frame, it is not physiologically possible for the neuron to fire a second time. In this time period, the sodium channels are open and remain open until the peak of the graph. These channels can not immediately re-open, and due to this, it would not be possible for the membrane to depolarize a second time.
- Activation Function: Allows for a neuron to make a decision (produce an output) along some continuous interval to adjust weights to learn.
- Algorithm: An algorithm is a process to define why the model is appropriate and how the logical strategy will be carried out.
- Back Propagation: Correction signals that run backwards from the output units to the hidden units and then are summed according to the hidden-to-output weights.
- Bottom-up Processing: Bottom-up organization refers to the reverse process, collecting data and then organizing it to create a theory.
- Classifier: A type of supervised learning that learns from training data and makes predictions on test data. Specific types of classifiers include:
  - Distance-based classifier
  - Boundary-based classifier
- Coefficient of Variation: Standard deviation divided by the mean of a set of data.
- Computational Neuroscience: Computational neuroscience is an interdisciplinary field that applies the principles of mathematics, philosophy, and computer science to study the inner workings of the brain.

- Computational Theory: Computational Theory a characterization of the system's goal.
- Conductance: Allows the flow of charge.
- Cost: Calculation in Backpropagation using the Mean Squared Error: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Yhat_i)^2$. Used to change weights with the goal of minimizing cost at each iteration.
- Cross-validation: A decoding scheme that repeatedly partitions the data set into a test set and the remaining data into a training set and makes predictions for every test set until predictions are made for every data point in the data set.
- Curse of dimensionality: As we move up to in dimensions and start calculating distances in hyperplanes, these operations become exponentially less efficient.
- Decoding: Field of neuroscience aimed at using action potential data in a single neuron or neural networks to identify the stimuli that caused the neural activity.
- Depolarization: This process of positive ions flowing into the cell.
- Driving Force: The pressure for an ion to move in or out of the cell.
- Emergent Phenomena: An emergent phenomenon is a case in which new mechanisms arise from the addition of a sufficient number of the same functional part.
- Equilibrium Potential: The membrane potential at which the flow of electric current from all types of ions into and out of the cell is balanced, so there is no net current and the membrane potential is not caused to change.
- Fano Factor: The fano factor is used to measure the spike variability in a spike train. It is calculated by the variance of the number of spikes divided by the mean number of spikes in a given time interval.
- Gating variable: The Fire Model is broken up into three separate conductance terms, each relating to a different ion channel called the m, h, and n variables.
- Hardware and Implementation: Hardware implementation is the physical machinery that realizes the algorithm.
- Hebbian Learning: One of the core levels of organization within the brain is the synapse. The synapse consists of a pre-synaptic cell, which sends a message to another neuron, or the post-synaptic cell. When many of these messages are sent between two cells, the connection between the two of them is strengthened. This is a theory of "synaptic plasticity" which can be applied through theoretical models within the field of neuroscience.

- Hidden Layers: Additional layers used when an output is not linearly separable (like XOR); these layers of neurons are chained together via multiple nonlinearities from the various units to solve the problem. Each of these hidden layers–as well as the output layer–will have its own activation function.
- Hyperpolarization: To decrease the membrane potential towards a more negative value through outward electrical current.
- Imaging techniques: There are multiple types of imaging techniques commonly employed by researchers to obtain data and recordings from participants. The techniques include:
  - EEG
  - MEG
  - fMRI
  - ECOG
- Interspike Interval: The time interval between every pair of spikes.
- Leak Current: Leak currents are the passive membranes that are dependent on the membrane potential to drive the electrical potentials of the permeable ions and concentration gradient.
- Linear discriminant analysis: Type of decision-based classifier algorithm that maximizes that distance between centroid means.
- Linear support vector machine (LSVM): type of decision-based classifier algorithm that creates a boundary that maximizes the distance between the hard examples in the training set (known as the support vectors); LSVM works well with high dimensional data.
- Linearly Separable: Different classes of outputs in space that can be separated with a single decision surface.
- McCulloch-Pitts (MCP) Neuron: Initial neural network model designed by McCulloch and Pitts that takes multiple inputs with associated weights to produce a single output.
- Membrane Potential: The potential difference across the cell membrane.
- Multivariate pattern analysis (MVPA): Method utilized for all the imaging techniques as a broad form of decoding that factors in the relationship between variables so they are not treated as independent variables.
- Negative Feedback: A process by which an initial change is opposed by a force caused by the initial change.
- Nernst Potential (Reversal Potential): The membrane potential at which the flow of a particular ion is in a dynamic equilibrium, meaning the outflow is precisely matched by the inflow of that ion.

- Neural Networks: Computing model comprised of basic processing elements strung together that take an input and give an appropriate output and can become increasingly layered to conquer more complex concepts and problems.
- Perceptron: An algorithm for transforming inputs to outputs with the corresponding weights, bias, and the activation function.
- Peri-Stimulus Time Histogram: Average time-dependent rate of action potentials (or spike rate) measured during a stimulus over a period of time.
- Poisson Process: Probabilistic production of events, such as spikes, at any point in time with equal probability per unit time.
- Positive Feedback: A process by which depolarization of the cell causes further depolarization. More generally, a positive feedback loop is a process that perpetuates itself.
- Rank measure: A type of decoding that ranks the probability for all labels and measures the distance between the predicted label and the top.
- Reconstructionism: Reconstructionism is similar to reductionism, except with the added step of reconstructing the reduced parts.
- Reductionism: Reductionism is breaking larger concepts or models down into smaller parts.
- Reinforcement Learning: Learning shaped through interactions with the environment through reward and punishment.
- Relative Refractory Period: The point after the absolute refractory period when a second stimulus, that is above a threshold, can elicit a second action potential without allowing the membrane to hyperpolarize back to its resting membrane potential.
- Representation: The representational scheme is the description of the functional elements that are used in the computation.
- Reverse Correlation: Process which implements the analysis of outputs to determine the inputs that the neuron will respond to with a spike.
- Sigmoid Activation Function: One type of non-linear activation function that determines the output, whose function is defined to be $f(x) = \frac{1}{(1+e^{-}x)}$
- Sodium-Potassium Pump: Uses just below 10% of your body's daily energy to pump three sodium ions out of the neuron for every two potassium ions pumped in, thus forming two respective concentration gradients.
- Spike Count Rate: The number of spikes per time interval.

- Spike Train: A sequence of recorded times at which a neuron fires an action potential.
- Spike Triggered Average: The average value of the stimulus during some time interval before a spike occurs.
- Step Function: One type of activation function that takes returns a binary output of 0 or 1.
- Supervised Learning: Learning situations where inputs and expected outputs are given information to predict future solutions.
- Top-down Processing: Top-down organization refers to the idea of designing a machine for an express predisposed class.
- Turing Machine: The Turing Machine is a theory created by Alan Turing involving an infinite strip of paper with binary cells which can be used to compute any questions theoretically.
- Unsupervised Learning: Learning that occurs in the absence of a teacher with expected outputs; a student simply looks at patterns and tries to maximize correlations or find a basic understanding or pattern.
- White Noise: A random variation where the value at each time point is independent of all other points and so can be employed in these instances to provide a receptive field without bias.