[1]

Pattern theory: old and new

Govind Menon

October 28, 2020

[1]This work is supported by the National Science Foundation (DMS 1714187).

Overview

The term *pattern theory* describes a Bayesian formalism for inference that was developed at Brown University by Ulf Grenander and his colleagues, especially Stu Geman, Basilis Gidas and David Mumford. Several variants of these ideas have been independently created and used by applied mathematicians, computer scientists and electrical engineers across the scientific community. In my view, what is distinctive about the work of my colleagues at Brown is a particular scientific spirit that bridges mathematical foundations with applications in science and (increasingly) the humanities. The purpose of these notes is to convey this spirit along with concrete mathematical techniques.

My interest in this circle of questions was stimulated by many years of animated conversations with Basilis Gidas. About five years ago, I began to engage with these ideas more carefully, to try and learn what the fuss was all about. As all mathematicians know, the best way to learn a subject is to teach it. Thus, I developed a one-year sequence consisting of a rigorous treatment of information theory in Fall, followed in Spring by a seminar that emphasized applications. The material for the first semester was cast in stone – it is impossible to improve on Shannon's development of his ideas and it was impossible to do better pedagogically than my colleagues Stu Geman and Matt Harrison. But there was a great deal of variability in the topics in the Spring semester and it took me three attempts to converge on a set of topics that felt about right. I express my gratitude to the many students whose enthusiasm for the material – often presented in scratchy and half-formed lectures– helped me comprehend the contours of this fascinating subject.

These notes are a faithful account of the topics presented in Spring 2020 to an audience of about 50 students, roughly split between undergraduate and graduate students. Most of these students had a strong background in mathematics, computer science, or physics. In contrast with courses in machine learning, pattern theory stresses the mathematical (and even philosophical) foundations of cognition and learning. The main challenge was to convey both foundations and applications in a way that made sense to undergraduates headed to the tech industry, as well as graduate students exploring topics for research. In order to 'scale-up' a seminar, the class was split into fast-moving lectures and substantive projects that matched the lectures, but required computational skill to implement. The lectures emphasize mathematical foundations and some parts (e.g. Ch.3–Ch. 6) are a crash course on stochastic processes. The projects in 2020

involved breaking a substitution cipher, the recognition of a music score, and character recognition from the MNIST database. I express my thanks to Zsolt Veraszto for his able assistance with the projects.

The primary source for these notes is the book *Pattern Theory* by Mumford and Desolneux [12]. These notes cover the same territory upto Chapter 7; the main difference with [12] is expository. Mumford and Desolneux's book was designed for a graduate topics class and the ideas and exposition require greater maturity than my students possessed. Thus, these notes include some background material, as well as a rearrangement of some of the material from [12]. The later chapters differ drastically from [12]. The main reason for this is that the explosion of research in machine learning has revealed flaws in the pattern theory formalism. While pattern theory remains a principled approach to artificial intelligence as a science, it has been outgunned as a technology. Convolutional neural networks implemented on GPUs solve several pattern theory problems (e.g. face recognition) in a fast and stable way. Therefore, while the detailed construction of hierarchical priors for computer vision was an important feature of pattern theory ten years ago, it doesn't seem as relevant to me today. Deep learning has won (albeit on a narrow class of cognition problems) and the more urgent task is to teach the students about CNN's, in order that they understand what is beneath the hood in most machine learning applications, and to get them to think more carefully about why deep learning works as well as it does.

My goal at the outset of the semester was to convey both these viewpoints. First, to explain the pattern theory formalism in problems of increasing complexity (text, music, speech, character recognition). Second, to explain the deep learning paradigm along with an implementation of benchmark problems such as character recognition and face recognition. I had also intended to treat Chomsky's grammars and the use of linguistic metaphors in pattern theory.

The class largely followed this plan, until the covid-19 crisis. The dismay of many students at the disruption of their education, as well as the global crisis, led me in a more introspective direction. My initial ambitions transmuted into a broader goal to provide students with a sense of how applied mathematics actually works – to demonstrate an interplay between modeling, analysis and computation characteristic of our discipline – as well as to share the eternal delight of beautiful mathematics. Thus, the later chapters are surveys in which I have tried to communicate essential ideas, rather than detailed techniques. These include two lectures on fast numerical methods and an introduction to fundamental limits and thermodynamics. The latter topic was largely inspired by Yann Le Cun's talk at IAS in Spring 2019 on the epistemology of deep learning. It leads naturally into the notion of reversible computation, with applications in molecular biology, following the work of Bennett and Landauer, but this must wait for another time (the interested reader is referred to [6, Ch.5]). I chose instead to discuss the philosophical debates on the nature of mathematics in the early 20th century that led to the creation of the computer in the 1940s. There is to my mind no greater demonstration of the power of mathematical thought to change the world.

Contents

Chapter 1

The elements of style

1.1 Introduction

The main idea of pattern theory is as follows. We assume that the world is a space of random signals. An observer records part of these signals and forms inferences about the world. The process of inference is modeled with a Bayesian paradigm. We assume the observer has a stochastic model that is capable of generating signals that are similar to the true signal. Inference then reduces to tuning the parameters of the model to obtain the best match between the generated and observed signals (in technical terms, we maximize a posteriori likelihood). The devil lies in the details since the difficulty of formulating a stochastic model and optimizing the parameters depends on the complexity of the underlying signal. For example, inferring properties of written text is much simpler than detecting edges in an image. Text consists of a linearly ordered string of a finite set of symbols. Human vision, on the other hand, decodes a three-dimensional arrangement of objects in space based on a two-dimensional projection onto the retina.

For these reasons, we study stochastic models of increasing complexity in tandem with numerical algorithms for optimization. The simplest class of stochastic models we consider are Markov chains. The purpose of this lecture is to review the basics of Markov chains and to apply these ideas to a stochastic model for written text introduced by Shannon [13, 14] [1] The power of these ideas is demonstrated in the first project: you will be given a scrambled body of text and your task is to decode it using the methods presented in the first two chapters.

[1]The first Markov chains were introduced to model problems of human cognition and communication. Markov himself was introduced in analyzing style in poetry and modeled Pushkin's novel in verse *Eugene Onegin* by tabulating the frequencies of transitions between consecutive vowels and consonants in the text (he did this by hand for 5446 lines of verse!). This idea was rediscovered by Shannon, who developed a set of models for language that we describe in Section 1.3 below.

1.2 Markov chains

1.2.1 General theory

We assume given a finite set $\mathcal{S}$ called the *state space* of the Markov chain. In some instances, such as modeling text, we denote the state space by $\mathcal{A}$ and call it the alphabet. The variable $t = 1, 2, 3, \ldots$ is discrete time. We will study a stochastic process $\{X_t\}_{t\in\mathbb{N}}$ such that X_t takes values in $\mathcal{S}$.

Definition 1. The stochastic process $\{X_t\}$, $t \in \mathbb{N}$, is a Markov chain if for all t

$$\mathbb{P}(X_{t+1}|X_1, \ldots, X_t) = \mathbb{P}(X_{t+1}|X_t).$$

A Markov chain is the simplest instance of a more general class of stochastic processes, called Markov processes. We use the term Markov chain when time is discrete (as above). In the examples in this chapter, both time and the state space are discrete. This allows us to build intuition, while also including many interesting applications.

The main modeling assumption in Markov process theory is that the future of the process depends on the current state, but not the past. This is the probabilistic analogue of the idea of Newtonian determinism (that the evolution of the future of a physical system can be completely determined by solving equations of motion, given the current state). For Markov processes, we can predict the law of the future evolution using the forward equation described below. Another interpretation of the Markov property is that the future and past are independent, conditional on the present. We typically simplify the structure of Markov chains with the following assumption.

Definition 2. The Markov chain $\{X_t\}$ is stationary if $\mathbb{P}(X_{t+1}|X_t) = \mathbb{P}(X_2|X_1)$ for all $t \in \mathbb{N}$. The transition matrix for a stationary Markov chain is

$$Q(x, y) = \mathbb{P}(X_2 = y|X_1 = x).$$

The most basic statistic associated to the Markov chain is the pmf (or law) of the random variable X_t [2]. We denote this by

$$\pi_t(x) = \mathbb{P}(X_t = x), \quad x \in \mathcal{S}. \tag{1.2.1}$$

We use the Markov property to obtain the recurrence relation

$$\begin{aligned} \pi_{t+1}(y) &= \mathbb{P}(X_{t+1} = y) \\ &= \sum_{x\in\mathcal{S}} \mathbb{P}(X_{t+1} = y|X_t = x)\mathbb{P}(X_t = x) = \sum_x Q(x, y)\pi_t(x), \end{aligned} \tag{1.2.2}$$

when we assume that the chain is stationary. This calculation yields the *forward equation*

$$\pi_{t+1} = \pi_t Q, \quad t \in \mathbb{N}. \tag{1.2.3}$$

[2] The abbreviation pmf stands for probability mass function. This term is used in applied probability when we work with discrete random variables.

Here we use the convention that π_t is a row vector (which is why we multiply Q by π_t on the left). The forward equation is linear and explicitly solvable when the chain is stationary. We proceed inductively to find

$$\pi_t = \pi_1 Q^t, \quad t \in \mathbb{N}. \tag{1.2.4}$$

One of the central questions in Markov chain theory is to understand the behavior of π_t as $t \to \infty$.

Definition 3. The pmf π is an equilibrium (or stationary) distribution if

$$\pi = \pi Q, \tag{1.2.5}$$

and π satisfies the normalization conditions

$$\pi(x) \geq 0 \quad \text{for all } x \text{ and } \quad \sum_x \pi(x) = 1. \tag{1.2.6}$$

The first equation expresses the fact that $\pi_t = \pi$ for all t if $\pi_1 = \pi$. That is, while the values of X_t change with time, its pmf does not. Thus, the equilibrium distribution, or equilibrium measure, of a Markov chain captures the intuitive idea of a *dynamic* equilibrium. The 'macroscopic' observable π_t is independent of t, whereas the 'microscopic' random variable X_t fluctuates in time.

Let us now turn to the task of computing π, given Q. Observe that π is a *left* eigenvector of Q with eigenvalue 1. Further, 1 must always be an eigenvalue of Q, since it corresponds to the *right* eigenvector $(1, \ldots, 1)^T$. Thus, if we know how to solve for eigenvectors, we can determine π. What is interesting in practice is actually the converse: for large transition matrices, say when $|\mathcal{S}|$ is $10^6 \times 10^6$, it is more efficient to use Markov chains to approximate π, than to use naive linear algebra. This observation plays an important role in web crawlers and search engines (as discussed in Section 10.6).

A central question in Markov chain theory is to understand how fast it 'mixes'. For example, how many shuffles does it take to obtain a truly random distribution if one begins with a perfectly ordered deck of cards?

The mathematical formulation of this problem is the following. We assume given a Markov chain with a unique equilibrium π, and we'd like to obtain rates of convergence for π_t to approach π. Thus, when shuffling a deck of cards, we model it as a Markov chain in the permutation group on 52 symbols, with different forms of shuffling determining different random walks in the permutation group. The act of shuffling a new pack of cards then translates into a random walk in the permutation group that begins at the identity permutation. The analysis of how fast the pmf π_t approaches the uniform distribution is then converted into the analysis of the eigenvalues of the transition matrix. An extensive study of these methods may be found in [5]. While we won't consider any rigorous theorems of this type in these notes, the following term will be used.

Definition 4. A stationary Markov chain X_t is *ergodic* if it has a unique equlibrium distribution.

Many conditions are known that ensure ergodicity. For example, if all terms of Q are strictly positive, it is possible for any state to get to any other state, and this ensures that the Markov chain is ergodic. This assumption is too restrictive in practice, since we are mainly interested in Markov chains with local moves such as random walks in a large state space. However, the Markov chain is ergodic if for every pair $x, y \in \mathcal{S}$ there is an integer $n(x, y)$ such that $Q^{n(x,y)} > 0$. Intuitively, this means that every state can visit every other state given enough time. We will always assume that this property holds.

Physicists define the term ergodic to mean the following

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} f(X_t) = \mathbb{E}_\pi(f),$$

where $\pi = \pi Q$ and $f : \mathcal{S} \to \mathbb{R}$ is arbitrary. This formulation has the interpretation that a Markov chain is ergodic when the "time average" (the left hand side) is equal to the "spatial average with respect to the equilibrium measure" (the right hand side). For Markov chains, this property is a consequence of the definition above, but it provides a different way to think about ergodicity for other classes of stochastic processes (such as stationary processes, discussed in Section 1.3 and Chapter 3).

Markov chains are important because they offer us a conceptually simple theory with many applications. It is important to note the following typical bottlenecks and simplifications.

1. $\mathcal{S}$ may be very large. For example $\mathcal{S}$ may be the permutation group S_N with size $|S_N| = N!$.

2. $Q(x, y)$ is typically sparse.

3. Often we don't need π, all that we need is to evaluate $\mathbb{E}_\pi(f)$, where $f : \mathcal{S} \to \mathbb{R}$ is a given function and $\mathbb{E}_\pi(f) = \sum_{x\in\mathcal{S}} \pi(x)f(x)$. For example, when studying the Ising model, our state space $\mathcal{S} = \{-1, 1\}^N$ and we 'move' in this state space by locally flipping a plus one to a minus one and vice versa. Rather than evaluate π in general, what we usually care about is the average of functions f such as $f(x) = \{\text{number of plus ones in } x \in \mathcal{S}\}$.

1.2.2 Examples

Example 5 (Random walks on graphs)**.** Let $G = (V, E)$ be a graph with vertex set V and edges E. Here the state space $\mathcal{S}$ consists of the vertices V. We may use the edges to define distances on the graph in the natural way, by defining $|x - y| = 1$ if x and y are joined by an edge. The degree of a vertex x is

$$d(x) = \sum_{y\to x} 1.$$

The uniform (or unbiased) random walk on G is the Markov chain with transition matrix

$$Q(x,y) = \begin{cases} 0, \text{ if } |x-y| \neq 1 \\ \frac{1}{d(x)}, \text{ if } \quad |x-y| = 1. \end{cases}$$

Example 6 (Random walks in the permutation group)**.** We consider the state space $\mathcal{S}$ consisting of permutations on N symbols. More formally, recall that a permutation on N symbols is a 1-1 function $\sigma : \{1, \ldots, N\} \to \{1, \ldots, N\}$, $12 \ldots N \mapsto \sigma_1 \sigma_2 \ldots \sigma_N$. We denote the set of permutations by $S_N =$. This is our basic example of a large, finite set. Recall that $|S_N| = N!$ and that $N! \approx N^N$ by Stirling's formula for large N.

In order to define Markov chains on S_N, we typically pick a set of simple rules that allow us to transform one permutation into another. One of the simplest such rules, is to say that $|\sigma - \tau| = 1$ when σ and τ are related by a transposition of adjacent elements as illustrated in the figure on the left. Another natural rule is to transpose any two symbols with equal probability. The number of edges adjacent to each vertex in these schemes differ. If we only allow the transposition of neighbors, σ has $N-1$ elements. On the other hand, when we allow transpositions of arbitrary pairs of symbols, for each pair of indices $\{j,k\}$, $1 \leq j < k \leq N$, we may join σ to τ_{jk} where $\tau_j = \sigma_k$, $\tau_k = \sigma_j$ and $\tau_i = \sigma_i$ for all other indices i. Thus, each permutation σ has $\binom{N}{2}$ neighbors τ. Random walk on this graph is ergodic and its equilibrium measure is uniform, that is

$$\mathbb{P}(\sigma) = \frac{1}{N!}, \quad \sigma \in S_N.$$

In our model of text, $N = 27$, but this assumption is clearly not necessary for the above argument.

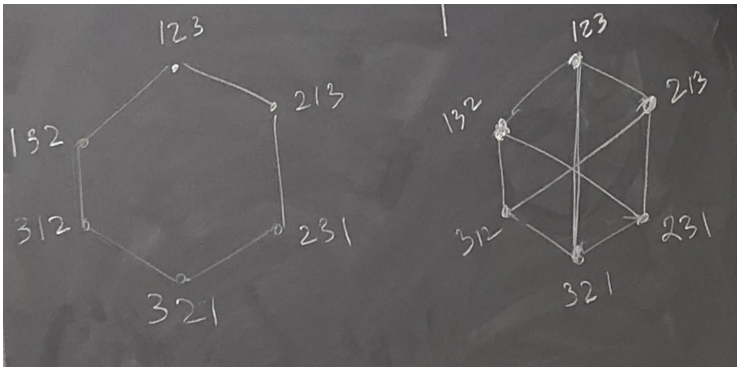

Figure 1.2.1: In the first picture edges correspond to transpositions of adjacent elements. In the second one the edges are transpositions of pairs of symbols.

Other rules may be introduced. In particular, different ways of shuffling a deck of cards correspond to different notions of nearest neighbors in the permutation groups.

1.3 Shannon's model of text (written language)

The main idea in Shannon's model is that text is a stationary ergodic stochastic process $\{X_t\}$ taking values in an alphabet $A = \{a, b, c, \ldots, z, \_\}$. This model is easily augmented to included punctuation and case, but we ignore these concepts in the first approximation.

Note that we are not assuming that this process is a Markov chain. To clarify this point, let us note that stationarity is distinct from the Markov property (neither implies the other).

Definition 7. A discrete time stochastic process is *stationary* if the probabilities are invariant under arbitrary shifts. More precisely,

$$\mathbb{P}(X_1 = x_1, \ldots, X_p = x_p) = \mathbb{P}(X_{k+1} = x_1, \ldots, X_{k+p} = x_p)$$

for all positive integers k and p.

Intuitively, this means that if we were to observe strings of length p, their statistics are not changed by shifting them forward in time by an integer k. For such processes, the notion of ergodicity again means that 'time averages' are equal to 'spatial averages', though now we must take functions of the form $f_p(x_1, x_2, \ldots, x_p)$ and equate the time averages:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} f_p(X_{t+1}, X_{t+2}, \ldots, X_{t+p})$$

with the 'spatial average'

$$\mathbb{E}(f_p(X_1, \ldots, X_p).$$

The above assumptions are introduced to conform to the idea that written text contains many hierarchical structures: letters are joined by phonetic rules to form words, words are linked by the rules of grammar into sentences, sentences are organized into paragraphs, and so on. Nevertheless, these complex rules are rapidly learnt by children, as they discover the rules empirically by playing with constructions. Strange as it may seem at first sight, these rules can be effectively modeled as random processes. The use of stationary ergodic processes provides us with a definition that is flexible enough to include hierarchical structures (though these may be complex to write down), and simple enough to allow us to computationally test the power of this idea (as in your homework).

The relation between the above discussion and the previous sections is that we may use Markov chains of increasing complexity to approximate the above "true" language. We will denote the law of the stationary stochastic process modeling written text by $\mathbb{P}_{true}$. In practice, this law is approximated by mining a large corpus (e.g. the works of Shakespeare) to determine the probabilities $\mathbb{P}_{true}(a_1, \ldots, a_p)$ for an arbitrary string $(a_1, \ldots, a_p) \in A^p$, $p \in \mathbb{N}$.

Example 8 (Digram model)**.** The state space is the alphabet $\mathcal{S} = \mathcal{A} = \{a, b, \ldots, z, \_\}$.

$$Q^{(2)}(x, y) = \mathbb{P}_{true}(X_2 = y | X_1 = x) = \frac{\mathbb{P}_{true}(X_2 = y, X_1 = x)}{\mathbb{P}_{true}(X_1 = x)}$$

This is called the digram model because text is modeled as a Markov chain where each letter depends only on the previous letter and nothing else. Intuitively, this seems too naive, so we can try to improve it by including more 'history'. For example, we may consider a Markov chain where each letter depends on the two preceding letters, not just its immediate predecessor. This yields

Example 9 (Trigram model)**.** Now our state space consists of pairs of letters $\mathcal{S} = \mathcal{A} \times \mathcal{A} = \{aa, ab, \ldots, az, \ldots, \_a, \ldots, \_\_\}$. Let $x = a_1a_2$, $y = b_1b_2$. In order to form a text of three letters from x and y, we must ensure that x and y agree on their overlap. That is, we can only form a text when $a_2 = b_1$ so that x and y combine to give us the string $a_1a_2b_2$.

With this restriction on x and y we obtain the associated transition matrix

$$Q^{(3)}(x, y) = \frac{\mathbb{P}_{true}(a_1a_2b_2)}{\mathbb{P}_{true}(a_1a_2)}$$

and $Q(x, y) = 0$ when $a_2 \neq b_1$.

The obvious generalization of these examples is

Example 10 (n-gram model)**.** The generalization of the above examples is as follows. The state space is now $\mathcal{S} = \mathcal{A}^n$ (the n-fold product of A). A state x, say $x = (a_1, \ldots, a_n)$ can be followed by a string $y = (b_1, \ldots, b_n)$ only if $a_2 = b_1$, $a_3 = b_2$, $\ldots$, $a_n = b_{n-1}$. With this restriction, we find the transition matrix

$$Q^{(n)}(x, y) = \frac{\mathbb{P}_{true}(a_1, a_2, \ldots, a_n, b_n)}{\mathbb{P}_{true}(a_1, a_2, \ldots, a_n)}.$$

A more detailed analysis of these models and samples of text generated by these Markov chains are presented in Section 3.3. Shannon proved that as $n \to \infty$, the law of text generated by the above Markov approximations converges to the law of the true language. Note however that a good proof may not correspond to a good algorithm. For example, as n increases, the size of the state space grows exponentially, since it is $|A|^n$. It follows that the matrix $Q^{(n)}$ requires storage of size $|A|^{2n}$, since its rows and columns are of size $|A|^n$. Worse yet, the size of the training data (to determine $Q^{(n)}(x, y)$ by mining), also expands exponentially.

This follows the rules of everyday language. Once n gets large enough, say 5 or 6 in practice, it is much more efficient to make our fundamental unit words, rather than letters, since the number of true words of length 5 is much lower than the 26^5 combinations that are possible. This reflects the true nature of the space _ as a special character denoting breaks between words. In effect, we are still using the digram model, we have simply switched to a new 'alphabet' whose fundamental units are words along with a Markov chain on words.

The development of written language is a fascinating chapter in human history. The most common writing systems used today are the writing systems developed in the Eurasian landmass. These show a sharp East-West divide. In the West, writing systems coalesced into a small number of scripts (Greek,

Latin, Cyrillic and a few related scripts). "Breaking the code" for an unknown script is a challenging problem that requires a combination of cultural and scientific knowledge. A striking example of this is Ventris' work on Linear B, a Minoan text from Crete, that was not expected to be a Greek language, until Ventris showed this to be the case.

Much less is known about Eastern scripts, because of historical reasons and the large number of written languages. Even the organization of the scripts used today as a family tree with a clear origin, is challenging. The Brahmi script, which is phonetic, originated in India at least as early as 500 BCE and has given birth to a great diversity of written scripts used today in South and Southeast Asia. The Chinese scripts are of a fundamentally different character from the Brahmi tree. Several scripts, such as the Khitan scripts from Mongolia, remain undeciphered. The development of Hangul in Korea served to greatly simplify older writing systems and is a wonderful example of the improvement in mass communication made possible by treating writing as a scientific problem.

I am not aware of careful mathematical models of the development of these scripts, though it is certainly possible to model the evolution (and decoding) of these scripts with the above ideas.

1.4 The Gibbs distribution

First, let us define the Gibbs distribution in the form in which it usually appears in physics. The setup is as follows. We assume given a state space $\mathcal{S}$, an energy function $E : \mathcal{S} \to \mathbb{R}$ and an inverse temperature $\beta > 0$. The associated Gibbs distribution is the pmf

$$p_\beta(x) = \frac{e^{-\beta E(x)}}{Z_\beta}, \quad x \in \mathcal{S} \tag{1.4.1}$$

where the normalization factor

$$Z_\beta := \sum_{y \in \mathcal{S}} e^{-\beta E(y)}$$

is called the *partition function*. An important task in statistical mechanics is to compute the partition function, since all physical observables can be obtained from it. These computations typically require great ingenuity.

The purpose of this section is to provide an explanation for the ubiquity of the Gibbs distribution on Bayesian principles. This avoids the reliance on heuristics from physics, in particular the use of ergodic theorems, and clarifies the fundamental role of entropy in constructing models for inference. This derivation is due to Jaynes (1957) and we will use it at several points in these notes [9]. But first let us briefly provide some context on the origin of the Gibbs distribution, since this has both cultural and scientific importance.

Statistical mechanics is the study of the macroscpic properties of matter based on microscopic models. A fundamental example of a macroscopic property are the laws of an ideal gas (such as the expression $pV = nRT$ that may be

familiar to you from high school chemistry). The microscopic model in this example is Newtonian mechanics, used in the following way. The gas is assumed to consist of a large number, say 10^{23}, of hard spheres that undergo elastic collisions when they meet. This model formalizes the question on whether Newtonian mechanics can be scaled up to describe macroscopic matter. The study of this problem caused a great deal of angst in the 19th century. First, the existence of atoms had not yet been established, so the microscopic assumptions were a matter of doubt. Worse yet, that the macroscopic behavior is irreversible ('the arrow of time') is familiar to us from our everyday experience. But Newtonian mechanics is invariant under time reversal, so this problem carries subtle paradoxes.

A similarly fundamental example, which is particularly relevant to pattern theory is the Ising model. This model was introduced to explain how the magnetization of permanent magnets (such as in a compass), can be obtained from a microscopic model of many small spins which balance energetic terms that favor alignment of neighboring spins, with thermal effects that cause them to fluctuate. For example, for a 1-D magnet, we have the following: $\mathcal{S} = \{$ strings of +,- of length $N\}$ and $V(x) = a\sum_{i=1}^{N} x_i + b\sum_{i=1}^{N-1}(x_i - x_{i-1})^2$ where a and b are fixed parameters. The study of this model has been very fruitful in physics: in particular, it led to the development of Markov Chain Monte Carlo (MCMC), a fundamental simulation technique that we will study in detail.

As noted by Geman and Geman [8], by changing the background energy, essentially the same model can be used to study the (unphysical!) problem of image segmentation (see Section 7.3). This connection raises a natural question: why should physical heuristics work on problems that seem to have little to do with physics? The answer lies in viewing information theory as a foundation for inference. This approach, which is due to Jaynes, inverts the historical development of ideas by viewing information theory as the primary model for developing statistical mechanics, rather than viewing statistical mechanics as the foundation for information theory. The simplicity of this line of reasoning is best illustrated by the following Bayesian derivation of the Gibbs distribution.

We consider the following thought experiment: Suppose X is a discrete random variable taking m values with probability $\{p_1, \ldots, p_m\}$. We do *not* assume that we know these probabilities (thus X is 'hidden' from us). However, we suppose that we have observations of some function $f(X)$, taking values f_i, for each state x_i of X, so that we may assume that $E(f(X)) = \theta$ is known.

The question now is the following: What is the best guess $\mathbb{E}g(X)$ where g is some other function? This is essentially the same as asking for our best guess of the pmf $(p_1, \ldots, p_m)$ of X. Naively, this is an ill-posed problem, in the sense that we may have many answers depending on what our notion of 'best' means. This is because there are m variables $(p_1, \ldots, p_m)$ subject to the constraints $\sum p_i = 1$ and $\sum f_i p_i = \alpha$, so clearly there are many answers to this question.

However, information theory provides a principled answer to this question. Shannon postulates that given a pmf $(p_1, \ldots, p_n)$ for a random variable X taking

m values, its entropy is

$$H(p) = -\sum_{i=1}^{m} p_i \ln p_i.$$

The entropy of a random variable describes its uncertainty. In particular, the notion of an optimal code tells us that entropy describes the optimal search procedure to determine the value of X through a series of yes/no questions. This is one of the basic interpretations of entropy in information theory. The relevance of this interpretation here is that it tells us that the most *unbiased* estimate of the pmf of X is obtained by maximizing the entropy of X subject to the constraint:

$$\mathbb{E}f(X) = \sum_{i=1}^{n} p_i f_i = \theta,$$

where θ is given.

This use of information theory converts an ill-posed problem to a standard constrained maximization problem:

$$p = \text{argmax}_{p|\sum p_i f_i = \theta, \sum p_i = 1} H(p)$$

where we have used p to denote the pmf $(p_1, \ldots, p_m)$. When we solve this maximization problem, we find that

$$p_i = \frac{e^{-\beta f_i}}{Z_\beta}.$$

Thus, the above maximization of entropy principle recovers the Gibbs distribution, without requiring any form of physical reasoning. This observation explains the ubiquity of the Gibbs distribution in problems of inference.

Now let's check the calculation. Let us introduce Lagrange multipliers α and β and consider the unconstrained maximization problem for the function

$$J(p) = H(p) - \beta \left(\sum_i f_i p_i - \theta \right) - \alpha \left(\sum p_i - 1 \right).$$

We now use the maximization criterion from calculus: we require

$$\frac{\partial J}{\partial p_k} = 0$$

for $k = 1, \ldots, m$. Differentiating the above expression we find that

$$0 = -(1 + \ln p_k) - \alpha - \beta f_k$$

which may be solved to yield

$$p_k = \frac{1}{Z_\beta} e^{-\beta f_k}, \quad 1 \leq k \leq m.$$

where we have written Z_β for $e^{1+\alpha}$. (This is just a convenient relabeling of the Lagrange multiplier, since Z_β is the normalizing factor that ensures $\sum p_i = 1$.)

Thus, we have m equations for the unknown variables $p_1, \ldots, p_n$. We also have the equation

$$\sum f_i p_i = \theta,$$

which determines the Lagrange multiplier β. We use our formula for p to obtain

$$\frac{1}{Z_\beta} \sum f_i e^{-\beta f_i} = \theta.$$

This shows that the inverse temperature β is the Lagrange multiplier corresponding to the given constraint on $\mathbb{E}(f(X))$. Finally, let's observe that the above equation may also be written as

$$-\frac{\partial}{\partial \beta} \ln Z_\beta = \theta.$$

This equation reflects a common theme in statistical mechanics. Since observables may be obtained by differentiating the partitition function Z_β with respect to the parameters of the model, the main task is to determine the partition function.

1.5 Bayesian inference and decoding text

In this section, we combine the ideas from the previous sections to develop a Bayesian method to decode a scrambled text.

The basic ideas of Bayesian inference are as follows. First, we assume that the world is stochastic and that inference may be modeled probabilistically using Bayes rule for conditional probabilities. Our task is to determine the values of a random variable, $\mathcal{S}$, that is hidden from us, based on observations O. In the example with scrambled text, the hidden state will be a permutation, while the observation is a given string of scrambled text. The underlying probabilistic model is that the 'world' consists of a true language that has been scrambled, letter by letter, by a random permutation.

The mathematical formalism of Bayes rule is simple. If both S and O are random events or random variables on the same probability space, we recall that conditional probabilities are defined by

$$\mathbb{P}(S \cap O) = \mathbb{P}(S|O)\mathbb{P}(O) = \mathbb{P}(O|S)\mathbb{P}(S)$$

Thus, we may rewrite this equation in the form

$$\underbrace{\mathbb{P}(S|O)}_{\text{posterior}} = \frac{\mathbb{P}(O|S)\,\mathbb{P}(S)}{\mathbb{P}(O)} \propto \underbrace{\mathbb{P}(O|S)\,\mathbb{P}(S)}_{\text{prior}}.$$

The terms prior and posterior reflect the use of Bayes formula in modeling. The right hand side of the formula reflects our assumption that we have a

probabilistic model, called the prior distribution, which generates the hidden variable and the observed signals, conditional on the hidden variable. The left hand side says that given an observation, we can use Bayes rules to update our prior for the hidden variable to obtain a new distribution called the posterior.

A fundamental idea in Bayesian inference is to find the mode of the posterior. More precisely, if what we have is our probabilistic model for interpreting the world, then our best guess for the hidden state is the mode of the posterior

$$S_* = \text{argmax}_S \mathbb{P}(S|O) = \text{argmax}_S \mathbb{P}(O|S)\,\mathbb{P}(S). \tag{1.5.1}$$

This simple idea acquires its richness from the diversity of its applications. Thus, let us flesh it out by applying it to decoding scrambled text. In order to get started we need a good probabilistic model for text as well as for the process of scrambling.

We model the scrambler as a permutation $\sigma \in S_{|A|}$, a permutation on $|A|$ symbols. The scrambler acts on the text letter by letter. That is, if $a_1 a_2 \dots a_n$ is a string of true language, the scrambler converts this to a string $b_1, b_2, \dots, b_n$ where $b_i = \sigma(a_i)$. Since the permutation acts letter by letter, we also have

$$\mathbb{P}(O = b_1 b_2 \dots b_n | S = \sigma) = \mathbb{P}(\text{scrambler input} = \sigma^{-1}(b_1) \dots \sigma^{-1}(b_n)).$$

This expression holds for all the probabilistic models of text (true language, n-gram approximations and word-based Markov chains) studied in Section 1.2. For simplicity, we will explain the process of inference using the digram model; other models may be explored in a similar way.

In the absence of any other information on the scrambling mechanism, the most natural choice for a prior on $\mathcal{S}$ is to assume that it is chosen uniformly on the permutation group. This ensures that $P(S)$ is independent of S. This choice of uniform measure should be intuitive, but note also that it is the maximizer of entropy.

Combining the above ideas, we see that our best guess for the unknown permutation σ_* is

$$\sigma_* = \text{argmax}_{\sigma \in S_{|A|}} L(\sigma), \tag{1.5.2}$$

where we have defined the *likelihood* or *plausibility* function

$$L(\sigma) = \mathbb{P}_{true}(\sigma^{-1}(b_1)) \prod_{j=1}^{n-1} Q^{(2)}(\sigma^{-1}(b_j), \sigma^{-1}(b_{j+1})). \tag{1.5.3}$$

At this stage, we have obtained a clear mathematical formulation for the problem of decoding a scrambled text. Now we need to see whether this idea works! The catch is the following: in order to find the argmax, we must maximize the likelihood function on a large space. This is computationally intractable without a fundamental randomized algorithm: Markov Chain Monte Carlo (MCMC). This is such an important subject that the next chapter is devoted to understanding this algorithm from different points of view.

Chapter 2

The Markov Chain Monte Carlo method

2.1 Sampling from a Gibbs distribution

The Metropolis scheme is a randomized algorithm for sampling from a Gibbs distribution. Let us briefly describe the sampling problem. We then connect it to the problem of decoding scrambled text. The mathematics under the hood is discussed in the next section.

Let us assume that we are given a finite state space $\mathcal{S}$, an energy function $E : \mathcal{S} \to \mathbb{R}$, and an inverse temperature β. Our task is to sample from the Gibbs distribution

$$p_\beta(x) = \frac{e^{-\beta E(x)}}{Z_\beta}, \quad x \in \mathcal{S}. \tag{2.1.1}$$

As discussed in Section 1.4 terms such as the Gibbs distribution, energy landscape and temperature have precise meaning even in the absence of an underlying physical model (such as our application to scrambled text).

Naively, sampling means that we'd like to generate an iid sequence, $\{X_t\}_{t=1}^{\infty}$, distributed according to p_β. The average of any observable $f(X)$ may then be approximated by the empirical mean

$$\mathbb{E}(f(X)) = \sum_{x \in \mathcal{S}} f(x) p_\beta(x) \approx \frac{1}{T} \sum_{t=1}^{T} f(X_t). \tag{2.1.2}$$

When $\mathcal{S}$ is large, generating iid sequences is prohibitively expensive. Instead, the Metropolis scheme generates a Markov chain $\{X_t\}_{t=1}^{\infty}$ which has p_β as an equilibrium measure. Provided the underlying Markov chain is ergodic, we may still use (2.1.2) to compute the expected value of an observable. The number of steps T required to obtain a good estimate depends on how fast the Markov chain decorrelates (i.e. the number of steps t needed so that X_1 and X_t are almost independent).

Let us illustrate these ideas with our model of scrambled text. The state space $\mathcal{S}$ is the set of permutations on 27 symbols, denoted S_{27}. In accordance with standard notation for permutations, we use σ instead of x to denote a point in $\mathcal{S}$. Let us define the energy function

$$E(\sigma) = -\ln L(\sigma), \tag{2.1.3}$$

and use (1.5.3) to obtain

$$E(\sigma) = -\ln \mathbb{P}_{true}(\sigma^{-1}(b_1)) - \sum_{j=1}^{n-1} \ln Q^{(2)}\left(\sigma^{-1}(b_j), \sigma^{-1}(b_{j+1})\right). \tag{2.1.4}$$

Rather than maximizing L, we must now minimize E, since

$$\sigma_* = \text{argmax}_\sigma L(\sigma) = \text{argmin}_\sigma E(\sigma).$$

The energy function E is a sum of nearest-neighbor terms. This structure arises because of our assumption that text is modeled by a Markov process, not because there is any 'physics' underlying text (physical models such as the Ising model have a similar structure; the common thread is the assumption of local spatial dependence for the stochastic process X_t).

2.2 The Metropolis scheme

The Metropolis scheme relies on biasing an easily generated Markov chain, such as a symmetric random walk, so that the new equilibrium distribution is p_β. Assume given a Markov chain on $\mathcal{S}$ whose equilibrium distribution is uniform. Let us first describe the scheme as a numerical recipe for decoding scrambled text, so that its description is concrete:

1. Estimate $\mathbb{P}_{true}(a)$ for $a \in A$ and $Q^2(a_1, a_2)$ for $a_1, a_2 \in A$ by mining a sufficiently large text such as *War and Peace*.

2. Generate a symmetric random walk in S_{27} using uniform random transpositions and bias it using the following acceptance/rejection rule. Consider a proposed move $\sigma \mapsto \tau$, where σ and τ are related through transposition. Compute $\Delta E = E(\tau) - E(\sigma)$. If $\Delta E < 0$ then accept the move. If $\Delta E \geq 0$ accept with probability $\exp(-\beta \Delta E)$ (where $\beta > 0$ is fixed, say $\beta = 1$ to be concrete).

The above scheme provides a Markov chain whose equilibrium distribution is $e^{-\beta E(\sigma)}/Z_\beta$, where $E(\sigma)$ is defined in equation (2.1.3). The scheme pushes σ 'down' the energy landscape, while allowing it a small probability of escaping local minima. In practice, given a sufficiently long string (i.e. when n is large enough), the energy landscape is sharply concentrated around its minima at σ_*. Thus, once the Markov chain hits the minima, it stays there forever. This is why a *randomized* scheme can be used to solve a *deterministic* minimization problem.

2.3 Why does the Metropolis scheme work?

2.3.1 The biased scheme and its transition matrix

Let us now describe the mathematical structure of the Metropolis scheme more carefully. We assume given a state space $\mathcal{S}$, an energy function $E : \mathcal{S} \to \mathbb{R}$ and an inverse temperature $\beta > 0$. The regime of interest is when $\mathcal{S}$ is so large that it is impossible to evaluate the partition function

$$Z_\beta = \sum_{x \in \mathcal{S}} e^{-\beta E(x)},$$

and thus the Gibbs distribution.

The Metropolis scheme relies on the existence of an easily sampled Markov chain. In the first applications of the method, the underlying model was the Ising model and the Markov chain was a random walk in the space of spins. Configurations of spins are said to be neighbors when they are related by spin flips as shown below.

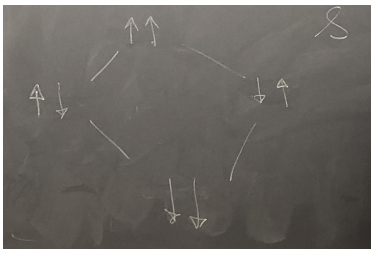

Figure 2.3.1: A graph of spin states joined by single spin flips

It is not necessary to assume such a specific structure on $\mathcal{S}$. Every transition matrix Q that is symmetric, i.e. $Q(x, y) = Q(y, x)$ for all x,$y \in \mathcal{S}$ has a uniform equilibrium distribution and may be used as a source of randomness for a Metropolis scheme.

This may be seen as follows. The equilibrium distribution π for Q satisfies the equation $\pi(y) = \sum_x \pi(x)Q(x, y)$. We must show that $\pi(x) = \frac{1}{|\mathcal{S}|}$ (i.e. π is uniform). Observe that

$$\sum_y Q(x, y) = 1,$$

since probabilities have to sum up to one. But since $Q(x, y) = Q(y, x)$ we also have

$$\sum_y Q(y, x) = 1. \tag{2.3.1}$$

Relabel indices to see that $\sum_x Q(x, y) = 1$ for every y. Thus, if $\tilde{\pi}(y) = 1$ for each y, we may rewrite (2.3.1) as

$$\sum_x \tilde{\pi}(x)Q(x, y) = \tilde{\pi}(y), \quad x, y \in \mathcal{S}.$$

Given a Markov chain generated by Q, we construct a new Markov chain with equilibrium p_β by biasing this chain as follows. Define the transition matrix

$$B(x,y) = \begin{cases} Q(x,y), \text{ if } p_\beta(x) < p_\beta(y) \\ \frac{p_\beta(y)}{p_\beta(x)} Q(x,y), \text{ if } p_\beta(x) \geq p_\beta(y) \\ 1 - \sum_{x'} B(x,x') \text{ when } x = y. \end{cases} \tag{2.3.2}$$

where p_β is the desired equilibrium distribution.

2.3.2 Detailed balance

The above biasing procedure looks 'asymmetric', but this is misleading. The matrix B is naturally suited to p_β because of the following identity, which is called *detailed balance*:

$$p_\beta(x)B(x,y) = p_\beta(y)B(y,x), \quad x,y \in \mathcal{S}. \tag{2.3.3}$$

This identity is easily established. Without loss of generality assume that $p_\beta(x) \geq p_\beta(y)$. Then

$$B(x,y) = \frac{p_\beta(y)}{p_\beta(x)} Q(x,y), \quad \text{and} \quad B(y,x) = Q(x,y),$$

which implies equation (2.3.3).

Detailed balance implies that the equilibrium of B is p_β. Indeed, summing over x in the right hand side of equation (2.3.3) gives

$$p_\beta(y) \sum_x B(y,x) = \sum_x p_\beta(x) B(x,y),$$

which may be rewritten as the vector equation,

$$p_\beta = p_\beta B.$$

The success of MCMC relies on the interplay between theoretical guarantees and practical implementation that originate in the condition of detailed balance. We consider some of these below.

2.3.3 Local moves

We wanted a Markov chain to sample from $p_\beta(x)$, and we obtain it by biasing $Q(x,y)$ by $\frac{p_\beta(y)}{p_\beta(x)}$. What makes the computation of p_β intractable is the sum over states in (2.3.1). However, the ratio

$$\frac{p_\beta(x)}{p_\beta(y)} = \frac{e^{-\beta E(x)}}{Z_\beta} \frac{Z_\beta}{e^{-\beta E(y)}} = e^{-\beta(E(x)-E(y))},$$

depends only on the difference in energy between two states and this is easily computed. In particular, for the Ising model, as well as the digram model of text,

the change in energy between neighboring states x and y requires an update of only a small number of terms. Thus, it is is inexpensive to compute.

This property reflect a typical strength and design principle for Metropolis schemes: they must rely on updates ΔE that are cheap to compute. On the other hand, this property also reflects a weakness of Metropolis schemes. If $\beta\Delta E$ is large, say at a local minimum of E, then many moves are rejected before the state escapes from the local minimum. In such cases, the Metropolis scheme must be replaced by modifications of it such as simulated annealing.

2.3.4 B generates a reversible Markov chain

While it is natural to think of increasing t as representing the arrow of time, the Markov property requires only that the future and past be independent, conditional on the present. Thus, given a transition matrix for a stationary Markov chain, we may always construct a time reversed Markov chain as follows. Since

$$Q(x,y) = \frac{\mathbb{P}(X_1 = x, X_2 = y)}{\mathbb{P}(X_1 = x)},$$

by conditioning on X_2 instead of X_1 we obtain the reversed chain with transition matrix

$$R(y,x) := \frac{\mathbb{P}(X_2 = y, X_1 = x)}{\mathbb{P}(X_2 = y)} = \frac{\mathbb{P}(X_1 = x)}{\mathbb{P}(X_2 = y)} Q(x,y).$$

The above equation may be rewritten as the identity

$$R(y,x)\pi(y) = Q(x,y)\pi(x), \tag{2.3.4}$$

where π is the equilibrium for Q, and thus R. (Note that the chain is stationary).

In general, the time-reversed Markov chain generated by R is not the same as the forward chain generated by Q. However, these are identical for the Metropolis scheme. Indeed, replacing $Q(x,y$ with $B(x,y)$ and $\pi(x)$ with $p_\beta(x)$ in equation (2.3.4), we find that the reversed chain satisfies

$$R(y,x) = \frac{\mathbb{P}(X_1 = x, X_2 = y)}{\mathbb{P}(X_2 = y)} = \frac{\mathbb{P}(X_1 = x, X_2 = y)}{p_\beta(y)}.$$

Thus, we may use equations (2.3.2) and (2.3.4) to see that

$$R(y,x) = B(y,x), \quad x, y \in \mathcal{S},$$

so that the chains are equivalent.

This calculation reflects the fact that detailed balance is a condition of local equilibrium between every pair of states in the Markov chain. This principle plays an important role in the description of chemical reaction networks. In this context, each state of the chain is a chemical compound, and the transition rates $B(x,y)$ and $B(y,x)$ reflect the rates of forward and backward equations. When the system is in equilibrium, each set of forward and backward reactions must be balanced.

2.3.5 B is a self-adjoint operator on $L^2_{p_\beta}$

A set of positive weights $m = (m_1, m_2, \ldots, m_N)$ may be used to define an inner product on $\mathbb{R}^N$ as follows:

$$\langle v, w\rangle_m := \sum_{i=1}^{N} m_i\, v_i w_i. \tag{2.3.5}$$

The *adjoint*, $A^\dagger$, of a linear operator $A : \mathbb{R}^N \to \mathbb{R}^N$ with respect to this inner-product is defined by

$$\langle A^\dagger v, w\rangle_m = \langle v, Aw\rangle_m, \quad v, w \in \mathbb{R}^n. \tag{2.3.6}$$

Finally, we say that an operator is self-adjoint when $A = A^\dagger$.

Now let us apply this idea to $\mathbb{R}^{|\mathcal{S}|}$ choosing $p_\beta(x)$ as the weight. This weight determines the vector space $L^2_{p_\beta}$, which is simply $\mathbb{R}^{|\mathcal{S}|}$ equipped with the inner product

$$\langle v, w\rangle_{p_\beta} := \sum_{x\in\mathcal{S}} p_\beta(x)\, v(x) w(x). \tag{2.3.7}$$

The adjoint of a linear operator A on $L^2_{p_\beta}$ is given by

$$A^\dagger(x, y) = \frac{p_\beta(x)}{p_\beta(y)} A(x, y), \quad x, y \in \mathcal{S}.$$

In particular, detailed balance implies that $B = B^\dagger$. Thus, B is self-adjoint.

Self-adjoint operators are of fundamental importance in mathematics and physics. They always have real eigenvaues and admit the following *spectral decomposition* which is a general form of diagonalization familiar to you from linear algebra. The spectral theorem states that every self-adjoint operator A may be written in the form

$$A(x, y) = \pi(y) \sum_{i=1}^{|\mathcal{S}|} \alpha_i \psi_i(x) \psi_i(y), \tag{2.3.8}$$

where the α_i and ψ_i are the eigenvalues and eigenvectors of A. It then follows that all powers of A may be expressed as

$$(A^n)(x, y) = \pi(y) \sum_{i=1}^{|\mathcal{S}|} \alpha_i^n \psi_i(x) \psi_i(y), \quad x, y \in \mathcal{S}.$$

We may also write equation (2.3.8) in the form

$$A = \sum_i \alpha_i \psi_i \psi_i^\dagger,$$

where given a column vector ψ with entries $\psi(x)$, $x \in \mathcal{S}$, its adjoint is a column vector$\psi^\dagger$ with entries $\pi(y)\psi(y)$, $y \in \mathcal{S}$. Then the rank-one operators $P_i := \psi\psi^\dagger$ are orthogonal projections in $L^2_{p_\beta}$ onto the space spanned by the vector ψ. The spectral theorem expresses A as a sum of projections onto the eigenspaces ψ_i, each weighted by the eigenvalue α_i.

2.3.6 Convergence of Markov chains

Let us now return to Markov chains, contrasting the general theory of convergence for Markov chains, with the theory for Markov chains with a transition matrix that is self-adjoint.

Suppose given a transition matrix P on a state space of size N which determines an ergodic Markov chain with probability distribution π. We would like to use the forward equation $\pi_{n+1} = \pi_n P$ to prove that $\lim_{n\to\infty} \pi_n = \pi$. We write $v_n = \pi_n - \pi$ for the difference and use $\pi = \pi P$ to obtain the equation

$$v_{n+1} = v_n P = v_1 P^n,$$

by induction on n. Thus, if $P = U\Lambda U^{-1}$, where Λ is the diagonal matrix of eigenvalues and U is a matrix of eigenvectors, we find that

$$v_{n+1} = v_1 U \Lambda^n U^{-1}.$$

The asymptotics as $n \to \infty$ depend only on Λ. The entries of this matrix are the eigenvalues of P, i.e. the roots of the characteristic polynomial $\det(\lambda I - P) = 0$. In general, these roots are complex numbers and about all that we can say for an arbitrary transition matrix P is that all eigenvalues of P must all lie strictly within the unit disk in the complex plane, with a unique eigenvalue on the boundary of the disk at $\lambda = 1$ (this eigenvalue corresponds to the equilibrium). These general assertions follow from the Perron-Frobenius theorem (you *don't* need to know this!). Since all eigenvalues except the first lie strictly within the unit disk, as $n \to \infty$,

$$\Lambda^n = \begin{pmatrix} 1^n & 0 & \dots \\ 0 & \lambda_2^n & \dots \\ \vdots & \ddots & \lambda_N \end{pmatrix} \to \begin{pmatrix} 1 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & \ddots & 0 \end{pmatrix}.$$

It is in this sense that we may use linear algebra to understand the convergence of Markov chains.

The problem with the general theory is that while it tells us that the chain converges to equilibrium, what we care about in practice is the *rate* of convergence, i.e. how fast all diagonal terms except the first converge to zero.

In general, we can't say very much. However, for a reversible Markov chain, such as the Metropolis scheme, we can say a lot more, using the fact that B is self-adjoint in $L^2_{p_\beta}$ and that it admits a spectral decomposition (2.3.8). In this case, we may order the eigenvalues, $1 = \lambda_1 > \lambda_2 \geq \lambda_2 \dots$, and we see that the rate of convergence of the Markov chain is $O(|\lambda_2 - \lambda_1|^n)$ as $n \to \infty$. Various techniques exist to estimate this spectral gap based on the geometry of the underlying state space (see [5] for examples on shuffling).

Chapter 3

From text to machine translation

In this section we first review some central ideas in information theory. These include the interpretation of entropy as the average depth of the optimal search to find a random variable, the entropy rate of a stationary process, the convergence of processes using relative entropy, and the mutual information. These ideas are then illustrated by applications to Markov chain models of text. These include the convergence of the n-gram approximations to true language as $n \to \infty$ and the use of mutual information to design a parsing scheme to detect word boundaries. The idea of a Hidden Markov model (HMM) is introduced in the last section through an application to machine translation.

3.1 Entropy and entropy rate

The entropy of a random variable X taking values in a finite alphabet A is

$$H(X) = -\sum_{x \in A} p(x) \log p(x).$$

The entropy $H(X_1, \ldots, X_n)$ of a finite sequence of random variables is also defined by the above formula, since a finite sequence taking values in A may be lumped into a single random variable $Y := (X_1, \ldots, X_n)$ taking values in A^n. However, in order to have a meaningful limit as $n \to \infty$, it is necessary to normalize the entropy by the size of the sequence.

Theorem 11 (Entropy rate)**.** *Suppose $\{X_k\}_{k=1}^{\infty}$ is a stationary stochastic process taking values in a finite alphabet. Then the following limits exist and are equal.*

$$\lim_{n \to \infty} \frac{H(X_1, \ldots, X_n)}{n} = \lim_{n \to \infty} H(X_n | X_{n-1}, \ldots, X_1) := F.$$

The limit F is called the entropy rate.

Existence of the entropy rate. Let $F_n = H(X_n|X_{n-1}, \dots, X_1)$. Since conditioning reduces entropy [1] and the sequence is stationary,

$$\begin{aligned} F_{n+1} = H(X_{n+1}|X_n, \dots, X_1) &\leq H(X_{n+1}|X_n, \dots X_2) \\ &= H(X_n|X_{n-1}, \dots, X_1) = F_n. \end{aligned}$$

Thus, $\{F_n\}_{n=1}^{\infty}$ is a decreasing sequence that is bounded below by zero. This shows that $F := \lim_n F_n$ exists.

The equivalence of the two limits in the definition of the entropy rate is seen as follows. We use the entropy chain rule to write

$$H(X_1, \dots, X_n) = H(X_1) + H(X_2|X_1) + H(X_3|X_1, X_2) = F_1 + \cdots + F_n.$$

Since $F_n \to F$, we see that for large n, we're adding a series of terms that are almost constant. We formalize this idea as follows. Let $\epsilon > 0$ and choose $k_*(\epsilon)$ so that $|F - F_k| < \epsilon$ for $k > k_*(\epsilon)$. We then split the sum into $n - k_*$ terms that are ϵ-close to the limit F and a sum of k_* initial terms as follows

$$\frac{H(X_1, \dots, X_n)}{n} = \frac{F_1 + \cdots + F_{k_*}}{n} + \frac{F_{k_*+1} + \cdots + F_n}{n}.$$

Since F_k is a positive decreasing sequence, the first term is bounded above and below by

$$0 \leq \frac{F_1 + \cdots + F_{k_*}}{n} \leq F_1 \frac{k_*}{n}.$$

Similarly, the second term is bounded above and below by

$$F\left(1 - \frac{k_*}{n}\right) \leq \frac{F_{k_*+1} + \cdots + F_n}{n} \leq (F + \epsilon)\left(1 - \frac{k_*}{n}\right),$$

using the fact that $F \leq F_{k_*} < F + \epsilon$. □

Example 12. Since a stationary Markov chain is also a stationary stochastic process, we may use the above formula to compute its entropy rate. To this end, suppose $\{X_k\}_{k=1}^{\infty}$ is a Markov chain with transition matrix $Q(x, y)$ and equilibrium pmf $\pi(x)$. We specialize the entropy rate formula as follows.

$$\begin{aligned} F = \lim_n H(X_n|X_{n-1}, \dots, X_1) &\underbrace{=}_{\text{Markov property}} \lim_n H(X_n|X_{n-1}) \underbrace{=}_{\text{stationarity}} H(X_2|X_1) \\ &= \sum_{x \in A} \mathbb{P}(X_1 = x) H(X_2|X_1 = x) = -\sum_{X \in A} \pi(x) \sum_{y \in A} Q(x, y) \log Q(x, y). \end{aligned}$$

3.2 Entropy and data compression

3.2.1 Entropy as coding length

One of the fundamental interpretations of entropy is that it corresponds to the length of an *optimal code*. We first review this idea for binary prefix codes.

[1] The reader unfamiliar with these definitions will find complete proofs in [?].

Suppose X is a random variable that takes values in a finite alphabet $A = \{1, \ldots, m\}$ with probabilities $(p_1, \ldots, p_m)$. A binary code is a map $C : A \to \{0,1\}^*$ where $\{0,1\}^*$ denotes the space of finite binary strings. We say that C is a prefix code if no codeword $C(x)$ is a prefix of another codeword $C(y)$ for all pairs $x, y \in A$. A prefix code is in one-to-one correspondence with a binary tree whose leaves correspond to the codewords $C(x)$, $x \in A$. This equivalence may be seen by drawing a binary tree and labeling the vertices in lexicographical order beginning with the root. The labels of the leaves are the codewords. In what follows, we only consider prefix codes.

Given a code C, the length of the codeword for x, denoted $l(x)$, is the number of terms in the binary string $C(x)$. Since every binary prefix code is in correspondence with a binary tree, $l(x)$ may also be interpreted as the length of the path joining the root to the leaf $C(x)$.

The fundamental bounds relating entropy to data compression are as follows. First, every binary prefix code for X is bounded below by the entropy:

$$H(X) \leq \mathbb{E}(l(X)) := \sum_x p(x) l(x). \tag{3.2.1}$$

A code is optimal if it achieves the minimum of $\mathbb{E}(l(X))$ over the space of codes. When C is optimal, the bound (3.2.1) has the matching upper bound

$$\mathbb{E}(l_*(X)) < H(X) + 1, \tag{3.2.2}$$

where we have denoted the optimal code by C_* and its length by l_* to distinguish it from the lower bound (3.2.1). An optimal code may be computed using Huffman's algorithm as explained below.

The dangling 1 in equation (3.2.2) can be improved by considering a block code, replacing the idea of entropy with entropy rate, and considering the code-length per symbol. More precisely, we consider a sequence $(X_1, \ldots, X_n)$, code it with an optimal code C_n, and use $l_n = l(x_1, \ldots, x_n)/n$ to denote the length per symbol of the codewords. We then use equations (3.2.1) and equations (3.2.2) to obtain the matching upper and lower bounds

$$\frac{1}{n} H(X_1, \ldots, X_n) \leq \mathbb{E}(l_n) < \frac{1}{n} H(X_1, \ldots, X_n) + \frac{1}{n}. \tag{3.2.3}$$

These bounds show that the entropy rate of a stationary stochastic process is the length per symbol of an optimal code.

3.2.2 Huffman's algorithm

Given a pmf $(p_1, \ldots, p_m)$, an optimal code may be computed in $O(m \log m)$ steps using Huffman's algorithm. This is a greedy algorithm that recursively builds a binary tree whose codewords are leaves. The algorithm has m steps and is indexed downward beginning with m. At the beginning of the k-th step assume given a pmf $q^{(k)} = (q_1, \ldots, q_k)$ and a forest $\tau^{(k)} := (\tau_1, \ldots, \tau_k)$ consisting of a collection of binary trees τ_j, $1 \leq j \leq k$. The algorithm is initialized at $k = m$

with the given pmf $p = (p_1, \ldots, p_m)$ and a forest consisting of m trees each with a single vertex. The $k-1$-th iterate is then obtained as follows:

1. Rerank the pmf $q^{(k)}$ in decreasing order. More precisely, choose a permutation $\sigma \in S_k$ such that $q_{\sigma_1} \geq q_{\sigma_2} \geq \ldots q_{\sigma_k}$. Let $s^{(k)}$ denote the reordered pmf, with $s_j = q_{\sigma_j}$.

2. Merge the two smallest probabilities s_{k-1} and s_k to obtain $q^{(k-1)} = (s_1, s_2, \ldots, s_{k-1} + s_k)$.

3. Use the permutation σ to relabel the trees within the forest, defining a new forest $\tilde{\tau} = (\tau_{\sigma_1}, \tau_{\sigma_2}, \ldots, \tau_{\sigma_k})$ with k subtrees.

4. Merge the trees $\tau_{\sigma_{k-1}}$ and τ_{σ_k} associated to the smallest probabilities s_{k-1} and s_k by adding a new root and two edges labeled with 0 and 1 respectively that connect the root to $\tau_{\sigma_{k-1}}$ and τ_{σ_k}. Define $\tau^{(k-1)}$ to be the resulting modification of the forest $\tilde{\tau}$.

3.2.3 Typical sequences and the AEP

The entropy rate also quantifies the effective size of our probability space. Fix an integer N and let Ω_N denote the space of sequences $\{a \in A^N \,|a = (a_1, \ldots, a_N)\}$. Given a stationary stochastic process $\{X_k\}_{k=1}^{\infty}$ let $\mathbb{P}_N$ denote the joint pmf of $(X_1, \ldots, X_N)$. The size of Ω_N is $|A|^N = 2^{N \log |A|}$. One of the fundamental interpretations of entropy is that it 'thins down' the space of all sequences to a set of 'typical sequences', which is of size 2^{NF}, where F denotes the entropy rate of $\{X_k\}_{k=1}^{\infty}$. As in Section 3.2.1, this shows that entropy quantifies data compression by focusing our attention on the 'set of sequences that matter'.

The main idea is as follows. We fix an arbitrary error threshold $\epsilon > 0$ and define the typical set $A_{\epsilon,N}$ to consist of sequences $a \in \Omega_N$ such that

$$2^{-N(F+\epsilon)} \leq \mathbb{P}_N(a_1, \ldots, a_N) \leq 2^{-N(F-\epsilon)}.$$

One may then use this definition and the weak law of large numbers for stationary processes to show that for every $\epsilon > 0$, there is an N_* such that for $N > N_*$ the probability and the size of the typical set satisfy the bounds

$$\mathbb{P}_N(A_{\epsilon,N}) \geq 1 - \epsilon, \quad 2^{N(F-\epsilon)} \leq |A_{\epsilon,N}| \leq 2^{N(F+\epsilon)}.$$

3.3 The entropy of English

3.3.1 Samples of random text

The entropy rate allows us to quantify the balance between freedom and redundancy in a stochastic process. Let us illustrate this idea with the following samples of randomly generated text, each of which is taken from Shannon's work [14].

1. XFOML RXKHRJFFJUJ ZLPWCFWKCYJ FFJEYVKCQSGHYD QPAAMKBZAACIBZHJQD

2. OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA OOBTTVA NAH BRL

3. ON IE ANTSOUTINYS ARE T INCTORE ST BE SDEAMY ACHIN D ILONASIVE TUCOOWE AT TEASONARE FUSO TIZIN ANDY TOBE SEACE CTISBE

4. IN NO IST LAT WHEY CRACTICT FROURE BIRS GROCID PONDENOME OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE

5. REPRESENTING AND SPEEDILY IS AN GOOD APT OR COME CAN DIFFERENT NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME TO FURNISHES THE LINE MESSAGE HAD BE THESE

6. THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR AN UNEXPECTED.

These strings are defined through the following stochastic processes respectively.

1. The letters form an iid sequence from the alphabet $\{A, B, \ldots, Z, \_\}$ of 27 symbols. Each of these symbols is chosen with equal probability.

2. An iid sequence from the same alphabet, but with the probabilities of the letters chosen according to their true frequencies in text. Thus, commonly used letters such as E, T and O appear more frequently than in (1).

3. A 2-gram stationary Markov chain on $\{A, B, \ldots, Z, \_\}$ with transition rates $Q^{(2)}$ determined by true language.

4. A 3-gram stationary Markov chain on $\{A, B, \ldots, Z, \_\}$ with transition rates $Q^{(3)}$ determined by true language.

5. The underlying alphabet is now the lexicon of the English language (i.e. the set of words in English). The text is an iid sequence of words, where the words are chosen according to the frequencies at which they appear. The spaces are added only to signify gaps between words.

6. A Markov chain on the lexicon which is the analogue of the 2-gram approximation in (3), replacing the alphabet of 27 letters with the lexicon.

These samples illustrate several ideas. Examples (1)–(4) show that the stochastic processes mimic true language with greater fidelity as the underlying transition probabilities use more information on joint distributions of the underlying

true language. Examples (5) and (6) illustrate a deeper principle: true language carries a hierarchy of structures along with rules for binding these structures. Examples (5) and (6) illustrate this principle with the simplest hierarchy – the difference between letters and words. Even within our naive formalism for language, this example shows that a true language can be modeled on different probability spaces – sequences of letters or sequences of words. In order to determine whether (5) is a better approximation than (6), we must formalize the idea of mapping one space to the other. Mapping a word to a sequence of letters is spelling; mapping a string of letters to a string of words is a type of parsing.

3.3.2 Markov chain models cannot be grammatical

The efficacy of Shannon's model, both in generating the above sequences and decoding scrambled text, should also give us pause. Even though the output of (6) is more natural than any of the previous approximations, it clearly doesn't 'feel right' because of the unusual construction of the sentence, as well as the semantic flaws. How do we model this?

The underlying rules that determines a language constitute its *grammar*. Native speakers of a language effortlessly perceive a violation of grammar as we do when reading a sentence such as (6). Such violations may be modeled within a Bayesian framework, since the notion that 'something doesn't feel right' merely means that an observation is very unlikely within a probabilistic model. But this also makes explicit the fact that in order to build a probabilistic model of language that is more realistic than Shannon's approximations, we need a deeper mathematical model of grammar.

The modern study of grammar was revolutionized a few years after Shannon's work by Chomsky [**?**]. In contrast with earlier work on linguistics that focused on classifying the grammars of existing languages, Chomsky proposed a set of generative rules that may be used to construct a language. A striking assertion in Chomsky's work, which provides an extreme counterpoint to Shannon's models above, is that *no* finite alphabet Markov chain approximation can constitute a true grammar (in Chomsky's sense)!

This assertion seems to contradict the efficacy of the Markov chain model in a practical setting such as decoding scrambling text, as well as the theorems that follow proving that these rules converge in a precise sense to true language. In order to resolve this (apparent) contradiction, it is necessary to consider Chomsky's definitions of grammars and hierarchy more carefully [2]. For now, we will view Shannon's model as a starting point, and simply use it as a proof of concept that language can be studied through probabilistic methods.

3.3.3 Convergence of the Markov chain models

We may now apply the ideas of the previous sections to form a precise notion of the entropy rate of English.

[2] This will have to wait till the next iteration of these notes.

1. We model English as a stationary stochastic process taking values in $A = \{a, b, \ldots, z, \_\}$. Thus, its entropy rate exists.

2. We construct the n-gram Markov chain approximations with transition matrix $Q^{(n)}$ and stationary distribution $\mathbb{P}_{\text{true}}$ independent of n. The entropy rate of these Markov chains may be computed using equation (3.1.1).

3. As $n \to \infty$ we show below that the approximations converge to true language in the sense of relative entropy.

The notation differs slightly from Section 3.1. We now use F_n to denote the entropy rate of the n-gram, letter-based, approximation to English. We will consider the word-based approximation after this example has been understood.

In order to compute the entropy rate using equation (3.1.1), we let $x = a_1 a_2 \ldots a_{n-1}$ and $y = a_2 \ldots a_n$ and recall that the equilibrium distribution and transition probability are given by

$$\pi^{(n)}(x) = \mathbb{P}_{\text{true}}(a_1 a_2 \ldots a_{n-1}), \quad Q^{(n)}(x, y) = \frac{\mathbb{P}_{\text{true}}(a_1 a_2 \ldots a_n)}{\mathbb{P}_{\text{true}}(a_1 a_2 \ldots a_{n-1})}. \tag{3.3.1}$$

Let H_n denote the entropy of the sequence $(X_1, X_2, \ldots, X_{n-1})$. Since $\mathbb{P}_{\text{true}}$ and the n-gram approximation agree on sequences of length $n-1$, we find that

$$H_n = -\sum_{a_1, \ldots, a_{n-1}} \mathbb{P}_{\text{true}}(a_1 a_2 \ldots a_{n-1}) \log \left(\mathbb{P}_{\text{true}}(a_1 a_2 \ldots a_{n-1}) \right).$$

This expression may be substituted in equation (3.3.1) to obtain

$$F_n = H_n - H_{n-1}, \quad \text{or} \quad H_n = F_1 + F_2 + \ldots F_n.$$

An argument similar to Section 3.1 shows that F_n is a decreasing sequence and that $F = \lim_{n \to \infty} F_n$ is the entropy rate of the true language.

This calculation shows that the n-gram approximations converge to true language, and thus so does the entropy rate. However, it does not provide us with an efficient way to compute the entropy rate F. The catch is that the alphabet size for the n-gram approximation has size 27^{n-1}. It becomes increasingly hard to estimate the parameters $Q^{(n)}$ as n increases. In order to get a feel for the numbers involved, observe that $26^3 = 19683$, $27^4 = 531441$, $27^5 = 14348907$. Shannon invented a guessing game that re-encodes English text in a way that allows a relatively easy way to estimate the entropy of English.

The idea that the n-gram approximations converge to the true language is formalized as follows. Both $\mathbb{P}_{\text{true}}$ and $Q^{(n)}$ define probability distributions on the space of sequences of length N, Ω_N, for every N. We fix the length N, as well as the scale of approximation n and use $\mathbb{P}_{\text{true,N}}$ and $\mathbb{P}_{N_n}$ to denote these probability distributions. We then compute the relative entropy $D(\mathbb{P}_{\text{true},N} \parallel \mathbb{P}_{n,N})$ as follows.

First, we note that $D(\mathbb{P}_{\text{true},N} \parallel \mathbb{P}_{n,N}) = 0$ when $n \geq N$, since both probability distributions agree on sequences of length n. Therefore, it is sufficient to restrict attention to $n < N$. In this case, we obtain

$$D\left(\mathbb{P}_{\text{true},N} \parallel \mathbb{P}_{n,N}\right) = \sum_{x_N = a_1 \ldots a_N} \mathbb{P}_{\text{true}}(x_N) \log \frac{\mathbb{P}_{\text{true}}(x_N)}{\mathbb{P}_N^{(n)}(x_N)} = H(\mathbb{P}_{N,n}) - H(\mathbb{P}_{\text{true},N}),$$

where the last expression follows from expanding the term within the logarithm and using the definition of $\mathbb{P}^{(n)}$. As in Section 3.1 we find that

$$H(\mathbb{P}_{\text{true},N}) = F_1 + F_2 + \ldots + F_N.$$

Similarly, we use the entropy chain rule to find that

$$H(\mathbb{P}_{N,n}) = F_1 + F_2 + \ldots + F_n + (N-n)F_n.$$

Therefore, the normalized relative entropy satisfies the inequality

$$\frac{1}{N} D\left(\mathbb{P}_{\text{true},N} \parallel \mathbb{P}_{n,N}\right) = \frac{1}{N} \sum_{k=n+1}^{N} (F_n - F_k) \leq \frac{N-n}{N}(F_n - F)$$

where $F = \lim_{n\to\infty} F_n$. Thus,

$$\limsup_{N\to\infty} \frac{1}{N} D\left(\mathbb{P}_{\text{true},N} \parallel \mathbb{P}_{n,N}\right) \leq (F_n - F).$$

3.4 Words and word boundaries

In this section, we consider the word based Markov chains in more detail. We first define the *lexicon*, $\mathcal{L}$ of a language to be the set of words that it contains. Since we will need to distinguish between sequences of letters and words, let $\Omega_{\mathcal{L}}$ and Ω_A denote the space of sequences taking values in $\mathcal{L}$ and A respectively.

So far we have modeled true language as a probability distribution $\mathbb{P}_{\text{true}}$ on the space Ω_A. However, since language *is* a sequence of words, we could also have chosen true language to be a probability distribution $\mathbb{Q}_{\text{true}}$ on the space $\Omega_{\mathcal{L}}$. What's going on here is that we need to distinguish between language as an abstract construct and its representations in different forms (text, speech, strings of words, etc.). Thus, in order to ensure that our definitions are consistent, we must find an invertible map between Ω_A and $\Omega_{\mathcal{L}}$ such that the probabilities of corresponding events are equal.

This formal description has a familiar interpretation. The act of spelling associates a unique sequence of letters to each word. Let us call this map $\texttt{spell} : \mathcal{L} \to A$, so that $\texttt{spell}(w)$ is a sequence of letters $a_1 \ldots a_p$ that spell the word w. This extends to a sequence of words in $\Omega_{\mathcal{L}}$ as follows. A sequence $\{w_1 w_2 \ldots w_p\}$ is mapped to $\texttt{spell}(w_1)\_\texttt{spell}(w_2)\_ \ldots \texttt{spell}(w_p)\_$. Conversely, given a sequence in Ω_A that is a sample of true language (i.e. lies in the support of $\mathbb{P}_{\text{true}}$), the associated sequence of words may be defined to be

the letters between spaces. (Note that one needs to be more careful mapping letters to words; a sequence of letters that isn't part of the lexicon, does not have a preimage. We avoid this issue by choosing only the sequences that live in the support of $\mathbb{P}_{\text{true}}$).

The use of a special character for the space is redundant. The fact that texts without spaces cannot be read easily reflects only our training, not a fundamental argument for the use of the space character. For example, it is possible to read slowly and break a sentence into words, even when the space has been removed. Similarly, several written scripts drop information (e.g. vowels in Semitic languages), which experienced readers judge from context. This issue becomes especially important when considering speech. By the same logic as above, speech is a manifestation of true language. The fundamental unit of speech is a *phoneme*, and the act of dictation is the conversion of a string in Ω_A to a string of phonemes. Unlike written text, words in speech are *not* separated by spaces (that would sound like staccato speech, whereas we prefer words to flow). Yet humans have no difficulty separating a sequence of phonemes into a string of words.

This discussion suggests that the space character is redundant and that it should be possible to parse a sequence of letters without spaces into a sequence of words. We will call this map $\texttt{parse} : \Omega_A \to \Omega_{\mathcal{L}}$. In what follows we describe a model that is similar to the digram model for text.

3.4.1 Parsing word boundaries

We adopt the following notation in this section. Let L denote the lexicon, which may be thought of as an "alphabet" of words; let A^* denote the space of infinite sequences of letters in $A = \{a, b, \ldots, z, \text{“ ”}\}$; and let L^* = space of sequences of words. Observe that a Markov chain on L generates a sequence in L^* whereas the n-gram model generates a sequence in A^*. These spaces are related through the following maps:

- `spell`: $L \to A^*$, $\omega \mapsto a_1 a_2 \ldots a_m$
- `parse`: $A^* \to L^*$, $a_1 a_2 \ldots a_m \mapsto \omega_1 \omega_2 \ldots$

Parsing a string in A^* is easy when A includes the space character. All we have to do is to the letters at spaces.

It is a more interesting question to ask how one may parse a string when we don't have a special character for space. We approach this problem by constructing an energy function that uses the mutual information. Given two random variables X, Y with joint law $P_{X,Y}$ the mutual information is defined as the relative entropy $I(X,Y) = D(P_{X,Y}||P_X P_Y)$. The first argument $P_{X,Y}$ is the joint distribution of X and Y. The second is the product of the marginals of X and Y, corresponding to the assumption that X and Y are independent. Loosely speaking $I(X,Y)$ quantifies how strongly coupled X and Y are.

We define an energy for word boundaries as follows. Consider two strings $\theta = a_1 \ldots a_m$ and $\tau = a_{m+1} \ldots a_n$ and let $\sigma = \theta\tau = a_1 \ldots a_m a_{m+1} \ldots a_n$ denote

their concatenation. We assume that X and Y are consecutive strings drawn from the true language, such that $X = \theta$ and $Y = \tau$. Then the joint distribution and marginals are

$$P_{X,Y}(X = \theta, Y = \tau) = \mathbb{P}_{true}(a_1 a_2 \dots a_{n+m}),$$
$$P_X(X = \theta) = \mathbb{P}_{true}(a_1 \dots a_m), \quad P_Y(Y = \tau) = \mathbb{P}_{true}(a_{m+1} \dots a_n).$$

The mutual information of these random variables is

$$I(X;Y) = \sum_{\theta,\tau} \mathbb{P}_{true}(\theta\tau) \left(\log \frac{\mathbb{P}_{true}(\theta\tau)}{\mathbb{P}_{true}(\theta)\, \mathbb{P}_{true}(\tau)} \right)$$

The mutual information vanishes when the strings X and Y are independent. While we do not expect two distinct words to be independent, heuristically it is clear that two consecutive strings from distinct words should be less closely related than two strings that are part of the same word. The simplest version of this idea is an analogue of the digram approximation. We consider two consecutive pairs of letters, i.e. $n = m = 2$ ande we define the *binding energy* for a sequence of four letters $a_1a_2a_3a_4$ to be

$$E(a_1a_2a_3a_4) = \log \left(\frac{\mathbb{P}_{true}(a_1a_2a_3a_4)}{\mathbb{P}_{true}(a_1a_2)\mathbb{P}_{true}(a_3a_4)} \right)$$

The following numerical experiment may now be carried out: assume the law of true English is known (say for all strings $a_1a_2a_3a_4$ of length 4). Now take a corpus of English text and 'clean' it by removing all punctuation and case, including the space character. This gives a string of letters in the Latin alphabet. In order to determine the word boundaries in this text, we define the function $e(k) = E(a_k a_{k+1} a_{k+2} a_{k+3})$, which computes the binding energy of consecutive 4-strings. We then place word boundaries at the minima of $e(k)$. This naive scheme works surprisingly well (see [12, Fig. 1.4]).

3.5 The HMM model for machine translation

Machine translation is a fundamental task in natural language processing. We assume given a string of words or phrases in one language (say French). Our task is to translate this string of words into another language (say English). The difficulty of this task depends on the differences between the languages, but it is challenging enough even for two languages within the same family, such as French and English. One of the difficulties is that word for word translation often provides poor results, since the use of different words in different languages relies strongly on context. Thus, what follows is a simplified model which nevertheless contains a central idea of succesful algorithms [**?**].

Assume given a sequence of words in French, which we denote by $\underline{f} = f_1 f_2 \dots f_m$. We seek a sequence of English words $\underline{e} = e_1 e_2 \dots e_m$ that is a

translation of $\underline{f}$. [3] In order to apply the Bayesian paradigm to this problem we must first associate a joint probability distribution to the strings $\underline{e}$ and $\underline{f}$. We may then apply Bayes rules as usual to obtain

$$\mathbb{P}(\underline{e}|\underline{f}) \propto \underbrace{\mathbb{P}(f|e)\,\mathbb{P}(e)}_{\text{constructed using the prior}} .$$

Here $\underline{e}$ is the hidden state – the unknown string in English, and $\underline{f}$ is the observation – the given English string. Our best guess of the translation is then

$$e_* = \text{argmax}_{\underline{e}} \log \mathbb{P}\left(\underline{f}|\underline{e}\right) \mathbb{P}\left(\underline{e}\right) \tag{3.5.1}$$

We construct the prior using Markov chains. To this end, we assume that

- Step 1: $\underline{e}$ is a Markov chain (for example, as in Section 1.3).
- Step 2: We assume we have an English to French statistical dictionary that provides probabilities $\mathbb{P}(f|e)$, i.e. the probability that an English word is associated to a Fench word f. It should be familiar from experience with a dictionary that a single word may have many meanings; in a similar manner, an English word may have many possible translations. Thus, *all* dictionaries are implicitly statistical, since a reader must typically use context to determine the meaning or translation of a word.

Observe that the Markov chain uses only the structure of the language of the observer. We use the transition kernel of this Markov chain to compute the probability of a string of English words $\underline{e} = e_1 e_2 \dots e_m$

$$\mathbb{P}(\underline{e}) = \mathbb{P}(e_1)\, Q(e_1, e_2)\, Q(e_2, e_3) \dots Q(e_{m-1}, e_m), \tag{3.5.2}$$

We then use the statistical dictionary to define the conditional probability

$$\mathbb{P}(\underline{f}\,||\underline{e}) = \mathbb{P}(f_1|e_1)\,\mathbb{P}(f_2|e_2) \dots \mathbb{P}(f_m|e_m). \tag{3.5.3}$$

With these two elements in place, the task of machine translation is reduced to the task of computing the argmax in equation (3.5.1). This maximum may be computed fast using the fact that the logarithm of the likelihood can be written as a sum of terms. This algorithm is of independent interest and is treated in the next section.

3.6 Dynamic programming (Bellman's algorithm)

Effective implementation of the HMM model requires a fast algorithm to determine the mode of the posterior. The log-likelihood of the posterior has the feature that it decomposes into a sum of terms that depend only on their nearest neighbors. This structure appears in several graphical models and it may be exploited as follows.

[3] It is not essential that the strings have the same number of words since we may map words to phrases, or map a word to a null character, but we will assume a word-to-word translation for simplicity.

Theorem 13 (Bellman's algorithm)**.** *Assume given a function of the form*

$$F(x_1, \dots, x_n) = f_1(x_1, x_2) + f_2(x_2, x_3) + \cdots + f_{n-1}(x_{n-1}, x_n), \tag{3.6.1}$$

where the x_k take values in a finite set S_k, such that $|S_k| \leq s$ for all k. Then the minimum of F can be determined in $\mathcal{O}(ns^2)$ steps.

Remark 14. The size of the search space is $\mathcal{O}(s^n)$ since $|S_k| = \mathcal{O}(s)$ for each k. The algorithm reduces the size of the search space dramatically using a recursive strategy.

Proof. The algorithm relies on a recursive decomposition that provides the min and argmin of partial sums. The last step is a form of back propagation that provides the min and argmin of F.
Step 1. Forward iteration. We define

$$\begin{aligned} h_k(x_k) &= \min_{x_1,\dots,x_{k-1}} [f_1(x_1, x_2) + \dots f_{k-1}(x_{k-1}, x_k)] \\ &= \min_{x_{k-1}} \left[h_{k-1}(x_{k-1}) + f_{k-1}(x_{k-1}, x_)\right]. \end{aligned}$$

and let $\Phi(x_k)$ denote the value of x_{k-1} at which the min is achieved:

$$\Phi_k(x_k) = \mathrm{argmin}_{x_{k-1}} \left[h_{k-1}(x_{k-1}) + f_{k-1}(x_{k-1}, x_)\right]$$

For each index k, we assume given a value x_k and we must sweep through all values of x_{k-1} to determine $h_k(x_k)$ and $\Phi_k(x_k)$. This requires at most s^2 steps. Since there are n possible indices, the entire procedure takes at most ns^2 steps.

Step 2. Back substitution. When we get to the last step, we observe that the minimum over x_n of $h_n(x_n)$ is the minimum of F. Let $\bar{x}_n$ be this argmin. Then

$$\begin{aligned} h_n(\bar{x}_n) &= \min_{x_1,x_2,\dots,x_{n-1},x_n} [f_1(x_1, x_2) + \cdots + f_n(x_{n-1}, x_n)] \\ &= \min_{x_1,x_2,\dots,x_{n-1},x_n} F(x_1, \dots, x_n). \end{aligned}$$

We determine the argmin of F by backsubstitution, setting

$$\bar{x}_{n-1} = \Phi_n(\bar{x}_n), \quad \bar{x}_{n-2} = \Phi_{n-1}(\bar{x}_{n-1}), \quad \dots, \quad \bar{x}_1 = \Phi_2(\bar{x}_2).$$

□

Chapter 4

Gaussian processes and Fourier analysis

4.1 Motivation

The Bayesian paradigm of inference relies on the construction of priors on the space of signals. These lectures began with models of text because of its simple structure: the signal $\{X_t\}_{t=1}^{\infty}$ takes values in a finite alphabet and the time t is discrete. When we extend our interest to signals such as music or speech, we must consider continuous random variables $\{X_t\}$ since the underlying auditory signals are pressure waves. Music is typically less complex than speech, since (many forms of) music can be transcribed with a score. Examples of such signals are shown in the Figures below.

In order to apply the Bayesian paradigm to such signals we must construct models that generate music or speech signals. This requires familiarity with stochastic processes in continuous space and time. While a 'proper' mathematical treatment of this subject can seem forbiddingly abstract, the theory has a strong intuitive foundation, since a small number of fundamental examples can be used to construct signals of great complexity. The most important examples are Gaussian processes, compound Poisson processes, and continuous time Markov processes. In each case, the underlying stochastic process is characterized by a relatively simple set of parameters (covariance kernels for Gaussians, jump kernels for compound Poisson processes and generators for Markov processes).

The purpose of this chapter and the next is to illustrate the Gaussian paradigm. We first review Gaussians on $\mathbb{R}^n$ and the Fourier transform. These ideas are then extended to random Gaussian functions and the construction of stationary Gaussian processes. The unifying thread is that all Gaussian processes are characterized by covariance matrices (which are called covariance kernels when $n \to \infty$). In the next chapter, these ideas culminate in theory of Brownian motion.

A central tool, especially when we consider random functions, is Fourier analysis. The main idea here is to decompose a space of functions into simple building blocks (a Fourier basis) and then to form random combinations of these basis functions. Fourier analysis admits many variants depending on the domain of the underlying function. When we consider periodic functions, we use the term Fourier series, function on the line give rise to the Fourier transform, and for discrete vectors we have the Discrete Fourier transform, which is simply a change of basis on $\mathbb{C}^n$. To be concrete, we first consider the Fourier transform of functions on $\mathbb{R}^n$. We then explain how these ideas extend to other forms of Fourier analysis.

4.2 Gaussian distributions on $\mathbb{R}^n$

A Gaussian random variable X is characterized by its mean, $m = \mathbb{E}(X)$, and variance, $\sigma^2 = \mathbb{E}\,(X-m)^2$. It has the pdf

$$p_{m,\sigma^2}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-m)^2}{2\sigma^2}}, \quad x \in \mathbb{R}.$$

Next consider Gaussians on $\mathbb{R}^n$. Write $X = (X_1, \ldots, X_n)$ for the associated random variable. Assume $\mathbb{E}X_i = m_i$ for all i and let $K_{ij} = \mathbb{E}((X_i - m_i)(X_j - m_j))$ denote the covariance matrix. Then the pdf of X is

$$p_{m,K}(x_1, \ldots, x_n) = \frac{1}{\sqrt{\det 2\pi K}} e^{-\frac{1}{2}(x-m)^T K^{-1}(x-m)}.$$

In what follows, we will typically set $m = 0$ and write p_K instead of $p_{m,K}$ since this makes the formula most transparent. The mismatch in notation between Gaussians on the line and Gaussians on $\mathbb{R}^n$ is unfortunate, but conventional. We assume that the covariance matrix K is symmetric and positive definite, so that K^{-1} is well-defined. The formulas have natural limits when K is singular.

The theory of Gaussian processes and Fourier transforms are intimately related. The Fourier transform of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined by

$$\hat{f}(\xi) := \int_{\mathbb{R}^n} e^{-ix\cdot\xi} f(x) dx, \quad \xi \in \mathbb{R}^n.$$

The Fourier transform is its own inverse, except for a minor change in sign and a normalizing factor of 2π

$$f(x) := \frac{1}{(2\pi)^n} \int_{\mathbb{R}^n} e^{ix\cdot\xi} \hat{f}(\xi) d\xi.$$

There are several conventions for where one places the factors of 2π in the Fourier transform. We use the probabilists convention since it provides simple formulas for the Fourier transform of Gaussians.

In probability theory, the Fourier transform is called the characteristic function of a random variable. Suppose X is a random variable in $\mathbb{R}^n$. We then define its characteristic function to be

$$\varphi(\xi) := \mathbb{E}(e^{-i\xi\cdot X}), \quad \xi \in \mathbb{R}^n.$$

When X has a pdf $f(x)$, the characteristic function $\varphi(\xi)$ is the same as $\hat{f}(\xi)$. But the characteristic function is well-defined even when X does not have a pdf. For example, when $X = \pm 1$ with probability $1/2$, we find that

$$\varphi(\xi) = \frac{1}{2}e^{i\xi} + \frac{1}{2}e^{-i\xi} = \cos\xi.$$

Assume X is a mean-zero Gaussian random variable with density

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-x^2/2}.$$

The Fourier transform of f may be computed by completing the square or using Cauchy's integral formula from complex analysis. We find that $\hat{f}$ is again an (un-normalized) Gaussian

$$\hat{f}(\xi) = e^{-\sigma^2\xi^2/2}.$$

We may also express the above integral in probabilistic notation

$$\mathbb{E}\left(e^{-i\xi X}\right) = e^{-\sigma^2\xi^2/2}, \quad \xi \in \mathbb{R}, \quad X \sim \mathcal{N}(0, \sigma^2).$$

In the limit $\sigma \to 0$, the random variable X takes the value 0 with probability 1. In this limit, $\hat{f}(\xi) \equiv 1$, so that all frequencies carry the same weight. Formally, we say that $f \to \delta$, the Dirac δ at 0.

Consider a multivariate, mean-zero Gaussian with covariance K. We find that its characteristic function is

$$\varphi(\xi) = e^{\frac{1}{2}\xi^T K\xi}, \quad \xi \in \mathbb{R}^n.$$

The entropy of an $\mathbb{R}^n$ valued random variable with pdf p is

$$H(X) = -\int_{\mathbb{R}^n} p(x)\ln p(x)dx.$$

When $X \in \mathbb{R}^n$ is Gaussian with covariance kernel K we may evaluate this integral to find

$$H(X) = \frac{1}{2}\ln\left(\det\left(2\pi eK\right)\right).$$

4.3 Gaussian random functions

We now construct random functions. The common underlying principle is that we are given an orthonormal basis of vectors and we form linear combinations

of these basis vectors. In $\mathbb{R}^n$, the underlying orthonormal basis is the standard basis $\{e_i\}_{i=1}^n$, so that $x = \sum_{i=1}^n x_i e_i$. Thus, when we form a Gaussian vector $X = X_1 e_1 + X_2 e_2 + \ldots + X_n e_n$, we are forming a random combination of basis vectors. This idea extends without any change to spaces of functions.

Fourier series provide the most familiar example. We consider the space $L^2(0,1)$ of complex-valued functions on the interval $[0,1]$ with the inner product

$$\langle f, g \rangle := \int_0^1 f(x)\bar{g}(x)dx.$$

The set $\{e^{2\pi ikx}\}_{k\in\mathbb{Z}}$ is orthonormal in $L^2(0,1)$, since

$$\langle e^{2\pi ikx}, e^{2\pi ilx} \rangle = \int_0^1 e^{2\pi i(k-l)x} dx = \begin{cases} 0 \text{ if } k \neq l \\ 1 \text{ if } k = l. \end{cases}$$

The Fourier series of a function $f \in L^2(0,1)$ is its expression in this basis:

$$f(x) = \sum_{k\in\mathbb{Z}} \hat{f}(k) e^{2\pi ikx}, \quad \hat{f}(k) = \int_0^1 e^{-2\pi ikx} f(x)\, dx.$$

This series is convergent in L^2 and it satisfies Parseval's equality

$$\|f\|^2_{L^2(0,1)} = \int_0^1 |f(x)|^2\, dx = \sum_{k\in\mathbb{Z}} |\hat{f}(k)|^2.$$

It is convenient to assume that f is complex valued, since the basis $\{e^{2\pi ikx}\}_{k\in\mathbb{Z}}$ is simpler to manipulate than a trigonometric basis. When f is real valued, the Fourier coefficients satisfy the relation

$$\overline{\hat{f}(k)} = \hat{f}(-k), \quad k \in \mathbb{Z}.$$

We may construct random functions by considering Fourier series with random coefficients. To be concrete, suppose we have a sequence of positive numbers $\{\sigma_k^2\}_{k\in\mathbb{Z}}$ and a sequence of iid standard normal random variables $\{X_k\}_{k\in\mathbb{Z}}$, and form the Fourier series

$$f(x) = \sum_{k\in\mathbb{Z}} \sigma_k X_k e^{2\pi ikx}.$$

This series converges almost surely provided the variances σ_k^2 decrease fast enough as $|k| \to \infty$. We use Parseval's equality and $\mathbb{E}(X_k^2) = 1$ for all k to find that

$$\mathbb{E}\left(\int_0^1 |f(x)|^2\, dx \right) = \sum_k \sigma_k^2,$$

which is finite when σ_k^2 decays fast enough.

The above series is an example of a Gaussian random function. We saw in the previous section that multivariate Gaussians are characterized by their covariances. This idea extends to complex-valued Gaussian random functions too. The Gaussian random function f is completely characterized through its *covariance kernel*

$$K(x,y) := \mathbb{E}\left(f(x)\overline{f(y)}\right) = \mathbb{E}\left(\sum_{k,l\in\mathbb{Z}} \hat{f}(k)\overline{\hat{f}(l)}e^{2\pi i(kx-ly)}\right).$$

Since $\mathbb{E}(X_k X_l) = \delta_{kl}$ all the cross-terms vanish and we obtain the covariance kernel [1]

$$K(x,y) = \sum_{k\in\mathbb{Z}} \sigma_k^2 e^{2\pi ik(x-y)}.$$

Thus, we see that the choice of a sequence $\{\sigma_k^2\}_{k\in\mathbb{Z}}$ determines a family of Gaussian random functions. The most important example of such random functions is Brownian motion. As a warm-up, let's first revisit the central limit theorem using Fourier analysis.

4.4 Fourier transforms and the central limit theorem

Theorem 15 (Central limit theorem)**.** *Assume $\{X_k\}_{k=1}^{\infty}$ is an iid sequence such that $\mathbb{E}(X_k) = 0$ and $\mathbb{E}(X_k^2) = 1$. Then*

$$\lim_{n\to\infty} \mathbb{P}\left(a \le \frac{X_1 + \cdots + X_n}{\sqrt{n}} \le b\right) = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-x^2/2} dx, \quad -\infty < a < b < \infty. \tag{4.4.1}$$

Proof. The proof has two aspects. The first is a reformulation of convergence in distribution of random variables in terms of convergence of their characteristic functions. The second is an almost explicit calculation that reveals the universality of Gaussians.

We won't prove the first fact, but here is the essential idea. Suppose $\{Z_n\}_{n=1}^{\infty}$ is a sequence of real-valued random variables. We say that Z_n converge in distribution to a random variable Z, if $\lim_{n\to\infty} \mathbb{P}(Z_n \le a) = \mathbb{P}(Z \le a)$ for every $a \in \mathbb{R}$. This definition is equivalent to the following: Z_n converges in distribution to Z if

$$\lim_{n\to\infty} \mathbb{E}(e^{i\xi Z_n}) = \mathbb{E}(e^{i\xi Z}), \quad \xi \in \mathbb{R}.$$

The relation between these definitions is roughly as follows. The first criterion may also be written as

$$\lim_{n\to\infty} \mathbb{E}\left(\mathbf{1}_{(-\infty,a]}(Z_n)\right) = \mathbb{E}\left(\mathbf{1}_{(-\infty,a]}(Z)\right), \quad a \in \mathbb{R},$$

[1] The Kronecker δ symbol means that $\delta_{kl} = 0$ unless $k = l$ and $\delta_{kl} = 1$ when $k = l$.

where the indicator function $\mathbf{1}_{(-\infty,a]}(x)$ takes the value 1 when $x \leq a$ and vanishes otherwise.

What is common to both definitions is the idea that we evaluate the expectations of the random variables on a suitably rich family of functions. In the above calculations, these are the functions $e^{i\xi x}$ or $\mathbf{1}_{(-\infty,a]}$ for arbitrary a and ξ. In order to show that both definitions are equivalent, we must show that it is possible to approximate every function $\mathbf{1}_{(-\infty,a]}$ by a linear combination of the functions $e^{i\xi x}$. To do this precisely requires some care, since $\mathbf{1}_{(-\infty,a]}$ is discontinuous. Such an approximation is obtained by first approximating the discontinuous function $\mathbf{1}_{(-\infty,a]}$ from above and below with piecewise linear continuous functions. Then we note that continuous functions can be uniformly approximated by polynomials, and thus trigonometric polynomials, using Weierstrass' theorem. Precise formulations of this idea may be found in [**?**].

The proof now reduces to a computation with characteristic functions. Since the X_k are iid, they all have the same characteristic function

$$\varphi(\xi) := \mathbb{E}(e^{i\xi X_k}).$$

We differentiate with respect to ξ to find that

$$\varphi'(\xi) = \mathbb{E}(iX_k e^{i\xi X_k}), \quad \varphi''(\xi) = \mathbb{E}((iX_k)^2 e^{i\xi X_k}). \tag{4.4.2}$$

We evaluate these expressions at $\xi = 0$, and use the assumptions on X_k to obtain

$$\varphi'(0) = i\,\mathbb{E}(X_k) = 0, \quad \varphi''(0) = -\mathbb{E}(X_k)^2) = -1. \tag{4.4.3}$$

Therefore, $\varphi(\xi)$ has a Taylor series expansion near zero of the form

$$\varphi(\xi) = 1 - \frac{\xi^2}{2} + o(\xi^2). \tag{4.4.4}$$

For example, when $X_k = \pm 1$ with probability $1/2$, $\varphi(\xi) = \cos\xi$, which clearly has the above Taylor expansion near zero.

We now combine these ideas. Equation (4.4.1) is equivalent to

$$\mathbb{E}(e^{\frac{i\xi(X_1+\cdots+X_n)}{\sqrt{n}}}) \to e^{-\xi^2/2}, \quad \xi \in \mathbb{R},$$

since the right hand side is the characteristic function of the standard Gaussian. We then compute the left hand side. Clearly,

$$\mathbb{E}\left(e^{i\xi(X_1+\cdots+X_n)}\right) = \mathbb{E}\left(e^{iX_1}, e^{iX_2}\ldots e^{iX_n}\right).$$

Since the X_k are iid the above expression equals

$$= \mathbb{E}\left(e^{iX_1}\right)\mathbb{E}\left(e^{iX_2}\right)\ldots\mathbb{E}\left(e^{iX_n}\right) = \underbrace{\varphi(\xi)\varphi(\xi)\ldots\varphi(\xi)}_{n \text{ times}} = \varphi(\xi)^n.$$

Now fix ξ, rescale by $1/\sqrt{n}$ and use the Taylor expansion (4.4.4) to see that

$$\mathbb{E}\left(e^{\frac{i\xi}{\sqrt{n}}(X_1+\cdots+X_n)}\right) = \varphi\left(\frac{\xi}{\sqrt{n}}\right)^n = \left(1 - \frac{\xi^2}{2n} + \ldots\right)^n \to e^{-\xi^2/2}.$$

□

This theorem reaches its full importance when used as the foundation for the theory of Brownian motion. We now turn to this important subject.

Chapter 5

Brownian motion

5.1 Introduction

The physical phenomenon of Brownian motion was first recorded by the botanist Robert Brown in 1827. He observed the erratic motion of pollen grains under a microscope, but did not provide an explanation for this motion. The origin of this phenomena remained obscure until 1905, when Albert Einstein recognized that the random motion of the pollen grains, or more generally colloidal particles, originated in a vast number of microscopic collisions with even smaller particles, invisible under a microscope. This allowed Einstein to provide a microscopic explanation for diffusion and a method to compute Avogadro's number. Einstein's predictions were verified in Perrin's experiments in 1908 and provided evidence for the existence of atoms and molecules (see Figure 5.1.1).

The construction of a mathematical theory of Brownian motion, beginning with the work of Norbert Wiener, is one of the triumphs of 20th century mathematics. Wiener modeled Brownian motion as a Gaussian random function, writing it as a Fourier series of the form

$$B(t) = X_0 t + \sqrt{2}\sum_{n=1}^{\infty} \frac{X_n}{n} \sin n\pi t, \quad t \in [0,1],$$

where $\{X_n\}_{n=0}^{\infty}$ is an iid sequence of standard normal random variables.

This is a series of the type we saw in Section 4.3. The specific choice of variance ensures that the process $B(t)$ has the following properties:

1. $B(0) = 0$.
2. The increment $B(t) - B(s)$ is independent of $B(s)$ for all $t > s$.
3. The increment $B(t) - B(s)$ is normal with mean zero and variance $t - s$ for all $0 \leq s < t$.
4. $B(t)$ is continuous with probability 1.

This construction reveals certain features of the Brownian motion, but obscures others. The most delicate property to establish is (4). In order to understand some of the subtleties involved, let us formally differentiate the above expression to obtain

$$B'(t) \stackrel{?}{=} X_0 + \sqrt{2}\,\pi \sum_{n=1}^{\infty} X_n \cos n\pi t.$$

In contrast with the series considered in Section 4.3, the above series diverges almost surely. Thus, the erratic nature of (physically observed) Brownian motion is refected in the fact that the (mathematically constructed) function $B(t)$ is *nowhere differentiable*.

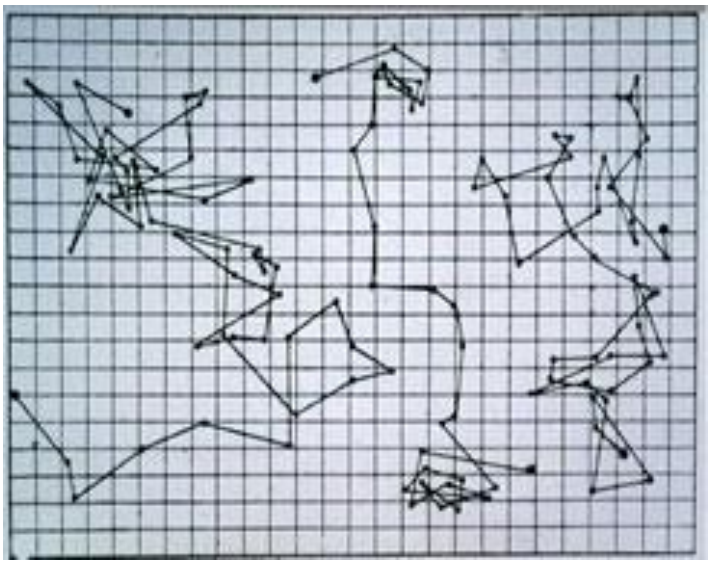

Figure 5.1.1: An image from Perrin's 1908 observations of Brownian motion

.

A proof of these facts would take us too far afield, but one may get a feeling for what is involved by studying simpler examples, called *lacunary series*. These are Fourier series of the form

$$f(x) = \sum_{n=1}^{\infty} \frac{\sin(n!\pi x)}{n^2}, \quad \text{or} \quad g(x) = \sum_{n=1}^{\infty} a^n \sin(b^n x),$$

with $0 < a < 1 < b$. Such series are called lacunary because most Fourier coefficients vanish (for example, in the first example $\hat{f}(k)$ =0, unless k is an integer of the form $k = n!$). The graphs of these series are typical examples of fractals, a term popularized in the 1970s by Benoit Mandelbrot.

It is easy to establish the convergence of such series. Since $|\sin(y)| \leq 1$, we find that

$$\max_{x\in[0,1]} |f(x) - f_N(x)| \leq \sum_{n=N+1}^{\infty} \frac{1}{n^2} = O(1/N),$$

where $f_N(x)$ denotes the finite sum

$$f_N(x) = \sum_{n=1}^{N} \frac{1}{n^2} \sin(n!\pi x).$$

In the late 1800s, Weierstrass proved that the uniform limit of continuous functions on $[0,1]$ is continuous. In particular, f_N converges uniformly to f, so that f is continuous. The problem however is that the derivative of f_N grows fast. Indeed, we see that the partial sums

$$f'_N(x) = \sum_{n=1}^{N} \frac{n!}{n^2} \cos(n!\pi x)$$

diverge as $N \to \infty$. With some work, one can deduce that the limiting function f, while continuous, does not possess a derivative at *any* point $x \in (0,1)$.

Weierstrass's construction of nowhere differentiable functions was rejected by many mathematicians of his day. [1] Even many who accepted it, viewed such functions as mathematical curiosities that were 'unphysical'.

Today we recognize the work of Weierstrass and Wiener as a triumph of reason. Nowhere differentiable functions are typical, not atypical, and their construction and analysis has deep theoretical and practical consequences.

5.2 The Lévy-Ciesielski construction of Brownian motion

The use of Fourier series makes Wiener's original construction of Brownian motion somewhat cumbersome. A simpler construction, in the same spirit, uses the Haar basis for $L^2(0,1)$. This basis is indexed by two positive integers: $n = 1, 2, 3, \ldots$ is used to divide $[0,1]$ into dyadic intervals of length 2^{-n}; and k, an odd integer between 0 and 2^n indexes an interval of length 2^{-n}. We then define the piecewise constant functions

$$H_k^{(n)}(t) = \begin{cases} 2^{\frac{n-1}{2}}, & \frac{k-1}{n} \leq t < \frac{k}{2^n}, \\ -2^{\frac{n-1}{2}}, & \frac{k}{2^n} \leq t < \frac{k+1}{2^n}, \\ 0 \text{ else.} \end{cases}$$

[1] A typical example, is a remark in a letter from Hermite to Stieltjes in May,1893: I turn away with fear and horror from the lamentable plague of continuous functions which do not have derivatives.

The normalizing factors are chosen to ensure that

$$\int_0^1 \left(H_k^{(n)(t)}\right)^2 dt = \frac{1}{2^{n-1}} 2^{n-1} = 1, \quad \text{and} \quad \langle H_k^{(n)}, H_l^{(m)} \rangle = 0,$$

unless $k = l$ and $n = m$. Next define the Schauder functions

$$S_k^{(n)}(t) = \int_0^t H_k^{(n)} ds.$$

The Haar functions form an orthonormal basis for $L^2[0,1]$. The Schauder functions form a basis for continuous functions on $[0,1]$. More precisely, every $f \in L^2[0,1]$ admits an expansion $f(t) = \sum f_{k,n} H_k^{(n)}(t)$, and this expansion is convergent in $L^2[0,1]$. Similarly, every $g \in C([0,1])$ admits an expansion $g(t) = \sum g_{k,n} H_k^{(n)}(t)$ and this series converges in $C([0,1])$.

We now define Brownian motion on $[0,1]$ as the random function

$$B(t) = \sum_{n=1}^{\infty} \sum_{k \text{ odd}} X_k^n S_k^{(n)}(t), \quad t \in [0,1],$$

where $\{X_k^{(n)}\}$ are iid standard Gaussians. The covariance kernel may be computed using the above definitions and we find that

$$\mathbb{E}\left(B(s)B(t)\right) = \min(s,t).$$

If we take a finite set of points $t_1 < t_2 < \ldots < t_m$, we find that the covariance of the Gaussian vector $B(t_1), B(t_2), \ldots, B(t_m)$ is given by the matrix

$$K = \begin{bmatrix} t_1 & t_1 & t_1 & \ldots & t_1 \\ t_1 & t_2 & t_2 & \ldots & t_2 \\ \vdots & t_2 & t_3 & \ldots & t_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1 & t_2 & t_3 & \ldots & t_m \end{bmatrix}.$$

Finally, we may construct Brownian motion on the interval $[0,\infty)$ by proceeding inductively from interval to interval. Assume that we have constructed Brownian motion on $[0,1]$ as above. We then construct an independent copy of the same process, denoted $\tilde{B}_t$, $t \in [0,1]$, and we set $B_t = B_1 + \tilde{B}_{t-1}$ for $t \in [1,2]$. Proceeding inductively, we obtain Brownian motion on the interval $[0,\infty)$.

5.3 The scaling limit of random walks

The above construction of Brownian motion has the advantage of being direct. However, it obscures the Markov property of Brownian motion (properties (2) and (3) above). This property is easier to see when we view Brownian motion as the scaling limit of random walks.

Suppose $\{Z_k\}_{k=1}^{\infty}$ are iid random variables such that $\mathbb{E}(Z_k) = 0$, $\mathbb{E}(Z_k^2) = 1$. Let $S_n = Z_1 + \cdots + Z_n$. Then the central limit theorem asserts that

$$\lim_{n\to\infty} \mathbb{P}(a \le \frac{S_n}{\sqrt{n}} \le b) = \frac{1}{\sqrt{2\pi}} \int_a^b e^{-s^2/2} ds, \quad -\infty < a < b < \infty.$$

We may interpret the central limit theorem as a statement about random walks in the following way. For each positive integer n, we define a random function $S^{(n)} : [0,\infty) \to \mathbb{R}$ by first defining its values on the grid of times $t = k/n$ by

$$S^{(n)}(t) = \frac{Z_1 + \cdots + Z_{nt}}{\sqrt{n}}, \quad t = \frac{k}{n}.$$

We then extend $S^{(n)}$ to a function on the interval $t \in [0,\infty)$ by piecewise linear interpolation. The central limit theorem implies that $S^{(n)}(t)$ converges to $N(0,t)$ in distribution as $n \to \infty$ for each $t > 0$.

The Markov property of Brownian motion may be obtained from the above approximation. Suppose $0 \le s < t$ and suppose for simplicity that $k = sn$ and $l = tn$ are integers. Then

$$S^{(n)}(s) = \frac{Z_1 + \cdots + Z_k}{\sqrt{n}}, \quad S^{(n)}(t) - S^{(n)}(s) = \frac{Z_{k+1} + \cdots + Z_l}{\sqrt{n}},$$

and it is immediate that the increment $S^{(n)}(t) - S^{(n)}(s)$ is independent of $S^{(n)}(s)$. In the limit $n \to \infty$, we obtain properties (2) and (3) of Brownian motion.

A precise description of the probability of Brownian motion goes as follows. We fix a finite set of times $t_1 < t_2 < t_3 < \cdots < t_m$ and m intervals $[a_i, b_i]$ for $1 \le i \le m$ and define the probability that a Brownian motion passes through these intervals at the given times as follows: [2]

$$\begin{aligned} &\mathbb{P}\left(B(t_1) \in [a_1,b_1], B(t_2) \in [a_2,b_2], \ldots, B(t_m) \in [a_m,b_m]\right) \\ &= \int_{a_1}^{b1} \int_{a_2}^{b2} \cdots \int_{a_m}^{bm} g_{t_1}(x_1) g_{t_2-t_1}(x_2 - x_1) \ldots g_{t_m - t_{m-1}}(x_m - x_{m-1})\, dx_1 \ldots dx_m, \end{aligned} \tag{5.3.1}$$

where

$$g_t(x) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/2t}, \quad -\infty < x < \infty, \quad t > 0.$$

This formula reflects the Markov property of Brownian motion. The factorization in equation (5.3.1) should be compared with the analogous factorization for Markov chains that we used in Chapter 1.1.

A deeper version of the central limit theorem, known as Donsker's theorem, establishes the convergence of the random functions $S^{(n)}$ to Brownian motion.

[2] Think of Brownian motion as a skier's path, the times t_i and intervals $[a_i, b_i]$ defining a set of "slalom gates" that the skier must pass through.

One of the implications of this theorem is the following version of the central limit theorem:

$$\lim_{n\to\infty} \mathbb{P}\left(S^{(n)}(t_1) \in [a_1, b_1], S^{(n)}(t_2) \in [a_2, b_2], \ldots, S^{(n)}(t_m) \in [a_m, b_m]\right)$$

is given by the expression in equation (5.3.1).

5.4 The heat equation

In order to understand what was novel about Einstein's approach to the theory of diffusion, it is necessary to first review the classical theory of diffusion (or heat flow). The heat equation was first introduced in Fourier's work in 1822. He modeled heat flow with a partial differential equation (PDE) for the temperature distribution $\rho(x,t)$ and solved this equation using the method of (Fourier) series. Let us first review this approach in its simplest setting.

The heat equation on the line is the PDE

$$\partial_t \rho = \frac{1}{2}\partial_x^2 \rho, \quad -\infty < x < \infty,\ t > 0 \tag{5.4.1}$$

Here $\rho(x,t)$ is a positive function describing the temperature in an infinitely long bar. This equation also models diffusion (this is the process that describes the spread of a dye in still water). In this context, ρ describes the density of the dye. The initial value problem for the heat equation is to solve (5.4.1) for $\rho(x,t)$ subject to the condition $\rho(x,0) = \rho_0(x)$, where $\rho_0(x)$ is a given initial temperature.

In what follows, we assume that $\rho(x,t)$ decays fast enough as $|x| \to \infty$. Then consider the Fourier transform

$$\hat{\rho}(\xi, t) = \int_{\mathbb{R}} e^{-\mathrm{i}\xi x} \rho(x,t) dx, \quad \xi \in \mathbb{R}.$$

The Fourier transform allows us to transform differentiation (in x) into multiplication (by ξ), as follows:

$$\begin{aligned}\int_{\mathbb{R}} e^{-\mathrm{i}x\xi} \partial_x^2 \rho(x,t) dx &= \int_{\mathbb{R}} \partial_x^2 \left(e^{-ix\xi}\right) \rho(x,t) dx \\ &= (-i\xi)^2 \int_{\mathbb{R}} e^{-2\pi i x\xi} \rho(x,t) dx = (-\xi^2)\hat{\rho}(\xi,t).\end{aligned}$$

Substituting this expression in the heat equation, we find that

$$\partial_t \hat{\rho}(\xi, t) = -\frac{\xi^2}{2}\hat{\rho}(\xi, t) \tag{5.4.2}$$

Thus, we have converted the PDE into an infinite family of ODE, which may be integrated explicitly. We solve equation (5.4.2) to obtain

$$\hat{\rho}(\xi, t) = e^{-\xi^2 t/2}\hat{\rho}_0(\xi), \quad \xi \in \mathbb{R}.$$

We now invert the Fourier transform to obtain the solution formula for the heat equation:

$$\rho(x,t) = \int_{\mathbb{R}} g_t(x-y)\rho_0(y)dy, \tag{5.4.3}$$

where

$$g_t(x) = \frac{1}{\sqrt{2\pi t}} e^{-x^2/t}.$$

The function g_t is called the fundamental solution to the heat equation, since it is the solution to the heat equation with a Dirac initial condition.

The appearance of $g_t(x)$ in the above solution formula has the following probabilistic interpretation. Formula (5.4.3) is equivalent to

$$\rho(x,t) = \mathbb{E}\left(\rho_0(x - B(t))\right),$$

where $B(t)$ is Brownian motion, since $\mathbb{P}(x - B(t) = y) = g_t(y)$.

This allows us to interpret the flow of heat, or diffusion, as arising from the random motion of individual particles. Think of ρ_0 as describing an initial population of particles. Each of these particles executes an independent Brownian motion and in order to obtain the density at the point x at time t, we sum over all the Brownian paths that end at (x,t).

In order to demonstrate the power of this viewpoint, we now explain its utility in situations where there is no exact solution to a PDE.

5.5 The probabilistic solution of the Dirichlet problem

The Laplacian (written Δ) is the differential operator that acts on smooth functions $f : \mathbb{R}^n \to \mathbb{R}$ by

$$\Delta f(x) = \partial^2_{x_1} f + \cdots + \partial^2_{x_n} f,$$

The heat equation in $\mathbb{R}^n$ is the equation

$$\partial_t \rho = \frac{1}{2}\Delta\rho$$

A closely related PDE is Laplace's equation is $\Delta u = 0$. When solving a PDE, we must also consider its initial values and boundary conditions. We will now consider Laplace's equation on a spatial domain $D \subset \mathbb{R}^n$ with boundary ∂D. The Dirichlet problem is as follows: we assume given boundary data $f : \partial D \to \mathbb{R}$ and we must solve

$$\Delta u = 0, \quad x \in D, \tag{5.5.1}$$

$$u(x) = f(x), \quad x \in \partial D. \tag{5.5.2}$$

The Dirichlet problem is explicitly solvable on certain simple domains such as the square or circle and their higher dimensional analogues. For example, when

$D \subset \mathbb{R}^2$ is a square, we may use separation of variables and Fourier series to solve the Dirichlet problem. However, this method is limited.

We will approach the problem in a different way. Let B_t now denote Brownian in $\mathbb{R}^n$ (which is simply n independent copies of Brownian motion in $\mathbb{R}$). Given $x \in D$, let

$$X_t^x = x + B_t,$$

denote a Brownian path starting at x. Given X_t^x define the (random) first exit time

$$T = \min_{t>0}\{X_t^x \text{ does not lie in } D\}.$$

Then the solution to the Dirichlet problem has the probabilistic representation

$$u(x) = \mathbb{E}\left(f(X_T^x)\right). \tag{5.5.3}$$

While it takes some work to establish this formula in full generality, the main idea underlying this formula is the interplay between martingales and harmonic functions, which can be understood in the simpler context of random walks on a lattice. This formula may then be generalized to Brownian motion.

To this end, consider the integer lattice $\mathbb{Z}^n$. Given a function $u : \mathbb{Z}^n \to \mathbb{R}$, define the discrete Laplacian

$$\Delta u(x) = \frac{1}{2^n} \sum_{|x-y|=1} \left(u(y) - u(x)\right), \quad x \in \mathbb{Z}^n.$$

Given a subset $D \subset \mathbb{Z}^n$, we define its boundary ∂D to be the set of points $x \in D$ such that at least one neighbor of x does not lie in D. The discrete Dirichlet problem is as follows. Given a function $f : \partial D \to \mathbb{R}$, we must solve

$$\Delta u = 0, \quad x \in D, \tag{5.5.4}$$

$$u(x) = f(x), \quad x \in \partial D. \tag{5.5.5}$$

This is the discrete analogue of equation (5.4.3). It admits an analogous probabilistic solution formula.

Let X_t^x denote the simple random walk on $\mathbb{Z}^n$, starting at x and $t = 0, 1, 2, \ldots$. Let $T(X_t^x) =$ denote the first time X_t^x hits ∂D. Then the solution to the discrete Dirichlet problem (5.5.5) is

$$u(x) = \mathbb{E}\left(f(X_T^x)\right). \tag{5.5.6}$$

This is the discrete analog of equation (5.5.3).

Given a spatial domain and a Laplacian on it (such as either of the examples above), we say that a function $v : D \to \mathbb{R}$ is *harmonic* if it satisfies $\triangle v(x) = 0$ for each $x \in D$.

The precise definition of a martingale requires some care, since it involves a description of measurable events that can be generated by a collection of random variables. We will sweep these technicalities under the rug and adopt the following working definition.

A real-valued discrete time stochastic process $\{Y_t\}$ is a *martingale* with respect to the filtration generated by the stochastic process $\{Z_t\}$ if

$$\mathbb{E}\left(Y_{t+1}\,|Z_t,\ldots,Z_1\right) = Y_t.$$

The interplay between harmonic functions and martingales is captured in the following.

Theorem 16. *Suppose B_t is simple random walk on $\mathbb{Z}^n$ and $v:\mathbb{Z}^n\to\mathbb{R}$ is harmonic. Then $v(B_t)$ is a martingale with respect to the filtration generated by B_t.*

Proof. The proof is a calculation. We first observe that

$$\mathbb{E}\left(v(B_{t+1})\,|B_t,\ldots,B_1\right) = \mathbb{E}\left(v(B_{t+1})\,|B_t\right)$$

by the Markov property of simple random walk. Next, since the random walk hops to each of its neighbors with probability 2^{-n}, we find that

$$\mathbb{E}\left(v(B_{t+1})\,|B_t\right) = \frac{1}{2^n}\sum_{|y-B_t|=1} v(y) = v(B_t) + \triangle v(B_t),$$

using the definition of the Laplacian. Since v is harmonic, $\triangle v = 0$, and we see that

$$\mathbb{E}\left(v(B_{t+1})\,|B_t\right) = v(B_t).$$

□

This theorem doesn't really suffice to solve the Dirichlet problem in a bounded domain D, but it does give us a feeling for what's going on. In order to solve the Dirichlet problem we want u such that $\Delta u = 0$ in D and $u = f$ on ∂D.

Let x be an interior point of D and let X_t^x denote a simple random walk starting at x. A calculation as in the above theorem tells us that if u solves $\Delta u = 0$ in D then $u(X_t^x)$ is a martingale *provided X_t^x has not exit the domain D.* In order to state this precisely, we define the *stopped* process

$$Y_t^x = X^x_{\min(T,t)}, \quad T = \min_s\{X_s^x \in \partial D\}.$$

It turns out that Y_t^x is also a Markov process and that we have the following stronger version of Theorem 16:

$$\mathbb{E}\left(u\left(Y_{t+1}^x\right)|Y_t^x\right) = u(Y_t^x).$$

Therefore, taking (unconditional) expectations we see that

$$\mathbb{E}\left(u\left(Y_{t+1}^x\right)\right) = \mathbb{E}\left(u(Y_t^x)\right).$$

Proceeding inductively, we find that

$$\mathbb{E}\left(u\left(Y_t^x\right)\right) = \mathbb{E}\left(u(Y_0^x)\right) = u(x).$$

Now let $t \to \infty$ on the left hand side. If the domain is bounded, then $\lim_{t\to\infty} Y_t^x = X_T^x$ with probability one, so that

$$u(x) = \mathbb{E}\left(u(X_T^x)\right) = \mathbb{E}\left(f(X_T^x)\right).$$

In conclusion, we see that the theory of Brownian motion provides a deeper understanding of heat flow by focusing our attention on the stochastic process that describes the 'mechanism' of heat flow. This viewpoint allows us to recover the classical solution formulas for Laplace's equation and the heat equation for domains in $\mathbb{R}^n$. Further, they provide us with a new mathematical structure – martingales – that expand the reach of the classical ideas to spaces that are very different from $\mathbb{R}^n$ (for example, the world-wide-web is a large graph, not a Euclidean space; but we may still define the notion of heat flow on it to model the transmission of information on the web).

Chapter 6

From Music to Markov processes

6.1 Introduction

In this chapter and the next, we apply the Bayesian paradigm to model music and character recognition. Music differs from text in that the underlying signal – sound waves– while linearly ordered in time, are continuous. Characters (such as letters of the alphabet or numbers) are collections of contours in the plane that represent a finite alphabet. These signals are no longer linearly ordered, but they are relatively simple because they represent a finite alphabet.

In order to apply the Bayesian paradigm to infer structure from noisy signals, it is necessary to have a generative model, with both hidden and observed variables, that can create the signals. Given an observation, such a model allows us to infer the hidden state by maximizing likelihood. In both the examples above, a subtle part of the modeling is to choose good priors for the continuous part of these signals (a single note in music, or a single curve within a character, such as the loop in an 'e'). We will use the theory of Brownian motion and stationary Gaussian processes to model this part of the signal.

These examples demonstrate the consistent nature of Bayesian modeling. However, they also reveal some of its flaws. As the structure of the signal becomes more complicated, the Bayesian paradigm may fail because the construction of a realistic generative model is too complicated, or because the computation of the posterior distribution is too expensive.

6.2 The score and the spectrum

Let us begin by reviewing the spectrogram of a music score (Figure 6.2.1).

This image has two distinct structures. The first is a discrete 'skeleton', the *musical score*, which consists of (at least) the following variables:

1. A positive integer m, the number of notes.

2. A set of m intervals $[t_{k-1}, t_k)$, $k = 1, \ldots, m$ denoting the periods of notes.

3. A set of m positive real numbers, ω_k, denoting the frequency of the k-th note.

We may further simplify the score, so that the frequencies ω_k take values in a finite set of frequencies (pure tones). For brevity, we denote

$$\underline{\omega} = (\omega_1, \omega_2, \ldots, \omega_n), \quad \underline{t} = (t_1, t_2, \ldots, t_m).$$

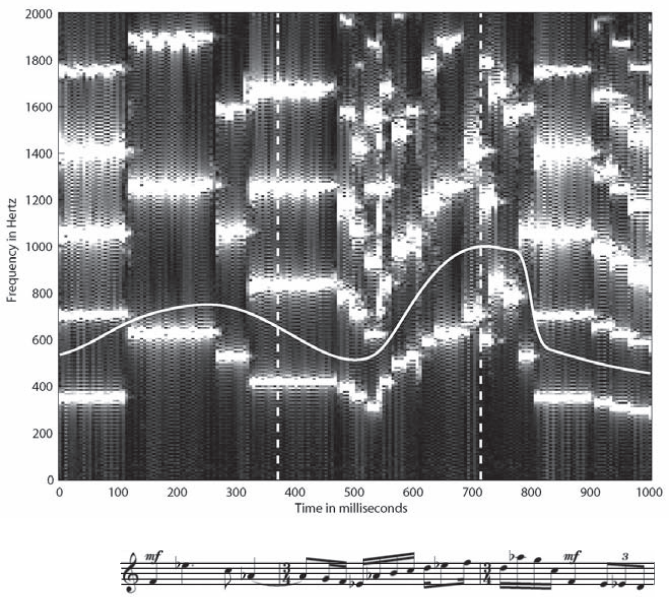

Figure 6.2.1: This image is Figure 2.1 in [12]. The top image is a spectrogram of an oboe playing a Beran cadenza. The spectrogram shows the power in different frequencies as a function of time (regions of high power are light colored). Observe that the power is mainly distributed in the fundamental frequency and harmonics. The variation of the fundamental frequency with time is described by the music score.

The second part of the model is the structure of the signal $s(t)$ in each time interval. The difference in sound between the same note, when played with different musical instruments, is the *timbre* of the instrument. Different instruments sound different, even when they play the same note, because their harmonics (multiples of the frequency of the note) are amplified differently.

Mathematically, we model a single note with base frequency ω as the Gaussian process

$$s(t) = \sum_{k \in \mathbb{Z}} X_k e^{\mathrm{i}k\omega t},$$

where $X_k = A_k + iB_k$, where $A_k \sim B_k \sim N(0, \sigma_k^2/2)$, and we require that $\bar{X}_k = -X_{-k}$ so that $s(t)$ is real, and $\{\sigma_k^2\}_{k=1}^{\infty}$ is a set of positive numbers that is fixed for each instrument. The sequence $\{\sigma_k\}_{k=1}^{\infty}$ describes the power $\mathbb{E}(|X_k|^2)$ in the kth harmonic of the frequency ω. It is called the power spectrum.

In summary, a minimal mathematical model of music consists of two types of stochastic processes:

1. A continuous time discrete space Markov processes (in order to model the score).

2. A stationary Gaussian process (in order to model the timbre of a single note).

Our generative model consists of the random variables $s(t)$, a real valued signal for $t \in [0, T]$, and the score $(m, \underline{\omega}, \underline{t})$. The separation of the discrete structure of the score from the continuous structure of each note may be probabilistically modeled by describing the law of each note, conditional on the score. This assumption yields the law

$$\mathbb{P}(s, m, \underline{\omega}, \underline{t}) = \prod_{j=1}^{m} \underbrace{\left(\mathbb{P}(s|_{I_j}|\omega_j)\right)}_{(a)} \underbrace{\mathbb{P}(m, \underline{\omega}, t)}_{(b)}$$

Part (a) is the law of an individual note of frequency ω_j on the interval $I_j = [t_{j-1}, t_j)$. It is modeled as a stationary Gaussian process. Part (b) is the law of the score. We model it as a continuous time Markov process.

For inference or parsing, we assume the observation $s(t)$ is given. An application of Bayes theorem then leads us back to the task of determining the hidden variables $(m, \underline{\omega}, \underline{t})$ to infer the score by maximum likelihood.

Finally, in order to implement such a model in practice, we need to choose priors for the score and a note. We will use the following choices, both of which are natural. The score is modeled as a compound Poisson process (which is perhaps the simplest class of continuous time, discrete space, Markov processes). We will also make a naive choice for the power spectrum of the instrument. We set $\sigma_k^2 = e^{-\alpha|k|}$. The rest of this chapter is mainly a description of the structure of such processes.

The model presented here is motivated by, and similar in spirit to, that in [12, Ch. 2]. The main difference is that Mumford and Desolneux use a discretized version of this model in their book and suggest some parameter choices for implementation [12]. The model chosen here is a continuous time Markov processes. This is of roughly the same complexity in implementation, but it has the advantage of introducing the reader to some fundamental probabilistic models. The rest of this lecture is primarily a description of continuous time

Markov processes. The implementation of this model to parse an audio signal is part of the second project.

6.3 The generator of a Markov process

We first describe Markov processes $\{X_t\}$ when the time is continuous, say $t \in [0, \infty)$, and X_t takes values in a discrete state space S.

As we have seen in Chapter 1.1, a discrete time homogeneous Markov chain on a finite state space S is completely described by its transition matrix Q_{old} (we use the subscript *old* to distinguish this matrix from an analogous matrix Q_t for a continuous time Markov process). The probability that X_n takes a value $x \in S$ is denoted by $\pi_n(x) := \mathbb{P}(X_n = x)$. By convention, π_n is a row vector. Given the initial law π_0, it is determined by the forward equation

$$\pi_{n+1} = \pi_n Q_{old}, \tag{6.3.1}$$

where $\sum_y Q_{old}(x, y) = 1$ and $Q_{old}(x, y) \geq 0$.

Continuous time Markov processes have a similar theory, except that the role of the transition matrix Q is now played by an $|S| \times |S|$ matrix A, called the *generator* of the Markov process. The generator satisfies the following properties:

1. $A(x, y) \geq 0$ when $x \neq y$.
2. $\sum_y A(x, y) = 0$ for all $x \in S$. Given the off-diagonal elements, this condition may always be imposed by setting $A(x, x) = -\sum_{y \neq x} A(x, y)$.

Now we must describe the law of $\{X_t\}_{t \geq 0}$. Introduce the notation

$$\pi_t(x) = \mathbb{P}(X_t = x), \quad Q_t(x, y) = \mathbb{P}(X_t = y | X_0 = x), \quad t \geq 0.$$

The relation between p_t, Q_t and A is

$$\pi_t = \pi_0 Q_t, \quad Q_t = e^{tA} = \sum_{m=0}^{\infty} \frac{(tA)^m}{m!}, \quad t \geq 0. \tag{6.3.2}$$

Here $\exp(tA)$ is the matrix exponential, which can be defined by either the series above, or as the fundamental solution to the matrix valued differential equation

$$\dot{Q} = AQ, \quad Q(0) = Id,$$

where Id is the identity matrix. The pmf π_t is a row-vector, just as when time is discrete. Equation (6.3.2) should be compared to equation6.3.1 for a Markov chain (in particular, compare Q_t with Q_{old}^n when $t = n$).

The above formulation holds for any finite state space S. When $|S|$ is infinite, say $S = \mathbb{R}$, the generator is a linear operator that could be unbounded. Many aspects of theory continue to hold, but to build intuition it is helpful to first consider the examples below.

Example 17 (Random walk on the circle)**.** Consider a continuous time simple random walk on a set of equally space points on a circle, labeled $S = \{0, 1, \ldots, N-1\}$. The generator of this process is

$$A = \begin{bmatrix} -2 & 1 & 0 & \ldots & 1 \\ 1 & -2 & 1 & \ldots & 0 \\ \vdots & \ddots & & & \vdots \\ 1 & 0 & \ldots & 1 & -2 \end{bmatrix}$$

The difference with a discrete time random walk lies only in the fact that the time between jumps are iid exponential random variables.

Example 18. We may formally extend this example to random walk on the lattice $S = \mathbb{Z}$. The generator is then the infinite matrix

$$A = \begin{bmatrix} \ddots & & & & \ldots & \ldots \\ 1 & -2 & 1 & \ldots & 0 & \\ \vdots & \ddots & \ddots & & \vdots & \\ \vdots & 0 & 1 & -2 & 1 & \ldots \\ \ddots & & & & \ddots\ddots & \end{bmatrix}$$

6.4 Poisson processes on the line

Poisson processes on the line are the 'clocks' for Markov processes. They may be constructed as follows.

Suppose $\{T_k\}_{k=1}^{\infty}$ are iid Exp(λ) random variables. This means that each T_k has the distribution function

$$\mathbb{P}(T_k > t) = \lambda \int_t^{\infty} e^{-\lambda s} ds.$$

The mean of T_k is related to the rate $\lambda > 0$ as follows:

$$\mathbb{E}(T_k) = \int_0^{\infty} t e^{-\lambda t} \lambda dt = \frac{1}{\lambda} \int_0^{\infty} s e^{-s} ds = \frac{1}{\lambda}.$$

We define the epochs $t_0 = 0$, $t_1 = T_1$, $t_2 = T_1 + T_2$, ..., such that $t_{k+1} = t_k + T_k$. It follows from the definition above that the expected number of points $\{t_k\}_{k=1}^{\infty}$ in a unit interval is λ. More generally, if we take a fixed interval I of length a then the random variable $\#(I) := \{\#$ of points in $I\}$ is a Poisson random variable with rate λa (written $\#(I) \sim P(0, \lambda a)$. Thus, for every integer $m \geq 0$

$$\mathbb{P}\left(\#(I) = m\right)) = \frac{(\lambda a)^m}{m!} e^{-\lambda a}.$$

Remark 19. Poisson processes are the fundamental probabilistic model to describe random collections of points. The main underlying assumption is that the number of points in non-overlapping domains are independent Poisson random variables. In particular, Poisson processes are not limited to one-dimensional 'clock rings' $\{t_k\}_{k=1}^{\infty}$, even if this example is of great utility. For example, if you're stuck in a rain shower, to a first approximation the locations at which the raindrops hit the ground are a stationary Poisson process in $\mathbb{R}^2$. This means that for each domain $D \subset \mathbb{R}^2$, $\#(D)$, the number of drops in D is a Poisson random variable with law

$$\mathbb{P}\left(\#(D) = m\right) = \frac{(\lambda|D|)^m}{m!} e^{-\lambda|D|},$$

where $|\cdot|$ = area of D and $\lambda > 0$ is a parameter called the rate of the process (the expected number of points per unit area). Further, if D_1 and D_2 are two regions of the plane that don't overlap, then $\#(D_1)$ and $\#(D_2)$ are independent random variables.

6.5 Compound Poisson processes

We now return to Poisson processes on the line and use them to build a class of Markov process, called compound Poisson processes, using the following random variables:

1. The clock increments $\{T_k\}_{k=1}^{\infty}$. These are iid $\mathrm{Exp}(\lambda)$ random variables that determine the jump times $t_{k+1} = t_k + T_k$, $k \geq 1$, $t_0 = 0$.
2. The magnitude of jumps : an iid sequence $\{X_k\}_{k=1}^{\infty}$ of real-valued random variables, that is also independent of the clock.

We then define the continuous time processes

$$N(t) = \mathrm{argmin}_k\{t_k \geq t\}, \quad S(t) = \sum_{k=1}^{N(t)} X_k, \quad t \geq 0.$$

The process $N(t)$ counts the number of jump times upto t and the process $S(t)$ is the value of a sum of $N(t)$ iid increments with the law of X. Both these processes are piecewise constant functions of time. By convention, these processes are chosen to be continuous from the right at the jump locations.

Compound Poisson process are easy to simulate and to analyze. Despite the fact that the state space is $\mathbb{R}$, the generator of a compound Poisson processes may be computed using Fourier transforms (characteristic functions) as follows.

Let $\varphi(\xi) = \mathbb{E}(\exp(\mathrm{i}\xi X))$ denote the characteristic function (Fourier transform) of a random variable with the law of X_k. In order to compute the law of $S(t)$ at any fixed t, it is enough to compute $\mathbb{E}(\exp(\mathrm{i}\xi St))$ for all ξ. We condition on the values of $N(t)$ to obtain

$$\mathbb{E}\left(e^{\mathrm{i}\xi S_t}\right) = \sum_{m=0}^{\infty} \mathbb{E}\left(e^{\mathrm{i}\xi S_t}|N(t) = m\right) \mathbb{P}(N(t) = m)$$

Since the increments in jump times are iid Exp(λ) random variables, for each $t > 0$, $N(t)$ is a Poisson random variable with

$$\mathbb{P}(N(t) = m) = \frac{(\lambda t)^m}{m!} e^{-\lambda t}.$$

Further, since the jump values X_k are iid

$$\mathbb{E}\left(e^{\mathrm{i}\xi S_t} | N(t) = m\right) = \mathbb{E}\left(e^{\mathrm{i}\xi \sum_{j=1}^m X_j}\right) \underbrace{=}_{\text{iid}} \prod_{j=1}^m \mathbb{E}\left(e^{\mathrm{i}\xi X_j}\right) = (\varphi(\xi))^m.$$

We substitute in equation (6.5) to find that

$$\mathbb{E}\left(e^{\mathrm{i}\xi S_t}\right) = e^{-\lambda t} \sum_{m=0}^{\infty} \frac{(\lambda t)^m}{m!} (\varphi(\xi))^m = e^{-\lambda t(1-\varphi(\xi))}.$$

6.6 Lévy processes

Compound Poisson processes allows us to generate *all* continuous time Markov processes *with independent increments*. This approach was first investigated by the pioneering French probabilist Paul Lévy, and the resulting processes are called Lévy processes in his honor. The main idea is to begin with the compound Poisson process, but then to go further by observing that the sum of two independent compound Poisson processes is again a compound Poisson process, but with a new rate and jump law (check!). Similarly, we can always add a drift of the form ct for fixed c to $S(t)$ without changing the fact that $S(t)$ has independent increments. This is useful since it allows us to build richer compound Poisson processes by 'layering' jumps while subtracting a compensating drift to keep things centered.

It would take us too far afield to describe these ideas in detail, but it is instructive to construct Brownian motion as a scaling limit of compound Poisson processes, since this provides some feeling for the theory.

Suppose $\{X_k\}$ are iid with $\mathbb{E}(X_k) = 0$ and $\mathbb{E}(X_k^2) = \delta^2 > 0$ and the jump times $\{T_k\}$ are Exp(λ) as above. We consider the limit $\lambda \to \infty$, $\delta^2 \to 0$ such that $\lambda\delta^2 = 1$. Since

$$\varphi(\xi) = \mathbb{E}(e^{\mathrm{i}\xi X}) = 1 - \frac{\delta^2\xi^2}{2} + \ldots, \quad \xi \to 0,$$

we find that

$$\lim_{\lambda\to\infty} \lambda t(1 - \varphi(\xi)) = -\lambda t \left(1 - \left(1 - \frac{\delta^2\xi^2}{2}\right) + \ldots\right) = \frac{t\xi^2}{2}\left(\lambda\delta^2\right) \to \frac{t\xi^2}{2}.$$

Since this calculation holds for all $t > 0$ and $\xi \in \mathbb{R}$, we have actually shown that

$$\lim_{\lambda\to\infty,\ \delta\to 0,\ \lambda\delta^2=1} \mathbb{E}\left(e^{\mathrm{i}\xi S_t}\right) = e^{-\frac{t\xi^2}{2}} = \mathbb{E}\left(e^{\mathrm{i}\xi B_t}\right), \quad \xi \in \mathbb{R},$$

where B_t is standard Brownian motion. Intuitively, this means that we have approximated Brownian motion by a random walk with really small jumps. What's different from our earlier calculations is that the jump times are also random, but this doesn't affect the limit.

This calculation also provides the generator of Brownian motion. The main conceptual shift is to think of the generator as a linear operator acting on test functions, rather than a matrix. We say that a function $f : \mathcal{S} \to \mathbb{R}$ lies in the domain of the generator $\mathcal{A}$ of a Markov process X_t when the limit

$$\lim_{t\to 0} \frac{1}{t} \left(\mathbb{E}(f(X_t^x)) - f(x)\right) := \mathcal{A}(f)(x)$$

exists. Here X_t^x denotes the Markov process X_t conditioned to start at $X_0 = x$.

When the state space is finite, every function $f : \mathcal{S} \to \mathbb{R}$ lies in the domain of the generator. When X_t is a compound Poisson process, for each $\xi \in \mathbb{R}$, the function $f(x) = e^{i\xi x}$ lies in the domain of the generator. The calculation above then shows that

$$\mathcal{A}(e^{i\xi x}) = -\lambda(1 - \varphi(\xi))e^{i\xi x}.$$

Thus, the functions $e^{i\xi x}$, $\xi \in \mathbb{R}$ are eigenfunctions of $\mathcal{A}$ with eigenvalues $-\lambda(1 - \varphi(\xi))$. In the scaling limit, we see that the generator of Brownian motions acts as follows:

$$\mathcal{A}e^{\mathrm{i}\xi s} = -\frac{\xi^2}{2} e^{\mathrm{i}\xi s}.$$

It should then come as no surprise that the generator of Brownian motion is

$$\mathcal{A} = \frac{1}{2}\frac{d^2}{dx^2}$$

and that the forward equation for Brownian motion is the heat equation

$$\partial_t p = \frac{1}{2}\frac{\partial^2 p}{\partial x^2}.$$

Chapter 7

Character recognition and Gibbs distributions

7.1 The Bayesian paradigm for computer vision

Character recognition is perhaps the simplest cognitive task in vision. The benchmark problem in this area is as follows: one is given a black and white image that contains a (pixelized) image of handwritten characters from a fixed set (see Figure 7.1.1). The task is to automatically associate an image with a character.

Figure 7.1.1: Handwritten digits from the MNIST database.

The Bayesian approach to character recognition requires the construction of a probabilistic model for the signals. As in our approach to music, we decompose the model into a discrete part (a parse tree that describes how pen strokes fit

together in a character) and a continuous part (contours that model a single pen stroke). We will use Gibbs distributions and stochastic differential equations to model contours.

The study of character recognition provides some appreciation for the subtlety of the cognitive tasks in vision. The space of visual signals –and the way humans process it – is very rich and we are far from a holistic understanding of the problem. It is first necessary to develop stochastic models for specific tasks such as edge detection, depth perception, face recognition, and the perception of color and texture. Simple examples reveal that images carry a subtle hierarchical decomposition of structure – a grammar in our linguistic metaphor– but what exactly the rules of such a grammar may be remain unelucidated. The study of stochastic models for images also leads into fascinating questions regarding universal statistical regularities in natural images. The later chapters of [12] explore these ideas.

The weakness of the Bayesian approach is that while it provides a principled aproach to computer vision, modern technology 'solves' many of these problems with a completely different paradigm: the training of neural networks to solve classification problems. For this reason, this chapter also marks an end of our use of the Bayesian paradigm. In the rest of these notes, we consider neural networks, alternative ideas in statistical learning theory, and the mathematical infrastructure of numerical linear algebra and optimization, in a way that provides some insight into deep learning. A central challenge in this area is to understand why deep learning works as well as it does. For example, does it have a Bayesian foundation?

7.2 The shape of a circle

We formalize the problem of character recognition as follows. Assume given a finite alphabet $\mathcal{A}$, such as the numerals $\{0, \ldots, 9\}$ or the Latin alphabet $\{a, b, \ldots, z\}$. The alphabet is an abstract set of symbols that is distinct from our representation of them as symbols on paper or a screen. The visual representation of the alphabet is a font, $\mathcal{F}$. Each element $f(a) \in \mathcal{F}$, $a \in \mathcal{A}$, is an idealized geometric shape consisting of a collection of curves linked through a parse tree. More precisely, $f(a)$ is a geometric algorithm to

1. Construct a parse tree that connects a finite collection of curves denoted $\gamma_{a,j}$, $j = 1, \ldots, m(a)$ for each $a \in \mathcal{A}$. The number of contours $m(a)$ and the way that they are bound together is distinct for each letter $a \in \mathcal{A}$.

2. A map $u_{j,a} : \gamma_{j,a} \to \mathbb{R}^2$ that embeds each curve into the plane. (An embedding of a curve is a smooth map that has a non-vanishing derivative at each point and never intersects itself.)

For simplicity, we have discounted the thickness of the contours in this model, though all practical fonts include a thickness that scales in proportion to the length of the contours.

If these rules sound excessively formal, it is only because writing symbols such as the numerals or Latin alphabet has become second nature to us after years of practice. One acquires a better appreciation of the complexity of writing when learning a foreign script, designing a font, or just admiring the patience of a kindergarten teacher instructing young children to write.

These definitions separate the task of developing a probabilistic model for character recognition into two parts: a discrete, or combinatorial, part that describes the parsing and gluing rules that hold the contours $\gamma_{j,a}$ together; and a continuous part consisting of a prior for each individual curve $\gamma_{j,a}$ in the font. Modeling the combinatorial aspect is (mathematically) straightforward, at least for small character sets, such as the numerals or the letters, since it may be solved by listing the combinatorial rules for each symbol (i.e learning an alphabet with all its quirks). While this task becomes more tedious for large writing systems – for example, the Japanese Kanji system contains more than 50,000 characters– it can still be solved by exhaustive enumeration. Computer fonts for all modern writing systems include such rules, since these are necessary to represent fonts on a computer screen and for the code to drive printers.

Mathematically, the more subtle part of the problem is to develop priors on curves. To understand the issue, pick the letter 'o'. This is the simplest letter combinatorially, since it consists of a single curve γ which we idealize as the unit circle $S^1 = \{(x, y) \in \mathbb{R}^2 \,|x^2 + y^2 = 1\}$. Now make ten copies of this letter on a sheet of paper. You will see that while each of these is a perfectly recognizable 'o' not a single one of them is a perfect circle!

There is a delicate mathematical problem lurking in the background: what does a 'typical' embedding of a circle into $\mathbb{R}^2$ look like? This is the simplest version of a central probabilistic question in the Bayesian approach to vision: how does one construct tractable priors on geometric continua in accordance with the natural symmetries of a problem?

In order to address this question, let us consider some of the probabilistic models we have used:

1. Markov chains (to model text on a finite alphabet).
2. Stationary Gaussian functions (to model a single note of music).
3. Continuous time pure jump Markov processes (to model a musical score).

These models implicitly relied on the idea of a linear signal, i.e. a signal with a clear 'direction of time'. While each curve in a character can be similarly parametrized, it is better to view the problem of designing priors from first principles, since this approach also applies to two and three dimensional geometries.

7.3 Gibbs distributions revisited

Let us first recall the structure of Gibbs distributions. We assume given a state space $\mathcal{S}$ and an energy function $E : \mathcal{S} \to \mathbb{R}$ (energy function). Assume at first

that $|\mathcal{S}|$ is finite. Then each state $s \in \mathcal{S}$ has the probability

$$p_\beta(s) = \frac{1}{Z_\beta} e^{-\beta E(s)}, \quad Z_\beta = \sum_{s \in \mathcal{S}} e^{-\beta E(s)}.$$

where $\beta > 0$ is a fixed parameter.

When using Gibbs distributions as models in computer vision, we often use heuristics from physics to associate energies to the states. Let us first illustrate this interplay with an important example. [1]

7.3.1 Ising models in vision

Example 20 (Ising model on a graph)**.** Consider a graph $G = (V, E)$ and the state space $\mathcal{S} = \{s : V \to \{-1, 1\}\}$. Let $|G|$ denotes the number of vertices in G, so that $|\mathcal{S}| = 2^{|V|}$.

A function $s \in \mathcal{S}$ is called a spin configuration, since this model was introduced to study ferromagnetism. We say that $x \sim y$, or that x and y are neighbors, if x and y are linked by an edge. The energy of a spin configuration is defined as follows:

$$E(s) = -\alpha \underbrace{\sum_{x \in V} s(x)}_{\text{total magnetization}} + \underbrace{\sum_{x \sim y} (s(x) - s(y))^2}_{\text{penalizes gradients}}$$

Here α is a parameter called the applied field that biases the energy based on the total magnetization, as shown in the figure. The second term is called the interaction energy and it prefers neighbors to be aligned.

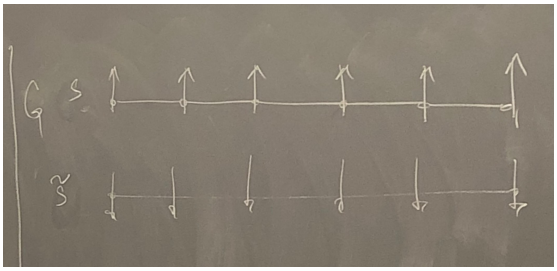

Figure 7.3.1: $E(s) = -\alpha|V|$, $E(\tilde{s}) = \alpha|V|$

Example 21 (Ising-like model for image segmentation)**.** Image segmentation is the task of labeling each pixel in an image with semantic attributes. In the

[1] The reader unfamiliar with statistical physics should note that as discussed in Chapter 1.1, physical heuristics while historically useful, are not logically necessary here. Gibbs distributions may be defined on *any* state space $\mathcal{S}$ provided one has a prior distribution on $\mathcal{S}$, a relative entropy with respect to the prior, and an energy $E : \mathcal{S} \to \mathbb{R}$. In discrete systems, the prior is the uniform distribution which is why the partition function takes the form it does. When $\mathcal{S}$ is $\mathbb{R}^n$ a natural prior is often a Gaussian distribution.

simplest setting, this is the task of classifying an image into two parts, labeled ± 1, (these may be a classification of each pixel as 'background' or 'foreground' for example).

This problem was formalized as an Ising model in a seminal paper by Geman and Geman [**?**]. Their formulation goes as follows. Assume that our graph is a square 2D grid with N vertices a side. Assume given an image, modeled as a greyscale function $I : V \to [-1, 1]$. Now modify the applied field in the Ising model so that it is aligned with the background image to obtain the energy

$$E_I(s) = -\alpha \underbrace{\sum_{x \in V} I(x)s(x)}_{\text{favors alignment with background}} + \underbrace{\sum_{x \sim y} (s(x) - s(y))^2}_{\text{penalizes edges}}.$$

The task of image segmentation now becomes one of sampling from the Gibbs distribution from this model. These samples reveal striking features, in particular the appearance of a phase transition at a critical temperature separating order from disorder (see Figure 7.3.2).

The above examples reveal the utility of the Gibbs distribution. Let us now develop similar ideas to model stochastic contours such as a typical 'o'. We need:

1. A systematic way to assign energies to curves (this is the theory of continuum mechanics).

2. A systematic way to define integration in infinite dimensions (Gaussian process theory).

7.3.2 The Gibbs distribution for Brownian motion

Before considering the general theory, let us consider the simplest Ising-like model that is inspired by the examples above. We will consider a Gibbs distribution for the graph of a curve $u : [0, 1] \to \mathbb{R}$ that is pinned at its left endpoint, i.e. $u(0) = 0$ with an energy that favors the flat configuration $u(x) = 0$, for all $x \in [0, 1]$. Let's first do this for the following discretization.

Fix an integer $N > 0$, consider the grid $0 = x_0 < x_1 < \cdots < x_N = 1$ with $x_{j+1} - x_j = 1/N$. Consider a function $u(x)$ defined at the grid points and define the Ising-like energy:

$$E(u) = N\left((u_1 - u_0)^2 + \cdots + (u_N - u_{N-1})^2\right).$$

Here we have used the notation $u_j = u(x_j)$. We have removed the applied field altogether, so the energy is minimized when $u_0 = u_1 = \ldots = u_N = 0$. Unlike the Ising model, our random variables u_j are real valued. The Gibbs distribution has the pdf

$$p_\beta(\underline{u})\, d\underline{u} = \frac{1}{Z_{N,\beta}} e^{-\frac{N\beta}{2}\sum_{j=0}^{N-1}(u_{j+1}-u_j)^2}\, du_1 du_2 \ldots du_N, \tag{7.3.1}$$

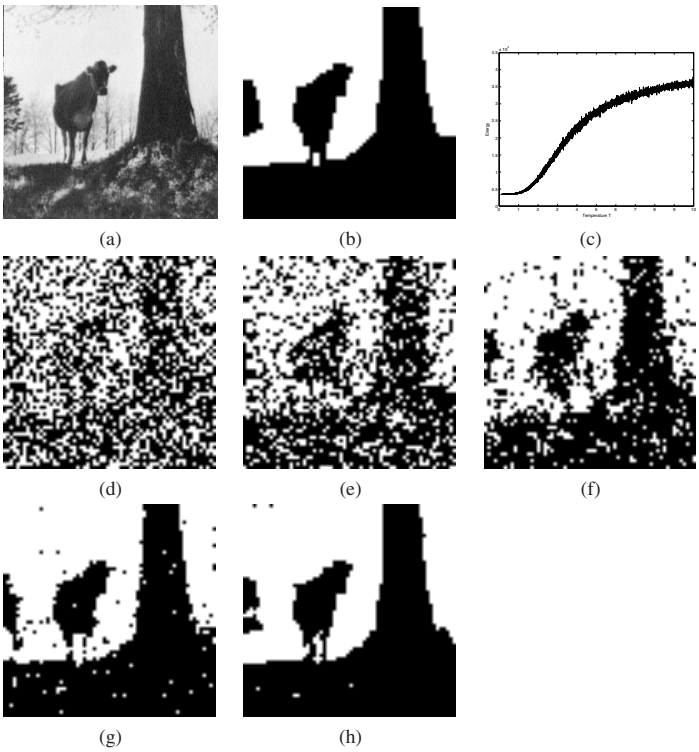

Figure 7.3.2: Samples at different temperatures from a Gibbs distribution for the Ising model of images. The image in the upper-left corner is the original greyscale image; the image next to it is obtained by thresholding that yields a black-and-white image (which may be normalized to the values {0,1}). This background image is used to define an energy, and thus Gibbs distribution, associated to the image. Images (d)–(h) show samples from the associated Gibbs distribution at temperatues 10, 4, 2.5, 1.5 and 1 respectively, with $\alpha = 4$ in the energy. As the temperature decreases, a phase transition separating 'order and disorder' takes place near temperature 2.5.

where $\underline{u}$ is an abbreviation for $(u_1, \ldots, u_N)$, and the partition function is defined by

$$Z_{N,\beta} = \int_{\mathbb{R}^N} e^{-\frac{N\beta}{2}\sum_{j=0}^{N-1}(u_{j+1}-u_j)^2}\, du_1 du_2 \ldots du_N. \tag{7.3.2}$$

Observe that we must integrate over an N-dimensional space since the random function $u(x)$ is determined by its values $u_1, \ldots, u_N$ at the N points $x_1, \ldots, x_N$ respectively. Note also that the partition function $Z_{N,\beta}$ may be computed explicitly, since it is a Gaussian density. Indeed, change variables to

$$v_1 = u_1, \quad v_2 = u_2 - u_1, \quad \ldots, v_N = u_N - u_{N-1}, \tag{7.3.3}$$

whose inverse is

$$u_1 = v_1, \quad u_2 = v_1 + v_2, \quad \ldots u_N = v_1 + v_2 + \ldots + v_N. \tag{7.3.4}$$

The Jacobian determinant of the linear transformations in (7.3.3) and (7.3.4) is 1 since they are represented by lower and upper triangular matrices respectively with 1's on the diagonal. Therefore, changing variables using (7.3.3),

$$Z_{N,\beta} = \int_{\mathbb{R}^N} e^{-\frac{N\beta}{2}\sum_{j=1}^{N} v_j^2}\, dv_1 dv_2 \ldots dv_N = (2\pi N\beta)^{N/2}. \tag{7.3.5}$$

In summary, even though we began with a Gibbs distribution for an Ising-type model, our probability measure (7.3.1) is nothing but the pdf of a collection of N iid Gaussian random variables. Indeed, $u_k = v_1 + \ldots v_k$ is simply a sum of iid Gaussians with variance $1/N\beta$.

Let us now consider the *continuum limit* as $N \to \infty$. The normalization factor of N in the energy is introduced so that we may take the continuum limit. More precisely, we view the difference of nearest neighbors in the vector $\underline{u} \in \mathbb{R}^N$ as the derivative of a piecewise linear function u_N, with

$$u_{j+1} - u_j = \frac{1}{N} u'(x), \quad x = j/N.$$

In the limit $N \to \infty$, we obtain the energy

$$E[u] = \int_0^1 \left(u'(x)\right)^2 dx. \tag{7.3.6}$$

Here we use the notation $[\cdot]$ to remind ourselves that E is a *functional* (i.e. a function of functions $u : [0,1] \to \mathbb{R}$). This is our simplest example of a continuum model, the Dirichlet energy.

Thus, our well-defined Gaussian pdf (7.3.1), has the suggestive, but formal, limit

$$p_\beta(u)\, d\underline{u} = \frac{1}{Z} e^{-\beta E[u]}\, d\underline{u}, \quad E[u] = \int_0^1 \left(u'(x)\right)^2 dx. \tag{7.3.7}$$

This expression is an example of a *path integral*. The space over which we integrate is a space of functions $u : [0,1] \to \mathbb{R}$ each of which has energy $E[u]$.

But how precisely does one give meaning to integration over this space of functions? That is, what does $d\underline{u}$ even mean since it is (formally) an infinite product $\prod_{j=1}^{\infty} du_j$?

We can resolve this issue by approaching the problem in a different way. Equations (7.3.4) and (7.3.5) show that the random variables $\{u_k\}_{k=1}^N$ have the same pdf as the random variables

$$U_N(x) = \frac{1}{\sqrt{N\beta}} \sum_{j=1}^{k} X_j, \quad x = \frac{k}{N}, 1 \leq k \leq N, \tag{7.3.8}$$

where $\{X_j\}_{j=1}^N$ are iid standard normal random variables. The scaling limit of this function is nothing but a Brownian motion with variance β. In this limit, the space and probability measure over which we integrate are well-defined.

The fact that the simplest Ising-type model has led us back to Brownian motion is neat, but we're not out of the woods. Physicists often view Brownian motion as the Gibbs distribution for curves with the Dirichlet energy (7.3.6). Unfortunately, equation (7.3.7) has two sources of difficulty. Even if we use the Wiener measure to make sense of integration over an infinite-dimensional space (the space of continuous functions), it turns out that every Brownian motion is nowhere differentiable and has $E[u] = \infty$!

7.3.3 Energies for contours

This example reveals a serious problem in defining Gibbs distributions for continuous curves, which only becomes worse as we consider two-dimensional random fields such as membranes. Resolving the divergences above is a subtle problem in both physics and mathematics called *renormalization.*

From the standpoint of modeling, we can dodge these issues, though not in an entirely satisfactory way, by using the following fixes:

1. Work with a large, but finite, discrete model.
2. Use SDE theory to build priors instead of using the Gibbs distribution.
3. Introduce penalty terms in the energy that force the curve to be smooth.

As an example of the penalty method, consider the energy

$$E[u] = \alpha_1 \int_0^1 (u')^2 \, dx + \alpha_2 \int_0^1 (u'')^2 \, dx.$$

Our constructions of lacunary series for nowhere differentiable functions are easily adapted to create functions $u(x)$ such that $|u''|$ is divergent, but u' is continuous. Thus, the addition of the second derivative term in the energy forces the curves to be smoother than those with just finite Dirichlet energy.

The penalty method is a quick fix. When possible, it is preferable to design energy functions using the geometry of space curves. The main examples of this type for contours are energies that penalize curvature.

In order to introduce this model, let us recall the formulas for space curves from calculus. Recall that the unit tangent vector $\underline{t}$ and curvature κ of a parametrized curve $\gamma : [0,1] \to \mathbb{R}^2$ with coordinates $x \mapsto \gamma(x) = (\gamma_1, \gamma_2)(x)$ are defined by

$$\underline{t}(x) = \frac{\gamma'(x)}{|\gamma'(x)|}, \quad \frac{d\underline{t}}{ds} = \kappa\, \underline{n}, \quad ds = |\gamma'(x)|dx.$$

Here ds denotes the arc length of the curve, κ is the curvature and $\underline{n}$ is the unit normal. A fundamental example of a *geometric* energy function is that of *Euler's elastica*:

$$E[u] = \int_\gamma (\kappa)^2 ds, \tag{7.3.9}$$

where we integrate over the length of the curve γ. The above parametrization of the integral is intrinsic (i.e. the curvature is treated as a function of the arc length). The integral may be reduced to an integral on $[0,1]$ for any parametrization $\gamma : [0,1] \to \mathbb{R}^2$.

Energies such as (7.3.9) are systematically derived in *continuum mechanics* by combining geometric assumptions on deformations with assumptions on material properties. Euler and Bernoulli derived the above energy to describe the deformation of *beams*. The geometric definition of a beam is that it is a curve that is inextensible (i.e. its length does not change as it is deformed) but resists bending (i.e. a change in its curvature from a fixed initial configuration). The fact that the energy is proportional to the square of the curvature relies on an additional geometric assumption and an assumption on the nature of the material that defines the beam.

1. First, the geometric assumption: "fatten" the beam normally by giving it a thickness that is uniform along the inextensible midline. We then assume that plane sections normal to the midline, stay plane during deformation. This allows us to assert that the strain above and below the inextensible midline of the beam is linearly proportional to the curvature of the midline.

2. We assume the material that constitutes the beam is linearly elastic. That is, the stress in the beam is linearly proportional to the strain.

Beam theory is important for civil and mechanical engineers, since it is used to assess the strength of materials when constructing buildings and machines. This example reveals that these theories remain relevant in the era of machine learning. Our beams are 'o"s on a sheet of paper but we may continue to model them with the modified energy

$$E_0[u] = \int_0^{2\pi} (\kappa - \kappa_0)^2 ds$$

which penalized deviations from a circle of radius κ_0^{-1}. Aside from the change of scale and physical parameters, this is still governed by a model introduced by Euler in the 1750s.

We may similarly define the energy for two dimensional objects by using natural geometric quantities such as the area, the perimeter and curvatures. For example, soap films are described by surfaces with the least area for a given boundary contour. This is an example of a theory of membranes. The two-dimensional analogues of beam theory, which penalize curvatures, are known as the theory of plates and shells.

The general theme here is that in order to define a prior on a geometric object (characters, faces, images), we consider Gibbs distributions with energies defined with continuum mechanics. The subtle part of the theory is that these energies, while geometrically natural and supported by continuum mechanics, must be renormalized so that they give rise to well-defined Gibbs distributions on continua. This is an unsolved problem in general. Finally, another approach to the problem of defining stochastic contours is through the formulation of stochastic differential equations to generate contours. Several such examples are discussed in [12, §3.2.2].

7.3.4 Constructing a character

We may summarize the Bayesian approach to character recognition as follows.

1. A stochastic model for each character is designed "by hand". These models use energies from continuum mechanics along with heuristics for binding contours and the inclusion of other features such as thickness.

2. The model relies on a hierarchical decomposition of the character into features. The lowest level is the raw greyscale image (a number in [0,1] for each pixel). The highest level is an abstract character. Such a hierarchical decomposition for the letter 'A' is shown in Figure 7.3.3).

3. Generating samples is much easier than inferring a character. That is given character, it is (relatively) straightforward to sample a greyscale image. However, given a greyscale image we must solve a hierarchical maximization problem to determine the character. This is tricky to implement and the difficulties are similar to those faced when parsing a music score.

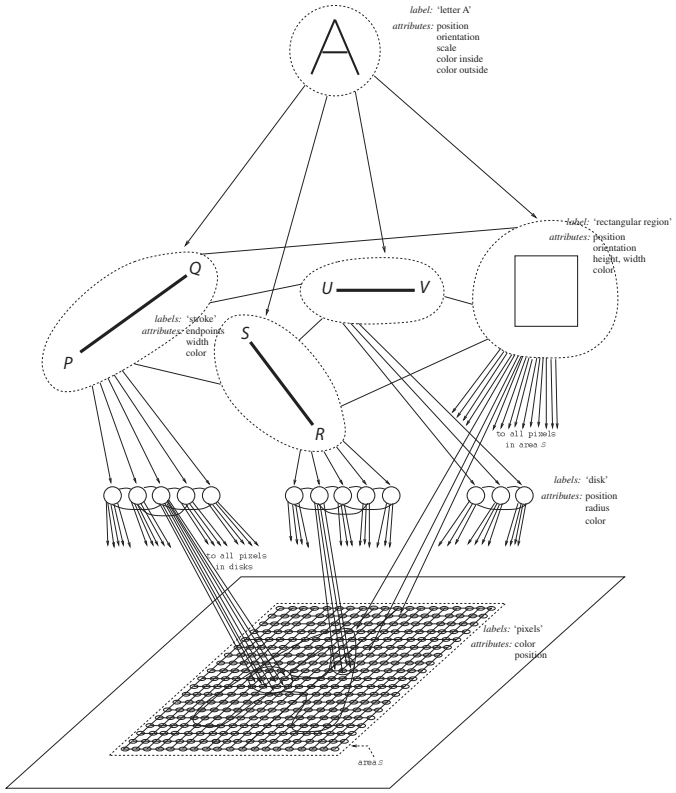

Figure 7.3.3: An example of hierarchical decomposition of characters in the Bayesian approach [12, p.160]. The character A is decomposed into four levels: the character, the strokes that constitute it, maximal disks for each stroke, and finally individual pixels.

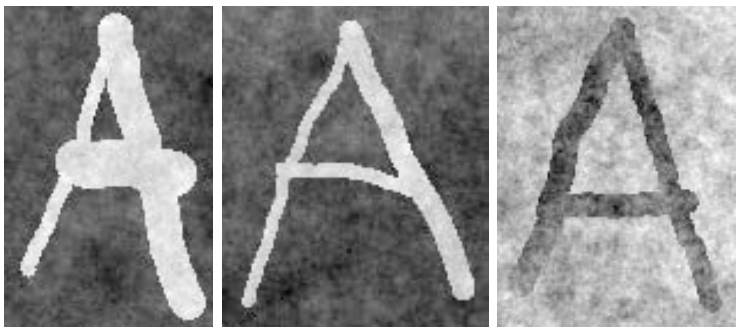

Figure 7.3.4: Samples of the character A using the hierarchical model described above. This figure is taken from [12, p.163]; the details on the generation of this image may be found there.

Chapter 8

Feed forward neural networks

8.1 Introduction

This chapter marks the end of our use of the Bayesian paradigm of pattern theory. In this chapter, we introduce the use of neural networks and the paradigm of statistical learning theory. There are many sources on this material, as well as several software packages that focus on fast implementations of convolutional neural networks (CNNs) for character and handwriting recognition (see [**?**, 16] for applications and theory respectively). Since our emphasis is on the mathematical underpinnings of these ideas, we will first focus on the simpler class of feed forward neural networks in order to introduce the ideas of gradient descent and the backpropagation algorithm. This is followed in the next chapter by an explanation of the modifications in architecture that are necessary for convolutional neural networks, along with some of their biological motivation.

The main advantage of the Bayesian approach is that it is principled. More precisely, the problem of inference and the generation of samples are treated together in a statistically consistent way. The previous chapters provide several examples of such stochastic models. The primary weakness, as seen in Section 7.3.4, is that the construction of priors on curves and membranes is subtle because of divergences caused by integrating on infinite-dimensional spaces. Further, the complexity of hand-designed feature (such as those in Figure 7.3.3) makes the task of inference inefficient.

The use of neural networks for problems of inference relies on a different paradigm, that of *statistical learning theory.* In this paradigm, character recognition is modeled as a classification problem for functions that map a high-dimensional input space to a low-dimensional output space. To be concrete, consider the problem of optical character recognition (OCR) of digits. Here one is given an input greyscale image and the task is to classify the image as a digit in the set $\{0, 1, \ldots, 9\}$. We may model the domain and range of our classifying function as follows: the input space is $[0, 1]^n$, where n is the number of pixels and the greyscale intensities of each pixel are normalized to $[0, 1]$. Let the output

space be $[0,1]$; let m denote the number of distinct characters; and let us further assume that the k-th character corresponds to the interval $[k/m,(k+1)/m)$. [1] Each function $f:[0,1]^n \to [0,1]$ may be used to partition the space $[0,1]^n$ into the *decision regions*, $f^{-1}([k/m,(k+1)/m))$. The decision region for the k-the character is then the set of input images in $[0,1]^n$ that map to the k-th interval of length $1/m$ in $[0,1]$.

Once one adopts the paradigm of statistical learning theory, the main task is to compute a *classifier*. This simply means that we must (somehow) construct a function from raw images to characters, so that when given an image, we may evaluate the value of this function and determine the character.

8.2 What is a neural network?

Neural networks may be thought of informally as "function machines with many knobs". A set of training data is used to adjust these knobs, providing us with a classifier. Let us now describe these "function machines" in greater detail.

A neural network consists of the following elements:

1. *An architecture* (a directed graph organized by depth and width). We use the term neuron, node and vertex in a loosely equivalent manner.
2. *States.* These are normalized values $y_i \in [0,1]$ at each node of the network.
3. *Weights.* These are real parameters denoted w_{ij} (on the edge linking nodes i and j), and θ_i (at each node).
4. *An integrate and fire rule at each neuron.* We assume that all neurons are identical and that there is a sigmoidal function, $g(y)$, which mimics the function of true biological neurons. The example we use is

$$g(x) = \frac{1}{2}\left(1 + \frac{\tanh x}{2}\right). \tag{8.2.1}$$

An example of the architecture of the neural network is shown in Figure 8.2.1. The weights and states are shown in Figure 8.2.2.

The integrate and fire rule at each neuron is as follows. The 'integrate' step involves summing over weighted inputs to a neuron. The 'fire' step is the application of the sigmoid function. We write this as

$$x_i = \sum_j w_{ij} y_j + \theta_i, \quad y_i = g(x_i). \tag{8.2.2}$$

The function $g(x)$ is a 'soft' approximation to the thresholding function $\max\{0,x\}$. The role of the parameter θ_i is to shift the location of the threshold,

[1] This choice is simply for conceptual simplicity. During implementation, intervals $[0,1]^m$, one interval for each character, are a better choice). The integer m should be thought of as much less than n)

i.e. $g(x+\theta) \approx \max\{-\theta, x\}$. The parameters w_{ij} amplify, attenuate or flip the sign of the input.

The description of a neural network as a "function machine" is as follows. Denote the values at the input and output layer as $X = (X_1, \ldots, X_n) \in [0,1]^n$ and $Y_1, \ldots, Y_m$ respectively. Let us also use W to denote the set of all weights θ_i and w_{ij}. Then the neural network is simply a function

$$Y = F(X; W), \quad X \in [0,1]^n, \quad Y \in [0,1]^m. \tag{8.2.3}$$

The integrate and fire model for the operation of an individual neuron is a caricature of the operation of biological neurons. Biophysical models of electrical activity in neurons were studied by Hodgkins and Huxley in 1952. They introduced a system of nonlinear differential equations to model the propagation of signals in the squid giant axon (a particularly large neuron). The model in equation (8.2.2) is a drastic simplification of these ideas. The sigmoid arises as a traveling wave solution to the partial differential equation

$$v_t + vv_x = 0, \quad -\infty < x < \infty, \quad t > 0. \tag{8.2.4}$$

Traveling waves are solutions of the form

$$v(x,t) = w(x - ct), \tag{8.2.5}$$

where the parameter c is the speed of the wave (thus, the profile w is steady in a frame of reference that moves with the wave). Substitution of the form of v in equation (8.2.4) gives an ordinary differential equation for w. The function g is the solution to this ordinary differential equation with the limits 0 and 1 at $\pm\infty$.

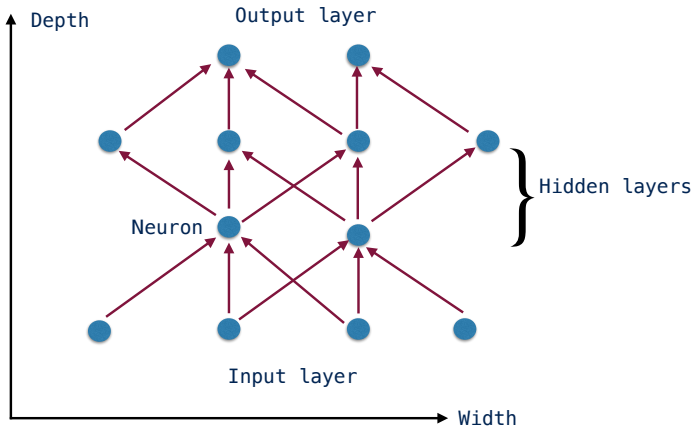

Figure 8.2.1: The architecture of a neural network

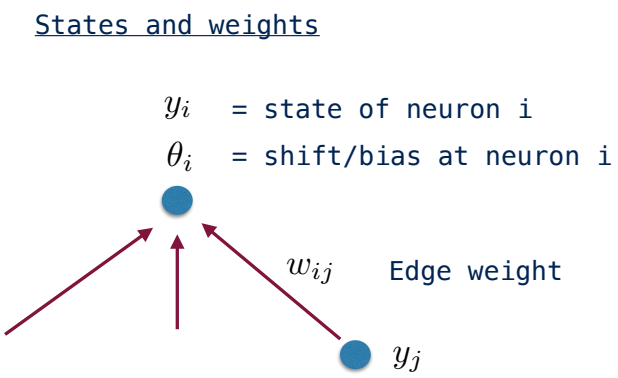

Figure 8.2.2: Weights leading into neuron i.

8.3 Supervised learning

Let us now assume that the architecture is fixed. The functions that may be produced with a given architecture are determined by the choice of weights. These weights are determined by training the network on a set of training data, denoted $(X^{(p)}, Y^{(p)})$, $1 \leq p \leq P$, that index known input and output relations on a data set of size P.

The weights are chosen by minimzing a loss function on the training set. A typical loss function has the form

$$L(W) = \sum_{p=1}^{P} |Y^{(p)} - F(X^{(p)}, W)|^2. \tag{8.3.1}$$

The weights for the network are determined by

$$W_* = \text{argmin}_W L(W). \tag{8.3.2}$$

But how does one minimize this function? A key idea in the success of neural networks is that the loss function is best minimized through gradient descent. [2] That is, we may solve the differential equations

$$\dot{w}_{ij} = -\frac{\partial L}{\partial w_{ij}}, \quad \dot{\theta}_i = -\frac{\partial L}{\partial \theta_i}, \tag{8.3.3}$$

in order to drive the weights $W(t)$ to a minimizer of a loss function $L(W)$.

Since this is an important idea, we switch notation slightly to illustrate the general principle. Suppose $E : \mathbb{R}^n \to \mathbb{R}$ is a smooth function (often called an energy) and consider the gradient flow

$$\dot{x} = -\nabla E(x), \quad x \in \mathbb{R}^n. \tag{8.3.4}$$

[2] In this book, we usually use the term gradient descent to refer to gradient flows in continuous time.

Now evaluate the change in energy along a solution to (8.3.5). We find that

$$\frac{d}{dt}E(x(t)) = -\nabla E(x)\cdot\dot{x} = -|\nabla E(x)|^2 \leq 0. \tag{8.3.5}$$

Thus, the energy decreases along solutions to (8.3.5) and the decrease is strict unless $x(t)$ is an equilibrium (i.e. $x(t)$ is a constant, say x_*, independent of time and $\nabla E(x_*) = 0$).

8.4 The backpropagation algorithm

The conceptual structure of gradient descent is best seen in continuous time. In practice, we approximately solve (8.3.3) by discretization. What this means is that we fix a step size ε and obtain a set of discrete updates

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \varepsilon\frac{\partial L}{\partial w_{ij}}(W^{(k)}), \quad \theta_i^{(k+1)} = \theta_i^{(k)} - \varepsilon\frac{\partial L}{\partial \theta_i}(W^{(k)}, \tag{8.4.1}$$

stopping when the gradients become sufficiently small.

The effective implementation of gradient descent to train the neural network relies fundamentally on the fact that we can compute gradients fast using the hierarchical architecture of the network. This is the backpropagation algorithm.

Let us first illustrate this hierarchical structure in a simple network. We then state the ideas in generality. Consider the network with depth two shown in Figure 8.4.1 and the loss function

$$L(w_0, w_1) = \frac{1}{2}(y_2 - Y)^2,$$

where Y is a fixed value. Observe that the states depend only on the weights in the layers below them; that is $y_2 = y_2(w_0, w_1)$, but $y_1 = y_1(w_0)$. When the gradient of L is computed using the chain rule we find that

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial w_1}, \quad \frac{\partial L}{\partial w_0} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial y_1}\frac{\partial y_1}{\partial w_0}.$$

These formulas show that the gradient of an arbitrary loss function can be determined once we have computed the derivatives

$$\frac{\partial y_2}{\partial w_1}, \quad \frac{\partial y_2}{\partial y_1}, \quad \frac{\partial y_1}{\partial w_0}. \tag{8.4.2}$$

Therefore, let us look at these terms more carefully. Recall that g is the sigmoidal function from equation (8.2.1). It is easy to check that g satisfies the differential equation [3]

$$g' = g(1-g). \tag{8.4.3}$$

We now compute the derivatives (8.4.2) using equation (8.4.3) to find

[3] This may look like magic, but this differential equation is 'built into' the theory of neural networks. It can be traced to the modeling assumption of Hodgkins and Huxley that neurons work through the propagation of electrical pulses, the simplification (8.2.4) of the Hodgkins-Huxley idea, and equation (8.2.5) satisfied by the traveling wave profiles.

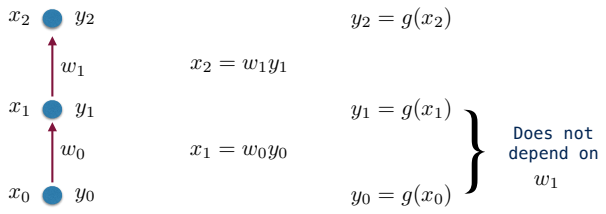

Figure 8.4.1: An illustration of the hierarchical structure of backpropagation

$$\frac{\partial y_2}{\partial w_1} = y_2(1-y_2), \quad \frac{\partial y_2}{\partial y_1} = y_2(1-y_2)w_1, \quad \frac{\partial y_1}{\partial w_0} = y_1(1-y_1)w_0.$$

Thus, the gradient of the loss term is

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y_2} y_2(1-y_2), \quad \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y_2} y_1(1-y_1)y_2(1-y_2)w_0 w_1. \tag{8.4.4}$$

In summary, the gradient of the loss function depends on its derivative with respect to the state variables at the highest level, along with a computation of Jacobian derivatives.

The general structure of the above formula, which is the main observation that underlies the backpropagation algorithm, is that these derivatives may be computed sequentially, going *down* the network from the top to the bottom. That is, the order of the computation is inverse to the dependence of the state variables y_i on the weights. Let us make this precise, using the notation shown in Figure 8.4.2.

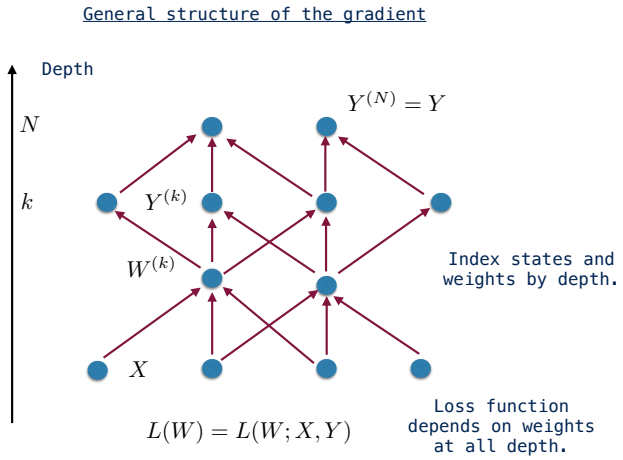

Figure 8.4.2: The hierarchical structure of backpropagation in a general network. The index k denotes the depth. All the weights and states at depth k are indexed by the depth.

The output of the network is built upwards, level by level. We write this in the form

$$Y^{(k)} = G^{(k)}(W^{(k)}, Y^{(k-1)}), \quad 1 \leq k \leq N. \tag{8.4.5}$$

As seen in the equation (8.4.4), the gradient of the loss function is determined by the Jacobian derivatives

$$\frac{\partial G^{(k)}}{\partial W^{(k)}} \quad \text{and} \quad \frac{\partial G^{(k)}}{\partial Y^{(k)}}. \tag{8.4.6}$$

These are polynomials that are quadratic in Y and linear in W. For large problems, it is more efficient to compute the Jacobian derivatives on the fly rather than to store the values. Once these derivatives are known, the gradient of the loss function may be computed as follows. Let us denote

$$W = \left(W^{(0)}, W^{(1)}, \ldots, W^{(N-1)}\right), \quad \nabla L = \left(\frac{\partial L}{\partial W^{(0)}}, \frac{\partial L}{\partial W^{(1)}}, \ldots, \frac{\partial L}{\partial W^{(N-1)}}\right). \tag{8.4.7}$$

It then follows from (8.4.5) that

$$\frac{\partial L}{\partial W^{(k)}} = \frac{\partial L}{\partial Y^{(k)}} \frac{\partial G^{(k)}}{\partial W^{(k)}}, \quad \frac{\partial L}{\partial Y^{(k-1)}} = \frac{\partial L}{\partial Y^{(k)}} \frac{\partial G^{(k)}}{\partial Y^{(k-1)}}. \tag{8.4.8}$$

Both the above terms are products of matrices; the indices of the terms in the matrix have been suppressed in order to make the hierarchical structure apparent. The term

$$\frac{\partial L}{\partial Y^{(k)}}$$

is obtained from computations at level $k+1$. Given this term, the other two terms on the right hand side of equation (8.4.8),

$$\frac{\partial G^{(k)}}{\partial W^{(k)}} \quad \text{and} \quad \frac{\partial G^{(k)}}{\partial Y^{(k-1)}},$$

are given by the Jacobians at level k. In this manner, we compute the gradient of the loss function with respect to the weights at level k, propagating down the network layer by layer.

8.5 Why neural nets work

Now that the notion of a neural network and its use in supervised learning has been established, let us return to the conceptual notion that neural networks are "function machines". [4]

Much of math relies on building complicated functions from simpler ones. For example, we first learn about polynomials, we then use polynomials to understand Taylor series, trigonometric polynomials, and finally Fourier series.

[4] The title and content of this section follow [10].

While these examples are familiar, it is not necessary to use such specific formulae to construct functions. Another method of building smooth functions is by adding together "bump" functions. This is done in the following way. First we construct a smooth cut-off function, such as

$$\varphi(x) = \begin{cases} e^{-1/x^2}, & x > 0 \\ 0, & x \leq 0. \end{cases} \tag{8.5.1}$$

This specific form is chosen so that the function φ is infinitely differentiable at zero (this is a calculation you should attempt). We may then rescale and shift φ, and take linear combinations and products of such functions. Thus, for example the function

$$\psi_\theta(x) = 1 - \frac{1}{2}\left(\varphi(\theta x) + \varphi(-\theta x)\right), \tag{8.5.2}$$

depending on the parameter θ, takes approximately equal to 1 in the neighborhood $|x| \ll \theta^{-1}$ and approaches 0 when $|x| \gg \theta^{-1}$. Similarly, the product $\varphi(x)\varphi(1-x)$ provides a function which vanishes outside the interval $[0, 2]$.

Specific formulae such as (8.5.1) are useful for our understanding, but they are not necessary for the idea of constructing bump functions. What matters simply is that we have a "machine" to construct bump functions. We will show that a simple neural network with two hidden layers, described by the motif shown in Figure 8.5.1 can build bump functions. Increasing the depth and width of the network has the effect of building more complex functions. This motif constructs bump functions in stages, as shown in Figures 8.5.2–8.5.9.

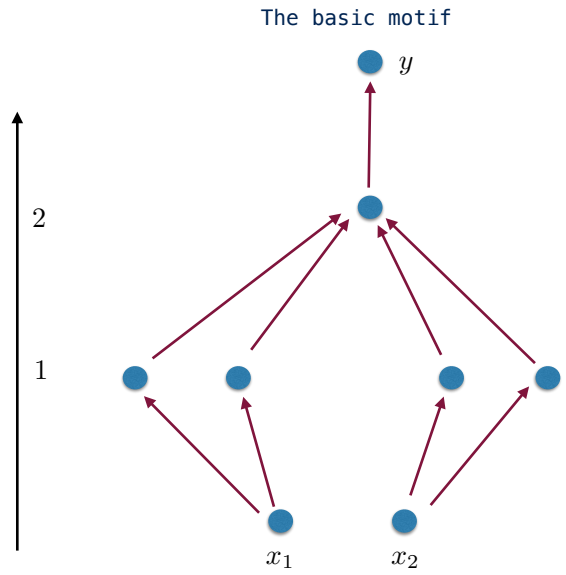

Figure 8.5.1: A neural network of depth two that builds a bump function on $\mathbb{R}^2$.

.

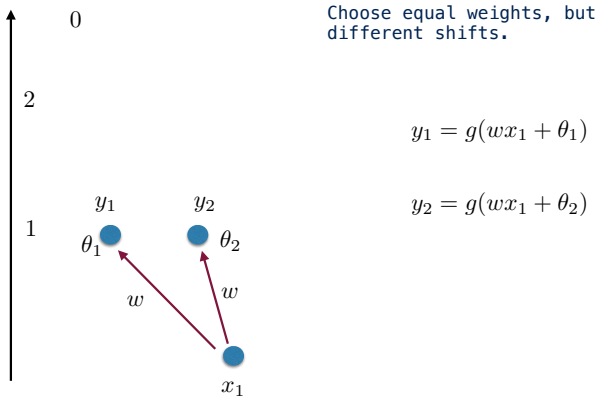

Figure 8.5.2: Each branch of the network at depth one, is used to build two sigmoids shifted relative to one another.

.

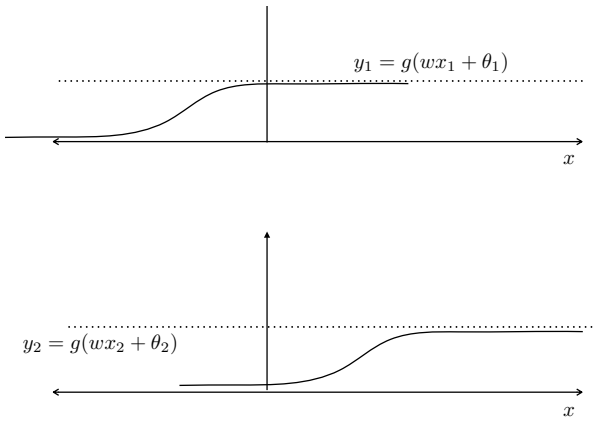

Figure 8.5.3: The role of the shift.

.

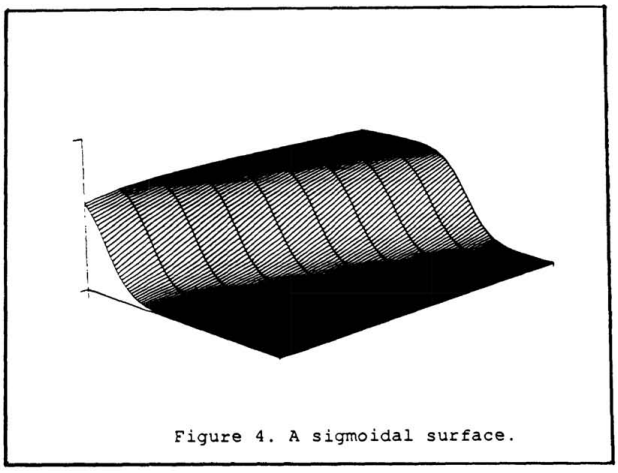

Figure 8.5.4: The sigmoidal surface corresponding to the function y_1 in Figure 8.5.2. The function y_2 is similar. Image from [10].

.

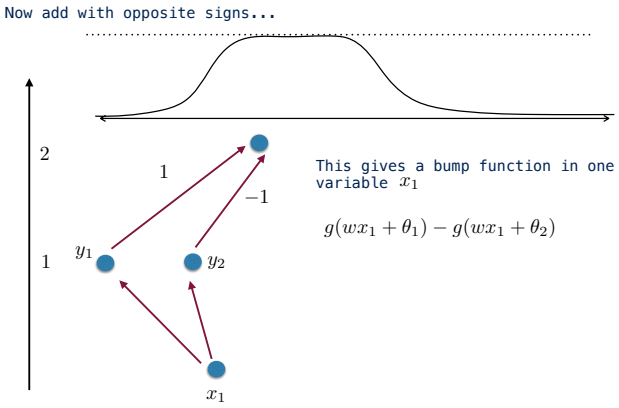

Figure 8.5.5: The left half of the motif in Figure 8.2.1 constructs a function that is localized in x_1, but is independent of x_2.

.

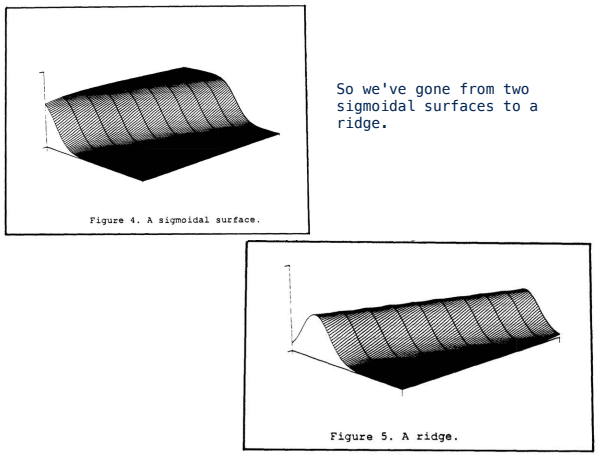

Figure 8.5.6: The construction of a ridge. Images from [10].

.

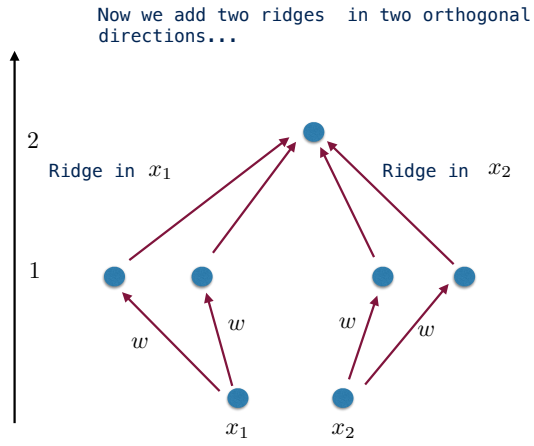

Figure 8.5.7: The use of ridges in both x_1 and x_2 gives a 'pseudo-bump' as shown in Figure 8.5.8.

.

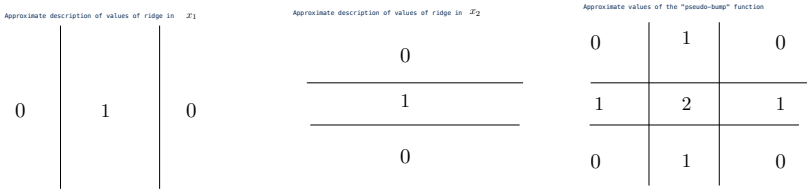

Figure 8.5.8: Approximate values of the pseudo-bump function obtained by adding the values of the ridge in x_1 and the ridge in x_2.

.

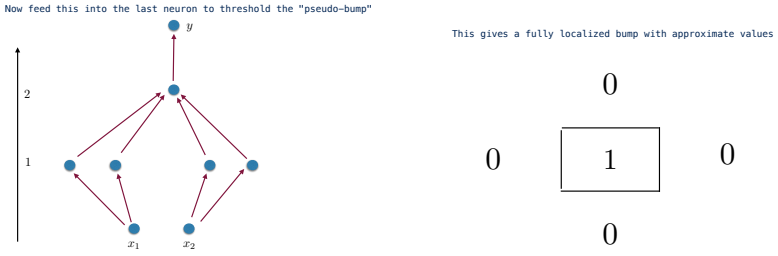

Figure 8.5.9: The last neuron provides a threshold that constructs a localized bump whose (approximate) values are shown in the image on the right.

.

Chapter 9

Convolutional neural networks: from biology to technology

9.1 Generalizability

The use of neural networks is a computationally effective method for approximating functions with applications to supervised learning. However, feed-forward neural networks without any additional structure have several limitations, most notably the size of the network needed for a specific learning task. Since the size of the training data must scale with the set of parameters, training the network could be too expensive when there are too many parameters. For example, consider an image consisting of 10×10 pixels and assume that the architecture has depth 2, with all-to-all connectivity from layer to layer. Since we have 100 neurons at input, we then have 10,000 edges in the first two layers and we need to fit at least 20,000 parameters just for these two layers.

Analysis of the number of parameters needed to train the network is of central importance in supervised learning. In broad strokes, there are two distinct undesirable scenarios – underfitting and overfitting (see Figure 9.1.1). Underfitting means that the model has too few parameters to capture the features of the training data. Overfitting means that the approximation captures too many details of the training data and is liable to fail when it is applied to the test data, i.e. when applied in practice.

A central concern in this approach to machine learning is *generalizability*. The generalization error is defined to be:

$$G(F(W)) = \mathbb{E}_{(X,Y)}(L(X;Y,W)), \tag{9.1.1}$$

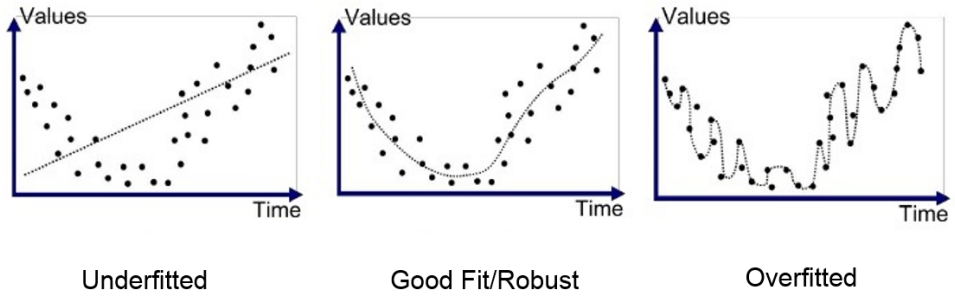

Figure 9.1.1: Underfitting and overfitting. (Image: wiki images)

.

and the *empirical error* is

$$G_n(F(W)) = \frac{1}{n}\sum_{k=1}^{n} L(X_k; Y_k, W).$$

Here L is the loss function, W is the set of weights, (which defines the input-output relation $F(W)$), and the expectation is taken over an unknown law of the input and output vectors X and Y. The empirical error can be measured, whereas the generalization error cannot. A significant part of the theoretical analysis in supervised learning is to prove that the empirical error is close to the generalization error. These topics lie beyond the scope of these lectures [16].

In contrast, the main unresolved theoretical issue in deep learning is that while the number of trainable parameters is so large that one would naively expect the neural network to overfit data, this does not happen in practice. Training through gradient descent of a loss function seems to be a mechanism that keeps the generalization error of the network in the sweet spot between underfitting and overfitting.

9.2 Convolutional neural networks

In this section, we discuss LeNet5, a convolutional neural network (CNN) introduced in [11] (see Figure 9.2.5). This network is one of the foundational examples in deep learning. It was introduced to solve the problem of handwriting recognition; however, we will restrict attention to its use for character recognition. The success of LeNet5 depended on both theoretical and implementational factors. Since there are many sources for this material – and the primary source is very readable – we will present only an outline of the main ideas. These are:

1. Convolutional neural networks retain the flexibility of neural networks to approximate functions.

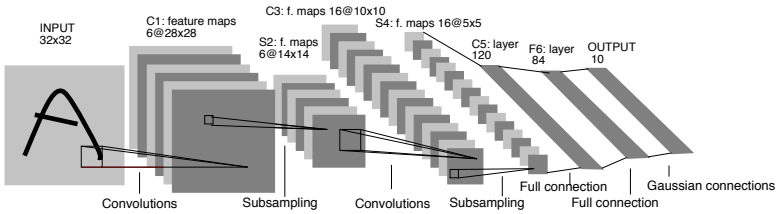

Figure 9.2.1: The architecture of LeNet5. Image from [11].

.

2. The number of weights is reduced by using spatial symmetries of the data (characters) to repeat important geometric motifs. Examples of such motifs are provided below.

3. Convolutional neural networks retain the hierarchical nature of Bayesian models with feature maps. However, the features are (mainly) trained by the machine, and are not designed 'by hand' as in the Bayesian model of character recognition discussed in Section 7.3.4.

As seen in Figure 9.2.1, the architecture of LeNet5 consists of a hierarchy of layers. The main layers are of two types: Convolutional filters and sub-sampling layers ("C-" and "S-layers" respectively). The output layer (there are 10 units for digit recognition) is not the naive choice discussed in the previous chapter. Instead, a standard pixelated image is constructed for each digit, and a loss function that measures the difference between the output of the layer F6 and these standard digits is computed. Let us now discuss the C- and S-layers in more detail.

The task of the C-layers is to amplify features in their input. Simple examples of features are the geometric motifs shown in Figure 9.2.2. These are of obvious importance in character recognition. A convolutional filter that detects an edge is shown in Figure 9.2.3. An essential aspect of LeNet5 is that the *same* weights act *locally* on contiguous blocks on the input layer. Thus, in Figure 9.2.3, the filter has $9 = 3 \times 3$ weights, and the same filter acts on all 3×3 square subunits of the 6×6 input layer. This has the effect of reducing the size of the output as shown. Similarly, we see that the first layer of LeNet5 has the effect of reducing an input images with 32×32 pixels into a feature map with 28×28 pixels.

The task of the S-layer is to coarse-grain the input from the C-layer in a manner that removes unecessary information. For example, when detecting features such as endpoints or edges, what matters is whether the feature is present in a region of the image. Its precise location and size within the region is largely irrelevant. An example of this idea is presented in Figure 9.2.5. Once we know that we have an endpoint, a corner and an edge in the blocks shown, we know that the character is a '7'. The relative placement of these features in these blocks is irrelevant. Several choices of nonlinearity may be used for sub-

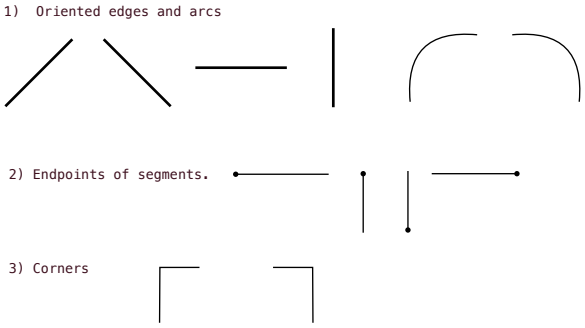

Figure 9.2.2: Examples of features.

.

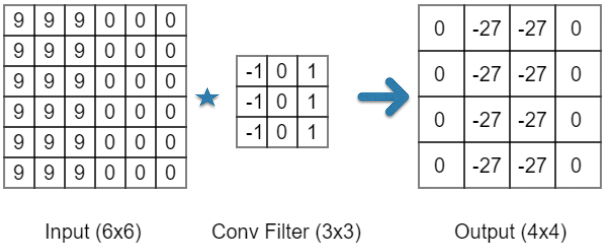

Figure 9.2.3: An example of a convolutional filter to detect a vertical edge. Image from the web.

.

Figure 9.2.4: We infer shapes from their absence in the images on the left. The image on the right reveals a caricaturist's ability to reduce an image to its essentials.

.

sampling, including averaging over the pool, choosing the max (Figure 9.2.6) and the rectified linear unity (ReLU) pooling shown in Figure 9.2.7.

These descriptions are intuitive. But they don't quite explain the mysterious, almost magical, ability of this network to recognize characters so effectively. In particular, we should note that while both the C- and S- layers can be initialized based on our visual intuition of what constitutes features (as in Figure 9.2.2 and Figure 9.2.5) there are more feature and sub-sampling layers than these. Further, *all* the weights for all these layers are chosen by training on data.

Thus, while these visual examples serve as good seeds for training for the first layer, as well as for our intuition, the deeper principles on why the structure works cannot be understood based only on these examples. In fact, CNNs work for several other recognition tasks where the notion of "features" is not as visually apparent. They also (as of May 2020) fail to work for sufficiently distorted characters (e.g. CAPTCHA), so at least some aspects of internet security remain intact.

9.3 Biological inspiration for LeNet5

The design of architectures remains a mysterious feature of deep learning. One of the most interesting aspects of LeNet5 is that it was inspired – and should be seen as a part of – the interplay between biology and technology.

The primary biological inspiration for LeNet5 was the experimental work of Hubel and Wiesel on vision in cats [**?**]. In their experiments, electrodes were implanted in an anesthesized cat's brain, the cat was subjected to visual stimuli and the output of the electrode was recorded when neurons fire. The main finding was that a subset of neurons fired for edges at a 45^o angle, whereas another set fired for edges at another angle. Thus, contiguous sets of neurons serve to identify specific edge orientations. This architecture of neurons in the

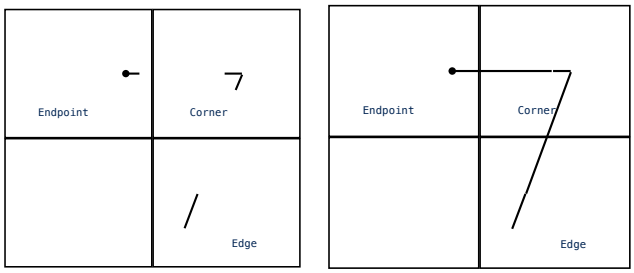

Figure 9.2.5: Once the features have been identified in the regions shown, it can immediately be inferred that the character is a seven. The relative placement, scale and distortion of these features does not affect this conclusion. Subsampling is introduced to capture this aspect of character recognition.

.

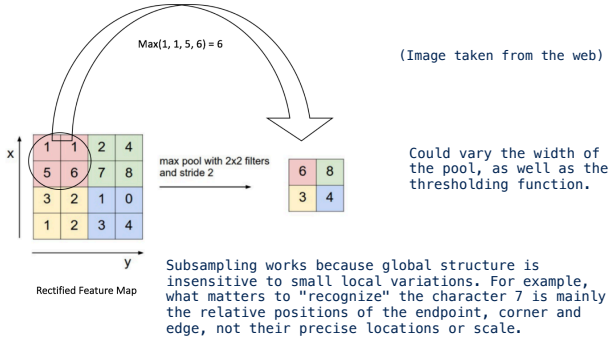

Figure 9.2.6: An example of subsampling with max-pooling.

.

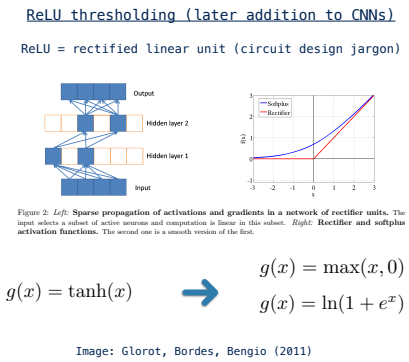

Figure 9.2.7: The sigmoid nonlinearity of neurons can be replaced by the Rectified Linear Unit (ReLU) nonlinearity. This imposes sparsity in the network.

.

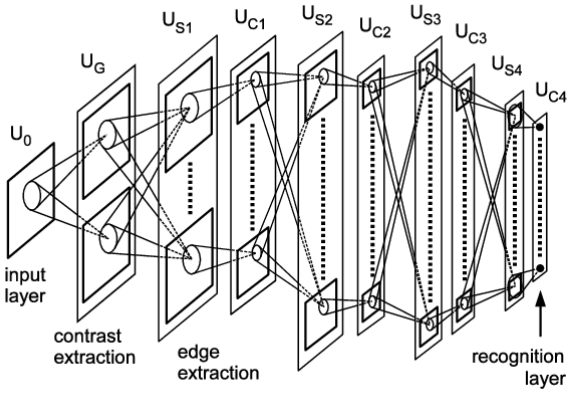

Figure 9.2.8: The architecture of the neocognitron contains alternating layers of simple and complex cells as in the findings of Hubel and Wiesel. Image from [7].

.

visual cortex is called "ocular dominance columns". The general conclusion of these seminal studies is that edge detectors, motion detectors, stereo vision, depth detection are developed in early childhood by specialization of simple and complex cells.

An important attempt to use this biological architecture for artifical pattern recognition was in the construction of a neural network called the neocognitron by [7]. See Figure 9.2.8. This neural network is the first true artificial multilayer neural network for pattern recognition. The main weakness of Fukushima's model is that it contained no true training mechanism. It also preceded the availability of training data and computational power that made LeNet5 so succesful.

In summary, the architecture of LeNet5 builds on empirical knowledge gained on predecessors of this model, as well as biological inspiration. The main insight is the repeated use of *local* units and the interspersing of feature extracting layers and sub-sampling in order to capture aspects of the image that are invariant under shift, scale and distortion. This significantly decreases the number of weight (there are 340,908 edges in LeNet5, but only 60,00 weights, since these are shared across a layer).

Chapter 10

Fast methods I: the FFT and numerical linear algebra

10.1 What is a fast method?

The purpose of this chapter, and the ones that follow, is to provide a bird's-eye view of some of the mathematical infrastructure that underlies the modern world. The success of any technique in machine learning ultimately relies on the availability of data to train models, inexpensive computational power and energy and (here is where the math comes in) fast algorithms. The purpose of this chapter is to provide examples of fast algorithms in different areas of scientific computing – Fourier analysis, numerical linear algebra, optimization – along with some of the beautiful mathematics that underlies these algorithms.

The notion of 'fast' is theoretical: we look at the operation count for some algorithms (FFT, matrix multiplication) where the essential argument can be seen without much mathematical background. For certain methods, such as iterative algorithms in numerical linear algebra, the precise notion of rates of convergence is more subtle.

One of the main purposes of this chapter is to illustrate the mathematical depth of what seem at first sight to be routine tasks requiring not much more than linear algebra and some calculus. These include the task of multiplying two matrices, solving linear systems, computing integrals (the Fourier transform), finding roots, and finding the maximum of an affine function on a convex domain (linear programming). All practising applied mathematicians know that the truth is very different from this perception: it is often the most simply stated problems that are the hardest to resolve. At the time of this writing, we still do not know what the optimal algorithm is for matrix multiplication. This is not just a theoretical issue. As an increasing range of our interactions moves online, the development of fast numerical methods is intimately tied to the energy usage by society.

10.2 The Fast Fourier Transform

Let us first recall the definitions of the continuous and discrete Fourier transforms. An integrable function $f : \mathbb{R} \to \mathbb{R}$ is related to its Fourier transform $\hat{f}$ through

$$\hat{f}(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x)\, dx, \quad f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx} \hat{f}(k)\, dk. \tag{10.2.1}$$

The discrete Fourier transform is a linear transformation between vectors f and $\hat{f}$ in $\mathbb{C}^n$. In a manner similar to equation (10.2.1) we set

$$\hat{f}_k = \sum_{n=0}^{N-1} e^{-2\pi ikn} f_n, \quad f_n = \frac{1}{N} \sum_{n=0}^{N-1} e^{-2\pi ikn} \hat{f}_k. \tag{10.2.2}$$

Here k and n are integers taking the values $0, 1, \ldots, N-1$. There is always some wiggle room in where one places the normalizing factors of 2π and N in equations (10.2.1) and (10.2.2) respectively. We have chosen these factors so that the exposition of the FFT does not rely on these factors. It is also convenient to write

$$\hat{f}_k = \sum_{n=0}^{N-1} z^{nk} f_n, \quad f_n = \frac{1}{N} \sum_{n=0}^{N-1} z^{-nk} \hat{f}_k, \quad z = e^{-2\pi i/N}.$$

This allows us to write the discrete Fourier transform as a matrix-vector product

$$\hat{f} = F_N(z) f, \quad f = \frac{1}{N} F_N(\bar{z}) \hat{f}, \quad (F_N(z))_{kn} = z^{kn}.$$

Observe that $\bar{z} = z^{-1}$.

Since the matrix $F_N(z)$ has N^2 entries and f is a vector of length N, we expect this product to require N^2 multiplications. This is not true! The Fast Fourier Transform uses the structure of the matrix $F_N(z)$ to compute the discrete Fourier transform of an arbitrary vector f in $N \log N$ operations.

There are several versions of the FFT. We will restrict ourselves to the Cooley-Tukey algorithm in the simplest setting. [1] To this end, we assume that $N = 2^p$ so that the divide-and-conquer nature of the scheme is transparent. We begin by separating the sum in equation (10.2.2) into odd and even terms as follows

$$\hat{f}_k = \sum_{\text{even } n} f_n z_N^{nk} + \sum_{\text{odd } n} f_n z_N^{nk}. \tag{10.2.3}$$

The algorithm is recursive. We now further split the input as follows. Let g and h be the vectors of length $N/2$ defined by

$$\begin{aligned} g &:= (g_0, \ldots, g_{N/2-1}) = (f_0, f_2, \ldots, f_{N-2}), \\ h &:= (h_0, \ldots, h_{N/2-1} = (f_1, f_3, \ldots, f_{N-1}). \end{aligned} \tag{10.2.4}$$

[1] The original paper is a short and attractive read. See [2].

Let us now consider the sum of the even terms. We observe that

$$\sum_{\text{even } n} f_n z_N^{nk} = \sum_{m=0}^{N/2-1} g_m e^{-\frac{2\pi i}{N} 2mk} = \sum_{m=0}^{N/2-1} g_m e^{-\frac{2\pi i}{N/2} mk} = \hat{g}_k, \tag{10.2.5}$$

provided that $0 \leq k \leq N/2 - 1$. (We need this restriction since the Fourier transform $\hat{g}_k$ is only defined for k in this range). A similar calculation for the odd terms yields

$$\sum_{\text{odd } n} f_n z_N^{nk} = \sum_{m=0}^{N/2-1} h_m e^{-\frac{2\pi i}{N}(2m+1)k} = z_N^k \sum_{m=0}^{N/2-1} h_m e^{-\frac{2\pi i}{N/2} mk} = z_N^k \hat{h}_k. \tag{10.2.6}$$

Thus, we have obtained the identity

$$\hat{f}_k = \hat{g}_k + z_N^k \hat{h}_k, \quad 0 \leq k \leq N/2 - 1. \tag{10.2.7}$$

We then revisit the above calculations to obtain the complementary identity

$$\hat{f}_{k+N/2} = \hat{g}_k - z_N^k \hat{h}_k, \quad 0 \leq k \leq N/2 - 1. \tag{10.2.8}$$

The identities (10.2.7) and (10.2.8) capture the recursive nature of the FFT. We next apply these identities to g and h respectively reducing the computation of these two terms to DFT's for vectors of length $N/4$ respectively.

We assumed that $N = 2^p$. Thus, $p = \log_2 N$ steps are needed to hit the lowest depth. We also need $N/2$ multiplications in each step (one for each value of k in equations (10.2.8) and (10.2.8) respectively). Thus, the total number of multiplications required is $(N/2)\log_2 N$.

While there are several variants in the implementation of this algorithm, especially when N is not a power of 2, this fundamental scheme underlies all modern digital signal processing applications.

10.3 Fast Matrix multiplication

How many multiplications does one need to multiply two 2×2 matrices? The naive answer is, of course, 8. Indeed, lets just work it out and count the number of operations when we multiply two 2×2 matrices, A and B to obtain $C = AB$

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}. \tag{10.3.1}$$

We are implicitly assuming here that multiplications are more expensive than additions.

An ingenious multiplication scheme of Strassen shows that we can do better

than 8 multiplications [**?**]. Form the intermediate products

$$\begin{aligned} x_1 &= (a_{11} + a_{22})(b_{11} + b_{22}), \\ x_2 &= (a_{21} + a_{22})b_{11}, \\ x_3 &= a_{11}(b_{12} - b_{22}), \\ x_4 &= a_{22}(b_{21} - b_{11}), \\ x_5 &= (a_{11} + a_{12})b_{22}, \\ x_6 &= (a_{21} - a_{11})(b_{11} + b_{12}), \\ x_7 &= (a_{12} - a_{22})(b_{21} + b_{22}). \end{aligned}$$

It then turns out that

$$\begin{aligned} c_{11} &= x_1 + x_4 - x_5 + x_7, \\ c_{12} &= x_3 + x_5, \\ c_{21} &= x_2 + x_4, \\ c_{22} &= x_1 - x_2 + x_3 + x_6. \end{aligned}$$

There is only one way to get a feel for these calculations, which is to work it all out, tedious as it seems, and check that it holds!

The full power of these calculations only becomes apparent when we use the divide-and-conquer idea as in the FFT. Suppose A and B are 4×4 matrices. We may break these matrices into 2×2 blocks and use Strassen's algorithm recursively. That is, write

$$C = AB = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \tag{10.3.2}$$

where each A_{ij} and B_{ij} is a 2×2 matrix. Now using Strassen's algorithm, we only need 7 block multiplications, each of which requires only 7 multiplications. Thus, instead of $4^3 = 64$ multiplications, we need $7^2 = 49$ multiplications.

Proceeding inductively, suppose $N = 2^p$, then we need to repeat this loop p times with cost

$$7^p = 7^{\log_2 N} = N^{\log_2 7} \approx N^{2.8074}.$$

This must be compared with the naive cost of multiplying matrices which is N^3. Some landmark results that build on Strassen's work, systematizing it with algebra and group theory, are listed below:

Authors	Year	Exponent
Strassen	1969	$\log_2 7 \approx 2.8074$
Coppersmith, Vinograd	1990	2.375477
Stothers, Williams, LeGall	2014	2.3728639.

All of this begs the question: is this the best one can do? What is the complexity of matrix multiplication? Is there an algorithm that multiplies matrices in $O(N^2 \log N)$ operations? The answer to this problem is not known. I can think of few problems in mathematics that are as simple to state, and are of such importance, that remain tantalizingly open.

10.4 The QR factorization

Numerical linear algebra is the art and science of two problems of primary importance: (i) solving linear systems (always written $Ax = b$) and (ii) the computation of eigenvalues. The simplicity and flexibility of these problems ensure that they have a wide range of applications. With some exaggeration one may say that a typical google search (or more precisely an implementation of the PageRank algorithm) is a computation of the top eigenvalue of a stochastic matrix whose size is on the order of billions. While one learns in elementary linear algebra that a square matrix A is invertible if and only if $\det A \neq 0$, and that the eigenvalues of A are the roots of the polynomial $\det(\lambda I - A)$, the one thing everyone should know about numerical linear algebra is that computing the inverse or eigenvalues of a matrix using determinants is just about the worst possible way of approaching these problems. In practice, it is necessary to pay close attention to the context when solving linear systems since the solution methods for a linear system $Ax = b$ depend strongly on the size and structure of A. Linear systems that arise from discretizations of partial differential equations are often sparse and symmetric. Even when A is symmetric, the methods for symmetric, positive definite A are different from methods for arbitrary A.

The central idea in numerical linear algebra is to compute fast, numerically stable matrix factorizations and then to use these to solve linear systems and for eigenvalue computation. This section provides an introduction to one such factorization, the QR factorization, along with applications to solving linear systems and eigenvalue computation. Good references for the material in this section are the books [4, 15].

Every finite dimensional inner product space V has an orthonormal basis that may be computed by the Gram-Schmidt procedure. Suppose $\dim(V) = n$ and let $S = \{v_1, v_2, \ldots, v_n\}$ be a set of linearly independent vectors in V. Denote the inner product of two vectors v and v' in V by $\langle v, v' \rangle$. We may construct an *orthonormal* basis Q from the basis S as follows. Let

$$w_1 = v_1, \quad q_1 = \frac{w_1}{|w_1|}, \quad |w| := \langle w, w \rangle^{1/2}.$$

Next let

$$w_2 = v_2 - \langle v_2, q_1 \rangle q_1, \quad q_2 = \frac{w_2}{|w_2|}.$$

We have subtracted off the projection of v_2 onto the vector q_1 and normalized. This ensures that $\langle q_1, q_2 \rangle = 0$ and $|q_1| = |q_2| = 0$. Let $S_2 = \text{span}\{q_1, q_2\}$. Proceed inductively, setting

$$w_k = v_3 - \langle v_2, q_1 \rangle q_1 - \ldots - \langle v_k, q_{k-1} \rangle q_{k-1}, \quad q_k = \frac{w_k}{|w_k|}.$$

Our assumption that S is a basis ensures that w_k does not vanish (if not, w_k would lie in $S_{k-1} = \text{span}\{w_1, \ldots, w_{k-1}\}$). The procedure terminates at the n-the step, yielding a set of orthonormal vectors S_n.

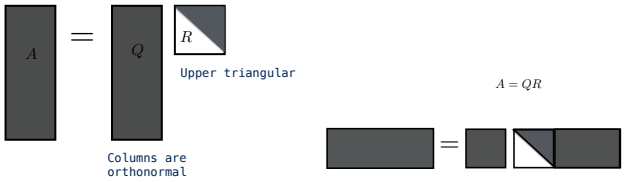

Figure 10.4.1: A caricature of the QR factorization. All blank entries are zero. The figure on the left corresponds to an overdetermined system. That on the right is underdetermined.

.

The QR factorization is a numerical interpretation of the Gram-Schmidt procedure. Assume given an $m \times n$ matrix A with linearly independent columns. We apply the above procedure to the column space $S = \text{span}\{A_1, \ldots, A_n\} \subset \mathbb{R}^m$, to obtain a set of column vectors $Q_1, \ldots, Q_n$ which have the same span as S. Here $\{A_j\}_{j=1}^n$ denote the columns of A. Conversely, we form the matrix

$$Q = \begin{pmatrix} Q_1 & Q_2 & \ldots & Q_n \end{pmatrix}.$$

By construction,

$$\text{span}\{A_1, \ldots, A_k\} = \text{span}\{Q_1, \ldots, Q_k\}, \quad 1 \leq k \leq n.$$

Therefore, we may write

$$A = QR, \quad Q^T Q = I_{n \times n}. \tag{10.4.1}$$

The caricatures shown in Figure 10.4.1 are a useful mnemonic.

10.5 Solving linear systems

Let us now use the QR factorization to solve the linear equation

$$Ax = b. \tag{10.5.1}$$

Despite its simplicity, there are two fundamentally distinct classes of linear systems, as shown in Figure 10.5.1. In order to keep things concrete, we will focus on the least squares solution for overdetermined systems. An example of a least squares problem is as follows. Suppose we are given m data points of the form $\{(t_i, y_i)\}_{i=1}^m$ and we'd like to fit a functional relation of the form $y = f(t)$ to this data. One way of constructing such a fit, is to choose a family of basis function, say polynomials $\{p_j(x)\}_{j=1}^m$ and to guess that

$$f(t) = \sum_{j=1}^{n} c_j p_j(t).$$

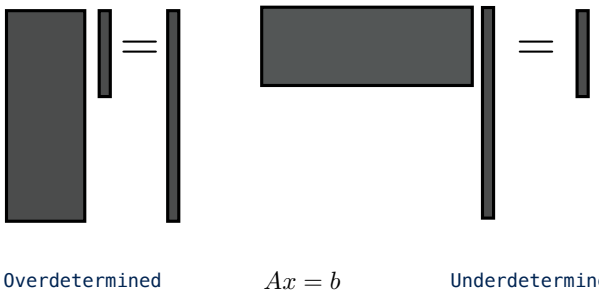

Figure 10.5.1: Overdetermined systems often arise in least squares and linear regression. Undetermined systems arise in applications such as compressed sensing and inverse problems.

.

The function $f(t)$ is a perfect fit if it satisfies

$$f_i = \sum_{j=1}^{m} c_j p_j(t_i), \quad 1 \leq i \leq m, \tag{10.5.2}$$

which is of the form $Ax = b$, with

$$x_j = c_j, \quad b_j = y_j, \quad A_{ij} = p_j(t_i), \quad 1 \leq i \leq n, 1 \leq j \leq m.$$

In practice, we do not expect an exact fit and our task is to find a principled solution to the problem that accounts for this fact.

The least squares solution is obtained by replacing the equation $Ax = b$ (which does not have a solution) with the equation $A^T Ax = A^T b$, (which does have a solution). This is best seen geometrically: multiplying a vector y by A^T is equivalent to orthogonal projection of y onto the column space of A. It is immediate that the matrix $A^T A$ is symmetric. Further, it is at least positive semi-definite, since

$$x^T A^T Ax = |Ax|^2 \geq 0,$$

and it is strictly positive definite when the columns of A are linearly independent. This is a mild assumption that usually holds in practice.

Let us now explain how the QR factorization may be used to solve the least squares problem. We substitute $A = QR$ in the the least squares equation

$$A^T Ax = b,$$

and use the fact that $Q^T Q = I$ to obtain the equation

$$R^T Rx = R^T Q^T b.$$

Since $A^T A$ is invertible, so is R, and the least squares solution satisfies

$$Rx = Q^T b.$$

This system is lower-triangular and may be solved by backsubstitution. To see this, assume that $n = 3$, write $c = Q^T b$ and observe that we have the system

$$\begin{aligned} r_{11}x_1 + r_{12}x_2 + r_{13}x_3 &= c_3, \\ r_{22}x_2 + r_{23}x_3 &= c_2, \\ r_{33}x_3 &= c_3. \end{aligned}$$

We first solve for x_3, then for x_2, and then x_1. Clearly, this procedure works for all n.

10.6 The power method

The *symmetric eigenvalue problem* is one of the fundamental problems in numerical linear algebra. We assume given a real symmetric matrix A. Our task is to compute its eigenvalues. The method we choose depends strongly on our needs. For example, in applications such as search engines, what matters the most are the top eigenvalues of a large matrix (in particular, we don't need *all* the eigenvalues). On the other hand, if our goal is to understand the linear stability of a control system, we may well need all the eigenvalues of A.

We have encountered a simple, but important, method to compute the top eigenvector of a matrix in the very first section of this book: the equilibrium distribution for a stationary ergodic Markov chain is the unique left eigenvector with eigenvalue 1. All other eigenvalues of the Markov chain lie within the unit disk in the complex plane. This is an important example (for example, it has the same structure as PageRank) and it is helpful to think dynamically. We expect the pmf of a Markov chain to approach its equilibrium distribution as time increases. Further, we also know that the pmf of the Markov chain evolves according to the forward equation. This suggests the following iterative algorithm called the *power method*. Suppose the eigenvalues of A are ordered in absolute value such that

$$|\lambda_1| > |\lambda_2| \geq \ldots |\lambda_k| \ldots \geq |\lambda_n|.$$

We have assumed the top eigenvalue is distinct for simplicity. Choose a random initial vector x_0, normalize it to length 1, and define the sequence

$$x_k = A^k x_0, k \geq 0.$$

Since x_0 is a linear combination of the eigenvectors $\{v_j\}_{j=1}^n$ of A, we have

$$x_0 = c_1 v_1 + c_2 v_2 + \ldots c_n v_n,$$

and

$$x_k = c_1 \lambda_1^k v_1 + \ldots c_n \lambda_n^k v_n.$$

Dividing through by λ_1, we see that

$$x_k = \lambda_1^k \left(c_1 v_1 + (\frac{\lambda_2}{\lambda_1})^k v_2 + \ldots + (\frac{\lambda_n}{\lambda_1})^k v_n \right).$$

As $k \to \infty$, all terms of this sum decay exponentially fast to zero. Therefore, normalizing by the length, we see that

$$\lim_{k\to\infty} \frac{x_k}{|x_k|} = v_1.$$

10.7 The QR algorithm and the the QR flow

The QR factorization is also the foundation for a successful algorithm for diagonalizing a matrix. The simplest version of this method goes as follows. Assume given a real symmetric matrix A. We construct a sequence of matrices $\{A_k\}_{k=0}^{\infty}$ with the initial condition $A_0 = A$ as follows. First factor

$$A_0 = Q_0 R_0$$

and then intertwine the factors setting

$$A_1 = R_0 Q_0.$$

Now proceed inductively, factoring and intertwining

$$A_k = Q_k R_k, \quad A_{k+1} = R_k Q_k, \quad k \geq 0.$$

Since Q is orthogonal, $Q^T = Q^{-1}$, and we find that

$$A_{k+1} = Q_k^T A_k Q_k.$$

This ensures that all the matrices A_k have the same eigenvalues (we say that they are *isospectral*). We may also write this sequence in the form

$$A_{k+1} = U_k^T A_0 U_k, \quad U_k = Q_0 Q_1 \cdots Q_n. \tag{10.7.1}$$

For typical initial conditions [2]

$$\lim_{k\to\infty} A_k = \Lambda, \quad \lim_{k\to\infty} U_k = U,$$

where Λ and U are the diagonal matrix of eigenvalues, and the matrix of eigenvectors respectively in the diagonalization

$$A = U\Lambda U^T.$$

This algorithm is a striking demonstration of the power of the QR factorization.

Practical implementation of the QR algorithm requires both preprocessing and acceleration. The preprocessing step is another factorization: a symmetric matrix A may be tridiagonalized through a sequence of easily computed rotations called Housholder matrices. It turns out that the QR algorithm preserves the tridiagonal structure; thus, we usually assume that the initial matrix A is

[2] That is, implement this scheme in MATLAB and check that this works!

tridiagonal. A second crucial modification of the QR algorithm is to shift it with a carefully chosen diagonal matrix prior to each factorization step, and then to 'unshift' at the end of the factorization step. We will not consider these steps here; see [15].

The convergence of the QR algorithm reflects a subtle mathematical structure. [3] In order to illustrate this structure, we first introduce a characterization of orthogonal matrices. Let

$$O(n) = \{Q \in \mathbb{M}_{n\times n} \,|\, Q^T Q = I\} \tag{10.7.2}$$

One may think about $O(n)$in several different ways. First, it is the solution set for $n(n+1)/2$ quadratic equations (since Q^TQ is symmetric, the matrix equation $Q^TQ = I$ is equivalent to $n(n+1)/2$ scalar equations). So we may think about it as an $n(n-1)/2$ dimensional "surface" in the n^2 dimensional space $\mathbb{M}_{n\times n}$.

More importantly, $O(n)$ is a *group*. This means that $O(n)$ is closed under multiplication, that it has an identity element and that the inverse of any element is also in $O(n)$. We won't verify all these properties, but in order to help the reader get started, here is how one checks that $O(n)$ is closed under multiplication. Consider the product Q_1Q_2 of two matrices Q_1 and Q_2 in $O(n)$. Then

$$(Q_1Q_2)^T Q_1Q_2 = Q_2^T Q_1^T Q_1Q_2 = Q_2^T Q_2 = I,$$

so that $Q_1Q_2 \in O(n)$.

The elements of the group $O(n)$ are generated by solutions to the linear differential equation

$$\dot{U} = UK, \quad U(0) = I, \tag{10.7.3}$$

where the matrix K is skew-symmetric, i.e.,

$$K = -K^T. \tag{10.7.4}$$

Equation (10.7.3) is a (matrix-valued) linear differential equation with constant coefficients, and it can be solved in much the same way as the linear differential equations you have seen in your first class on the subject. The solution is

$$U(t) = e^{tK}U(0), \quad e^{tK} = \sum_{m=0}^{\infty} \frac{1}{m!}(tK)^m. \tag{10.7.5}$$

In fact, if one has a curve $K(t)$ in the space of skew-symmetric matrices, it is always the case that the solution to the differential equation

$$\dot{U} = UK(t), \quad U(0) = I \tag{10.7.6}$$

is a curve $U(t)$ in $O(n)$. This observation provides a way to generate isospectral curves in the space of symmetric matrices. Assume given a curve $K(t)$, let $U(t)$

[3] There are other ways to prove convergence of the QR algorithm. Our purpose here is to illustrate an unexpected connection with differential equations.

solve equation 10.7.6, assume that B_0 is a real symmetric matrix and consider the matrices $B(t)$ defined by

$$B(t) = U(t)^T B_0 U(t). \tag{10.7.7}$$

We differentiate equation (10.7.7) and use equation (10.7.6) to find

$$\dot{B} = BK - KB := [B, K], \tag{10.7.8}$$

where we have defined the bracket $[\cdot, \cdot]$ in the second equality. [4]

We need two more notions. Given any matrix $M \in \mathbb{M}_{n\times n}$, let M_-, denotes the matrix whose strictly lower-triangular entries are the same as M and other entries are zero; let M_0 be the diagonal matrix whose entries are the diagonal of M; and let M_+ be the strictly upper-triangular matrix analogous to M_-. Clearly,

$$M = M_- + M_0 + M_+.$$

We define the projection of a matrix onto the space of skew-symmetric matrices through the relation

$$P_s M := M_- - M_-^T.$$

The second notion we need is that of the logarithm of a matrix. Given a real symmetric matrix A, with diagonalization $A = U\Lambda U^T$, we define the matrix

$$\ln A = U \ln \Lambda U^T, \quad \ln \Lambda = \operatorname{diag}(\ln \lambda_1, \ldots, \ln \lambda_n).$$

Here finally is the link between the QR algorithm and differential equations. Consider the initial value problem

$$\dot{A} = [A, P_s \ln A], \quad A(0) = A_0, \tag{10.7.9}$$

where A_0 is the initial condition for the QR algorithm. Let $A(t)$ denote the solution to equation (10.7.9). We call this the QR flow. Then the QR flow and the QR algorithm agree at integer times: that is,

$$A(k) = A_k, \quad k \geq 0. \tag{10.7.10}$$

A full explanation of these facts requires more machinery than we can develop here. The interested reader is referred to [3]. Our purpose is to illustrate some of the 'magic' – in the sense of unexpected connections between areas– that lives behind successful algorithms.

[4] This is called the Lie (pronounced Lee) bracket.

Chapter 11

Fast methods II: Gradient flows

11.1 Introduction

All problems in machine learning require a fast method to approximate the minimum of a function. Bayesian methods require a computation of the maximum a posteriori estimate and neural nets rely on training by minimizing a loss function. In the previous chapters, we have encountered three algorithms that address this problem:

1. Markov Chain Monte Carlo (e.g. when deciphering a substitution cipher).

2. Dynamic programming (e.g. when parsing a music score).

3. The backpropagation algorithm (when training a feed-forward neural network).

In this chapter, we step away from learning and focus on optimization. This allows us to illustrate a central question in applied mathematics " How does one find the minimum of a function?"

We will study this question through the lens of gradient flows, paying explicit attention to the underlying Riemannian metric. This topic, along with the closely related idea of mass transportation, has been an important development in applied mathematics in the past twenty years. We illustrate these ideas through numerical applications (root finding, linear and semidefinite programming), and examples of non-Euclidean geometry. The relation between these 'applied' and 'pure' ideas is that Karmarkar's method, a fundamental interior point scheme for linear and semidefinite programming is a gradient flows with respect to a non-Euclidean geometry. Similarly, the heat equation is the gradient descent of entropy with respect to a Wasserstein geometry.

11.2 Newton's method

In this section, we assume given a function $f : \mathbb{R}^d \to \mathbb{R}$. Our task is to find its minima. We used gradient flows and gradient descent to solve this problem in Section 8.3. The term gradient flow always refers to ordinary differential equations like

$$\dot{x} = -\nabla f(x), \quad x \in \mathbb{R}^d, \tag{11.2.1}$$

and its generalization to Riemannian metric later in this chapter. The term gradient descent is used to describe discrete time evolution, as in the following discretization of equation (11.2.1)

$$x_{n+1} = x_n - \lambda_n \nabla f(x_n), \quad n \geq 0. \tag{11.2.2}$$

Here the parameter λ_n denotes the size of the (continuous) time increment at the (discrete) time n. The interplay between gradient flows and gradient descent is as follows. It is conceptually simpler to work in continuous time. For example, the convergence of gradient flows is typically easier to establish. On the other hand, numerical implementations are always in discrete time.

Newton's method (or the Newton-Raphson method) was initially viewed as a numerical scheme to determine the roots of a function $g : \mathbb{R}^d \to \mathbb{R}$. But since the critical points of a function $f : \mathbb{R}^d \to \mathbb{R}$ are also the zeros of $g(x) := \nabla f(x)$ the same method may be used for optimization.

We first illustrate the method for functions $g : \mathbb{R} \to \mathbb{R}$ and then extend it to $g : \mathbb{R}^d \to \mathbb{R}$. We generate a sequence of approximants $\{x_n\}_{n=1}^{\infty}$ to a root x_* as follows. Given x_n, we shoot forward from $(x_n, g(x_n))$ along a ray with slope $g'(x_n)$ until it intersects the x-axis as shown in Figure 11.2.1. Given an initial guess x_0, this geometric rule yields the iterative scheme

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}, \quad n \geq 0. \tag{11.2.3}$$

When this scheme converges, it works like a charm, because of the *quadratic* rate of convergence. This error analysis goes as follows. We expand the function $g(x)$ in a neighborhood of x_* using Taylor series to obtain

$$\begin{aligned} g(x) &= g(x_*) + g'(x_*)(x - x_*) + \frac{1}{2} g''(x_*)(x - x_*)^2 + \dots \\ &= g'(x_*)(x - x_*) + \frac{1}{2} g''(x_*)(x - x_*)^2 + \dots, \end{aligned} \tag{11.2.4}$$

since $g(x_*) = 0$. Similarly, we may also expand the slope

$$g'(x) = g'(x_*) + g''(x_*)(x - x_*) + \dots$$

to find

$$\frac{g(x_n)}{g'(x_n)} = (x_n - x_*) - \frac{g''(x_*)}{2g'(x_*)}(x_n - x_*)^2 + \dots \tag{11.2.5}$$

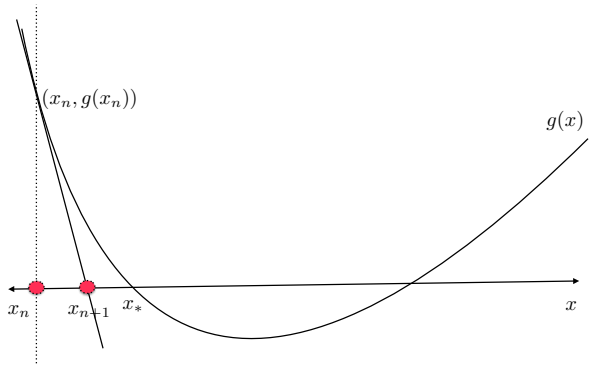

Figure 11.2.1: Newton's method on the line. Given $(x_n, g(x_n)$ and the slope $g'(x_n)$, the next point x_{n+1} is determined as shown in the figure. These iterates x_n converge to the root x_* of g.

.

The definition of the iteration and equation (11.2.5) now yield the approximation

$$x_{n+1} - x_* = \frac{g''(x_*)}{2g'(x_*)}(x_n - x_*)^2 + \dots \tag{11.2.6}$$

This is the quadratic relationship that underlies the fast convergence of Newton's method. Rather than pin down error rates with an $\epsilon - \delta$ argument, we illustrate the main idea through a comparison between two recurrence relations: the linear recurrence with a parameter $\theta > 1$

$$a_{n+1} = \frac{1}{\theta} a_n, \tag{11.2.7}$$

and the quadratic recurrence with parameter $\kappa \in (0, \infty)$

$$b_{n+1} = \kappa b_n^2. \tag{11.2.8}$$

The solution to the linear recurrence is

$$a_n = \theta^{-n} a_0 = e^{-n \ln \theta} a_0. \tag{11.2.9}$$

Thus, $a_n \to 0$ exponentially with rate $\ln \theta$. On the other hand, the solution to the quadratic recurrence is

$$b_n = \kappa^{2^n - 1} b_0^{2^n}, \tag{11.2.10}$$

so that $b_n \to 0$ at a *super-exponential* rate provided $\kappa^2 b_0^2 < 1$. This rate is the margin of victory for Newton's method. The main subtlety in Newton's method, however, is that may not converge at all if the initial condition of the scheme is not chosen appropriately.

Let us now explain how to extend this method to optimization. Assume given a function $f : \mathbb{R}^d \to \mathbb{R}$. In order to determine a minimum of f, we apply Newton's scheme to find the zeros of ∇f. This yields the iteration

$$x_{n+1} = x_n - (D^2 f(x_n))^{-1} \nabla f(x_n), \quad n \geq 0. \tag{11.2.11}$$

The gradient ∇f is the vector-valued replacement for the term $g(x)$ and the inverse of the Hessian matrix replaces the term $1/g'(x)$ in equation (11.2.3). When we compare equation (11.2.11) with equation (11.2.2), we see that the time step λ_n has been replaced with the inverse of the Hessian.

When choosing between gradient descent (11.2.2) and Newton's method (11.2.11), we must balance the cost of computing the Hessian ($O(d^2)$ terms to be evaluated at each time step) against the accelerated rate of convergence of Newton's method.

The primary global criterion for the convergence of (11.2.11) is *convexity*. A function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be convex if

$$f((1-\theta)x + \theta y) \leq (1-\theta) f(x) + \theta f(y), \quad x, y \in \mathbb{R}^d, \theta \in (0,1). \tag{11.2.12}$$

We say that f is *strictly* convex, if the above inequality is strict whenever x and y are distinct points. When f is twice differentiable equation (11.2.12) implies that the Hessian matrix $D^2 f(x)$ is positive definite for each $x \in \mathbb{R}^d$.

Theorem 22. *Assume $f : \mathbb{R}^d \to \mathbb{R}$ is strictly convex. Then f has a unique minimizer.*

Proof. Suppose f has two distinct minimizers x and y and consider the value of f at the midpoint of the segment joining x and y. Equation 11.2.12 then implies

$$f(\frac{1}{2}(x+y)) < \frac{1}{2} f(x) + \frac{1}{2} f(y), \tag{11.2.13}$$

contradicting the assumption that x and y are minimizers. □

Theorem 23. *Assume $f : \mathbb{R}^d \to \mathbb{R}$ is strictly convex and twice differentiable. Then the gradient flow (11.2.1) converges to the unique minimizer x_*.*

Proof. This is an easy consequence of the basic energy estimate for gradient flows. We evaluate $f(x(t))$ along a solution to find that

$$\frac{df(x(t))}{dt} = \nabla f(x) \cdot \dot{x} = -|\nabla f(x)|^2 \leq 0,$$

with strict inequality unless $x(t) = x_*$. Thus, $f(x(t))$ decreases in time and is bounded below by $f(x_*)$. Thus, $\lim_{t\to\infty} f(x(t))$ exists and by the above calculation, it must be $f(x_*)$. Since f is strictly convex, it has a unique minimum, and $\lim_{t\to\infty} x(t) = x_*$. □

The convergence of Newton's method (11.2.11), assuming convexity of f, should seem intuitively natural based on Figure 11.2.1. However, the above

estimates do not extend to (11.2.11). It is only possible to establish convergence of Newton's method in a neighborhood of x_*. In general, the convergence of Newton's method is subtle. Striking examples of non-convergence have been studied in the field of complex dynamics (one typically considers a polynomial $p : \mathbb{C} \to \mathbb{C}$ and we seeks its zeros using Newton's method). This *should* be a simple situation, but it is not!

11.3 Riemannian metrics

In this section, we introduce the idea of a Riemannian metric in preparation for the idea of a gradient flow relative to a metric. These notions are usually discussed in classes in differential geometry. We will discuss these ideas in a more formal manner, illustrating them with examples. This treatment will suffice for a users-end view of gradient flows, while still providing enough motivation for the mathematically inclined reader to learn more about these ideas.

To get started, lets recall the geometry of curves in the plane. Assume given a smooth curve $\gamma : [0, 1] \to \mathbb{R}^2$ in the plane. The derivative $\dot{\gamma}(t)$ is a tangent vector 'rooted' at the point $\gamma(t)$ and the length of the curve is defined to be

$$L[\gamma] = \int_0^1 |\dot{\gamma}(t)|\, dt, \quad |v| := \sqrt{v_1^2 + v_2^2}, \quad v \in \mathbb{R}^2. \tag{11.3.1}$$

This expression should be familiar. An underlying assumption, which we will soon modify, is that the length of a vector v rooted at $\gamma(t)$, does not depend on $\gamma(t)$, and is given by the expression in equation (11.3.1). The set of all vectors 'rooted at $\gamma(t)$' is called the tangent space at $\gamma(t)$. In the case of $\mathbb{R}^n$, the tangent space at any point $x \in \mathbb{R}^n$ is a copy of $\mathbb{R}^n$. However, this is not true for even the simplest surfaces, such as spheres, which is why it is necessary to separate the concept of the tangent space from that of the ambient space.

Let $\mathbb{P}_+(n)$ denote the space of (real, symmetric) positive definite matrices. A *Riemannian metric* on $\mathbb{R}^n$ (or metric for short, when the context is clear) is a map $g : \mathbb{R}^n \to \mathbb{P}_+(n)$. Given a metric g on $\mathbb{R}^n$, the length of a vector v rooted at x is defined by

$$|v|^2_{g(x)} := v^T g(x) v = \sum_{i,j=1}^{n} g_{ij}(x) v_i v_j. \tag{11.3.2}$$

The positive definite matrix g allows us to introduce anisotropy as shown in Figure 11.3.1. Aside from this, familiar formulas work in much the same way. For example, the length of a parametrized curve $\gamma : [0, 1] \to \mathbb{R}^n$ is then given by

$$L_g(\gamma) = \int_0^1 |\dot{\gamma}(t)|_{g(\gamma(t))}\, dt. \tag{11.3.3}$$

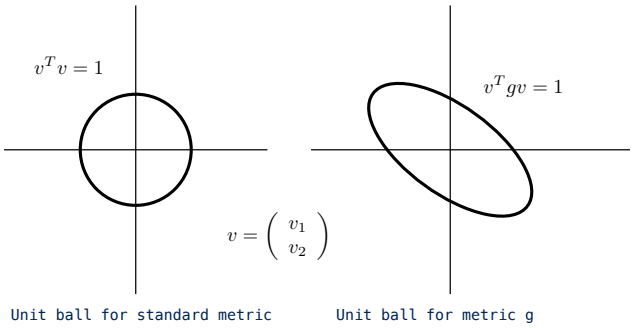

Figure 11.3.1: The unit ball in a non-Euclidean metric defines an ellipsoid in Euclidean space.

.

The shortest path between two points a and b, also called the *geodesic* connecting a and b, satisfies the following nonlinear differential equation

$$\ddot{x}_i + \Gamma^i_{jk} \dot{x}_j \dot{x}_k = 0, \tag{11.3.4}$$

with the boundary conditions $x(0) = a$ and $x(1) = b$. The terms Γ^i_{jk} are defined through the metric

$$\Gamma^i_{jk} = \frac{1}{2} \sum_l g^{il} \left(\frac{\partial g_{lj}}{\partial x_k} + \frac{\partial g_{lk}}{\partial x_j} - \frac{\partial g_{jk}}{\partial x_l} \right), \quad g^{lm} := (g^{-1})_{lm} \tag{11.3.5}$$

Equation (11.3.4) is obtained by computing the first variation of the length $L[\gamma]$ as follows. Let $\eta : [0,1] \to \mathbb{R}^n$ be a smooth function such that $\eta(0) = \eta(1) = 0$. We define the derivative of L in the direction η through the formula

$$dL[\gamma](\eta) := \frac{d}{d\tau} L[\gamma + \tau\eta] \Big|_{\tau=0}. \tag{11.3.6}$$

This definition provides a way to extend the calculus criterion for a extremum (that the derivative of a function vanishes at a critical point), to functionals like $L[\gamma]$. The reader should compute this derivative by differentiating with respect to τ under the integral sign, then integrating by parts to obtain an integrand that is a multiple of η, and then finally using the fact that η is arbitrary to deduce equation (11.3.5). It is somewhat simpler to first do the calculation for the *action* functional

$$\mathcal{A}_g[\gamma] = \frac{1}{2} \int_0^1 |\dot{\gamma}(t)|^2_{g(\gamma(t))} \, dt. \tag{11.3.7}$$

Historically, the idea of a metric arose from mathematical, physical and philosophical considerations on the nature of space in the 19th century. The

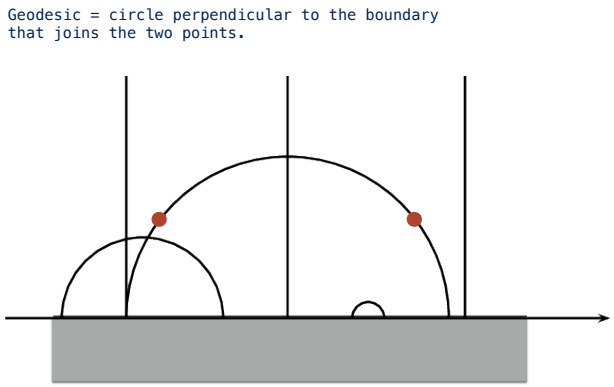

Figure 11.3.2: Geodesics in the upper half plane with the metric (11.3.8).

.

fundamental question considered at the time, especially by the German mathematician Bernhard Riemann, was whether space is absolute or relative and whether one could deduce the properties of space from local measurements. The importance of these ideas for physics only became apparent in the early 20th century when Einstein introduced the general theory of relativity, abandoning Newton's theory of gravitation.

Today, the idea of Riemannian geometry serves as a unifying tool for several areas, including the rather pragmatic considerations of this chapter. In order to make this connection, we briefly discuss a high point of 19th century mathematics: hyperbolic space. Consider the upper-half plane with metric

$$\mathbb{H} = \{(x, y) \in \mathbb{R}^2, y > 0\}, \quad g(x, y) = \frac{1}{y^2} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{11.3.8}$$

In this case, the geodesic connecting two points may be computed explicitly and is shown in Figure 11.3.2. One way to prove this fact is to substitute the above expression for g in equation (11.3.4) and to solve the resulting differential equation. Despite the fact that equation (11.3.4) is nonlinear, these equations can be solved and the solutions expressed in terms of hyperbolic sines and cosines.

Aside from their intrinsic beauty (and artistic possibilities; see Figure 11.3.4) these geodesic reveal a new geometric idea: intrinsic curvature. Given a Riemannian metric on $\mathbb{R}^n$ and three distinct points a, b and c in $\mathbb{R}^n$ we define the geodesic triangle abc to be the polygon whose sides are the geodesics ab, bc and ca. We may measure the angle between tangents to these geodesics at each vertex of abc. Unlike Euclidean space, the sum of the angles at the three vertices of the triangle is *strictly* less than π. The hyperbolic plane is *negatively curved* as shown in Figure 11.3.3! [1]

[1] A precise notion of intrinsic curvature is more subtle and it turns out that the space

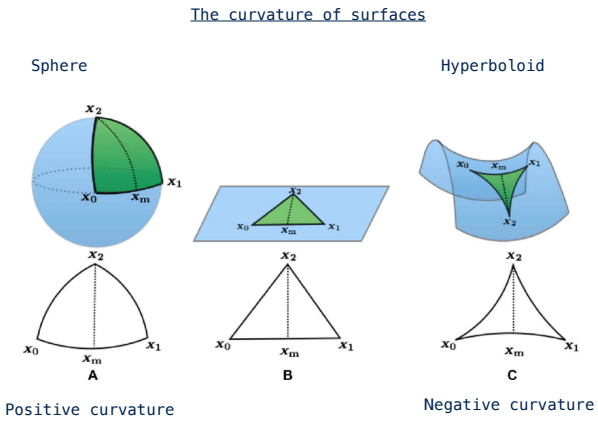

Figure 11.3.3: Geodesic triangles on surfaces of constant curvature.

.

Figure 11.3.4: Tilings of the upper-half plane in the work of the Dutch artist M.C. Escher.

.

The hyperbolic plane is just the first of a fascinating collection of negatively curved spaces. Another space, which is directly relevant to us, is a generalization of $\mathbb{H}$ to positive-definite matrices. We illustrate this idea with two explicit examples of metrics on matrices, one flat and one negatively curved.

1. The space $\mathbb{M}_{m,n}$ of real $m \times n$ matrices with metric

$$\|M\|_2^2 := \sum_{j,k} M_{jk}^2.$$

   This metric is called the Frobenius metric (or Frobenius norm) and it is the generalization to matrices of the usual Euclidean norm $|x|^2 = \sum_j x_j^2$ for $x \in \mathbb{R}^n$. It is a flat metric.

2. The space $\mathbb{P}(n)$ of positive definite matrices with the metric

$$g(P)(X, X) := \mathrm{Tr}(P^{-1}XP^{-1}X).$$

   Here X is a real symmetric matrix and is a tangent vector rooted at P. The above formula defines its length. The notational shift from lower-case to upper-case letters is to remind us that we're working with matrices, not vectors. This metric is called the *trace metric* . [2]

The geodesic equations may be solved for both these geometries. Given two matrices M_0 and M_1 in $\mathbb{M}_{m,n}$, the geodesic joining them is (as expected) the straight line

$$M(t) = (1-t)M_0 + tM_1, \quad t \in [0,1].$$

The trace metric is more interesting and the geodesic connecting A and B in $\mathbb{P}(n)$ is

$$P(t) = A^{1/2}\left(A^{-1/2}BA^{-1/2}\right)^t A^{1/2}, \quad 0 \le t \le 1.$$

This is a more subtle formula. Each positive definite matrix has a unique positive definite square-root and these square-roots are used here. Similarly, the power P^t of a positive definite matrix is defined using its diagonalization. This geometry is discussed at length in [1, Ch. 6]. The space $\mathbb{P}(n)$ is negatively curved.

11.4 Gradient flows

We will distinguish between the differential of a function and its gradient. The differential of a function $f : \mathbb{R}^n \to \mathbb{R}$ is defined using the notion of a smooth curve in $\mathbb{R}^n$ alone, as shown in Figure 11.4.1. As an operator, the differential

$\mathbb{H}$ has constant negative curvature, while the two-sphere has constant positive curvature. Nevertheless, the use of geodesic triangles is a robust idea and provides valuable intuition for what curvature means.

[2] The trace of a square matrix A, written $\mathrm{Tr}(A)$, is the sum of its diagonal entries $\sum_j a_{jj}$.

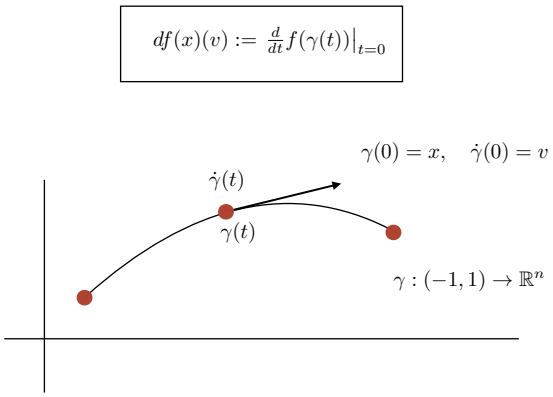

Figure 11.4.1: The differential of a function depends only on the tangent to the curve at a point, not on the underlying metric.

.

of f takes as input a tangent vector v in the tangent space at x and spits out a number $df(x)(v)$ (read the differential of f at x acting on v). Thus, the differential $df(x)$ is an element of the dual space to the tangent space at x.

On the other hand, the gradient of a function at x, written $\mathrm{grad} f(x)$ is a vector in the tangent space at x. It is defined by duality as follows:

$$(\mathrm{grad} f(x), v)_g = df(x)v, \tag{11.4.1}$$

for every vector v in the tangent space at x. On the left hand side, we have the inner-product between the vectors v and $\mathrm{grad} f(x)$. On the right hand side, we have the differential acting on v. Since this equality holds for all v, it determines the gradient completely.

The main reason for being so cautious with this definition is that the idea of differentiation is distinct from the idea of distance. Conceptually, the notion of differentiation requires only the notion of a smooth curve. The role of the metric is to convert the differential into a tangent vector at x as shown in Figure 11.4.1.

Gradient flows may now be defined in a form that is the natural generalization of equation (11.2.1)

$$\dot x = -\mathrm{grad}_g f(x). \tag{11.4.2}$$

The fundamental estimate for gradient flows remains largely the same, though we must use inner-products with respect to the metric g. Thus, when $x(t)$ solves (11.4.3) we have

$$\frac{d}{dt} f(x(t)) = -(\mathrm{grad}_g f(x), \dot x)_g = -|\mathrm{grad} f(x)|_g^2. \tag{11.4.3}$$

11.5 Linear programming and Karmarkar's flow

We illustrate the idea of a gradient flow on two fundamental classes of optimization problems, linear and semidefinite programming (abbreviated LP and SDP respectively).

Linear programming was independently formulated during the 1940s in the US (by Dantzig, Koopmans, von Neumann) and the former Soviet Union (by Kantorovich). These methods were introduced to solve problems of logistics and economic planning during the second world war. Today they constitute one of the major applications of mathematics and underly all modern scheduling applications (e.g. communication and transportation systems). The main modeling ideas in LP and SDP are

1. Affine equalities are used to model constraints (e.g on resources).
2. The cost function is affine.

The standard form of a linear program is as follows. We seek to maximize the cost function

$$c^T x, \tag{11.5.1}$$

subject to m constraints

$$a_k^T x \leq b_k, \tag{11.5.2}$$

and the positivity constraint

$$x_j \geq 0, \quad 1 \leq j \leq n. \tag{11.5.3}$$

Here n is the dimension of the ambient space; x, c and each a_j are vectors in $\mathbb{R}^n$. The number of constraints, m, has no relation to n.

The constraints (11.5.2) and (11.5.3) determine a convex polytope $\mathcal{P}$, called the feasible region. Since a linear function achieves its maximum on a convex set at a boundary point, it is immediate that the solution to (11.5.1) must lie on the boundary $\mathcal{S} = \partial\mathcal{P}$.

The main computational task in linear programming is to find extremizing boundary point(s) efficiently. In order to get a feel for this task, let us consider a specific linear programming problem called the *assignment problem*, shown in Figure 11.5.1. Assume the number of vertices, indexed by i in Figure 11.5.1, is n.

Strictly speaking, the state space for this is the set of permutations on n symbols. We may reduce it to a linear programming problem in $\mathbb{R}^n$ by embedding the set of permutations into $\mathbb{R}^n$ as follows: map each permutation σ (which is a map from $i \in \{1, \ldots, n\}$ to $\{1, \ldots, n\}$) to the point $(\sigma(1), \ldots, \sigma(n)) \in \mathbb{R}^n$. These points form the vertices of a convex polytope called the permutohedron. The cost function in Figure 11.5.1 is now well-defined at each vertex, and by linearity, on each point in the interior of the permutohedron. This example illustrates a typical challenge in linear programming: we have a minimization problem that is simple to state, but it is defined on a finite set that is too large to search ($n!$ in this case).

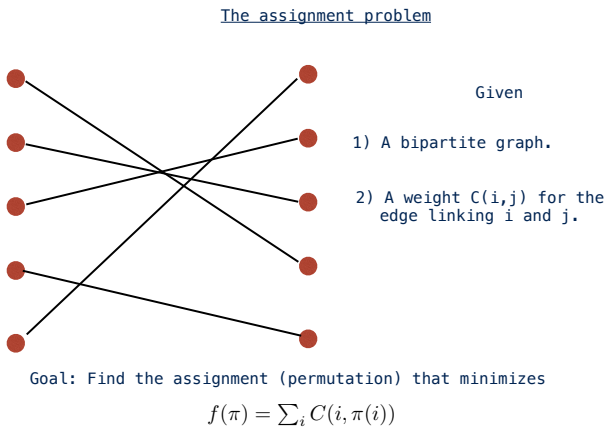

Figure 11.5.1: The assignment problem

.

Bibliography

[1] R. Bhatia, *Positive definite matrices*, vol. 24, Princeton University Press, 2009.

[2] J. W. Cooley and J. W. Tukey, *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19 (1965), pp. 297–301.

[3] P. Deift, C. Levermore, and C. Wayne, *Dynamical systems and probabilistic methods in partial differential equations*, American Mathematical Society, Providence,, 31 (1996), pp. 103–138.

[4] J. W. Demmel, *Applied numerical linear algebra*, vol. 56, SIAM, 1997.

[5] P. Diaconis, *Group representations in probability and statistics*, vol. 11, Institute of Mathematical Statistics, 1988.

[6] R. P. Feynman, *Feynman lectures on computation*, CRC Press, 2018.

[7] K. Fukushima, *Neocognitron: A hierarchical neural network capable of visual pattern recognition*, Neural Networks, 1 (1988), pp. 119–130.

[8] S. Geman and D. Geman, *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*, IEEE Transactions on pattern analysis and machine intelligence, (1984), pp. 721–741.

[9] E. T. Jaynes, *Information theory and statistical mechanics*, Physical Review, 106 (1957), p. 620.

[10] A. S. Lapedes and R. M. Farber, *How neural nets work*, in Neural Information Processing Systems, 1988, pp. 442–456.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, Proceedings of the IEEE, 86 (1998), pp. 2278–2324.

[12] D. Mumford and A. Desolneux, *Pattern theory: the stochastic analysis of real-world signals*, CRC Press, 2010.

[13] C. E. Shannon, *Prediction and entropy of printed English*, Bell System Technical Journal, 30 (1951), pp. 50–64.

[14] C. E. Shannon, *The mathematical theory of communication*, University of Illinois Press, 1962.

[15] L. N. Trefethen and D. Bau III, *Numerical linear algebra*, vol. 50, SIAM, 1997.

[16] V. Vapnik, *The nature of statistical learning theory*, Springer Science & Business Media, 2013.