6.450: Principles of Digital Communication

Chapters 1 to 3

Robert G. Gallager, M.I.T.

January 12, 2007

Preface: introduction and objectives

The digital communication industry is an enormous and rapidly growing industry, roughly comparable in size to the computer industry. The objective of this text is to study those aspects of digital communication systems that are unique to those systems. That is, rather than focusing on hardware and software for these systems, which is much like hardware and software for many other kinds of systems, we focus on the fundamental system aspects of modern digital communication.

Digital communication is a field in which theoretical ideas have had an unusually powerful impact on system design and practice. The basis of the theory was developed in 1948 by Claude Shannon, and is called information theory. For the first 25 years or so of its existence, information theory served as a rich source of academic research problems and as a tantalizing suggestion that communication systems could be made more efficient and more reliable by using these approaches. Other than small experiments and a few highly specialized military systems, the theory had little interaction with practice. By the mid 1970's, however, mainstream systems using information theoretic ideas began to be widely implemented. The first reason for this was the increasing number of engineers who understood both information theory and communication system practice. The second reason was that the low cost and increasing processing power of digital hardware made it possible to implement the sophisticated algorithms suggested by information theory. The third reason was that the increasing complexity of communication systems required the architectural principles of information theory.

The theoretical principles here fall roughly into two categories - the first provide analytical tools for determining the performance of particular systems, and the second put fundamental limits on the performance of any system. Much of the first category can be understood by engineering undergraduates, while the second category is distinctly graduate in nature. It is not that graduate students know so much more than undergraduates, but rather that undergraduate engineering students are trained to master enormous amounts of detail and to master the equations that deal with that detail. They are not used to the patience and deep thinking required to understand abstract performance limits. This patience comes later with thesis research.

My original purpose was to write an undergraduate text on digital communication, but experience teaching this material over a number of years convinced me that I could not write an honest exposition of principles, including both what is possible and what is not possible, without losing most undergraduates. There are many excellent undergraduate texts on digital communication describing a wide variety of systems, and I didn't see the need for another. Thus this text is now aimed at graduate students, but accessible to patient undergraduates.

The relationship between theory, problem sets, and engineering/design in an academic subject is rather complex. The theory deals with relationships and analysis for *models* of real systems. A good theory (and information theory is one of the best) allows for simple analysis of simplified models. It also provides structural principles that allow insights from these simple models to be applied to more complex and realistic models. Problem sets provide students with an opportunity to analyze these highly simplified models, and, with patience, to start to understand the general principles. Engineering deals with making the approximations and judgment calls to create simple models that focus on the critical elements of a situation, and from there to design workable systems.

The important point here is that engineering (at this level) cannot really be separated from theory. Engineering is necessary to choose appropriate theoretical models, and theory is necessary to find the general properties of those models. To oversimplify it, engineering determines what the reality is and theory determines the consequences and structure of that reality. At a deeper level, however, the engineering perception of reality heavily depends on the perceived structure (all of us carry oversimplified models around in our heads). Similarly, the structures created by theory depend on engineering common sense to focus on important issues. Engineering sometimes becomes overly concerned with detail, and theory overly concerned with mathematical niceties, but we shall try to avoid both these excesses here.

Each topic in the text is introduced with highly oversimplified toy models. The results about these toy models are then related to actual communication systems and this is used to generalize the models. We then iterate back and forth between analysis of models and creation of models. Understanding the performance limits on classes of models is essential in this process.

There are many exercises designed to help understand each topic. Some give examples showing how an analysis breaks down if the restrictions are violated. Since analysis always treats models rather than reality, these examples build insight into how the results about models apply to real systems. Other exercises apply the text results to very simple cases and others generalize the results to more complex systems. Yet others explore the sense in which theoretical models apply to particular practical problems.

It is important to understand that the purpose of the exercises is not so much to get the 'answer' as to acquire understanding. Thus students using this text will learn much more if they discuss the exercises with others and think about what they have learned after completing the exercise. The point is not to manipulate equations (which computers can now do better than students) but rather to understand the equations (which computers can not do).

As pointed out above, the material here is primarily graduate in terms of abstraction and patience, but requires only a knowledge of elementary probability, linear systems, and simple mathematical abstraction, so it can be understood at the undergraduate level. For both undergraduates and graduates, I feel strongly that learning to reason about engineering material is more important, both in the workplace and in further education, than learning to pattern match and manipulate equations.

Most undergraduate communication texts aim at familiarity with a large variety of different systems that have been implemented historically. This is certainly valuable in the workplace, at least for the near term, and provides a rich set of examples that are valuable for further study. The digital communication field is so vast, however, that learning from examples is limited, and in the long term it is necessary to learn the underlying principles. The examples from undergraduate courses provide a useful background for studying these principles, but the ability to reason abstractly that comes from elementary pure mathematics courses is equally valuable.

Most graduate communication texts focus more on the analysis of problems with less focus on the modeling, approximation, and insight needed to see how these problems arise. Our objective here is to use simple models and approximations as a way to understand the general principles. We will use quite a bit of mathematics in the process, but the mathematics will be used to establish general results precisely rather than to carry out detailed analyses of special cases.

Contents

Chapter 1

Introduction to digital communication

Communication has been one of the deepest needs of the human race throughout recorded history. It is essential to forming social unions, to educating the young, and to expressing a myriad of emotions and needs. Good communication is central to a civilized society.

The various communication disciplines in engineering have the purpose of providing technological aids to human communication. One could view the smoke signals and drum rolls of primitive societies as being technological aids to communication, but communication technology as we view it today became important with telegraphy, then telephony, then video, then computer communication, and today the amazing mixture of all of these in inexpensive, small portable devices.

Initially these technologies were developed as separate networks and were viewed as having little in common. As these networks grew, however, the fact that all parts of a given network had to work together, coupled with the fact that different components were developed at different times using different design principles, forced increasing attention on developing both understanding and archtectural principles to make the systems continue to evolve.

This need for understanding was probably best understood at American Telephone and Telegraph (AT&T) where Bell Laboratories was created as the research and development arm of AT&T. The Math center at Bell Labs became the predominant center for communication research in the world, and held that position until quite recently. The central core of the principles of communication technology were developed at that center.

Perhaps the greatest contribution from the math center was the creation of Information Theory [17] by Claude Shannon in 1948. For perhaps the first 25 years of its existence, Information Theory was regarded as a beautiful theory but not as a central guide to the architecture and design of communication systems. After that time, however, both the device technology and the engineering understanding of the theory were sufficient to enable system development to follow information theoretic principles.

A number of information theoretic ideas and how they affect communication system design will be explained carefully in subsequent chapters. One pair of ideas, however, is central to almost every topic. The first is to view all communication sources, e.g., speech waveforms, image waveforms, text files, as being representable by binary sequences. The second is to design communication systems that first convert the source output into a binary sequence and then convert that binary sequence into a form suitable for transmission over particular physical media such as cable, twisted wire pair, optical fiber, or electromagnetic radiation through space.

Digital communication systems, by definition, are communication systems that use such a digital[1] sequence as an interface between the source and the channel input (and similarly between the channel output and final destination) (see Figure 1.1).

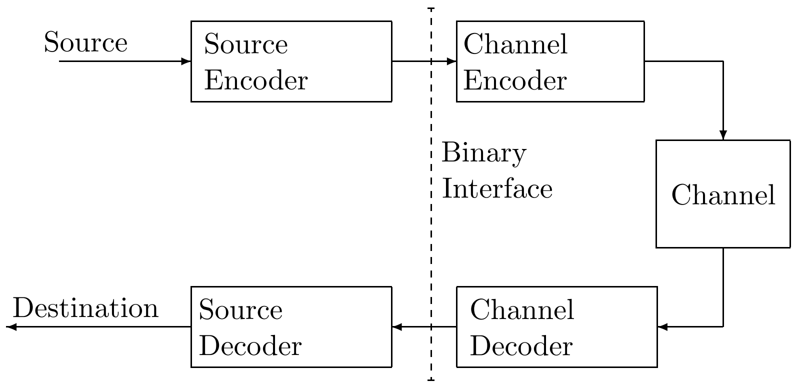


Figure 1.1: Placing a binary interface between source and channel. The source encoder converts the source output to a binary sequence and the channel encoder (often called a modulator) processes the binary sequence for transmission over the channel. The channel decoder (demodulator) recreates the incoming binary sequence (hopefully reliably), and the source decoder recreates the source output.

The idea of converting an analog source output to a binary sequence was quite revolutionary in 1948, and the notion that this should be done before channel processing was even more revolutionary. By today, with digital cameras, digital video, digital voice, etc., the idea of digitizing any kind of source is commonplace even among the most technophobic. The notion of a binary interface before channel transmission is almost as commonplace. For example, we all refer to the speed of our internet connection in bits per second.

There are a number of reasons why communication systems now usually contain a binary interface between source and channel (*i.e.*, why digital communication systems are now standard). These will be explained with the necessary qualifications later, but briefly they are as follows:

- Digital hardware has become so cheap, reliable, and miniaturized, that digital interfaces are eminently practical.
- A standardized binary interface between source and channel simplifies implementation and understanding, since source coding/decoding can be done independently of the channel, and, similarly, channel coding/decoding can be done independently of the source.

[1]A digital sequence is a sequence made up of elements from a finite alphabet (*e.g.*, the binary digits $\{0, 1\}$, the decimal digits $\{0, 1, \dots, 9\}$, or the letters of the English alphabet) . The binary digits are almost universally used for digital communication and storage, so we only distinguish digital from binary in those few places where the difference is significant.

- A standardized binary interface between source and channel simplifies networking, which now reduces to sending binary sequences through the network.
- One of the most important of Shannon's information theoretic results is that if a source can be transmitted over a channel in any way at all, it can be transmitted using a binary interface between source and channel. This is known as the *source/channel separation theorem.*

In the remainder of this chapter, the problems of source coding and decoding and channel coding and decoding are briefly introduced. First, however, the notion of layering in a communication system is introduced. One particularly important example of layering was already introduced in Figure 1.1, where source coding and decoding are viewed as one layer and channel coding and decoding are viewed as another layer.

1.1 Standardized interfaces and layering

Large communication systems such as the Public Switched Telephone Network (PSTN) and the Internet have incredible complexity, made up of an enormous variety of equipment made by different manufacturers at different times following different design principles. Such complex networks need to be based on some simple architectural principles in order to be understood, managed, and maintained.

Two such fundamental architectural principles are *standardized interfaces* and *layering.*

A standardized interface allows the user or equipment on one side of the interface to ignore all details about the other side of the interface except for certain specified interface characteristics. For example, the binary interface[2] above allows the source coding/decoding to be done independently of the channel coding/decoding.

The idea of layering in communication systems is to break up communication functions into a string of separate layers as illustrated in Figure 1.2.

Each layer consists of an input module at the input end of a communcation system and a 'peer' output module at the other end. The input module at layer i processes the information received from layer $i{+}1$ and sends the processed information on to layer $i{-}1$. The peer output module at layer i works in the opposite direction, processing the received information from layer $i{-}1$ and sending it on to layer i.

As an example, an input module might receive a voice waveform from the next higher layer and convert the waveform into a binary data sequence that is passed on to the next lower layer. The output peer module would receive a binary sequence from the next lower layer at the output and convert it back to a speech waveform.

As another example, a *modem* consists of an input module (a $\underline{\text{mo}}$dulator) and an output module (a $\underline{\text{dem}}$odulator). The modulator receives a binary sequence from the next higher input layer and generates a corresponding modulated waveform for transmission over a channel. The peer module is the remote demodulator at the other end of the channel. It receives a more-or-less faithful replica of the transmitted waveform and reconstructs a typically faithful replica of the binary sequence. Similarly, the local demodulator is the peer to a remote modulator (often collocated with the remote demodulator above). Thus a modem is an input module for

[2]The use of a binary sequence at the interface is not quite enough to specify it, as will be discussed later.

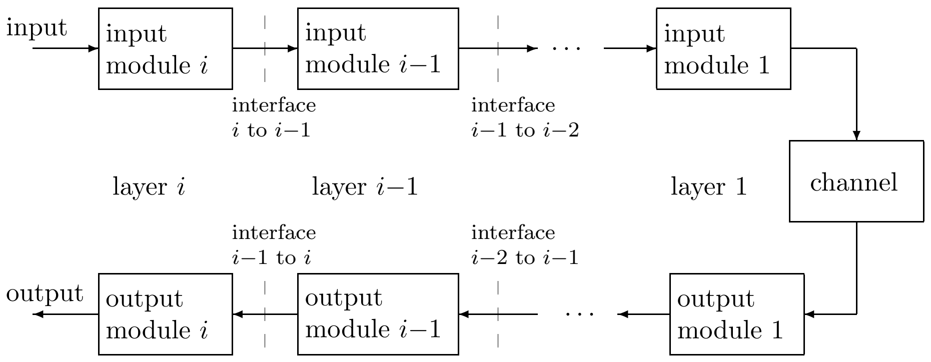


Figure 1.2: Layers and interfaces: The specification of the interface between layers i and $i-1$ should specify how input module i communicates with input module $i-1$, how the corresponding output modules communicate, and, most important, the input/output behavior of the system to the right of interface. The designer of layer $i-1$ uses the input/output behavior of the layers to the right of $i-1$ to produce the required input/output performance to the right of layer i. Later examples will show how this multi-layer process can simplify the overall system design.

communication in one direction and an output module for independent communication in the opposite direction. Later chapters consider modems in much greater depth, including how noise affects the channel waveform and how that affects the reliability of the recovered binary sequence at the output. For now, however, it is enough to simply view the modulator as converting a binary sequence to a waveform, with the peer demodulator converting the waveform back to the binary sequence.

As another example, the source coding/decoding layer for a waveform source can be split into 3 layers as shown in Figure 1.3. One of the advantages of this layering is that discrete sources are an important topic in their own right (treated in Chapter 2) and correspond to the inner layer of Figure 1.3. Quantization is also an important topic in its own right, (treated in Chapter 3). After both of these are understood, waveform sources become quite simple to understand.

The channel coding/decoding layer can also be split into several layers, but there are a number of ways to do this which will be discussed later. For example, binary error-correction coding/decoding can be used as an outer layer with modulation and demodulation as an inner layer, but it will be seen later that there are a number of advantages in combining these layers into what is called coded modulation.[3] Even here, however, layering is important, but the layers are defined differently for different purposes.

It should be emphasized that layering is much more than simply breaking a system into components. The input and peer output in each layer encapsulate all the lower layers, and all these lower layers can be viewed in aggregate as a communication channel. Similarly, the higher layers can be viewed in aggregate as a simple source and destination.

The above discussion of layering implicitly assumed a point-to-point communication system with one source, one channel, and one destination. Network situations can be considerably

[3]Notation is nonstandard here. A channel coder (including both coding and modulation) is often referred to (both here and elsewhere) as a modulator. It is also often referred to as a modem, although a modem is really a device that contains both modulator for communication in one direction and demodulator for communication in the other.

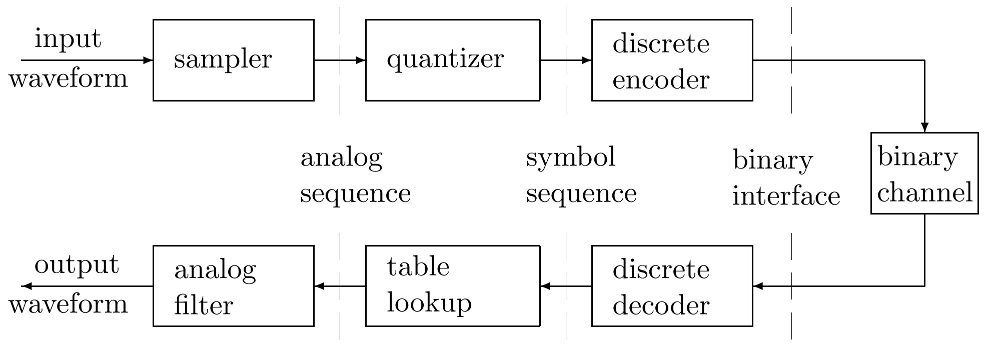


Figure 1.3: Breaking the source coding/decoding layer into 3 layers for a waveform source. The input side of the outermost layer converts the waveform into a sequence of samples and output side converts the recovered samples back to the waveform. The quantizer then converts each sample into one of a finite set of symbols, and the peer module recreates the sample (with some distortion). Finally the inner layer encodes the sequence of symbols into binary digits.

more complex. With broadcasting, an input module at one layer may have multiple peer output modules. Similarly, in multiaccess communication a multiplicity of input modules have a single peer output module. It is also possible in network situations for a single module at one level to interface with multiple modules at the next lower layer or the next higher layer. The use of layering is at least as important for networks as for point-to-point communications systems. The physical layer for networks is essentially the channel encoding/decoding layer discussed here, but textbooks on networks rarely discuss these physical layer issues in depth. The network control issues at other layers are largely separable from the physical layer communication issues stressed here. The reader is referred to [1], for example, for a treatment of these control issues.

The following three sections give a fuller discussion of the components of Figure 1.1, *i.e.*, of the fundamental two layers (source coding/decoding and channel coding/decoding) of a point-to-point digital communication system, and finally of the interface between them.

1.2 Communication sources

The source might be discrete, *i.e.*, it might produce a sequence of discrete symbols, such as letters from the English or Chinese alphabet, binary symbols from a computer file, etc. Alternatively, the source might produce an analog waveform, such as a voice signal from a microphone, the output of a sensor, a video waveform, etc. Or, it might be a sequence of images such as X-rays, photographs, etc.

Whatever the nature of the source, the output from the source will be modeled as a sample function of a random process. It is not obvious why the inputs to communication systems should be modeled as random, and in fact this was not appreciated before Shannon developed information theory in 1948.

The study of communication before 1948 (and much of it well after 1948) was based on Fourier analysis; basically one studied the effect of passing sine waves through various kinds of systems and components and viewed the source signal as a superposition of sine waves. Our study of channels will begin with this kind of analysis (often called Nyquist theory) to develop basic results about sampling, intersymbol interference, and bandwidth.

Shannon's view, however, was that if the recipient knows that a sine wave of a given frequency is to be communicated, why not simply regenerate it at the output rather than send it over a long distance? Or, if the recipient knows that a sine wave of unknown frequency is to be communicated, why not simply send the frequency rather than the entire waveform?

The essence of Shannon's viewpoint is that the set of possible source outputs, rather than any particular output, is of primary interest. The reason is that the communication system must be designed to communicate whichever one of these possible source outputs actually occurs. The objective of the communication system then is to transform each possible source output into a transmitted signal in such a way that these possible transmitted signals can be best distinguished at the channel output. A probability measure is needed on this set of possible source outputs to distinguish the typical from the atypical. This point of view drives the discussion of all components of communication systems throughout this text.

1.2.1 Source coding

The source encoder in Figure 1.1 has the function of converting the input from its original form into a sequence of bits. As discussed before, the major reasons for this almost universal conversion to a bit sequence are as follows: digital hardware, standardized interfaces, layering, and the source/channel separation theorem.

The simplest source coding techniques apply to discrete sources and simply involve representing each succesive source symbol by a sequence of binary digits. For example, letters from the 27-symbol English alphabet (including a SPACE symbol) may be encoded into 5-bit blocks. Since there are 32 distinct 5-bit blocks, each letter may be mapped into a distinct 5-bit block with a few blocks left over for control or other symbols. Similarly, upper-case letters, lower-case letters, and a great many special symbols may be converted into 8-bit blocks ("bytes") using the standard ASCII code.

Chapter 2 treats coding for discrete sources and generalizes the above techniques in many ways. For example the input symbols might first be segmented into m-tuples, which are then mapped into blocks of binary digits. More generally yet, the blocks of binary digits can be generalized into variable-length sequences of binary digits. We shall find that any given discrete source, characterized by its alphabet and probabilistic description, has a quantity called *entropy* associated with it. Shannon showed that this source entropy is equal to the minimum number of binary digits per source symbol required to map the source output into binary digits in such a way that the source symbols may be retrieved from the encoded sequence.

Some discrete sources generate finite segments of symbols, such as email messages, that are statistically unrelated to other finite segments that might be generated at other times. Other discrete sources, such as the output from a digital sensor, generate a virtually unending sequence of symbols with a given statistical characterization. The simpler models of Chapter 2 will correspond to the latter type of source, but the discussion of universal source coding in Section 2.9 is sufficiently general to cover both types of sources, and virtually any other kind of source.

The most straightforward approach to analog source coding is called analog to digital (A/D) conversion. The source waveform is first sampled at a sufficiently high rate (called the “Nyquist rate”). Each sample is then quantized sufficiently finely for adequate reproduction. For example, in standard voice telephony, the voice waveform is sampled 8000 times per second; each sample is then quantized into one of 256 levels and represented by an 8-bit byte. This yields a source coding bit rate of 64 Kbps.

Beyond the basic objective of conversion to bits, the source encoder often has the further objective of doing this as efficiently as possible— *i.e.*, transmitting as few bits as possible, subject to the need to reconstruct the input adequately at the output. In this case source encoding is often called data compression. For example, modern speech coders can encode telephone-quality speech at bit rates of the order of 6-16 kb/s rather than 64 kb/s.

The problems of sampling and quantization are largely separable. Chapter 3 develops the basic principles of quantization. As with discrete source coding, it is possible to quantize each sample separately, but it is frequently preferable to segment the samples into n-tuples and then quantize the resulting n-tuples. As shown later, it is also often preferable to view the quantizer output as a discrete source output and then to use the principles of Chapter 2 to encode the quantized symbols. This is another example of layering.

Sampling is one of the topics in Chapter 4. The purpose of sampling is to convert the analog source into a sequence of real-valued numbers, *i.e.*, into a discrete-time, analog-amplitude source. There are many other ways, beyond sampling, of converting an analog source to a discrete-time source. A general approach, which includes sampling as a special case, is to expand the source waveform into an orthonormal expansion and use the coefficients of that expansion to represent the source output. The theory of orthonormal expansions is a major topic of Chapter 4. It forms the basis for the signal space approach to channel encoding/decoding. Thus Chapter 4 provides us with the basis for dealing with waveforms both for sources and channels.

1.3 Communication channels

We next discuss the channel and channel coding in a generic digital communication system.

In general, a channel is viewed as that part of the communication system between source and destination that is given and not under the control of the designer. Thus, to a source-code designer, the channel might be a digital channel with binary input and output; to a telephone-line modem designer, it might be a 4 KHz voice channel; to a cable modem designer, it might be a physical coaxial cable of up to a certain length, with certain bandwidth restrictions.

When the channel is taken to be the physical medium, the amplifiers, antennas, lasers, etc. that couple the encoded waveform to the physical medium might be regarded as part of the channel or as as part of the channel encoder. It is more common to view these coupling devices as part of the channel, since their design is quite separable from that of the rest of the channel encoder. This, of course, is another example of layering.

Channel encoding and decoding when the channel is the physical medium (either with or without amplifiers, antennas, lasers, etc.) is usually called *(digital) modulation* and *demodulation* respectively. The terminology comes from the days of analog communication where modulation referred to the process of combining a lowpass signal waveform with a high frequency sinusoid, thus placing the signal waveform in a frequency band appropriate for transmission and regulatory requirements. The analog signal waveform could modulate the amplitude, frequency, or phase, for example, of the sinusoid, but in any case, the original waveform (in the absence of noise) could be retrieved by demodulation.

As digital communication has increasingly replaced analog communication, the modulation/demodulation terminology has remained, but now refers to the entire process of digital encoding and decoding. In most such cases, the binary sequence is first converted to a baseband waveform and the resulting baseband waveform is converted to bandpass by the same type of procedure used for analog modulation. As will be seen, the challenging part of this problem is the conversion of binary data to baseband waveforms. Nonetheless, this entire process will be referred to as modulation and demodulation, and the conversion of baseband to passband and back will be referred to as frequency conversion.

As in the study of any type of system, a channel is usually viewed in terms of its possible inputs, its possible outputs, and a description of how the input affects the output. This description is usually probabilistic. If a channel were simply a linear time-invariant system (*e.g.*, a filter), then it could be completely characterized by its impulse response or frequency response. However, the channels here (and channels in practice) always have an extra ingredient— noise.

Suppose that there were no noise and a single input voltage level could be communicated exactly. Then, representing that voltage level by its infinite binary expansion, it would be possible in principle to transmit an infinite number of binary digits by transmitting a single real number. This is ridiculous in practice, of course, precisely because noise limits the number of bits that can be reliably distinguished. Again, it was Shannon, in 1948, who realized that noise provides the fundamental limitation to performance in communication systems.

The most common channel model involves a waveform input $X(t)$, an added noise waveform $Z(t)$, and a waveform output $Y(t) = X(t) + Z(t)$ that is the sum of the input and the noise, as shown in Figure 1.4. Each of these waveforms are viewed as random processes. Random processes are studied in Chapter 7, but for now they can be viewed intuitively as waveforms selected in some probabilitistic way. The noise $Z(t)$ is often modeled as white Gaussian noise (also to be studied and explained later). The input is usually constrained in power and bandwidth.

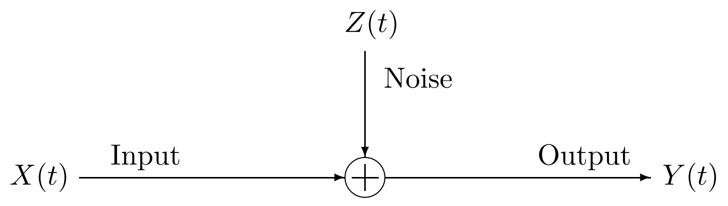


Figure 1.4: An additive white Gaussian noise (AWGN) channel.

Observe that for any channel with input $X(t)$ and output $Y(t)$, the noise could be defined to be $Z(t) = Y(t) - X(t)$. Thus there must be something more to an additive-noise channel model than what is expressed in Figure 1.4. The additional required ingredient for noise to be called additive is that its probabilistic characterization does not depend on the input.

In a somewhat more general model, called a *linear Gaussian channel*, the input waveform $X(t)$ is first filtered in a linear filter with impulse response $h(t)$, and then independent white Gaussian noise $Z(t)$ is added, as shown in Figure 1.5, so that the channel output is

$$Y(t) = X(t) * h(t) + Z(t),$$

where "$*$" denotes convolution. Note that Y at time t is a function of X over a range of times, *i.e.*,

$$Y(t) = \int_{-\infty}^{\infty} X(t-\tau)h(\tau)\,d\tau + Z(t)$$

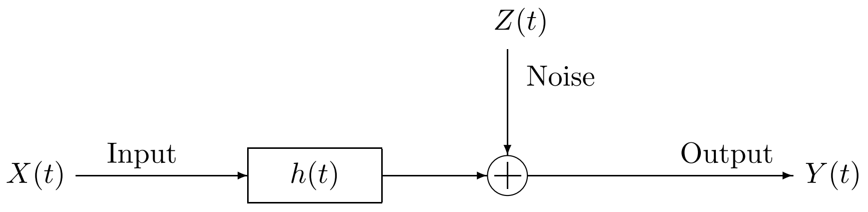


Figure 1.5: Linear Gaussian channel model.

The linear Gaussian channel is often a good model for wireline communication and for line-of-sight wireless communication. When engineers, journals, or texts fail to describe the channel of interest, this model is a good bet.

The linear Gaussian channel is a rather poor model for non-line-of-sight mobile communication. Here, multiple paths usually exist from source to destination. Mobility of the source, destination, or reflecting bodies can cause these paths to change in time in a way best modeled as random. A better model for mobile communication is to replace the time-invariant filter $h(t)$ in Figure 1.5 by a randomly-time-varying linear filter, $H(t,\tau)$, that represents the multiple paths as they change in time. Here the output is given by $Y(t) = \int_{-\infty}^{\infty} X(t-u)H(u,t)du + Z(t)$. These randomly varying channels will be studied in Chapter 9.

1.3.1 Channel encoding (modulation)

The channel encoder box in Figure 1.1 has the function of mapping the binary sequence at the source/channel interface into a channel waveform. A particularly simple approach to this is called binary pulse amplitude modulation (2-PAM). Let $\{u_1, u_2, \dots,\}$ denote the incoming binary sequence, and let each $u_n = \pm 1$ (rather than the traditional 0/1). Let $p(t)$ be a given elementary waveform such as a rectangular pulse or a $\frac{sin(\omega t)}{\omega t}$ function. Assuming that the binary digits enter at R bits per second (bps), the sequence $u_1, u_2, \dots$ is mapped into the waveform $\sum_n u_n p(t - \frac{n}{R})$.

Even with this trivially simple modulation scheme, there are a number of interesting questions, such as how to choose the elementary waveform $p(t)$ so as to satisfy frequency constraints and reliably detect the binary digits from the received waveform in the presence of noise and intersymbol interference.

Chapter 6 develops the principles of modulation and demodulation. The simple 2-PAM scheme is generalized in many ways. For example, multi-level modulation first segments the incoming bits into m-tuples. There are $M = 2^m$ distinct m-tuples, and in M-PAM, each m-tuple is mapped into a different numerical value (such as $\pm 1, \pm 3, \pm 5, \pm 7$ for $M = 8$). The sequence

$u_1, u_2, \ldots$ of these values is then mapped into the waveform $\sum_n u_n p(t - \frac{mn}{R})$. Note that the rate at which pulses are sent is now m times smaller than before, but there are 2^m different values to be distinguished at the receiver for each elementary pulse.

The modulated waveform can also be a complex baseband waveform (which is then modulated up to an appropriate passband as a real waveform). In a scheme called quadrature amplitude modulation (QAM), the bit sequence is again segmented into m-tuples, but now there is a mapping from binary m-tuples to a set of $M = 2^m$ complex numbers. The sequence $u_1, u_2, \ldots,$ of outputs from this mapping is then converted to the complex waveform $\sum_n u_n p(t - \frac{mn}{R})$.

Finally, instead of using a fixed signal pulse $p(t)$ multiplied by a selection from M real or complex values, it is possible to choose M different signal pulses, $p_1(t), \ldots, p_M(t)$. This includes frequency shift keying, pulse position modulation, phase modulation, and a host of other strategies.

It is easy to think of many ways to map a sequence of binary digits into a waveform. We shall find that there is a simple geometric "signal-space" approach, based on the results of Chapter 4, for looking at these various combinations in an integrated way.

Because of the noise on the channel, the received waveform is different from the transmitted waveform. A major function of the demodulator is that of detection. The detector attempts to choose which possible input sequence is most likely to have given rise to the given received waveform. Chapter 7 develops the background in random processes necessary to understand this problem, and Chapter 8 uses the geometric signal-space approach to analyze and understand the detection problem.

1.3.2 Error correction

Frequently the error probability incurred with simple modulation and demodulation techniques is too high. One possible solution is to separate the channel encoder into two layers, first an error-correcting code, and then a simple modulator.

As a very simple example, the bit rate into the channel encoder could be reduced by a factor of 3, and then each binary input could be repeated 3 times before entering the modulator. If at most one of the 3 binary digits coming out of the demodulator were incorrect, it could be corrected by majority rule at the decoder, thus reducing the error probability of the system at a considerable cost in data rate.

The scheme above (repetition encoding followed by majority-rule decoding) is a very simple example of error-correction coding. Unfortunately, with this scheme, small error probabilities are achieved only at the cost of very small transmission rates.

What Shannon showed was the very unintuitive fact that more sophisticated coding schemes can achieve arbitrarily low error probability at any data rate above a value known as the *channel capacity*. The channel capacity is a function of the probabilistic description of the output conditional on each possible input. Conversely, it is not possible to achieve low error probability at rates above the channel capacity. A brief proof of this *channel coding theorem* is given in Chapter 8, but readers should refer to texts on Information Theory such as [4] or [3]) for detailed coverage.

The channel capacity for a bandlimited additive white Gaussian noise channel is perhaps the most famous result in information theory. If the input power is limited to P, the bandwidth limited to W, and the noise power per unit bandwidth is N_0, then the capacity (in bits per second) is

$$C = W \log_2 \left(1 + \frac{P}{N_0 W} \right).$$

Only in the past few years have channel coding schemes been developed that can closely approach this channel capacity.

Early uses of error-correcting codes were usually part of a two-layer system similar to that above, where a digital error-correcting encoder is followed by a modulator. At the receiver, the waveform is first demodulated into a noisy version of the encoded sequence, and then this noisy version is decoded by the error-correcting decoder. Current practice frequently achieves better performance by combining error correction coding and modulation together in coded modulation schemes. Whether the error correction and traditional modulation are separate layers or combined, the combination is generally referred to as a modulator and a device that does this modulation on data in one direction and demodulation in the other direction is referred to as a modem.

The subject of error correction has grown over the last 50 years to the point where complex and lengthy textbooks are dedicated to this single topic (see, for example, [?] and [?].) This text provides only an introduction to error-correcting codes.

The final topic of the text is channel encoding and decoding for wireless channels. Considerable attention is paid here to modeling physical wireless media. Wireless channels are subject not only to additive noise but also random fluctuations in the strength of multiple paths between transmitter and receiver. The interaction of these paths causes fading, and we study how this affects coding, signal selection, modulation, and detection. Wireless communication is also used to discuss issues such as channel measurement, and how these measurements can be used at input and output. Finally there is a brief case study of CDMA (code division multiple access), which ties together many of the topics in the text.

1.4 Digital interface

The interface between the source coding layer and the channel coding layer is a sequence of bits. However, this simple characterization does not tell the whole story. The major complicating factors are as follows:

- Unequal rates: The rate at which bits leave the source encoder is often not perfectly matched to the rate at which bits enter the channel encoder.
- Errors: Source decoders are usually designed to decode an exact replica of the encoded sequence, but the channel decoder makes occasional errors.
- Networks: Encoded source outputs are often sent over networks, traveling serially over several channels; each channel in the network typically also carries the output from a number of different source encoders.

The first two factors above appear both in point-to-point communication systems and in networks. They are often treated in an ad hoc way in point-to-point systems, whereas they must be treated in a standardized way in networks. The third factor, of course, must also be treated in a standardized way in networks.

The usual approach to these problems in networks is to convert the superficially simple binary interface above into multiple layers as illustrated in Figure 1.6

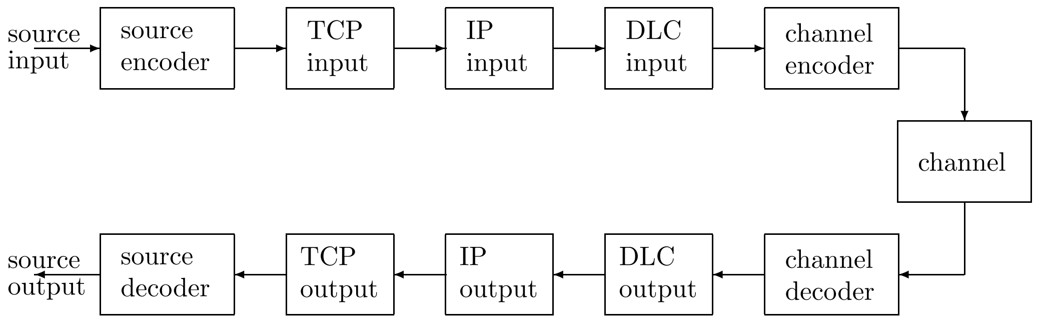


Figure 1.6: The replacement of the binary interface in Figure 1.6 with 3 layers in an oversimplified view of the internet: There is a TCP (transport control protocol) module associated with each source/destination pair; this is responsible for end-to-end error recovery and for slowing down the source when the network becomes congested. There is an IP (internet protocol) module associated with each node in the network; these modules work together to route data through the network and to reduce congestion. Finally there is a DLC (data link control) module associated with each channel; this accomplishes rate matching and error recovery on the channel. In network terminology, the channel, with its encoder and decoder, is called the *physical layer*.

How the layers in Figure 1.6 operate and work together is a central topic in the study of networks and is treated in detail in network texts such as [1]. These topics are not considered in detail here, except for the very brief introduction to follow and a few comments as needed later.

1.4.1 Network aspects of the digital interface

The output of the source encoder is usually segmented into packets (and in many cases, such as email and data files, is already segmented in this way). Each of the network layers then adds some overhead to these packets, adding a header in the case of TCP (transmission control protocol) and IP (internet protocol) and adding both a header and trailer in the case of DLC (data link control). Thus what enters the channel encoder is a sequence of frames, where each frame has the structure illustrated in Figure 1.7.

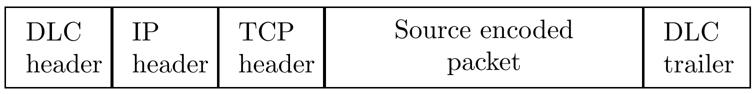


Figure 1.7: The structure of a data frame using the layers of Figure 1.6

.

These data frames, interspersed as needed by idle-fill, are strung together and the resulting bit stream enters the channel encoder at its synchronous bit rate. The header and trailer supplied by the DLC must contain the information needed for the receiving DLC to parse the received bit stream into frames and eliminate the idle-fill.

The DLC also provides protection against decoding errors made by the channel decoder. Typically this is done by using a set of 16 or 32 parity checks in the frame trailer. Each parity check specifies whether a given subset of bits in the frame contains an even or odd number of 1's. Thus if errors occur in transmission, it is highly likely that at least one of these parity checks will fail in the receiving DLC. This type of DLC is used on channels that permit transmission in both directions. Thus when an erroneous frame is detected, it is rejected and a frame in the opposite direction requests a retransmission of the erroneous frame. Thus the DLC header must contain information about frames traveling in both directions. For details about such protocols, see, for example, [1].

An obvious question at this point is why error correction is typically done both at the physical layer and at the DLC layer. Also, why is feedback (*i.e.*, error detection and retransmission) used at the DLC layer and not at the physical layer? A partial answer is that using both schemes together yields a smaller error probability than using either one separately. At the same time, combining both procedures (with the same overall overhead) and using feedback at the physical layer can result in much smaller error probabilities. The two layer approach is typically used in practice because of standardization issues, but in very difficult communication situations, the combined approach can be preferable. From a tutorial standpoint, however, it is preferable to acquire a good understanding of channel encoding and decoding using transmission in only one direction before considering the added complications of feedback.

When the receiving DLC accepts a frame, it strips off the DLC header and trailer and the resulting packet enters the IP layer. In the IP layer, the address in the IP header is inspected to determine whether the packet is at its destination or must be forwarded through another channel. Thus the IP layer handles routing decisions, and also sometimes the decision to drop a packet if the queues at that node are too long.

When the packet finally reaches its destination, the IP layer strips off the IP header and passes the resulting packet with its TCP header to the TCP layer. The TCP module then goes through another error recovery phase[4] much like that in the DLC module and passes the accepted packets, without the TCP header, on to the destination decoder. The TCP and IP layers are also jointly responsible for congestion control, which ultimately requires the ability to either reduce the rate from sources as required or to simply drop sources that cannot be handled (witness dropped cell-phone calls).

In terms of sources and channels, these extra layers simply provide a sharper understanding of the digital interface between source and channel. That is, source encoding still maps the source output into a sequence of bits, and from the source viewpoint, all these layers can simply be viewed as a channel to send that bit sequence reliably to the destination.

In a similar way, the input to a channel is a sequence of bits at the channel's synchronous input rate. The output is the same sequence, somewhat delayed and with occasional errors.

Thus both source and channel have digital interfaces, and the fact that these are slightly different because of the layering is in fact an advantage. The source encoding can focus solely on minimizing the output bit rate (perhaps with distortion and delay constraints) but can ignore

[4]Even after all these layered attempts to prevent errors, occasional errors are inevitable. Some are caught by human intervention, many don't make any real difference, and a final few have consequences. C'est la vie. The purpose of communication engineers and network engineers is not to eliminate all errors, which is not possible, but rather to reduce their probability as much as practically possible.

the physical channel or channels to be used in transmission. Similarly the channel encoding can ignore the source and focus solely on maximizing the transmission bit rate (perhaps with delay and error rate constraints).

Chapter 2

Coding for Discrete Sources

2.1 Introduction

A general block diagram of a point-to-point digital communication system was given in Figure 1.1. The source encoder converts the sequence of symbols from the source to a sequence of binary digits, preferably using as few binary digits per symbol as possible. The source decoder performs the inverse operation. Initially, in the spirit of source/channel separation, we ignore the possibility that errors are made in the channel decoder and assume that the source decoder operates on the source encoder output.

We first distinguish between three important classes of sources:

- **Discrete sources**

  The output of a discrete source is a sequence of symbols from a given discrete alphabet $\mathcal{X}$. This alphabet could be the alphanumeric characters, the characters on a computer keyboard, English letters, Chinese characters, the symbols in sheet music (arranged in some systematic fashion), binary digits, etc.

  The discrete alphabets in this chapter are assumed to contain a finite set of symbols.[1]

  It is often convenient to view the sequence of symbols as occurring at some fixed rate in time, but there is no need to bring time into the picture (for example, the source sequence might reside in a computer file and the encoding can be done off-line).

  This chapter focuses on source coding and decoding for discrete sources." Supplementary references for source coding are Chapter 3 of [4] and Chapter 5 of [3]. A more elementary partial treatment is in Sections 4.1-4.3 of [14].

- **Analog waveform sources**

  The output of an analog source, in the simplest case, is an analog real waveform, representing, for example, a speech waveform. The word analog is used to emphasize that the waveform can be arbitrary and is not restricted to taking on amplitudes from some discrete set of values.

[1] A set is usually defined to be discrete if it includes either a finite or countably infinite number of members. The countably infinite case does not extend the basic theory of source coding in any important way, but it is occasionally useful in looking at limiting cases, which will be discussed as they arise.

It is also useful to consider analog waveform sources with outputs that are complex functions of time; both real and complex waveform sources are discussed later.

More generally, the output of an analog source might be an image (represented as an intensity function of horizontal/vertical location) or video (represented as an intensity function of horizontal/vertical location and time). For simplicity, we restrict our attention to analog waveforms, mapping a single real variable, time, into a real or complex-valued intensity.

- **Discrete-time sources with analog values (analog sequence sources)**

  These sources are halfway between discrete and analog sources. The source output is a sequence of real numbers (or perhaps complex numbers). Encoding such a source is of interest in its own right, but is of interest primarily as a subproblem in encoding analog sources. That is, analog waveform sources are almost invariably encoded by first either sampling the analog waveform or representing it by the coefficients in a series expansion. Either way, the result is a sequence of numbers, which is then encoded.

There are many differences between discrete sources and the latter two types of analog sources. The most important is that a discrete source can be, and almost always is, encoded in such a way that the source output can be uniquely retrieved from the encoded string of binary digits. Such codes are called *uniquely decodable*[2]. On the other hand, for analog sources, there is usually no way to map the source values to a bit sequence such that the source values are uniquely decodable. For example, an infinite number of binary digits is required for the exact specification of an arbitrary real number between 0 and 1. Thus, some sort of quantization is necessary for these analog values, and this introduces distortion. Source encoding for analog sources thus involves a trade-off between the bit rate and the amount of distortion.

Analog sequence sources are almost invariably encoded by first quantizing each element of the sequence (or more generally each successive n-tuple of sequence elements) into one of a finite set of symbols. This symbol sequence is a discrete sequence which can then be encoded into a binary sequence.

Figure 2.1 summarizes this layered view of analog and discrete source coding. As illustrated, discrete source coding is both an important subject in its own right for encoding text-like sources, but is also the inner layer in the encoding of analog sequences and waveforms.

The remainder of this chapter discusses source coding for discrete sources. The following chapter treats source coding for analog sequences and the fourth chapter treats waveform sources.

2.2 Fixed-length codes for discrete sources

The simplest approach to encoding a discrete source into binary digits is to create a code $\mathcal{C}$ that maps each symbol x of the alphabet $\mathcal{X}$ into a distinct codeword $\mathcal{C}(x)$, where $\mathcal{C}(x)$ is a block of binary digits. Each such block is restricted to have the same block length L, which is why the code is called a *fixed-length code.*

[2]Uniquely-decodable codes are sometimes called noiseless codes in elementary treatments. *Uniquely decodable* captures both the intuition and the precise meaning far better than *noiseless*. Unique decodability is defined shortly.

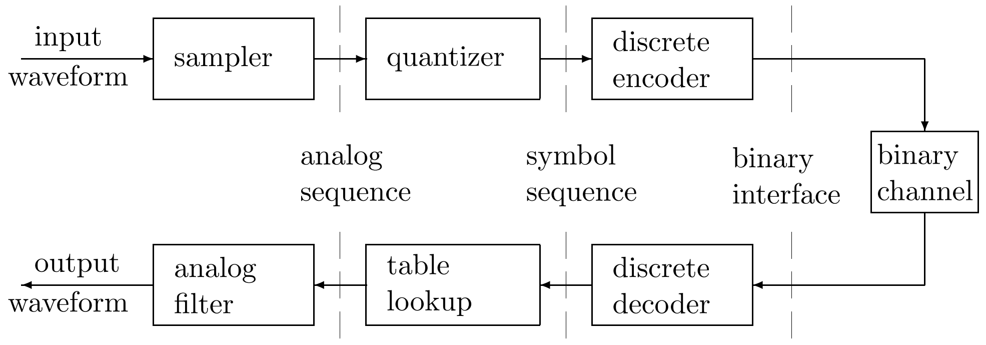


Figure 2.1: Discrete sources require only the inner layer above, whereas the inner two layers are used for analog sequences and all three layers are used for waveforms sources.

For example, if the alphabet $\mathcal{X}$ consists of the 7 symbols $\{a, b, c, d, e, f, g\}$, then the following fixed-length code of block length $L = 3$ could be used.

$$\begin{aligned}
\mathcal{C}(a) &= \quad 000\\
\mathcal{C}(b) &= \quad 001\\
\mathcal{C}(c) &= \quad 010\\
\mathcal{C}(d) &= \quad 011\\
\mathcal{C}(e) &= \quad 100\\
\mathcal{C}(f) &= \quad 101\\
\mathcal{C}(g) &= \quad 110.
\end{aligned}$$

The source output, $x_1, x_2, \dots,$ would then be encoded into the encoded output $\mathcal{C}(x_1)\mathcal{C}(x_2)\dots$ and thus the encoded output contains L bits per source symbol. For the above example the source sequence $bad\dots$ would be encoded into $001000011\dots$. Note that the output bits are simply run together (or, more technically, concatenated).

There are 2^L different combinations of values for a block of L bits. Thus, if the number of symbols in the source alphabet, $M = |\mathcal{X}|$, satisfies $M \le 2^L$, then a different binary L-tuple may be assigned to each symbol. Assuming that the decoder knows where the beginning of the encoded sequence is, the decoder can segment the sequence into L bit blocks and then decode each block into the corresponding source symbol.

In summary, if the source alphabet has size M, then this coding method requires $L = \lceil \log_2 M \rceil$ bits to encode each source symbol, where $\lceil w \rceil$ denotes the smallest integer greater than or equal to the real number w. Thus $\log_2 M \le L < \log_2 M + 1$. The lower bound, $\log_2 M$, can be achieved with equality if and only if M is a power of 2.

A technique to be used repeatedly is that of first segmenting the sequence of source symbols into successive blocks of n source symbols at a time. Given an alphabet $\mathcal{X}$ of M symbols, there are M^n possible n-tuples. These M^n n-tuples are regarded as the elements of a super-alphabet. Each n-tuples can be encoded rather than encoding the original symbols. Using fixed-length source coding on these n-tuples, each source n-tuple can be encoded into $L = \lceil \log_2 M^n \rceil$ bits. The rate $\overline{L} = L/n$ of encoded bits per original source symbol is then bounded by $\log_2 M \le \overline{L} < \log_2 M + \frac{1}{n}$, because:

$$\begin{aligned}
\overline{L} = \frac{\lceil \log_2 M^n \rceil}{n} &\geq \frac{n \log_2 M}{n} = \log_2 M; \\
\overline{L} = \frac{\lceil \log_2 M^n \rceil}{n} &< \frac{n(\log_2 M) + 1}{n} = \log_2 M + \frac{1}{n}.
\end{aligned}$$

The conclusion is that by letting n become sufficiently large, the average number of coded bits per source symbol can be made arbitrarily close to $\log_2 M$, regardless of whether M is a power of 2.

Some remarks:

- This simple scheme to make $\overline{L}$ arbitrarily close to $\log_2 M$ is of greater theoretical interest than practical interest. As shown later, $\log_2 M$ is the minimum possible binary rate for uniquely-decodable source coding if the source symbols are independent and equiprobable. Thus this scheme asymptotically approaches this minimum.
- This result begins to hint at why measures of information are logarithmic in the alphabet size.[3] The logarithm is usually taken to the base 2 in discussions of binary codes. Henceforth $\log n$ means "$\log_2 n$."
- This method is nonprobabilistic; it takes no account of whether some symbols occur more frequently than others, and it works robustly regardless of the symbol frequencies. But if it is known that some symbols occur more frequently than others, then the rate $\overline{L}$ of coded bits per source symbol can be reduced by assigning shorter bit sequences to more common symbols in a *variable-length source code*. This will be our next topic.

2.3 Variable-length codes for discrete sources

The motivation for using variable-length encoding on discrete sources is the intuition that data compression can be achieved by mapping more probable symbols into shorter bit sequences, and less likely symbols into longer bit sequences. This intuition was used in the Morse code of old-time telegraphy in which letters were mapped into strings of dots and dashes, using shorter strings for common letters and longer strings for less common letters.

A *variable-length code* $\mathcal{C}$ maps each source symbol a_j in a source alphabet $\mathcal{X} = \{a_1, \ldots, a_M\}$ to a binary string $\mathcal{C}(a_j)$, called a *codeword*. The number of bits in $\mathcal{C}(a_j)$ is called the *length* $l(a_j)$ of $\mathcal{C}(a_j)$. For example, a variable-length code for the alphabet $\mathcal{X} = \{a, b, c\}$ and its lengths might be given by

$$\begin{array}{rlcl}
\mathcal{C}(a) = & 0 & & l(a) = 1 \\
\mathcal{C}(b) = & 10 & & l(b) = 2 \\
\mathcal{C}(c) = & 11 & & l(c) = 2
\end{array}$$

Successive codewords of a variable-length code are assumed to be transmitted as a continuing sequence of bits, with no demarcations of codeword boundaries (*i.e.*, no commas or spaces). The

[3]The notion that information can be viewed as a logarithm of a number of possibilities was first suggested by Hartley [8] in 1927.

source decoder, given an original starting point, must determine where the codeword boundaries are; this is called *parsing*.

A potential system issue with variable-length coding is the requirement for buffering. If source symbols arrive at a fixed rate and the encoded bit sequence must be transmitted at a fixed bit rate, then a buffer must be provided between input and output. This requires some sort of recognizable 'fill' to be transmitted when the buffer is empty and the possibility of lost data when the buffer is full. There are many similar system issues, including occasional errors on the channel, initial synchronization, terminal synchronization, etc. Many of these issues are discussed later, but they are more easily understood after the more fundamental issues are discussed.

2.3.1 Unique decodability

The major property that is usually required from any variable-length code is that of *unique decodability*. This essentially means that for any sequence of source symbols, that sequence can be reconstructed unambiguously from the encoded bit sequence. Here initial synchronization is assumed: the source decoder knows which is the first bit in the coded bit sequence. Note that without initial synchronization, not even a fixed-length code can be uniquely decoded.

Clearly, unique decodability requires that $\mathcal{C}(a_j) \neq \mathcal{C}(a_k)$ for each $k \neq j$. More than that, however, it requires that strings[4] of encoded symbols be distinguishable. The following definition says this precisely:

Definition 2.3.1. A code $\mathcal{C}$ for a discrete source is uniquely decodable if, for any string of source symbols, say $x_1, x_2, \dots, x_n$, the concatenation[5] of the corresponding codewords, $\mathcal{C}(x_1)\mathcal{C}(x_2)\cdots\mathcal{C}(x_n)$, differs from the concatenation of the codewords $\mathcal{C}(x'_1)\mathcal{C}(x'_2)\cdots\mathcal{C}(x'_m)$ for any other string $x'_1, x'_2, \dots, x'_m$ of source symbols.

In other words, $\mathcal{C}$ is uniquely decodable if all concatenations of codewords are distinct.

Remember that there are no commas or spaces between codewords; the source decoder has to determine the codeword boundaries from the received sequence of bits. (If commas were inserted, the code would be ternary rather than binary.)

For example, the above code $\mathcal{C}$ for the alphabet $\mathcal{X} = \{a, b, c\}$ is soon shown to be uniquely decodable. However, the code $\mathcal{C}'$ defined by

$$\begin{aligned} \mathcal{C}'(a) &= \ 0 \\ \mathcal{C}'(b) &= \ 1 \\ \mathcal{C}'(c) &= \ 01 \end{aligned}$$

is not uniquely decodable, even though the codewords are all different. If the source decoder observes 01, it cannot determine whether the source emitted $(a\,b)$ or (c).

Note that the property of unique decodability depends only on the set of codewords and not on the mapping from symbols to codewords. Thus we can refer interchangeably to uniquely-decodable codes and uniquely-decodable codeword sets.

[4]A *string* of symbols is an n-tuple of symbols for any finite n. A *sequence* of symbols is an n-tuple in the limit $n \to \infty$, although the word sequence is also used when the length might be either finite or infinite.

[5]The concatenation of two strings, say $u_1 \cdots u_l$ and $v_1 \cdots v_{l'}$ is the combined string $u_1 \cdots u_l v_1 \cdots v_{l'}$.

2.3.2 Prefix-free codes for discrete sources

Decoding the output from a uniquely-decodable code, and even determining whether it is uniquely decodable, can be quite complicated. However, there is a simple class of uniquely-decodable codes called *prefix-free codes*. As shown later, these have the following advantages over other uniquely-decodable codes:[6]

- If a uniquely-decodable code exists with a certain set of codeword lengths, then a prefix-free code can easily be constructed with the same set of lengths.
- The decoder can decode each codeword of a prefix-free code immediately on the arrival of the last bit in that codeword.
- Given a probability distribution on the source symbols, it is easy to construct a prefix-free code of minimum expected length.

Definition 2.3.2. A *prefix* of a string $y_1 \cdots y_l$ is any initial substring $y_1 \cdots y_{l'}$, $l' \leq l$ of that string. The prefix is *proper* if $l' < l$. A code is *prefix-free* if no codeword is a prefix of any other codeword.

For example, the code $\mathcal{C}$ with codewords $0, 10$, and 11 is prefix-free, but the code $\mathcal{C}'$ with codewords 0, 1, and 01 is not. Every fixed-length code with distinct codewords is prefix-free.

We will now show that every prefix-free code is uniquely decodable. The proof is constructive, and shows how the decoder can uniquely determine the codeword boundaries.

Given a prefix-free code $\mathcal{C}$, a corresponding binary code tree can be constructed which grows from a root on the left to leaves on the right representing codewords. Each branch is labelled 0 or 1 and each node represents the binary string corresponding to the branch labels from the root to that node. The tree is extended just enough to include each codeword. That is, each node in the tree is either a codeword or proper prefix of a codeword (see Figure 2.2).

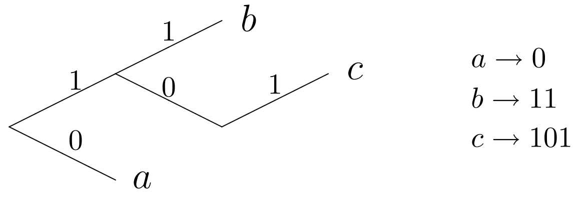
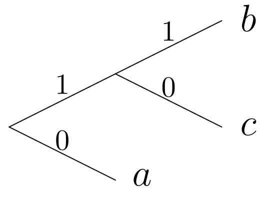


Figure 2.2: The binary code tree for a prefix-free code.

The prefix-free condition ensures that each codeword corresponds to a leaf node (*i.e.*, a node with no adjoining branches going to the right). Each intermediate node (*i.e.*, nodes having one or more adjoining branches going to the right) is a prefix of some codeword reached by traveling right from the intermediate node.

[6]With all the advantages of prefix-free codes, it is difficult to understand why the more general class is even discussed. This will become clearer much later.

The tree of Figure 2.2 has an intermediate node, 10, with only one right-going branch. This shows that the codeword for c could be shortened to 10 without destroying the prefix-free property. This is shown in Figure 2.3.

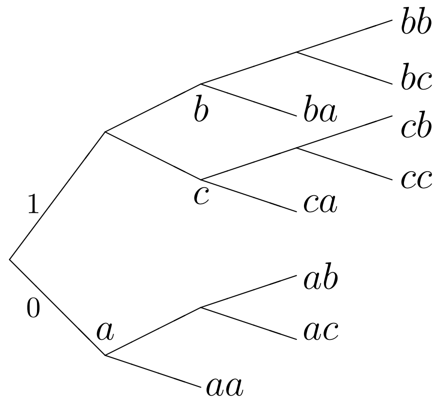


$a \to 0$
$b \to 11$
$c \to 10$

Figure 2.3: A more efficient code than that of Figure 2.2.

A prefix-free code will be called *full* if no new codeword can be added without destroying the prefix-free property. As just seen, a prefix-free code is also full if no codeword can be shortened without destroying the prefix-free property. Thus the code of Figure 2.2 is not full, but that of Figure 2.3 is.

To see why the prefix-free condition guarantees unique decodability, consider the tree for the concatenation of two codewords. This is illustrated in Figure 2.4 for the code of Figure 2.3. This new tree has been formed simply by grafting a copy of the original tree onto each of the leaves of the original tree. Each concatenation of two codewords thus lies on a different node of the tree and also differs from each single codeword. One can imagine grafting further trees onto the leaves of Figure 2.4 to obtain a tree representing still more codewords concatenated together. Again all concatenations of code words lie on distinct nodes, and thus correspond to distinct binary strings.



$aa \to 00$
$ab \to 011$
$ac \to 010$
$ba \to 110$
$bb \to 1111$
$bc \to 1110$
$ca \to 100$
$cb \to 1011$
$cc \to 1010$

Figure 2.4: Binary code tree for two codewords; upward branches represent 1's.

An alternative way to see that prefix-free codes are uniquely decodable is to look at the codeword parsing problem from the viewpoint of the source decoder. Given the encoded binary string for any strong of source symbols, the source decoder can decode the first symbol simply by reading the string from left to right and following the corresponding path in the code tree until it reaches a leaf, which must correspond to the first codeword by the prefix-free property. After stripping off the first codeword, the remaining binary string is again a string of codewords, so the source decoder can find the second codeword in the same way, and so on *ad infinitum*.

For example, suppose a source decoder for the code of Figure 2.3 decodes the sequence $1010011\cdots$. Proceeding through the tree from the left, it finds that 1 is not a codeword, but that 10 is the codeword for c. Thus c is decoded as the first symbol of the source output, leaving the string $10011\cdots$. Then c is decoded as the next symbol, leaving $011\cdots$, which is decoded into a and then b, and so forth.

This proof also shows that prefix-free codes can be decoded with no delay. As soon as the final bit of a codeword is received at the decoder, the codeword can be recognized and decoded without waiting for additional bits. For this reason, prefix-free codes are sometimes called instantaneous codes.

It has been shown that all prefix-free codes are uniquely decodable. The converse is not true, as shown by the following code:

$$\begin{aligned} \mathcal{C}(a) &= \quad 0 \\ \mathcal{C}(b) &= \quad 01 \\ \mathcal{C}(c) &= \quad 011 \end{aligned}$$

An encoded sequence for this code can be uniquely parsed by recognizing 0 as the beginning of each new code word. A different type of example is given in Exercise 2.6.

With variable-length codes, if there are errors in data transmission, then the source decoder may lose codeword boundary synchronization and may make more than one symbol error. It is therefore important to study the synchronization properties of variable-length codes. For example, the prefix-free code $\{0, 10, 110, 1110, 11110\}$ is instantaneously self-synchronizing, because every 0 occurs at the end of a codeword. The shorter prefix-free code $\{0, 10, 110, 1110, 1111\}$ is probabilistically self-synchronizing; again, any observed 0 occurs at the end of a codeword, but since there may be a sequence of 1111 codewords of unlimited length, the length of time before resynchronization is a random variable. These questions are not pursued further here.

2.3.3 The Kraft inequality for prefix-free codes

The Kraft inequality [6] is a condition determining whether it is possible to construct a prefix-free code for a given discrete source alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ with a given set of codeword lengths $\{l(a_j); 1 \le j \le M\}$.

Theorem 2.3.1 (Kraft inequality for prefix-free codes). *Every prefix-free code for an alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ with codeword lengths $\{l(a_j); 1 \le j \le M\}$ satisfies*

$$\sum_{j=1}^{M} 2^{-l(a_j)} \le 1. \tag{2.1}$$

Conversely, if (2.1) is satisfied, then a prefix-free code with lengths $\{l(a_j); 1 \le j \le M\}$ exists.

Moreover, every full prefix-free code satisfies (2.1) with equality and every non-full prefix-free code satisfies it with strict inequality.

For example, this theorem implies that there exists a full prefix-free code with codeword lengths $\{1, 2, 2\}$ (two such examples have already been given), but there exists no prefix-free code with codeword lengths $\{1, 1, 2\}$.

Before proving the theorem, we show how to represent codewords as base 2 expansions (the base 2 analog of base 10 decimals) in the binary number system. After understanding this representation, the theorem will be almost obvious. The base 2 expansion $.y_1y_2\cdots y_l$ represents the rational number $\sum_{j=1}^{l} y_j 2^{-j}$. For example, .011 represents $1/4 + 1/8$.

Ordinary decimals with l digits are frequently used to indicate an approximation of a real number to l places of accuracy. Here, in the same way, the base 2 expansion $.y_1y_2\cdots y_l$ is viewed as 'covering' the interval[7] $[\sum_{i=1}^{l} y_i 2^{-i},\ \sum_{i=1}^{l} y_i 2^{-i} + 2^{-l})$. This interval has size 2^{-l} and includes all numbers whose base 2 expansions start with $.y_1 \dots y_l$.

In this way, any codeword $\mathcal{C}(a_j)$ of length l is represented by a rational number in the interval [0, 1) and covers an interval of size 2^{-l} which includes all strings that contain $\mathcal{C}(a_j)$ as a prefix (see Figure 2.3). The proof of the theorem follows:

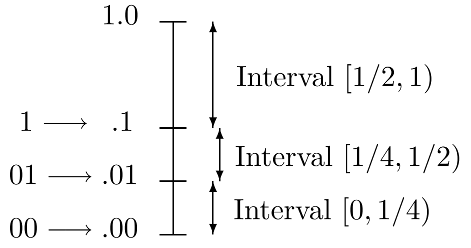


Figure 2.5: Base 2 expansion numbers and intervals representing codewords. The codewords represented above are (00, 01, and 1).

Proof: First, assume that $\mathcal{C}$ is a prefix-free code with codeword lengths $\{l(a_j),\ 1 \le j \le M\}$. For any distinct a_j and a_k in $\mathcal{X}$, it was shown above that the base 2 expansion corresponding to $\mathcal{C}(a_j)$ cannot lie in the interval corresponding to $\mathcal{C}(a_k)$ since $\mathcal{C}(a_k)$ is not a prefix of $\mathcal{C}(a_j)$. Thus the lower end of the interval corresponding to any codeword $\mathcal{C}(a_j)$ cannot lie in the interval corresponding to any other codeword. Now, if two of these intervals intersect, then the lower end of one of them must lie in the other, which is impossible. Thus the two intervals must be disjoint and thus the set of all intervals associated with the codewords are disjoint. Since all these intervals are contained in the interval $[0, 1)$ and the size of the interval corresponding to $\mathcal{C}(a_j)$ is $2^{-l(a_j)}$, (2.1) is established.

Next note that if (2.1) is satisfied with strict inequality, then some interval exists in $[0, 1)$ that does not intersect any codeword interval; thus another codeword can be 'placed' in this interval and the code is not full. If (2.1) is satisfied with equality, then the intervals fill up $[0, 1)$. In this case no additional code word can be added and the code is full.

Finally we show that a prefix-free code can be constructed from any desired set of codeword lengths $\{l(a_j),\ 1 \le j \le M\}$ for which (2.1) is satisfied. Put the set of lengths in nondecreasing order, $l_1 \le l_2 \le \cdots \le l_M$ and let $u_1, \dots, u_M$ be the real numbers corresponding to the codewords in the construction to be described. The construction is quite simple: $u_1 = 0$, and for all

[7]Brackets and parentheses, respectively, are used to indicate closed and open boundaries; thus the interval $[a, b)$ means the set of real numbers u such that $a \le u < b$.

$j, 1 < j \leq M,$

$$u_j = \sum_{i=1}^{j-1} 2^{-l_i}. \tag{2.2}$$

Each term on the right is an integer multiple of 2^{-l_j}, so u_j is also an integer multiple of 2^{-l_j}. From (2.1), $u_j < 1$, so u_j can be represented by a base 2 expansion with l_j places. The corresponding codeword of length l_j can be added to the code while preserving prefix-freedom (see Figure 2.6).

□

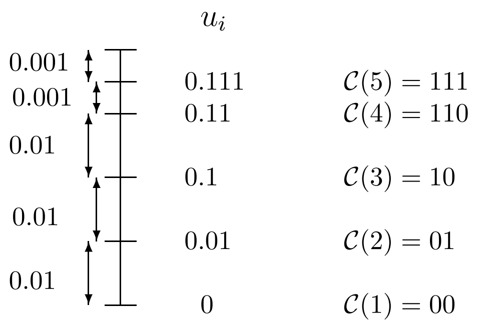


Figure 2.6: Construction of codewords for the set of lengths $\{2, 2, 2, 3, 3\}$. $\mathcal{C}(i)$ is formed from u_i by representing u_i to l_i places.

Some final remarks on the Kraft inequality:

- Just because a code has lengths that satisfy (2.1), it does not follow that the code is prefix-free, or even uniquely decodable.
- Exercise 2.11 shows that Theorem 2.3.1 also holds for all uniquely-decodable codes— *i.e.*, there exists a uniquely-decodable code with codeword lengths $\{l(a_j), 1 \leq j \leq M\}$ if and only if (2.1) holds. This will imply that if a uniquely-decodable code exists with a certain set of codeword lengths, then a prefix-free code exists with the same set of lengths. So why use any code other than a prefix-free code?

2.4 Probability models for discrete sources

It was shown above that prefix-free codes exist for any set of codeword lengths satisfying the Kraft inequality. When does it desirable to use one of these codes?– *i.e.*, when is the expected number of coded bits per source symbol less than $\log M$ and why is the expected number of coded bits per source symbol the primary parameter of importance?

This question cannot be answered without a probabilistic model for the source. For example, the $M = 4$ prefix-free set of codewords $\{0, 10, 110, 111\}$ has an expected length of $2.25 > 2 = \log M$ if the source symbols are equiprobable, but if the source symbol probabilities are $\{1/2, 1/4, 1/8, 1/8\}$, then the expected length is $1.75 < 2$.

The discrete sources that one meets in applications usually have very complex statistics. For example, consider trying to compress email messages. In typical English text, some letters such as e and o occur far more frequently than q, x, and z. Moreover, the letters are not independent; for example h is often preceded by t, and q is almost always followed by u. Next, some strings of letters are words, while others are not; those that are not have probability near 0 (if in fact the text is correct English). Over longer intervals, English has grammatical and semantic constraints, and over still longer intervals, such as over multiple email messages, there are still further constraints.

It should be clear therefore that trying to find an accurate probabilistic model of a real-world discrete source is not going to be a productive use of our time. An alternative approach, which has turned out to be very productive, is to start out by trying to understand the encoding of "toy" sources with very simple probabilistic models. After studying such toy sources, it will be shown how to generalize to source models with more and more general structure, until, presto, real sources can be largely understood even without good stochastic models. This is a good example of a problem where having the patience to look carefully at simple and perhaps unrealistic models pays off handsomely in the end.

The type of toy source that will now be analyzed in some detail is called a discrete memoryless source.

2.4.1 Discrete memoryless sources

A *discrete memoryless source* (DMS) is defined by the following properties:

- The source output is an unending sequence, $X_1, X_2, X_3, \ldots$, of randomly selected symbols from a *finite* set $\mathcal{X} = \{a_1, a_2, \ldots, a_M\}$, called the *source alphabet*.
- Each source output $X_1, X_2, \ldots$ is selected from $\mathcal{X}$ using the same probability mass function (pmf) $\{p_X(a_1), \ldots, p_X(a_M)\}$. Assume that $p_X(a_j) > 0$ for all j, $1 \le j \le M$, since there is no reason to assign a code word to a symbol of zero probability and no reason to model a discrete source as containing impossible symbols.
- Each source output X_m is statistically independent of the previous outputs $X_1, \ldots, X_{m-1}$.

The randomly chosen symbols coming out of the source are called *random symbols*. They are very much like random variables except that they may take on nonnumeric values. Thus, if X denotes the result of a fair coin toss, then it can be modeled as a random symbol that takes values in the set {HEADS, TAILS} with equal probability. Note that if X is a nonnumeric random symbol, then it makes no sense to talk about its expected value. However, the notion of statistical independence between random symbols is the same as that for random variables, *i.e.*, the event that X_i is any given element of $\mathcal{X}$ is independent of the events corresponding to the values of the other random symbols.

The word memoryless in the definition refers to the statistical independence between different random symbols, *i.e.*, each variable is chosen with no memory of how the previous random symbols were chosen. In other words, the source symbol sequence is independent and identically distributed (iid).[8]

In summary, a DMS is a semi-infinite iid sequence of random symbols

$$X_1, X_2, X_3, \ldots$$

[8]Do not confuse this notion of memorylessness with any non-probabalistic notion in system theory.

each drawn from the finite set $\mathcal{X}$, each element of which has positive probability.

A sequence of independent tosses of a biased coin is one example of a DMS. The sequence of symbols drawn (with replacement) in a Scrabble™ game is another. The reason for studying these sources is that they provide the tools for studying more realistic sources.

2.5 Minimum $\overline{L}$ for prefix-free codes

The Kraft inequality determines which sets of codeword lengths are possible for prefix-free codes. Given a discrete memoryless source (DMS), we want to determine what set of codeword lengths can be used to *minimize* the expected length of a prefix-free code for that DMS. That is, we want to minimize the expected length subject to the Kraft inequality.

Suppose a set of lengths $l(a_1), \dots, l(a_M)$ (subject to the Kraft inequality) is chosen for encoding each symbol into a prefix-free codeword. Define $L(X)$ (or more briefly L) as a random variable representing the codeword length for the randomly selected source symbol. The expected value of L for the given code is then given by

$$\overline{L} = \mathsf{E}[L] = \sum_{j=1}^{M} l(a_j) p_X(a_j).$$

We want to find $\overline{L}_{\min}$, which is defined as the minimum value of $\overline{L}$ over all sets of codeword lengths satisfying the Kraft inequality.

Before finding $\overline{L}_{\min}$, we explain why this quantity is of interest. The number of bits resulting from using the above code to encode a long block $\boldsymbol{X} = (X_1, X_2, \dots, X_n)$ of symbols is $S_n = L(X_1) + L(X_2) + \cdots + L(X_n)$. This is a sum of n iid random variables (rv's), and the law of large numbers, which is discussed in Section 2.7.1, implies that S_n/n, the number of bits per symbol in this long block, is very close to $\overline{L}$ with probability very close to 1. In other words, $\overline{L}$ is essentially the rate (in bits per source symbol) at which bits come out of the source encoder. This motivates the objective of finding $\overline{L}_{\min}$ and later of finding codes that achieve the minimum.

Before proceeding further, we simplify our notation. We have been carrying along a completely arbitrary finite alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$ of size $M = |\mathcal{X}|$, but this problem (along with most source coding problems) involves only the probabilities of the M symbols and not their names. Thus define the source alphabet to be $\{1, 2, \dots, M\}$, denote the symbol probabilities by $p_1, \dots, p_M$, and denote the corresponding codeword lengths by $l_1, \dots, l_M$. The expected length of a code is then

$$\overline{L} = \sum_{j=1}^{M} l_j p_j$$

Mathematically, the problem of finding $\overline{L}_{\min}$ is that of minimizing $\overline{L}$ over all sets of integer lengths $l_1, \dots, l_M$ subject to the Kraft inequality:

$$\overline{L}_{\min} = \min_{l_1, \dots, l_M : \sum_j 2^{-l_j} \le 1} \left\{ \sum_{j=1}^{M} p_j l_j \right\}. \tag{2.3}$$

2.5.1 Lagrange multiplier solution for the minimum $\overline{L}$

The minimization in (2.3) is over a function of M variables, $l_1, \dots, l_M$, subject to constraints on those variables. Initially, consider a simpler problem where there are no integer constraint on the l_j. This simpler problem is then to minimize $\sum_j p_j l_j$ over all real values of $l_1, \dots, l_M$ subject to $\sum_j 2^{-l_j} \le 1$. The resulting minimum is called $\overline{L}_{\min}$(noninteger).

Since the allowed values for the lengths in this minimization include integer lengths, it is clear that $\overline{L}_{\min}(\text{noninteger}) \le \overline{L}_{\min}$. This noninteger minimization will provide a number of important insights about the problem, so its usefulness extends beyond just providing a lower bound on $\overline{L}_{\min}$.

Note first that the minimum of $\sum_j l_j p_j$ subject to $\sum_j 2^{-l_j} \le 1$ must occur when the constraint is satisfied with equality, for otherwise, one of the l_j could be reduced, thus reducing $\sum_j p_j l_j$ without violating the constraint. Thus the problem is to minimize $\sum_j p_j l_j$ subject to $\sum_j 2^{-l_j} = 1$.

Problems of this type are often solved by using a Lagrange multiplier. The idea is to replace the minimization of one function, subject to a constraint on another function, by the minimization of a linear combination of the two functions, in this case the minimization of

$$\sum_j p_j l_j + \lambda \sum_j 2^{-l_j}. \tag{2.4}$$

If the method works, the expression can be minimized for each choice of λ (called a *Lagrange multiplier*); λ can then be chosen so that the optimizing choice of $l_1, \dots, l_M$ satisfies the constraint. The minimizing value of (2.4) is then $\sum_j p_j l_j + \lambda$. This choice of $l_1, \dots, l_M$ minimizes the original constrained optimization, since for any $l'_1, \dots, l'_M$ that satisfies the constraint $\sum_j 2^{-l'_j} = 1$, the expression in (2.4) is $\sum_j p_j l'_j + \lambda$, which must be greater than or equal to $\sum_j p_j l_j + \lambda$.

We can attempt[9] to minimize (2.4) simply by setting the derivitive with respect to each l_j equal to 0. This yields

$$p_j - \lambda(\ln 2)2^{-l_j} = 0; \quad 1 \le j \le M. \tag{2.5}$$

Thus $2^{-l_j} = p_j/(\lambda \ln 2)$. Since $\sum_j p_j = 1$, λ must be equal to $1/\ln 2$ in order to satisfy the constraint $\sum_j 2^{-l_j} = 1$. Then $2^{-l_j} = p_j$, or equivalently $l_j = -\log p_j$. It will be shown shortly that this stationary point actually achieves a minimum. Substituting this solution into (2.3),

$$\overline{L}_{\min}(\text{noninteger}) = -\sum_{j=1}^{M} p_j \log p_j. \tag{2.6}$$

The quantity on the right side of (2.6) is called the *entropy*[10] of X, and denoted as $\mathsf{H}[X]$. Thus

$$\mathsf{H}[X] = -\sum_j p_j \log p_j.$$

[9]There are well-known rules for when the Lagrange multiplier method works and when it can be solved simply by finding a stationary point. The present problem is so simple, however, that this machinery is unnecessary.

[10]Note that X is a random symbol and carries with it all of the accompanying baggage, including a pmf. The entropy $\mathsf{H}[X]$ is a numerical function of the random symbol including that pmf; in the same way $\mathsf{E}[L]$ is a numerical function of the rv L. Both $\mathsf{H}[X]$ and $\mathsf{E}[L]$ are expected values of particular rv's. In distinction, $L(X)$ above is an rv in its own right; it is based on some function $l(x)$ mapping $\mathcal{X} \to \mathbb{R}$ and takes the sample value $l(x)$ for all sample points such that $X = x$.

In summary, the entropy $\mathsf{H}[X]$ is a lower bound to $\overline{L}$ for prefix-free codes and this lower bound is achieved when $l_j = -\log p_j$ for each j. The bound was derived by ignoring the integer constraint, and can be met only if $-\log p_j$ is an integer for each j; *i.e.*, if each p_j is a power of 2.

2.5.2 Entropy bounds on $\overline{L}$

We now return to the problem of minimizing $\overline{L}$ with an integer constraint on lengths. The following theorem both establishes the correctness of the previous non-integer optimization and provides an upper bound on $\overline{L}_{\min}$.

Theorem 2.5.1 (Entropy bounds for prefix-free codes). *Let X be a discrete random symbol with symbol probabilities $p_1, \ldots , p_M$. Let $\overline{L}_{\min}$ be the minimum expected codeword length over all prefix-free codes for X. Then*

$$H[X] \leq \overline{L}_{\min} < H[X] + 1 \quad \text{bit/symbol.} \tag{2.7}$$

Furthermore, $\overline{L}_{\min} = H[X]$ if and only if each probability p_j is an integer power of 2.

Proof: It is first shown that $\mathsf{H}[X] \leq \overline{L}$ for all prefix-free codes. Let $l_1, \ldots , l_M$ be the codeword lengths of an arbitrary prefix-free code. Then

$$\mathsf{H}[X] - \overline{L} \;=\; \sum_{j=1}^{M} p_j \log \frac{1}{p_j} - \sum_{j=1}^{M} p_j l_j \;=\; \sum_{j=1}^{M} p_j \log \frac{2^{-l_j}}{p_j}, \tag{2.8}$$

where $\log 2^{-l_j}$ has been substituted for $-l_j$.

We now use the very useful inequality $\ln u \leq u - 1$, or equivalently $\log u \leq (\log e)(u - 1)$, which is illustrated in Figure 2.7. Note that equality holds only at the point $u = 1$.

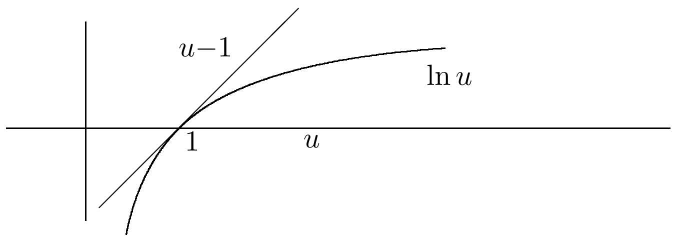


Figure 2.7: The inequality $\ln u \leq u - 1$. The inequality is strict except at $u = 1$.

Substituting this inequality in (2.8),

$$\mathsf{H}[X] - \overline{L} \;\leq\; (\log e) \sum_{j=1}^{M} p_j \left(\frac{2^{-l_j}}{p_j} - 1 \right) \;=\; (\log e) \left(\sum_{j=1}^{M} 2^{-l_j} - \sum_{j=1}^{M} p_j \right) \leq 0, \tag{2.9}$$

where the Kraft inequality and $\sum_j p_j = 1$ has been used. This establishes the left side of (2.7). The inequality in (2.9) is strict unless $2^{-l_j}/p_j = 1$, or equivalently $l_j = -\log p_j$, for all j. For integer l_j, this can be satisfied with equality if and only if p_j is an integer power of 2 for all j. For arbitrary real values of l_j, this proves that (2.5) minimizes (2.3) without the integer constraint, thus verifying (2.6.)

To complete the proof, it will be shown that a prefix-free code exists with $\overline{L} < \mathsf{H}[X]+1$. Choose the codeword lengths to be

$$l_j = \lceil -\log p_j \rceil ,$$

where the ceiling notation $\lceil u \rceil$ denotes the smallest integer less than or equal to u. With this choice,

$$-\log p_j \le l_j < -\log p_j + 1. \tag{2.10}$$

Since the left side of (2.10) is equivalent to $2^{-l_j} \le p_j$, the Kraft inequality is satisfied:

$$\sum_j 2^{-l_j} \le \sum_j p_j = 1.$$

Thus a prefix-free code exists with the above lengths. From the right side of (2.10), the expected codeword length of this code is upperbounded by

$$\overline{L} = \sum_j p_j l_j < \sum_j p_j \left(-\log p_j + 1\right) = \mathsf{H}[X] + 1.$$

Since $\overline{L}_{\min} \le \overline{L}$, $\overline{L}_{\min} < \mathsf{H}[X] + 1$, completing the proof.
□

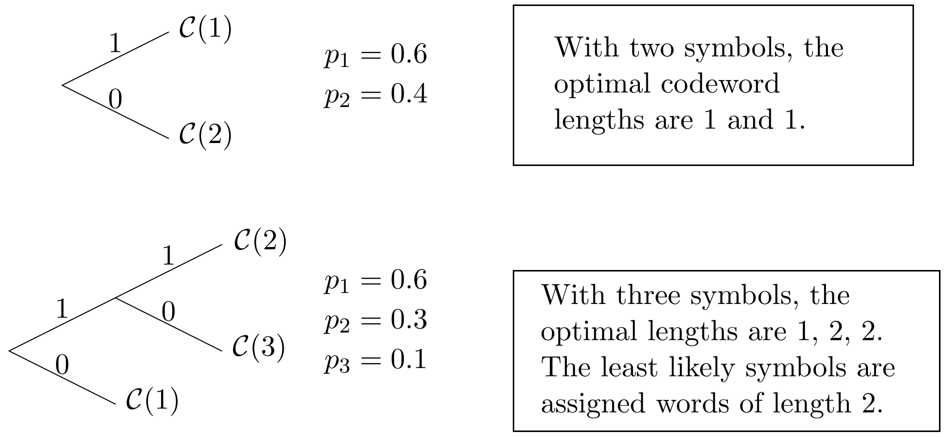
Both the proof above and the noninteger minimization in (2.6) suggest that the optimal length of a codeword for a source symbol of probability p_j should be approximately $-\log p_j$. This is not quite true, because, for example, if $M = 2$ and $p_1 = 2^{-20}$, $p_2 = 1{-}2^{-20}$, then $-\log p_1 = 20$, but the optimal l_1 is 1. However, the last part of the above proof shows that if each l_i is chosen as an integer approximation to $-\log p_i$, then $\overline{L}$ is at worst within one bit of $\mathsf{H}[X]$.

For sources with a small number of symbols, the upper bound in the theorem appears to be too loose to have any value. When these same arguments are applied later to long blocks of source symbols, however, the theorem leads directly to the source coding theorem.

2.5.3 Huffman's algorithm for optimal source codes

In the very early days of information theory, a number of heuristic algorithms were suggested for choosing codeword lengths l_j to approximate $-\log p_j$. Both Claude Shannon and Robert Fano had suggested such heuristic algorithms by 1948. It was conjectured at that time that, since this was an integer optimization problem, its optimal solution would be quite difficult. It was quite a surprise therefore when David Huffman [9] came up with a very simple and straightforward algorithm for constructing optimal (in the sense of minimal $\overline{L}$) prefix-free codes. Huffman developed the algorithm in 1950 as a term paper in Robert Fano's information theory class at MIT.

Huffman's trick, in today's jargon, was to "think outside the box." He ignored the Kraft inequality, and looked at the binary code tree to establish properties that an optimal prefix-free code should have. After discovering a few simple properties, he realized that they led to a simple recursive procedure for constructing an optimal code.



1
0
$\mathcal{C}(1)$
$\mathcal{C}(2)$
$p_1 = 0.6$
$p_2 = 0.4$
With two symbols, the optimal codeword lengths are 1 and 1.
1
1
0
0
$\mathcal{C}(2)$
$\mathcal{C}(3)$
$\mathcal{C}(1)$
$p_1 = 0.6$
$p_2 = 0.3$
$p_3 = 0.1$
With three symbols, the optimal lengths are 1, 2, 2. The least likely symbols are assigned words of length 2.

Figure 2.8: Some simple optimal codes.

The simple examples in Figure 2.8 illustrate some key properties of optimal codes. After stating these properties precisely, the Huffman algorithm will be almost obvious.

The property of the length assignments in the three-word example above can be generalized as follows: the longer the codeword, the less probable the corresponding symbol must be. More precisely:

Lemma 2.5.1. *Optimal codes have the property that if $p_i > p_j$, then $l_i \leq l_j$.*

Proof: Assume to the contrary that a code has $p_i > p_j$ and $l_i > l_j$. The terms involving symbols i and j in $\overline{L}$ are $p_i l_i + p_j l_j$. If the two code words are interchanged, thus interchanging l_i and l_j, this sum decreases, *i.e.*,

$$(p_i l_i + p_j l_j) - (p_i l_j + p_j l_i) = (p_i - p_j)(l_i - l_j) > 0.$$

Thus $\overline{L}$ decreases, so any code with $p_i > p_j$ and $l_i > l_j$ is nonoptimal.
□

An even simpler property of an optimal code is as follows:

Lemma 2.5.2. *Optimal prefix-free codes have the property that the associated code tree is full.*

Proof: If the tree is not full, then a codeword length could be reduced (see Figures 2.2 and 2.3).

□

Define the *sibling* of a codeword as the binary string that differs from the codeword in only the final digit. A sibling in a full code tree can be either a codeword or an intermediate node of the tree.

Lemma 2.5.3. *Optimal prefix-free codes have the property that, for each of the longest codewords in the code, the sibling of that codeword is another longest codeword.*

Proof: A sibling of a codeword of maximal length cannot be a prefix of a longer codeword. Since it cannot be an intermediate node of the tree, it must be a codeword.

□

For notational convenience, assume that the $M = |\mathcal{X}|$ symbols in the alphabet are ordered so that $p_1 \geq p_2 \geq \cdots \geq p_M$.

Lemma 2.5.4. *Let X be a random symbol with a pmf satisfying $p_1 \geq p_2 \geq \cdots \geq p_M$. There is an optimal prefix-free code for X in which the codewords for $M-1$ and M are siblings and have maximal length within the code.*

Proof: There are finitely many codes satisfying the Kraft inequality with equality,[11] so consider a particular one that is optimal. If $p_M < p_j$ for each $j < M$, then, from Lemma 2.5.1, $l_M \geq l_j$ for each and l_M has maximal length. If $p_M = p_j$ for one or more $j < M$, then l_j must be maximal for at least one such j. Then if l_M is not maximal, $\mathcal{C}(j)$ and $\mathcal{C}(M)$ can be interchanged with no loss of optimality, after which l_M is maximal. Now if $\mathcal{C}(k)$ is the sibling of $\mathcal{C}(M)$ in this optimal code, then l_k also has maximal length. By the argument above, $\mathcal{C}(M-1)$ can then be exchanged with $\mathcal{C}(k)$ with no loss of optimality.

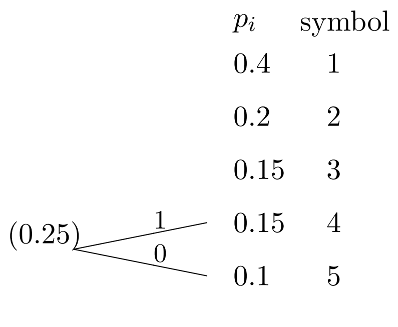
□

The Huffman algorithm chooses an optimal code tree by starting at the leaves for the least likely symbols and working in. In particular, the codewords for the two least likely symbols are chosen to be siblings (it makes no difference which sibling ends in 1 and which in 0). How is the rest of the tree to be chosen?

If the above pair of siblings is removed, the rest of the tree will have $M-1$ leaves, namely the $M-2$ leaves for the original first $M-2$ symbols, and the parent node of the removed siblings. The probability p'_{M-1} associated with this new leaf is taken as $p_{M-1} + p_M$. This tree of $M-1$ leaves is viewed as a code for a reduced random symbol X' with a reduced set of probabilities given as $p_1, \dots, p_{M-2}$ for the original first $M-2$ symbols and p'_{M-1} for the new symbol $M-1$.

To complete the algorithm, an optimal code is constructed for X'. It will be shown that an optimal code for X can be generated by constructing an optimal code for X', and then grafting siblings onto the leaf corresponding to $M-1$. The proof is postponed until giving an example of the algorithm.

The following example in Figures 2.10 to 2.12 starts with a random symbol X with probabilities $\{0.4, 0.2, 0.15, 0.15, 0.1\}$. It first generates the reduced random symbol X' and then recursively uses the same procedure on X'.

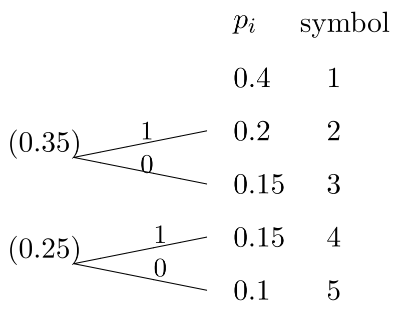


The two least likely symbols, 4 and 5 have been combined as siblings. The reduced set of probabilities then becomes {0.4, 0.2, 0.15, 0.25}.

Figure 2.9: Step 1 of the Huffman algorithm; finding X' from X

.

[11]Exercise 2.10 proves this for those who enjoy such things.



The two least likely symbols in the reduced set, with probabilities 0.15 and 0.2, have been combined as siblings. The reduced set of probabilities then becomes {0.4, 0.35, 0.25}.

Figure 2.10: Finding X'' from X'.

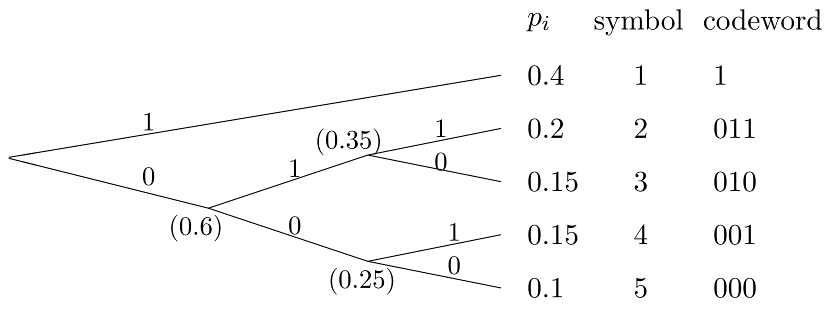


Figure 2.11: The completed Huffman code.

Another completed example that leads to a different set of codeword lengths is given in Figure 2.12:

It is next shown that an optimal code for the reduced random symbol X' yields an optimal code for X. First consider Figure 2.13, which shows the code tree for X' corresponding to X in Figure 2.12.

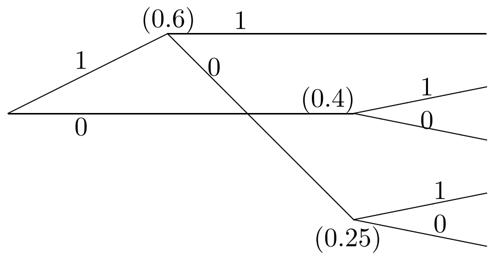
Note that Figures 2.12 and 2.13 differ in that $\mathcal{C}(4)$ and $\mathcal{C}(5)$, each of length 3 in Figure 2.12, have been replaced by a single codeword of length 2 in Figure 2.13. The probability of that single symbol is the sum of the two probabilities in Figure 2.12. Thus the expected codeword length for Figure 2.12 is that for Figure 2.13, increased by $p_4 + p_5$. This accounts for the unit length increase for $\mathcal{C}(4)$ and $\mathcal{C}(5)$.

In general, comparing the expected length $\overline{L}'$ of *any* code for X' and the corresponding $\overline{L}$ of the code generated by extending $\mathcal{C}'(M-1)$ in the code for X' into two siblings for $M-1$ and M, it is seen that

$$\overline{L} = \overline{L}' + p_{M-1} + p_M.$$

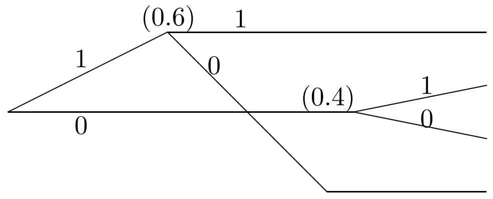
This relationship holds for all codes for X in which $\mathcal{C}(M-1)$ and $\mathcal{C}(M)$ are siblings (which includes at least one optimal code). This proves that $\overline{L}$ is minimized by minimizing $\overline{L}'$, and also shows that $\overline{L}_{\min} = \overline{L}'_{\min} + p_{M-1} + p_M$. This completes the proof of the optimality of the Huffman algorithm.

It is curious that neither the Huffman algorithm nor its proof of optimality give any indication of the entropy bounds, $\mathsf{H}[X] \le \overline{L}_{\min} < \mathsf{H}[X] + 1$. Similarly, the entropy bounds do not suggest



p_i	symbol	codeword
0.35	1	11
0.2	2	01
0.2	3	00
0.15	4	101
0.1	5	100

Figure 2.12: Completed Huffman code for a different set of probabilities.



p_i	symbol	codeword
0.35	1	11
0.2	2	01
0.2	3	00
0.25	4	10

Figure 2.13: Completed reduced Huffman code for Figure 2.12.

the Huffman algorithm. One is useful in finding an optimal code; the other provides insightful performance bounds.

As an example of the extent to which the optimal lengths approximate $-\log p_i$, the source probabilities in Figure 2.11 are $\{0.40, 0.20, 0.15, 0.15, 0.10\}$, so $-\log p_j$ takes the set of values $\{1.32, 2.32, 2.74, 2.74, 3.32\}$ bits; this approximates the lengths $\{1, 3, 3, 3, 3\}$ of the optimal code quite well. Similarly, the entropy is $\mathsf{H}[X] = 2.15$ bits/symbol and $\overline{L}_{\min} = 2.2$ bits/symbol, quite close to $\mathsf{H}[X]$. However, it would be difficult to guess these optimal lengths, even in such a simple case, without the algorithm.

For the example of Figure 2.12, the source probabilities are $\{0.35, 0.20, 0.20, 0.15, 0.10\}$, the values of $-\log p_i$ are $\{1.51, 2.32, 2.32, 2.74, 3.32\}$, and the entropy is $\mathsf{H}[X] = 2.20$. This is not very different from Figure 2.11. However, the Huffman code now has lengths $\{2, 2, 2, 3, 3\}$ and average length $\overline{L} = 2.25$ bits/symbol. (The code of Figure 2.11 has average length $\overline{L} = 2.30$ for these source probabilities.) It would be hard to predict these perturbations without carrying out the algorithm.

2.6 Entropy and fixed-to-variable-length codes

Entropy is now studied in more detail, both to better understand the entropy bounds and to understand the entropy of n-tuples of successive source letters.

The entropy $\mathsf{H}[X]$ is a fundamental measure of the randomness of a random symbol X. It has many important properties. The property of greatest interest here is that it is the smallest expected number $\overline{L}$ of bits per source symbol required to map the sequence of source symbols into a bit sequence in a uniquely decodable way. This will soon be demonstrated by generalizing the variable-length codes of the last few sections to codes in which multiple source symbols are encoded together. First, however, several other properties of entropy are derived.

Definition: The *entropy* of a discrete random symbol[12] X with alphabet $\mathcal{X}$ is

$$\mathsf{H}[X] = \sum_{x\in\mathcal{X}} p_X(x)\log\frac{1}{p_X(x)} = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x). \tag{2.11}$$

Using logarithms to the base 2, the units of $\mathsf{H}[X]$ are *bits/symbol.* If the base of the logarithm is e, then the units of $\mathsf{H}[X]$ are called nats/symbol. Conversion is easy; just remember that $\log y = (\ln y)/(\ln 2)$ or $\ln y = (\log y)/(\log e)$, both of which follow from $y = e^{\ln y} = 2^{\log y}$ by taking logarithms. Thus using another base for the logarithm just changes the numerical units of entropy by a scale factor.

Note that the entropy $\mathsf{H}[X]$ of a discrete random symbol X depends on the probabilities of the different outcomes of X, but not on the names of the outcomes. Thus, for example, the entropy of a random symbol taking the values GREEN, BLUE, and RED with probabilities $0.2, 0.3, 0.5$, respectively, is the same as the entropy of a random symbol taking on the values SUNDAY, MONDAY, FRIDAY with the same probabilities $0.2, 0.3, 0.5$.

The entropy $\mathsf{H}[X]$ is also called the *uncertainty* of X, meaning that it is a measure of the randomness of X. Note that entropy is the expected value of the rv $\log(1/p_X(X))$. This random variable is called the *log pmf* rv.[13] Thus the entropy is the expected value of the log pmf rv.

Some properties of entropy:

- For any discrete random symbol X, $\mathsf{H}[X] \geq 0$. This follows because $p_X(x) \leq 1$, so $\log(1/p_X(x)) \geq 0$. The result follows from (2.11).
- $\mathsf{H}[X] = 0$ if and only if X is deterministic. This follows since $p_X(x)\log(1/p_X(x)) = 0$ if and only if $p_X(x)$ equals 0 or 1.
- The entropy of an equiprobable random symbol X with an alphabet $\mathcal{X}$ of size M is $\mathsf{H}[X] = \log M$. This follows because, if $p_X(x) = 1/M$ for all $x \in \mathcal{X}$, then

  $$\mathsf{H}[X] = \sum_{x\in\mathcal{X}} \frac{1}{M}\log M = \log M.$$

  In this case, the rv $-\log p_X(X)$ has the constant value $\log M$.
- More generally, the entropy $\mathsf{H}[X]$ of a random symbol X defined on an alphabet $\mathcal{X}$ of size M satisfies $\mathsf{H}[X] \leq \log M$, with equality only in the equiprobable case. To see this, note

[12] If one wishes to consider discrete random symbols with one or more symbols of zero probability, one can still use this formula by recognizing that $\lim_{p\to 0} p\log(1/p) = 0$ and then defining $0\log 1/0$ as 0 in (2.11). Exercise 2.18 illustrates the effect of zero probability symbols in a variable-length prefix code.

[13] This rv is often called *self information* or *surprise*, or *uncertainty.* It bears some resemblance to the ordinary meaning of these terms, but historically this has caused much more confusion than enlightenment. Log pmf, on the other hand, emphasizes what is useful here

that

$$\begin{aligned} \mathsf{H}[X] - \log M &= \sum_{x\in\mathcal{X}} p_X(x)\left[\log\frac{1}{p_X(x)} - \log M\right] = \sum_{x\in\mathcal{X}} p_X(x)\left[\log\frac{1}{Mp_X(x)}\right] \\ &\le (\log e)\sum_{x\in\mathcal{X}} p_X(x)\left[\frac{1}{Mp_X(x)} - 1\right] = 0, \end{aligned}$$

This uses the inequality $\log u \le (\log e)(u-1)$ (after omitting any terms for which $p_X(x) = 0$). For equality, it is necessary that $p_X(x) = 1/M$ for all $x \in \mathcal{X}$.

In summary, of all random symbols X defined on a given finite alphabet $\mathcal{X}$, the highest entropy occurs in the equiprobable case, namely $\mathsf{H}[X] = \log M$, and the lowest occurs in the deterministic case, namely $\mathsf{H}[X] = 0$. This supports the intuition that the entropy of a random symbol X is a measure of its randomness.

For any pair of discrete random symbols X and Y, XY is another random symbol. The sample values of XY are the set of all pairs xy, $x \in \mathcal{X}, y \in \mathcal{Y}$ and the probability of each sample value xy is $p_{XY}(x,y)$. An important property of entropy is that if X and Y are independent discrete random symbols, then $\mathsf{H}[XY] = \mathsf{H}[X] + \mathsf{H}[Y]$. This follows from:

$$\begin{aligned} \mathsf{H}[XY] &= -\sum_{\mathcal{X}\times\mathcal{Y}} p_{XY}(x,y)\log p_{XY}(x,y) \\ &= -\sum_{\mathcal{X}\times\mathcal{Y}} p_X(x)p_Y(y)\left(\log p_X(x) + \log p_Y(y)\right) = \mathsf{H}[X] + \mathsf{H}[Y]. \end{aligned} \tag{2.12}$$

Extending this to n random symbols, the entropy of a random symbol $\boldsymbol{X}^n$ corresponding to a block of n iid outputs from a discrete memoryless source is $\mathsf{H}[\boldsymbol{X}^n] = n\mathsf{H}[X]$; *i.e.*, each symbol increments the entropy of the block by $\mathsf{H}[X]$ bits.

2.6.1 Fixed-to-variable-length codes

Recall that in Section 2.2 the sequence of symbols from the source was segmented into successive blocks of n symbols which were then encoded. Each such block was a discrete random symbol in its own right, and thus could be encoded as in the single-symbol case. It was seen that by making n large, fixed-length codes could be constructed in which the number $\overline{L}$ of encoded bits per source symbol approached $\log M$ as closely as desired.

The same approach is now taken for cariable-length coding of discrete memoryless sources. A block of n source symbols, $X_1, X_2, \dots, X_n$ has entropy $\mathsf{H}(\boldsymbol{X}^n) = n\mathsf{H}(X)$. Such a block is a random symbol in its own right and can be encoded using a variable-length prefix-free code. This provides a fixed-to-variable-length code, mapping n-tuples of source symbols to variable-length binary sequences. It will be shown that the expected number $\overline{L}$ of encoded bits per source symbol can be made as close to $\mathsf{H}[X]$ as desired.

Surprisingly, this result is very simple. Let $\mathsf{E}[L(\boldsymbol{X}^n)]$ be the expected length of a variable-length prefix-free code for $\boldsymbol{X}^n$. Denote the minimum expected length of any prefix-free code for $\boldsymbol{X}^n$ by $\mathsf{E}[L(\boldsymbol{X}^n)]_{\min}$. Theorem 2.5.1 then applies. Using (2.7),

$$\mathsf{H}[\boldsymbol{X}^n] \le \mathsf{E}[L(\boldsymbol{X}^n)]_{\min} < \mathsf{H}[\boldsymbol{X}^n] + 1. \tag{2.13}$$

Define $\overline{L}_{\min,n} = \frac{\mathsf{E}[L(\boldsymbol{X}^n)]_{\min}}{n}$; *i.e.*, $\overline{L}_{\min,n}$ is the minimum number of bits per source symbol over all prefix-free codes for $\boldsymbol{X}^n$. From (2.13),

$$\mathsf{H}[X] \leq \overline{L}_{\min,n} < \mathsf{H}[X] + \frac{1}{n}. \tag{2.14}$$

This simple result establishes the following important theorem:

Theorem 2.6.1 (Prefix-free source coding theorem). *For any discrete memoryless source with entropy $\mathsf{H}[X]$, and any integer $n \geq 1$, there exists a prefix-free encoding of source n-tuples for which the expected codeword length per source symbol $\overline{L}$ is at most $\mathsf{H}[X] + 1/n$. Furthermore, no prefix-free encoding of fixed-length source blocks of any length n results in an expected codeword length $\overline{L}$ less than $\mathsf{H}[X]$.*

This theorem gives considerable significance to the entropy $\mathsf{H}[X]$ of a discrete memoryless source: $\mathsf{H}[X]$ is the minimum expected number $\overline{L}$ of bits per source symbol that can be achieved by fixed-to-variable-length prefix-free codes.

There are two potential questions about the significance of the theorem. First, is it possible to find uniquely-decodable codes other than prefix-free codes for which $\overline{L}$ is less than $\mathsf{H}[X]$? Second, is it possible to further reduce $\overline{L}$ by using variable-to-variable-length codes?

For example, if a binary source has $p_1 = 10^{-6}$ and $p_0 = 1 - 10^{-6}$, fixed-to-variable-length codes must use remarkably long n-tuples of source symbols to approach the entropy bound. Run-length coding, which is an example of variable-to-variable-length coding, is a more sensible approach in this case: the source is first encoded into a sequence representing the number of source 0's between each 1, and then this sequence of integers is encoded. This coding technique is further developed in Exercise 2.23.

The next section strengthens Theorem 2.6.1, showing that $\mathsf{H}[X]$ is indeed a lower bound to $\overline{L}$ over all uniquely-decodable encoding techniques.

2.7 The AEP and the source coding theorems

We first review the weak[14] law of large numbers (WLLN) for sequences of iid rv's. Applying the WLLN to a particular iid sequence, we will establish a form of the remarkable asymptotic equipartition property (AEP).

Crudely, the AEP says that, given a very long string of n iid discrete random symbols $X_1, \ldots, X_n$, there exists a "typical set" of sample strings $(x_1, \ldots, x_n)$ whose aggregate probability is almost 1. There are roughly $2^{n\mathsf{H}[X]}$ typical strings of length n, and each has a probability roughly equal to $2^{-n\mathsf{H}[X]}$. We will have to be careful about what the words "almost" and "roughly" mean here.

The AEP will give us a fundamental understanding not only of source coding for discrete memoryless sources, but also of the probabilistic structure of such sources and the meaning of entropy. The AEP will show us why general types of source encoders, such as variable-to-variable-length encoders, cannot have a strictly smaller expected length per source symbol than the best fixed-to-variable-length prefix-free codes for discrete memoryless sources.

[14]The word *weak* is something of a misnomer, since this is one of the most useful results in probability theory. There is also a strong law of large numbers; the difference lies in the limiting behavior of an infinite sequence of rv's, but this difference is not relevant here. The weak law applies in some cases where the strong law does not, but this also is not relevant here.

2.7.1 The weak law of large numbers

Let $Y_1, Y_2, \dots ,$ be a sequence of iid rv's. Let $\overline{Y}$ and σ_Y^2 be the mean and variance of each Y_j. Define the *sample average* A_Y^n of $Y_1, \dots , Y_n$ as

$$A_Y^n = \frac{S_Y^n}{n} \quad where \quad S_Y^n = Y_1 + \cdots + Y_n.$$

The sample average A_Y^n is itself an rv, whereas, of course, the mean $\overline{Y}$ is simply a real number. Since the sum S_Y^n has mean $n\overline{Y}$ and variance $n\sigma_Y^2$, the sample average A_Y^n has mean $\mathsf{E}[A_Y^n] = \overline{Y}$ and variance $\sigma^2_{A_Y^n} = \sigma^2_{S_Y^n}/n^2 = \sigma_Y^2/n$. It is important to understand that the variance of the sum *increases* with n and the variance of the normalized sum (the sample average, A_Y^n), *decreases* with n.

The Chebyshev inequality states that if $\sigma_X^2 < \infty$ for an rv X, then, $\Pr\{|X - \overline{X}| \geq \varepsilon\} \leq \sigma_X^2/\varepsilon^2$ for any $\varepsilon > 0$ (see Exercise 2.3 or any text on probability such as [?]). Applying this inequality to A_Y^n yields the simplest form of the WLLN: for any $\varepsilon > 0$,

$$\Pr\{|A_Y^n - \overline{Y}| \geq \varepsilon\} \leq \frac{\sigma_Y^2}{n\varepsilon^2}. \tag{2.15}$$

This is illustrated in Figure 2.14.

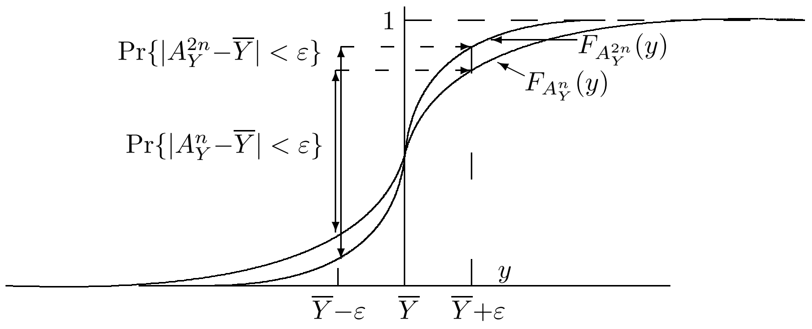


Figure 2.14: Sketch of the distribution function of the sample average for different n. As n increases, the distribution function approaches a unit step at $\overline{Y}$. The closeness to a step within $\overline{Y} \pm \varepsilon$ is upper bounded by (2.15).

Since the right side of (2.15) approaches 0 with increasing n for any fixed $\varepsilon > 0$,

$$\lim_{n\to\infty} \Pr\{|A_Y^n - \overline{Y}| \geq \varepsilon\} = 0. \tag{2.16}$$

For large n, (2.16) says that $A_Y^n - \overline{Y}$ is small with high probability. It does not say that $A_Y^n = \overline{Y}$ with high probability (or even nonzero probability), and it does not say that $\Pr(|A_Y^n - \overline{Y}| \geq \varepsilon) = 0$. As illustrated in Figure 2.14, both a nonzero ε and a nonzero probability are required here, even though they can be made simultaneously as small as desired by increasing n.

In summary, the sample average A_Y^n is an rv whose mean $\overline{Y}$ is independent of n, but whose standard deviation $\sigma_Y/\sqrt{n}$ approaches 0 as $n \to \infty$. Therefore the distribution of the sample average becomes concentrated near $\overline{Y}$ as n increases. The WLLN is simply this concentration property, stated more precisely by either (2.15) or (2.16).

The WLLN, in the form of (2.16), applies much more generally than the simple case of iid rv's. In fact, (2.16) provides the central link between probability models and the real phenomena being modeled. One can observe the outcomes both for the model and reality, but probabilities are assigned only for the model. The WLLN, applied to a sequence of rv's in the model, and the concentration property (if it exists), applied to the corresponding real phenomenon, provide the basic check on whether the model corresponds reasonably to reality.

2.7.2 The asymptotic equipartition property

This section starts with a sequence of iid random symbols and defines a sequence of random variables (rv's) as functions of those symbols. The WLLN, applied to these rv's, will permit the classification of sample sequences of symbols as being 'typical' or not, and then leadto the results alluded to earlier.

Let $X_1, X_2, \dots$ be the sequence of iid discrete random symbols with a common pmf $p_X(x){>}0$, $x{\in}\mathcal{X}$. For each symbol x in the alphabet $\mathcal{X}$, define $w(x) = -\log p_X(x)$ as a real-valued function of $x \in \mathcal{X}$. For each m, define $W(X_m)$ to be the rv that takes the value $w(x)$ for $X_m = x$. Then $W(X_1), W(X_2), \dots$ is a sequence of iid discrete rv's, each with mean

$$\mathsf{E}[W(X_m)] = -\sum_{x\in\mathcal{X}} p_X(x)\log p_X(x) = \mathsf{H}[X], \tag{2.17}$$

where $\mathsf{H}[X]$ is the entropy of the random symbol X.

The rv $W(X_m)$ is called[15] the *log pmf* of X_m and the entropy of X_m is the mean of $W(X_m)$.

The most important property of the log pmf for iid random symbols comes from observing, for example, that for the event $X_1 = x_1, X_2 = x_2$, the outcome for $W(X_1) + W(X_2)$ is

$$w(x_1) + w(x_2) = -\log p_X(x_1) - \log p_X(x_2) = -\log\{p_{X_1X_2}(x_1x_2)\}. \tag{2.18}$$

In other words, the joint pmf for independent random symbols is the product of the individual pmf's, and therefore *the log of the joint pmf is the sum of the logs of the individual pmf's.*

We can generalize (2.18) to a string of n random symbols, $\boldsymbol{X}^n = (X_1, \dots, X_n)$. For an event $\boldsymbol{X}^n = \boldsymbol{x}^n$ where $\boldsymbol{x}^n = (x_1, \dots, x_n)$, the outcome for the sum $W(X_1) + \cdots + W(X_n)$ is

$$\sum\nolimits_{m=1}^{n} w(x_m) = -\sum\nolimits_{m=1}^{n} \log p_X(x_m) = -\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n). \tag{2.19}$$

The WLLN can now be applied to the sample average of the log pmfs. Let

$$A_W^n = \frac{W(X_1) + \cdots + W(X_n)}{n} = \frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{X}^n)}{n} \tag{2.20}$$

be the sample average of the log pmf.

From (2.15), it follows that

$$\Pr\Big(\big|A_W^n - \mathsf{E}[W(X)]\big| \geq \varepsilon\Big) \leq \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.21}$$

[15]It is also called self information and various other terms which often cause confusion.

Substituting (2.17) and (2.20) into (2.21),

$$\Pr\left(\left|\frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{X}^n)}{n} - \mathsf{H}[X]\right| \geq \varepsilon\right) \leq \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.22}$$

In order to interpret this result, define the *typical set* T_ε^n for any $\varepsilon > 0$ as

$$T_\varepsilon^n = \left\{\boldsymbol{x}^n : \left|\frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)}{n} - \mathsf{H}[X]\right| < \varepsilon\right\}. \tag{2.23}$$

Thus T_ε^n is the set of source strings of length n for which the sample average of the log pmf is within ε of its mean $\mathsf{H}[X]$. Eq. (2.22) then states that the aggregrate probability of all strings of length n not in T_ε^n is at most $\sigma_W^2/(n\varepsilon^2)$. Thus,

$$\Pr(\boldsymbol{X}^n \in T_\varepsilon^n) \geq 1 - \frac{\sigma_W^2}{n\varepsilon^2}. \tag{2.24}$$

As n increases, the aggregate probability of T_ε^n approaches 1 for any given $\varepsilon > 0$, so T_ε^n is certainly a typical set of source strings. This is illustrated in Figure 2.15.

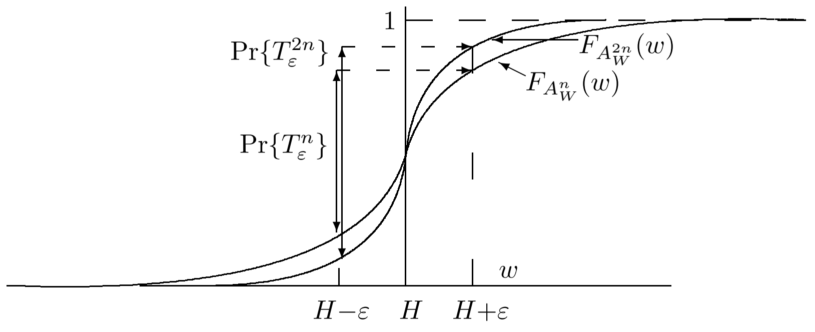


Figure 2.15: Sketch of the distribution function of the sample average log pmf. As n increases, the distribution function approaches a unit step at H. The typical set is the set of sample strings of length n for which the sample average log pmf stays within ε of H; as illustrated, its probability approaches 1 as $n \to \infty$.

Rewrite (2.23) in the form

$$T_\varepsilon^n = \left\{\boldsymbol{x}^n : \; n(\mathsf{H}[X] - \varepsilon) < -\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < n(\mathsf{H}[X] + \varepsilon)\right\}.$$

Multiplying by -1 and exponentiating,

$$T_\varepsilon^n = \left\{\boldsymbol{x}^n : \; 2^{-n(\mathsf{H}[X]+\varepsilon)} < p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < 2^{-n(\mathsf{H}[X]-\varepsilon)}\right\}. \tag{2.25}$$

Eq. (2.25) has the intuitive connotation that the n-strings in T_ε^n are approximately equiprobable. This is the same kind of approximation that one would use in saying that $10^{-1001} \approx 10^{-1000}$; these numbers differ by a factor of 10, but for such small numbers it makes sense to compare the exponents rather than the numbers themselves. In the same way, the ratio of the upper to lower bound in (2.25) is $2^{2\varepsilon n}$, which grows unboundedly with n for fixed ε. However, as seen in (2.23),

$-\frac{1}{n}\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ is approximately equal to $\mathsf{H}[X]$ for all $\boldsymbol{x}^n \in T_\varepsilon^n$. This is the important notion, and it does no harm to think of the n-strings in T_ε^n as being approximately equiprobable.

The set of all n-strings of source symbols is thus separated into the typical set T_ε^n and the complementary atypical set $(T_\varepsilon^n)^c$. The atypical set has aggregate probability no greater than $\sigma_W^2/(n\varepsilon^2)$, and the elements of the typical set are approximately equiprobable (in this peculiar sense), each with probability about $2^{-n\mathsf{H}[X]}$.

The typical set T_ε^n depends on the choice of ε. As ε decreases, the equiprobable approximation (2.25) becomes tighter, but the bound (2.24) on the probability of the typical set is further from 1. As n increases, however, ε can be slowly decreased, thus bringing the probability of the typical set closer to 1 and simultaneously tightening the bounds on equiprobable strings.

Let us now estimate the number of elements $|T_\varepsilon^n|$ in the typical set. Since $p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) > 2^{-n(\mathsf{H}[X]+\varepsilon)}$ for each $\boldsymbol{x}^n \in T_\varepsilon^n$,

$$1 \geq \sum_{\boldsymbol{x}^n \in T_\varepsilon^n} p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) > |T_\varepsilon^n|\, 2^{-n(\mathsf{H}[X]+\varepsilon)}.$$

This implies that $|T_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\varepsilon)}$. In other words, since each $\boldsymbol{x}^n \in T_\varepsilon^n$ contributes at least $2^{-n(\mathsf{H}[X]+\varepsilon)}$ to the probability of T_ε^n, the number of these contributions can be no greater than $2^{n(\mathsf{H}[X]+\varepsilon)}$.

Conversely, since $\Pr(T_\varepsilon^n) \geq 1 - \sigma_W^2/(n\varepsilon^2)$, $|T_\varepsilon^n|$ can be lower bounded by

$$1 - \frac{\sigma_W^2}{n\varepsilon^2} \leq \sum_{\boldsymbol{x}^n \in T_\varepsilon^n} p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) < |T_\varepsilon^n| 2^{-n(\mathsf{H}[X]-\varepsilon)},$$

which implies $|T_\varepsilon^n| > [1 - \sigma_W^2/(n\varepsilon^2)]2^{n(\mathsf{H}[X]-\varepsilon)}$. In summary,

$$\left(1 - \frac{\sigma_W^2}{n\varepsilon^2}\right) 2^{n(\mathsf{H}[X]-\varepsilon)} < |T_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\varepsilon)}. \tag{2.26}$$

For large n, then, the typical set T_ε^n has aggregate probability approximately 1 and contains approximately $2^{n\mathsf{H}[X]}$ elements, each of which has probability approximately $2^{-n\mathsf{H}[X]}$. That is, asymptotically for very large n, the random symbol $\boldsymbol{X}^n$ resembles an equiprobable source with alphabet size $2^{n\mathsf{H}[X]}$.

The quantity $\sigma_W^2/(n\varepsilon^2)$ in many of the equations above is simply a particular upper bound to the probability of the atypical set. It becomes arbitrarily small as n increases for any fixed $\varepsilon > 0$. Thus it is insightful to simply replace this quantity with a real number δ; for any such $\delta > 0$ and any $\varepsilon > 0$, $\sigma_W^2/(n\varepsilon^2) \leq \delta$ for large enough n.

This set of results, summarized in the following theorem, is known as the *asymptotic equipartition property* (AEP).

Theorem 2.7.1 (Asymptotic equipartition property). *Let $\boldsymbol{X}^n$ be a string of n iid discrete random symbols $\{X_m; 1 \leq m \leq n\}$ each with entropy $H[X]$. For all $\delta > 0$ and all sufficiently large n, $\Pr(T_\varepsilon^n) \geq 1 - \delta$ and $|T_\varepsilon^n|$ is bounded by*

$$(1-\delta)2^{n(H[X]-\varepsilon)} < |T_\varepsilon^n| < 2^{n(H[X]+\varepsilon)}. \tag{2.27}$$

Finally, note that the total number of different strings of length n from a source with alphabet size M is M^n. For non-equiprobable sources, namely sources with $\mathsf{H}[X] < \log M$, the ratio of the number of typical strings to total strings is approximately $2^{-n(\log M - \mathsf{H}[X])}$, which approaches 0 exponentially with n. Thus, for large n, the great majority of n-strings are atypical. It may be somewhat surprising that this great majority counts for so little in probabilistic terms. As shown in Exercise 2.26, the most probable of the individual sequences are also atypical. There are too few of them, however, to have any significance.

We next consider source coding in the light of the AEP.

2.7.3 Source coding theorems

Motivated by the AEP, we can take the approach that an encoder operating on strings of n source symbols need only provide a codeword for each string $\boldsymbol{x}^n$ in the typical set T_ε^n. If a sequence $\boldsymbol{x}^n$ occurs that is not in T_ε^n, then a source coding failure is declared. Since the probability of $\boldsymbol{x}^n \notin T_\varepsilon^n$ can be made arbitrarily small by choosing n large enough, this situation is tolerable.

In this approach, since there are less than $2^{n(\mathsf{H}[X]+\varepsilon)}$ strings of length n in T_ε^n, the number of source codewords that need to be provided is fewer than $2^{n(\mathsf{H}[X]+\varepsilon)}$. Choosing fixed-length codewords of length $\lceil n(\mathsf{H}[X]+\varepsilon)\rceil$ is more than sufficient and even allows for an extra codeword, if desired, to indicate that a coding failure has occurred. In bits per source symbol, taking the ceiling function into account, $\overline{L} \leq \mathsf{H}[X]+\varepsilon+1/n$. Note that $\varepsilon > 0$ is arbitrary, and for any such ε, $\Pr\{\text{failure}\} \to 0$ as $n \to \infty$. This proves the following theorem:

Theorem 2.7.2 (Fixed-to-fixed-length source coding theorem). *For any discrete memoryless source with entropy $\mathsf{H}[X]$, any $\varepsilon > 0$, any $\delta > 0$, and any sufficiently large n, there is a fixed-to-fixed-length source code with* $\Pr\{\text{failure}\} \leq \delta$ *that maps blocks of n source symbols into fixed-length codewords of length $\overline{L} \leq \mathsf{H}[X] + \varepsilon + 1/n$ bits per source symbol.*

We saw in section 2.2 that the use of fixed-to-fixed-length source coding requires $\log M$ bits per source symbol if unique decodability is required (*i.e.*, no failures are allowed), and now we see that this is reduced to arbitrarily little more than $\mathsf{H}[X]$ bits per source symbol if arbitrarily rare failures are allowed. This is a good example of a situation where 'arbitrarily small $\delta > 0$' and 0 behave very differently.

There is also a converse to this theorem following from the other side of the AEP theorem. This says that the error probability approaches 1 for large n if strictly fewer than $\mathsf{H}[X]$ bits per source symbol are provided.

Theorem 2.7.3 (Converse for fixed-to-fixed-length codes). *Let $\boldsymbol{X}^n$ be a string of n iid discrete random symbols $\{X_m; 1 \leq m \leq n\}$, with entropy $\mathsf{H}[X]$ each. For any $\nu > 0$, let $\boldsymbol{X}^n$ be encoded into fixed-length codewords of length $\lfloor n(\mathsf{H}[X] - \nu)\rfloor$ bits. For every $\delta > 0$ and for all sufficiently large n given δ,*

$$\Pr\{\text{failure}\} > 1 - \delta - 2^{-\nu n/2}. \tag{2.28}$$

Proof: Apply the AEP, Theorem 2.7.1, with $\varepsilon = \nu/2$. Codewords can be provided for at most $2^{n(\mathsf{H}[X]-\nu)}$ typical source n-sequences, and from (2.25) each of these has a probability at most $2^{-n(\mathsf{H}[X]-\nu/2)}$. Thus the aggregate probability of typical sequences for which codewords are provided is at most $2^{-n\nu/2}$. From the AEP theorem, $\Pr\{T_\varepsilon^n\} \geq 1 - \delta$ is satisfied for large enough n. Codewords[16] can be provided for at most a subset of T_ε^n of probability $2^{-n\nu/2}$, and

[16]Note that the proof allows codewords to be provided for atypical sequences; it simply says that a large portion of the typical set cannot be encoded.

the remaining elements of T_ε^n must all lead to errors, thus yielding (2.28).
□

In going from fixed-length codes of slightly more than $\mathsf{H}[X]$ bits per source symbol to codes of slightly less than $\mathsf{H}[X]$ bits per source symbol, the probability of failure goes from almost 0 to almost 1, and as n increases, those limits are approached more and more closely.

2.7.4 The entropy bound for general classes of codes

We have seen that the expected number of encoded bits per source symbol is lower bounded by $\mathsf{H}(X)$ for iid sources using either fixed-to-fixed-length or fixed-to-variable-length codes. The details differ in the sense that very improbable sequences are simply dropped in fixed-length schemes but have abnormally long encodings, leading to buffer overflows, in variable-length schemes.

We now show that other types of codes, such as variable-to-fixed, variable-to-variable, and even more general codes are also subject to the entropy limit. This will be done without describing the highly varied possible nature of these source codes, but by just defining certain properties that the associated decoders must have. By doing this, it is also shown that yet undiscovered coding schemes must also be subject to the same limits. The fixed-to-fixed-length converse in the last subsection is the key to this.

For any encoder, there must be a decoder that maps the encoded bit sequence back into the source symbol sequence. For prefix-free codes on k-tuples of source symbols, the decoder waits for each variable length codeword to arrive, maps it into the corresponding k-tuple of source symbols, and then starts decoding for the next k-tuple. For fixed-to-fixed-length schemes, the decoder waits for a block of code symbols and then decodes the corresponding block of source symbols.

In general, the source produces a non-ending sequence $X_1, X_2, \dots$ of source letters which are encoded into a non-ending sequence of encoded binary digits. The decoder observes this encoded sequence and decodes source symbol X_n when enough bits have arrived to make a decision on it.

For any given coding and decoding scheme for a given iid source, define the rv D_n as the number of received bits that permit a decision on $\boldsymbol{X}^n = X_1, \dots, X_n$. This includes the possibility of coders and decoders for which decoding is either incorrect or postponed indefinitely, and for these failure instances, the sample value of D_n is taken to be infinite. It is assumed, however, that all decisions are final in the sense that the decoder cannot decide on a particular $\boldsymbol{x}^n$ after observing an initial string of the encoded sequence and then change that decision after observing more of the encoded sequence. What we would like is a scheme in which decoding is correct with high probability and the sample value of the rate, D_n/n, is small with high probability. What the following theorem shows is that for large n, the sample rate can be strictly below the entropy only with vanishingly small probability. This then shows that the entropy lower bounds the data rate in this strong sense.

Theorem 2.7.4 (Converse for general coders/decoders for iid sources). *Let $\boldsymbol{X}^\infty$ be a sequence of discrete random symbols $\{X_m; 1 \le m \le \infty\}$. For each integer $n \ge 1$, let $\boldsymbol{X}^n$ be the first n of those symbols. For any given encoder and decoder, let D_n be the number of received bits at which the decoder can correctly decode $\boldsymbol{X}^n$. Then for any $\nu > 0$ and $\delta > 0$, and for any* *sufficiently large n given ν and δ,*

$$\Pr\{D_n \le n[H(X) - \nu]\} < \delta + 2^{-\nu n/2}. \tag{2.29}$$

Proof: For any sample value $\boldsymbol{x}^\infty$ of the source sequence, let $\boldsymbol{y}^\infty$ denote the encoded sequence. For any given integer $n \ge 1$, let $m = \lfloor n[\mathsf{H}(X)-\nu] \rfloor$. Suppose that $\boldsymbol{x}^n$ is decoded upon observation of $\boldsymbol{y}^j$ for some $j \le m$. Since decisions are final, there is only one source n-string, namely $\boldsymbol{x}^n$, that can be decoded by time $\boldsymbol{y}^m$ is observed. This means that out of the 2^m possible initial m-strings from the encoder, there can be at most[17] 2^m n-strings from the source that be decoded from the observation of the first m encoded outputs. The aggregate probability of any set of 2^m source n-strings is bounded in Theorem 2.7.3, and (2.29) simply repeats that bound.
□

2.8 Markov sources

The basic coding results for discrete memoryless sources have now been derived. Many of the results, in particular the Kraft inequality, the entropy bounds on expected length for uniquely-decodable codes, and the Huffman algorithm, do not depend on the independence of successive source symbols.

In this section, these results are extended to sources defined in terms of finite-state Markov chains. The state of the Markov chain[18] is used to represent the "memory" of the source. Labels on the transitions between states are used to represent the next symbol out of the source. Thus, for example, the state could be the previous symbol from the source, or could be the previous 300 symbols. It is possible to model as much memory as desired while staying in the regime of finite-state Markov chains.

Example: Consider a binary source with outputs $X_1, X_2, \dots$. Assume that the symbol probabilities for X_n are conditioned on X_{n-2} and X_{n-1} but are independent of all previous symbols given these past 2 symbols. This pair of previous symbols is modeled by a *state* S_{n-1}. The alphabet of possible states is then the set of binary pairs, $\mathcal{S} = \{[00], [01], [10], [11]\}$. In Figure 2.16, the states are represented as the nodes of the graph representing the Markov chain, and the source outputs are labels on the graph transitions. Note, for example, that from the state $S_{n-1} = [01]$ (representing $X_{n-2}{=}0, X_{n-1}{=}1$), the output $X_n{=}1$ causes a transition to $S_n = [11]$ (representing $X_{n-1}{=}1, X_n{=}1$). The chain is assumed to start at time 0 in a state S_0 given by some arbitrary pmf.

Note that this particular source is characterized by long strings of zeros and long strings of ones interspersed with short transition regions. For example, starting in state 00, a representative output would be

$$00000000101111111111111011111111010100000000 \cdots$$

[17] There are two reasons why the number of decoded n-strings of source symbols by time m can be less than 2^m. The first is that the first n source symbols might not be decodable until after the mth encoded bit is received. The second is that multiple m-strings of encoded bits might lead to decoded strings with the same first n source symbols.

[18] The basic results about finite-state Markov chains, including those used here, are established in many texts such as [5] and [16] . These results are important in the further study of digital communcation, but are not essential here.

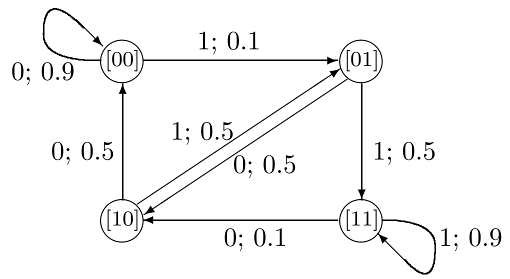


Figure 2.16: Markov source: Each transition $s' \to s$ is labeled by the corresponding source output and the transition probability $\Pr\{S_n = s|S_{n-1} = s'\}$.

Note that if $s_n = [x_{n-1}x_n]$ then the next state must be either $s_{n+1} = [x_n 0]$ or $s_{n+1} = [x_n 1]$; *i.e.*, each state has only two transitions coming out of it.

The above example is now generalized to an arbitrary discrete Markov source.

Definition: A *finite-state Markov chain* is a sequence $S_0, S_1, \dots$ of discrete random symbols from a finite alphabet, $\mathcal{S}$. There is a pmf $q_0(s)$, $s \in \mathcal{S}$ on S_0, and for all $n \geq 1$ and all $s \in \mathcal{S}$, $s' \in \mathcal{S}$,

$$\Pr(S_n{=}s|\, S_{n-1}{=}s') = \Pr(S_n{=}s|\, S_{n-1}{=}s', \dots , S_0{=}s_0) = Q(s|\, s'). \tag{2.30}$$

There is said to be a *transition* from s' to s, denoted $s' \to s$, if $Q(s|\, s') > 0$.

Note that (2.30) says, first, that the conditional probability of a state, given the past, depends only on the previous state, and second, that these transition probabilities $Q(s|s')$ do not change with time.

Definition: A *Markov source* is a sequence of discrete random symbols $\mathcal{X}_1, \mathcal{X}_2, \dots$ with a common alphabet $\mathcal{X}$ which is based on a finite-state Markov chain $S_0, S_1, \dots$. Each transition $(s' \to s)$ in the Markov chain is labeled with a symbol from $\mathcal{X}$; each symbol from $\mathcal{X}$ can appear on at most one outgoing transition from each state.

Note that the state alphabet $\mathcal{S}$ and the source alphabet $\mathcal{X}$ are in general different. Since each source symbol appears on at most one transition from each state, the initial state $S_0{=}s_0$, combined with the source output, $X_1{=}x_1, X_2{=}x_2, \dots$, uniquely identifies the state sequence, and, of course, the state sequence uniquely specifies the source output sequence. If $x \in \mathcal{X}$ labels the transition $s' \to s$, then the conditional probability of that x is given by $P(x|\, s') = Q(s|\, s')$. Thus, for example, in the transition $[00] \to [0]1$ in Figure 2.16, $Q([01]|\, [00]) = P(1|\, [00])$.

The reason for distinguishing the Markov chain alphabet from the source output alphabet is to allow the state to represent an arbitrary combination of past events rather than just the previous source output. It is this feature that permits Markov source models to reasonably model both simple and complex forms of memory.

A state s is *accessible* from state s' in a Markov chain if there is a path in the corresponding graph from $s' \to s$, *i.e.*, if $\Pr(S_n{=}s|\, S_0{=}s') > 0$ for some $n > 0$. The period of a state s is the greatest common divisor of the set of integers $n \geq 1$ for which $\Pr(S_n{=}s|\, S_0{=}s) > 0$. A finite-state Markov chain is *ergodic* if all states are accessible from all other states and if all states are aperiodic, *i.e.*, have period 1.

We will consider only Markov sources for which the Markov chain is ergodic. An important fact about ergodic Markov chains is that the chain has steady-state probabilities $q(s)$ for all $s \in \mathcal{S}$, given by the unique solution to the linear equations

$$\begin{aligned} q(s) &= \sum_{s'\in\mathcal{S}} q(s')Q(s|\,s'); \quad s\in\mathcal{S} \\ \sum_{s\in\mathcal{S}} q(s) &= 1. \end{aligned} \tag{2.31}$$

These steady-state probabilities are approached asymptotically from any starting state, *i.e.*,

$$\lim_{n\to\infty} \Pr(S_n{=}s|\,S_0{=}s') = q(s) \qquad \text{for all } s, s'\in\mathcal{S}. \tag{2.32}$$

2.8.1 Coding for Markov sources

The simplest approach to coding for Markov sources is that of using a separate prefix-free code for each state in the underlying Markov chain. That is, for each $s\in\mathcal{S}$, select a prefix-free code whose lengths $l(x,s)$ are appropriate for the conditional pmf $P(x|\,s) > 0$. The codeword lengths for the code used in state s must of course satisfy the Kraft inequality $\sum_x 2^{-l(x,s)} \le 1$. The minimum expected length, $\overline{L}_{\min}(s)$ for each such code can be generated by the Huffman algorithm and satisfies

$$\mathsf{H}[X|\,s] \le \overline{L}_{\min}(s) < \mathsf{H}[X|\,s] + 1. \tag{2.33}$$

where, for each $s\in\mathcal{S}$, $\mathsf{H}[X|\,s] = \sum_x -P(x|\,s)\log P(x|\,s)$.

If the initial state S_0 is chosen according to the steady-state pmf $\{q(s); s\in\mathcal{S}\}$, then, from (2.31), the Markov chain remains in steady state and the overall expected codeword length is given by

$$\mathsf{H}[X|\,S] \le \overline{L}_{\min} < \mathsf{H}[X|\,S] + 1, \tag{2.34}$$

where

$$\overline{L}_{\min} = \sum_{s\in\mathcal{S}} q(s)\overline{L}_{\min}(s) \qquad \text{and} \tag{2.35}$$

$$\mathsf{H}[X|\,S] = \sum_{s\in\mathcal{S}} q(s)\mathsf{H}[X|\,s]. \tag{2.36}$$

Assume that the encoder transmits the initial state s_0 at time 0. If M' is the number of elements in the state space, then this can be done with $\lceil \log M' \rceil$ bits, but this can be ignored since it is done only at the beginning of transmission and does not affect the long term expected number of bits per source symbol. The encoder then successively encodes each source symbol x_n using the code for the state at time $n-1$. The decoder, after decoding the initial state s_0, can decode x_1 using the code based on state s_0. The decoder can then determine the state s_1, and from that can decode x_2 using the code based on s_1. The decoder can continue decoding each source symbol, and thus the overall code is uniquely decodable. We next must understand the meaning of the conditional entropy in (2.36).

2.8.2 Conditional entropy

It turns out that the conditional entropy $\mathsf{H}[X|\,S]$ plays the same role in coding for Markov sources as the ordinary entropy $\mathsf{H}[X]$ plays for the memoryless case. Rewriting (2.35),

$$\mathsf{H}[X|\,S] = \sum_{s\in\mathcal{S}}\sum_{x\in\mathcal{X}} q(s)P(x|\,s)\log\frac{1}{P(x|\,s)}.$$

This is the expected value of the rv $\log[1/P(X|\,S)]$.

An important entropy relation, for any discrete rv's, is

$$\mathsf{H}[XS] = \mathsf{H}[S] + \mathsf{H}[X|\,S]. \tag{2.37}$$

To see this,

$$\begin{aligned}
\mathsf{H}[XS] &= \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{q(s)P(x|\,s)} \\
&= \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{q(s)} + \sum_{s,x} q(s)P(x|\,s)\log\frac{1}{P(x|\,s)} \\
&= \mathsf{H}[S] + \mathsf{H}[X|\,S].
\end{aligned}$$

Exercise 2.19 demonstrates that

$$\mathsf{H}[XS] \le \mathsf{H}[S] + \mathsf{H}[X]$$

Comparing this and (2.37), it follows that

$$\mathsf{H}[X|\,S] \le \mathsf{H}[X]. \tag{2.38}$$

This is an important inequality in information theory. If the entropy $\mathsf{H}[X]$ as a measure of mean uncertainty, then the conditional entropy $\mathsf{H}[X|\,S]$ should be viewed as a measure of mean uncertainty after the observation of the outcome of S. If X and S are not statistically independent, then intuition suggests that the observation of S should reduce the mean uncertainty in X; this equation indeed verifies this.

Example: Consider Figure 2.16 again. It is clear from symmetry that, in steady state, $p_X(0) = p_X(1) = 1/2$. Thus $\mathsf{H}[X] = 1$ bit. Conditional on S=00, X is binary with pmf {0.1, 0.9}, so $\mathsf{H}[X|\,[00]] = -0.1\log 0.1 - 0.9\log 0.9 = 0.47$ bits. Similarly, $\mathsf{H}[X|\,[11]] = 0.47$ bits, and $\mathsf{H}[X|\,[01]] = \mathsf{H}[X|\,[10]] = 1$ bit. The solution to the steady-state equations in (2.31) is $q([00]) = q([11]) = 5/12$ and $q([01]) = q([10]) = 1/12$. Thus, the conditional entropy, averaged over the states, is $\mathsf{H}[X|\,S] = 0.558$ bits.

For this example, it is particularly silly to use a different prefix-free code for the source output for each prior state. The problem is that the source is binary, and thus the prefix-free code will have length 1 for each symbol no matter what the state. As with the memoryless case, however, the use of fixed-to-variable-length codes is a solution to these problems of small alphabet sizes and integer constraints on codeword lengths.

Let $\mathsf{E}[L(\boldsymbol{X}^n)]_{\mathrm{min,s}}$ be the minimum expected length of a prefix-free code for $\boldsymbol{X}^n$ conditional on starting in state s. Then, applying (2.13) to the situation here,

$$\mathsf{H}[\boldsymbol{X}^n \mid s] \le \mathsf{E}[L(\boldsymbol{X}^n)]_{\mathrm{min},s} < \mathsf{H}[\boldsymbol{X}^n \mid s] + 1.$$

Assume as before that the Markov chain starts in steady state S_0. Thus it remains in steady state at each future time. Furthermore assume that the initial *sample state* is known at the decoder. Then the sample state continues to be known at each future time. Using a minimum expected length code for each initial sample state,

$$\mathsf{H}[\boldsymbol{X}^n \mid S_0] \le \mathsf{E}[L(\boldsymbol{X}^n)]_{\mathrm{min},S_0} < \mathsf{H}[\boldsymbol{X}^n \mid S_0] + 1. \tag{2.39}$$

Since the Markov source remains in steady state, the average entropy of each source symbol given the state is $H(X \mid S_0)$, so intuition suggests (and Exercise 2.32 verifies) that

$$\mathsf{H}[\boldsymbol{X}^n \mid S_0] = n\mathsf{H}[X| S_0]. \tag{2.40}$$

Defining $\overline{L}_{\min,n} = \mathsf{E}[L(\boldsymbol{X}^n)]_{\min,S_0}/n$ as the minimum expected codeword length per input symbol when starting in steady state,

$$\mathsf{H}[X| S_0] \leq \overline{L}_{\min,n} < \mathsf{H}[X| S_0] + 1/n. \tag{2.41}$$

The asymptotic equipartition property (AEP) also holds for Markov sources. Here, however, there are[19] approximately $2^{n\mathsf{H}[X| S]}$ typical strings of length n, each with probability approximately equal to $2^{-n\mathsf{H}[X| S]}$. It follows as in the memoryless case that $\mathsf{H}[X| S]$ is the minimum possible rate at which source symbols can be encoded subject either to unique decodability or to fixed-to-fixed-length encoding with small probability of failure. The arguments are essentially the same as in the memoryless case.

The analysis of Markov sources will not be carried further here, since the additional required ideas are minor modifications of the memoryless case. Curiously, most of our insights and understanding about souce coding come from memoryless sources. At the same time, however, most sources of practical importance can be insightfully modeled as Markov and hardly any can be reasonably modeled as memoryless. In dealing with practical sources, we combine the insights from the memoryless case with modifications suggested by Markov memory.

The AEP can be generalized to a still more general class of discrete sources called *ergodic sources*. These are essentially sources for which sample time averages converge in some probabilistic sense to ensemble averages. We do not have the machinery to define ergodicity, and the additional insight that would arise from studying the AEP for this class would consist primarily of mathematical refinements.

2.9 Lempel-Ziv universal data compression

The Lempel-Ziv data compression algorithms differ from the source coding algorithms studied in previous sections in the following ways:

- They use variable-to-variable-length codes in which both the number of source symbols encoded and the number of encoded bits per codeword are variable. Moreover, the codes are time-varying.
- They do not require prior knowledge of the source statistics, yet over time they adapt so that the average codeword length $\overline{L}$ per source symbol is minimized in some sense to be discussed later. Such algorithms are called *universal*.
- They have been widely used in practice; they provide a simple approach to understanding universal data compression even though newer schemes now exist.

The Lempel-Ziv compression algorithms were developed in 1977-78. The first, LZ77 [23], uses string-matching on a sliding window; the second, LZ78 [24], uses an adaptive dictionary. LZ78

[19]There are additional details here about whether the typical sequences include the initial state or not, but these differences become unimportant as n becomes large.

was implemented many years ago in the UNIX `compress` algorithm, and in many other places. Implementations of LZ77 appeared somewhat later (Stac Stacker, Microsoft Windows) and is still widely used.

In this section, the LZ77 algorithm is described. accompanied by a high-level description of why it works. Finally, an approximate analysis of its performance on Markov sources is given, showing that it is effectively optimal.[20] In other words, although this algorithm operates in ignorance of the source statistics, it compresses substantially as well as the best algorithm designed to work with those statistics.

2.9.1 The LZ77 algorithm

The LZ77 algorithm compresses a sequence $\boldsymbol{x} = x_1, x_2, \dots$ from some given discrete alphabet $\mathcal{X}$ of size $M = |\mathcal{X}|$. At this point, no probabilistic model is assumed for the source, so $\boldsymbol{x}$ is simply a sequence of symbols, not a sequence of random symbols. A subsequence $(x_m, x_{m+1}, \dots , x_n)$ of $\boldsymbol{x}$ is representedby $\boldsymbol{x}_m^n$.

The algorithm keeps the w most recently encoded source symbols in memory. This is called a sliding window of size w. The number w is large, and can be thought of as being in the range of 2^{10} to 2^{20}, say. The parameter w is chosen to be a power of 2. Both complexity and, typically, performance increase with w.

Briefly, the algorithm operates as follows. Suppose that at some time the source symbols $\boldsymbol{x}_1^P$ have been encoded. The encoder looks for the longest match, say of length n, between the not-yet-encoded n-string $\boldsymbol{x}_{P+1}^{P+n}$ and a stored string $\boldsymbol{x}_{P+1-u}^{P+n-u}$ starting in the window of length w. The clever algorithmic idea in LZ77 is to encode this string of n symbols simply by encoding the integers n and u; *i.e.*, by pointing to the previous occurrence of this string in the sliding window. If the decoder maintains an identical window, then it can look up the string $\boldsymbol{x}_{P+1-u}^{P+n-u}$, decode it, and keep up with the encoder.

More precisely, the LZ77 algorithm operates as follows:

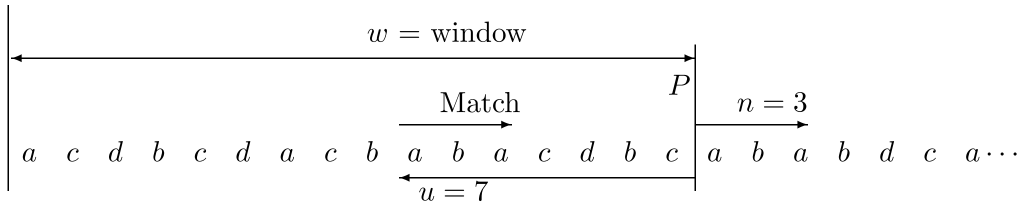
(1) Encode the first w symbols in a fixed-length code without compression, using $\lceil \log M \rceil$ bits per symbol. (Since $w\lceil \log M \rceil$ will be a vanishing fraction of the total number of encoded bits, the efficiency of encoding this preamble is unimportant, at least in theory.)

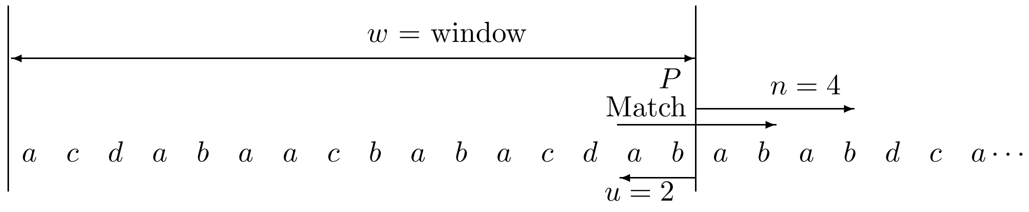
(2) Set the pointer $P = w$. (This indicates that all symbols up to and including x_P have been encoded.)

(3) Find the largest $n \geq 2$ such that $\boldsymbol{x}_{P+1}^{P+n} = \boldsymbol{x}_{P+1-u}^{P+n-u}$ for some u in the range $1 \leq u \leq w$. (Find the longest match between the not-yet-encoded symbols starting at $P + 1$ and a string of symbols starting in the window; let n be the length of that longest match and u the distance back into the window to the start of that match.) The string $\boldsymbol{x}_{P+1}^{P+n}$ is encoded by encoding the integers n and u.)

Here are two examples of finding this longest match. In the first, the length of the match is $n = 3$ and the match starts $u = 7$ symbols before the pointer. In the second, the length of the match is 4 and it starts $u = 2$ symbols before the pointer. Tis illustrates that that the string and its match can overlap.

[20]A proof of this optimality for discrete ergodic sources has been given by Wyner and Ziv [22].


w = window
P
Match
$n = 3$
a c d b c d a c b a b a c d b c a b a b d c $a\cdots$
$u = 7$


w = window
P
Match
$n = 4$
a c d a b a a c b a b a c d a b a b a b d c $a\cdots$
$u = 2$

If no match exists for $n \geq 2$, then, independently of whether a match exists for $n = 1$, set $n = 1$ and directly encode the single source symbol x_{P+1} without compression.

(4) Encode the integer n into a codeword from the unary-binary code. In the unary-binary code, a positive integer n is encoded into the binary representation of n, preceded by a prefix of $\lfloor \log_2 n \rfloor$ zeroes; *i.e.*,

n	prefix	base 2 exp.	codeword
1		1	1
2	0	10	010
3	0	11	011
4	00	100	00100
5	00	101	00101
6	00	110	00110
7	00	111	00111
8	000	1000	0001000

Thus the codewords starting with $0^k 1$ correspond to the set of 2^k integers in the range $2^k \leq n \leq 2^{k+1} - 1$. This code is prefix-free (picture the corresponding binary tree). It can be seen that the codeword for integer n has length $2\lfloor \log n \rfloor + 1$; it is seen later that this is negligible compared with the length of the encoding for u.

(5) If $n > 1$, encode the positive integer $u \leq w$ using a fixed-length code of length $\log w$ bits. (At this point the decoder knows n, and can simply count back by u in the previously decoded string to find the appropriate n-tuple, even if there is overlap as above.)

(6) Set the pointer P to $P + n$ and go to step (3). (Iterate forever.)

2.9.2 Why LZ77 works

The motivation behind LZ77 is information-theoretic. The underlying idea is that if the unknown source happens to be, say, a Markov source of entropy $\mathsf{H}(X|S)$, then the AEP says that, for any large n, there are roughly $2^{n\mathsf{H}[X|S]}$ typical source strings of length n. On the other hand, a window of size w contains w source strings of length n, counting duplications. This means that if $w \ll 2^{n\mathsf{H}[X|S]}$, then most typical sequences of length n cannot be found in the window, suggesting that matches of length n are unlikely. Similarly, if $w \gg 2^{n\mathsf{H}[X|S]}$, then it is reasonable to suspect that most typical sequences will be in the window, suggesting that matches of length n or more are likely.

The above argument, approximate and vague as it is, suggests that when n is large and w is truly humongous, the typical size of match n_t satisfies $w \approx 2^{n_t\mathsf{H}(X|S)}$, which really means

$$n_t \approx \frac{\log w}{\mathsf{H}(X|S)}; \qquad \text{typical match size.} \tag{2.42}$$

The encoding for a match requires $\log w$ bits for the match location and $2\lfloor \log n_t \rfloor + 1$ for the match size n_t. Since n_t is proportional to $\log w$, $\log n_t$ is negligible compared to $\log w$ for very large w. Thus, for the typical case, about $\log w$ bits are used to encode about n_t source symbols. Thus, from (2.42), the required rate, in bits per source symbol, is about $\overline{L} \approx \mathsf{H}(X|S)$.

The above argument is very imprecise, but the conclusion is that, for very large window size, $\overline{L}$ is reduced to the value required when the source is known and an optimal fixed-to-variable prefix-free code is used.

The imprecision above involves more than simply ignoring the approximation factors in the AEP. A more conceptual issue is that the strings of source symbols that must be encoded are somewhat special since they start at the end of previous matches. The other conceptual difficulty comes from ignoring the duplications of typical sequences within the window.

This argument has been made precise Wyner and Ziv [22].

2.9.3 Discussion

Let us recapitulate the basic ideas behind the LZ77 algorithm:

(1) Let N_x be the number of occurrences of symbol x in a window of size w. The WLLN asserts that the relative frequency N_x/w of appearances of x in the window will satisfy $N_x/w \approx p_X(x)$ with high probability. Similarly, let $N_{\boldsymbol{x}^n}$ be the number of occurrences of $\boldsymbol{x}^n$ which start in the window. The relative frequency $N_{\boldsymbol{x}^n}/w$ will then satisfy $N_{\boldsymbol{x}^n}/w \approx p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ with high probability for very large w. This association of relative frequencies with probabilities is what makes LZ77 a universal algorithm which needs no prior knowledge of source statistics.[21]

(2) Next, as explained in the previous section, the probability of a typical source string $\boldsymbol{x}^n$ for a Markov source is approximately $2^{-n\mathsf{H}[X|S]}$. If $w >> 2^{n\mathsf{H}[X|S]}$, then, according to the previous item, $N_{\boldsymbol{x}^n} \approx w p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)$ should be large and $\boldsymbol{x}^n$ should occur in the window with high probability. Alternatively, if $w << 2^{n\mathsf{H}[X|S]}$, then $\boldsymbol{x}^n$ will probably not occur. Consequently the match will usually occur for $n \approx (\log w)/\mathsf{H}[X|S]$ as w becomes very large.

(3) Finally, it takes about $\log w$ bits to point to the best match in the window. The unary-binary code uses $2\lfloor \log n \rfloor + 1$ bits to encode the length n of the match. For typical n, this is on the order of $2\log(\log w/\mathsf{H}[X|S])$ which is negigible for large enough w compared to $\log w$.

[21] As Yogi Berra said, "You can observe a whole lot just by watchin'."

Consequently, LZ77 requires about $\log w$ encoded bits for each group of about $(\log w)/\mathsf{H}[X|\,S]$ source symbols, so it nearly achieves the optimal efficiency of $\overline{L} = \mathsf{H}[X|\,S]$ bits/symbol, as w becomes very large.

Discrete sources, as they appear in practice, often can be viewed over different time scales. Over very long time scales, or over the sequences presented to different physical encoders running the same algorithm, there is often very little common structure, sometimes varying from one language to another, or varying from text in a language to data from something else.

Over shorter time frames, corresponding to a single file or a single application type, there is often more structure, such as that in similar types of documents from the same language. Here it is more reasonable to view the source output as a finite length segment of, say, the output of an ergodic Markov source.

What this means is that universal data compression algorithms must be tested in practice. The fact that they behave optimally for unknown sources that can be modeled to satisfy the AEP is an important guide, but not the whole story.

The above view of different time scales also indicates that a larger window need not always improve the performance of the LZ77 algorithm. It suggests that long matches will be more likely in recent portions of the window, so that fixed length encoding of the window position is not the best approach. If shorter codewords are used for more recent matches, then it requires a shorter time for efficient coding to start to occur when the source statistics abruptly change. It also then makes sense to start coding from some arbitrary window known to both encoder and decoder rather than filling the entire window with data before starting to use the LZ77 alogorithm.

2.10 Summary of discrete source coding

Discrete source coding is important both for discrete sources such as text and computer files and also as an inner layer for discrete-time analog sequences and fully analog sources. It is essential to focus on the range of possible outputs from the source rather than any one particular output. It is also important to focus on *probabilistic* models so as to achieve the best compression for the most common outputs with less care for very rare outputs. Even universal coding techniques, such as LZ77, which are designed to work well in the absence of a probability model, require probability models to understand and evaluate how they work.

Variable-length source coding is the simplest way to provide good compression for common source outputs at the expense of rare outputs. The necessity to concatenate successive variable-length codewords leads to the non-probabilistic concept of unique decodability. Prefix-free codes provide a simple class of uniquely-decodable codes. Both prefix-free codes and the more general class of uniquely-decodable codes satisfy the Kraft inequality on the number of possible code words of each length. Moreover, for any set of lengths satisfying the Kraft inequality, there is a simple procedure for constructing a prefix-free code with those lengths. Since the expected length, and other important properties of codes, depend only on the codewords lengths (and how they are assigned to source symbols), there is usually little reason to use variable-length codes that are not also prefix free.

For a DMC with given probabilities on the symbols of a source code, the entropy is a lower bound on the expected length of uniquely decodable codes. The Huffman algorithm provides a simple procedure for finding an optimal (in the sense of minimum expected codeword length) variable-length prefix-free code. The Huffman algorithm is also useful for deriving properties about optimal variable length source codes (see Exercises 2.12 to 2.18).

All the properties of variable-length codes extend immediately to fixed-to-variable-length codes in which the source output sequence is segmented into blocks of n symbols which are then encoded as a single symbol from the alphabet of source n-tuples. For a DMC the minimum expected codeword length per source symbol then lies between $\mathsf{H}(U)$ and $\mathsf{H}(U) + 1/n$. Thus prefix-free fixed-to-variable-length codes can approach the entropy bound as closely as desired.

One of the disadvantages of fixed-to-variable-length codes is that bits leave the encoder at a variable rate relative to incoming symbols. Thus if the incoming symbols have a fixed rate and the bits must be fed into a channel at a fixed rate (perhaps with some idle periods), then the encoded bits must be queued and there is a positive probability that any finite length queue will overflow.

An alternative point of view is to consider fixed-length to fixed-length codes. Here, for a DMC, the set of possible n-tuples of symbols from the source can be partitioned into a typical set and an atypical set. For large n, the AEP says that there are essentially $2^{n\mathsf{H}(U)}$ typical n-tuples with an aggregate probability approaching 1 with increasing n. Encoding just the typical n-tuples requires about $\mathsf{H}(U)$ bits per symbol, thus approaching the entropy bound without the above queueing problem, but, of course, with occasional errors.

As detailed in the text, the AEP can be used to look at the long-term behavior of arbitrary source coding algorithms to show that the entropy bound cannot be exceeded without a failure rate that approaches 1.

The above results for discrete memoryless sources extend easily to ergodic Markov sources. The text does not carry out this analysis in detail since readers are not assumed to have the requisite knowledge about Markov chains (see [4] for the detailed analysis). The important thing here is to see that Markov sources can model n-gram statistics for any desired n and thus can model fairly general sources (at the cost of very complex models). From a practical standpoint, universal source codes, such as LZ77 are usually a more reasonable approach to complex and partly unknown sources.

2A Appendix: Review of probability

The reader is assumed to have a basic familiarity with probability theory at the undergraduate level. Many students, however, become adept at solving well-posed probability problems without thinking much about the relation of probabilistic models to reality. This appendix provides a very rudimentary review of this relationship and also indicates some of the terminology used here; Students who are rusty (or who have difficulty with Exercises 2.2 to 2.5 should do some further review of undergraduate probability. Suggested texts are [2] or [15].

A probability model, or probability space, consists of a sample space and a probability measure on that space. The sample space is an arbitrary set of elements called *sample points* or *sample outcomes*. Subsets of the sample space are called[22] *events*.

The intuitive notion is that an 'experiment' is performed. Prior to performing the experiment,

[22]Mathematically, only measurable subsets of the sample space are called events. This is an important mathematical issue, but it can be safely ignore here.

any sample outcome could occur, and after the experiment, one and only one of these sample outcomes occurs. This sample outcome specifies *everything* about the result of the experiment. The occurrence of an event specifies that some sample outcome within that event occurred. For example, for discrete sources, the sample point is the actual sequence of symbols that come out of the source. An example of an event is that the first symbol in the output sequence is the letter a. This event can then be thought of as the set of output sequences (sample points) that start with a. Note that the technical meanings of sample outcome and event are not quite the same as their ordinary usage in English. A sample outcome completely determines the result of the experiment, whereas an event partially determines the result, specifying only that the sample outcome lies in a given subset of the sample space.

A probability measure is a rule for assigning probabilities to each of the events. These probabilities lie between 0 and 1 and the probability of the sample space itself is 1. The probability of a union of disjoint events is the sum of the probabilities of those events; this is called the *union rule*. For example, consider a probability model where the sample points are strings of length n from a discrete source. The probability measure can be viewed as assigning a probability to each such sample point, which, through the union rule, determines a probability for each event.

For the example of an unending sequence of outputs from a discrete source, the situation is more complicated. Each sample point (each sequence of particular source outputs) will typically have probability 0, so probabilities must be assigned directly to more complex events (such as the event that the first output symbol in the sequence is the letter a). The description of discrete memoryless sources in Section 2.4.1 is an example of how a probability measure can be formulated for this sample space.

For a discrete source, the output at any given time n is denoted by X_n. This source output can be any symbol a_j in the alphabet $\mathcal{X} = \{a_1, \dots, a_M\}$, and for each $a_j \in \mathcal{X}$, there is an event $X_n = a_j$ which is the set of all output sequences for which the nth output is the particular symbol a_j. For each $x \in \mathcal{X}$, this event has a probability denoted by $\Pr(X_n = x)$ which is abbreviated $p_{X_n}(x)$. Clearly $\sum_{x \in \mathcal{X}} p_{X_n}(x) = 1$. X_n is called a random symbol. In general, a random symbol[23] is a mapping from the sample space to the elements of some set $\mathcal{X}$, and the elements of $\mathcal{X}$ are called the possible outcomes for X_n.

One might ask why all this machinery is required to talk about a random symbol, when it might appear that all we need is the probability mass function (pmf) $p_{X_n}(x)$ defined for all possible outcomes $x \in \mathcal{X}$. The reason is that one is often interested not only in which event $X_n = x$ occurs for a given random symbol X_n, but also in the joint probabilities between different random symbols. The definition here implicitly defines all possible joint probabilities.

A *random variable* (rv) W is a random symbol in which the mapping is from the sample space to a set $\mathcal{W}$ of real numbers, *i.e.*, $\mathcal{W} \subseteq \mathbb{R}$. All rv's have a *distribution function*, $\Pr(W \leq w)$ which gives a complete probabilistic characterization of the rv in isolation from other rv's. That is, $\Pr(W \leq w)$ does not specify how W relates statistically to other rv's, but in principle it determines all the probabilistic properties of W itself.

If $\mathcal{W}$ contains a finite (or countably infinite) number of possible values, then W is said to be a *discrete rv* and the probability assignment on the possible outcomes of W can be given by a *probability mass function* (pmf) $p_W(w) = \Pr(W{=}w)$ for all $w \in \mathcal{W}$. In this case, the distribution function is a staircase function with a jump $p_W(w)$ at each possible outcome w.

[23]As defined in this generality, random variables, random vectors, etc. can all be viewed as special cases of random symbols. Random symbols as used in the text are usually discrete however.

If, on the other hand, the distribution function of W is continuous and differentiable, then W is said to be analog.[24] The distribution of W is then specified by the *probability density function* (pdf) $f_W(w) = d\,\Pr(W \le w)/dw$. Some rv's are neither discrete nor analog according to these definitions, but such rv's rarely appear in sensible models of communcation.

In this text, random symbols are always denoted by capital letters and the outcomes for those variables by lower-case letters. The most common source of errors in working with probability arises from confusing random symbols or rv's with their outcomes.

A random symbol X is *deterministic* or *nonrandom* if $p_X(x) = 1$ for some $x \in \mathcal{X}$, which necessarily implies that $p_X(x) = 0$ for all other $x \in \mathcal{X}$. A discrete random symbol X is *equiprobable* or *uniform* if $p_X(x) = 1/M$ for all $x \in \mathcal{X}$, where $M = |\mathcal{X}|$. Intuitively, these are respectively the least random and most random discrete random symbols taking values in $\mathcal{X}$.

The *expectation* or *mean* or *average value* of a discrete or analog rv W is defined as

$$\mathsf{E}[W] = \sum_{w\in\mathcal{W}} p_W(w)w \quad \text{(discrete)} \qquad \mathsf{E}[W] = \int_w f_W(w)w\,dw \quad \text{(analog)}$$

Expectations are often denoted with an overbar, *i.e.*, $\mathsf{E}[W] = \overline{W}$. Often an rv W is defined in terms of a discrete random symbol X (*i.e.*, the outcome w for W is uniquely determined as a function $w(x)$ of the outcome for X). Such an rv is often denoted as $W(X)$. Thus, if X is discrete, this leads to $\mathsf{E}[W(X)] = \sum_{x\in\mathcal{X}} p_X(x)w(x)$.

Expectations are additive: $\mathsf{E}[W_1 + W_2] = \mathsf{E}[W_1] + \mathsf{E}[W_2]$.

The Cartesian product $\mathcal{X} \times \mathcal{Y} = \{(x, y) \mid x \in \mathcal{X}, y \in \mathcal{Y}\}$ of two discrete alphabets $\mathcal{X}$ and $\mathcal{Y}$ is another discrete alphabet. If, in some probability space, X and Y are random symbols taking values in $\mathcal{X}$ and $\mathcal{Y}$ respectively, then XY is another random symbol whose pmf $p_{XY}(x, y)$ is defined for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. Such a pmf is called a *joint pmf.* The pmf of X is related to that for XY by

$$p_X(x) = \sum_{y\in\mathcal{Y}} p_{XY}(x, y).$$

For each $x \in \mathcal{X}$ with $p_X(x) > 0$, the *conditional pmf* of Y is defined by

$$p_{Y|X}(y \mid x) = \frac{p_{XY}(x, y)}{p_X(x)}.$$

For each $x \in \mathcal{X}$, this is a 'valid' pmf on $\mathcal{Y}$ in the sense that its elements are nonnegative and sum to 1. The true pmf on Y is often called the marginal pmf to avoid confusion with these conditional pmf's.

Analog rv's can be treated the same way. The joint density is given in terms of the joint distribution function as $f_{XY}(x, y) = \frac{\partial}{\partial(x,y)} \Pr(X \le x, Y \le y)$. The marginal density for X is then given by $f_X(x) = \int_y f_{XY}(x, y)$. Similarly, the conditional density is given by $f_{X|Y}(x|y) = \frac{f_{X|Y}(xy)}{f_Y(y)}$. There are many mathematical subtleties that can arise with conditional densities, but we will treat these matters with common sense rather than developing the (quite extensive) needed mathematical structure.

Two discrete random symbols X and Y are *independent* if their joint pmf factors into the product of the marginals: $p_{XY}(x, y) = p_X(x)p_Y(y)$ for each $x \in \mathcal{X}, y \in \mathcal{Y}$. Equivalently, the

[24] Analog rv's are often called continuous rv's; this notation is avoided here since it sounds precise, but is not, since it is the differentiability of the distribution function that is important.

conditional pmf for Y (resp. X) is not a function of x (resp. y): $p_{Y|X}(y \mid x) = p_Y(y)$ (resp. $p_{X|Y}(x \mid y) = p_X(x)$) for each $x \in \mathcal{X}$, $y \in \mathcal{Y}$. Similarly, two analog rv's are independent if $f_{XY}(x, y) = f_X(x)f_Y(y)$ for each $x \in \mathcal{X}, y \in \mathcal{Y}$.

An important property of expectations is that the expectation of the product of two *independent* rv's is the product of their expectations:

$$\mathsf{E}[W_1W_2] = \mathsf{E}[W_1] \cdot \mathsf{E}[W_2].$$

The *nth moment* of an rv W is defined as $\mathsf{E}[W^n]$. Note that W^n is an rv in its own right, since, for each sample point, the outcome w for W specifies the outcome w^n for W^n.

The first moment of W is its *mean* $\overline{W} = \mathsf{E}[W]$. The *fluctuation* of W is the zero-mean rv $W - \overline{W}$. The *variance* of W is the second moment of its fluctuation:

$$\sigma_W^2 = \mathsf{E}[(W - \overline{W})^2] = \mathsf{E}[W^2] - \overline{W}^2.$$

Evidently the variance of W is nonnegative, and is equal to 0 if and only if W is deterministic *i.e.*, its fluctuation is equal to 0 with probability 1.

2.E Exercises

2.1. Chapter 1 pointed out that voice waveforms could be converted to binary data by sampling at 8000 times per second and quantizing to 8 bits per sample, yielding 64kb/s. It then said that modern speech coders can yield telephone-quality speech at 6-16 kb/s. If your objective were simply to reproduce the words in speech recognizably without concern for speaker recognition, intonation, etc., make an estimate of how many kb/s would be required. Explain your reasoning. (Note: There is clearly no "correct answer" here; the question is too vague for that. The point of the question is to get used to questioning objectives and approaches.)

2.2. Let V and W be discrete rv's defined on some probability space with a joint pmf $p_{VW}(v,w)$.

(a) Prove that $\mathsf{E}[V+W] = \mathsf{E}[V] + \mathsf{E}[W]$. Do not assume independence.

(b) Prove that if V and W are independent rv's, then $\mathsf{E}[V \cdot W] = \mathsf{E}[V] \cdot \mathsf{E}[W]$.

c) Assume that V and W are not independent. Find an example where $\mathsf{E}[V \cdot W] \neq \mathsf{E}[V] \cdot \mathsf{E}[W]$ and another example where $\mathsf{E}[V \cdot W] = \mathsf{E}[V] \cdot \mathsf{E}[W]$.

d) Assume that V and W are independent and let σ_V^2 and σ_W^2 be the variances of V and W respectively. Show that the variance of $V+W$ is given by $\sigma_{V+W}^2 = \sigma_V^2 + \sigma_W^2$.

2.3. (a) For a nonnegative integer-valued rv N, show that $\mathsf{E}[N] = \sum_{n>0} \Pr(N \geq n)$.

(b) Show, with whatever mathematical care you feel comfortable with, that for an arbitrary nonnegative rv X that $\mathsf{E}(X) = \int_0^\infty \Pr(X \geq a)da$.

(c) Derive the Markov inequality, which says that for any nonnegative rv, $\Pr(X \geq a) \leq \frac{\mathsf{E}[X]}{a}$. Hint: Sketch $\Pr(X > a)$ as a function of a and compare the area of the a by $\Pr(X \geq a)$ rectangle in your sketch with the area corresponding to $\mathsf{E}[X]$.

(d) Derive the Chebyshev inequality, which says that $\Pr(|Y - \mathsf{E}[Y]| \geq b) \leq \frac{\sigma_Y^2}{b^2}$ for any rv Y with finite mean $\mathsf{E}[Y]$ and finite variance σ_Y^2. Hint: Use part (c) with $(Y - \mathsf{E}[Y])^2 = X$.

2.4. Let $X_1, X_2, \ldots, X_n, \ldots$ be a sequence of independent identically distributed (iid) analog rv's with the common probability density function $f_X(x)$. Note that $\Pr\{X_n{=}\alpha\} = 0$ for all α and that $\Pr\{X_n{=}X_m\} = 0$ for $m \neq n$.

(a) Find $\Pr\{X_1 \leq X_2\}$. [Give a numerical answer, not an expression; no computation is required and a one or two line explanation should be adequate.]

(b) Find $\Pr\{X_1 \leq X_2; X_1 \leq X_3\}$ (in other words, find the probability that X_1 is the smallest of $\{X_1, X_2, X_3\}$). [Again, think— don't compute.]

(c) Let the rv N be the index of the first rv in the sequence to be less than X_1; that is, $\Pr\{N{=}n\} = \Pr\{X_1 \leq X_2; X_1 \leq X_3; \cdots; X_1 \leq X_{n-1}; X_1 > X_n\}$. Find $\Pr\{N \geq n\}$ as a function of n. Hint: generalize part (b).

(d) Show that $\mathsf{E}[N] = \infty$. Hint: use part (a) of Exercise 2.3.

(e) Now assume that $X_1, X_2 \ldots$ is a sequence of iid rv's each drawn from a finite set of values. Explain why you can't find $\Pr\{X_1 \leq X_2\}$ without knowing the pmf. Explain why $\mathsf{E}[N] = \infty$.

2.5. Let $X_1, X_2, \dots, X_n$ be a sequence of n binary iid rv's. Assume that $\Pr\{X_m{=}1\} = \Pr\{X_m{=}0\} = \frac{1}{2}$. Let Z be a parity check on $X_1, \dots, X_n$; that is, $Z = X_1 \oplus X_2 \oplus \cdots \oplus X_n$ (where $0 \oplus 0 = 1 \oplus 1 = 0$ and $0 \oplus 1 = 1 \oplus 0 = 1$).

(a) Is Z independent of X_1? (Assume $n > 1$.)

(b) Are $Z, X_1, \dots, X_{n-1}$ independent?

(c) Are $Z, X_1, \dots, X_n$ independent?

(d) Is Z independent of X_1 if $\Pr\{X_i{=}1\} \neq \frac{1}{2}$? You may take $n = 2$ here.

2.6. Define a suffix-free code as a code in which no codeword is a suffix of any other codeword.

(a) Show that suffix-free codes are uniquely decodable. Use the definition of unique decodability in Section 2.3.1, rather than the intuitive but vague idea of decodability with initial synchronization.

(b) Find an example of a suffix-free code with codeword lengths (1, 2, 2) that is not a prefix-free code. Can a codeword be decoded as soon as its last bit arrives at the decoder? Show that a decoder might have to wait for an arbitrarily long time before decoding (this is why a careful definition of unique decodability is required).

(c) Is there a code wih codeword lengths (1, 2, 2) that is both prefix-free and suffix-free? Explain.

2.7. The algorithm given in essence by (2.2) for constructing prefix-free codes from a set of codeword lengths uses the assumption the lengths have been ordered first. Give an example in which the algorithm fails if the lengths are not ordered first.

2.8. Suppose that, for some reason, you wish to encode a source into symbols from a D-ary alphabet (where D is some integer greater than 2) rather than into a binary alphabet. The development of Section 2.3 can be easily extended to the D-ary case, using D-ary trees rather than binary trees to represent prefix-free codes. Generalize the Kraft inequality, (2.1), to the D-ary case and outline why it is still valid.

2.9. Suppose a prefix-free code has symbol probabilities $p_1, p_2, \dots, p_M$ and lengths $l_1, \dots, l_M$. Suppose also that the expected length $\overline{L}$ satisfies $\overline{L} = \mathsf{H}(X)$.

(a) Explain why $p_i = 2^{-l_i}$ for each i.

(b) Explain why the sequence of encoded binary digits is a sequence of iid equiprobable binary digits. Hint: Use figure 2.4 to illustrate this phenomenon and explain in words why the result is true in general. Do not attempt a general proof.

2.10. (a) Show that in a code of M codewords satisfying the Kraft inequality with equality, the maximum length is at most $M - 1$. Explain why this ensures that the number of distinct such codes is finite.

(b) Consider the number $S(M)$ of distinct full code trees with M terminal nodes. Count two trees as being different if the corresponding set of codewords is different. That is, ignore the set of source symbols and the mapping between source symbols and codewords. Show that $S(2) = 1$ and show that for $M > 2$, $S(M) = \sum_{j=1}^{M-1} S(j)S(M-j)$ where $S(1) = 1$ by convention.

2.11. (Proof of the Kraft inequality for uniquely decodable codes) (a) Assume a uniquely decodable code has lengths $l_1, \dots, l_M$. In order to show that $\sum_j 2^{-l_j} \le 1$, demonstrate the following identity for each integer $n \ge 1$:

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n = \sum_{j_1=1}^{M}\sum_{j_2=1}^{M}\cdots\sum_{j_n=1}^{M} 2^{-(l_{j_1}+l_{j_2}+\cdots+l_{j_n})}$$

(b) Show that there is one term on the right for each concatenation of n codewords (*i.e.*, for the encoding of one n-tuple $\boldsymbol{x}^n$) where $l_{j_1} + l_{j_2} + \cdots + l_{j_n}$ is the aggregate length of that concatenation.

(c) Let A_i be the number of concatenations which have overall length i and show that

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n = \sum_{i=1}^{nl_{\max}} A_i\, 2^{-i}$$

(d) Using the unique decodability, upper bound each A_i and show that

$$\left[\sum_{j=1}^{M} 2^{-l_j}\right]^n \le nl_{\max}$$

(e) By taking the nth root and letting $n \to \infty$, demonstrate the Kraft inequality.

2.12. A source with an alphabet size of $M = |\mathcal{X}| = 4$ has symbol probabilities $\{1/3, 1/3, 2/9, 1/9\}$.

(a) Use the Huffman algorithm to find an optimal prefix-free code for this source.

(b) Use the Huffman algorithm to find another optimal prefix-free code with a different set of lengths.

(c) Find another prefix-free code that is optimal but cannot result from using the Huffman algorithm.

2.13. An alphabet of $M = 4$ symbols has probabilities $p_1 \ge p_2 \ge p_3 \ge p_4 > 0$.

(a) Show that if $p_1 = p_3 + p_4$, then a Huffman code exists with all lengths equal and another exists with a codeword of length 1, one of length 2, and two of length 3.

(b) Find the largest value of p_1, say $p_{\max}$, for which $p_1 = p_3 + p_4$ is possible.

(c) Find the smallest value of p_1, say $p_{\min}$, for which $p_1 = p_3 + p_4$ is possible.

(d) Show that if $p_1 > p_{\max}$, then every Huffman code has a length 1 codeword.

(e) Show that if $p_1 > p_{\max}$, then every optimal prefix-free code has a length 1 codeword.

(f) Show that if $p_1 < p_{\min}$, then all codewords have length 2 in every Huffman code.

(g) Suppose $M > 4$. Find the smallest value of $p'_{\max}$ such that $p_1 > p'_{\max}$ guarantees that a Huffman code will have a length 1 codeword.

2.14. Consider a source with M equiprobable symbols.

(a) Let $k = \lceil \log M \rceil$. Show that, for a Huffman code, the only possible codeword lengths are k and $k - 1$.

(b) As a function of M, find how many codewords have length $k = \lceil \log M \rceil$. What is the expected codeword length $\overline{L}$ in bits per source symbol?

(c) Define $y = M/2^k$. Express $\overline{L} - \log M$ as a function of y. Find the maximum value of this function over $1/2 < y \leq 1$. This illustrates that the entropy bound, $\overline{L} < \mathsf{H}[X] + 1$ is rather loose in this equiprobable case.

2.15. Let a discrete memoryless source have M symbols with alphabet $\{1, 2, \dots, M\}$ and ordered probabilities $p_1 > p_2 > \cdots > p_M > 0$. Assume also that $p_1 < p_{M-1} + p_M$. Let $l_1, l_2, \dots, l_M$ be the lengths of a prefix-free code of minimum expected length for such a source.

(a) Show that $l_1 \leq l_2 \leq \cdots \leq l_M$.

(b) Show that if the Huffman algorithm is used to generate the above code, then $l_M \leq l_1 + 1$. Hint: Look only at the first step of the algorithm.

(c) Show that $l_M \leq l_1 + 1$ whether or not the Huffman algorithm is used to generate a minimum expected length prefix-free code.

(d) Suppose $M = 2^k$ for integer k. Determine $l_1, \dots, l_M$.

(e) Suppose $2^k < M < 2^{k+1}$ for integer k. Determine $l_1, \dots, l_M$.

2.16. (a) Consider extending the Huffman procedure to codes with ternary symbols $\{0, 1, 2\}$. Think in terms of codewords as leaves of ternary trees. Assume an alphabet with $M = 4$ symbols. Note that you cannot draw a full ternary tree with 4 leaves. By starting with a tree of 3 leaves and extending the tree by converting leaves into intermediate nodes, show for what values of M it is possible to have a complete ternary tree.

(b) Explain how to generalize the Huffman procedure to ternary symbols bearing in mind your result in part (a).

(c) Use your algorithm for the set of probabilities $\{0.3, 0.2, 0.2, 0.1, 0.1, 0.1\}$.

2.17. Let X have M symbols, $\{1, 2, \dots, M\}$ with ordered probabilities $p_1 \geq p_2 \geq \cdots \geq p_M > 0$. Let X' be the reduced source after the first step of the Huffman algorithm.

(a) Express the entropy $\mathsf{H}(X)$ for the original source in terms of the entropy $\mathsf{H}(X')$ of the reduced source as

$$\mathsf{H}(X) = \mathsf{H}(X') + (p_M + p_{M-1})H(\gamma), \tag{2.43}$$

where $H(\gamma)$ is the binary entropy function, $H(\gamma) = -\gamma \log \gamma - (1-\gamma)\log(1-\gamma)$. Find the required value of γ to satisfy (2.43).

(b) In the code tree generated by the Huffman algorithm, let v_1 denote the intermediate node that is the parent of the leaf nodes for symbols M and $M-1$. Let $q_1 = p_M + p_{M-1}$ be the probability of reaching v_1 in the code tree. Similarly, let $v_2, v_3, \dots$, denote the subsequent intermediate nodes generated by the Huffman algorithm. How many intermediate nodes are there, including the root node of the entire tree?

(c) Let $q_1, q_2, \dots$, be the probabilities of reaching the intermediate nodes $v_1, v_2, \dots$, (note that the probability of reaching the root node is 1). Show that $\overline{L} = \sum_i q_i$. Hint: Note that $\overline{L} = \overline{L}' + q_1$.

(d) Express $\mathsf{H}(X)$ as a sum over the intermediate nodes. The ith term in the sum should involve q_i and the binary entropy $H(\gamma_i)$ for some γ_i to be determined. You may find it helpful to define α_i as the probability of moving upward from intermediate node v_i, conditional on reaching v_i. (Hint: look at part a).

(e) Find the conditions (in terms of the probabilities and binary entropies above) under which $\overline{L} = \mathsf{H}(X)$.

(f) Are the formulas for $\overline{L}$ and $\mathsf{H}(X)$ above specific to Huffman codes alone, or do they apply (with the modified intermediate node probabilities and entropies) to arbitrary full prefix-free codes?

2.18. Consider a discrete random symbol X with $M+1$ symbols for which $p_1 \geq p_2 \geq \cdots \geq p_M > 0$ and $p_{M+1} = 0$. Suppose that a prefix-free code is generated for X and that for some reason, this code contains a codeword for $M+1$ (suppose for example that p_{M+1} is actaully positive but so small that it is approximated as 0.

(a) Find $\overline{L}$ for the Huffman code including symbol $M+1$ in terms of $\overline{L}$ for the Huffman code omitting a codeword for symbol $M+1$.

(b) Suppose now that instead of one symbol of zero probability, there are n such symbols. Repeat part (a) for this case.

2.19. In (2.12), it is shown that if X and Y are independent discrete random symbols, then the entropy for the random symbol XY satisfies $\mathsf{H}[XY] = \mathsf{H}[X] + \mathsf{H}[Y]$. Here we want to show that, without the assumption of independence, we have $\mathsf{H}[XY] \leq \mathsf{H}[X] + \mathsf{H}[Y]$.

(a) Show that

$$\mathsf{H}[XY] - \mathsf{H}[X] - \mathsf{H}[Y] = \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} p_{XY}(x,y) \log \frac{p_X(x) p_Y(y)}{p_{X,Y}(x,y)}.$$

(b) Show that $\mathsf{H}[XY] - \mathsf{H}[X] - \mathsf{H}[Y] \leq 0$, *i.e.*, that $\mathsf{H}[XY] \leq \mathsf{H}[X] + \mathsf{H}[Y]$.

(c) Let $X_1, X_2, \dots, X_n$ be discrete random symbols, not necessarily independent. Use (b) to show that

$$H(X_1 X_2 \cdots X_n) \leq \sum_{j=1}^{n} H(X_j).$$

2.20. Consider a random symbol X with the symbol alphabet $\{1, 2, \dots, M\}$ and a pmf $\{p_1, p_2, \dots, p_M\}$. This exercise concerns the relationship between the entropy $\mathsf{H}(X)$ and the probability p_1 of the first symbol. Let Y be a random symbol that is 1 if $X = 1$ and 0 otherwise. For parts (a) through (d), consider M and p_1 to be fixed.

(a) Express $\mathsf{H}(Y)$ in terms of the binary entropy function, $H_b(\alpha) = -\alpha \log(\alpha) - (1-\alpha)\log(1-\alpha)$.

(b) What is the conditional entropy $\mathsf{H}(X|Y=1)$?

(c) Give a good upper bound to $\mathsf{H}(X|Y=0)$ and show how this bound can be met with equality by appropriate choice of $p_2, \dots, p_M$. Use this to upper bound $\mathsf{H}(X|Y)$.

(d) Give a good upper bound for $\mathsf{H}(X)$ and show that how this bound can be met with equality by appropriate choice of $p_2, \dots, p_M$.

(e) For the same value of M as before, let $p_1, \dots, p_M$ be arbitrary and let $p_{\max}$ be $\max\{p_1, \dots, p_M\}$. Is your upper bound in (d) still valid if you replace p_1 by $p_{\max}$? Explain.

2.21. A discrete memoryless source emits iid random symbols $X_1, X_2, \ldots$. Each random symbol X has the symbols $\{a, b, c\}$ with probabilities $\{0.5,\ 0.4,\ 0.1\}$, respectively.

(a) Find the expected length $\overline{L}_{\min}$ of the best variable-length prefix-free code for X.

(b) Find the expected length $\overline{L}_{\min,2}$, normalized to bits per symbol, of the best variable-length prefix-free code for X^2.

(c) Is it true that for any DMS, $\overline{L}_{\min} \geq \overline{L}_{\min,2}$? Explain.

2.22. For a DMS X with alphabet $\mathcal{X} = \{1, 2, \ldots, M\}$, let $L_{\min,1}$, $L_{\min,2}$, and $L_{\min,3}$ be the normalized average length in bits per source symbol for a Huffman code over $\mathcal{X}$, $\mathcal{X}^2$ and $\mathcal{X}^3$ respectively. Show that $L_{\min,3} \leq \frac{2}{3} L_{\min,2} + \frac{1}{3} L_{\min,1}$.

2.23. (Run-Length Coding) Suppose $X_1, X_2, \ldots$, is a sequence of binary random symbols with $p_X(a) = 0.9$ and $p_X(b) = 0.1$. We encode this source by a variable-to-variable-length encoding technique known as run-length coding. The source output is first mapped into intermediate digits by counting the number of a's between each b. Thus an intermediate output occurs on each occurence of the symbol b. Since we don't want the intermediate digits to get too large, however, the intermediate digit 8 corresponds to 8 a's in a row; the counting restarts at this point. Thus, outputs appear on each b and on each 8 a's. For example, the first two lines below illustrate a string of source outputs and the corresponding intermediate outputs.

b	a	a	a	b	a	a	a	a	a	a	a	a	a	a	b	b	a	a	a	a	b
0				3								8			2	0					4
0000				0011								1			0010	0000					0100

The final stage of encoding assigns the codeword 1 to the intermediate integer 8, and assigns a 4 bit codeword consisting of 0 followed by the three bit binary representation for each integer 0 to 7. This is illustrated in the third line above.

(a) Show why the overall code is uniquely decodable.

(b) Find the expected total number of output bits corresponding to each occurrence of the letter b. This total number includes the four bit encoding of the letter b *and* the one bit encodings for each string of 8 letter a's preceding that letter b.

(c) By considering a string of 10^{20} binary symbols into the encoder, show that the number of b's to occur per input symbol is, with very high probability, very close to 0.1.

(d) Combine parts (b) and (c) to find the $\overline{L}$, the expected number of output bits per input symbol.

2.24. (a) Suppose a DMS emits h and t with probability 1/2 each. For $\varepsilon = 0.01$, what is T_ε^5?

(b) Find T_ε^1 for $\Pr(h) = 0.1$, $\Pr(t) = 0.9$, and $\varepsilon = 0.001$.

2.25. Consider a DMS with a two symbol alphabet, $\{a, b\}$ where $p_X(a) = 2/3$ and $p_X(b) = 1/3$. Let $\boldsymbol{X}^n = X_1, \ldots, X_n$ be a string of random symbols from the source with $n = 100{,}000$.

(a) Let $W(X_j)$ be the log pmf rv for the jth source output, *i.e.*, $W(X_j) = -\log 2/3$ for $X_j = a$ and $-\log 1/3$ for $X_j = b$. Find the variance of $W(X_j)$.

(b) For $\varepsilon = 0.01$, evaluate the bound on the probability of the typical set in (2.24).

(c) Let N_a be the number of a's in the string $\boldsymbol{X}^n = X_1, \dots, X_n$. The rv N_a is the sum of n iid rv's. Show what these rv's are.

(d) Express the rv $W(\boldsymbol{X}^n)$ as a function of the rv N_a. Note how this depends on n.

(e) Express the typical set in terms of bounds on N_a (*i.e.*, $T_\varepsilon^n = \{\boldsymbol{x}^n : \alpha < N_a < \beta\}$ and calculate α and β).

(f) Find the mean and variance of N_a. Approximate $\Pr\{T_\varepsilon^n\}$ by the central limit theorem approximation. The central limit theorem approximation is to evaluate $\Pr\{T_\varepsilon^n\}$ assuming that N_a is Gaussian with the mean and variance of the actual N_a.

One point of this exercise is to illustrate that the Chebyshev inequality used in finding $\Pr(T_\varepsilon)$ in the notes is very weak (although it is a strict bound, whereas the Gaussian approximation here is relatively accurate but not a bound). Another point is to show that n must be very large for the typical set to look typical.

2.26. For the rv's in the previous exercise, find $\Pr\{N_a = i\}$ for $i = 0, 1, 2$. Find the probability of each individual string $\boldsymbol{x}^n$ for those values of i. Find the particular string $\boldsymbol{x}^n$ that has maximum probability over all sample values of $\boldsymbol{X}^n$. What are the next most probable n-strings? Give a brief discussion of why the most probable n-strings are not regarded as typical strings.

2.27. Let $X_1, X_2, \dots,$ be a sequence of iid symbols from a finite alphabet. For any block length n and any small number $\varepsilon > 0$, define the *good* set of n-tuples $\mathbf{x^n}$ as the set

$$G_\varepsilon^n = \left\{ \mathbf{x}^n : \; p_{\mathbf{X}^n}(\mathbf{x}^n) > 2^{-n[\mathsf{H}[X]+\varepsilon]} \right\}.$$

(a) Explain how G_ε^n differs from the typical set T_ε^n.

(b) Show that $\Pr(G_\varepsilon^n) \geq 1 - \frac{\sigma_W^2}{n\varepsilon^2}$ where W is the log pmf rv for X. Nothing elaborate is expected here.

(c) Derive an upper bound on the number of elements in G_ε^n of the form $|G_\varepsilon^n| < 2^{n(\mathsf{H}[X]+\alpha)}$ and determine the value of α. (You are expected to find the smallest such α that you can, but not to prove that no smaller value can be used in an upper bound).

(d) Let $G_\varepsilon^n - T_\varepsilon^n$ be the set of n-tuples $\boldsymbol{x}^n$ that lie in G_ε^n but not in T_ε^n. Find an upper bound to $|G_\varepsilon^n - T_\varepsilon^n|$ of the form $|G_\varepsilon^n - T_\varepsilon^n| \leq 2^{n(\mathsf{H}[X]+\beta)}$. Again find the smallest β that you can.

(e) Find the limit of $|G_\varepsilon^n - T_\varepsilon^n|/|T_\varepsilon^n|$ as $n \to \infty$.

2.28. The typical set T_ε^n defined in the text is often called a weakly typical set, in contrast to another kind of typical set called a strongly typical set. Assume a discrete memoryless source and let $N_j(\boldsymbol{x}^n)$ be the number of symbols in an n string $\boldsymbol{x}^n$ taking on the value j. Then the strongly typical set $S_{\varepsilon_s}^n$ is defined as

$$S_{\varepsilon_s}^n = \left\{ \boldsymbol{x}^n : \; p_j(1-\varepsilon_s) < \frac{N_j(\boldsymbol{x}^n)}{n} < p_j(1+\varepsilon_s); \quad \text{for all}\, j \in \mathcal{X} \right\}.$$

(a) Show that $p_{\boldsymbol{X}^n}(\boldsymbol{x}^n) = \prod_j p_j^{N_j(\boldsymbol{x}^n)}$.

(b) Show that every $\boldsymbol{x}^n$ in $S^n_{\varepsilon_s}$ has the property that

$$\mathsf{H}(X)(1-\varepsilon_s) < \frac{-\log p_{\boldsymbol{X}^n}(\boldsymbol{x}^n)}{n} < \mathsf{H}(X)(1+\varepsilon_s)$$

(c) Show that if $\boldsymbol{x}^n \in S^n_{\varepsilon_s}$, then $\boldsymbol{x}^n \in T^n_\varepsilon$ with $\varepsilon = \mathsf{H}(X)\varepsilon_s$, *i.e.*, that $S^n_{\varepsilon_s} \subseteq T^n_\varepsilon$.

(d) Show that for any $\delta > 0$ and all sufficiently large n,

$$\Pr\left(\boldsymbol{X}^n \notin S^n_{\varepsilon_s}\right) \le \delta$$

Hint:Taking each letter j separately, $1 \le j \le M$, show that for all sufficiently large n, $\Pr\left(\left|\frac{N_j}{n} - p_j\right| \ge \varepsilon_s\right) \le \frac{\delta}{M}$.

(e) Show that for all $\delta > 0$ and all suffiently large n,

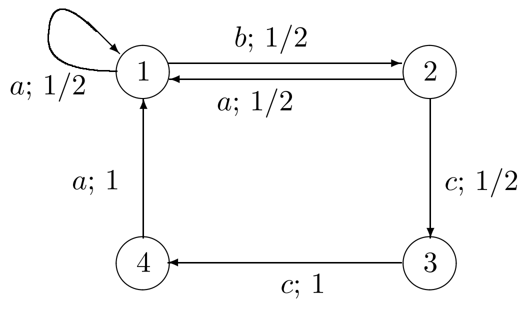
$$(1-\delta)2^{n(\mathsf{H}(X)-\varepsilon_s)} < T^n_{\varepsilon_s} < 2^{n(\mathsf{H}(X)+\varepsilon_s)}. \tag{2.44}$$

Note that parts (d) and (e) constitute the same theorem for the strongly typical set as Theorem 2.7.1 establishes for the weakly typical set. Typically the n required for (2.44) to hold (with the above correspondence between ε and ε_s) is considerably larger than than that for (2.27) to hold. We will use strong typicality later in proving the noisy channel coding theorem.

2.29. (a) The random variable D_n in Subsection 2.7.4 was defined as the initial string length of encoded bits required to decode the first n symbols of the source input. For the run-length coding example in Exercise 2.23, list the input strings and corresponding encoded output strings that must be inspected to decode the first source letter and from this find the pmf function of D_1. Hint: As many as 8 source letters must be encoded before X_1 can be decoded.

(b)Find the pmf of D_2. One point of this exercise is to convince you that D_n is a useful rv for proving theorems, but not a rv that is useful for detailed computation. It also shows clearly that D_n can depend on more than the first n source letters.

2.30. The Markov chain $S_0, S_1, \ldots$ below starts in steady state at time 0 and has 4 states, $\mathcal{S} = \{1,2,3,4\}$. The corresponding Markov source $X_1, X_2, \ldots$ has a source alphabet $\mathcal{X} = \{a,b,c\}$ of size 3.



(a) Find the steady-state probabilities $\{q(s)\}$ of the Markov chain.

(b) Find $\mathsf{H}[X_1]$.

(c) Find $\mathsf{H}[X_1|S_0]$.

(d) Describe a uniquely-decodable encoder for which $\overline{L} = \mathsf{H}[X_1|S_0)$. Assume that the initial state is known to the decoder. Explain why the decoder can track the state after time 0.

(e) Suppose you observe the source output without knowing the state. What is the maximum number of source symbols you must observe before knowing the state?

2.31. Let $X_1, X_2, \dots, X_n$ be discrete random symbols. Derive the following chain rule:

$$\mathsf{H}(X_1, \dots, X_n) = \mathsf{H}(X_1) + \sum_{j=2}^{n} \mathsf{H}(X_j|X_1, \dots, X_{j-1})$$

Hint: Use the chain rule for $n = 2$ in (2.37) and ask yourself whether a k tuple of random symbols is itself a random symbol.

2.32. Consider a discrete ergodic Markov chain $S_0, S_1, \dots$ with an arbitrary initial state distribution.

(a) Show that $\mathsf{H}[S_2|S_1S_0] = \mathsf{H}[S_2|S_1]$ (use the basic definition of conditional entropy).

(b) Show with the help of Exercise 2.31 that for any $n \geq 2$,

$$\mathsf{H}[S_1S_2 \cdots S_n|S_0] = \sum_{k=1}^{n} \mathsf{H}[S_k|S_{k-1}].$$

(c) Simplify this for the case where S_0 is in steady state.

(d) For a Markov source with outputs $X_1X_2\cdots$, explain why $\mathsf{H}[X_1 \cdots X_n|S_0] = \mathsf{H}[S_1 \cdots S_n|S_0]$. You may restrict this to $n = 2$ if you desire.

(e) Verify (2.40).

2.33. (Not for the faint of heart) In this exercise, the strong typicality discussed in Exercise 2.28 is used to show that source coding designed using only the lower order statistics of a source performs as well as a Markov source defined by those lower order statistics. In other words, what a designer doesn't know doesn't hurt, although encorporating the higher order statistics could help. Suppose a stationary source is described by kth order statistics for all $k \geq 1$ with $p(\boldsymbol{x}^k)$ denoting the steady state probability of $\boldsymbol{x}^k$. Let $N_{\boldsymbol{x}^k}(\boldsymbol{x}^n_{-k(+)2})$ be the number of appearances of $\boldsymbol{x}^k$ in $\boldsymbol{x}^n_{-k+1}$. Assume that the source is ergodic in the sense that for all $\delta > 0, \varepsilon > 0$, $\boldsymbol{x}^k$, and for all large enough n,

$$\Pr\left(\left|\frac{N_{\boldsymbol{x}^k}(\boldsymbol{X}^n_{-k+2})}{n} - p(\boldsymbol{x}^k)\right| \leq \varepsilon p(\boldsymbol{x}^k)\right) \geq 1 - \delta$$

For any given k, define the kth order Markov model for this source as a model where the state space is the set of all k-tuples $\boldsymbol{x}^k$ and the transition probabilities are given by $p(j\,|s = \boldsymbol{x}^k) = \frac{p(\boldsymbol{x}^k * j))}{p(\boldsymbol{x}^k)}$ where $\boldsymbol{x}^k * j$ is the (k+1) tuple formed by concatenating $\boldsymbol{x}^k$ with the source symbol j.

(a) Show that for this kth order Markov source model, the probability of a string $\boldsymbol{x}_1^n$ conditional on the state $s_0 = \boldsymbol{x}^0_{-k+1}$ is given by

$$q_k(\boldsymbol{x}_1^n|s_0) = \prod_{\boldsymbol{x}^k, j} p(j\,|s = \boldsymbol{x}^k)^{N_{\boldsymbol{x}^k * j}(\boldsymbol{x}^n_{-k+1})}$$

(b) The strongly typical set $S^n_{e,k}$ is defined as

$$S^n_{\varepsilon,k} = \left\{ \boldsymbol{x}^n_{-k+1} : p(\boldsymbol{x}^{k+1})(1-\varepsilon) < \frac{N_{\boldsymbol{x}^{k+1}}(\boldsymbol{X}^n_{-k+2})}{n} < p(\boldsymbol{x}^{k+1})(1+\varepsilon) \quad \text{for all } \boldsymbol{x}^{k+1} \right\}.$$

Show that for any $\varepsilon > 0$ and any $k \geq 1$, the probability (in the original probability space) of $S^n_{\varepsilon,k}$ approaches 1 with increasing n.

2.34. Perform an LZ77 parsing of the string $\underline{00011101}0010101100$. Assume a window of length $W = 8$; the initial window is underlined above. You should parse the rest of the string using the Lempel-Ziv algorithm.

2.35. Suppose that the LZ77 algorithm is used on the binary string $x_1^{10{,}000} = 0^{5000}1^{4000}0^{1000}$. This notation means 5000 repetitions of 0 followed by 4000 repetitions of 1 followed by 1000 repetitions of 0. Assume a window size $w = 1024$.

(a) Describe how the above string would be encoded. Give the encoded string and describe its substrings.

(b) How long is the encoded string?

(c) Suppose that the window size is reduced to $w = 8$. How long would the encoded string be in this case? (Note that such a small window size would only work well for really simple examples like this one.)

(d) Create a Markov source model with 2 states that is a reasonably good model for this source output. You are not expected to do anything very elaborate here; just use common sense.

(e) Find the entropy in bits per source symbol for your source model.

2.36. (a) Show that if an optimum (in the sense of minimum expected length) prefix-free code is chosen for any given pmf (subject to the condition $p_i > p_j$ for $i < j$), the code word lengths satisfy $l_i \leq l_j$ for all $i < j$. Use this to show that for all $j \geq 1$

$$l_j \geq \lfloor \log j \rfloor + 1$$

(c) The asymptotic efficiency of a prefix-free code for the positive integers is defined to be $\lim_{j\to\infty} \frac{l_j}{\log j}$. What is the asymptotic efficiency of the unary-binary code?

(d) Explain how to construct a prefix-free code for the positive integers where the asymptotic efficiency is 1. Hint: Replace the unary code for the integers $n = \lfloor \log j \rfloor + 1$ in the unary-binary code with a code whose length grows more slowly with increasing n.

Chapter 3

Quantization

3.1 Introduction to quantization

The previous chapter discussed coding and decoding for discrete sources. Discrete sources are a subject of interest in their own right (for text, computer files, etc.) and also serve as the inner layer for encoding analog source sequences and waveform sources (see Figure 3.1). This chapter treats coding and decoding for a sequence of analog values. Source coding for analog values is usually called *quantization*. Note that this is also the middle layer for waveform source/decoding.

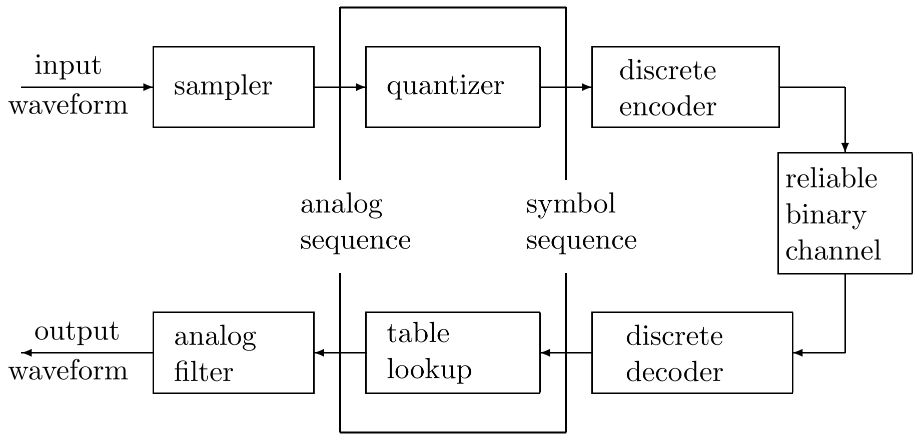


Figure 3.1: Encoding and decoding of discrete sources, analog sequence sources, and waveform sources. Quantization, the topic of this chapter, is the middle layer and should be understood before trying to understand the outer layer, which deals with waveform sources.

The input to the quantizer will be modeled as a sequence $U_1, U_2, \cdots$, of analog random variables (rv's). The motivation for this is much the same as that for modeling the input to a discrete source encoder as a sequence of random symbols. That is, the design of a quantizer should be responsive to the set of possible inputs rather than being designed for only a single sequence of numerical inputs. Also, it is desirable to treat very rare inputs differently from very common inputs, and a probability density is an ideal approach for this. Initially, $U_1, U_2, \ldots$ will be taken as independent identically distributed (iid) analog rv's with some given probability density function (pdf) $f_U(u)$.

A quantizer, by definition, maps the incoming sequence $U_1, U_2, \cdots$, into a sequence of discrete rv's $V_1, V_2, \cdots$, where the objective is that V_m, for each m in the sequence, should represent U_m with as little distortion as possible. Assuming that the discrete encoder/decoder at the inner layer of Figure 3.1 is uniquely decodable, the sequence $V_1, V_2, \cdots$ will appear at the output of the discrete encoder and will be passed through the middle layer (denoted 'table lookup') to represent the input $U_1, U_2, \cdots$. The output side of the quantizer layer is called a 'table lookup' because the alphabet for each discrete random variables V_m is a finite set of real numbers, and these are usually mapped into another set of symbols such as the integers 1 to M for an M symbol alphabet. Thus on the output side a look-up function is required to convert back to the numerical value V_m.

As discussed in Section 2.1, the quantizer output V_m, if restricted to an alphabet of M possible values, cannot represent the analog input U_m perfectly. Increasing M, *i.e.*, quantizing more finely, typically reduces the distortion, but cannot eliminate it.

When an analog rv U is quantized into a discrete rv V, the mean-squared distortion is defined to be $\mathsf{E}[(U-V)^2]$. Mean-squared distortion (often called mean-sqared error) is almost invariably used in this text to measure distortion. When studying the conversion of waveforms into sequences in the next chapter, it will be seen that mean-squared distortion is particularly convenient for converting the distortion for the sequence into mean-squared distortion for the waveform.

There are some disadvantages to measuring distortion only in a mean-squared sense. For example, efficient speech coders are based on models of human speech. They make use of the fact that human listeners are more sensitive to some kinds of reconstruction error than others, so as, for example, to permit larger errors when the signal is loud than when it is soft. Speech coding is a specialized topic which we do not have time to explore (see, for example, [7]. Understanding compression relative to a mean-squared distortion measure, however, will develop many of the underlying principles needed in such more specialized studies.

In what follows, scalar quantization is considered first. Here each analog rv in the sequence is quantized independently of the other rv's. Next vector quantization is considered. Here the analog sequence is first segmented into blocks of n rv's each; then each n-tuple is quantized as a unit.

Our initial approach to both scalar and vector quantization will be to minimize mean-squared distortion subject to a constraint on the size of the quantization alphabet. Later, we consider minimizing mean-squared distortion subject to a constraint on the *entropy* of the quantized output. This is the relevant approach to quantization if the quantized output sequence is to be source-encoded in an efficient manner, *i.e.*, to reduce the number of encoded bits per quantized symbol to little more than the corresponding entropy.

3.2 Scalar quantization

A *scalar quantizer* partitions the set $\mathbb{R}$ of real numbers into M subsets $\mathcal{R}_1, \ldots, \mathcal{R}_M$, called *quantization regions.* Assume that each quantization region is an interval; it will soon be seen why this assumption makes sense. Each region $\mathcal{R}_j$ is then represented by a *representation point* $a_j \in \mathbb{R}$. When the source produces a number $u \in \mathcal{R}_j$, that number is quantized into the point a_j. A scalar quantizer can be viewed as a function $\{v(u) : \mathbb{R} \to \mathbb{R}\}$ that maps analog real values u into discrete real values $v(u)$ where $v(u) = a_j$ for $u \in \mathcal{R}_j$.

An analog sequence $u_1, u_2, \dots$ of real-valued symbols is mapped by such a quantizer into the discrete sequence $v(u_1), v(u_2) \dots$. Taking $u_1, u_2 \dots$, as sample values of a random sequence $U_1, U_2, \dots$, the map $v(u)$ generates an rv V_k for each U_k; V_k takes the value a_j if $U_k \in \mathcal{R}_j$. Thus each quantized output V_k is a discrete rv with the alphabet $\{a_1, \dots , a_M\}$. The discrete random sequence $V_1, V_2, \dots$, is encoded into binary digits, transmitted, and then decoded back into the same discrete sequence. For now, assume that transmission is error-free.

We first investigate how to choose the quantization regions $\mathcal{R}_1, \dots , \mathcal{R}_M$, and how to choose the corresponding representation points. Initially assume that the regions are intervals, ordered as in Figure 3.2, with $\mathcal{R}_1 = (-\infty, b_1], \mathcal{R}_2 = (b_1, b_2], \dots , \mathcal{R}_M = (b_{M-1}, \infty)$. Thus an M-level quantizer is specified by $M - 1$ interval endpoints, $b_1, \dots , b_{M-1}$, and M representation points, $a_1, \dots , a_M$.

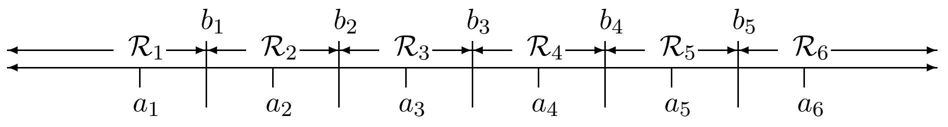


Figure 3.2: Quantization regions and representation points.

For a given value of M, how can the regions and representation points be chosen to minimize mean-squared error? This question is explored in two ways:

- Given a set of representation points $\{a_j\}$, how should the intervals $\{\mathcal{R}_j\}$ be chosen?
- Given a set of intervals $\{\mathcal{R}_j\}$, how should the representation points $\{a_j\}$ be chosen?

3.2.1 Choice of intervals for given representation points

The choice of intervals for given representation points, $\{a_j; 1 \le j \le M\}$ is easy: given any $u \in \mathbb{R}$, the squared error to a_j is $(u - a_j)^2$. This is minimized (over the fixed set of representation points $\{a_j\}$) by representing u by the closest representation point a_j. This means, for example, that if u is between a_j and a_{j+1}, then u is mapped into the closer of the two. Thus the boundary b_j between $\mathcal{R}_j$ and $\mathcal{R}_{j+1}$ must lie halfway between the representation points a_j and $a_{j+1}, 1 \le j \le M - 1$. That is, $b_j = \frac{a_j + a_{j+1}}{2}$. This specifies each quantization region, and also shows why each region should be an interval. Note that this minimization of mean-squared distortion does not depend on the probabilistic model for $U_1, U_2, \dots$.

3.2.2 Choice of representation points for given intervals

For the second question, the probabilistic model for $U_1, U_2, \dots$ is important. For example, if it is known that each U_k is discrete and has only one sample value in each interval, then the representation points would be chosen as those sample value. Suppose now that the rv's $\{U_k\}$

are iid analog rv's with the pdf $f_U(u)$. For a given set of points $\{a_j\}$, $V(U)$ maps each sample value $u \in \mathcal{R}_j$ into a_j. The mean-squared distortion (or mean-squared error MSE) is then

$$\text{MSE} = \mathsf{E}[(U - V(U))^2] = \int_{-\infty}^{\infty} f_U(u)(u - v(u))^2\, du = \sum_{j=1}^{M} \int_{\mathcal{R}_j} f_U(u)\,(u - a_j)^2\, du. \tag{3.1}$$

In order to minimize (3.1) over the set of a_j, it is simply necessary to choose each a_j to minimize the corresponding integral (remember that the regions are considered fixed here). Let $f_j(u)$ denote the conditional pdf of U given that $\{u \in \mathcal{R}_j\}$; *i.e.*,

$$f_j(u) = \begin{cases} \frac{f_U(u)}{Q_j}, & \text{if} \quad u \in \mathcal{R}_j; \\ 0, & \text{otherwise,} \end{cases} \tag{3.2}$$

where $Q_j = \Pr\{U \in \mathcal{R}_j\}$. Then, for the interval $\mathcal{R}_j$,

$$\int_{\mathcal{R}_j} f_U(u)\,(u - a_j)^2\, du = Q_j \int_{\mathcal{R}_j} f_j(u)\,(u - a_j)^2\, du. \tag{3.3}$$

Now (3.3) is minimized by choosing a_j to be the mean of a random variable with the pdf $f_j(u)$. To see this, note that for any rv Y and real number a,

$$\overline{(Y - a)^2} = \overline{Y^2} - 2a\overline{Y} + a^2,$$

which is minimized over a when $a = \overline{Y}$.

This provides a set of conditions that the endpoints $\{b_j\}$ and the points $\{a_j\}$ must satisfy to achieve the MSE — namely, each b_j must be the midpoint between a_j and a_{j+1} and each a_j must be the mean of an rv U_j with pdf $f_j(u)$. In other words, a_j must be the conditional mean of U conditional on $U \in \mathcal{R}_j$.

These conditions are necessary to minimize the MSE for a given number M of representation points. They are not sufficient, as shown by an example at the end of this section. Nonetheless, these necessary conditions provide some insight into the minimization of the MSE.

3.2.3 The Lloyd-Max algorithm

The *Lloyd-Max algorithm*[1] is an algorithm for finding the endpoints $\{b_j\}$ and the representation points $\{a_j\}$ to meet the above necessary conditions. The algorithm is almost obvious given the necessary conditions; the contribution of Lloyd and Max was to define the problem and develop the necessary conditions. The algorithm simply alternates between the optimizations of the previous subsections, namely optimizing the endpoints $\{b_j\}$ for a given set of $\{a_j\}$, and then optimizing the points $\{a_j\}$ for the new endpoints.

The Lloyd-Max algorithm is as follows. Assume that the number M of quantizer levels and the pdf $f_U(u)$ are given.

1. Choose an arbitrary initial set of M representation points $a_1 < a_2 < \cdots < a_M$.

[1]This algorithm was developed independently by S. P. Lloyd in 1957 and J. Max in 1960. Lloyd's work was done in the Bell Laboratories research department and became widely circulated, although unpublished until 1982 [10]. Max's work [11] was published in 1960.

2. For each $j; 1 \leq j \leq M-1$, set $b_j = \frac{1}{2}(a_{j+1} + a_j)$.
3. For each $j; 1 \leq j \leq M$, set a_j equal to the conditional mean of U given $U \in (b_{j-1}, b_j]$ (where b_0 and b_M are taken to be $-\infty$ and $+\infty$ respectively).
4. Repeat steps (2) and (3) until further improvement in MSE is negligible; then stop.

The MSE decreases (or remains the same) for each execution of step (2) and step (3). Since the MSE is nonnegative, it approaches some limit. Thus if the algorithm terminates when the MSE improvement is less than some given $\varepsilon > 0$, then the algorithm must terminate after a finite number of iterations.

Example:. This example shows that the algorithm might reach a local minimum of MSE instead of the global minimum. Consider a quantizer with $M = 2$ representation points, and an rv U whose pdf $f_U(u)$ has three peaks, as shown in Figure 3.3.

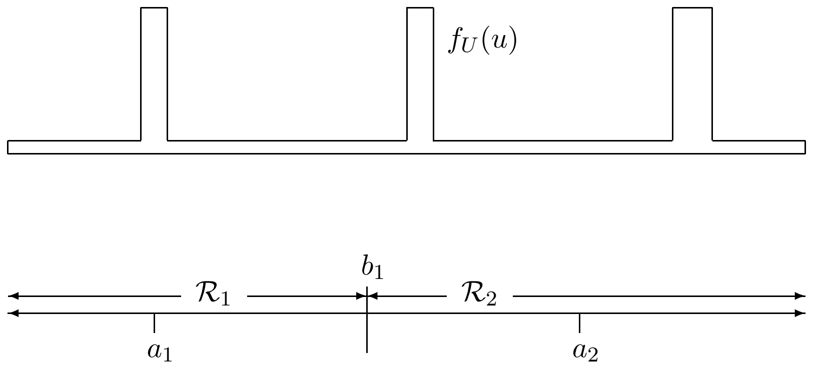


Figure 3.3: Example of regions and representaion points that satisfy Lloyd-Max conditions without minimizing mean-squared distortion.

It can be seen that one region must cover two of the peaks, yielding quite a bit of distortion, while the other will represent the remaining peak, yielding little distortion. In the figure, the two rightmost peaks are both covered by $\mathcal{R}_2$, with the point a_2 between them. Both the points and the regions satisfy the necessary conditions and cannot be locally improved. However, it can be seen in the figure that the rightmost peak is more probable than the other peaks. It follows that the MSE would be lower if $\mathcal{R}_1$ covered the two leftmost peaks.

The Lloyd-Max algorithm is a type of hill-climbing algorithm; starting with an arbitrary set of values, these values are modified until reaching the top of a hill where no more local improvements are possible.[2] A reasonable approach in this sort of situation is to try many randomly chosen starting points, perform the Lloyd-Max algorithm on each and then take the best solution. This is somewhat unsatisfying since there is no general technique for determining when the optimal solution has been found.

[2]It would be better to call this a valley-descending algorithm, both because a minimum is desired and also because binoculars can not be used at the bottom of a valley to find a distant lower valley.

3.3 Vector quantization

As with source coding of discrete sources, we next consider quantizing n source variables at a time. This is called *vector quantization*, since an n-tuple of rv's may be regarded as a vector rv in an n-dimensional vector space. We will concentrate on the case $n = 2$ so that illustrative pictures can be drawn.

One possible approach is to quantize each dimension independently with a scalar (one-dimensional) quantizer. This results in a rectangular grid of quantization regions as shown below. The MSE per dimension is the same as for the scalar quantizer using the same number of bits per dimension. Thus the best 2D vector quantizer has an MSE per dimension at least as small as that of the best scalar quantizer.

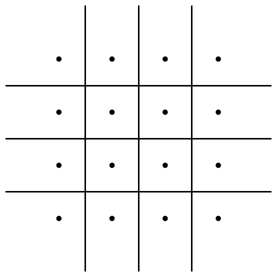


Figure 3.4: 2D rectangular quantizer.

To search for the minimum-MSE 2D vector quantizer with a given number M of representation points, the same approach is used as with scalar quantization.

Let (U, U') be the two rv's being jointly quantized. Suppose a set of M 2D representation points $\{(a_j, a'_j)\}$, $1 \le j \le M$ is chosen. For example, in the figure above, there are 16 representation points, represented by small dots. Given a sample pair (u, u') and given the M representation points, which representation point should be chosen for the given (u, u')? Again, the answer is easy. Since mapping (u, u') into (a_j, a'_j) generates a squared error equal to $(u - a_j)^2 + (u' - a'_j)^2$, the point (a_j, a'_j) which is closest to (u, u') in Euclidean distance should be chosen.

Consequently, the region $\mathcal{R}_j$ must be the set of points (u, u') that are closer to (a_j, a'_j) than to any other representation point. Thus the regions $\{\mathcal{R}_j\}$ are minimum-distance regions; these regions are called the *Voronoi* regions for the given representation points. The boundaries of the Voronoi regions are perpendicular bisectors between neighboring representation points. The minimum-distance regions are thus in general convex polygonal regions, as illustrated in the figure below.

As in the scalar case, the MSE can be minimized for a given set of regions by choosing the representation points to be the conditional means within those regions. Then, given this new set of representation points, the MSE can be further reduced by using the Voronoi regions for the new points. This gives us a 2D version of the Lloyd-Max algorithm, which must converge to a local minimum of the MSE. This can be generalized straightforwardly to any dimension n.

As already seen, the Lloyd-Max algorithm only finds local minima to the MSE for scalar quantizers. For vector quantizers, the problem of local minima becomes even worse. For example, when $U_1, U_2, \cdots$ are iid, it is easy to see that the rectangular quantizer in Figure 3.4 satisfies the Lloyd-Max conditions if the corresponding scalar quantizer does (see Exercise 3.9). It will

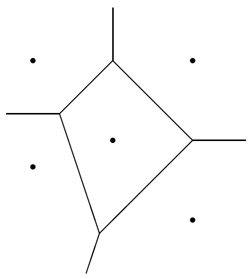


Figure 3.5: Voronoi regions for given set of representation points.

soon be seen, however, that this is not necessarily the minimum MSE.

Vector quantization was a popular research topic for many years. The problem is that quantizing complexity goes up exponentially with n, and the reduction in MSE with increasing n is quite modest, unless the samples are statistically highly dependent.

3.4 Entropy-coded quantization

We must now ask if minimizing the MSE for a given number M of representation points is the right problem. The minimum expected number of bits per symbol, $\overline{L}_{\min}$, required to encode the quantizer output was shown in Chapter 2 to be governed by the entropy $\mathsf{H}[V]$ of the quantizer output, not by the size M of the quantization alphabet. Therefore, anticipating efficient source coding of the quantized outputs, we should really try to minimize the MSE for a given entropy $\mathsf{H}[V]$ rather than a given number of representation points.

This approach is called *entropy-coded quantization* and is almost implicit in the layered approach to source coding represented in Figure 3.1. Discrete source coding close to the entropy bound is similarly often called entropy coding. Thus entropy-coded quantization refers to quantization techniques that are designed to be followed by entropy coding.

The entropy $\mathsf{H}[V]$ of the quantizer output is determined only by the probabilities of the quantization regions. Therefore, given a set of regions, choosing the representation points as conditional means minimizes their distortion without changing the entropy. However, given a set of representation points, the optimal regions are not necessarily Voronoi regions (e.g., in a scalar quantizer, the point separating two adjacent regions is not necessarily equidistant from the two represention points.)

For example, for a scalar quantizer with a constraint $\mathsf{H}[V] \leq \frac{1}{2}$ and a Gaussian pdf for U, a reasonable choice is three regions, the center one having high probability $1 - 2p$ and the outer ones having small, equal probability p, such that $\mathsf{H}[V] = \frac{1}{2}$.

Even for scalar quantizers, minimizing MSE subject to an entropy constraint is a rather messy problem. Considerable insight into the problem can be obtained by looking at the case where the target entropy is large— *i.e.*, when a large number of points can be used to achieve small MSE. Fortunately this is the case of greatest practical interest.

Example:. For the following simple example, consider the minimum-MSE quantizer using a constraint on the number of representation points M compared to that using a constraint on the entropy $\mathsf{H}[V]$.

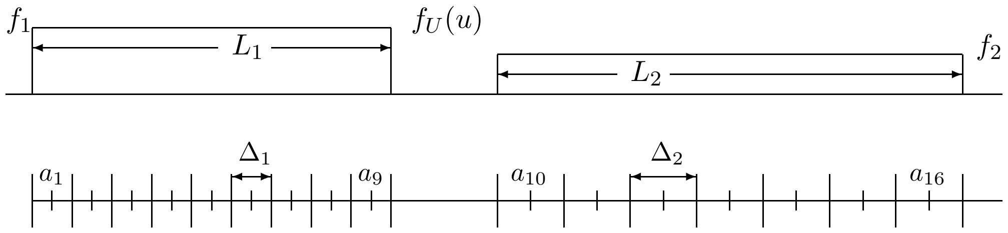


Figure 3.6: Comparison of constraint on M to constraint on $\mathsf{H}(U)$.

The example shows a piecewise constant pdf $f_U(u)$ that takes on only two positive values, say $f_U(u) = f_1$ over an interval of size L_1, and $f_U(u) = f_2$ over a second interval of size L_2. Assume that $f_U(u) = 0$ elsewhere. Because of the wide separation between the two intervals, they can be quantized separately without providing any representation point in the region between the intervals. Let M_1 and M_2 be the number of representation points in each interval. In the figure, $M_1 = 9$ and $M_2 = 7$. Let $\Delta_1 = L_1/M_1$ and $\Delta_2 = L_2/M_2$ be the lengths of the quantization regions in the two ranges (by symmetry, each quantization region in a given interval should have the same length). The representation points are at the center of each quantization interval. The MSE, conditional on being in a quantization region of length Δ_i, is the MSE of a uniform distribution over an interval of length Δ_i, which is easily computed to be $\Delta_i^2/12$. The probability of being in a given quantization region of size Δ_i is $f_i\Delta_i$, so the overall MSE is given by

$$\text{MSE} = M_1\frac{\Delta_1^2}{12}f_1\Delta_1 + M_2\frac{\Delta_2^2}{12}f_2\Delta_2 = \frac{1}{12}\Delta_1^2 f_1 L_1 + \frac{1}{12}\Delta_2^2 f_2 L_2. \tag{3.4}$$

This can be minimized over Δ_1 and Δ_2 subject to the constraint that $M = M_1 + M_2 = L_1/\Delta_1 + L_2/\Delta_2$. Ignoring the constraint that M_1 and M_2 are integers (which makes sense for M large), Exercise 3.4 shows that the minimum MSE occurs when Δ_i is chosen inversely proportional to the cube root of f_i. In other words,

$$\frac{\Delta_1}{\Delta_2} = \left(\frac{f_2}{f_1}\right)^{1/3}. \tag{3.5}$$

This says that the size of a quantization region decreases with increasing probability density. This is reasonable, putting the greatest effort where there is the most probability. What is perhaps surprising is that this effect is so small, proportional only to a cube root.

Perhaps even more surprisingly, if the MSE is minimized subject to a constraint on entropy for this example, then Exercise 3.4 shows that the quantization intervals all have the same length! A scalar quantizer in which all intervals have the same length is called a *uniform scalar quantizer*. The following sections will show that uniform scalar quantizers have remarkable properties for high-rate quantization.

3.5 High-rate entropy-coded quantization

This section focuses on high-rate quantizers where the quantization regions can be made sufficiently small so that the probability density is approximately constant within each region. It will be shown that under these conditions the combination of a uniform scalar quantizer followed by discrete entropy coding is nearly optimum (in terms of mean-squared distortion) within the class of scalar quantizers. This means that a uniform quantizer can be used as a universal quantizer with very little loss of optimality. The probability distribution of the rv's to be quantized can be explointed at the level of discrete source coding. Note however that this essential optimality of uniform quantizers relies heavily on the assumption that mean-squared distortion is an appropriate distortion measure. With voice coding, for example, a given distortion at low signal levels is for more harmful than the same distortion at high signal levels.

In the following sections, it is assumed that the source output is a sequence $U_1, U_2, \dots$, of iid real analog-valued rv's, each with a probability density $f_U(u)$. It is further assumed that the probability density function (pdf) $f_U(u)$ is smooth enough and the quantization fine enough that $f_U(u)$ is almost constant over each quantization region.

The analogue of the entropy $\mathsf{H}[X]$ of a discrete rv is the differential entropy $\mathsf{h}[U]$ of an analog rv. After defining $\mathsf{h}[U]$,the properties of $\mathsf{H}[U]$ and $\mathsf{h}[U]$ will be compared.

The performance of a uniform scalar quantizer followed by entropy coding will then be analyzed. It will be seen that there is a tradeoff between the rate of the quantizer and the mean-squared error (MSE) between source and quantized output. It is also shown that the uniform quantizer is essentially optimum among scalar quantizers at high rate.

The performance of uniform vector quantizers followed by entropy coding will then be analyzed and similar tradeoffs will be found. A major result is that vector quantizers can achieve a gain over scalar quantizers (*i.e.*, a reduction of MSE for given quantizer rate), but that the reduction in MSE is at most a factor of $\pi e/6 = 1.42$.

As in the discrete case, generalizations to analog sources with memory are possible, but not discussed here.

3.6 Differential entropy

The differential entropy $\mathsf{h}[U]$ of an analog random variable (rv) U is analogous to the entropy $\mathsf{H}[X]$ of a discrete random symbol X. It has many similarities, but also some important differences.

Definition The *differential entropy* of an analog real rv U with pdf $f_U(u)$ is

$$\mathsf{h}[U] = \int_{-\infty}^{\infty} -f_U(u) \log f_U(u)\ du.$$

The integral may be restricted to the region where $f_U(u) > 0$, since $0 \log 0$ is interpreted as 0. Assume that $f_U(u)$ is smooth and that the integral exists with a finite value. Exercise 3.7 gives an example where $\mathsf{h}(U)$ is infinite.

As before, the logarithms are base 2 and the units of $\mathsf{h}[U]$ are bits per source symbol.

Like $\mathsf{H}[X]$, the differential entropy $\mathsf{h}[U]$ is the expected value of the rv $-\log f_U(U)$. The log of the joint density of several independent rv's is the sum of the logs of the individual pdf's, and this can be used to derive an AEP similar to the discrete case ([**?**]).

Unlike $\mathsf{H}[X]$, the differential entropy $\mathsf{h}[U]$ can be negative and depends on the scaling of the outcomes. This can be seen from the following two examples.

Example: (*Uniform distributions*). Let $f_U(u)$ be a uniform distribution over an interval $[a, a+\Delta]$ of length Δ; *i.e.*, $f_U(u) = 1/\Delta$ for $u \in [a, a+\Delta]$, and $f_U(u) = 0$ elsewhere. Then $-\log f_U(u) = \log \Delta$ where $f_U(u) > 0$ and

$$\mathsf{h}[U] = \mathsf{E}[-\log f_U(U)] = \log \Delta.$$

Example: (*Gaussian distribution*). Let $f_U(u)$ be a Gaussian distribution with mean m and variance σ^2; *i.e.*,

$$f_U(u) = \sqrt{\frac{1}{2\pi\sigma^2}}\ \exp\left\{-\frac{(u-m)^2}{2\sigma^2}\right\}.$$

Then $-\log f_U(u) = \frac{1}{2}\log 2\pi\sigma^2 + (\log e)(u-m)^2/(2\sigma^2)$. Since $\mathsf{E}[(U-m)^2] = \sigma^2$,

$$\mathsf{h}[U] = \mathsf{E}[-\log f_U(U)] = \frac{1}{2}\log(2\pi\sigma^2) + \frac{1}{2}\log e = \frac{1}{2}\log(2\pi e\sigma^2).$$

It can be seen from these expressions that by making Δ or σ^2 arbitrarily small, the differential entropy can be made arbitrarily negative, while by making Δ or σ^2 arbitrarily large, the differential entropy can be made arbitrarily positive.

If the rv U is rescaled to αU for some scale factor $\alpha > 0$, then the differential entropy is increased by $\log \alpha$, both in these examples and in general. In other words, $\mathsf{h}[U]$ is not invariant to scaling. Note, however, that differential entropy is invariant to translation of the pdf, *i.e.*, an rv and its fluctuation around the mean have the same differential entropy.

One of the important properties of entropy is that it does not depend on the labeling of the elements of the alphabet, *i.e.*, it is invariant to invertible transformations. Differential entropy is very different in this respect, and, as just illustrated, it is modified by even such a trivial transformation as a change of scale. The reason for this is that the probability density is a probability per unit length, and therefore depends on the measure of length. In fact, as seen more clearly later, this fits in very well with the fact that source coding for analog sources also depends on an error term per unit length.

Definition The *differential entropy* of an n-tuple of rv's $\boldsymbol{U}^n = (U_1, \cdots, U_n)$ with joint pdf $f_{\boldsymbol{U}^n}(\boldsymbol{u}^n)$ is

$$\mathsf{h}[\boldsymbol{U}^n] = \mathsf{E}[-\log f_{\boldsymbol{U}^n}(\boldsymbol{U}^n)].$$

Like entropy, differential entropy has the property that if U and V are independent rv's, then the entropy of the joint variable UV with pdf $f_{UV}(u,v) = f_U(u)f_V(v)$ is $\mathsf{h}[UV] = \mathsf{h}[U] + \mathsf{h}[V]$. Again, this follows from the fact that the log of the joint probability density of independent rv's is additive, *i.e.*, $-\log f_{UV}(u,v) = -\log f_U(u) - \log f_V(v)$.

Thus the differential entropy of a vector rv $\boldsymbol{U}^n$, corresponding to a string of n iid rv's $U_1, U_2, \ldots, U_n$, each with the density $f_U(u)$, is $\mathsf{h}[\boldsymbol{U}^n] = n\mathsf{h}[U]$.

3.7 Performance of uniform high-rate scalar quantizers

This section analyzes the performance of uniform scalar quantizers in the limit of high rate. Appendix A continues the analysis for the nonuniform case and shows that uniform quantizers are effectively optimal in the high-rate limit.

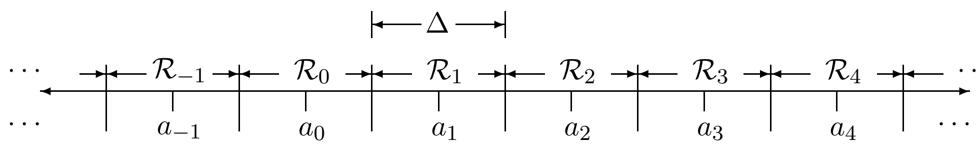


Figure 3.7: Uniform scalar quantizer.

For a uniform scalar quantizer, every quantization interval $\mathcal{R}_j$ has the same length $|\mathcal{R}_j| = \Delta$. In other words, $\mathbb{R}$ (or the portion of $\mathbb{R}$ over which $f_U(u) > 0$), is partitioned into equal intervals, each of length Δ.

Assume there are enough quantization regions to cover the region where $f_U(u) > 0$. For the Gaussian distribution, for example, this requires an infinite number of representation points, $-\infty < j < \infty$. Thus, in this example the quantized discrete rv V has a countably infinite alphabet. Obviously, practical quantizers limit the number of points to a finite region $\mathcal{R}$ such that $\int_{\mathcal{R}} f_U(u)\ du \approx 1$.

Assume that Δ is small enough that the pdf $f_U(u)$ is approximately constant over any one quantization interval. More precisely, define $\overline{f}(u)$ (see Figure 3.8) as the average value of $f_U(u)$ over the quantization interval containing u,

$$\overline{f}(u) = \frac{\int_{\mathcal{R}_j} f_U(u)du}{\Delta} \qquad \text{for} \quad u \in \mathcal{R}_j. \tag{3.6}$$

From (3.6) it is seen that $\Delta\overline{f}(u) = \Pr(\mathcal{R}_j)$ for all integer j and all $u \in \mathcal{R}_j$.

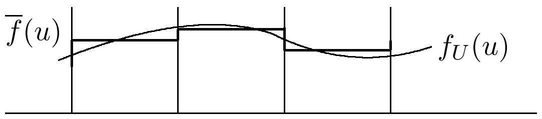


Figure 3.8: Average density over each $\mathcal{R}_j$.

The *high-rate assumption* is that $f_U(u) \approx \overline{f}(u)$ for all $u \in \mathbb{R}$. This means that $f_U(u) \approx \Pr(\mathcal{R}_j)/\Delta$ for $u \in \mathcal{R}_j$. It also means that the conditional pdf $f_{U|\mathcal{R}_j}(u)$ of U conditional on $u \in \mathcal{R}_j$ is approximated by

$$f_{U|\mathcal{R}_j}(u) \approx \begin{cases} 1/\Delta, & u \in \mathcal{R}_j; \\ 0, & u \notin \mathcal{R}_j. \end{cases}$$

Consequently the conditional mean a_j is approximately in the center of the interval $\mathcal{R}_j$, and the mean-squared error is approximately given by

$$\text{MSE} \approx \int_{-\Delta/2}^{\Delta/2} \frac{1}{\Delta} u^2 du = \frac{\Delta^2}{12} \tag{3.7}$$

for each quantization interval $\mathcal{R}_j$. Consequently this is also the overall MSE.

Next consider the entropy of the quantizer output V. The probability p_j that $V = a_j$ is given by both

$$p_j = \int_{\mathcal{R}_j} f_U(u)\, du \quad \text{and, for all } u \in \mathcal{R}_j, \quad p_j = \overline{f}(u)\Delta. \tag{3.8}$$

Therefore the entropy of the discrete rv V is

$$\begin{aligned} \mathsf{H}[V] &= \sum_j -p_j \log p_j = \sum_j \int_{\mathcal{R}_j} -f_U(u) \log[\overline{f}(u)\Delta]\, du \\ &= \int_{-\infty}^{\infty} -f_U(u) \log[\overline{f}(u)\Delta]\; du \qquad (3.9) \\ &= \int_{-\infty}^{\infty} -f_U(u) \log[\overline{f}(u)]\; du\; - \log \Delta, \qquad (3.10) \end{aligned}$$

where the sum of disjoint integrals were combined into a single integral.

Finally, using the high-rate approximation[3] $f_U(u) \approx \overline{f}(u)$, this becomes

$$\begin{aligned} \mathsf{H}[V] &\approx \int_{-\infty}^{\infty} -f_U(u) \log[f_U(u)\Delta]\; du \\ &= \mathsf{h}[U] - \log \Delta. \qquad (3.11) \end{aligned}$$

Since the sequence $U_1, U_2, \dots$ of inputs to the quantizer is memoryless (iid), the quantizer output sequence $V_1, V_2, \dots$ is an iid sequence of discrete random symbols representing quantization points— *i.e.*, a discrete memoryless source. A uniquely-decodable source code can therefore be used to encode this output sequence into a bit sequence at an average rate of $\overline{L} \approx \mathsf{H}[V] \approx \mathsf{h}[U] - \log \Delta$ bits/symbol. At the receiver, the mean-squared quantization error in reconstructing the original sequence is approximately MSE $\approx \Delta^2/12$.

The important conclusions from this analysis are illustrated in Figure 3.9 and are summarized as follows:

- Under the high-rate assumption, the rate $\overline{L}$ for a uniform quantizer followed by discrete entropy coding depends only on the differential entropy $\mathsf{h}[U]$ of the source and the spacing Δ of the quantizer. It does not depend on any other feature of the source pdf $f_U(u)$, nor on any other feature of the quantizer, such as the number M of points, so long as the quantizer intervals cover $f_U(u)$ sufficiently completely and finely.
- The rate $\overline{L} \approx \mathsf{H}[V]$ and the MSE are parametrically related by Δ, *i.e.*,

$$\overline{L} \approx \mathsf{h}(U) - \log \Delta; \qquad \text{MSE} \approx \frac{\Delta^2}{12}. \tag{3.12}$$

  Note that each reduction in Δ by a factor of 2 will reduce the MSE by a factor of 4 and increase the required transmission rate $\overline{L} \approx \mathsf{H}[V]$ by 1 bit/symbol. Communication engineers express this by saying that each additional bit per symbol decreases the mean-squared distortion[4] by 6 dB. Figure 3.9 sketches MSE as a function of $\overline{L}$.

[3]Exercise 3.6 provides some insight into the nature of the approximation here. In particular, the difference between $\mathsf{h}[U] - \log \Delta$ and $\mathsf{H}[V]$ is $\int f_U(u) \log[\overline{f}(u)/f_U(u)]\; du$. This quantity is always nonpositive and goes to zero with Δ as Δ^2. Similarly, the approximation error on MSE goes to 0 as Δ^4.

[4]A quantity x expressed in dB is given by $10 \log_{10} x$. This very useful and common logarithmic measure is discussed in detail in Chapter 6.

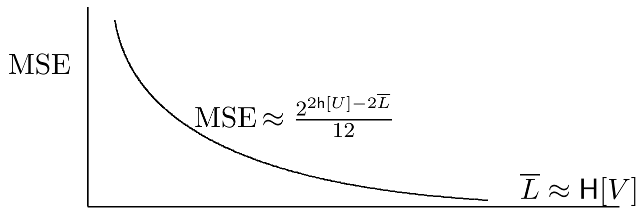


Figure 3.9: MSE as a function of $\overline{L}$ for a scalar quantizer with the high-rate approximation. Note that changing the source entropy $\mathsf{h}(U)$ simply shifts the figure right or left. Note also that log MSE is linear, with a slope of -2, as a function of $\overline{L}$.

Conventional b-bit analog-to-digital (A/D) converters are uniform scalar 2^b-level quantizers that cover a certain range $\mathcal{R}$ with a quantizer spacing $\Delta = 2^{-b}|\mathcal{R}|$. The input samples must be scaled so that the probability that $u \notin \mathcal{R}$ (the "overflow probability") is small. For a fixed scaling of the input, the tradeoff is again that increasing b by 1 bit reduces the MSE by a factor of 4.

Conventional A/D converters are not usually directly followed by entropy coding. The more conventional approach is to use A/D conversion to produce a very high rate digital signal that can be further processed by digital signal processing (DSP). This digital signal is then later compressed using algorithms specialized to the particular application (voice, images, etc.). In other words, the clean layers of Figure 3.1 oversimplify what is done in practice. On the other hand, it is often best to view compression in terms of the Figure 3.1 layers, and then use DSP as a way of implementing the resulting algorithms.

The relation $\mathsf{H}[V] \approx \mathsf{h}[u] - \log\Delta$ provides an elegant interpretation of differential entropy. It is obvious that there must be some kind of tradeoff between MSE and the entropy of the representation, and the differential entropy specifies this tradeoff in a very simple way for high rate uniform scalar quantizers. $\mathsf{H}[V]$ is the entropy of a finely quantized version of U, and the additional term $\log\Delta$ relates to the "uncertainty" within an individual quantized interval. It shows explicitly how the scale used to measure U affects $\mathsf{h}[U]$.

Appendix A considers nonuniform scalar quantizers under the high rate assumption and shows that nothing is gained in the high-rate limit by the use of nonuniformity.

3.8 High-rate two-dimensional quantizers

The performance of uniform two-dimensional (2D) quantizers are now analyzed in the limit of high rate. Appendix B considers the nonuniform case and shows that uniform quantizers are again effectively optimal in the high-rate limit.

A 2D quantizer operates on 2 source samples $\boldsymbol{u} = (u_1, u_2)$ at a time; *i.e.*, the source alphabet is $\boldsymbol{U}^2 = \mathbb{R}^2$. Assuming iid source symbols, the joint pdf is then $f_{\boldsymbol{U}^2}(\boldsymbol{u}^2) = f_U(u_1)f_U(u_2)$, and the joint differential entropy is $\mathsf{h}[\boldsymbol{U}^2] = 2\mathsf{h}[U]$.

Like a uniform scalar quantizer, a uniform 2D quantizer is based on a fundamental quantization region $\mathcal{R}$ ("quantization cell") whose translates tile[5] the 2D plane. In the one-dimensional case,

[5]A region of the 2D plane is said to *tile* the plane if the region, plus translates and rotations of the region, fill the plane without overlap. For example the square and the hexagon tile the plane. Also, rectangles tile the

there is really only one sensible choice for $\mathcal{R}$, namely an interval of length Δ, but in higher dimensions there are many possible choices. For two dimensions, the most important choices are squares and hexagons, but in higher dimensions, many more choices are available.

Notice that if a region $\mathcal{R}$ tiles $\mathbb{R}^2$, then any scaled version $\alpha\mathcal{R}$ of $\mathcal{R}$ will also tile $\mathbb{R}^2$, and so will any rotation or translation of $\mathcal{R}$.

Consider the performance of a uniform 2D quantizer with a basic cell $\mathcal{R}$ which is centered at the origin $\boldsymbol{0}$. The set of cells, which are assumed to tile the region, are denoted by[6] $\{\mathcal{R}_j; j \in \mathbb{Z}^+\}$ where $\mathcal{R}_j = \boldsymbol{a}_j + \mathcal{R}$ and $\boldsymbol{a}_j$ is the center of the cell $\mathcal{R}_j$. Let $A(\mathcal{R}) = \int_{\mathcal{R}} d\boldsymbol{u}$ be the area of the basic cell. The average pdf in a cell $\mathcal{R}_j$ is given by $\Pr(\mathcal{R}_j)/A(\mathcal{R}_j)$. As before, define $\overline{f}(\boldsymbol{u})$ to be the average pdf over the region $\mathcal{R}_j$ containing $\boldsymbol{u}$. The high-rate assumption is again made, *i.e.*, assume that the region $\mathcal{R}$ is small enough that $f_{\boldsymbol{U}}(\boldsymbol{u}) \approx \overline{f}(\boldsymbol{u})$ for all $\boldsymbol{u}$.

The assumption $f_{\boldsymbol{U}}(\boldsymbol{u}) \approx \overline{f}(\boldsymbol{u})$ implies that the conditional pdf, conditional on $\boldsymbol{u} \in \mathcal{R}_j$ is approximated by

$$f_{\boldsymbol{U}|\mathcal{R}_j}(\boldsymbol{u}) \approx \begin{cases} 1/A(\mathcal{R}), & \boldsymbol{u} \in \mathcal{R}_j; \\ 0, & \boldsymbol{u} \notin \mathcal{R}_j. \end{cases} \tag{3.13}$$

The conditional mean is approximately equal to the center $\boldsymbol{a}_j$ of the region $\mathcal{R}_j$. The mean-squared error per dimension for the basic quantization cell $\mathcal{R}$ centered on 0 is then approximately equal to

$$\text{MSE} \approx \frac{1}{2}\int_{\mathcal{R}} \|\boldsymbol{u}\|^2 \frac{1}{A(\mathcal{R})}\, d\boldsymbol{u}. \tag{3.14}$$

The right side of (3.14) is the MSE for the quantization area $\mathcal{R}$ using a pdf equal to a constant; it will be denoted MSE_c. The quantity $\|\boldsymbol{u}\|$ is the length of the vector u_1, u_2, so that $\|\boldsymbol{u}\|^2 = u_1^2 + u_2^2$. Thus MSE_c can be rewritten as

$$\text{MSE} \approx \text{MSE}_c = \frac{1}{2}\int_{\mathcal{R}} (u_1^2 + u_2^2)\frac{1}{A(\mathcal{R})}\, du_1 du_2. \tag{3.15}$$

MSE_c is measured in units of squared length, just like $A(\mathcal{R})$. Thus the ratio $G(\mathcal{R}) = \text{MSE}_c/A(\mathcal{R})$ is a dimensionless quantity called the normalized second moment. With a little effort, it can be seen that $G(\mathcal{R})$ is invariant to scaling, translation and rotation. $G(\mathcal{R})$ does depend on the shape of the region $\mathcal{R}$, and, as seen below, it is $G(\mathcal{R})$ that determines how well a given shape performs as a quantization region. By expressing

$$\text{MSE}_c = G(\mathcal{R})A(\mathcal{R}),$$

it is seen that the MSE is the product of a shape term and an area term, and these can be chosen independently.

As examples, $G(\mathcal{R})$ is given below for some common shapes.

- Square: For a square Δ on a side, $A(\mathcal{R}) = \Delta^2$. Breaking (3.15) into two terms, we see that each is identical to the scalar case and $\text{MSE}_c = \Delta^2/12$. Thus $G(\text{Square}) = 1/12$.

plane, and equilateral triangles with rotations tile the plane.

[6] $\mathbb{Z}^+$ denotes the set of positive integers, so $\{\mathcal{R}_j; j \in \mathbb{Z}^+\}$ denotes the set of regions in the tiling, numbered in some arbitrary way of no particular interest here.

- Hexagon: View the hexagon as the union of 6 equilateral triangles Δ on a side. Then $A(\mathcal{R}) = 3\sqrt{3}\Delta^2/2$ and $\mathrm{MSE}_c = 5\Delta^2/24$. Thus $G(\text{hexagon}) = 5/(36\sqrt{3})$.
- Circle: For a circle of radius r, $A(\mathcal{R}) = \pi r^2$ and $\mathrm{MSE}_c = r^2/4$ so $G(\text{circle}) = 1/(4\pi)$.

The circle is not an allowable quantization region, since it does not tile the plane. On the other hand, for a given area, this is the shape that minimizes MSE_c. To see this, note that for any other shape, differential areas further from the origin can be moved closer to the origin with a reduction in MSE_c. That is, the circle is the 2D shape that minimizes $G(\mathcal{R})$. This also suggests why $G(\text{Hexagon}) < G(\text{Square})$, since the hexagon is more concentrated around the origin than the square.

Using the high rate approximation for any given tiling, each quantization cell $\mathcal{R}_j$ has the same shape and area and has a conditional pdf which is approximately uniform. Thus MSE_c approximates the MSE for each quantization region and thus approximates the overall MSE.

Next consider the entropy of the quantizer output. The probability that $\boldsymbol{U}$ falls in the region $\mathcal{R}_j$ is

$$p_j = \int_{\mathcal{R}_j} f_{\boldsymbol{U}}(\boldsymbol{u})\, d\boldsymbol{u} \qquad \text{and, for all } \boldsymbol{u} \in \mathcal{R}_j, \quad p_j = \overline{f}(\boldsymbol{u})A(\mathcal{R}).$$

The output of the quantizer is the discrete random symbol V with the pmf p_j for each symbol j. As before, the entropy of $\boldsymbol{V}$ is given by

$$\begin{aligned}
\mathsf{H}[\boldsymbol{V}] &= -\sum_j p_j \log p_j \\
&= -\sum_j \int_{\mathcal{R}_j} f_{\boldsymbol{U}}(\boldsymbol{u}) \log[\overline{f}(\boldsymbol{u})A(\mathcal{R})]\, d\boldsymbol{u} \\
&= -\int f_{\boldsymbol{U}}(\boldsymbol{u})\ [\log \overline{f}(\boldsymbol{u}) + \log A(\mathcal{R})]\, d\boldsymbol{u} \\
&\approx -\int f_{\boldsymbol{U}}(\boldsymbol{u})\ [\log f_{\boldsymbol{U}}(\boldsymbol{u})]\, d\boldsymbol{u} + \log A(\mathcal{R})] \\
&= 2\mathsf{h}[U] - \log A(\mathcal{R}),
\end{aligned}$$

where the high rate approximation $f_{\boldsymbol{U}}(\boldsymbol{u}) \approx \bar{f}(\boldsymbol{u})$ was used. Note that, since $\boldsymbol{U} = U_1U_2$ for iid variables U_1 and U_2, the differential entropy of $\boldsymbol{U}$ is $2\mathsf{h}[U]$.

Again, an efficient uniquely-decodable source code can be used to encode the quantizer output sequence into a bit sequence at an average rate per source symbol of

$$\overline{L} \approx \frac{\mathsf{H}[\boldsymbol{V}]}{2} \approx \mathsf{h}[U] - \frac{1}{2}\log A(\mathcal{R}) \quad \text{bits/symbol.} \tag{3.16}$$

At the receiver, the mean-squared quantization error in reconstructing the original sequence will be approximately equal to the MSE given in (3.14).

We have the following important conclusions for a uniform 2D quantizer under the high-rate approximation:

- Under the high-rate assumption, the rate $\overline{L}$ depends only on the differential entropy $\mathsf{h}[U]$ of the source and the area $A(\mathcal{R})$ of the basic quantization cell $\mathcal{R}$. It does not depend on any other feature of the source pdf $f_U(u)$, and does not depend on the shape of the quantizer region, *i.e.*, it does not depend on the normalized second moment $G(\mathcal{R})$.

- There is a tradeoff between the rate $\overline{L}$ and MSE that is governed by the area $A(\mathcal{R})$. From (3.16), an increase of 1 bit/symbol in rate corresponds to a decrease in $A(\mathcal{R})$ by a factor of 4. From (3.14), this decreases the MSE by a factor of 4, *i.e.*, by 6 dB.

- The ratio $G(\text{Square})/G(\text{Hexagon})$ is equal to $3\sqrt{3}/5 = 1.0392$. This is called the *quantizing gain* of the hexagon over the square. For a given $A(\mathcal{R})$ (and thus a given $\overline{L}$), the MSE for a hexagonal quantizer is smaller than that for a square quantizer (and thus also for a scalar quantizer) by a factor of 1.0392 (0.17 db). This is a disappointingly small gain given the added complexity of 2D and hexagonal regions and suggests that uniform scalar quantizers are good choices at high rates.

3.9 Summary of quantization

Quantization is important both for digitizing a sequence of analog signals and as the middle layer in digitizing analog waveform sources. Uniform scalar quantization is the simplest and often most practical approach to quantization. Before reaching this conclusion, two approaches to optimal scalar quantizers were taken. The first attempted to minimize the expected distortion subject to a fixed number M of quantization regions, and the second attempted to minimize the expected distortion subject to a fixed entropy of the quantized output. Each approach was followed by the extension to vector quantization.

In both approaches, and for both scalar and vector quantization, the emphasis was on minimizing mean square distortion or error (MSE), as opposed to some other distortion measure. As will be seen later, MSE is the natural distortion measure in going from waveforms to sequences of analog values. For specific sources, such as speech, however, MSE is not appropriate. For an introduction to quantization, however, focusing on MSE seems appropriate in building intuition; again, our approach is building understanding through the use of simple models.

The first approach, minimizing MSE with a fixed number of regions, leads to the Lloyd-Max algorithm, which finds a local minimum of MSE. Unfortunately, the local minimum is not necessarily a global minimum, as seen by several examples. For vector quantization, the problem of local (but not global) minima arising from the Lloyd-Max algorithm appears to be the typical case.

The second approach, minimizing MSE with a constraint on the output entropy is also a difficult problem analytically. This is the appropriate approach in a two layer solution where the quantizer is followed by discrete encoding. On the other hand, the first approach is more appropriate when vector quantization is to be used but cannot be followed by fixed-to-variable-length discrete source coding.

High-rate scalar quantization, where the quantization regions can be made sufficiently small so that the probability density in almost constant over each region, leads to a much simpler result when followed by entropy coding. In the limit of high rate, a uniform scalar quantizer minimizes MSE for a given entropy constraint. Moreover, the tradeoff between Minimum MSE and output entropy is the simple univeral curve of Figure 3.9. The source is completely characterized by its differential entropy in this tradeoff. The approximations in this result are analyzed in Exercise 3.6.

Two-dimensional vector quantizatioon under the high-rate approximation with entropy coding leads to a similar result. Using a square quantization region to tile the plane, the tradeoff between MSE per symbol and entropy per symbol is the same as with scalar quantization. Using a hexagonal quantization region to tile the plane reduces the MSE by a factor of 1.0392, which seems hardly worth the trouble. Using non-uniform quantization regions at high rate leads to a lower bound on MSE which is lower than that for the scalar uniform quantizer by a factor of 1.0472, which, even if achievable, is scarcely worth the trouble.

3A Appendix A: Nonuniform scalar quantizers

This appendix shows that the approximate MSE for uniform high-rate scalar quantizers in Section 3.7 provides an approximate lower bound on the MSE for any nonuniform scalar quantizer, again using the high-rate approximation that the pdf of U is constant within each quantization region. This shows that in the high-rate region, there is little reason to further consider nonuniform scalar quantizers.

Consider an arbitrary scalar quantizer for an rv U with a pdf $f_U(u)$. Let Δ_j be the width of the jth quantization interval, *i.e.*, $\Delta_j = |\mathcal{R}_j|$. As before, let $\overline{f}(u)$ be the average pdf within each quantization interval, *i.e.*,

$$\overline{f}(u) = \frac{\int_{\mathcal{R}_j} f_U(u)\, du}{\Delta_j} \qquad \text{for} \quad u \in \mathcal{R}_j.$$

The high-rate approximation is that $f_U(u)$ is approximately constant over each quantization region. Equivalently, $f_U(u) \approx \overline{f}(u)$ for all u. Thus, if region $\mathcal{R}_j$ has width Δ_j, the conditional mean a_j of U over $\mathcal{R}_j$ is approximately the midpoint of the region, and the conditional mean-squared error, MSE$_j$, given $U{\in}\mathcal{R}_j$, is approximately $\Delta_j^2/12$.

Let V be the quantizer output, *i.e.*, the discrete rv such that $V = a_j$ whenever $U \in \mathcal{R}_j$. The probability p_j that $V{=}a_j$ is $p_j = \int_{\mathcal{R}_j} f_U(u)\, du$

The unconditional mean-squared error, *i.e.*. $\mathsf{E}[(U-V)^2]$ is then given by

$$\text{MSE} \approx \sum_j p_j \frac{\Delta_j^2}{12} = \sum_j \int_{\mathcal{R}_j} f_U(u)\, \frac{\Delta_j^2}{12}\, du. \tag{3.17}$$

This can be simplified by defining $\Delta(u) = \Delta_j$ for $u \in \mathcal{R}_j$. Since each u is in $\mathcal{R}_j$ for some j, this defines $\Delta(u)$ for all $u \in \mathbb{R}$. Substituting this in (3.17),

$$\begin{aligned} \text{MSE} &\approx \sum_j \int_{\mathcal{R}_j} f_U(u)\, \frac{\Delta(u)^2}{12}\, du & (3.18)\\ &= \int_{-\infty}^{\infty} f_U(u)\, \frac{\Delta(u)^2}{12}\, du\ . & (3.19) \end{aligned}$$

Next consider the entropy of V. As in (3.8), the following relations are used for p_j

$$p_j = \int_{\mathcal{R}_j} f_U(u)\, du \quad \text{and, for all } u \in \mathcal{R}_j, \quad p_j = \overline{f}(u)\Delta(u).$$

$$\begin{aligned}
\mathsf{H}[V] &= \sum_j -p_j \log p_j \\
&= \sum_j \int_{\mathcal{R}_j} -f_U(u) \log[\,\overline{f}(u)\Delta(u)]\; du && (3.20)\\
&= \int_{-\infty}^{\infty} -f_U(u) \log[\overline{f}(u)\Delta(u)]\; du, && (3.21)
\end{aligned}$$

where the multiple integrals over disjoint regions have been combined into a single integral. The high-rate approximation $f_U(u) \approx \overline{f}(u)$ is next substituted into (3.21).

$$\begin{aligned}
\mathsf{H}[V] &\approx \int_{-\infty}^{\infty} -f_U(u) \log[f_U(u)\Delta(u)]\; du \\
&= \mathsf{h}[U] - \int_{-\infty}^{\infty} f_U(u) \log \Delta(u)\; du. && (3.22)
\end{aligned}$$

Note the similarity of this to (3.11).

The next step is to minimize the mean-squared error subject to a constraint on the entropy $\mathsf{H}[V]$. This is done approximately by minimizing the approximation to MSE in (3.22) subject to the approximation to $\mathsf{H}[V]$ in (3.19). Exercise 3.6 provides some insight into the accuracy of these approximations and their effect on this minimization.

Consider using a Lagrange multiplier to perform the minimization. Since MSE decreases as $\mathsf{H}[V]$ increases, consider minimizing $\text{MSE} + \lambda\mathsf{H}[V]$. As λ increases, MSE will increase and $\mathsf{H}[V]$ decrease in the minimizing solution.

In principle, the minimization should be constrained by the fact that $\Delta(u)$ is constrained to represent the interval sizes for a realizable set of quantization regions. The minimum of $\text{MSE} + \lambda\mathsf{H}[V]$ will be lower bounded by ignoring this constraint. The very nice thing that happens is that this unconstrained lower bound occurs where $\Delta(u)$ is constant. This corresponds to a uniform quantizer, which is clearly realizable. In other words, subject to the high-rate approximation, the lower bound on MSE over all scalar quantizers is equal to the MSE for the uniform scalar quantizer. To see this, use (3.19) and (3.22),

$$\begin{aligned}
\text{MSE} + \lambda\mathsf{H}[V] &\approx \int_{-\infty}^{\infty} f_U(u)\, \frac{\Delta(u)^2}{12}\; du \;+ \lambda\mathsf{h}[U] - \lambda \int_{-\infty}^{\infty} f_U(u) \log \Delta(u)\; du \\
&= \lambda\mathsf{h}[U] + \int_{-\infty}^{\infty} f_U(u) \left\{ \frac{\Delta(u)^2}{12} - \lambda \log \Delta(u) \right\}\; du. && (3.23)
\end{aligned}$$

This is minimized over all choices of $\Delta(u) > 0$ by simply minimizing the expression inside the braces for each real value of u. That is, for each u, differentiate the quantity inside the braces with respect to $\Delta(u)$, getting $\Delta(u)/6 - \lambda(\log e)/\Delta(u)$. Setting the derivative equal to 0, it is seen that $\Delta(u) = \sqrt{\lambda(\log e)/6}$. By taking the second derivative, it can be seen that this solution actually minimizes the integrand for each u. The only important thing here is that the minimizing $\Delta(u)$ is independent of u. This means that the approximation of MSE is minimized, subject to a constraint on the approximation of $\mathsf{H}[V]$, by the use of a uniform quantizer.

The next question is the meaning of minimizing an approximation to something subject to a constraint which itself is an approximation. From Exercise 3.6, it is seen that both the approximation to MSE and that to $\mathsf{H}[V]$ are good approximations for small Δ, *i.e.*, for high-rate. For any given high-rate nonuniform quantizer then, consider plotting MSE and $\mathsf{H}[V]$ on

Figure 3.9. The corresponding approximate values of MSE and $\mathsf{H}[V]$ are then close to the plotted value (with some small difference both in the ordinate and abscissa). These approximate values, however, lie above the approximate values plotted in Figure 3.9 for the scalar quantizer. Thus, in this sense, the performance curve of MSE versus $\mathsf{H}[V]$ for the approximation to the scalar quantizer either lies below or close to the points for any nonuniform quantizer.

In summary, it has been shown that for large $\mathsf{H}[V]$ (*i.e.*, high-rate quantization), a uniform scalar quantizer approximately minimizes MSE subject to the entropy constraint. There is little reason to use nonuniform scalar quantizers (except perhaps at low rate). Furthermore the MSE performance at high-rate can be easily approximated and depends only on $\mathsf{h}[U]$ and the constraint on $\mathsf{H}[V]$.

3B Appendix B: Nonuniform 2D quantizers

For completeness, the performance of nonuniform 2D quantizers is now analyzed; the analysis is very similar to that of nonuniform scalar quantizers. Consider an arbitrary set of quantization intervals $\{\mathcal{R}_j\}$. Let $A(\mathcal{R}_j)$ and MSE_j be the area and mean-squared error per dimension respectively of $\mathcal{R}_j$, *i.e.*,

$$A(\mathcal{R}_j) = \int_{\mathcal{R}_j} d\boldsymbol{u}\ ; \qquad \mathrm{MSE}_j = \frac{1}{2}\int_{\mathcal{R}_j} \frac{\|\boldsymbol{u}-\boldsymbol{a}_j\|^2}{A(\mathcal{R}_j)}\ d\boldsymbol{u},$$

where $\boldsymbol{a}_j$ is the mean of $\mathcal{R}_j$. For each region $\mathcal{R}_j$ and each $\boldsymbol{u} \in \mathcal{R}_j$, let $\overline{f}(\boldsymbol{u}) = \Pr(\mathcal{R}_j)/A(\mathcal{R}_j)$ be the average pdf in $\mathcal{R}_j$. Then

$$p_j = \int_{\mathcal{R}_j} f_{\boldsymbol{U}}(\boldsymbol{u})\ d\boldsymbol{u} = \overline{f}(\boldsymbol{u})A(\mathcal{R}_j).$$

The unconditioned mean-squared error is then

$$\mathrm{MSE} = \sum_j p_j\ \mathrm{MSE}_j.$$

Let $A(\boldsymbol{u}) = A(\mathcal{R}_j)$ and $\mathrm{MSE}(\boldsymbol{u}) = \mathrm{MSE}_j$ for $\boldsymbol{u} \in A_j$. Then,

$$\mathrm{MSE} = \int f_{\boldsymbol{U}}(\boldsymbol{u})\ \mathrm{MSE}(\boldsymbol{u})\ d\boldsymbol{u}. \tag{3.24}$$

Similarly,

$$\begin{aligned}
\mathsf{H}[V] &= \sum_j -p_j \log p_j \\
&= \int -f_{\boldsymbol{U}}(\boldsymbol{u}) \log[\overline{f}(\boldsymbol{u})A(\boldsymbol{u})]\ d\boldsymbol{u} \\
&\approx \int -f_{\boldsymbol{U}}(\boldsymbol{u}) \log[f_{\boldsymbol{U}}(\boldsymbol{u})A(\boldsymbol{u})]\ d\boldsymbol{u} && (3.25)\\
&= 2\mathsf{h}[U] - \int f_{\boldsymbol{U}}(\boldsymbol{u}) \log[A(\boldsymbol{u})]\ d\boldsymbol{u}. && (3.26)
\end{aligned}$$

A Lagrange multiplier can again be used to solve for the optimum quantization regions under the high-rate approximation. In particular, from (3.24) and (3.26),

$$\mathrm{MSE} + \lambda \mathsf{H}[V] \approx \lambda 2\mathsf{h}[U] + \int_{\mathbb{R}^2} f_{\boldsymbol{U}}(\boldsymbol{u}) \, \{\mathrm{MSE}(\boldsymbol{u}) - \lambda \log A(\boldsymbol{u})\} \; du. \qquad (3.27)$$

Since each quantization area can be different, the quantization regions need not have geometric shapes whose translates tile the plane. As pointed out earlier, however, the shape that minimizes MSE_c for a given quantization area is a circle. Therefore the MSE can be lower bounded in the Lagrange multiplier by using this shape. Replacing $\mathrm{MSE}(\boldsymbol{u})$ by $A(\boldsymbol{u})/(4\pi)$ in (3.27),

$$\mathrm{MSE} + \lambda \mathsf{H}[V] \approx 2\lambda \mathsf{h}[U] + \int_{\mathbb{R}^2} f_{\boldsymbol{U}}(\boldsymbol{u}) \left\{ \frac{A(\boldsymbol{u})}{4\pi} - \lambda \log A(\boldsymbol{u}) \right\} \; du. \qquad (3.28)$$

Optimizing for each $\boldsymbol{u}$ separately, $A(\boldsymbol{u}) = 4\pi\lambda \log e$. The optimum is achieved where the same size circle is used for each point $\boldsymbol{u}$ (independent of the probability density). This is unrealizable, but still provides a lower bound on the MSE for any given $\mathsf{H}[V]$ in the high-rate region. The reduction in MSE over the square region is $\pi/3 = 1.0472$ (0.2 db). It appears that the uniform quantizer with hexagonal shape is optimal, but this figure of $\pi/3$ provides a simple bound to the possible gain with 2D quantizers. Either way, the improvement by going to two dimensions is small.

The same sort of analysis can be carried out for n dimensional quantizers. In place of using a circle as a lower bound, one now uses an n dimensional sphere. As n increases, the resulting lower bound to MSE approaches a gain of $\pi e/6 = 1.4233$ (1.53 db) over the scalar quantizer. It is known from a fundamental result in information theory that this gain can be approached arbitrarily closely as $n \to \infty$.

3.E Exercises

3.1. Let U be an analog rv (rv) uniformly distributed between -1 and 1.

(a) Find the three-bit ($M = 8$) quantizer that minimizes the mean-squared error.

(b) Argue that your quantizer satisfies the necessary conditions for optimality.

(c) Show that the quantizer is unique in the sense that no other 3-bit quantizer satisfies the necessary conditions for optimality.

3.2. Consider a discrete-time, analog source *with memory*, *i.e.*, $U_1, U_2, \ldots$ are dependent rv's. Assume that each U_m is uniformly distributed between 0 and 1 but that $U_{2n} = U_{2n-1}$ for each $n \geq 1$. Assume that $\{U_{2n}\}_{n=1}^{\infty}$ are independent.

(a) Find the one-bit ($M = 2$) scalar quantizer that minimizes the mean-squared error.

(b) Find the mean-squared error for the quantizer that you have found in (a).

(c) Find the one-bit-per-symbol ($M = 4$) two-dimensional vector quantizer that minimizes the MSE.

(d) Plot the two-dimensional regions and representation points for both your scalar quantizer in part (a) and your vector quantizer in part (c).

3.3. Consider a binary scalar quantizer that partitions the reals $\mathbb{R}$ into two subsets, $(-\infty, b]$ and (b, ∞) and then represents $(-\infty, b]$ by $a_1 \in \mathbb{R}$ and (b, ∞) by $a_2 \in \mathbb{R}$. This quantizer is used on each letter U_n of a sequence $\cdots, U_{-1}, U_0, U_1, \cdots$ of iid random variables, each having the probability density $f(u)$. Assume throughout this exercise that $f(u)$ is symmetric, *i.e.*, that $f(u) = f(-u)$ for all $u \geq 0$.

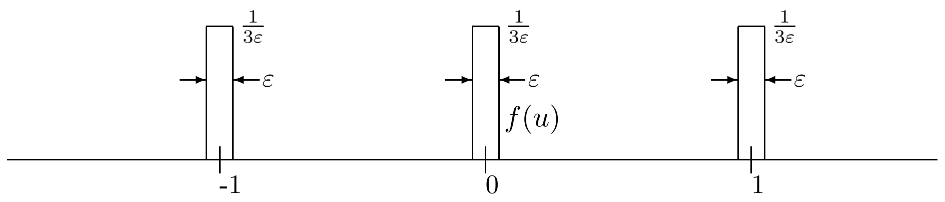
(a) Given the representation levels a_1 and $a_2 > a_1$, how should b be chosen to minimize the mean square distortion in the quantization? Assume that $f(u) > 0$ for $a_1 \leq u \leq a_2$ and explain why this assumption is relevant.

(b) Given $b \geq 0$, find the values of a_1 and a_2 that minimize the mean square distortion. Give both answers in terms of the two functions $Q(x) = \int_x^{\infty} f(u)\, du$ and $y(x) = \int_x^{\infty} u f(u)\, du$.

(c) Show that for $b = 0$, the minimizing values of a_1 and a_2 satisfy $a_1 = -a_2$.

(d) Show that the choice of b, a_1, and a_2 in part (c) satisfies the Lloyd-Max conditions for minimum mean square distortion.

(e) Consider the particular symmetric density below



Find all sets of triples, $\{b, a_1, a_2\}$ that satisfy the Lloyd-Max conditions and evaluate the MSE for each. You are welcome in your calculation to replace each region of non-zero probability density above with an impulse *i.e.*, $f(u) = \frac{1}{3}[\delta(-1) + \delta(0) + \delta(1)]$, but you should use the figure above to resolve the ambiguity about regions that occurs when b is -1, 0, or +1.

(e) Give the MSE for each of your solutions above (in the limit of $\varepsilon \to 0$). Which of your solutions minimizes the MSE?

3.4. In Section 3.4, we partly analyzed a minimum-MSE quantizer for a pdf in which $f_U(u) = f_1$ over an interval of size L_1, $f_U(u) = f_2$ over an interval of size L_2 and $f_U(u) = 0$ elsewhere. Let M be the total number of representation points to be used, with M_1 in the first interval and $M_2 = M - M_1$ in the second. Assume (from symmetry) that the quantization intervals are of equal size $\Delta_1 = L_1/M_1$ in interval 1 and of equal size $\Delta_2 = L_2/M_2$ in interval 2. Assume that M is very large, so that we can approximately minimize the MSE over M_1, M_2 without an integer constraint on M_1, M_2 (that is, assume that M_1, M_2 can be arbitrary real numbers).

(a) Show that the MSE is minimized if $\Delta_1 f_1^{1/3} = \Delta_2 f_2^{1/3}$, *i.e.*, the quantization interval sizes are inversely proportional to the cube root of the density. [Hint: Use a Lagrange multiplier to perform the minimization. That is, to minimize a function $\mathrm{MSE}(\Delta_1, \Delta_2)$ subject to a constraint $M = f(\Delta_1, \Delta_2)$, first minimize $\mathrm{MSE}(\Delta_1, \Delta_2) + \lambda f(\Delta_1, \Delta_2)$ without the constraint, and, second, choose λ so that the solution meets the constraint.]

(b) Show that the minimum MSE under the above assumption is given by

$$\mathrm{MSE} = \frac{\left(L_1 f_1^{1/3} + L_2 f_2^{1/3}\right)^3}{12M^2}.$$

(c) Assume that the Lloyd-Max algorithm is started with $0 < M_1 < M$ representation points in the first interval and $M_2 = M - M_1$ points in the second interval. Explain where the Lloyd-Max algorithm converges for this starting point. Assume from here on that the distance between the two intervals is very large.

(d) Redo part (c) under the assumption that the Lloyd-Max algorithm is started with $0 < M_1 \le M - 2$ representation points in the first interval, one point between the two intervals, and the remaining points in the second interval.

(e) Express the exact minimum MSE as a minimum over $M - 1$ possibilities, with one term for each choice of $0 < M_1 < M$ (assume there are no representation points between the two intervals).

(f) Now consider an arbitrary choice of Δ_1 and Δ_2 (with no constraint on M). Show that the entropy of the set of quantization points is

$$H(V) = -f_1 L_1 \log(f_1 \Delta_1) - f_2 L_2 \log(f_2 \Delta_2).$$

(g) Show that if we minimize the MSE subject to a constraint on this entropy (ignoring the integer constraint on quantization levels), then $\Delta_1 = \Delta_2$.

3.5. Assume that a continuous valued rv Z has a probability density that is 0 except over the interval $[-A, +A]$. Show that the differential entropy $h(Z)$ is upper bounded by $1 + \log_2 A$.

(b) Show that $h(Z) = 1 + \log_2 A$ if and only if Z is uniformly distributed between $-A$ and $+A$.

3.6. Let $f_U(u) = 1/2 + u$ for $0 < u \le 1$ and $f_U(u) = 0$ elsewhere.

(a) For $\Delta < 1$, consider a quantization region $\mathcal{R} = (x, x + \Delta]$ for $0 < x \le 1 - \Delta$. Find the conditional mean of U conditional on $U \in \mathcal{R}$.

(b) Find the conditional mean-squared error (MSE) of U conditional on $U \in \mathcal{R}$. Show that, as Δ goes to 0, the difference between the MSE and the approximation $\Delta^2/12$ goes to 0 as Δ^4.

(c) For any given Δ such that $1/\Delta = M$, M a positive integer, let $\{\mathcal{R}_j = ((j-1)\Delta, j\Delta]\}$ be the set of regions for a uniform scalar quantizer with M quantization intervals. Show that the difference between $\mathsf{h}[U] - \log\Delta$ and $\mathsf{H}[V]$ as given (3.10) is

$$\mathsf{h}[U] - \log\Delta - \mathsf{H}[V] = \int_0^1 f_U(u)\log[\overline{f}(u)/f_U(u)]\;du.$$

(d) Show that the difference in (3.6) is nonnegative. Hint: use the inequality $\ln x \le x - 1$. Note that your argument does not depend on the particular choice of $f_U(u)$.

(e) Show that the difference $\mathsf{h}[U] - \log\Delta - \mathsf{H}[V]$ goes to 0 as Δ^2 as $\Delta \to 0$. Hint: Use the approximation $\ln x \approx (x-1) - (x-1)^2/2$, which is the second-order Taylor series expansion of $\ln x$ around $x = 1$.

The major error in the high-rate approximation for small Δ and smooth $f_U(u)$ is due to the slope of $f_U(u)$. Your results here show that this linear term is insignificant for both the approximation of MSE and for the approximation of $\mathsf{H}[V]$. More work is required to validate the approximation in regions where $f_U(u)$ goes to 0.

3.7. (Example where $\mathsf{h}(U)$ is infinite. Let $f_U(u)$ be given by

$$f_U(u) = \begin{cases} \frac{1}{u(\ln u)^2} & \text{for} \quad u \ge e \\ 0 & \text{for} \quad u < e, \end{cases}$$

(a) Show that $f_U(u)$ is non-negative and integrates to 1.

(b) Show that $\mathsf{h}(U)$ is infinite.

(c) Show that a uniform scalar quantizer for this source with any separation Δ $(0 < \Delta < \infty)$ has infinite entropy. Hint: Use the approach in Exercise 3.6, parts (c, d.)

3.8. Consider a discrete source U with a finite alphabet of N real numbers, $r_1 < r_2 < \cdots < r_N$ with the pmf $p_1 > 0, \dots, p_N > 0$. The set $\{r_1, \dots, r_N\}$ is to be quantized into a smaller set of $M < N$ representation points $a_1 < a_2 < \cdots < a_M$.

(a) Let $\mathcal{R}_1, \mathcal{R}_2, \dots, \mathcal{R}_M$ be a given set of quantization intervals with $\mathcal{R}_1 = (-\infty, b_1], \mathcal{R}_2 = (b_1, b_2], \dots, \mathcal{R}_M = (b_{M-1}, \infty)$. Assume that at least one source value r_i is in $\mathcal{R}_j$ for each j, $1 \le j \le M$ and give a necessary condition on the representation points $\{a_j\}$ to achieve minimum MSE.

(b) For a given set of representation points $a_1, \dots, a_M$ assume that no symbol r_i lies exactly halfway between two neighboring a_i, *i.e.*, that $r_i \ne \frac{a_j + a_{j+1}}{2}$ for all i, j. For each r_i, find the interval $\mathcal{R}_j$ (and more specifically the representation point a_j) that r_i must be mapped into to minimize MSE. Note that it is not necessary to place the boundary b_j between $\mathcal{R}_j$ and $\mathcal{R}_{j+1}$ at $b_j = [a_j + a_{j+1}]/2$ since there is no probability in the immediate vicinity of $[a_j + a_{j+1}]/2$.

(c) For the given representation points, $a_1, \dots, a_M$, now assume that $r_i = \frac{a_j + a_{j+1}}{2}$ for some source symbol r_i and some j. Show that the MSE is the same whether r_i is mapped into a_j or into a_{j+1}.

(d) For the assumption in part c), show that the set $\{a_j\}$ cannot possibly achieve minimum MSE. Hint: Look at the optimal choice of a_j and a_{j+1} for each of the two cases of part c).

3.9. Assume an iid discrete-time analog source $U_1, U_2, \cdots$ and consider a scalar quantizer that satisfies the Lloyd-Max conditions. Show that the rectangular 2-dimensional quantizer based on this scalar quantizer also satisfies the Lloyd-Max conditions.

3.10. (a) Consider a square two dimensional quantization region $\mathcal{R}$ defined by $-\frac{\Delta}{2} \leq u_1 \leq \frac{\Delta}{2}$ and $-\frac{\Delta}{2} \leq u_2 \leq \frac{\Delta}{2}$. Find MSE_c as defined in (3.15) and show that it's proportional to Δ^2.

(b) Repeat part (a) with Δ replaced by $a\Delta$. Show that $\mathrm{MSE}_c/A(\mathcal{R})$ (where $A(\mathcal{R})$ is now the area of the scaled region) is unchanged.

(c) Explain why this invariance to scaling of $\mathrm{MSE}_c/A(\mathcal{R})$ is valid for any two dimensional region.

Bibliography

[1] D. Bertsekas and R. G. Gallager, *Data Networks , 2nd ed*, Prentice Hall 1992.

[2] D. Bertsekas and J. Tsitsiklis, *An Introduction to Probability Theory*, Athena.

[3] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley, 1991. A modern text on information theory.

[4] R.G. Gallager, *Information Theory and Reliable Communication*, Wiley, 1968. A Classic treatment of Information Theory.

[5] R.G. Gallager, *Discrete Stochastic Processes*, Kluwer 1996.

[6] L. G. Kraft, "A device for quantizing, grouping, and coding amplitude modulated pulses," M.S. Thesis, Dept. of E.E., MIT, Cambridge, MA.

[7] text or review article on speech coding.

[8] R. V. L. Hartley,"Transmission of information," Bell System Technical Journal, July 1928, p. 535.

[9] D. A. Huffman, "A method for the construction of minimum redundancy codes," Proc. IRE, 40, 1098-1101, 1952.

[10] S. P. Lloyd, "Least squares quantization in PCM," IEEE Trans. Inform. Theory, vol. IT-28, #2, 129-136, 1982.

[11] J. Max, "Quantization for minimum distortion," IRE Trans. Inform. Theory, vol IT-6, #2, 7-12, March 1960.

[12] H. Nyquist, "Certain Topics in Telegraph Transmission theory," Trans. AIEE 47: pp. 627-644, 1928.

[13] J.G. Proakis, *Digital Communications*, 4th ed., McGraw-Hill, 2000. This covers a larger variety of systems than the text, but in less depth.

[14] J.G. Proakis and M. Salehi, *Communication Systems Engineering*, Prentice Hall, 1994. This is an undergraduate version of the above text.

[15] S. Ross, *A First Course in Probability*, 4th Ed., McMillan & Co., 1994.

[16] S. Ross, *Stochastic Processes*, Wiley and sons, 1983.

[17] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, **27**, 1948, 379-423 and 623-656.

[18] C. E. Shannon, "The zero-error capacity of a noisy channel," *IRE Trans. Inform. Theary*, vol. IT-2; 8-19, 1956.

[19] S.G. Wilson, *Digital Modulation and Coding*, Prentice-Hall, 1996. Another general text covering mostly the material of the last two thirds of this text.

[20] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, Wiley, 1965. This text first developed the signal space view of communication.

[21] D. Tse and P. Viswanath, *Fundamentals of Wireless Communication*, Cambridge Press, 2005. An excellent treatment of the Principles and modern practice of wireless communication.

[22] A. Wyner and J. Ziv, "The sliding window Lemper-Ziv algorithm is asymptotically optimal," *Proc. IEEE*, June 1994, pp. 872-877.

[23] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. IT*, May 1977, pp. 337-343.

[24] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. IT*, Sept. 1978, pp. 530-536.