1 Coding and Capacity

1.1 Digital Data Communication and Storage

Digital communication systems are ubiquitous in our daily lives. The most obvious examples include cell phones, digital television via satellite or cable, digital radio, wireless internet connections via Wi-Fi and WiMax, and wired internet connection via cable modem. Additional examples include digital data-storage devices, including magnetic (“hard”) disk drives, magnetic tape drives, optical disk drives (e.g., CD, DVD, blu-ray), and flash drives. In the case of data-storage, information is communicated from one point in time to another rather than one point in space to another. Each of these examples, while widely different in implementation details, generally fits into a common digital communication framework first established by C. Shannon in his 1948 seminal paper, *A Mathematical Theory of Communication* [1]. This framework is depicted in Figure 1.1, whose various components are described as follows.

Source and *user* (or *sink*). The information source may be originally in analog form (e.g., speech or music) and then later digitized, or it may be originally in digital form (e.g., computer files). We generally think of its output as a sequence of bits, which follow a probabilistic model. The user of the information may be a person, a computer, or some other electronic device.

Source encoder and *source decoder*. The encoder is a processor that converts the information source bit sequence into an alternative bit sequence with a more efficient representation of the information, i.e., with fewer bits. Hence, this operation is often called *compression*. Depending on the source, the compression can be *lossless* (e.g., for computer data files) or *lossy* (e.g., for video, still images, or music, where the loss can be made to be imperceptible or acceptable). The source decoder is the encoder’s counterpart which recovers the source sequence exactly, in the case of lossless compression, or approximately, in the case of lossy compression, from the encoder’s output sequence.

Channel encoder and *channel decoder*. The role of the channel encoder is to protect the bits to be transmitted over a channel subject to noise, distortion, and interference. It does so by converting its input into an alternate sequence possessing redundancy, whose role is to provide immunity from the various

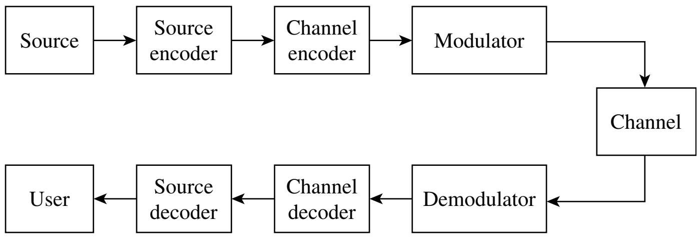

Figure 1.1 Basic digital communication- (or storage-) system block diagram due to Shannon.

channel impairments. The ratio of the number of bits that enter the channel encoder to the number that depart from it is called the *code rate*, denoted by R, with $0 < R < 1$. For example, if a 1000-bit codeword is assigned to each 500-bit information word, $R = 1/2$, and there are 500 redundant bits in each codeword. The function of the channel decoder is to recover from the channel output the input to the channel encoder (i.e., the compressed sequence) in spite of the presence of noise, distortion, and interference in the received word.

Modulator and *demodulator*. The modulator converts the channel-encoder output bit stream into a form that is appropriate for the channel. For example, for a wireless communication channel, the bit stream must be represented by a high-frequency signal to facilitate transmission with an antenna of reasonable size. Another example is a so-called modulation code used in data storage. The modulation encoder output might be a sequence that satisfies a certain runlength constraint (runs of like symbols, for example) or a certain spectral constraint (the output contains a null at DC, for example). The demodulator is the modulator's counterpart which recovers the modulator input sequence from the modulator output sequence.

Channel. The channel is the physical medium through which the modulator output is conveyed, or by which it is stored. Our experience teaches us that the channel adds noise and often interference from other signals, on top of the signal distortion that is ever-present, albeit sometimes to a minor degree. For our purposes, the channel is modeled as a probabilistic device, and examples will be presented below. Physically, the channel can included antennas, amplifiers, and filters, both at the transmitter and at the receiver at the ends of the system. For a hard-disk drive, the channel would include the write head, the magnetic medium, the read head, the read amplifier and filter, and so on.

Following Shannon's model, Figure 1.1 does not include such blocks as encryption/decryption, symbol-timing recovery, and scrambling. The first of these is optional and the other two are assumed to be ideal and accounted for in the probabilistic channel models. On the basis of such a model, Shannon showed that a channel can be characterized by a parameter, C, called the *channel capacity*, which is a measure of how much information the channel can convey, much like the capacity of a plumbing system to convey water. Although C can be represented in several different units, in the context of the channel code rate R, which has the units information bits per channel bit, Shannon showed that codes exist that provide arbitrarily reliable communication provided that the code rate satisfies $R < C$. He further showed that, conversely, if $R > C$, there exists no code that provides reliable communication.

Later in this chapter, we review the capacity formulas for a number of commonly studied channels for reference in subsequent chapters. Prior to that, however, we give an overview of various channel-coding approaches for error avoidance in data-transmission and data-storage scenarios. We then introduce the first channel code invented, the (7,4) Hamming code, by which we mean a code that assigns to each 4-bit information word a 7-bit codeword according to a recipe specified by R. Hamming in 1950 [2]. This will introduce to the novice some of the elements of channel codes and will serve as a launching point for subsequent chapters. After the introduction to the (7,4) Hamming code, we present code- and decoder-design criteria and code-performance measures, all of which are used throughout this book.

1.2 Channel-Coding Overview

The large number of coding techniques for error prevention may be partitioned into the set of automatic request-for-repeat (ARQ) schemes and the set of forward-error-correction (FEC) schemes. In ARQ schemes, the role of the code is simply to reliably detect whether or not the received word (e.g., received packet) contains one or more errors. In the event a received word does contain one or more errors, a request for retransmission of the same word is sent out from the receiver back to the transmitter. The codes in this case are said to be *error-detection codes*. In FEC schemes, the code is endowed with characteristics that permit error correction through an appropriately devised decoding algorithm. The codes for this approach are said to be *error-correction codes*, or sometimes *error-control codes*. There also exist *hybrid FEC/ARQ schemes* in which a request for retransmission occurs if the decoder fails to correct the errors incurred over the channel and detects this fact. Note that this is a natural approach for data-storage systems: if the FEC decoder fails, an attempt to re-read the data is made. The codes in this case are said to be *error-detection-and-correction codes*.

The basic ARQ schemes can broadly be subdivided into the following protocols. First is the *stop-and-wait ARQ* scheme in which the transmitter sends a codeword (or encoded packet) and remains idle until the ACK/NAK status signal is returned from the receiver. If a positive acknowledgment (ACK) is returned, a new packet is sent; otherwise, if a negative acknowledgment (NAK) is returned, the current packet, which was stored in a buffer, is retransmitted. The stop-and-wait method is inherently inefficient due to the idle time spent waiting for confirmation.

In *go-back-N ARQ*, the idle time is eliminated by continually sending packets while waiting for confirmations. If a packet is negatively acknowledged, that packet and the $N-1$ subsequent packets sent during the round-trip delay are retransmitted. Note that this preserves the ordering of packets at the receiver.

In *selective-repeat ARQ*, packets are continually transmitted as in go-back-N ARQ, except only the packet corresponding to the NAK message is retransmitted. (The packets have "headers," which effectively number the information block for identification.) Observe that, because only one packet is retransmitted rather than N, the throughput of accepted packets is increased with selective-repeat ARQ relative to go-back-N ARQ. However, there is the added requirement of ample buffer space at the receiver to allow re-ordering of the blocks.

In *incremental-redundancy ARQ*, upon receiving a NAK message for a given packet, the transmitter transmits additional redundancy to the receiver. This additional redundancy is used by the decoder together with the originally received packet in a second attempt to recover the original data. This sequence of steps – NAK, additional redundancy, re-decode – can be repeated a number of times until the data are recovered or the packet is declared lost.

While ARQ schemes are very important, this book deals exclusively with FEC schemes. However, although the emphasis is on FEC, each of the FEC codes introduced can be used in a hybrid FEC/ARQ scheme where the code is used for both correction and detection. There exist many FEC schemes, employing both linear and nonlinear codes, although virtually all codes used in practice can be characterized as linear or linear at their core. Although the concept will be elucidated in Chapter 3, a *linear code* is one for which any sum of codewords is another codeword in the code. Linear codes are traditionally partitioned to the set of block codes and convolutional, or trellis-based, codes, although the turbo codes of Chapter 7 can be seen to be a hybrid of the two. Among the linear block codes are the cyclic and quasi-cyclic codes (defined in Chapter 3), both of which have more algebraic structure than do standard linear block codes. Also, we have been tacitly assuming binary codes, that is, codes whose code symbols are either 0 or 1. However, codes whose symbols are taken from a larger alphabet (e.g., 8-bit ASCII characters or 1000-bit packets) are possible, as described in Chapters 3 and 14.

This book will provide many examples of each of these code types, including nonbinary codes, and their decoders. For now, we introduce the first FEC code, due to Hamming [2], which provides a good introduction to the field of channel codes.

1.3 Channel-Code Archetype: The (7,4) Hamming Code

The (7,4) Hamming code serves as an excellent channel-code prototype since it contains most of the properties of more practical codes. As indicated by the notation (7,4), the codeword length is $n = 7$ and the data word length is $k = 4$, so

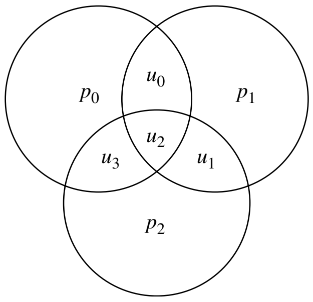

Figure 1.2 Venn-diagram representation of (7,4) Hamming-code encoding and decoding rules.

the code rate is $R = 4/7$. As shown by R. McEliece, the Hamming code is easily described by the simple Venn diagram in Figure 1.2. In the diagram, the information word is represented by the vector $\mathbf{u} = (u_0, u_1, u_2, u_3)$ and the redundant bits (called *parity bits*) are represented by the parity vector $\mathbf{p} = (p_0, p_1, p_2)$. The codeword (also, code vector) is then given by the concatenation of $\mathbf{u}$ and $\mathbf{p}$:

$$\mathbf{v} = (\mathbf{u}\ \mathbf{p}) = (u_0, u_1, u_2, u_3, p_0, p_1, p_2) = (v_0, v_1, v_2, v_3, v_4, v_5, v_6).$$

The encoding rule is trivial: the parity bits are chosen so that each circle has an even number of 1s, i.e., the sum of bits in each circle is 0 modulo 2. From this encoding rule, we may write

$$\begin{aligned} p_0 &= u_0 + u_2 + u_3, \\ p_1 &= u_0 + u_1 + u_2, \\ p_2 &= u_1 + u_2 + u_3, \end{aligned} \tag{1.1}$$

from which the 16 codewords are easily found:

0000 000	1000 110	0010 111
1111 111	0100 011	1001 011
	1010 001	1100 101
	1101 000	1110 010
	0110 100	0111 001
	0011 010	1011 100
	0001 101	0101 110

As an example encoding, consider the third codeword in the middle column, (1010 001), for which the data word is $\mathbf{u} = (u_0, u_1, u_2, u_3) = (1, 0, 1, 0)$. Then,

$$\begin{aligned} p_0 &= 1 + 1 + 0 = 0, \\ p_1 &= 1 + 0 + 1 = 0, \\ p_2 &= 0 + 1 + 0 = 1, \end{aligned}$$

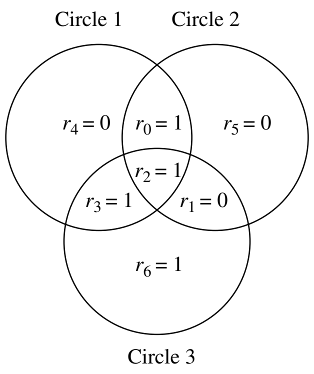

Figure 1.3 Venn-diagram setup for the Hamming decoding example.

yielding $\mathbf{v} = (\mathbf{u}\ \mathbf{p}) = (1010\ 001)$. Observe that this code is linear because the sum of any two codewords yields a codeword. Note also that this code is cyclic: a cyclic shift of any codeword, rightward or leftward, gives another codeword.

Suppose now that $\mathbf{v} = (1010\ 001)$ is transmitted, but $\mathbf{r} = (1011\ 001)$ is received. That is, the channel has converted the 0 in code bit v_3 into a 1. The Venn diagram of Figure 1.3 can be used to decode $\mathbf{r}$ and correct the error. Note that Circle 2 in the figure has an even number of 1s in it, but Circles 1 and 3 do not. Thus, because the code rules are not satisfied by the bits in $\mathbf{r}$, we know that $\mathbf{r}$ contains one or more errors. Because a single error is more likely than two or more errors for most practical channels, we assume that $\mathbf{r}$ contains a single error. Then, the error must be in the intersection of Circles 1 and 3. However, $r_2 = 1$ in the intersection cannot be in error because it is in Circle 2 whose rule is satisfied. Thus, it must be $r_3 = 1$ in the intersection that is in error. Thus, v_3 must be 0 rather than the 1 shown in Figure 1.3 for r_3. In conclusion, the decoded codeword is $\hat{\mathbf{v}} = (1010\ 001)$, from which the decoded data $\hat{\mathbf{u}} = (1010)$ may be recovered.

It can be shown (see Chapter 3) that this particular single error is not special and that, independently of the codeword transmitted, all seven single errors are correctable and no error patterns with more than one error are correctable. The novice might ask what characteristic of these 16 codewords endows them with the ability to correct all single-errors. This is easily explained using the concept of the *Hamming distance* $d_{\mathrm{H}}(\mathbf{x}, \mathbf{x}')$ between two length-n words $\mathbf{x}$ and $\mathbf{x}'$, which is the number of locations in which they disagree. Thus, $d_{\mathrm{H}}(1000\ 110, 0010\ 111) = 3$. It can be shown, either exhaustively or using the principles developed in Chapter 3, that $d_{\mathrm{H}}(\mathbf{v}, \mathbf{v}') \geq 3$ for any two different codewords $\mathbf{v}$ and $\mathbf{v}'$ in the Hamming code. We say that the code's *minimum distance* is therefore $d_{\min} = 3$. Because $d_{\min} = 3$, a single error in some transmitted codeword $\mathbf{v}$ yields a received vector $\mathbf{r}$ that is closer to $\mathbf{v}$, in the sense of Hamming distance, than any other codeword. It is for this reason that all single errors are correctable.

Generalizations of the Venn-diagram code description for the more complex codes used in applications are presented in Chapter 3 and subsequent chapters. In the chapters to follow, we will revisit the Hamming code a number of times, particularly in the problems. We will see how to reformulate encoding so that it employs a so-called generator matrix or, better, a simple shift-register circuit. We will also see how to reformulate decoding so that it employs a so-called parity-check matrix, and we will see many different decoding algorithms. Further, we will see applications of codes to a variety of channels, particularly ones introduced in the next section. Finally, we will see that a "good code" generally has the following properties: it is easy to encode, it is easy to decode, it a has large $d_{\min}$, and/or the number of codewords at the distance $d_{\min}$ from any other codeword is small. We will see many examples of good codes in this book, and of their construction, their encoding, and their decoding.

1.4 Design Criteria and Performance Measures

Although there exist many channel models, it is usual to start with the two most frequently encountered memoryless channels: the binary symmetric channel (BSC) and the binary-input additive white-Gaussian-noise channel (BI-AWGNC). Examination of the BSC and BI-AWGNC illuminates many of the salient features of code and decoder design and code performance. For the sake of uniformity, for both channels, we denote the ith channel input by x_i and the ith channel output by y_i. Given channel input $x_i = v_i \in \{0,1\}$ and channel output $y_i \in \{0,1\}$, the BSC is completely characterized by the channel transition probabilities $P(y_i|x_i)$ given by

$$P(y_i = 1|x_i = 0) = P(y_i = 0|x_i = 1) = \varepsilon,$$
$$P(y_i = 1|x_i = 1) = P(y_i = 0|x_i = 0) = 1 - \varepsilon,$$

where ε is called the *crossover probability*. For the BI-AWGNC, the code bits are mapped to the channel inputs as $x_i = (-1)^{v_i} \in \{\pm 1\}$ so that $x_i = +1$ when $v_i = 0$. The BI-AWGNC is then completely characterized by the channel transition probability density function (pdf) $p(y_i|x_i)$ given by

$$p(y_i|x_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\Big[-(y_i - x_i)^2 / (2\sigma^2)\Big],$$

where σ^2 is the variance of the zero-mean Gaussian noise sample n_i that the channel adds to the transmitted value x_i (so that $y_i = x_i + n_i$). As a consequence of its memorylessness, we have for the BSC

$$P(\mathbf{y}|\mathbf{x}) = \prod_i P(y_i|x_i)\,, \tag{1.2}$$

where $\mathbf{y} = [y_1, y_2, y_3, \ldots]$ and $\mathbf{x} = [x_1, x_2, x_3, \ldots]$. A similar expression exists for the BI-AWGNC with $P(\cdot)$ replaced by $p(\cdot)$.

The most obvious design criterion applicable to the design of a decoder is the minimum-probability-of-error criterion. When the design criterion is to minimize the probability that the decoder fails to decode to the correct codeword, i.e., to minimize the probability of a *codeword error*, it can be shown that this is equivalent to maximizing the *a posteriori* probability $P(\mathbf{x}|\mathbf{y})$ (or $p(\mathbf{x}|\mathbf{y})$ for the BI-AWGNC). The optimal decision for the BSC is then given by

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{x}|\mathbf{y}) = \arg\max_{\mathbf{v}} \frac{P(\mathbf{y}|\mathbf{x})P(\mathbf{x})}{P(\mathbf{y})}, \tag{1.3}$$

where $\arg\max_{\mathbf{v}} f(\mathbf{v})$ equals the argument $\mathbf{v}$ that maximizes the function $f(\mathbf{v})$. Frequently, the channel-input words are equally likely and, hence, $P(\mathbf{x})$ is independent of $\mathbf{x}$ (hence, $\mathbf{v}$). Because $P(\mathbf{y})$ is also independent of $\mathbf{v}$, the maximum *a posteriori* (MAP) rule (1.3) can be replaced by the *maximum-likelihood* (ML) rule

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{y}|\mathbf{x}).$$

Using (1.2) and the monotonicity of the log function, we have for the BSC

$$\begin{aligned}
\hat{\mathbf{v}} &= \arg\max_{\mathbf{v}} \log \prod_i P(y_i|x_i) \\
&= \arg\max_{\mathbf{v}} \sum_i \log P(y_i|x_i) \\
&= \arg\max_{\mathbf{v}} \left[d_{\mathrm{H}}(\mathbf{y},\mathbf{x})\log(\varepsilon) + (n - d_{\mathrm{H}}(\mathbf{y},\mathbf{x}))\log(1-\varepsilon)\right] \\
&= \arg\max_{\mathbf{v}} \left[d_{\mathrm{H}}(\mathbf{y},\mathbf{x})\log\left(\frac{\varepsilon}{1-\varepsilon}\right) + n\log(1-\varepsilon)\right] \\
&= \arg\min_{\mathbf{v}} d_{\mathrm{H}}(\mathbf{y},\mathbf{x}),
\end{aligned}$$

where n is the codeword length and the last line follows since $\log[\varepsilon/(1-\varepsilon)] < 0$ and $n\log(1-\varepsilon)$ is not a function of $\mathbf{v}$.

For the BI-AWGNC, the ML decision is

$$\hat{\mathbf{v}} = \arg\max_{\mathbf{v}} P(\mathbf{y}|\mathbf{x}),$$

keeping in mind the mapping $\mathbf{x} = (-1)^{\mathbf{v}}$. Following a similar set of steps (and dropping irrelevant terms), we have

$$\begin{aligned}
\hat{\mathbf{v}} &= \arg\max_{\mathbf{v}} \sum_i \log\left(\frac{1}{\sqrt{2\pi}\sigma}\exp\left[-(y_i - x_i)^2/(2\sigma^2)\right]\right) \\
&= \arg\min_{\mathbf{v}} \sum_i (y_i - x_i)^2 \\
&= \arg\min_{\mathbf{v}} d_{\mathrm{E}}(\mathbf{y},\mathbf{x}),
\end{aligned}$$

where

$$d_{\mathrm{E}}(\mathbf{y},\mathbf{x}) = \sqrt{\sum_i (y_i - x_i)^2}$$

is the *Euclidean distance* between $\mathbf{y}$ and $\mathbf{x}$, and on the last line we replaced $d_{\mathrm{E}}^2(\cdot)$ by $d_{\mathrm{E}}(\cdot)$ due to the monotonicity of the square-root function for non-negative arguments. Note that, once a decision is made on the codeword, the decoded data word $\hat{\mathbf{u}}$ may easily be recovered from $\hat{\mathbf{v}}$, particularly when the codeword is in the form $\mathbf{v} = (\mathbf{u}\ \mathbf{p})$.

In summary, for the BSC, the ML decoder chooses the codeword $\mathbf{v}$ that is closest to the channel output $\mathbf{y}$ in a Hamming-distance sense; for the BI-AWGNC, the ML decoder chooses the code sequence $\mathbf{x} = (-1)^{\mathbf{v}}$ that is closest to the channel output $\mathbf{y}$ in a Euclidean-distance sense. The implication for code design on the BSC is that the code should be designed to maximize the minimum Hamming distance between two codewords (and to minimize the number of codeword pairs at that distance). Similarly, the implication for code design on the BI-AWGNC is that the code should be designed to maximize the minimum Euclidean distance between any two code sequences on the channel (and to minimize the number of code-sequence pairs at that distance).

Finding the codeword $\mathbf{v}$ that minimizes the Hamming (or Euclidean) distance in a brute-force, exhaustive fashion is very complex except for very simple codes such as the (7,4) Hamming code. Thus, ML decoding algorithms have been developed that exploit code structure, vastly reducing complexity. Such algorithms are presented in subsequent chapters. Suboptimal but less complex algorithms, which perform slightly worse than the ML decoder, will also be presented in subsequent chapters. These include so-called bounded-distance decoders, list decoders, and iterative decoders involving component sub-decoders. Often these component decoders are based on the *bit-wise MAP criterion* which minimizes the probability of *bit error* rather than the probability of *codeword* error. This bit-wise MAP criterion is

$$\hat{v}_i = \arg\max_{v_i} P(x_i|\mathbf{y}) = \arg\max_{v_i} \frac{P(\mathbf{y}|x_i)P(x_i)}{P(\mathbf{y})},$$

where the *a priori* probability $P(x_i)$ is constant (and ignored together with $P(\mathbf{y})$) if the decoder is operating in isolation, but is supplied by a companion decoder if the decoder is part of an iterative decoding scheme. This topic will also be discussed in subsequent chapters.

The most commonly used performance measure is the *bit-error probability*, P_b, defined as the probability that the decoder output decision $\hat{u}_i$ does not equal the encoder input bit u_i,

$$P_b \triangleq \Pr\{\hat{u}_i \neq u_i\}.$$

Strictly speaking, we should average over all i to obtain P_b. However, $\Pr\{\hat{u}_i = u_i\}$ is frequently independent of i, although, if it is not, the averaging is understood.

P_b is often called the *bit-error rate*, denoted BER. Another commonly used performance measure is the codeword-error probability, P_{cw}, defined as the probability that the decoder output decision $\hat{\mathbf{v}}$ does not equal the encoder output codeword $\mathbf{v}$,

$$P_{cw} \triangleq \Pr\{\hat{\mathbf{v}} \neq \mathbf{v}\}.$$

In the coding literature, various alternative terms are used for P_{cw}, including *word-error rate* (WER) and *frame-error rate* (FER). A closely related error probability is the probability $P_{uw} \triangleq \Pr\{\hat{\mathbf{u}} \neq \mathbf{u}\}$, which can be useful for some applications, but we shall not emphasize this probability, particularly since $P_{uw} \approx P_{cw}$ for many coding systems. Lastly, for nonbinary codes, the symbol-error probability P_s is pertinent. It is defined as

$$P_s \triangleq \Pr\{\hat{u}_i \neq u_i\},$$

where in this case the encoder input symbols u_i and the decoder output symbols $\hat{u}_i$ are nonbinary. P_s is also called the *symbol-error rate* (SER). We shall use the notation introduced in this paragraph throughout this book.

1.5 Channel-Capacity Formulas for Common Channel Models

From the time of Shannon's seminal work in 1948 until the early 1990s, it was thought that the only codes capable of operating near capacity are long, impractical codes, that is, codes that are impossible to encode and decode in practice. However, the invention of turbo codes and low-density parity-check (LDPC) codes in the 1990s demonstrated that near-capacity performance was possible in practice. (As explained in Chapter 5, LDPC codes were first invented circa 1960 by R. Gallager and later independently re-invented by MacKay and others circa 1995. Their capacity-approaching properties with practical encoders/decoders could not be demonstrated with 1960s technology, so they were mostly ignored for about 35 years.) Because of the advent of these capacity-approaching codes, knowledge of information theory and channel capacity has become increasingly important for both the researcher and the practicing engineer. In this section we catalog capacity formulas for a variety of commonly studied channel models. We point out that these formulas correspond to infinite-length codes. However, we will see numerous examples in this book where finite-length codes operate very close to capacity, although this is possible only with long codes ($n > 5000$, say).

No derivations are given for the various capacity formulas. For such information, see [3–9]. However, it is useful to highlight the general formula for the mutual information between the channel output represented by Y and the channel input represented by X. When the input and output take values from a discrete set, then the *mutual information* may be written as

$$I(X;Y) = H(Y) - H(Y|X), \quad (1.4)$$