

ERROR-CORRECTING CODES: HAMMING DISTANCE*

Don Johnson

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0[†]

Abstract

So-called linear codes create error-correction bits by combining the data bits linearly. Topics discussed include generator matrices and the Hamming distance.

So-called **linear codes** create error-correction bits by combining the data bits linearly. The phrase "linear combination" means here single-bit binary arithmetic.

$0 \oplus 0 = 0$	$1 \oplus 1 = 0$	$0 \oplus 1 = 1$	$1 \oplus 0 = 1$
$0 \cdot 0 = 0$	$1 \cdot 1 = 1$	$0 \cdot 1 = 0$	$1 \cdot 0 = 0$

Table 1

For example, let's consider the specific (3, 1) error correction code described by the following coding table and, more concisely, by the succeeding matrix expression.

$$c(1) = b(1)$$

$$c(2) = b(1)$$

$$c(3) = b(1)$$

or

$$c = Gb$$

where

$$G = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$c = \begin{pmatrix} c(1) \\ c(2) \\ c(3) \end{pmatrix}$$

$$b = \begin{pmatrix} b(1) \end{pmatrix}$$

*Version 2.29: Jun 11, 2009 9:28 am -0500

[†]<http://creativecommons.org/licenses/by/1.0>

The length- K (in this simple example $K = 1$) block of data bits is represented by the vector b , and the length- N output block of the channel coder, known as a **codeword**, by c . The **generator matrix** G defines all block-oriented linear channel coders.

As we consider other block codes, the simple idea of the decoder taking a majority vote of the received bits won't generalize easily. We need a broader view that takes into account the **distance** between codewords. A length- N codeword means that the receiver must decide among the 2^N possible datawords to select which of the 2^K codewords was actually transmitted. As shown in Figure 1, we can think of the datawords geometrically. We define the **Hamming distance** between binary datawords c_1 and c_2 , denoted by $d(c_1, c_2)$ to be the minimum number of bits that must be "flipped" to go from one word to the other. For example, the distance between codewords is 3 bits. In our table of binary arithmetic, we see that adding a 1 corresponds to flipping a bit. Furthermore, subtraction and addition are equivalent. We can express the Hamming distance as

$$d(c_1, c_2) = \text{sum}(c_1 \oplus c_2) \quad (1)$$

Exercise 1

(Solution on p. 5.)

Show that adding the error vector $\text{col}[1,0,\dots,0]$ to a codeword flips the codeword's leading bit and leaves the rest unaffected.

The probability of one bit being flipped anywhere in a codeword is $Np_e(1-p_e)^{N-1}$. The number of errors the channel introduces equals the number of ones in e ; the probability of any particular error vector decreases with the number of errors.

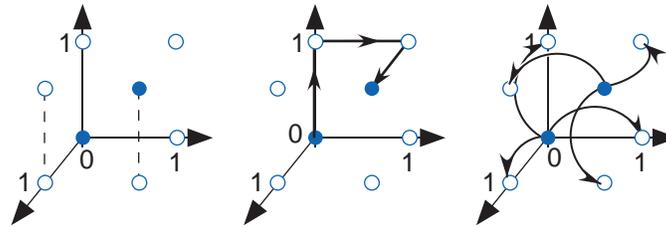


Figure 1: In a (3,1) repetition code, only 2 of the possible 8 three-bit data blocks are codewords. We can represent these bit patterns geometrically with the axes being bit positions in the data block. In the left plot, the filled circles represent the codewords $[0\ 0\ 0]$ and $[1\ 1\ 1]$, the only possible codewords. The unfilled ones correspond to the transmission. The center plot shows that the distance between codewords is 3. Because distance corresponds to flipping a bit, calculating the Hamming distance geometrically means following the axes rather than going "as the crow flies". The right plot shows the datawords that result when one error occurs as the codeword goes through the channel. The three datawords are unit distance from the original codeword. Note that the received dataword groups do not overlap, which means the code can correct all single-bit errors.

To perform decoding when errors occur, we want to find the codeword (one of the filled circles in Figure 1) that has the highest probability of occurring: the one closest to the one received. Note that if a dataword lies a distance of 1 from two codewords, it is **impossible** to determine which codeword was actually sent. This criterion means that if any two codewords are two bits apart, then the code **cannot** correct the channel-induced error. **Thus, to have a code that can correct all single-bit errors, codewords must have a minimum separation of three.** Our repetition code has this property.

Introducing code bits increases the probability that any bit arrives in error (because bit interval durations decrease). However, using a well-designed error-correcting code corrects bit reception errors. Do we win or

lose by using an error-correcting code? The answer is that we can win **if** the code is well-designed. The (3,1) repetition code demonstrates that we can lose (here¹). To develop good channel coding, we need to develop first a general framework for channel codes and discover what it takes for a code to be maximally efficient: Correct as many errors as possible using the fewest error correction bits as possible (making the efficiency $\frac{K}{N}$ as large as possible.) We also need a systematic way of finding the codeword closest to any received dataword. A much better code than our (3,1) repetition code is the following (7,4) code.

$$\begin{aligned}c(1) &= b(1) \\c(2) &= b(2) \\c(3) &= b(3) \\c(4) &= b(4) \\c(5) &= b(1) \oplus b(2) \oplus b(3) \\c(6) &= b(2) \oplus b(3) \oplus b(4) \\c(7) &= b(1) \oplus b(2) \oplus b(4)\end{aligned}$$

where the generator matrix is

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{pmatrix}$$

In this (7,4) code, $2^4 = 16$ of the $2^7 = 128$ possible blocks at the channel decoder correspond to error-free transmission and reception.

Error correction amounts to searching for the codeword c closest to the received block \hat{c} in terms of the Hamming distance between the two. The error correction capability of a channel code is limited by how close together any two error-free blocks are. Bad codes would produce blocks close together, which would result in ambiguity when assigning a block of data bits to a received block. The quantity to examine, therefore, in designing code error correction codes is the minimum distance between codewords.

$$\forall c_i \neq c_j : (d_{\min} = \min(d(c_i, c_j))) \quad (2)$$

To have a channel code that can correct all single-bit errors, $d_{\min} \geq 3$.

Exercise 2

(Solution on p. 5.)

Suppose we want a channel code to have an error-correction capability of n bits. What must the minimum Hamming distance between codewords d_{\min} be?

How do we calculate the minimum distance between codewords? Because we have 2^K codewords, the number of possible unique pairs equals $2^{K-1} (2^K - 1)$, which can be a large number. Recall that our channel coding procedure is linear, with $c = Gb$. Therefore $c_i \oplus c_j = G(b_i \oplus b_j)$. Because $b_i \oplus b_j$ always yields another block of **data** bits, we find that the difference between any two codewords is another codeword! Thus, to find d_{\min} we need only compute the number of ones that comprise all non-zero codewords. Finding these codewords is easy once we examine the coder's generator matrix. Note that the columns of G are codewords

¹"Block Channel Coding", Exercise 1 <<http://cnx.org/content/m0094/latest/#exer2>>

(why is this?), and that all codewords can be found by all possible pairwise sums of the columns. To find d_{\min} , we need only count the number of bits in each column and sums of columns. For our example (7, 4), G 's first column has three ones, the next one four, and the last two three. Considering sums of column pairs next, note that because the upper portion of G is an identity matrix, the corresponding upper portion of all column sums must have exactly two bits. Because the bottom portion of each column differs from the other columns in at least one place, the bottom portion of a sum of columns must have at least one bit. Triple sums will have at least three bits because the upper portion of G is an identity matrix. Thus, no sum of columns has fewer than three bits, which means that $d_{\min} = 3$, and we have a channel coder that can correct all occurrences of one error within a received 7-bit block.

Solutions to Exercises in this Module

Solution to Exercise (p. 2)

In binary arithmetic (see Table 1), adding 0 to a binary value results in that binary value while adding 1 results in the opposite binary value.

Solution to Exercise (p. 3)

$$d_{\min} = 2n + 1$$