

## MAS 160/510 Additional Notes: Error-Correcting Codes

The fundamental idea behind coders for noisy discrete channels is redundancy. We're going to make individual messages somehow "more unique" so that noise won't corrupt enough of the symbols in a message to destroy its uniqueness, and we're going to try to do it in a more organized way than just repeating the same message over and over again.

Another key ingredient in coders for noisy channels is often called noise averaging, and we can see what this means by a simple example. Suppose on average 1% of the received bits have been corrupted by the channel. Looking at a bit, can we tell if it's a "good" bit or a "bad" bit? No. If we look at a block of ten bits, can we identify the bad ones? No, but we can make some statistical observations about how likely it is that some number of bits is incorrect:

$$p(10 \text{ good}) = .99^{10} \approx 9 \times 10^{-1}$$

$$p(1 \text{ bad}) = .99^9 \times .01 \times (\text{number of combinations} : \frac{10!}{1!(10-1)!}) \approx 9 \times 10^{-2}$$

$$p(2 \text{ bad}) = .99^8 \times .01^2 \times \frac{10!}{2!(10-2)!} \approx 4 \times 10^{-3}$$

$$p(3 \text{ bad}) = .99^7 \times .01^3 \times \frac{10!}{3!(10-3)!} \approx 1 \times 10^{-4}$$

Summing these and subtracting from 1, we find that the probability that more than three bits in a block have been corrupted is about  $2 \times 10^{-6}$ . So on average only two blocks in a million will have more than three bad bits. If we had some way of correcting no more than three bad bits out of a block of ten, 999,998 times out of a million we'd be alright, which might be good enough for many applications. Were we to make the blocks still bigger than 10 bits, we'd find that the average number of bad bits per block would converge to the 1% figure.

The redundancy requirement clearly suggests that if we have  $n$ -bit blocks we can't allow all  $2^n$  possible sequences to be valid codes, or else we wouldn't know when a received block is bad. Going further, it seems as if correcting all possible errors of  $t$  bits/block will require us to make every legitimate message block differ from every other legitimate message block in at least  $2t + 1$  bit positions (the number of bit positions in which two code blocks differ is called the Hamming distance).

An error-correcting code, then, is an orderly way of mapping some number of symbols into a greater number so as to increase the Hamming distance between any two valid sequences. They can be divided into two basic types:

- Block codes map a  $k$ -symbol input sequence into an  $n$ -symbol output sequence, where each  $n$ -symbol output depends upon only the  $k$  symbols input and no others. Block codes are usually specified as  $(n, k)$  codes, where  $k$  is in the range of 3 to a few hundred, and  $k/n$  is between roughly 1/4 and 7/8. Block codes are also sometimes called group codes.
- Tree codes are more like state machines – each set of output symbols depends on the current set of inputs as well as some number of earlier inputs. These codes are said to have memory, and are also called convolutional codes, since the encoding operation can be considered as convolving a sequence of inputs with an "impulse response".

In this handout, we'll look at block codes, which can be modeled using binary linear algebra. The simplest block code is a *parity-check* code. We can write the  $(4,3)$  example as taking three bits  $(a_0, a_1, a_2)$  and producing a 4-vector  $\vec{a}$ . As this is usually written as a column vector, for typographical convenience we'll show it transposed:

$$\vec{a}^T = (a_0, a_1, a_2, a_0 + a_1 + a_2),$$

where the addition is modulo-2, so  $1 + 1 = 0$  and subtraction is equivalent to addition. While this won't correct errors (not enough distance between valid sequences to identify which bit is bad if the fourth entry isn't the modulo-2 sum of the other three) it will identify blocks with a single bit error. A better code might be the  $(6, 3)$  code given by:

$$\vec{a}^T = (a_0, a_1, a_2, a_0 + a_1, a_1 + a_2, a_0 + a_1 + a_2).$$

This can be written as a system of linear equations:

$$a_3 = a_0 + a_1$$

$$a_4 = a_1 + a_2$$

$$a_5 = a_0 + a_1 + a_2$$

Since in modulo-2,  $-a = a$  we can say

$$a_0 + a_1 + a_3 = 0$$

$$a_1 + a_2 + a_4 = 0$$

$$a_0 + a_1 + a_2 + a_5 = 0$$

and this can also be written as a "parity-check matrix":

$$H = \begin{bmatrix} 110100 \\ 011010 \\ 111001 \end{bmatrix}.$$

Now we can say that a valid code word is one that satisfies the equation

$$H\vec{a} = \vec{0}.$$

To test this idea, we can encode a particular sequence, say  $(100)$ , which comes out as  $\vec{a}^T = (100101)$ , and indeed we find that

$$\begin{bmatrix} 110100 \\ 011010 \\ 111001 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

There is a particular set of  $(2^m - 1, 2^m - m - 1)$  codes called Hamming codes, in which the columns of the parity-check matrix are the nonzero binary numbers up to  $2^m$ . Thus the matrix for the  $(7, 4)$  Hamming code is

$$H = \begin{bmatrix} 0001111 \\ 0110011 \\ 1010101 \end{bmatrix},$$

and  $\vec{a}$  is valid if

$$a_3 + a_4 + a_5 + a_6 = 0$$

$$a_1 + a_2 + a_5 + a_6 = 0$$

$$a_0 + a_2 + a_4 + a_6 = 0.$$

Since  $a_0$ ,  $a_1$ , and  $a_3$  appear only once here, it'll be mathematically easier if we let  $a_2$ ,  $a_4$ ,  $a_5$ , and  $a_6$  be our 4-bit input block and compute the other three bits from the above equations. For example:

$$(1010) \Rightarrow (a_0 a_1 a_3 010) = (1011010).$$

Now, how do we correct an error? Consider that we have sent some good sequence  $\vec{g}$ , but an error pattern  $\vec{e}$  has been added by the channel, so what is received is

$$\vec{r} = \vec{g} + \vec{e}.$$

Upon receiving  $\vec{r}$ , we multiply it by  $H$  and get a vector  $\vec{s}$ , called the “syndrome” (Webster: “A group of signs that occur together characterizing a particular abnormality.”). Thus

$$\vec{s} = H\vec{r} = H\vec{g} + H\vec{e}.$$

But recall that by definition

$$H\vec{g} = \vec{0},$$

so

$$\vec{s} = H\vec{e}.$$

In other words, the syndrome depends upon only the error, no matter what the message is. Let’s take our valid code (1011010) above, and add an error in the last position: (1011011). Multiplying it by  $H$  (try it yourself!) we get the answer

$$\vec{s}^T = (111),$$

and therefore

$$\vec{e}^T = (0000001).$$

In general, in Hamming codes, the syndrome resulting from an error in the  $n$ -th position is the  $n$ -th column of the  $H$  array. But the  $n$ -th column is also the number of the bit position with the error, making correction easy. More complicated methods exist for constructing codes that correct multiple-bit errors, and for shuffling data to optimize for particular kinds of errors (predominantly impulsive noise versus drop-outs of several bits in sequence, for instance). For more information see Peterson and Weldon, *Error-Correcting Codes* (Barker TK5101.P485), or Clark and Cain, *Error-Correction Coding for Digital Communications* (Barker TK5102.5.C52).