HAMMING CODES:

The linearity property of the code word is that the sum of two code words is also a code word. Hamming codes are the type of **linear error correcting** codes, which can detect up to two bit errors or they can correct one bit errors without the detection of uncorrected errors.

While using the hamming codes, extra parity bits are used to identify a single bit error. To get from one-bit pattern to the other, few bits are to be changed in the data. Such number of bits can be termed as **Hamming distance**. If the parity has a distance of 2, one-bit flip can be detected. But this can't be corrected. Also, any two bit flips cannot be detected.

However, Hamming code is a better procedure than the previously discussed ones in error detection and correction.

Hamming Weight: The Hamming weight of this code scheme is the largest number of 1‟s ina valid code word. This number is 3 among the 10 code words we have chosen. (the ones in the while space).

Concept of Hamming Distance: In continuous variables, we measure distance by Euclidean concepts such as lengths, angles and vectors. In the binary world, distances are measured between two binary words by something called the Hamming distance. The Hamming distance is the number of disagreements between two binary sequences of the same size. The Hamming distance between sequences 001 and 101 is = 1 The Hamming distance between sequences 0011001 and 1010100 is = 4. Hamming ***distance*** and ***weight*** are very important and useful concepts in coding. The knowledge of Hamming distance is used to determine the capability of a code to detect and correct errors. Although the Hamming ***weight*** of our chosen code set is 3, the minimum Hamming ***distance*** is 1. We can generalize this to say that the maximum number of error bits that can be detected is

t = dmin –1.

Where dmin is Hamming distance of the code words.

For a code with dmin = 3, we can both detect 1 and 2 bit errors. So we want to have a code set with as large a Hamming distance as possible since this directly effects our ability to detect errors. The number of errors that we can correct is given by

$$t(\text{int}) = \frac{d_{\min} - 1}{2}$$

For any positive integer $m > 3$, there exists a linear block code with the following parameters:

code length $n = 2m - 1$

number of message bits $k = 2m - m - 1$

number of parity-check bits $n - k = m$

Such a linear block code for which the error-correcting capability $t = 1$ is called a Hamming code.3 To be specific, consider the example of $m = 3$, yielding the (7, 4) Hamming code with $n = 7$ and $k = 4$. The generator of this code is defined by

$$\mathbf{G} = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{array}\right] = [\underbrace{\mathbf{P}}_{\mathbf{P}} \quad \underbrace{\mathbf{I}_k}_{\mathbf{I}_k}]$$

$$\mathbf{H} = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] = [\underbrace{\mathbf{I}_{n-k}}_{} \quad \underbrace{\mathbf{P}^{\mathrm{T}}}_{}]$$

The operative property embodied in this equation is that the columns of the parity-check matrix **P** consist of all the nonzero *m*-tuples, where $m = 3$. With $k = 4$, there are $2k = 16$ distinct message words, which are listed in Table 10.1. For ma given message word, the corresponding codeword is obtained by using (10.13). Thus, the application of this equation results in the 16 codewords listed in Table 10.1.

In Table 10.1, we have also listed the Hamming weights of the individual codewords in the (7,4) Hamming code. Since the smallest of the Hamming weights for the nonzero codewords is 3, it follows that the minimum distance of the code is 3, which is what it should be by definition. Indeed, all Hamming codes have the property that the minimum distance $d\text{min} = 3$, independent of the value assigned to the number of parity bits *m*. To illustrate the relation between the minimum distance *d*min and the structure of the parity-check matrix **H**, consider the codeword 0110100.

Table 10.1 Codewords of a (7,4) Hamming code

Message word	Codeword	Weight of codeword	Message word	Codeword	Weight of codeword
0000	0000000	0	1000	1101000	3
0001	1010001	3	1001	0111001	4
0010	1110010	4	1010	0011010	3
0011	0100011	3	1011	1001011	3
0100	0110100	3	1100	1011100	4
0101	1100101	4	1101	0001101	3
0110	1000110	3	1110	0101110	4
0111	0010111	4	1111	1111111	7

An important property of binary Hamming codes is that they satisfy the condition of with the equality sign, assuming that $t = 1$. Thus, assuming single-error patterns, we may formulate the error patterns listed in the right-hand column of Table 10.2. The corresponding eight syndromes, listed in the left-hand column, are calculated in accordance with (10.20). The zero syndrome signifies no transmission errors.

Suppose, for example, the code vector [1110010] is sent and the received vector is [1100010] with an error in the third bit. Using (10.19), the syndrome is calculated to be

$$\mathbf{s} = [1100010] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

$$= [0\ 0\ 1]$$

the corresponding coset leader (i.e., error pattern with the highest probability of occurrence) is found to be [0010000], indicating correctly that the third bit of the received vector is erroneous. Thus, adding this error pattern to the received vector, in accordance yields the correct code vector actually sent.

Table 10.2 Decoding table for the (7,4) Hamming code defined in Table 10.1

Syndrome	Error pattern
000	0000000
100	1000000
010	0100000
001	0010000
110	0001000
011	0000100
111	0000010
101	0000001

Encoding a message by Hamming Code

The procedure used by the sender to encode the message encompasses the following steps

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Calculating the values of each redundant bit.

Once the redundant bits are embedded within the message, this is sent to the user.

Step 1 − Calculation of the number of redundant bits.

If the message contains *mm*number of data bits, *rr*number of redundant bits are added to it so that *mr* is able to indicate at least (*m* + *r*+ 1) different states. Here,

($m + r$) indicates location of an error in each of ($m + r$) bit positions and one additional state indicates no error. Since, rr bits can indicate $2^r r$ states, $2^r r$ must be at least equal to ($m + r + 1$). Thus the following equation should hold $2^r \geq$ m+r+

Step 2 − Positioning the redundant bits.

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc. They are referred in the rest of this text as r_1 (at position 1), r_2 (at position 2), r_3 (at position 4), r_4 (at position 8) and so on.

Step 3 − Calculating the values of each redundant bit.

The redundant bits are parity bits. A parity bit is an extra bit that makes the number of 1s either even or odd. The two types of parity are −

- **Even Parity** − Here the total number of bits in the message is made even.
- **Odd Parity** − Here the total number of bits in the message is made odd.

Each redundant bit, r_i, is calculated as the parity, generally even parity, based upon its bit position. It covers all bit positions whose binary representation includes a 1 in the i^{th} position except the position of r_i. Thus −

- r_1 is the parity bit for all data bits in positions whose binary representation includes a 1 in the least significant position excluding 1 (3, 5, 7, 9, 11 and so on)
- r_2 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 2 from right except 2 (3, 6, 7, 10, 11 and so on)
- r_3 is the parity bit for all data bits in positions whose binary representation includes a 1 in the position 3 from right except 4 (5-7, 12-15, 20-23 and so on)

Decoding a message in Hamming Code

Once the receiver gets an incoming message, it performs recalculations to detect errors and correct them. The steps for recalculation are −

- **Step 1** − Calculation of the number of redundant bits.
- **Step 2** − Positioning the redundant bits.
- **Step 3** − Parity checking.
- **Step 4** − Error detection and correction

Step 1 − Calculation of the number of redundant bits

Using the same formula as in encoding, the number of redundant bits are ascertained.

$2^r \geq m + r + 1$ where *m* is the number of data bits and *r* is the number of redundant bits.

Step 2 − Positioning the redundant bits

The *r* redundant bits placed at bit positions of powers of 2, i.e. 1, 2, 4, 8, 16 etc.

Step 3 − Parity checking

Parity bits are calculated based upon the data bits and the redundant bits using the same rule as during generation of c_1,c_2 ,c_3 ,c_4 etc. Thus

c_1 = parity(1, 3, 5, 7, 9, 11 and so on)

c_2 = parity(2, 3, 6, 7, 10, 11 and so on)

c_3 = parity(4-7, 12-15, 20-23 and so on)

Step 4 − Error detection and correction

The decimal equivalent of the parity bits binary values is calculated. If it is 0, there is no error. Otherwise, the decimal value gives the bit position which has error. For example, if $c_1c_2c_3c_4$ = 1001, it implies that the data bit at position 9, decimal equivalent of 1001, has error. The bit is flipped to get the correct message.