X Lecture 11, Error Control Coding

The Concept of error Control Coding consists in adding extra (redundant) bits (or symbols) to the data stream so that the presence of errors can be detecte and, possibly corrected.

Example: Assume that each transmission attempt consists of sending an n-bit packet. The n bit in the packet take 2^n possible values. If one bit in the packet is changed from 0 to 1 or vice versa, we get another n-bit pattern and we will decide in favour of that bit pattern. Note that the n-bit packets each have 0, 1, 2, ---, n ones, i.e., Some have 0, 2, 4, --- 1's and some other have 1, 3, 5, --- 1's. An error changes a pattern with odd number of ones into one with an even number of ones and vice versa.

As long as we have both type of patterns, even a simple error as this remains undetecte

But if we had only odd (or only even) number of ones in the packets we send, we could detect one bit flip errors.

Take the case of 4 bit patterns. If we add one bit to them to make them 5-bit blocks with say <u>even</u> parity then any single error makes them one with <u>odd</u> parity and the error is detected.

0	0	0	0	<u>0</u>	← added <u>0</u> so that the number of 1's remain zero (even)
0	0	0	1	<u>1</u>	← added one to make the number of 1's even.
0	0	1	0	1	"
0	0	1	1	0	
0	1	0	0	1	"
0	1	0	1	0	
⋮	⋮	⋮	⋮	⋮	
1	1	1	0	1	
1	1	1	1	0	

There are two basic approaches to error control coding:

1) To detect the presence of error and ask for retransmission. This is called Automatic Repeat Request (ARQ). This approach is good in sense of efficiency since you need little overhead to detect (as compared to correct) errors. But the scheme is not suitable for real-time applications.

2) Forward Error Correction (FEC): In this case, enough redundant bits is sent to correct errors. FEC is the suitable scheme for real-time applications, but it involves more overhead. Also, you need to know the average condition of the channel in order not to leave errors undetected (i.e., having less parity than needed)

or have more overhead (parity) than required.

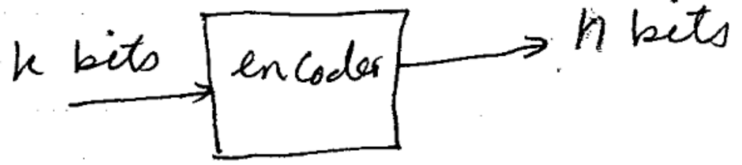
There are two basic categories of Forward Error Correcting (FEC) codes:

- Block Codes
- Convolutional Codes.

Linear Block Codes

A code is linear if any two codewords (A codeword is the patterns that are included as legitimate patterns) when added modulo-2 (XOR'd) result in another codeword.

In a linear block code k-bit patterns are mapped into (encoded to) n-bit pattern where n > k. The n-k extra bits are parity bits



Note that n bits can take 2^n possible values, but we are only using 2^k of them. So, for every codeword there are $2^{n-k}-1$ patterns that are excluded. So if a pattern $\underline{c} = (c_0, c_1, \cdots, c_{n-1})$ is sent and some error changes it to one of those excluded patterns, we can detect and possibly correct the error.

<u>Systematic</u> Codes are the ones where <u>message bits</u> (the bits that the user is interested in) are separated from the <u>parity bits</u> (bits used just for error control

$b_0\ b_1\ b_2 \cdots b_{n-k+1}$	$m_0, m_1, \cdots m_{k-1}$
parity bits	message bits

or,

$$c_i = \begin{cases} b_i & i = 0, 1, \cdots, n-k+1 \\ m_{i+k-n} & i = n-k, n-k+1, \cdots n-1 \end{cases}$$

The parity bits are the linear sums (XOR's) of the message bits.

$$b_i = p_{0i} m_0 + p_{1i} m_1 + \cdots + p_{k-1,i} m_{k-1}$$

where,

$$p_{ij} = \begin{cases} 1 & \text{if } b_i \text{ depends on } m_j \\ 0 & \text{if } b_i \text{ does not depend on } m_j \end{cases}$$

In matrix (vector) form:

$\underline{m} = [m_0, m_1, \cdots, m_{k-1}]$ message vector

$\underline{b} = [b_0, b_1, \cdots, b_{n-k-1}]$ parity vector

and

$\underline{c} = [c_0, c_1, \cdots, c_{n-1}]$ Codeword or Codevector.

we can write

$$\underline{b} = \underline{m}\,\underline{P}$$

where $\underline{P}$ is the parity matrix.

$$\underline{P} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}$$

$$\underline{c} = [\underline{b} \mid \underline{m}] = \underline{m}[\underline{P} \mid I_k]$$

I_k is the k-by-k identity matrix

$$I_k = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & & & \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

Now, we can define the k-by-n

<u>Generator Matrix</u> G

$$G = [P \mid I_k]$$

and write

$$\underline{c} = \underline{m}\,\underline{G}$$

Define the matrix $\underline{H}$ as:

$$\underline{H} = [I_{n-k} \mid P^T]$$

Then

$$H G^T = [I_{n-k} \mid P^T]\begin{bmatrix} P^T \\ \hline I_k \end{bmatrix} = P^T + P^T = \underline{0}$$

Similarly $G H^T = \underline{0}$

So, if we multiply any code $\underline{c} = \underline{m}\,\underline{G}$ by $\underline{H}^T$

$$\underline{c}\,\underline{H}^T = \underline{m}\,\underline{G}\,H^T = \underline{0}$$

The matrix $\underline{H}$ is called the <u>parity-check</u> matrix

__Example__: (7, 4) Hamming Code.

In this case $k=4$ and $n=7$. That is to each 4-bit message 3 bits of parity is added.

The parities are formed as follows:

$$b_0 = m_0 + m_2 + m_3$$

$$b_1 = m_0 + m_1 + m_2$$

$$b_2 = m_1 + m_2 + m_3$$

So $p_{00} = 1$, $p_{1,0} = 0$, $p_{2,0} = 1$, $p_{3,0} = 1$

$p_{0,1} = 1$, $p_{1,1} = 1$, $p_{2,1} = 1$, $p_{3,1} = 0$

$p_{0,2} = 0$, $p_{1,2} = 1$, $p_{2,2} = 1$, $p_{3,2} = 1$

So, the generator matrix is:

$$G = \left[\underbrace{\begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{matrix}}_{P} \;\vdots\; \underbrace{\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}}_{I_4} \right]$$

The parity-check matrix is:

$$H = \left[\begin{array}{ccc:cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right]$$

$$\underbrace{\qquad}_{I_3} \qquad \underbrace{\qquad\qquad}_{P^T}$$

The Codewords are given in the following Table:

TABLE 10.4 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	4
0100	0110100	3	1100	1011100	4
0101	1100101	4	1101	0001101	3
0110	1000110	3	1110	0101110	4
0111	0010111	4	1111	1111111	7

To see how this Code Can Correct a single error assume that we have 0101 to transmit. We encode it as 1100101. Then we modulate and send it over a noisy channel. Assume that the detected pattern is 1ɸ00100.

We form the parities ~~for~~ this pattern, i.e.,

$$b_0' = 0, \quad b_1' = 1, \quad b_2' = 1.$$

Comparing these parities with the received parities, we see that $b_0' \neq b_0$
$b_1' = b_1$, $b_2' \neq b_2$.

Since $\quad b_0 = m_0 + m_2 + m_3$

- $b_0' \neq b_0$ means that there is an error either in $\underset{\checkmark}{m_0}, \underset{\checkmark}{m_2}, \underset{=}{m_3}$ or b_0 itself.
- $b_1' = b_1$ indicates that there is no error in $\underset{\checkmark}{m_0}, \underset{\checkmark}{m_1}, \underset{\checkmark}{m_2}$, or $\underset{\checkmark}{b_1}$
- $b_2' \neq b_2$ indicates an error in $\underset{\checkmark}{m_1}, \underset{\checkmark}{m_2}, \underset{=}{m_3}$ or b_2.

Since m_3 is common in 1'st and 3rd. Syndrome, we decided that m_3 is in error. We flip it $(0 \Rightarrow 1)$ to correct the error and we get our origing 4 bits as $\underline{\underline{0101}}$.

<u>Syndromes</u> and <u>Syndrome</u> decoding:

When a code word $\underline{c}$ is transmitted, we detect a vector $\underline{r} = [r_0, r_1, \cdots, r_{n-1}]$.

Hopefully $r_0 = c_0, r_1 = c_1, \cdots r_{n-1} = c_{n-1}$.

If it is not true and say we have $r_i \neq c_i$ and $r_j \neq c_j$ then we say we have errors in locations i and j.

Since in a binary case, an error means a "0" becoming a "1" or a "1" changing into a "0", we can define the error as $\underline{c}$ being added bit-by-bit with a vector $\underline{e}$ with bits in i-th and j-th place equal to 1 and the other $n-2$ bits being zero. In general

$$\underline{r} = \underline{c} + \underline{e}$$

where $\underline{e}$ is the error added through transmission (and detection).

Remember that any codeword $\underline{c}$ when multiplied by H^T results in $\underline{0}$.

So,

$$\underline{s} = \underline{r} H^T = (\underline{c} + \underline{e}) H^T = \underline{c} H^T + \underline{e} H^T = \underline{e} H^T$$

$S = \underline{r} H^T$ is called the Syndrome (Syndrome vector). It has n-k bits. So, it can take 2^{n-k} values.

Note that $\underline{S}$ only depends on $\underline{e}$ (not what was transmitted, i.e., $\underline{c}$). So, $\underline{S}$ can be used to know which error pattern has occured. But, there are 2^n possible error patterns and we only can count 2^{n-k} of them. So, we list the error patterns that are more likely, i.e., those with least number of errors.

For the Hamming (7,4) Code, we have $n-k=7-4=3$. So, we have 8 Syndrome values 000, 001, 010, ..., 111. Therefore we can count 8 patterns. These include no error pattern 0000000, and 7 patterns with one error, i.e., 1000000, ..., 0000001. For each of these patterns we find the Syndrome as $\underline{S} = \underline{e} H^T$

The list is given as:

TABLE 10.5 Decoding table for the (7, 4) Hamming code defined in Table 10.4

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

Decoding can be performed by finding Syndrome and then look-up for the error pattern and add it to the received bit stream.

Going back to the previous example, assume that we have received

$\underline{r} = 1100100$

We find:

$$S = \underline{r}\, H^T = [1100100] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [1\ 0\ 1]$$

From the table, we see that [101] corresponds to the error pattern $\underline{e} = 0000001$.

Adding $\underline{e}$ to $\underline{r}$ we get:

$$\underline{r} + \underline{e} = [1\,1\,0\,0\,1\,0\,0] + [0\,0\,0\,0\,0\,0\,1]$$
$$= [1\,1\,0\,0\,1\,0\,1]$$

which is a valid Codeword.

Now assume that two bits are in error.

That is, 1100101 is transmitted and we get 1000111. That is, an error is made in the second bit (c_1) and another in the sixth bit (c_6).

The Syndrome will be:

$$\underline{S} = [1\,0\,0\,0\,1\,1\,1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} = [1\,0\,1]$$

From Table 10.5:

[101] Corresponds to error pattern [0000001]. So, we get

$$[1\,0\,0\,0\,1\,1\,1] + [0\,0\,0\,0\,0\,0\,1] = [1\,0\,0\,0\,1\,1\,0]$$

So, one more error is added.

The error-correcting capability of a code depends on its minimum distance. The minimum distance of a code is the number of bits in which the closest codewords differ.
For linear codes, the minimum distance between any codeword and all other codewords is the same. So, we can find the minimum distance d_{min} as the distance of the codeword closest to 00...0 to the codeword 00...0 itself. This is, in fact the number of ~~one~~'s in the codeword with the minimum number of one's. So, for the linear codes, the minimum distance, d_{min} is equal to the minimum weight. For the Hamming code (the (7, 4) one) the minimum distance is three (3): from Table 10.4.
The maximum number of errors that a linear block code can correct is $\left[\frac{d_{min}-1}{2}\right]$. That is $t \leq \left[\frac{d_{min}-1}{2}\right]$. For the Hamming code $t \leq \left[\frac{3-1}{2}\right] = 1$.