Error-Correcting Code

Introduction

Error-Correcting Code is used when transmitting bits over a communication channel to detect when bits become corrupted. The goal of error-correcting code is to detect the corruption and correct it automatically.

A single bit error is where just one bit was corrupted, a burst error is where more than one bits have been altered. Most of this paper will discuss solutions to detecting and correcting a single bit error.

Matrix Algebra with Bits

The matrix algebra used in error-correcting code uses bits rather than numbers. Bits can only be 0 or 1, where numbers have infinite options. Bits have slightly different operations from numbers. Bit addition drops carry values. Possible additions of bits are as follow: 0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, 1 + 1 = 0. Possibilities for bit multiplication is as follows: 0 * 0 = 0, 0 * 1 = 0, 1 * 0 = 0, 1 * 1 = 1. Due to these properties of bits, a bit is the negative of itself. 0 – 1 = 0 + 1 = 1, 1 – 1 = 1 + 1 = 0…

Example of Bit Matrix Multiplication.

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 + 1 + 0 \\ 0 + 1 + 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Repetition Codes

A simple way to detect errors is to send multiple of that item. This raises the probability of the correct bit passing through a communication channel uncorrupted.

For example, an original message [1 1 0] could be sent as [1 1 1 1 0 0]. If the bits aren't matching, then we know that segment had an error. Still, with this repetition, we cannot determine what the original value was. We could repeat each bit three times instead, sending [1 1 1 1 1 1 0 0 0]. This way, we could assume the majority result for each segment was likely the original. However, these solutions require twice to three times as much data, taking up bandwidth in our channel.

Parity Codes

A simple way to decrease the number of bits send, but still detect errors, is using parity bits. Parity bits can determine if the total 1s in the sent code is even or odd. Simple parity checks can detect an odd number of errors. In order to determine where an error occurred, multiple parity bits are typically required.

An example use of parity code expands on the repetition code. Rather than repeating each bit three times, each bit is repeated twice and combined with another bit to determine if the bits were the same value. This can detect small errors with lower overhead.

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x \\ x \\ y \\ y \\ x + y \end{bmatrix}$$

Parity codes can become more complicated to compare more bits to be transmitted. When transmitting a message, k bits (original message) plus r (= n – k) parity bits are sent with a message of size n. An example of this can be found in the Hamming Code section. Parity checks can be used to detect single errors or more, depending on the implementation.

Single Parity Check, or Vertical Redundancy Check (VRC), is a method to detect a single bit error. This is accomplished by adding a single parity bit after every data unit. This parity bit determines if the total number of 1s is even or odd, this can be calculated by adding the bits since we are using bit addition. A parity bit of 0 contains an even number of 1s and a 1 is an odd number. This can also be calculated by the modulus 2 of the sum of bits. Single Parity Check, for example, where the data is 11010011, has the parity bit 1, and the result 11010011 1 is transmitted.

This method can only detect single bit errors and only detects when an odd number of bits have changed. For a channel that experiences few errors, this could be an effective method of error detection.

Two Dimensional Parity Check, or Longitudinal Redundancy Check (LRC), is a stronger error detection method. This involves dividing the block of bits into rows. A redundant row of bits is added to this block. This is organized into a table, and a parity bit is calculated for each row and column. This allows checking for burst errors, however it is limited to certain patterns of errors that will not be discussed in this paper.

Suppose there are m rows and n columns. For Two Dimensional Parity checks, we will compute m + n + 1 parity bits and send mn + m + n + 1 total bits. A table is built with each row being a section of the code and each column being a position of the data. To calculate the result, the parity bit for each row is calculated then the parity bit of each column is calculated. A row parity bit is also calculated for the row of column parity bits.

Example: There are 4 sections of data to be sent with 7 bits each. This produces 4 rows and 7 columns. This means there are (4 + 7 + 1) = 12 parity bits, and the result will be (4*7 + 4 + 7 + 1) = 40 bits in the resulting transmission.

Original Data: 1100111 1011101   0111001   0101001   (Each is a row, in same order)

‘1’s: 5	1	1	0	0	1	1	1	***1***
‘1’s: 5	1	0	1	1	1	0	1	***1***
‘1’s: 4	0	1	1	1	0	0	1	***0***
‘1’s: 3	0	1	0	1	0	0	1	***1***
‘1’s: 3	*0*	*1*	*0*	*1*	*0*	*1*	*0*	***1***

Key: ***Row Parity,*** *Column Parity*

This will Send: 1100111 1   1011101 1   0111001 0   0101001 1   0101010 1

The column parity is calculated by the even/odd bit status of each individual column, and the row parity is determined by the even/odd bit status of each row. The sent information contains the original information followed by the row parity. This also sends the column parity row and its corresponding row parity.

Two Dimensional Parity checks provide a stronger error detection and can detect multiple errors, however it requires more checks to determine errors.

Hamming Code

The goal of Hamming Code is to detect and correct errors while sending less data than repetition codes. This process is used in computing and telecommunications as an efficient 1-error and 2-error detecting code. Hamming Codes are linear codes, and can be described as [n,k], for length n and dimension k. A Generator matrix, G, is a k x n matrix that has a rowspace equal to the code being sent. A check matrix is used to verify the sent data, it is formed using the calculates used to create the sent data.

Hamming Code requires the use of a generator matrix to build the data to be send. A 3x1 generator matrix G holds the values of x, where x has the values that need to be sent. The matrix Gx is what is transmitted. The value of Gx is calculated and sent, then the vector c is received.

$$\mathrm{Gx} = \begin{bmatrix}1\\1\\1\end{bmatrix} \text{ } if\ x = [1] \qquad Gx = \begin{bmatrix}0\\0\\0\end{bmatrix} \text{ } if\ x = [0] \quad Recipient\ gets{:}\ c = \begin{bmatrix}c1\\c2\\c3\end{bmatrix}$$

Ideally, c = Gx. But if an error did occur, the recipient can find it using a parity check. If the message has no errors, $c_1 + c_2$ and $c_2 + c_3$ are even, or in other words, the same bit. Using bit addition, the result of each equation should be zero. And a parity check matrix, P, can be formed. This is how the recipient will determine if the transmission contains errors.

$$\mathrm{c1 + c2 = 0, c2 + c3 = 0 \ : \ P} = \begin{bmatrix}1 & 1 & 0\\0 & 1 & 1\end{bmatrix}$$

The recipient then computes Pc, which has four possible outcomes.

$$\mathrm{Pc} = \begin{bmatrix}0\\0\end{bmatrix} \text{ } The\ result\ of\ the\ additions\ were\ both\ even. This\ means\ that\ no\ errors\ were\ detected.$$

$$\mathrm{Pc} = \begin{bmatrix}1\\0\end{bmatrix} \text{ } This\ means\ c1 + c2\ was\ odd\ and\ c2 + c3\ was\ even. So\ c2 = c3, and\ the\ error\ was\ likely\ in\ c1.$$

$$\mathrm{Pc} = \begin{bmatrix}0\\1\end{bmatrix} \text{ } This\ means\ c1 + c2\ was\ even\ and\ c2 + c3\ was\ odd. So\ c1 = c2, and\ the\ error\ was\ likely\ in\ c3.$$

$$\mathrm{Pc} = \begin{bmatrix}1\\1\end{bmatrix} \text{ } This\ means\ c1 + c2\ and\ c2 + c3\ were\ odd. The\ error\ was\ likely\ in\ c2.$$

Using these results, the recipient can determine where, and if, an error occurred and get the correct value. This isn’t a guarantee, since it is still possible that two or three bits get flipped, however it is more likely to be correct.

This idea can be expanded to send larger amounts of information and check that information efficiently using matrices. This will be demonstrated through an example.

An example of Hamming Code is (7,4) Hamming code, which uses groups of four bits to be sent and encodes it in seven bits. This produces a ratio of 7/4=1.75 times as many bits as the original, which is preferable to 3 times as much using parity bits.

Suppose we want to send four bits: x_1, x_2, x_3 and x_4. The first four bits to be sent are the bits themselves. The other three are the following combinations: $x_1 + x_3 + x_4$, $x_1 + x_2 + x_4$, and $x_2 + x_3 + x_4$. These extra three bits are parity bits, which must be sent in the order specified. Another way to generate these bits is the following code generator matrix G.

$$\mathrm{G} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

The matrix Gx (7x1) is sent in place of the original message x (4x1).

Suppose we want to send [0 1 0 1]. We should get the matrix Gx = [0 1 0 1 1 0 0].

The coefficient matrix of this linear system, and parity check matrix, is as follows.

$$\mathrm{P} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

The recipient can compute Pc, c as the 7x1 result vector, to detect errors. There are eight possible results for Pc.

$$\mathrm{Pc} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \textit{Means no error was detecting in the parity bits.}$$

$$\mathrm{Pc} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \textit{Means two errors were detected, in the corresponding parity bits.}$$

$$\textit{This result detects an error in the first bit.}$$

$$\mathrm{Pc} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \textit{Means an error occured in the second parity bit.}$$

$$\textit{However, there was no error detected in the message bits.}$$

We can use the matrix P to easily see where an error occurred. The column of P that matches the result is the bit that contains an error. If one of the parity bits contains an error, the original bits are correct. However, if the result detects on of the four bits, the error affected the original message. In the case of a zero vector result, no errors were detected.

This process can be expanded to send larger messages. Suppose there are 4k bits to be send (k is a positive integer).

$$\mathrm{X} = \begin{bmatrix} x1 & x5 & \ldots & x(4k-3) \\ x2 & x6 & \ldots & x(4k-2) \\ x3 & x7 & \ldots & x(4k-1) \\ x4 & x8 & \ldots & x(4k) \end{bmatrix}$$

This message can encoded as GX, (7x4 G code generator matrix and X 7xk matrix), the resulting 7xk matrix C is sent. The recipient can then compute PC (3xk matrix). Each column detects errors in that section of bits in the same method as above.

Hamming code can be further improved using different ratios. For example, the ratio (15, 11), which requires 15 bits for each 11 bits to be sent. This creates a ratio of 1.36 bits sent for each bit. This sends the original 11 bits and 4 parity bits. However, I will not go into detail on this Hamming Code. Keep in mind that this process can only detect if one error has occurred, if multiple errors occurred the detection may fail.

Conclusion

Error-Correcting Code can detect and correct errors after transmission of data. Using linear algebra in Hamming Code, we are able to detect errors quickly and efficiently.

Resources

http://www.math.union.edu/~jaureguj/ECC.pdf

http://orion.math.iastate.edu/linglong/Math690F04/HammingCodes.pdf

http://www.math.toronto.edu/afenyes/writing/error-correction%20(october%202015).pdf

http://circuit.ucsd.edu/~yhk/ece154c-spr17/pdfs/ErrorCorrectionI.pdf

http://www.mee.tcd.ie/~ledoyle/TEACHING/12%20-%20error2.pdf

https://www.slideshare.net/ImeshaPerera/parity-checkerror-detecting-codes