Error Correction Coding: Weeks 5-6

Linear Codes

Eric DeFelice
Department of Electrical Engineering
Rochester Institute of Technology
ebd9423@rit.edu

Abstract - **In coding theory, a linear code is an error-correcting code in which any linear combination of codewords is also a codeword. The algebraic structure of these codes provides a framework where efficient encoding and decoding algorithms can be constructed. There are two types of linear codes, block codes and convolutional codes. These two different types will be researched later on during my studies, as this paper will provide an overview of linear codes in general. Many of the current error-correction codes that are in use in systems are subclasses of linear codes defined by imposing additional constraints.**

In this paper, an introduction of linear codes will be presented, and generator matrices will be looked at. Hamming codes will be introduced and analyzed, and perfect codes will be defined. Finally, encoding and decoding algorithms for linear codes can be introduced.

I. INTRODUCTION

After looking into the additive white Gaussian noise channel and finite field theory, we now can start looking at linear codes. The concepts that were reviewed over the past two weeks will act as a base for the advanced coding schemes that will be researched later. To begin looking at linear codes, we can look at what makes up a code and some of the properties of a linear block code.

Suppose that we have a set of messages that we want to transmit over a channel. Let us say that this is a set of k-tuples having components in a field, F, with q elements. With this definition, there would be q^k messages that we can send to the receiver. This set is considered to be a vector space. To expand on this, let us look at an example, considering the English alphabet. There are 26 letters and a space, so we have 27 elements that we need to code. If we use a binary code, we would need at least 27 codes to have unique identifiers for these elements. Using a binary code, we would have 5-tuples, giving us a total of 2^5, or 32, possible codes to use for this set of messages.

Now that we can encode our message into a bit-stream, we will need to add some redundancy to be able to detect and correct errors. This is where the linear code is used. The linear code will take the k bits and encode them into n bits, where $n>k$. This code is designated as an (n, k) code. In doing this, we will add n-k redundant bits. One way of adding these redundant bits is by using a *generator matrix*, which will be looked at further in the next section. After the codeword has been transmitted, we will need a way to detect and correct the errors that may have been caused by the channel. A *parity-check matrix* can be used for this purpose, and we will also look more into these in a subsequent section. Hamming codes will then be introduced, as they are a subset of linear codes that have some added constraints on their construction.

II. GENERATOR MATRICES

A *generator matrix* G for an (n, k) code C is a $k \times n$ matrix whose rows are a vector space basis for C [3]. The codewords of a linear code C with generator matrix G are all linear combinations of the rows of G. Therefore they are all in the *row space* of G. When using a generator matrix, the codewords that are output from the generation operation are said to be generated by the matrix G. The dimension k, defined above, for the code to be generated is the number of linearly independent rows in the defining matrix, or the *rank* of the matrix.

Given that the generator matrix is a set of basis functions, it can be written as follows:

$$G = \{g_0, g_1, g_2, \dots, g_{k-1}\}$$

Since G is a set of basis vectors, we can represent the coding operation as matrix multiplication. With this representation, the codewords to be transmitted are created by just multiplying the messages, m, with the generator matrix. The first step is to format G in matrix form so that it can be used in the multiplication. The generator matrix is a $k \times n$ matrix so it can be written as follows:

$$G = \begin{bmatrix} g_0 \\ g_1 \\ \dots \\ g_{k-1} \end{bmatrix}$$

Now that we have defined the structure of the generator matrix, we can look at how it can be used to generate codewords. Suppose we have vectors $v_1 = (10000), v_2 = (11010)$, and $v_3 = (11101)$. These generator vectors for a basis for a (5,3) code C, over an F_2 field. Since k is 3 and we have a binary field, we can have 2^3, or 8, possible codewords. For this example, let us first write out the generator in matrix form:

$$G = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{bmatrix}$$

To get the codewords, we must perform matrix multiplication between the possible messages and the generator matrix. For an example, let us generate one of the possible codwords using message m = (011).

$$c_3 = m_3 G = (011)\begin{bmatrix}1 & 0 & 0 & 0 & 0\\ 1 & 1 & 0 & 1 & 0\\ 1 & 1 & 1 & 0 & 1\end{bmatrix} = (00111)$$

The simple example above shows how a codeword can be created from a message using a generator matrix. In this case, the codeword is (00111) when using this particular generator G and encoding message 3. In an actual implementation, this multiplication can be replaced with a lookup table using the message as the input and the codeword as the output.

Using this generator matrix, it is not very easy to look at the received codeword and get the original bits without some complicated decoding. Another option is to essentially add parity bits to the original bits. Let us look at a different generator matrix to see if this is possible. Suppose that instead of v_1, v_2, and v_3 we select the basis vectors u_1 = (10000), u_2 = (01010), and u_3 = (00111) for the generator. These vectors are still in the same 3-dimensional subspace. Using these vectors, the generator matrix becomes:

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

With this generator matrix, the transmit bits used before, (011), will now become the codeword, c = (01101), which is the original information bits, but with two parity bits added. Using this scheme, the receiver only needs to take the first three bits of the codeword to get the data, making the decoding algorithm much easier. This generator is also called a *parity-check matrix*, because it just appends parity bits to the information bits.

III. PARITY-CHECK MATRIX

Now that generator matrices have been described, we can move onto a special type of generator matrix, called the *parity-check matrix*. We can start by defining what a parity-check matrix is exactly. Let C be an *(n, k)* code over the field F. If H is a generator matrix for $C^{\perp}$, then H is called a parity-check matrix for C. If $G = [I_k A]$ is the generator matrix for C, then $H = [-A^T I_{n-k}]$ is a parity-check matrix for C, and a generator matrix for $C^{\perp}$. Now that we have the definitions for the generator matrix G and the parity-check matrix H, we can specify a linear code C by giving either of these matrices. Given one of them, the other could be found using linear algebra.

To further understand the parity-check matrix let us look at a brief example. Suppose we have a message, $m = (m_1 m_2 \dots m_k)$, that we are going to embed into a codeword c of length *n*. Also suppose that the information symbols, m_k, occur in the first *k* components of the codeword. Then c has the form:

$$c = (m_1 m_2 \dots m_k x_1 x_2 \dots x_{n-k})$$

In this definition, x_i are referred to as the *check symbols* since they are the redundant symbols that allow the receiver to detect and correct errors. For our example, let C be the (6,3)-code over a binary field generated by the generator matrix:

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

This generator matrix does not satisfy the requirements for the definition proposed earlier for a parity-check matrix. To get the generator in the correct format, we can apply some elementary row operations to obtain a matrix G that generates the same code. After performing the linear algebra operations to $G^{'}$ we get:

$$G = \begin{bmatrix}1 & 0 & 0 & 1 & 1 & 1\\ 0 & 1 & 0 & 0 & 1 & 1\\ 0 & 0 & 1 & 0 & 1 & 0\end{bmatrix} = [I_3 A]$$

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

Now that the generator matrix is in this form we can get the parity-check matrix by using the definition. This gives us:

$$H = [-A^T I_3] = \begin{bmatrix}1 & 0 & 0 & 1 & 0 & 0\\ 1 & 1 & 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 0 & 0 & 1\end{bmatrix}$$

This matrix H is the parity-check matrix for the code C or the generator matrix for the orthogonal complement of C, $C^{\perp}$. A simple check to show that the matrix H does in fact satisfy these requirements is to check that G and H are inverse matrices. This means that GH^T=0, which in fact it does in this example.

IV. HAMMING CODES

Now that we have reviewed generator matrices and parity-check matrices, we can examine a class of single-error correcting codes called *Hamming codes*. They will be defined in terms of a parity-check matrix. A *Hamming code* of order r over a field of rank q is an (n,k)-code where $n = \frac{q^r - 1}{q-1}$ and $k = n - r$, with parity-check matrix H_r. Since columns of the parity-check matrix are pairwise linearly independent and there exist three columns which sum up to the all-zero vector, the minimum Hamming distance for the binary Hamming code is 3 [2]. This code can detect up to 2 errors and correct 1 error. Note that $q^r - 1$ is the number of non-zero *r*-tuples that exist, and $q - 1$ is the number of non-zero scalars. This means that if another column was appended to H_r, it would be a scalar multiple of one that already exists, which would be unnecessary. With this definition, these codes can carry the maximum number of information symbols for a given value of *r*.

We can look at an example Hamming code of the order $r = 3$. This code is defined by the parity-check matrix:

$$H_3 = \begin{bmatrix}1 & 0 & 0 & 1 & 0 & 1 & 1\\ 0 & 1 & 0 & 1 & 1 & 0 & 1\\ 0 & 0 & 1 & 0 & 1 & 1 & 1\end{bmatrix}$$

This is a (7,4)-code with distance 3. From this code it is easy to see that the first three bits of the codeword are the information bits, followed by the parity bits. All possible combinations of the original 3-bit symbol are present in the codeword, making the Hamming code efficient for its distance.

V. Decoding Algorithms

An error correction code is only useful if it can be decoded efficiently and accurately. A nearest neighbor decoding algorithm was studied and will be presented in this section. This algorithm is capable of correcting a single error per codeword. Before introducing the decoding algorithm, we need to discuss the concept of an *error vector*, a convenient way to represent the errors in the received code so we have an algebraic way of manipulating them. We can look at a brief example to further describe the error vector.

Suppose we have a binary codeword, $c = (010111)$, which is transmitted over a noisy channel. Now suppose the receiver sees the codeword, $r = (100111)$. The received codeword can be represented as $r = c + e$ where e is the error vector. In this case e is (110000). If we let H be a parity-check matrix for the linear code C, and we let the weight of the error vector be less than or equal to 1, so that a single-error-correcting code can correct all error patterns that are introduced, then we should be able to get back to the original transmitted codeword. To do this we must compute the following:

$$Hr^T = H(c+e)^T = Hc^T + He^T = Hc^T$$

This is true because $Hc^T = 0$. If the error vector is 0, then this operation also equals 0, and there are no errors in the received codeword. If e does not equal 0, then one of the bits in the error vector is non-zero. Suppose the *i*th bit is in error and we give this bit a value of a. Then $He^T = ah_i$ where h_i is the *i*th column of the parity-check matrix H. With these assumptions, we can define a decoding procedure as follows.

(1) Compute Hr^T

(2) If $Hr^T = 0$, then the received codeword r is accepted as the transmitted codeword.

(3) If $Hr^T = s^T \neq 0$, then compare s^T with the columns of H.

(4) If there is some i such that $s^T = ah_i$, then e is the vector with a in the *i*th position with 0's elsewhere; correct r to $c = r - e$.

(5) If (4) doesn't hold true, then more than one error has occurred and the codeword will need to be retransmitted.

This decoding algorithm provides an efficient way to detect and correct a single-bit error in a transmitted codeword. To show this algorithm working on a real system, we can analyze an example.

Consider a single-error-correcting code C that has a parity-check matrix

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

Let us say that we are transmitting the codeword c = (110100) and the channel introduces a single bit error in the 5th position, making the error pattern e = (000010). With this situation, the receiver would receive r = (110110). Now we must compute Hr^T which is

$$Hr^T = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

This result is equivalent to the 5th column in the parity-check matrix, so we can conclude that the 5th bit is in error, or the error vector is e = (000010). This does match the error pattern that we introduced earlier. Now that we have this error vector, we can subtract it from the received codeword to get the valid transmitted codeword. This would give us:

$$c = r - e = (110110) - (000010) = (110100)$$

This does match our original transmitted codeword, and this process occurred with just one matrix multiplication and one set of additions (binary addition is the same as subtraction). There are other decoding algorithms that are more suitable for correcting multiple bit-errors, but those will be studied in subsequent weeks, when other block coding schemes are looked at.

VI. Conclusion

In this paper, linear codes were introduced and their operation was described. The concept of generator matrices was reviewed and some examples were presented to show how a codeword can be created with one of these matrices, starting from a bitstream at an encoder. Similar to the generator matrix, the parity-check matrix was also introduced. The parity-check matrix can be used in the decoding algorithm to provide means for an efficient decoder. Next, a particular linear code, called the Hamming code, was introduced and the properties of this code were looked at. It was shown that the Hamming code can detect up to two bit errors and can correct a single bit error. Finally, a simple decoding algorithm was presented that can correct a single bit error in the transmit codeword. This algorithm made use of the error vector and the parity-check matrix to correct single bit errors in a Hamming code.

The next couple of weeks will be spent continuing my studies into linear block codes. This will include looking more in depth into specific linear block codes such as Reed-Muller, Reed-Solomon, and BCH codes. The encoders and decoders of these error correction codes will be studied and from their structure, the possible applications will be looked at.

References

[1] San Ling, Chaoping Xing, *Coding Theory: A First Course*. Cambridge University Press, 2004

[2] Andre Neubauer, Jurgen Freudenberger, Volker Kuhn, *Coding Theory: Algorithms, Architectures, and Applications*. John Wiley & Sons Ltd., 2007

[3] Scott A. Vanstone, Paul C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*. Kluwer Academic Publishers., 1989