A Mathematical Approach To Linear Error-Correcting Codes

Anna L. Villani
Department of Mathematics and Computer Science College of the Holy Cross, Worcester, MA , 01610 USA

The world is a social place, and people rely on a mechanism for them to pass information between each other. Communication is necessary not only for people to sustain relationships, but for businesses to thrive as well. In today's world, most people use computer networks to communicate with people whom they do not have direct contact with. It is clear that networks allow for fast and reliable communication, but there is a lot that goes on behind the scenes to ensure this. The information is passed through a series of layers in the network down to the transmission of bits across a physical wire. Bit is short for binary digit, which is an element of the set: S = {0,1}. Throughout these layers, the information is susceptible to corruption. The computer system must be able to detect when codes have been corrupted, and have a means for correcting them. *Coding theory* is the study of methods to transmit information reliably over imperfect communication systems. One very useful method is *error-correcting coding,* which is the act of efficiently adding redundancy to a message so that corruption can be detected and the message can be recovered.

I. Introduction

Before delving into error-correcting codes, it is important to have a basic understanding of how computer systems communicate with each other. A *communication system* is a means for transmitting information form a source to a destination through a channel. A communication can be as simple as two people standing in front of each other talking, or as complex as two people communicating across a network. Often, the information being transmitted needs to be encoded at the source and then decoded at the destination. One reason for this is that the information the source has may be of a different type than the information the receiver is expecting. A simple example of this would be communication between two people who do not speak the same language. A translator would be both the encoder and the decoder in this case. He receives information from the source and encodes it into a language the receiver understands. The receiver passes along a response, and the translator decodes it back to the original language the source can understand. An example regarding computer system communication is the information at the source is made up of analog signals and the receiver is expecting digital signals.[3] The encoder and decoder would be used in a similar way as the previous example. The encoder and decoder are both places where it is likely for the code to get corrupted. Networks can be physically improved to prevent the bits traveling through from getting corrupted, but it is timely and expensive to change the physical components (i.e. wires, etc.) of the network. Even if this was done, there would still be a small chance that errors would occur anyway. It is more efficient to develop algorithms for the system so that it is able to deal with errors when they do occur. The object of an error-correcting code is to notice when an error has occurred in a code, and recover the correct code. If two people are standing in front of each other talking, and the sent message gets garbled, the receiver will almost always be able to immediately tell there is a mistake. If the error is small (i.e. the speaker mixes up a letter or two), the receiver can usually fix the error himself. If there is a large chunk of information garbled, the receiver may need to ask the sender to repeat the message. This analogy can be applied to computer systems. If there are a small number of bit errors, which occurs when a binary digit gets corrupted (or flipped), it most likely can be corrected using error-correcting codes. If there is a large stream of errors, then the message will need to be resent. The difference between people and computers is that a computer receives a string of 0's and 1's, so the computer itself cannot tell if an error has occurred. Thus, it must rely on some mechanism that can.

II. Error-correcting Linear Codes

A. Terms

Since error-correcting codes are only useful for bit errors, any further code errors discussed are assumed to be bit errors. Bits belong to a finite field of order 2. A *finite field* is a field with finite field order, which is a prime or power of a prime and for each prime power there exists exactly one finite field GP(p^n) = F_p^n. A finite field of order 2 is GP(2) = {0,1}. Let the space V consist of all n-tuples of 0's and 1's with addition of vectors modulo 2. A vector is simply a string of 0's and 1's, and their addition modulo 2 means that $1+1 \equiv 0 \pmod 2$. A *code* is a finite set C= {$c_1, c_2, \dots, c_n$} with the strings c_i € V for each i = {1,2,…,n} called *codewords*. The number of bits in each codeword is the length of the code, and the number of codewords in each code is the size of the code. A binary code is linear if the sum of any two vectors in the code is also in the code. An *(n,k) linear binary code* is the set of all linear combinations of k linearly independent vectors in V, called the *basis* of C. To construct this code, start with the k linearly independent vectors. Take the sum of each possible pair of vectors and those sums with the k linearly independent vectors make up the code.

B. Linear Codes

Now let's consider an (n,k) linear binary code C that corresponds to a message to be sent. Before it is sent, the message is *encoded* by adding redundancy bits. The code to be sent now consists of both *information bits* and *redundancy bits*. If x is the codeword to be sent and y is the codeword received, then the *error word* is:

$$e = |y-x|.$$

The error word is used to compare the word that was sent and the word that was actually received. If e > 0, than an error has occurred. The *Hamming distance*, d(x,y), between two words x, y is the number of coordinates in which they differ. The *Hamming weight* of e, w(e), is the number of nonzero entries in e. The minimum distance of a code C is the minimum Hamming distance between any distinct codewords in C.

With some thought one could notice that d(x,y) = w(e), but it is seen clearly with this example:

Let x, y be codewords in an (6,2) linear binary code C. Suppose x = $\underline{0}0010\underline{0}$, y = $\underline{1}0010\underline{1}$. These two words differ in two postions, so d(x,y) = 2.

$$e = |y-x| = 100101 – 000100 = 100001.$$

The number of nonzero entities is two, so w(e) = 2 = d(x,y). So, the minimum distance of a code C is the same as the minimum weight of the code.

III. Methods

A. Simple Methods

The process of adding redundancy digits to a codeword depends on the method for developing an error-correcting code. One very simple method is to simply repeat the word three times. For example, consider the codeword 1010. Using this method, the message is encoded by repeating the code three times to obtain the string 101010101010. This message is sent into the network and along the way gets corrupted so that the receiver gets the string 101010101011. Using a majority rule, the receiver can detect that the error occurs in the last chunk of bits, because it is different from the previous two chunks. This method is extremely inefficient in practice for large codes and can only correct single-bit errors. Another very simple method that is used to detect errors is the parity check. This is done by adding one extra digit to the end of the message. The digit is a 0 if the number of 1's in the message is even and 1 if it is odd. Suppose the code 1001010 is received. It is obvious that there is an error because there is an odd number of 1's in the message, but the last bit tells us there should be an even number. This method is a lot more efficient than repeated codes, but it can only detect if an error is present and has no way of correcting it.

B. Syndrome Decoding

The previous two examples theoretically work and are very easy to understand, but they do not accomplish very much. In a real world setting, large chunks of data are continuously being transferred across a network and thus many instances of errors can occur. In general, it is easier to design an algorithm for a specific code than one that will work for any code. The next algorithm to be explored will detect and correct errors for any code, but it is more efficient than the repeated message method. Named syndrome decoding, it is more complicated than repeated message. Some theorems and definitions are needed before it can be fully defined. Recall a (n,k) linear binary code that is generated from linear combinations of the k linearly independent vectors of V. The matrix whose rows are the k independent vectors is called the *generator matrix* (G) for that code. A generator matrix is called *systematic* if and only if it has the form (I|A), where I is a k*k identity matrix and A is a k*(n-k) matrix. This is equivalent to saying the matrix is in *reduced echelon form*. A code C has a systematic generator matrix if the first k columns of any generator matrix are linearly independent. If the conditions do not hold for some code C, the coordinates can be permuted to obtain an equivalent code C* for which the conditions do hold. This can be done because equivalent codes have exactly the same error-correcting properties[2].

IV. Algorithm

A. Creating Standard Array and Finding Coset Leader

For the remainder of the paper, any code discussed will be assumed to fall into one of the two cases:

Case 1: C has a systematic generator matrix

Case 2: C does not have a systematic generator matrix. The coordinates of C have been permuted to obtain equivalent code C*, and C* has a systematic generator matrix.

If C is an (n,k) linear binary code over F_p^n, the *parity-check matrix* of C is an (n-k)*n matrix, H, over F such that for every c F_p^n: c C <=> $Hc^T = 0$.

Theorem 1: If G is a systematic generator matrix (I|A), then H is the (n-k)*n matrix $(-A^T|I)$.[3]
(In the binary case, A = -A)

To use syndrome decoding, cosets are needed. Let C be an (n,k) linear code over $F_p^n(p^n = q)$ and let a V.

The *coset* of C (represented by a) is a + C = {a+c|c C}

The vector with the smallest weight in the coset is known as the *coset leader*. If y is the received vector, it can be expressed as x + e with x as the original message and e as the error vector. It is clear that y is some coset of C, containing all the expressions e = y - x for each x C. In order to be able to use cosets to correct errors, a standard array for the code C is constructed. The *standard array* for a code C is defined to be a table of vectors whose rows are the cosets of C arranged as follows:

1) The first row is C, with zero vector in first column

2) The remaining rows have a coset leader of C in the first column with the remainder of the row constructed by adding the coset leader to the corresponding codeword in that column.

The coset leader is important because we want to find the vector x that is closest to y, so we want e to be as small as possible

B. Example

Let C be a linear [5,2] code over F_2 with G = = $(I_2|A)$. From Theorem 1 above, H is a 3*5 matrix. The standard array for this code is represented by **TABLE 1.** Using this table, find the coset leader e in the coset y + C and decode y to x + e. Since e is the coset leader, it is the smallest vector in the coset. Thus, x must be the closest vector y. For example, suppose the received word is 11011. It is clear that the word is incorrect because it is not in the code C. Using the table, its corresponding coset leader is located, 10000. Subtracting 10000 from 11011 gives 01011, which is the correct codeword. Subtraction is not even needed if the standard array is accessible because the correct codeword can be found by tracing up the column of the received word.

TABLE 1.

00000	10110	01011	11101
00001	10111	01010	11100
00100	10010	01111	11001
01000	11110	00011	10101
10000	00110	<u>11011</u>	01101
00101	10011	01110	11000
10001	00111	11010	01100

C. Syndromes

Creating the standard array and using it to find a vector y and then decoding it to find x is another method that theoretically works. However, in practice the code will be a lot larger than the example. If C is a (200, 160) linear binary code, then the standard array will have 2^{200} entries. To remove a lot of these entries, the syndrome of a vector is used. If H is a parity check matrix of an (n,k) linear binary code C with rows $h_1,h_2,\ldots,h_{n-k}$ and y is a vector in V, then the *syndrome* of y is the column vector with each row in H multiplied by y:

$$\text{syn}(y) = Hy^T$$

Let a + C, be a coset of C. Take two elements from the coset: a $+c_1$, $a+c_2$ with c_1, c_2 C. Then $h_i(a+c_1) = h_i{*}a + h_ic_1$ for each row h_i of H

$= h_i{*}a$ ($Hc^T=0$, so $h_i{*}c_i = 0$ for each $i=0,1,\ldots,(n-k)$)

$= h_i{*}a + h_ic_2$

$=h_i(a+c_2)$

This proof shows that if two vectors are in the same coset, then they have the same syndrome. This fact leads to the theorem:

> <u>Theorem 2:</u> Every vector in a fixed coset has the same syndrome. Vectors in different cosets have different syndromes. All possible q^{n-k} syndromes occur as syndromes of some vectors[2]

Since $Hc^T = 0$ when c is a codeword in C, then we can deduce that if the received vector y is a codeword, it has no errors. Thus, y = 0. If $y \neq 0$, then $\text{syn}(y) = Hy^T = Hx^T+He^T = He^T$. This leads to the theorem:

> <u>Theorem 2m:</u> If C is a binary code and e is a coset leader, the syndrome of e is the sum of those columns of H where e has nonzero components[2]

Now, syndrome decoding can be fully defined:

1) Start with a given generator matrix G
2) Find all the codewords of C using the matrix G
3) Find H using G
4) Start with the zero vector as the first coset leader.
5) Choose coset leaders of weight one and compute the syndrome. If a new syndrome is found then it is added to the list
6) Compute syndromes for all coset leaders of weight i, i+1, i+2,… until all q^{n-k} syndromes have been found.

With this complete list of syndromes, a received vector y can easily be decoded to the correct vector x. Calculate the syndrome of y and find it in the list. That syndrome corresponds to a coset leader, which is subtracted from y to obtain x. Consider the same example from above with generator matrix G = . H is a 3x5 matrix and the coset leaders are all 1x5 matrices. Multiplying H by the transpose of the coset leader (5x1 matrix) results in the syndrome as a 3x1 matrix. Start with zero vector first and compute syndrome:

00000

Coset leaders of weight 1:

00001 00010 00100 01000 10000

Coset leaders of weight 2:

00101 10001

All other coset leaders of weight 2 produce syndromes that have already been computed, so we ignore them. All the coset leaders of weight two should be tested, or until $2^3 = 8$ syndromes are found (q^{n-k} syndromes, n=5, k = 2). Now considering the same received word, y = 11011, the correct word can be recovered without using the standard array. Multiply the word by H by the transpose of y using matrix multiplication. Looking in the list of syndromes, find the corresponding coset leader, 10000 and subtract this from y to obtain the correct codeword.

IV. Limitations

The use of syndromes in this algorithm improves the efficiency of using cosets when the redundancy n-k is small. The (200,160) linear binary code from above improved from a table of 2^{200} entries to a table of 2^{40} entries, which is a dramatic improvement. Since 2^{n-k} syndromes need to be computed and stored in a list, this algorithm is not suitable for codes with very large redundancies. This algorithm also only works for small bit errors. Considering all of the networks in existence and running right now, there are countless instances of more complicated errors than bit errors, and codes with large redundancies. So it is clear that this algorithm does not even come close to completely covering error-correction. In fact, there are many different algorithms to deal with certain kinds of errors and certain kinds of codes. In general, it is easier to develop an algorithm for a specific code than to find one that works for all different kinds of codes. Since one algorithm cannot handle every case of error in a network, the variety of different algorithms is meant to 'fill' all the different kinds of gaps (or errors) in a network. Communication across networks is still not perfect and can always be improved, so coding theory is an area of study that is still very prevalent today.

References

1. Michael, T. S. "Chapter 7." *How to Guard an Art Gallery and Other Discrete Mathematical Adventures*. Baltimore: Johns Hopkins UP, 2009. N. pag. Print.
2. Pless, Vera. "Chapters 1 & 2." *Introduction to the Theory of Error-correcting Codes*. New York: Wiley, 1998. N. pag. Print.
3. Roth, Ron. "Chapters 1 & 2." *Introduction to Coding Theory*. Cambridge: Cambridge UP, 2006. N. pag. Print.