

9. Matrix Hamming Codes

We now address the question: *how can we construct useful error-correcting codes that are easier to handle than randomly chosen codes?*

Suppose that the messages we will want to encode are all sequences of 0's and 1's. We shall design a system to take any sequence of length M , which we call the **message word**, and encode it by turning it into a somewhat longer sequence of length N called a **code word**, in such a manner that we can retrieve our message word from the code word, even if one of the digits has been altered. If m is our message, we will say that $c(m)$ is its code word.¹

To simplify coding and decoding, we start by giving structure to the message words and code words. Namely, we introduce the notion of **addition**, both of message words and code words, so that we treat both of these as vectors rather than mere sequences.

The notion of addition that we introduce is not the ordinary addition of number you are used to. It differs in two respects: first, 2 is the same as 0, so $1 + 1 = 0$. Second, there is no carrying; each bit is independent of each other bit. (Those familiar with modular arithmetic will recognize this as addition mod 2).

Thus, for example, $1011110 + 1101011 = 0110101$ in this system.

Remember, in everything that follows $2 = 0$!

9.1 Linear Codes

The first step we take toward creating codes that are easy to encode and decode is to look at **linear codes**, which are also called **matrix codes**. This means that the code word for the sum of two message words will be the sum of the code words of the two message words. So if x and y are two message words, then the following is true for a linear code:

$$c(x) + c(y) = c(x + y)$$

This means, for example, that the code word for 0110101 must be the sum of the code words for the two other sequences above that sum to it. Furthermore, any code word can be written as the sum of two other code words, and the sum of any two code words is another code word.

¹ It is not an unrealistic simplification to look at this problem in terms of binary messages because with the advent of computers, almost all messages are converted to binary before being stored or transmitted.

This simple condition produces a tremendous simplification in encoding. It means that to define a linear code we need only provide the code words to the members of a basis in the space of messages. Then, since each message can be written as a sum of the basis vectors, the code word for any message is just the sum of the code words for the basis vectors that sum to it. (If you have not taken linear algebra this may not make sense to you. Consult a linear algebra textbook, a professor, or a fellow student to get the definition of a basis for a vector space).

We now make some helpful definitions.

The **weight** of a word or vector is the number of 1's in it. So 0110101 has a weight of 4.

The **standard basis** for message words will be the vectors of weight 1; if that 1 occurs in the j -th place in the message, we denote it as $\langle j \rangle$. So, for example, 1000, 0100, 0010, and 0001 are the standard basis vectors of length 4, and $\langle 3 \rangle$ is the vector 0010 since the 1 is in the third place of the message vector.

We can, as noted above, define our code completely by assigning a code word to each of the message vectors $\langle j \rangle$. We form these code words into a matrix C , called the **encoding matrix**, whose j -th row is the code word for the message vector $\langle j \rangle$.

Here is an example of a code matrix C , which defines a code with $N=7$, given a message space with message vectors of length 4. We have 4 basis vectors so our matrix will have 4 rows, each of length 7:

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

The code word for $1010 = \langle 1 \rangle + \langle 3 \rangle$ will then be the sum of the first and third rows of the matrix, and a similar statement holds for any other combination of the basis vectors.

The rule for encoding a message vector \mathbf{m} is:

$$C(\mathbf{m}) = \mathbf{m}C$$

where the product $\mathbf{m}C$ is ordinary matrix multiplication² of the row vector \mathbf{m} and the matrix C (we use bold face to denote a vector).

² The rule for matrix multiplication is that you take the dot product of each row of the left factor with each column of the right factor and put the result in the row corresponding to that of the left factor and the column corresponding to the column chosen in the right factor.

In our case, \mathbf{m} has only one row while C has 7 columns, so our code word then is row vector of length 7.

In this case, we need only define code words for 4 basis vectors in order to define code words for 2^4 or 16 potential messages. With longer message vectors, the difference between M and 2^M becomes much greater. Thus, for $M=10$ we have only to define 10 basis message code words to provide a code that allows 2^{10} or 1024 possible messages.

9.2 Constructing Linear Error Correcting Codes

We have now shown how we can rather quickly develop a code by simply finding code words for the basis vectors of the message space. However, we have not yet addressed error correction, which is what we are trying to do. We now ask: *what do we have to do to obtain a single error correcting linear code?* (Later we will see how to correct more than 1 error).

If we are using messages of length 4 and codes of length 7 as in the example above, we see that of the $2^7 = 128$ possible sequences of length 7, only 16 of them are code words. This means that if an error is introduced into a code word, and one bit is changed, then it is likely that the new sequence may not be a code word at all. This fact will be very helpful in the discussion that follows.

We can decode a received message, one that may have an error, by finding the code word that differs in the fewest bits from it. Thus, we will look for the code word whose Hamming distance from the received word is smallest. If the received message is a code word, then we will assume that no error has been introduced and just decode it as it stands.

However, what if the error is such that it changes one code word into another? In this case, there would be no way for us to detect whether or not an error has occurred. Furthermore, it could be that there are two code words that both have a Hamming distance of 1 from the received message. If we seek the ability always to be able to find a unique code word when there is one error, there must be only one code word, the true one, that has Hamming distance 1 from the received word; all others must have distance at least 2 from it, and hence at least 3 from the true code word.

Thus, to be able to correct 1 error always and uniquely, the minimum Hamming distance between code words must be at least 3. Similarly, if we want to be able to correct k errors, the minimum Hamming distance between code words must be at least $2K+1$.³

³ Another strategy here is to use a code to detect errors rather than correct them. The receiver can then request that the message be re-sent if errors are found. In this case, we just have to guarantee that the Hamming distance between code words is $k+1$. When the received word is not a code word, we report that errors have been made.

So we now ask: *how can we construct a matrix code with minimum Hamming distance 3?*

We first note that the Hamming distance between two code words is the weight of their sum (See the exercises at the end). This means that if all weights of non-zero code words are at least 3, the minimum Hamming distance between code words will be at least 3. To see this, consider any two code words A and B. Their sum, C, will be another code word, and if all code words have weight at least 3, then C will have weight at least 3. And, looking at the first statement of this paragraph, this tells us that A and B will have Hamming distance at least 3, which is what we want. This is true for any code words A and B, and thus for any code words in general.

We now try to construct a code where every code word is at least weight 3.

One convenient form of a code (but not the only or most convenient form) is to have the first bits of the code word state the message exactly, and then tack on extra “check bits” that provide the redundancy needed to allow for error correction.

In that case the left portion of the encoding matrix is an identity matrix, which, when multiplied by the message vector will give back the message if there are no errors. This identity matrix already has weight 1 in each of its rows, so the rest of the matrix is only required to have weight 2 in each row.

(Notice that if a message word has weight 3 or more in itself, the weight of the code word will also be at least 3 since the message here forms a part of the code word. Therefore, we only need to worry about the weight 3 criterion for code words of messages whose weight is 1 or 2).

We want our basis code words to all be of at least weight 3, and to differ from each other by at least 3 bits. We can accomplish this by giving each row a weight of at least 2 in its check bit part, and by making the check bit part of each row different from that of the other rows.

Let us construct some codes by these rules. First, suppose that we want to use 2 check bits. Then there is only one possible arrangement of check bits with weight 2, and therefore is at most 1 row to our matrix, which will have the form (11).

This is kind of a naive code: it corresponds to repeating each bit three times. You can then decode by using majority rule on each bit of the message.

Suppose that we use 3 check bits. Then there are 4 possible check bit rows, (110, 101, 011, 111) and will result in a code like the one given in section 9.1. Such a code can have up to 4 rows.

Suppose we use k check bits. The matrix for such a code will then consist of an identity matrix with an M by k matrix of check bits on its right. The number of possible

rows, that is, the value of M for which such a code can correct an error is the number of distinct vectors of length k with weight at least 2. (See the exercises for more on this).

Codes of this kind (but not necessarily in exactly this form) are called **Hamming codes**.

9.3 Finding and Correcting an Error in a Hamming Code

Suppose now that you are using one of the codes described in the previous section, and you receive a message, \mathbf{R} . You are then faced with the task of determining whether there is an error, and if so locating it so that you can decode the message.

The received message \mathbf{R} here consists of the encoded message plus perhaps an error, which we here can assume is in a single bit. We note that changing one bit at position j is equivalent to adding $\langle j \rangle$ to our message. In this case, if the error is in bit j , we can write:

$$\mathbf{R} = \mathbf{mC} + \langle j \rangle$$

so in general we have

$$\mathbf{R} = \mathbf{mC} + a\langle j \rangle$$

where a is 0 or 1 and j is the up to now unknown error location. We use a because we are assuming that we do not know whether an error has occurred, and we let $a = 0$ correspond to the no error case.

To find the error, if any, we use a “message killer” which eliminates the message part of \mathbf{R} and allows us to focus on the error. For a message killer, all we need is a matrix D that obeys $CD = 0$ and whose rows are all distinct. If we have such a matrix, then we can form the matrix product \mathbf{RD} , with:

$$\mathbf{RD} = \mathbf{mCD} + a\langle j \rangle D = a\langle j \rangle D$$

If we get the zero vector as an answer here, we can conclude that $a = 0$ and there was no error, otherwise we get $\langle j \rangle D$, which is the j -th row of D by matrix multiplication.

This leads to the important question: *How do we find our D matrix message killer?*

Recall that our encoding matrix C is an identity matrix followed by another (M by k) matrix, which we will call Z . Our matrix C then can be written as $[I_M \ Z]$, where I_M is the M by M identity matrix. Z then is the matrix consisting of the check bits only.

We then find that the matrix with Z above a $(k$ by $k)$ identity matrix I_k , written as $\begin{bmatrix} Z \\ I_k \end{bmatrix}$, is a message killer. This is because $CD = I_M Z + ZI_k = 2Z = 0$, since $2=0$.⁴

So here is a procedure for error correction:

1. Form the matrix D that consists of Z above an identity matrix.
2. When you receive your message \mathbf{R} , form the matrix product \mathbf{RD} . This will be a vector of length k .
3. Identify which row of D is equal to this vector. Its row number will be j , the error bit location. Change that bit and you have corrected the message. This gives us the original sent message.

Example:

We will use

$$C = [I \ Z] = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

as before. In this case, Z is the matrix consisting of the last 3 columns of C . Since D is Z above an identity matrix, we get that D is:

$$D = \begin{bmatrix} Z \\ I \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, suppose that $\mathbf{m} = 1101$. Then $c(\mathbf{m}) = \mathbf{mC} = 1101010$. We will introduce an error at $j=5$, so $\mathbf{R} = 1101110$. Now, we see that:

$$\mathbf{RD} = 100$$

which corresponds to the 5th row of D , and thus tells us, correctly, that the error is at $j=5$. This tells us that $c(\mathbf{m})=1101010$ and we take the first four digits of this to get our original message \mathbf{m} .

⁴ The first equality follows from block-matrix multiplication, which is explained in more detail in notes 25

9.4 More on Hamming Codes

So far, all of the Hamming encoding matrices we have considered are in the form $I_M Z$, as discussed above. These matrices lead to code words that consist of the original message followed by k check bits. However, this is only the simplest class of Hamming matrices. We shall see that it is not generally necessary that a Hamming matrix code be of this form.

Suppose we have an $I_M Z$ code matrix C ; since each row of C is the code word for a basis vector of the message space, this matrix will describe a code whose code words are all the linear combinations of the rows of C . Note that since part of C is an identity matrix, all of its rows are linearly independent. This is important because otherwise two different messages could have the same code word, and that would prevent our code from being able to correct errors.

Now, we can take any set of M linearly independent code words (that is, linearly independent linear combinations of the rows of C) and make them into a matrix Q . The linear combinations of Q 's rows will be the same as the linear combinations of C 's. The reason we use linear combinations of the rows of C is that we still have to guarantee that our code words are at least of weight 3. They have to be linearly independent for the reason specified in the previous paragraph.

For example, given the matrix used in previous examples:

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

we can replace the first row by its sum with the second and third, and get a new matrix Q :

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

Since all the rows of Q are linear combinations of the rows of C , any linear combination of the rows of Q is also a linear combination of the rows of C and thus a code word of C . Thus, C and Q have the same code words.

Recall that the meaning of a coding matrix, say Q , is that the basis vector $\langle j \rangle$ gets the code word given by the j -th row of Q . This means, for example, that, using coding matrix Q , the message (1110), which is $\langle 1 \rangle + \langle 2 \rangle + \langle 3 \rangle$, gets the code word $1110000 + 0100011 + 0010101 = 1000110$.

Note that if we had started with Q , by adding the second and third rows to the first we would get C . This holds in general. Given any encoding matrix Q , you can, by replacing rows with their sums with other rows judiciously (also known as row reduction), put it in $I_M Z$ form. The rows of the new matrix will be the code words not of $\langle j \rangle$ necessarily but rather of linear combinations of them.

For example, suppose the coding matrix you were using was Q . Then when put in $I_M Z$ form it becomes the matrix C above. The first row of C will not be the code word for $\langle 1 \rangle$, but rather the code word for $\langle 1 \rangle + \langle 2 \rangle + \langle 3 \rangle$, since we added rows 1, 2, and 3 of Q together and put the result in row 1 in order to put Q in $I_M Z$ form.

Now suppose we had a Hamming matrix like Q and wanted to use it as an error correcting code. First, we would put it in $I_M Z$ form then find the error in the manner described in the previous section. This will give us the original message, but in terms of the basis of the $I_M Z$ matrix. Let's call this message s . In order to get back the message in its original form, we add together the rows of Q that correspond to the 1's in s . So if there are 1's in positions 1, 3, and 5 of s , then our original message was the sum of rows 1, 3 and 5 of Q .

Let us look at an example to clarify this process. Suppose that using the matrix C we error corrected and decoded our received message to (1100). Then we would decode this message as the sum of the messages whose code words were the first two rows of C . This is the sum of (1110) and (0100), so decoding would give us (1010) as the original message.

We can put this procedure in a different perspective as follows. To go from Q to C we are actually multiplying on the left by the inverse of the matrix consisting of the first M columns of Q . This, after all, is what makes these first M columns into an identity matrix.

If we denote those columns by Q_M , then we have $C = Q_M^{-1}Q$, and C will be in $I_M Z$ form. If we receive the word R and correct it to the code word sC , this will be $sQ_M^{-1}Q$, so our message m will be sQ_M^{-1} (recall that $c(m)$ with respect to code Q is mQ).

9.5 Some Comments

These single error correcting matrix codes still leave several things to be desired.

First, they do not give us much help toward constructing codes that correct several errors. This is important since although we saw that the ratio of check bits to message bits is much smaller for larger M and k , with such long messages you have to worry about there being more than one error.

A second problem with the Hamming Matrix codes is that in order to describe one you need to write out a whole (M by k) matrix Z . This is no particular problem for $M=7$, but it is mildly cumbersome for M on the order of thousands.

In the next few sets of notes, we will delve further into this problem by introducing more structure on our message sequences. We have defined addition for them and treated them as vectors. To create multi-error correcting codes, we shall define multiplication, and treat them as polynomials.

Exercises

Exercise 1 Show that the Hamming distance between two code words is equal to the weight of their sum

Exercise 2 What is the number of message bits that can be handled using k check bits in a single error correcting code? Construct a single error correcting code with maximal M value for $k = 4$

Exercise 3 Construct a coder and error finder for your code using a spreadsheet which does the following: If you input an 11 bit message it forms the code word and if you give a 15 bit received word it locates the error, fixes it, and gives the original message.

Hint: Matrix multiplication can be done on a spreadsheet with one tedious instruction judiciously copied. Fixing the error is easy if you give each row of D a numerical identifier; then you can locate which row of D RD is from its identifier, and correct the appropriate entry of R . A good identifier for a row of D is the decimal number that it is the binary representation for.

Additional Sources:

Jacobs, Bill. *Linear Algebra*. (1989). P. 132-136.

~Edited by Jacob Green