

9. Matrix Hamming Codes

9.1 Linear Codes

We now turn to the question: how can we construct useful codes that are easier to handle than randomly chosen codes?

The first step we take toward creating codes that are easy to encode and decode is to look at **linear codes**. These are also called **matrix codes**.

Suppose that the messages we will want to encode are all sequences of 0's and 1's of length M . Our code words will be somewhat longer sequences, of length N , of the same two symbols.

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 numbers you are used to. It differs in two respects:
First, 2 is the same as 0. So 1 and 1 is none.
Second, there is no carrying: each bit is independent of each other bit.

Thus, for example, the sum of 1011110 and 1101011 is 0110101 here.

We now require that our code be **linear**. This means that the **code word for the sum of two words will be the sum of the code words of the two summands**:

$$\mathbf{c}(\mathbf{x}) + \mathbf{c}(\mathbf{y}) = \mathbf{c}(\mathbf{x} + \mathbf{y}).$$

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.

This simple condition produces a tremendous simplification in encoding. For, it means that to define a linear **code we need only provide the code words of the members of a basis in the space of messages. The code word for any sum of basis vectors must then be the same sum of their code words.**

We now define some notations.

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

The standard basis among message words will be: the vectors of weight 1; **if that 1 occurs in the j -th place in the message, we denote this basis vector as $\langle j \rangle$.**

We can, as we have noted, define our code completely by assigning a code word to each the various basis 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:

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

The code word for $\langle 1 \rangle + \langle 3 \rangle$ 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 implied by the matrix C for encoding a message vector m is:**

$$C(m) = mC,$$

where the product mC is **ordinary matrix multiplication of the row vector m and the matrix C .**

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 it in the row corresponding to that of the left factor and the column corresponding to the column chosen in the right factor. Because we are working with addition mod 2, we must use the dot product mod 2.

In our example m has only one row, while C has 7 columns. Our code word mC then is a 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 encodings for 2^{10} or 1024 possible messages all together.

9.2 Constructing Linear Error Correcting Codes

We now ask, what do we have to do to obtain a linear code that can correct a single error? Or a code that can correct k errors?

We will decode a received message by finding the code word that is closest to it, in that it differs in the fewest bits from it. Thus we will look for the message whose Hamming distance from the received word is smallest.

If we seek the ability always to be able to find a unique such 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 **a codeword with one 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 $2k+1$.**

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 that case to be able to detect up to k errors in all cases requires minimum Hamming distance between code words of $k+1$. 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 notice that **it is sufficient to arrange it so that the minimum weight of a code word is 3.**

Why? Because the **Hamming distance between two code words, A and B say, is the weight of their sum $A+B$, which sum is another code word.** This means that if all weights of non-zero code words are at least 3 (or k) the minimum Hamming distance between code words will be at least 3 (or k).

We now try to construct codes with minimum weight 3.

One convenient form of a code (but not the only form and not even the 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 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.**

We want to apply the weight at least 3 criterion to **all** code words other than the all 0 word.

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.**

Each message of weight 1 corresponds to a row of C , so if we require that the non-identity part of this row have weight at least 2, that will give the code word weight at least 3.

For messages of weight 2 we only need that the non-identity part of its code word have weight at least 1; and this only means that the two rows that add up to give the non-identity part of its message **cannot be identical.**

If these rows, beyond the identity matrix are different, the weight of their difference is that of their sum and it will be at least 1.

We conclude that all we need to construct a single error correcting linear code of the kind we are considering is to

- a. give each row weight at least two in their check bit part and**
- b. make all of them different from one another in their check bit part.**

Let us construct some codes by these rules.

First suppose we want to use 2 check bits.

Then there is only one possible arrangement of check bits and therefore is at most 1 row to our

matrix, which will have the form (111).

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

OK, **suppose we use 3 check bits**

Then there are 4 possible check bit rows, (110, 101, 011, 111) and a code like the one given at the beginning of the chapter will be a single error correcting code. 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 a M by k matrix of check bits on its right, as in the example at the beginning of the chapter for k=3.

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.**

Exercise: 9.1 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.

Codes of this kind (but not necessarily in the message bit check bit form) are called **Hamming codes.**

9.3 Finding and Correcting an Error in a Hamming Code

Suppose now that you are using one of these codes, and you receive a message, **R**. You are then faced with the task of determining whether there is an error, and if so locating it; so that you can discover the message that was encoded when it was sent.

The message R here consists of the encoded message plus perhaps an error, which we here can assume is in a single bit. If the error is in bit j, we can write

$$R = mC + \langle j \rangle$$

so in general we have

$$R = mC + a \langle j \rangle$$

where a is 0 or 1 and j is the up to now unknown error location; a=0 corresponds to the no error case.

To find the error, if any, we use a "**message killer**" which eliminates the message part of R and allows us to focus on the error.

For a message killer, we want some kind of operation which eliminates the codeword and leaves a part which only depends on the error. Since we're dealing with linear codes, the natural operation to use is matrix multiplication. So we want some kind of matrix D that takes $mC + \langle j \rangle$ to a vector which only depends on the error and also uniquely identifies the error. That is, we want

$$(mC + \langle j \rangle)D = \langle j \rangle D$$

and furthermore, we want $\langle j \rangle D$ to uniquely identify the error; that is, we want $\langle j \rangle D \neq \langle k \rangle D$ if $j \neq k$. However, $\langle j \rangle D$ is just the j^{th} row of D . So all we need is a **matrix D that obeys $CD = (0)$ and whose rows are all distinct**. If we have such a matrix, we can form RD , with

$$RD = mCD + a \langle j \rangle D = a \langle j \rangle D.$$

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

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; let us call that matrix Z . Our matrix C then can be written as $(I_M \ Z)$, where I_M is the M by M identity matrix. Z 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 is a message killer. For, CD is $I_M Z + Z I_k = 2Z = (0)$ since $2=0$. (We do not need $2=0$ here to make this work, because we could have used $(Z \text{ above } -I_k)$ rather than $(Z \text{ above } I_k)$ to get (0) here in any case.)

So here is a procedure for error correction:

Form the matrix D which consists of Z above an identity matrix.

When you receive your message R form the matrix product RD . This will be a vector of length k .

Identify which row of D is equal to this vector. Its row number will be the error bit location. Change that bit and you have corrected the message.

Then read off the message sent from the resulting message bits.

9.4 Some Comments

These single error correcting matrix codes are lots of fun, but they 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 the ratio of check bits to message bits is much smaller for larger M and k , but if you use such, you have to worry about more than one error.

Second, to describe one you need to write out a whole (M by k) matrix Z . This is no particular problem for $M=11$ and $k=4$, but it is mildly cumbersome for M on the order of thousands.

We will go further by introducing still more structure on our message sequences. We have defined addition for them and treated them as vectors. Next we shall define multiplication, and treat them as polynomials.

Before doing so we address two questions:

1. How do we find errors and decode when the encoding matrix C is not in IZ form?
2. When do matrix C and matrix Q lead to the same code, perhaps with a different correspondence between message words and code words?

The answer to the first is to use the answer to the second. Find a code with the same code words that is in IZ form, then find the error, and the original message in terms of the basis for which the matrix takes IZ form; then take the same linear combination of those basis vectors as the message is of them.

This sounds obscure but isn't. We first answer the second question, and then go through these steps.

Suppose we have a code matrix C ; this matrix will describe a code whose code words are all the linear combinations of the rows of C .

We could 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 , and the linear combinations of Q 's rows will be the same as the linear combinations of C 's. This is little more than the statement that a linear combination of linear combinations is a linear combination.

For example, given the matrix we first gave as an example above,

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

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

$$Q = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

which

will have the same code words. Similarly, had we started with Q , by adding the second and third rows to the first we would get C .

The meaning of a coding matrix, say Q , is that the basis vector $\langle j \rangle$ gets code word given by the j -th row of Q . The preceding remarks tell us that if this is so then the vector (1110) gets the code word 1000110 in the code defined by Q .

Similar remarks hold in general. Given any encoding matrix Q , you can, by replacing rows with their sums with others judiciously, put it in IZ form. The rows of the new matrix will be the code words not of the $\langle j \rangle$ necessarily but rather of linear combinations of them.

If you keep track of which linear combinations get code words given by your IZ matrix, you can decode using it, and take the linear combination of these messages given by your decoding.

For example, suppose the coding matrix you were using was Q . Then when put in IZ form it becomes the matrix C above.

Suppose using the matrix C above we 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 the decoding would give you (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 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 IZ 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} .

One more comment. We have assumed here that the first M columns of Q are an invertible matrix. If somehow this is not so, you can pick M other columns of Q that are invertible and rename them temporarily as the first M columns. If no M columns of Q are invertible your matrix Q is no damned good at all. Several messages will get the same code word, and the weight of their difference will be 0.

Exercises:

9.1 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 $k=4$.

9.2 Construct a coder and error finder and decoder for your code using a spreadsheet, You input an 11 bit message; it should form the code word. If you give a 15 bit received word it locates the error fixes it and gives the message

(matrix multiplication can be done 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 number it represents in binary.)