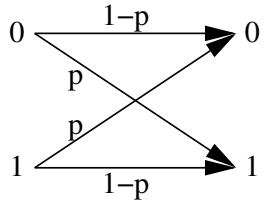
Hamming Code Notes
18.310, Spring 2010, Prof. Peter Shor

Last time we proved a special case of Shannon's theorem: Suppose you have a binary symmetric channel with error probability p.



Then with n uses of this channel we can encode Rn bits for any rate $R < 1-H(p)$ and for sufficiently large n, and have all the encoded bits get through intact with probability nearly 1. We used block coding in that we took blocks of Rn bits for some R less than the channel capacity $1-H(p)$, mapped them into strings of n bits, and showed that it was theoretically possible to decode the output of the channel most of the time.

However, the scheme we came up with used random codewords, and was completely impractical. There were three problems with it:

1. The code cannot be specified compactly; we need to describe the 2^{Rn} codewords.

2. We don't know how to encode efficiently.

3. We don't know how to decode efficiently.

What we will do is show how to fix two of these problems by using *linear* binary codes. These will take care of problems 1. and 2., leaving only problem 3. This we will have to solve by choosing our codes very, very carefully.

A linear code is a code with the property that if c_1 and c_2 are codewords, then $c_1 + c_2$ is also a codeword, where we are adding in binary arithmetic; that is, we use addition modulo 2 (without any carries). Since for any codeword c, we have $c + c = 000\ldots0$, the word $000\ldots0$ must also be in the code.

Suppose that you have a binary linear code. I'd like to claim that you can specify it by using just a basis for the codewords, rather than all the codewords. We pick any set of codewords b_1, b_2, ..., b_n that is linearly independent, that is, there is no non-empty subset $\{b_{i_1}\ldots b_{i_l}\}$ of them such that

$$b_{i_1} + b_{i_2} + \ldots + b_{i_l} = 0.$$

We can restrict ourselves to considering linear combinations of the basis codewords with coefficients 0 or 1 by noticing that, in binary arithmetic, $2 = 0$ and $3 = 1$. Thus, any linear combination with coefficients greater or equal to 2 is equivalent to a linear combination with only $\{0, 1\}$ coefficients.

Using the linearity property, it is easy to show that any word which is a sum of basis elements is a codeword. Conversely, every codeword is a sum of basis elements, since if we had a codeword c which was not a sum of the basis elements, we could include it in the basis. This new set would still be linearly indepednent, because if it were not, then we would have some linear combination

$$c + b_{i_1} + b_{i_2} + \ldots + b_{i_l} = 0$$

which gives us an expression for c as a linear combinations of basis elements.

Finally, no two different linear combinations of the basis elements can be equal. This would give us

$$b_{i_1} + b_{i_2} + \ldots + b_{i_l} = b_{j_1} + b_{j_2} + \ldots + b_{j_{l'}}$$

which, if the set of i_x is different from the set $\{j_x\}$, also gives us a linear combination of basis elements which equals 0.

If we want to have 2^k codewords for some large k, using a linear code means we can specify all these codewords using just k of them. If we want to use blocks of n bits, for $n \approx 10,000$, we need to remember just Rn codewords, which is quite reasonable number for a computer, instead of 2^{Rn}.

Now, suppose we have a set of basis elements. We can write these as rows of what we will call a *generator matrix* G. For example, we might have

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

How do we encode using the generator matrix? Every codeword is a sum of a subset of the rows. To get the codeword that is the sum of the first, second, and fourth rows, we need to add these three vectors in binary. Doing this gives

$$1\ 1\ 0\ 1\ 1\ 0\ 0.$$

Suppose we take the message $m = 1101$ and multiply it by the matrix G. We see that mG is the also sum of the of the first, second, and fourth rows. Thus, to encode a given message, it is sufficient to multiply it by the generator matrix G. Thus, we have solved (1) and (2) of the list of difficulties of our code.

We define the Hamming weight of a string to be the number of non-zero bits it contains (in the binary case, these bits have to be ones). We will define the Hamming distance between two binary strings as the number of bits you have to change in one string to obtain the other string. The Hamming distance $d_H(c_1, c_2)$ between c_1 and c_2 is just the Hamming weight of $c_1 - c_2$. It is easy to check that the Hamming distance satisfies the triangle inequality; that is,

$$d_H(a, c) \leq d_H(a, b) + d_H(b, c).$$

What are the conditions for the code to correct t errors? What we would like is that every possible string is within distance t of at most one legal codeword. If this is the case, then correcting the received message to the closest codeword is guaranteed to give the codeword that was sent, as long as there are at most t errors in it. Suppose there was some string s that was within distance t of two different codewords. Then we would have

$$d(c_1, c_2) \leq d(c_1, s) + d(s, c_2) \leq 2t.$$

Thus, if we make sure that any two codewords are Hamming distance at least $2t+1$ distance apart, this code is a t-error correcting code.

How do we check whether all pairs of codewords are within distance d? There is actually a simpler criterion that tells whether this is the case. The Hamming distance between two codewords, $d_H(c_1, c_2)$, is the Hamming weight of $c_1 - c_2$. But $c_1 - c_2$ (the same thing as $c_1 + c_2$, since we're working in binary arithmetic) is a codeword, because this is a linear code. Thus, if all non-zero codewords have Hamming weight at least d, then the code will correct $(d-1)/2$ errors.

We will now give an explicit matrix construction of 1-error correcting codes. Suppose we have some generator matrix G. We can use row operations (adding one row to another row), and possibly permute the columns, to turn it into a matrix that looks like $(I|S)$ where I is the identity and S is some 0-1 matrix. If you add one row to another row, you have not changed the set of codewords (as the set of codewords correspond to the sum of all possible subsets of rows), and you can make the matrix G simpler by doing this. If you've seen Gaussian elimination in linear algebra, this is the technique we use. Here is an example. Consider the following matrix G':

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

Adding the first row to the second and fourth (to clean up the first column), we

get:

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

Adding now the second row to all other rows, we obtain:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{pmatrix}.$$

The first two columns now match the first two columns of an identity matrix. To clean up the third column, we add the third row to the last row to get:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{pmatrix}.$$

The first four columns now form an identity matrix. In fact, the resulting code is essentially the same as the one given by G (after permuting certain rows and certain columns).

Now, let us assume that the matrix G in the form $(I|S)$. What do we require on S for every non-zero codeword to have weight at least 3. First, let's worry about the codewords which are the rows. We need them to have weight at least three, so we need that all the rows of S have weight at least two. Second, what about the codewords that are the sums of two rows? These have weight two in the first part, so we need to worry about the sum of two rows of S; this sum must have weight at least 1. This translates to the requirement that all rows of S are different. Sums of at least three rows automatically have Hamming weight at least three, so that's it. Thus, we can create a Hamming code by taking a matrix G of form (I, S) where

- all rows of S contain at least two 1's.
- no two rows of S are the same.

For example, if we took a 4×4 identity, we might get the following generator matrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Choosing a generator matrix G of this form has another real advantage: if there are no errors then it makes the code extremely easy to decode. Say the codeword is $c = mG$. If the first k columns of G form the identity matrix I then the first k bits of the codeword c are exactly m.

How big a code can we obtain in this way? Let the identity I be of size $k \times k$ and S be $k \times m$. (So G is $k \times n$ with $n = m + k$.) There are 2^m possible binary rows of length m for S, but we have to eliminate the ones with Hamming weight 1 (there are m of these) and the one with Hamming weight 0. This leaves $2^m - m - 1$ possible rows for S. Thus, if we take $k = 2^m - m - 1$, we get that $n = k + m = 2^m - 1$, and we obtain a Hamming code that maps $2^m - m - 1$ bits into $2^m - 1$ bits, and corrects one error. The simplest such code comes when $m = 2$. This (go ahead and check this statement) is the repetition code which maps $1 \to 111$ and $0 \to 000$. The next case is when $m = 3$, and is the Hamming code encoding 4 bits into 7 bits. When $m = 4$, we encode 11 bits into 15 bits, and when $m = 5$, we encode 26 bits into 31 bits, and so on.

Now, we have to tell you how to decode the Hamming code. There is a very clever way to do this, most of which is a general technique for decoding any linear code.

What we do is find a matrix H with $n - k$ linearly independent (when using binary arithmetic) columns such that $GH = 0$. The matrix H will be called the *parity check* matrix for the code. The columns of H will be a basis for the perpendicular subspace to the codewords. That is, every column of H will be perpendicular to all rows of G, where by perpendicular, I mean that the inner product mod 2 is 0. From theorems of linear algebra, the dimension of a subspace and its perpendicular subspace add up to the dimension n of the space, so H will be $n \times n - k$. Not all the proofs of this theorem on the real numbers extend to finite fields, because over finite fields we have the strange phenomenon that a vector can be perpendicular to itself ... if v has an even number of 1's, then $v \cdot v$ will be even, and thus 0. However, the theorem still holds.

How can we find the subspace $G^{\perp}$? For an arbitrary G, we can do it using linear algebra. For G in the special form $(I|S)$, we can simply take H to be

$$\left(\frac{S}{I}\right)$$

Note that the identity matrix in H has size $n - k \times n - k$ (and S is still $k \times n - k$). Observe that, thanks to the identity matrix, H indeed has $n - k$ linearly independent columns. Multiplying, we obtain $GH = IS + SI = 2S = 0$. For the generator matrix

G above, we get

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

Now, to encode, we take a message m and encode it by mG. When we send it over a noisy channel, we may get an error e, which we will assume has at most one bit. The received message is thus $mG + e$. To decode, what we will do is first find the error, second, subtract it from the received, and third, decode. We know how to decode (when there are no errors), and subtraction is easy. How do we find the error?

To find the error, we take the received codeword $mG + e$, and multiply it by H. We get

$$(mG + e)H = mGH + eH = eH$$

since $GH = 0$. We thus have a bit string which depends only on the error. This is called the *syndrome* (since we use it to diagnose the error). The hard part of error correcting codes in general is going from the syndrome to the error; however, for Hamming codes, this is very easy. We know that e has only one bit, let's say it is in the ℓ'th position; then eH is the ℓ'th row of H (observe that allrows of H are different, and thus there cannot be any ambiguity). Thus, we need only look at the syndrome, and identify which row of H it equals. This gives the error, and we have decoded.

Let's do an example. Suppose we start with $m = 1001$. Then

$$c = mG = 1001001.$$

Suppose an error occurs in the third position, to get 1011001. Multiplying this by H, we obtain 011, which is the third row of H. We add the error 0010000 to the received message, 1011001, to get the codeword 1001001. We can decode this by just looking at the first four positions, 1001.