LECTURE 12

ERROR CORRECTING CODES

There are two subject matters in this lecture; the first is the ostensible topic, error correcting codes, and the other is how the process of discovery sometimes goes - you all know that I am the official discoverer of the Hamming error correcting codes. Thus I am presumably in a position to describe how they were found. But you should beware of any reports of this kind. It is true that at that time I was already very interested in the process of discovery, believing that in many cases the method of discovery is more important than what is discovered. I knew enough not to think about the process when doing research, just as athletes do not think about style when the engage in sports, but they practice the style until it is more or less automatic. I had thus established the habit that after something of great of small importance was discovered of going back and trying to trace the steps by which it apparently happened. But do not be deceived; at best I can give the conscious part, and a bit of the upper subconscious part, but we simply do not know how the unconscious works its magic.

I was using the Model 5 relay computer in NYC in preparation for delivering it to Aberdeen Proving Grounds, along with the some required software programs (mainly mathematical routines). When an error was detected by the 2-out-of-5 block codes the machine would when unattended repeat the step, up to three times, before dropping it and picking up the next problem in the hope that the defective equipment would not be involved in the new problem. Being at that time low man on the totem pole, as they say, I got free machine time only over the weekends - meaning from Friday at around 5:00 P.M. to Monday morning around 8:00 A.M. which is a lot of time! Thus I would load up the input tape with a large number of problems and promise my friends, back at Murray Hill N.J. where the Research Department was located, that I would deliver them the answers on Tuesday. Well, one weekend, just after we left on a Friday night, the machine failed completely and I got essentially nothing on Monday. I had to apologize to my friends and promised them the answers on the next Tues. Alas! The same thing happened again! I was angry to say the least, and said, "If the machine can locate that there is an error, why can it not locate <u>where</u> it is, and then fix it by simply changing the bit to the opposite state?" (The actual language used was perhaps a bit stronger!)

Notice first that this essential step happened only because there was a great deal of emotional stress on me at the moment, and this is characteristic of most great discoveries. Working calmly will let you elaborate and extend things, but the break throughs generally come only after great frustration and emotional involvement. The calm, cool, uninvolved researcher seldom makes really great, new steps.

Back to the story. I knew from previous discussions that of course you could build three copies of a machine, include comparing circuits, and use the majority vote - hence error correcting machines could exist. But at what cost! Surely there were better methods. I also knew, as discussed in the last lecture, a great deal about parity checks; I had examined their fundamentals very carefully.

Another aside. Pasteur says, "Luck favors the prepared mind." You see that I was prepared by the immediately previous work I had done. I had become more than acquainted with the 2-out-of-5 codes, I had understood them fundamentally, and had worked out and understood the general implications of a parity check.

After some thought I recognized that if I arranged the message bits of any message symbol in a rectangle, and put parity checks on each row and each column, then the two failing parity checks would give me the coordinates of the single error, and this would include the corner added parity bit (which could be set consistently if I used even parities), Figure 12-1. The redundancy, the ratio of what you use to the minimum amount needed, is

$$R = mn/(m-1)(n-1) \quad = 1 + 1/(m-1) + 1/(n-1) + 1/(m-1)(n-1)$$

It is obvious to anyone who ever took the calculus that the closer the rectangle is to a square the lower is the redundancy for the same amount of message. And of course big m's and n's would be better than small ones, but then the risk of a double error might be too great; again an engineering judgment. Note that if two errors occurred then you would have: (1) if they were not in the same column and not in the same row, then just two failing rows and two failing columns would occur and you could not know which diagonal pair caused them; and (2) if two were in the same row, (or column) then you would have only the the columns (or rows) but not the rows (columns).

We now move to some weeks later. To get to NYC I would go a bit early to the Murray Hill, N.J. location where I worked and get a ride on the company mail delivery car. Well, riding through north Jersey in the early morning is not a great sight, so I was, as I had the habit of doing, reviewing successes so I would have the style in hand automatically; in particular I was reviewing in my mind the rectangular codes. Suddenly, and I can give no reason for it, I realized that if I took a triangle and put the parity checks along the diagonal, with each parity check checking both the row and column it was in, then I would have a more favorable redundancy, Figure 12-2.

My smugness vanished immediately! Did I have the best code this time? A few miles of thought on the matter, (remember there were no distractions in the north Jersey scenery), I realized that cube of information bits, with parity checks across the entire planes and the parity check bit on the axes, for all three axes, would give me the three coordinates of the error at the cost of 3n - 2 parity checks for the whole n^3 encoded message. Better! But was it best? No! Being a mathematician I promptly realized that a 4-dimensional cube (I did not have to arrange them that way, only interwire them that way) would be better. So an even higher dimensional cube would be still better. It was soon obvious (say five miles) that a 2x2x2...x2 cube, with n+1 parity checks, would be the best - apparently!

But having burnt my fingers once, I was not about to settle for what looked good - I had made that mistake before! Could I prove it was best? How to make a proof? One obvious approach was to try a counting argument. I had n + 1 parity checks, whose result was a string of n + 1 bits, a binary number of length n + 1 bits, and this could represent any of 2^{n+1} things. But I needed only $2^n + 1$ things, the 2^n points in the cube plus the one result that the message was correct. I was off by almost a factor of 2. Alas! I arrived at the door of the company, had to sign in, and go to a conference so I had to let the idea rest.

When I got back to the idea after some days of distractions, (after all I was supposed to be contributing to the team effort of the company), I finally decided that a good approach would be to use the syndrome of the error as a binary number that named the place of the error, with, of course, all 0's being the correct answer, (an easier test than for all 1's on most computers). Notice that familiarity with the binary system, which was not common then, (1947-8), repeatedly played a prominent role in my thinking. It pays to know more that just what is needed at the moment!

How do you design this particular case of an error correcting code? Easy! Write out the positions in the binary code:

1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

etc.

It is now obvious that the parity check on the right hand side of the syndrome must involve all positions which have a 1 in the right hand column; that the second digit from the right must involve the numbers which have a 1 in the second column, etc. Therefore you have:

Parity check #1	1, 3, 5, 7, 9,11,13,15,...
Parity check #2	2, 3, 6, 7,10,11,14,15,...

Parity check #3	4, 5, 6, 7,12,13,14,15,...
Parity check #4	8, 9,10,11,12,13,14,15,...
etc.	

Thus if any error occurs in some position, those parity checks, and only those, will fail and give 1's in the syndrome, and this will produce exactly the binary representation of the position of the error. It is that simple!

To see the code in operation suppose we confine ourselves to 4 message and 3 check positions. These numbers satisfy the condition

$$2^3 \geq 7 + 1$$

which is clearly a necessary condition, and the equality is sufficient. We pick as the positions for the checking bits, (so that the setting of the parity check will be easy), the check positions 1, 2, and 4. The the message positions are therefore 3, 5, 6, 7. Let the message be

1001

We (1) write the message on the top line, (2) encode on the next line, (3) insert an error at position 6 on the next line, and (4) on the next three lines compute the three parity checks.

1	2	3	4	5	6	7	position
		1		0	0	1	message
0	0	1	1	0	0	1	encoded message
0	0	1	1	0	1	1	message with error

You apply the parity checks to the received message.

Check #1 --> 0
Check #2 --> 1
Check #3 --> 1

Binary number 110 --> 6; hence change the digit in position 6, and drop the check positions 1, 2 and 4, and you have the original message, 1001.

If it seems magical, then think of the all 0 message, which will have all 0 checks, and then think of a single digit changing and you will see that as the position of the error is moved around then the syndrome binary number will change correspondingly and will always exactly match the position of the error. Next, note that the sum of any two correct messages is still a correct message (the parity checks are additive modulo 2 hence the proper messages form an additive group modulo 2). A correct message will give all zeros, and hence the sum of a correct message plus an error in one position will give the position of the error regardless of the message being sent. The parity checks concentrate on the error and ignore the message.

Now it is immediately evident that any interchange of any two or more of the columns, once agreed upon at each end of the channel, will have no essential effect; the code will be <u>equivalent</u>. Similarly, the interchanging of 0 and 1 in any column (complementing that particular position) will not be an essentially different code. The particular (so called) Hamming code is merely a cute arrangement, and in practice you might want the check bits to all come at the end of the message rather than being scattered in the middle of it.

How about a double error? If we want to catch (but not be able to correct) a double error we simply add a single new parity check over the whole message we are sending. Let us see what will then happen at your end.

old syndrome	new parity check	meaning
000	0	right answer
000	1	new parity check wrong
xxx	1	old parity check works
xxx	0	must be a double error

A single error correcting plus double error detecting code is often a good balance. Of course, the redundancy in the short message of 4 bits, with now 4 bits of check, is bad, but the number of parity bits rises roughly like the log of the message length. Too long a message and you risk a double uncorrectable error, (which in a single error correcting code you will "correct" into a third error), too short a message and the cost in redundancy is too high. Again an engineering judgment depending on the particular situation.

From analytic geometry you learned the value of using the alternate algebraic and geometric views. A natural representation of a string of bits is to use an n-dimensional cube, each string being a vertex of the cube. Given this picture and finally noting that any error in the message moves the message along one edge, two errors along two edges, etc., I slowly realized that I was to operate in the space of L_1. The distance between symbols is the number of positions in which they differ. Thus we have a <u>metric</u> in the space and it satisfies the three standard conditions for a distance (see Lecture 10 where it is identified as the standard L_1 distance):

1. $D(x,y) \geq 0$ (non negative)

2. $D(x,y) = 0$ if and only if $x = y$ (Identity)

3. $D(x,y) = D(y,x)$ (symmetry)

4. $D(x,y) + D(y,z) \geq D(x,z)$ (triangle inequality)

Thus I had to take seriously what I had learned as an abstraction of the Pythagorean distance function.

With a distance we can define a sphere as all points (vertices, as that is all there is in the space of vertices), at a fixed distance from the center. For example, in the 3-dimensional cube which can be easily sketched, Figure 12-3, the points (0,0,1), (0,1,0), and (1,0,0) are all unit distance from (0,0,0), while the points (1,1,0), (1,0,1), and (0,1,1) are all two units away, and finally the point (1,1,1) is three units away from the origin.

We now go to n-dimensions, and draw a sphere of unit radius about each point and <u>suppose</u> that the spheres do not overlap. It is obvious that if the centers of these spheres are code points, and only these points, then at the receiving end any single error in a message will result in a non-code point and you can recognize where the error came from, it will be in the sphere about the point I sent to you, or equivalently in a sphere of radius 1 about the point you received. Hence we have an error correcting code. The minimum distance between code points is 3. If we use non-overlapping spheres of radius 2 then a double error can be corrected because the received point will be nearer to the original code point than any other point; double error correction, minimum distance of 5. The following table gives the equivalence of the minimum distance between code points and the correctability of errors:

min. distance	meaning
1	unique decoding
2	single error detecting
3	single error correcting
4	1 error correct and 2 error detect
5	double error correcting
2k + 1	k error correction
2k + 2	k error correction and k+1 error detection

Thus finding an error correcting code is the same as finding a set of code points in the n-dimensional space that has the required minimum distance between legal messages since the above conditions are both necessary and sufficient. It should also be clear that some error correction can be exchanged for more detection; give up one error correction and you get two more in error detection

I earlier showed how to design codes to meet the conditions in the cases where the minimum distance is 1, 2, 3, or 4. Codes for higher minimum distances are not so easily found, and we will not go farther in that direction. It is easy to give an upper bound on how large the higher distance codes can be. It is obvious that the number of points in a sphere of radius k is, (C(n,k) is a binomial coefficient)

$$1 + C(n,1) + C(n,2) + \ldots + C(n,k)$$

Hence if we divide the size of the volume of the whole space, 2^n, by the volume of a sphere then the quotient is an upper bound on the number of non-overlapping spheres, code points, in the corresponding space. To get an extra error detection we simply, as before, add an overall parity check, thus increasing the minimum distance, which before was $2k + 1$ to $2k + 2$ (since any two points at the minimum distance will have the overall parity check set differently thus increasing the minimum distance by 1).

Let us summarize where we are. We see that by proper code design we can build a system from unreliable parts and get a much more reliable machine, and we see just how much we must pay in equipment, though we have not examined the cost in speed of computing if we build a computer with that level of error correcting into it. But I have previously stressed the other gain, namely field maintenance, and I want to mention it again and again. The more elaborate the equipment is, and we are obviously going in that direction, the more field maintenance is vital, and error correcting codes not only mean that in the field the equipment will give (probably) the right answers, but that it can be maintained successfully by low level experts.

The use of error detecting and error correcting codes is rising steadily in our society. In sending messages from the space vehicles we sent to the outer planets, we often have a mere 20 watts or less of power, (possibly as low as 5 watts), and had to use codes that corrected 100's of errors in a single block of message - the correction being done here on earth, of course. When you are not prepared to overcome the noise, as in the above situation, or in cases of "deliberate jamming", then such codes are the only known answer to the situation.

In the late summer of 1961 I was driving across the country from my sabbatical at Stanford, Cal. to Bell Telephone Laboratories in N.J. and I made an appointment to stop at Morris, Illinois where the telephone company was installing the first electronic central office that was not an experimental one. I knew that it used Hamming codes extensively, and I was, of course, welcomed. They told me that they had never had a field installation go in so easily as this one did. I said to myself, "Of course, that is what I have been preaching for the past 10 years." When, during initial installation, any unit is set up and running properly, (and you sort of know that it is because of the self checking and correcting properties), and you then turned your back on it to get the next part going, if the one you were neglecting developed a flaw, it told you so! The ease of initial installation, as well as later maintenance, was being verified right before their eyes! I can not say too loudly, error correction not only gets the right answer when running, it can by proper design also contribute significantly to field installation and field maintenance; and the more elaborate the equipment the more essential these two things are.

I now want to turn to the other part of the talk. I have carefully told you a good deal of what I faced at each stage in discovering the error correcting codes, and what I did. I did it for two reasons. First, I wanted to be honest with you and show you how easy, if you will follow Pasteur's rule, "Luck favors the prepared mind.", to succeed by merely preparing yourself to succeed. Yes, there were elements of luck in the discovery; but there were many other people in much the same situation, and they did not do it! Why me? Luck, to be sure, but also I was preparing myself by trying to understand what was going on - more than the other people around who were merely reacting to things as they happened, and not thinking deeply as to what was behind the surface phenomena.

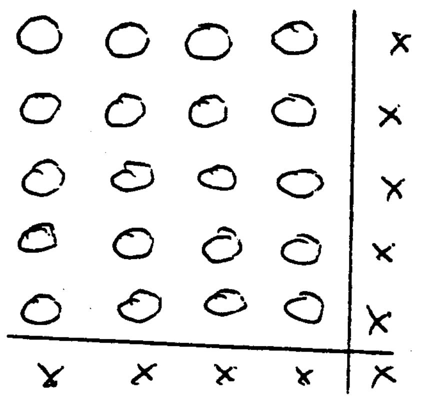
I now challenge you. What I told you in less than one hour was done in the course of a total of about three to six months, mainly working at odd moments while carrying on my main duties to the company. (Patent rights delayed the publication for more than a year.) Does anyone dare to say that they, in my position, could not have done it? Yes, you are just as capable as I was to have done it - <u>if</u> you had been there <u>and</u> you had prepared yourself as well!

Of course as you go through life you do not know what you are preparing yourself for - only that you want to do significant things and not spend the whole of your life being a "janitor of science" or whatever your profession is. Of course luck plays a prominent role. But so far as I can see, life presents you with many, many opportunities for doing great things (define them as you will) and the prepared person usually hits one or more successes, and the unprepared person will miss almost every time.

The above opinion is not based on this one experience, or merely on my own experiences, it is the result of studying the lives of many great scientists. I wanted to be a scientist hence I studied them, and I looked into discoveries that happened where I was and asked questions of those who did them. This opinion is also based on common sense. You establish in yourself the style of doing great things, and then when opportunity comes you almost automatically respond with greatness in your actions. You have trained yourself to think and act in the proper ways.

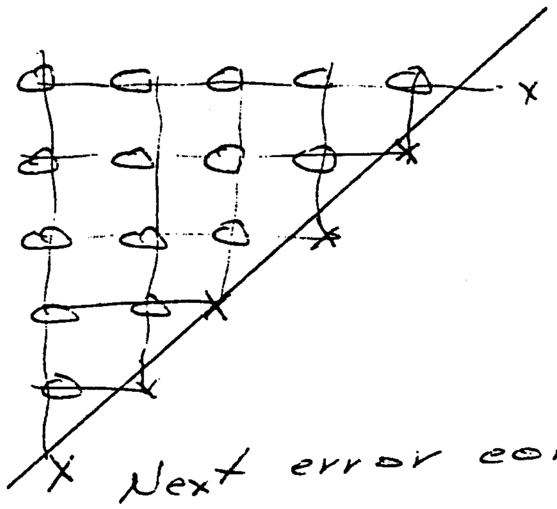
There is one nasty thing to be mentioned, however, What it takes to be great in one age is not what is required in the next one. Thus you, in preparing yourself for future greatness, (and the possibility of greatness is more common and easy to achieve than you think, since it is not common to recognize greatness when it happens under one's nose) you have to think of the nature of the future you will live in. The past is a partial guide, and about the only one you have besides history is the constant use of your own imagination. Again, a random walk of random decisions will not get you anywhere near as far as those taken with your own vision of what your future should be.

I have both told and shown you how to be great; now you have no excuse for not doing so!



First error correcting codes.

Figure 12-1



Next error correcting codes

Figure 12-2