Chapter One: Error-Correcting Codes

Section 1. From the Alt Code to the Hamming Code

In a war, those who start it are usually at a safe distance from the front. Orders are sent to the troops via binary messages, sequences of 0s and 1s which may be interpreted according to a code by both the transmitter and the receiver. For example, if we want the troops to “Do nothing”, we send the message 0. If we want the troops to ”Drop the bomb”, we send the message 1. This process is called encoding.

When the transmitted message is received, it must be interpreted. This process is called decoding. Here, 0 is decoded as ”Do nothing” and 1 as ”Drop the bomb”. However, electronic transmissions may contain errors, due to imperfect equipments or enemy sabotage. We will assume that an error consists of a single digit-reversal per transmission.

In other words, the worst that can happen is that one of the 0s may turn into a 1, or one of the 1s may turn into a 0, but neither can happen twice and both cannot happen together in a single transmission. Even this is serious enough. In our example, if a 1 turns into a 0, we may not wake up in time to find that we have lost World War II. On the other hand, if a 0 turns into a 1, we may have started World War III while still fighting World War II.

We would like to have some way of telling whether we can trust what we have received. So we modify our simple code as follows: 00 would mean ”Do nothing” and 11 would mean ”Drop the bomb”. If an error occurs, we will receive either 10 or 01, and we will know something has happened to the message. This is a prototype of what is known as an error-detecting code.

When we learn that an error has occurred, the natural thing is to ask headquarters to retransmit the message. If errors occur only infrequently, this is tolerable. If they occur often enough, it is at least a nuisance. Moreover, the request for retransmission is also sent electronically, and errors can occur there too.

What we would like is a code which not only tells us when something has gone wrong, but tells us exactly what has gone wrong. This is asking for a lot, but as is often the case, when we believe that something can be done, there may just be a way to do so. We now modify our code again, with 000 meaning ”Do nothing” and 111 meaning ”Drop the bomb”. If we receive 001, 010 or 100, we do nothing. If we receive 110, 101 or 011, we drop the bomb. This is a prototype of what is known as an error-correcting code.
Note that it is unrealistic to increase the length of the message and continue to assume that there is at most one error per transmission. So let us fix the length of any message to 15 binary digits, with at most one digit-reversal per transmission.

In some sense, even this is unrealistic since if one digit-reversal can occur, there is no reason why a second one cannot. What must be emphasized is that our assumption amounts to a mathematical model which simulates reality, but is not reality itself. We can get away with it if the probability of a single digit-reversal is high enough to worry us, but the probability of multiple digit-reversals is low enough to lose any sleep over.

Of the 15 digits, we may view some of them as conveying the intended message while the remaining ones are for security measure. The efficiency of a code using this 15-digit transmitter is defined as \( \frac{n}{15} \), where \( n \) is the number of digits used for the message.

How can we extend our earlier examples with short messages to 15-digit messages? In an error-detecting code, we need to distinguish between two scenarios, whether the message contains an error or not. A single binary digit would allow us to do so. Hence the efficiency of such a code can be as high as \( \frac{14}{15} \). Obviously, we cannot have \( \frac{15}{15} \) as we will have no protection at all.

How can we encode the intended message 10110100010111? Should we add a 0 or a 1 as the fifteen digit? Let us reexamine the simple example earlier. To the message 0, we add a 0, and to the message 1, we add a 1. Note that we are not copying the message, but arrange for the coded message to contain an even number of 1s. Since 10110100010111 has 8 1s, we add a 0 to yield the coded message 101101000101110. This code is called the parity-check code, and the added digit is called the parity-check digit.

Decoding is straight-forward. Simply count the number of 1s in the received message. If no digit-reversal has occurred, this number is even as agreed. If a single digit-reversal has occurred, whether a 0 turned into a 1 or vice versa, the number of 1s will become odd. Thus a received message 001100101010110 contains an error. We do not know the intended message as any of the 15 digits may be the one which has been reversed.

In an error-correcting code on the 15-digit transmitter, we need to distinguish between sixteen scenarios, whether the message contains an error, and if so, which of the 15 digits has been reversed. We need four binary digits to do so because \( 2^4 = 16 \). Thus the efficiency of such a code cannot be higher than \( \frac{11}{15} \).