Understanding Transmissions with Error-Correcting Codes

Data transmission is often noisy. Information can get easily garbled, and imperfect information frequently has a cost associated with it. Coding theory is a field of mathematics that deals with trying to make transmission more reliable by using error correcting codes, which are methods of detecting and correcting errors.

Throughout this delve into error correcting codes, we will be considering the transmission of strings of binary (or base 2), as every letter, symbol or pixel from an image can be represented as a string of 0s and 1s. Additionally, when a message of length *k* is sent, there are 2^k possible messages being sent, as there are 2 options for each bit. Error correcting codes send more than *k* binary digits, with these extra digits, called parity digits, helping to detect and correct codes.

One example of error correction codes is repetition codes, where we send each message multiple times. For example, if we sent 0011 twice, as 00110011, then the second block of four bits could be compared by the receiver against the first block. A recurrent term in coding theory is *information rate* (R), with it recording how much information is carried on average for every bit that is sent. For repetition codes, the information rate is often really low, here 0.5 and this can be lower if blocks are sent even more times. Therefore, in practice, we would prefer to use other error correction codes.

The *weight* of a binary sequence is the number of bits in the message that are equal to 1. Parity check codes work by adding a parity check bit at the end that will make a message have even (or odd) weight. For example, consider a scenario where we want an even weight, and the message we are trying to send is 0011. There are 2 (which is an even count) 1s right now, so we add a 0 at the end, so that we have even weight. If we were trying to send 1110, then we would add a 1. This is the equivalent of making the parity digit equal to the sum of all the digits modulo 2. Then, when a message received, if this parity bit does not represent the weight of the message being sent, then a bit might have flipped in transmission. Although we can't be sure where an error may have occurred, or if an even number of changes occurred, the information rate for this code is 4/5 = 0.8.

If we consider trying to find a single-error-detecting code for a message of length 4 bits, as we need to always add a parity bit, an overall message of length 5 is the shortest possible length, so 0.8 is the highest possible information rate for a message of length 4. In general, parity check codes for messages of length *n* have information rates of $R = \frac{n}{1+n}$.

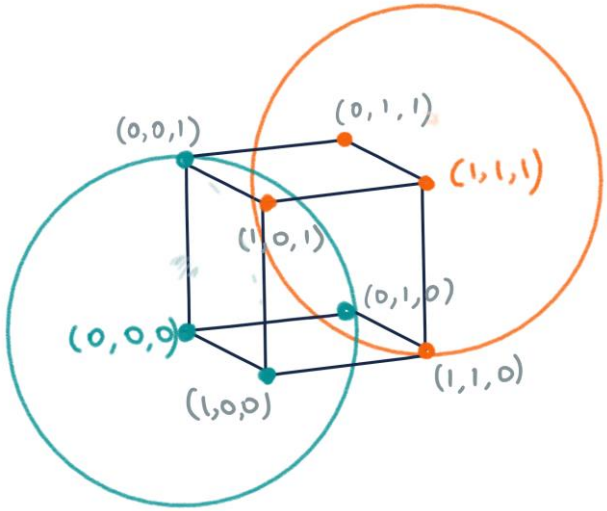
So far, what has been described has been focused on trying to get good error detection, but we still need to correct these errors and try and find the original message. I will be referring to *codewords*, which are the overall transmissions received that combine our original message and the added bits, that help transmit messages with less likelihood of error. Another important definition is that of *Hamming distances,* which are the total number of different bits when comparing two codewords. For example, 1010 and 1001 have a Hamming distance of 2, as the last two bits are different.

Let's take a scenario where we are trying to send either a True or a False, with 1 for True and 0 for False. If an error occurred when sending this message, then we would never be able to know if the original message was True or False. Now, let's say that 11 is True and 00 is False. We still would not be able to know what the original message was for 01, as this has a Hamming distance of 1 from both 00 and 11.

Therefore, let's say that 111 is True and 000 is False. If we received either 101,110, 011, then we could assume that an error had occurred and what was originally sent was True, as there is only one change. Therefore, if we received 001, 010, or 100, then the original message was False. This is known as

majority logic. This code is called the (3,1) repetition code, as three bits are sent, with the message of length 1 (0 or 1) being identified.

There is a link with geometry here – this is where sphere packing comes in. Sphere packing concerns the arranging of non-overlapping spheres within a space, where we are trying to maximise the total volume of all the spheres that fit within this space. If we consider (0,0,0) and (1,1,1) as points in a 3D space, then all our other possible messages, where only one error has occurred, can also be visualised as vertices of a cube. Two spheres of radius 1 can then be centred at (0,0,0) and (1,1,1). The vertices contained within the sphere can then be interpreted as codewords for the point at the centre of the sphere.



In order to fit more original messages being sent, we would like more of these spheres to be packed around points, so that more messages can be transmitted. Sphere packing, where all spheres are disjoint, can be used to find error correcting codes that can always detect the position of errors. Perfect Hamming codes are a well-known class of codes that can correct single bit errors, and these occur when all vertices, in however many dimensions of Euclidean space, can be contained within spheres that have the smallest radius we can make possible. This means that they have attained the Hamming bound, or the sphere-packing bound. Another way of writing this is to say that a perfect code occurs when all vertices are either codewords themselves, or are only one edge (or a Hamming distance of 1) away from a codeword.

The most well-known Hamming code is the (7,4) code which uses a 'generator matrix' to create three parity bits added to our four bits that make up the message, and is a code that can detect and correct single errors. It is thought to have a relatively high information rate, as the rate is higher than 0.333 for the (3,1) code; the R for the (7,4) code is 4/7 = 0.571 (3 significant figures).

What is interesting to note is that (7,4) is the first perfect Hamming code after (3,1). After this, the next perfect code is (15,11). A pattern you may have spotted is that the first number in all of these brackets is one less than a power of two. In order to explain this, let us consider the (7,4) scenario. If we consider code words that are on a 7-dimensional hypercube, every codeword would have 7 edges exiting this point, and so, including itself, there are 8 vertices involved for every message. Now, as we are using messages of length 4, there are 16 possible messages. 16 x 8 is 128 which is 2^7. On a hypercube in n-dimensions, the total number of vertices is always 2^n. This makes having an overall

message length of 2^{n-1} a perfect scenario, as every single vertex is involved with a codeword (2^n for each possible message), so there is a most effective use of spheres or space.

Perfect Hamming codes are a method of efficiently correcting single bit errors, but it is important to note that these processes are not going to be able to correct all of the bits that may be corrupted in a data transmission error. There are also many other famous codes that I have not delved into, including Reed-Muller codes, the famous Golay (23,12) code that can correct up to three errors, and the Leech lattice in 24-dimensional space. There are also many links between error correcting and probability that have not been mentioned.

With an increasing focus on quantum computing and how powerful this field can be, especially when we think about their impact on cryptography, it is interesting to think about qubits, which also transmit information. Qubits can be in any superposition of the states 0 and 1, and will also have errors, but recent research has shown that space-time could be involved in building error correcting code for qubits and quantum computers. Perhaps this will be the area that coding theory focuses on next.

Radhika Iyer

References

https://mathworld.wolfram.com/CodingTheory.html

https://brilliant.org/wiki/error-correcting-codes/

Thompson, Thomas M. (2014) - From Error-Correcting Codes Through Sphere Packings to Simple (Chapter 1) doi: 10.5948/UPO9781614440215.002

Butler, Michael K – An Introduction to Error Correcting Codes: https://www.academia.edu/40425856/An_Introduction_to_Error_Correcting_Codes

Spheres and Code Words – Numberphile: https://www.youtube.com/watch?v=T46FTuHnbvY

The Perfect Code – Computerphile: https://www.youtube.com/watch?v=WPoQfKQlOjg

Quanta Magazine: How Space and Time Could Be a Quantum Error Correcting Code - https://www.quantamagazine.org/how-space-and-time-could-be-a-quantum-error-correcting-code-20190103/