ERROR CORRECTION

October 1, 2020

Outline

1. Error Correction vs Error Detection
2. Message Geometry
3. Hamming Distance
4. Redundancy Bits
5. Hamming Code (Not Required Material)

Correction vs Detection

Error Correction

- Error correction, like error detection, uses the idea of redundancy.
  - However, this time more redundancy is needed! Why?
- Because error correction is more difficult than error detection!
  - Not only you must detect that an error occurred!
  - You have to correct it, as well!
- To correct a bit-error it is enough to locate it! Why?
- Because
  - If you can locate the error, then "flip" the bit at that location!
  - So, this is ok because we use binary data representation.

Message Space

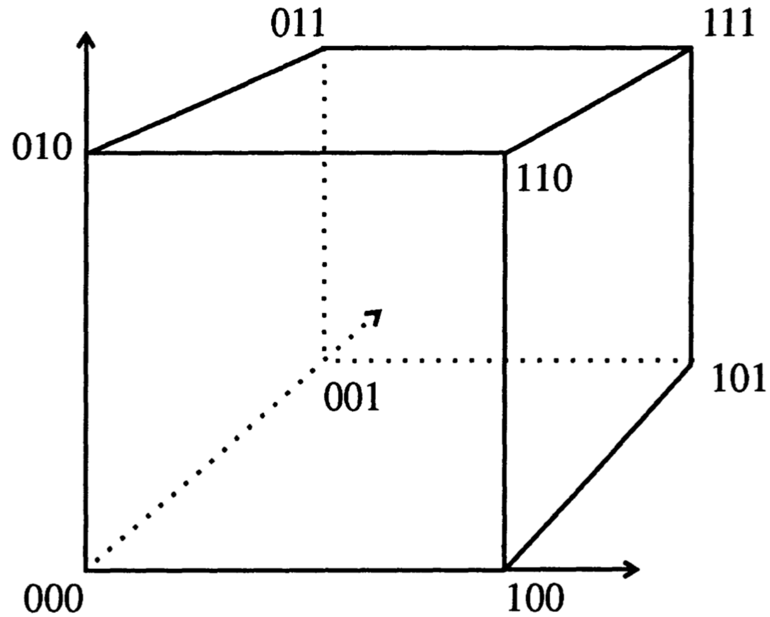
Messages

- Message space is made up of the messages that we want to transmit (also known as words).
- We are used to thinking of a space as something which can be many-dimensional, either continuous or discrete, and whose points can be labeled by coordinates.
- Message space is a multidimensional discrete space, some or all of whose points correspond to messages.
- To make matters a little more concrete, consider a three-bit binary code, with acceptable words:

$$000, 001, 010, 011, 100, 101, 110, 111$$

Space

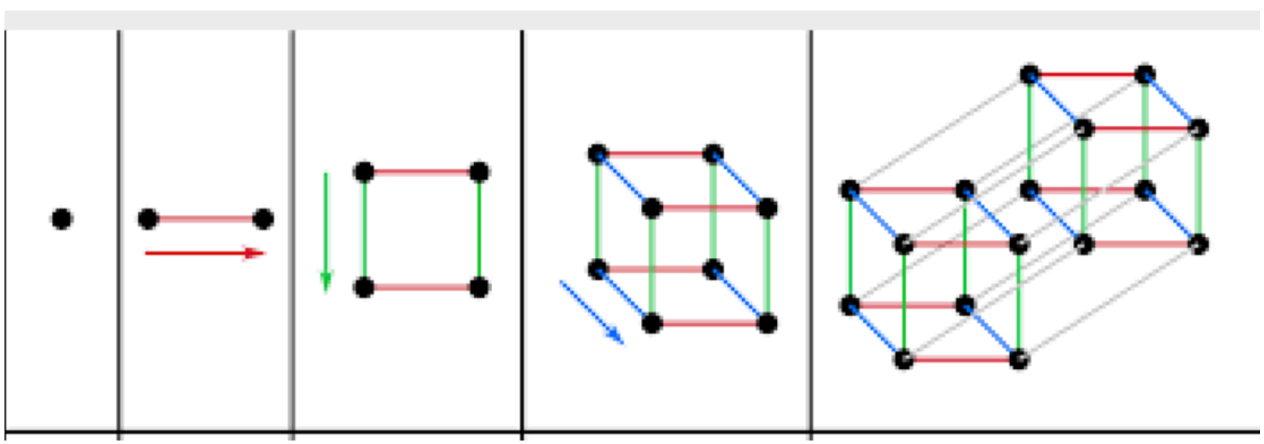
- These are just the binary numbers zero to seven.



- We can consider these numbers to be the coordinates of the vertices of a cube in three-dimensional space

Space

- This cube is the message space corresponding to the three-bit messages.
- The only points in this space are the vertices of the cube - the space between them in the diagram, the edges, and whatnot are not part of it.
- There is also the n-cube corresponding to n-bit messages: it is also called the hypercube of dimension n.



Errors in Space

- What happens if there is an error in transmission?
- This will change the bits in the sent message, and correspond to moving us to some other point in the message space.
- Intuitively, it makes sense to think that the more errors there are, the "further" we move in message space;
- in the diagram above, (111) is "further" from (000) than is (001) or (100).
- This of course leads to the Hamming distance, defined later

Hamming Distance

How Do We Handle Errors?

- Normally, a code (also called frame) consists of m data (i.e., message) bits and r redundant, or check, bits.
- Let the total length be n (i.e., $n = m + r$): message plus redundancy bits.
- An n-bit unit containing data and check bits is often referred to as an n-bit codeword (also known as code).
- Given any two codewords, say, 10001001 and 10110001, it is possible to determine how many corresponding bits differ.
  - In this case, 3 bits differ.
- To determine how many bits differ, just XOR the two codewords and count the number of 1 bits in the result.

$$(10001001) \oplus (10110001) = 00111000$$

Number of Different Bits

- The number of bit positions in which two codewords differ is called the Hamming distance (Hamming, 1950).
  - Its significance is that if two codewords are a Hamming distance d apart, it will require d single-bit errors to convert one into the other.
- The Hamming distance of 10001001 and 10110001 is 3.
- **NB.** In most data transmission applications, all 2^m possible data messages are legal, but due to the way the check bits are computed, not all of the 2^n $(= 2^{m+r})$ possible codewords are used.

Distance and Space

- The notion of distance is useful for discussing errors.
- Clearly, a single error moves us from one point in message space to another a Hamming distance of one away; a double error puts us a Hamming distance of two away, and so on.
- For a given number of errors e we can draw about each point in our hypercubic message space a "sphere of error", of radius e, which is such that it encloses all of the other points in message space which could be reached from that point as a result of up to e errors occurring.
- This gives us a nice geometrical way of thinking about the coding process.

Distance and Space

- Whenever. we code a message M, we rewrite it into a longer message M_e.
- We can build a message space for M_e just as we can for M; of course, the space for M_e will be bigger, having more dimensions and points.
- Clearly, not every point within this space can be associated one-on-one with points in the M-space; there is some redundancy.
- This redundancy is actually central to coding. e-Error correction involves designing a set of acceptable coded messages in M_e such that if, during the transmission process, any of them develops at most e errors, we can locate the original message with certainty.

Distance and Space

- In our geometrical picture, acceptable messages correspond to certain points within the message space of M_c
- Errors make us move to other points, and to have error correction we must ensure that if we find ourselves at a point which does not correspond to an acceptable message, we must be able to backtrack, uniquely, to one that does.
- A straightforward way to ensure this is to make sure that, in M_c all acceptable coded message points lie at least a Hamming distance of:

$$d = 2e + 1$$

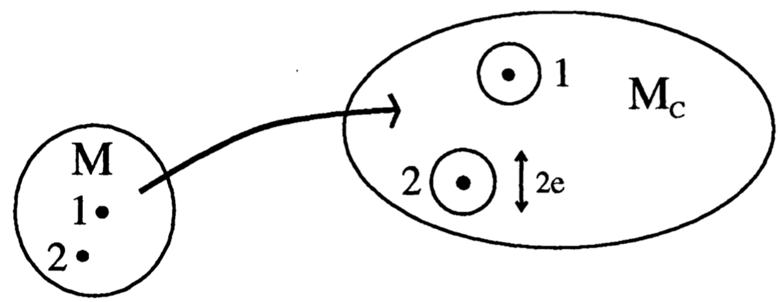
  from each other
- Why does this work?

Distance and Space

- Suppose we send an acceptable message M, and allow e errors to occur in transmission.
- The received message M' will lie at a point in M_e which is e units away from the original.
- How do we get back?
- Because of the separation of $d = 2e + 1$ we have demanded, M is the closest acceptable message to M': all other acceptable messages must lie at a Hamming distance $\geq e + 1$ from M'.
- Note that we can have simple error detection more cheaply; in this case, we can allow acceptable points in M_e to be within $2e$ of one another.
- The demand that points be $(2e + 1)$ apart enables us to either correct e errors or detect $2e$.

Distance and Space

- Each element of M is associated with a point in M_e such that no other acceptable points lie within a Hamming distance of $2e+1$ units.



- We can envisage the space for M_e as built out of packed spheres, each of radius e units, centered on acceptable coded message points
- If we find our received message to lie anywhere within one of these spheres, we know exactly which point corresponds to the original message.

Example: Error Detection

- Consider a code in which a single parity bit is appended to the data.
- The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd).
  - For example,
    - in even parity, when 1011010 is sent a bit is added to the end to make it 10110100
    - in odd parity, when 1011000 is sent a bit is added to the end to make it 10110001.
- A code with a single parity bit has a distance 2, since any single-bit error produces a codeword with the wrong parity. It can be used to detect single errors.

Redundancy Bits

Error Correction: # of Redundancy Bits

- If we have m bits of data and r bits of redundancy (this is called the *syndrome*), $m + r$ bits are transmitted.
- If we decide that a vanishing syndrome is to represent no error, that leaves at most $(2^r - 1)$ message error positions that can be coded. However, errors can occur in the syndrome as well as the original message we are sending.
- We need to determine the location of any of these $m + r$ bits:
  - these r bits must be able to indicate any of the $m + r$ positions!

m bit locations	r bit locations

- Hence, r must be chosen so that we have

$$2^r - 1 \geq m + r$$

Requirements of Error Correction: Condition $2^r > m + r$

# Data Bits m	# Redundancy Bits r	# Total Bits $m + r$
1	2	3
2	3	5
3	3	6
4	3	7
5	4	9
6	4	10
7	4	11
16	5	21
32	6	38

Example

- If we wanted to send a message 11 bits in length, we would have to include a syndrome of at least four bits, making the full message fifteen bits long.
- This does not seem particularly efficient
  - efficiency $= 11/15$ or about 70%.
- However, if the original message was, say, 1000 bits long, we would only need ten bits in our syndrome ($2^{10} = 1024$) which is a considerable improvement!

Hamming Codes

(Not Required)

Idea of Hamming Code

- Use extra parity bits to allow the identification of a single error.
- Create the code word as follows:
  1. Mark all bit positions that are powers of two as parity bits.
     - positions: $1, 2, 4, 8, 16, 32, 64$, etc.
  2. All other bit positions are for the data (message) to be encoded.
     - positions: $3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17$, etc.
  3. Each parity bit calculates the parity for some of the bits in the code word.
  4. The position of the parity bit determines the sequence of bits that it alternately checks and skips.

Error Correction: Example of ASCII Characters

- ASCII characters consist of 7 bits.
- To correct an error on an ASCII character the algorithm must determine which of the seven bits has changed.
- An ASCII character has 7 bits.
  - Three redundancy bits are enough to identify any one of seven positions.
- What if an error occurs in a redundancy bit?

Hamming Code

- The **Hamming code** provides a practical solution to the error correction problem.
- For simplicity, in the sequel we discuss the case of ASCII character bit strings (these are bit strings of length 7).
  - By the previous discussion we must choose r so that $2^r > 7 + r$.
  - The minimum possible such value of r is 4 and so we must use four redundancy bits.
- It can be designed to work to data units of any given length.

Hamming Code for ASCII Characters

- Where do you locate the four redundancy bits?
- To form the eleven bits of the Hamming code redundancy bits are placed in positions $1 = 2^0, 2 = 2^1, 4 = 2^2, 8 = 2^3$:

$$
\begin{array}{lccccccccccc}
\text{input} & & & d & & d & d & d & & d & d & d \\
\text{insertions} & 2^0 & 2^1 & & 2^2 & & & & 2^3 & & & \\
\text{position} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
 & \downarrow & \downarrow & & \downarrow & & & & \downarrow & & & \\
\text{type of bit} & r_1 & r_2 & d & r_4 & d & d & d & r_8 & d & d & d
\end{array}
$$

- r with subscripts indicate the redundancy bit, and d the data bit.

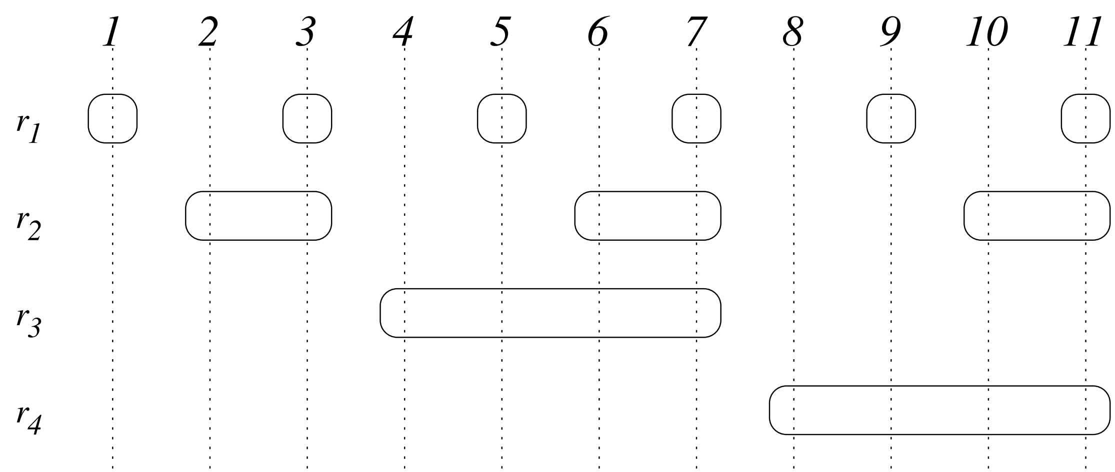
Hamming Code for ASCII Characters: Example

- For the sequence 1010110 of seven bits we have the following Hamming code

$$
\begin{array}{ccccccccccc}
 & & 1 & & 0 & 1 & 0 & & 1 & 1 & 0 \\
\boxed{r_1} & \boxed{r_2} & 1 & \boxed{r_4} & 0 & 1 & 0 & \boxed{r_8} & 1 & 1 & 0
\end{array}
$$

Hamming Code for ASCII Characters

- The redundancy bits are essentially parity bits computed in a special way.
- What are the values of the redundancy bits?
- Take as parity bit the XOR of the bits in positions indicated below!



Hamming Code for ASCII Characters

redundancy bit	positions used for parity
r_1	parity check bits of $\boxed{1}, 3, 5, 7, 9, 11$
r_2	parity check bits of $\boxed{2}, 3, 6, 7, 10, 11$
r_4	parity check bits of $\boxed{4}, 5, 6, 7$
r_8	parity check bits of $\boxed{8}, 9, 10, 11$

Hamming Code for ASCII Characters: Example

- For example for the sequence $\boxed{r_1}\boxed{r_2}1\boxed{r_4}010\boxed{r_8}110$ we obtain the following equations

$$
\begin{aligned}
0 &= r_1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 \oplus 0 \\
0 &= r_2 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 0 \\
0 &= r_4 \oplus 0 \oplus 1 \oplus 0 \\
0 &= r_8 \oplus 1 \oplus 1 \oplus 0
\end{aligned}
$$

- Solving these equations we obtain

$$r_1 = 0, r_2 = 1, r_4 = 1, r_8 = 0$$

Example of Hamming Code

- Consider the sequence of bits 1011001

1	2	3	4	5	6	7	8	9	10	11
$\downarrow$	$\downarrow$		$\downarrow$				$\downarrow$			
r	r	1	r	0	1	1	r	0	0	1
$\boxed{0}$	r	1	r	0	1	1	r	0	0	1
$\boxed{0}$	$\boxed{1}$	1	r	0	1	1	r	0	0	1
$\boxed{0}$	$\boxed{1}$	1	$\boxed{1}$	0	1	1	r	0	0	1
$\boxed{0}$	$\boxed{1}$	1	$\boxed{1}$	0	1	1	$\boxed{0}$	0	0	1

- So the sender transmits the sequence 01110110001
- By independence of vectors of positions it is possible to locate error! How is this done?

Locating Errors in a Hamming Code

- In previous example, suppose 7th bit of message is in error, i.e., receiver receives 011101 $\boxed{0}$ 0001
- Receiver does not know there is an error, but recalculates the values r_1, r_2, r_4, r_8.

$$
\begin{array}{ccccccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\downarrow & \downarrow & & \downarrow & & & & \downarrow & & & \\
0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

$$\rightarrow r_1 = 0$$
$$\rightarrow r_2 = 1$$
$$\rightarrow r_4 = 1$$
$$\rightarrow r_8 = 1$$

Locating Errors in a Hamming Code

- Look at locations $1, 2, 4, 8$ and compare value received with value calculated: if equal: bit-value is 0, and if different: bit-value is 1.

Bit Positions :	1	2	4	8
Value Received :	1	0	0	1
Value Calculated :	0	1	1	1
Difference :	1	1	1	0

- The last sequence of bits gives the location of the error

$$1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + 0 \cdot 2^3 = 7$$

- The 7th bit is in error!!! Fix this bit and you are done!!!
- **Correcting Bursts:** To detect bursts of a certain length an even higher redundancy is required! We won't cover this here!

Exercises[a]

1. Generalize the message space to m-bit messages, which would have a message space that was a 2^m-vertex "hypercube", which unfortunately our brains can't visualize!
2. Show that in the m-cube every vertex has degree 2^m.
3. Show that the m-cube has diameter 2^m.

[a] Do not submit