Extended Introduction to Computer Science
CS1001.py

Module I Error Detection and Correction Codes

Instructors: Elhanan Borenstein, Michal Kleinbort

Teaching Assistants: Noam Parzanchevsky,

Asaf Cassel, Shaked Dovrat, Omri Porat

School of Computer Science
Tel-Aviv University
Spring Semester 2021
http://tau-cs1001-py.wikidot.com

* Slides based on a course designed by Prof. Benny Chor

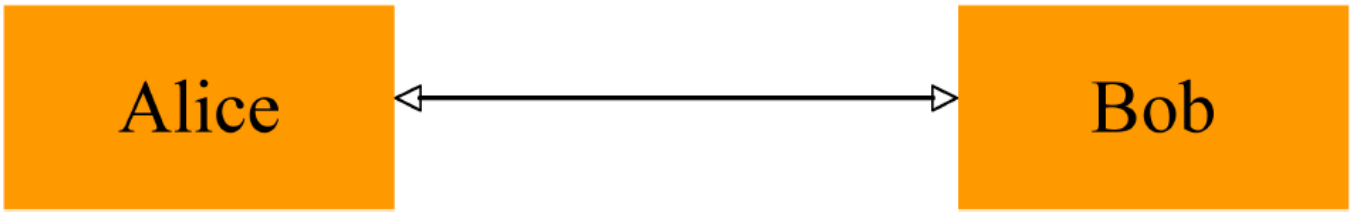
Plan for the next lectures

Error detection and error correction codes

- Basic notions of codes
  - The binary symmetric channel
  - Hamming distance
  - The geometry of codes - spheres around codewords
- Additional simple codes
  - Repetition code
  - Parity bit code
- Hamming (7, 4, 3) code

Communication

Two parties, traditionally names Alice and Bob, have access to a communication line between them, and wish to exchange information.



This is a very general scenario. It could be two kids in class sending notes, written on pieces of paper or (god forbid) text messages under the teacher's nose. Could be you and your friend talking using "traditional" phones, cell phones, or Skype. Could be an unmanned NASA satellite orbiting Mars and communicating with Houston using radio frequencies. It could be the hard drive in your laptop communicating with the CPU over a bus, or your laptop running code in the "cloud" via the "net".
In each scenario, the parties employ communication channels with different characteristics and requirements.

Three Basic Challenges in Communication

1. Reliable communication over unreliable (noisy) channels.

2. Secure (confidential) communication over insecure channels.

3. Frugal (economical) communication over expensive channels.

Three Basic Challenges in Communication

1. Reliable communication over unreliable (noisy) channels.

Solved using error detection and correction codes.

2. Secure (confidential) communication over insecure channels.

Solved using cryptography (encryption/ decryption).

3. Frugal (economical) communication over expensive channels.

Solved using compression (and decompression).

We treat each requirement separately (in separate classes). Of course, in a real scenario, solutions should be combined carefully so the three challenges are efficiently addressed (*e.g.* usually compression should be applied before encryption).

Today, we will discuss error detection and correction codes.

Reliable Communication over Unreliable Channels

"The first step to fixing an error is recognizing it."
(Seneca the Younger: A Roman dramatist, Stoic philosopher, and politician, 3 BC - 65 AD).

In this and the next class, we will look at some issues related to the question of how can information be sent reliably over noisy, unreliable communication channels.

It should be realized that these questions are not limited to phone lines or satellite communication, but rather arise in daily contexts such as ID or credit card numbers, ISBN codes (in books), audio encoded on compact disks, barcodes and QR codes, etc., etc. This is a rich and vivid topic, and here we will merely introduce it and scratch its surface.

The role of ID Check Digits

- Add a digit to the number, computed from the other digits.
- The redundancy makes it possible to identify an incorrect id (when given the full id, including the check digit).

The Check Digit of an Israeli ID Number, in Python

```
def control_digit(ID):
    """ compute the check digit in an Israeli ID number,
        given as a string """

    total = 0
    for i in range(8):
        val = int(ID[i]) # converts a char to its numeric integer value
        if i % 2 == 0:
            total = total+val
        else:
            if val < 5:
                total += 2*val
            else:
                total += (2*val % 10) + 1 # sum of digits in 2*val
    total = total % 10
    check_digit= (10 - total) % 10  # the complement mod 10 of sum

    return str(check_digit)
```

The Check Digit of an Israeli ID Number, in Python

```
def is_valid_ID():
    # prompts for input from user
    ID = input("pls type all 9 digits (left to right) of your ID: "

    assert len(ID)==9

    if control_digit(ID[:-1]) == ID[-1]:
        print("Valid ID number")
    else:
        print("ID number is incorrect")
```

Reminder: The input command prompts for a "string" input.

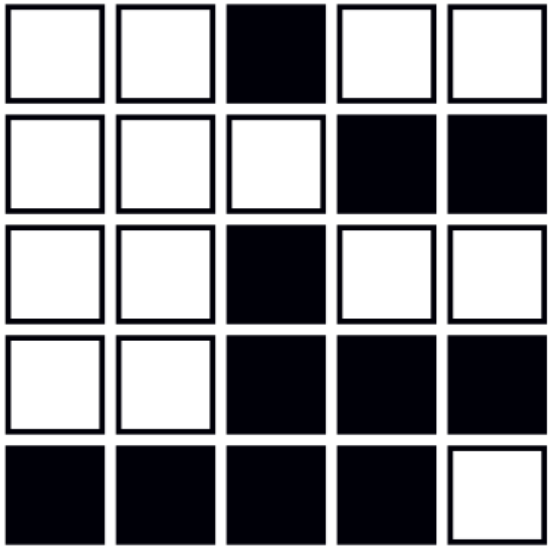
Detection and Correction of Errors

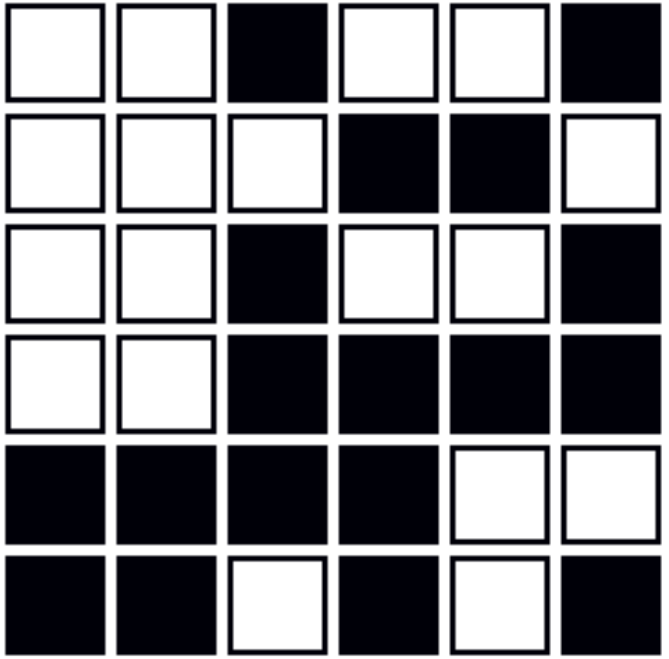
- The ID number code is capable of detecting any single digit error.
- It is also capable of detecting all but one transpositions of adjacent digits (there is one exception - find it!).
- It cannot correct any single error or adjacent transposition.
- It cannot detect many combinations of two-digit errors, or transposition of non-adjacent digits.

Next, we will explore a card magic trick, capable not only of error detection, but also of error correction.

The Card Magic Trick

The Card Magic Trick





(Source: Computer Science Unplugged.)

The 2D Card trick - explanation

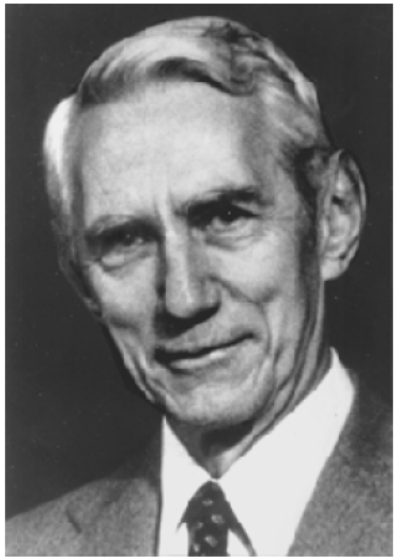
- After the 5 by 5 cards are placed, we add the 6th column in a way that ensures that each row has an even number of colored cards.
- The 6th row is added so that each column has an even number of colored cards.
- When a single card is flipped, there is exactly one row with an odd number of colored cards, and exactly one column with an odd number of colored cards.
- So the flipped card is in the intersection of these row and column.

Detection and Correction of Errors

- The 2D cards magic code can correct any single bit error.
- The 2D cards code can detect any combinations of two or three bits errors.
- It cannot detect some combinations of four-bit errors.
- Can it detect transposition errors?

Claude Shannon, the Father of Information Theory

Claude Elwood Shannon (April 30, 1916–February 24, 2001) was an American mathematician, electronic engineer, and cryptographer known as "the father of information theory".



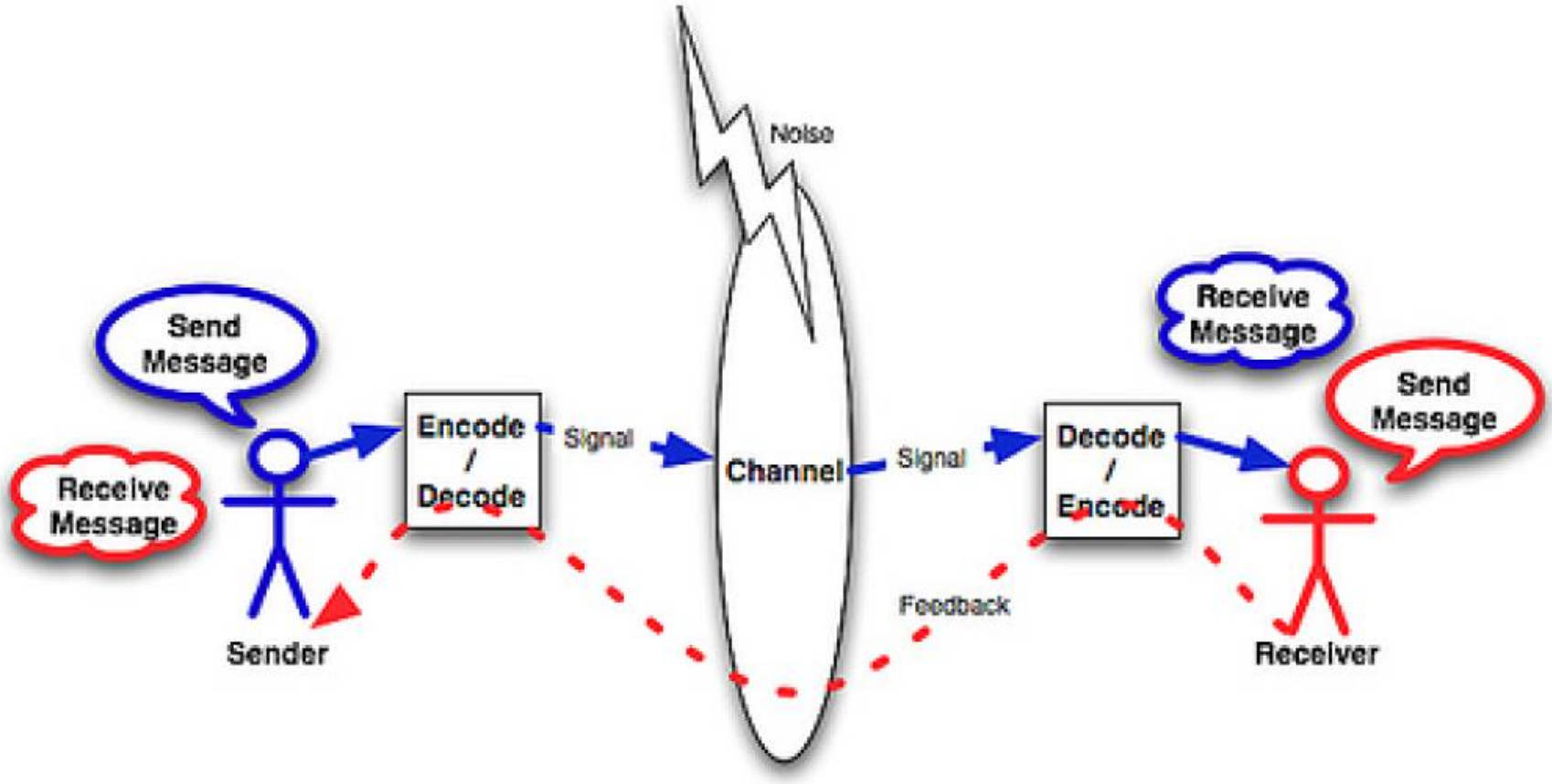
Shannon is famous for having founded information theory with one landmark paper published in 1948. But he is also credited with founding both digital computer and digital circuit design theory in 1937, when, as a 21 year old master's student at MIT, he wrote a thesis demonstrating that electrical application of Boolean algebra could construct and resolve any logical, numerical relationship.

It has been claimed that this was the most important master's thesis of all time. Shannon contributed to the field of cryptanalysis during World War II and afterwards, including basic work on code breaking.

For two months early in 1943, Shannon came into contact with the leading British cryptanalyst and mathematician Alan Turing. Turing had been posted to Washington to work with the US Navy's cryptanalytic service.

(text from Wikipedia)

The Shannon-Weaver Model of Communication (1949)

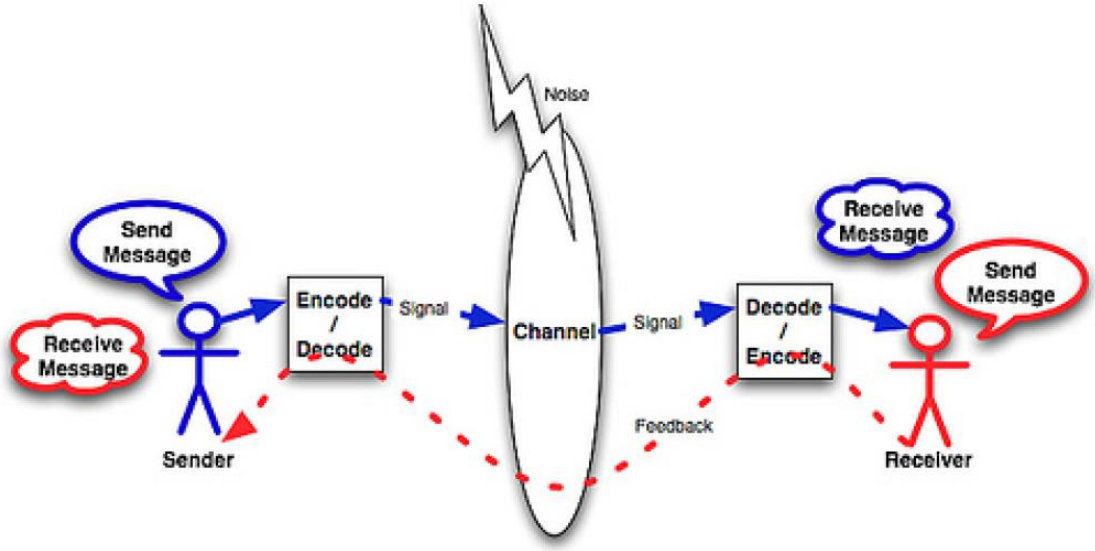


Source of figure is somewhat unexpected.

The Shannon-Weaver Model of Communication (1949)

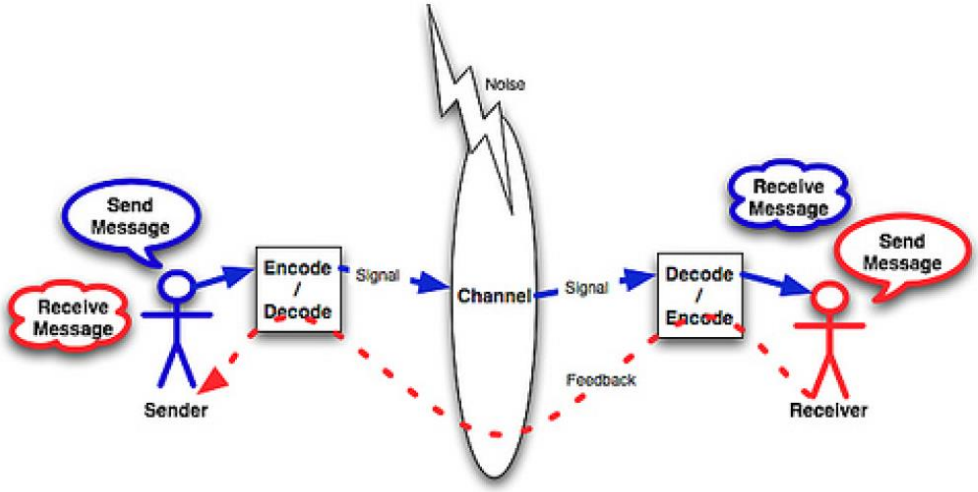
"We may have knowledge of the past but cannot control it; We may control the future but cannot know it.".

Claude Shannon, 1959



For simplicity, let every original message be a fixed length block of bits. The channel is noisy, so a subset of sent bits may get altered (reversed) along the way, with non-zero probability.

The Shannon-Weaver Model of Communication, cont.



Sender passes original message through an encoder, which typically produces a longer signal by concatenating so called parity check bits (which may, of course, get altered themselves).

The (possibly altered) signal reaches the recipient's decoder, which translates it to a message, whose length equals the length of the original message.

Goal: Prob(original message equals decoded message) ≈ 1 while adding as few bits as possible

Detecting vs. Correcting Errors

The receiver gets the signal (with zero or more bits flipped) and applies the decoding function.

Error detecting code:

- The receiver identifies errors in the transmission and asks for resending it. The receiver can state that there is an error in the signal received but does not know where.

Error correcting code:

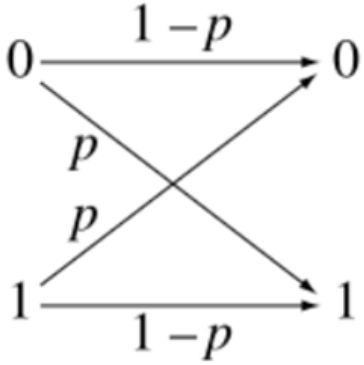
- The receiver restores the original message, even if there were errors in the transmission.

There may be cases where some errors can be corrected, and others can only be detected.

The Binary Symmetric Channel (BSC)

A convenient model for the noisy communication channel:

Prob(received bit=1 | sent bit=0) = *p*
Prob(received bit=0 | sent bit=1) = *p*
Error probability (of any single bit) satisfies *p < 1/2*



Errors on different subsets of bits are mutually independent.
Bits neither appear nor disappear.

Model is over simplified, yet very useful in practice.

Implication of the models

- The signal sent is a string of bits of a fixed size n.
- The signal received has the same length n as the signal sent.
- An error means that one or more bits were flipped
  - 0 was sent but 1 received
  - or 1 was sent but 0 received.
- A single error occurs with probability $n \cdot p \cdot (1-p)^{n-1}$
- Two errors occur with probability $\binom{n}{2} \cdot p^2 \cdot (1-p)^{n-2}$
- In general, we expect roughly pn of the bits to flip (note that $0 \leq p < \frac{1}{2}$)
- The Hamming Distance (defined next) between the signal sent and the one received will model the number of errors that occurred.

Hamming Distance

Richard W. Hamming (1915 –1998)

- Let $x, y \in \Sigma^n$ be two length n words over alphabet Σ. The Hamming distance between x, y is the number of coordinates where they differ.

- The Hamming distance satisfies the three usual requirements from a distance function
  1. For every x, $\Delta(x,x) = 0$
  2. For every x, y, $\Delta(x,y) = \Delta(y,x) \geq 0$, with equality iff $x = y$
  3. For every x, y, z, $\Delta(x,y) + \Delta(y,z) \geq \Delta(x,z)$ (triangle inequality)

  where $x, y, z \in \Sigma^n$ (same length)

- Examples:
  1. $\Delta(00101, 00101) = 0$
  2. $\Delta(00101, 11010) = 5$ (maximum possible for length 5 vectors)
  3. $\Delta(00101, 00101000)$ is undefined (unequal lengths)
  4. $\Delta(BEN, RAN) = 2$

Hamming Distance

```
def hamming_distance(s1, s2):
    assert len(s1) == len(s2)
    return sum(s1[i] != s2[i] for i in range(len(s1)))
```

```
>>> hamming_distance((1,2,3),(3,4,5))
3
>>> hamming_distance("00101","00101")
0
>>> hamming_distance("00101","11010")
5
>>> hamming_distance("00101","001010")
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    hamming_distance("00101","001010")
  File "/Users/benny/Dropbox/InttroCS2012/Code/intro23/Hamming.py",
    assert len(s1) == len(s2)
AssertionError
```

Definitions (1)

- A one-to-one *Encoding* function $C: \{0,1\}^k \rightarrow \{0,1\}^n$ from k to n bits ($k < n$)
- A *message* $m \in \{0,1\}^k$ mapped to a *codeword* $x = C(m) \in \{0,1\}^n$
- The set of codewords, often called the code, is:

$$Im(C) = \{y \in \{0,1\}^n \mid \exists x \in \{0,1\}^k, C(x) = y\}$$

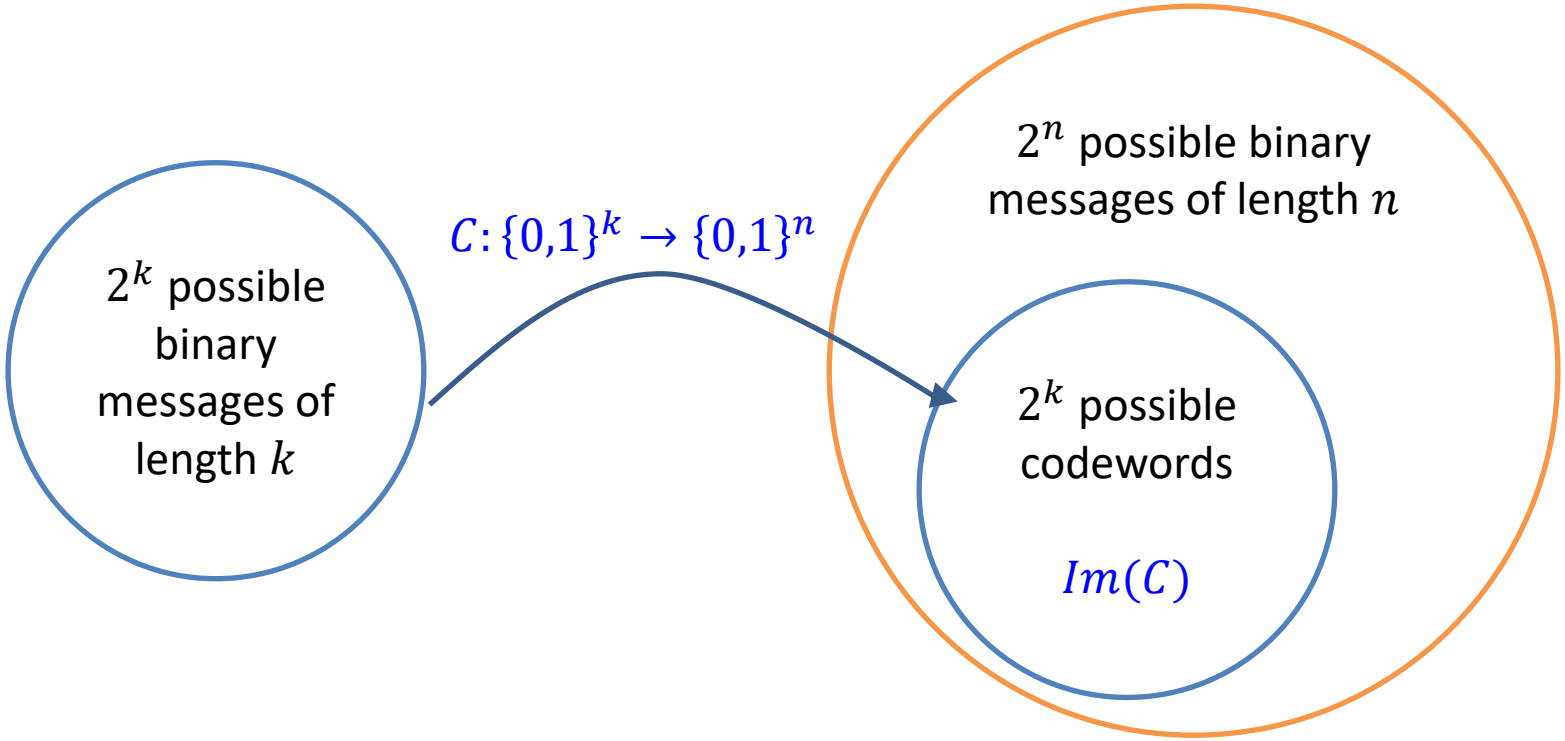
It holds that:

- $Im(C) \subset \{0,1\}^n$
- $|Im(C)| = 2^k$

- Throughout this lecture we will use C and $Im(C)$ interchangeably to denote the code

Definitions (2)

- A one-to-one *Encoding* function $C: \{0,1\}^k \rightarrow \{0,1\}^n$ from k to n bits ($k < n$)
- A *message* $m \in \{0,1\}^k$ mapped to a *codeword* $x = C(m) \in \{0,1\}^n$
- The set of codewords, often called the code, is $Im(C)$



Definitions (3)

- A *Decoding* function $D: \{0,1\}^n \rightarrow \{0,1\}^k$ from n to k bits ($k < n$)
- Obviously: $D\big(C(m)\big) = m$

Definitions (4)

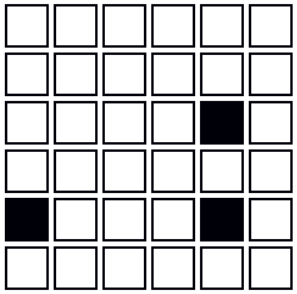
- Sender sends the codeword $C(m)$
- Receiver gets a noisy codeword $\widetilde{C(m)}$
  - Possibly $\widetilde{C(m)} \neq C(m)$
  - Hopefully $D\left(\widetilde{C(m)}\right) = m$

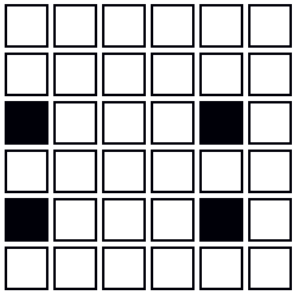
Closest Codeword Decoding - Definitions

- Let $\Delta(y, z)$ denote the Hamming distance between y, z
- Given a code C and an element $t \in \{0,1\}^n$, the closest codeword decoding function D maps t to a message $m \in \{0,1\}^k$ that minimizes $\Delta(t, C(x))_{x \in \{0,1\}^k}$
- If there is more than one codeword $C(m)$ that attains the minimum distance, then the decoding function D announces an error

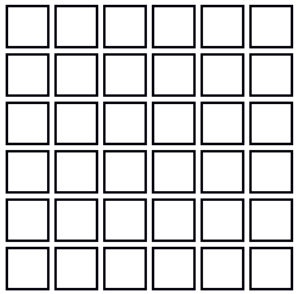
Closest Codeword Decoding: Example

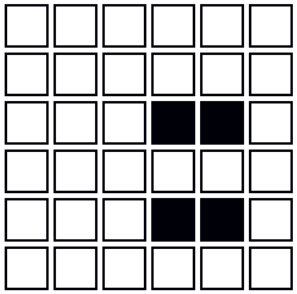
In the card magic code, suppose we receive the string over $\{0,1\}^6 \times \{0,1\}^6$ on the left. There is a single codeword at distance 1 from this string, depicted to the right.





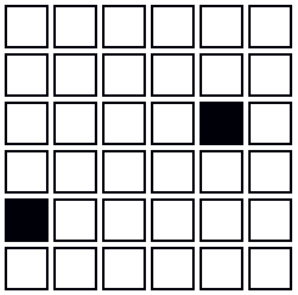
There is no codeword at distance 2 from this string (why?), and many that are at distance 3. Some of those are shown below.



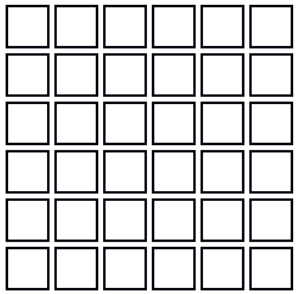


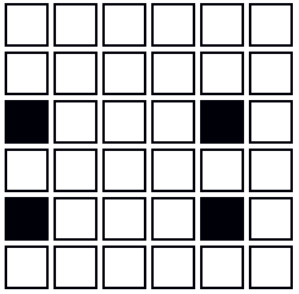
Closest Codeword Decoding: Example 2

In the card magic code, suppose we receive the following string over $\{0,1\}^6 \times \{0,1\}^6$:

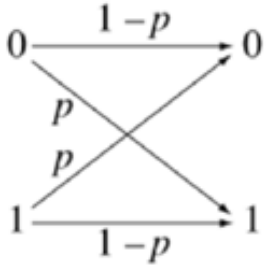


There is no codeword at distance 1 from this string (why?). There are exactly two codewords that are at distance 2 from this string. They are shown below. In such situation, closest codeword decoding announces an error.





Closest Codeword is Maximum Likelihood Decoding



Observation: For the binary symmetric channel ($p < \frac{1}{2}$), closest codeword decoding of t, if defined, outputs the message m that maximizes the likelihood of producing t, namely

$$\forall x \neq m \in \{0,1\}^k:$$
$$\Pr[t \text{ received} \mid C(x) \text{ sent}] < \Pr[t \text{ received} \mid C(m) \text{ sent}].$$

Proof: If $C(m)$ has distance s and $C(x)$ has distance $s + r$ (for some positive r) then $\Pr[t \text{ received} \mid C(m) \text{ sent}] = p^s(1-p)^{n-s}$, while $\Pr[t \text{ received} \mid C(x) \text{ sent}] = p^{s+r}(1-p)^{n-(s+r)}$.

Since $p < 1/2$ we obtain the claim. ■

Minimum Hamming Distance of Codes: Definition

- The minimum distance d of a code is the minimal Hamming distance between pairs of codewords:

$$d = \Delta(C) = \min_{y \neq x \in Im(C)} \{\Delta(y, x)\}$$

- In words: The minimum distance d of a code C is the minimal Hamming distance over all pairs of codewords in $Im(C)$
- d is an important parameter that determines how many errors may be detected and/or corrected.
  - Note that to evaluate the performance of a code we need to consider its worst-case performance.

Error detection vs. correction

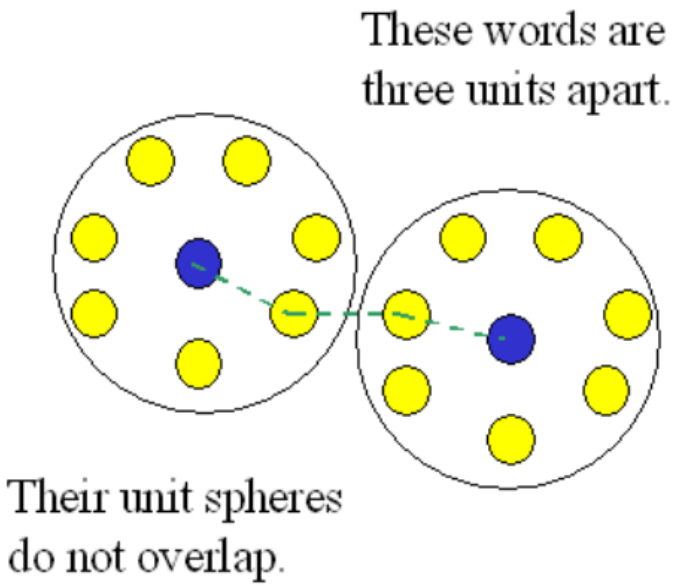
- Given a reception $\widetilde{C(m)}$ which contains errors
  - Error detection: detecting that $\widetilde{C(m)}$ cannot be a codeword
  - Error correction: returning the unique closest codeword to $\widetilde{C(m)}$
- Error detection is often easier than error correction

Hamming Distances in the 2D Card Code

- The minimum distance d of the 2D card code is 4
- So, words whose Hamming distance from a codeword is 1, are at distance 3 from another codeword
- When such a word is received, it is assumed the correct codeword is the one at distance 1 (higher probability than the one at distance 3)
- So, the 2D cards code can correct any single error
- Two errors cannot be corrected, because there are (at least) two equal probability codewords (both at Hamming distance 2)
- If we only want to detect errors, the 2D cards code can detect up to three-bits errors

An Important Geometric Property

Proposition:
Suppose $d = \Delta(C)$ is the minimum distance of a code, C.
Then this code is capable of detecting up to $d - 1$ errors, and correcting up to $\lfloor (d-1)/2 \rfloor$ errors.



(figure from course EE 387, by John Gill, Stanford University, 2010.)

An Important Geometric Property, cont.

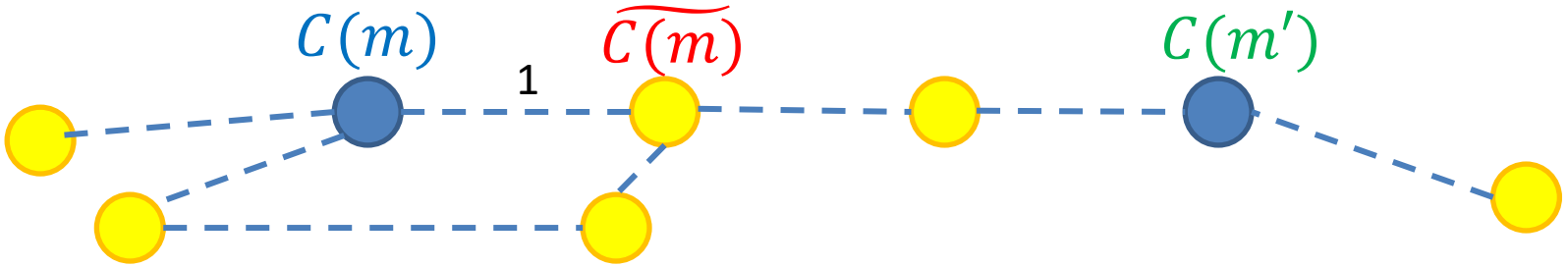
Proposition:
Suppose $d = \Delta(C)$ is the minimum distance of a code C.
Then this code is capable of detecting up to $d - 1$ errors.

Proof: Let $C(m) \in \{0,1\}^n$ be a codeword. Suppose it experienced h errors, where $1 \leq h \leq d - 1$. In other words, $C(m)$ was sent, and $\widetilde{C(m)}$ was received, where the Hamming distance between $C(m)$ and $\widetilde{C(m)}$ is h.
The minimum distance of this code is d. Therefore, $\widetilde{C(m)}$ cannot be a codeword. The receiving side can detect this fact.

An Important Geometric Property, cont.

Proposition:
Suppose $d = \Delta(C)$ is the minimum distance of a code C.
Then this code is capable of correcting up to $\lfloor (d-1)/2 \rfloor$ errors.

Demonstrating with $\boldsymbol{d = 3}$:

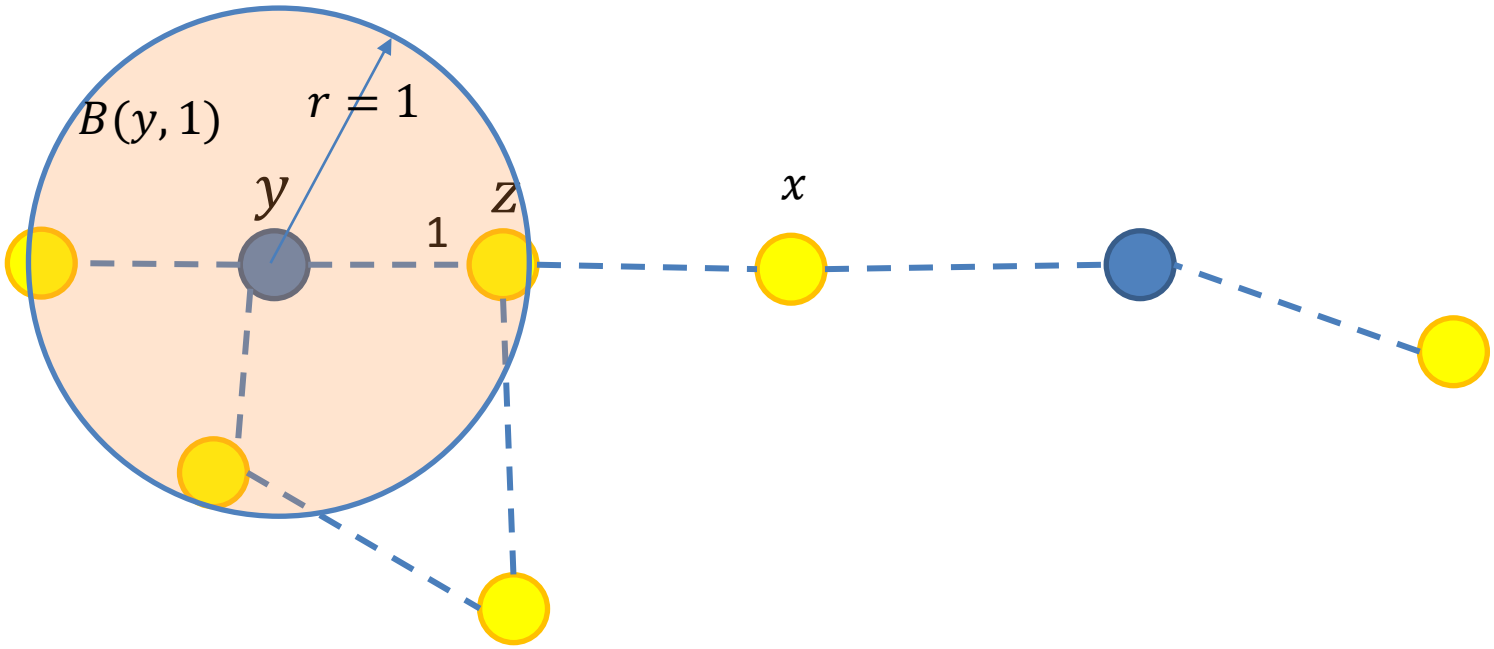


If $C(m)$ was sent but $\boldsymbol{h \le \lfloor (d-1)/2 \rfloor}$ errors have occurred, then the reception $\widetilde{C(m)}$ is not a codeword and $C(m)$ is the closest codeword to $\widetilde{C(m)}$.
If not, there are two codewords $C(m), C(m')$ such that:
$\Delta\left(C(m), \widetilde{C(m)}\right) = \boldsymbol{h}$ and $\Delta\left(C(m'), \widetilde{C(m)}\right) \le \boldsymbol{h} \longrightarrow$
By the triangle inequality $\Delta(C(m), C(m')) \le \boldsymbol{2 \cdot h \le d - 1 < d} \longrightarrow$ contradiction.

Equivalent Definitions

- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set

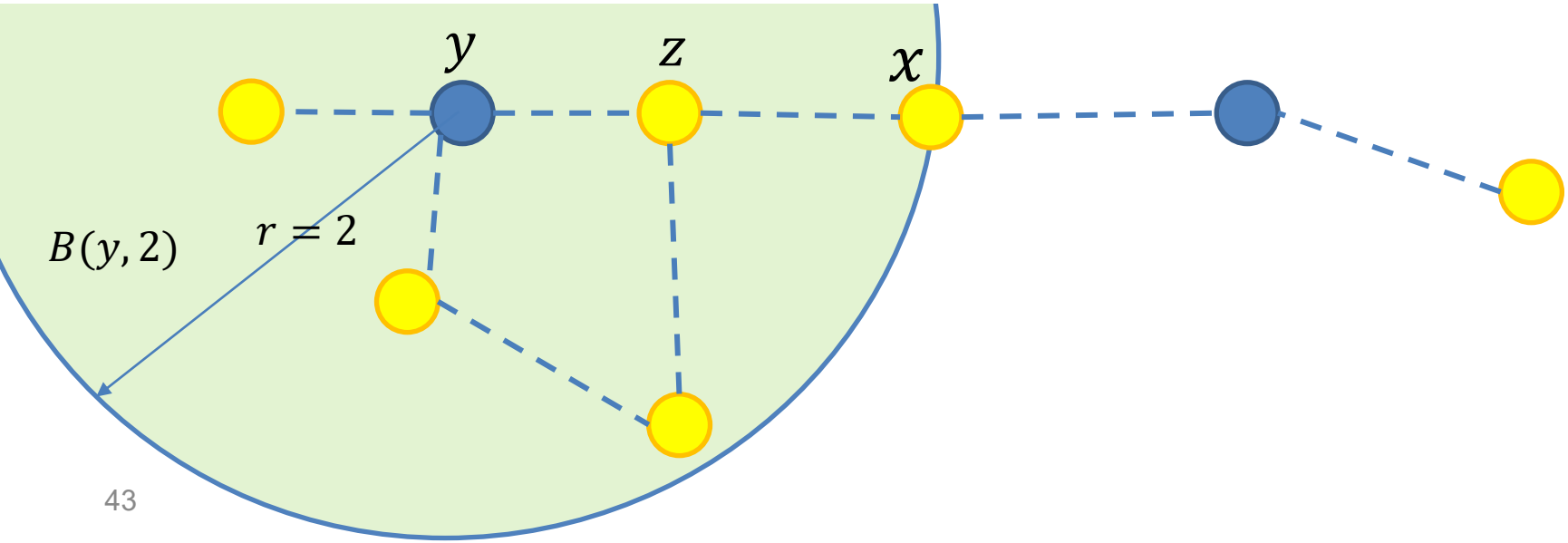
$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$



Equivalent Definitions

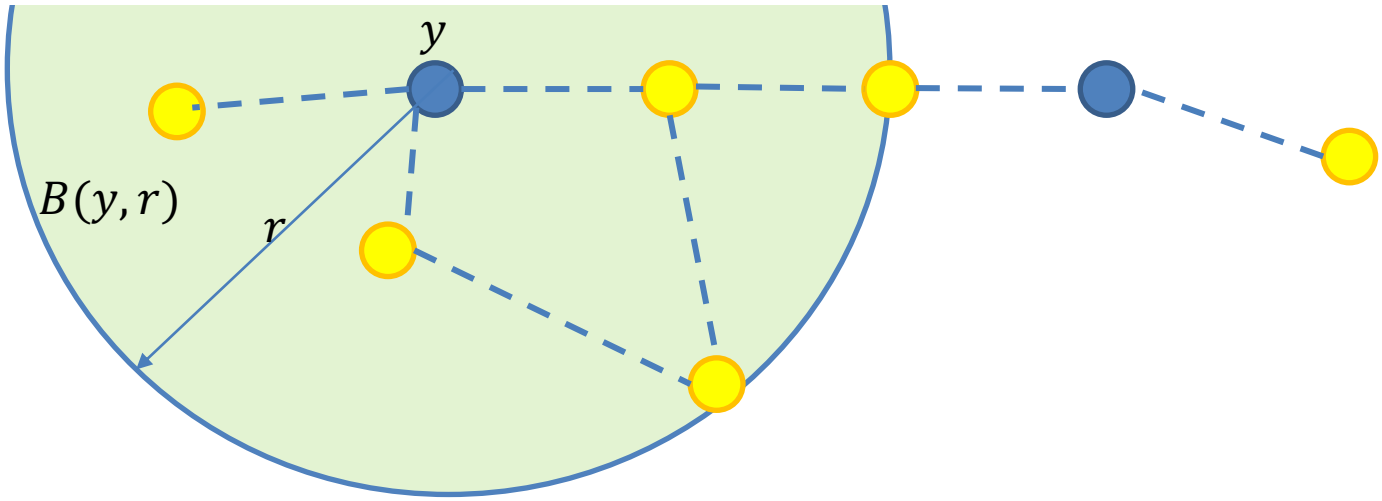
- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set

$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$



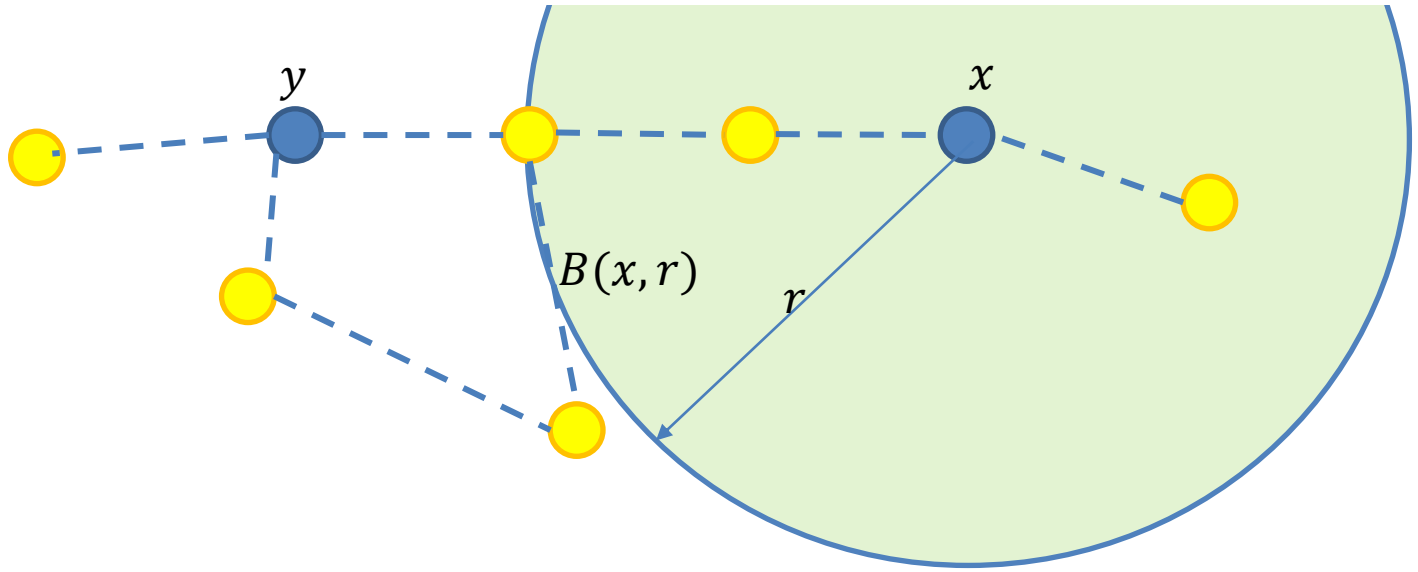
Equivalent Definitions

- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set
$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$
- A code is capable of detecting r errors if
for every codeword $y \in Im(C)$, $B(y,r) \cap Im(C) = \{y\}$



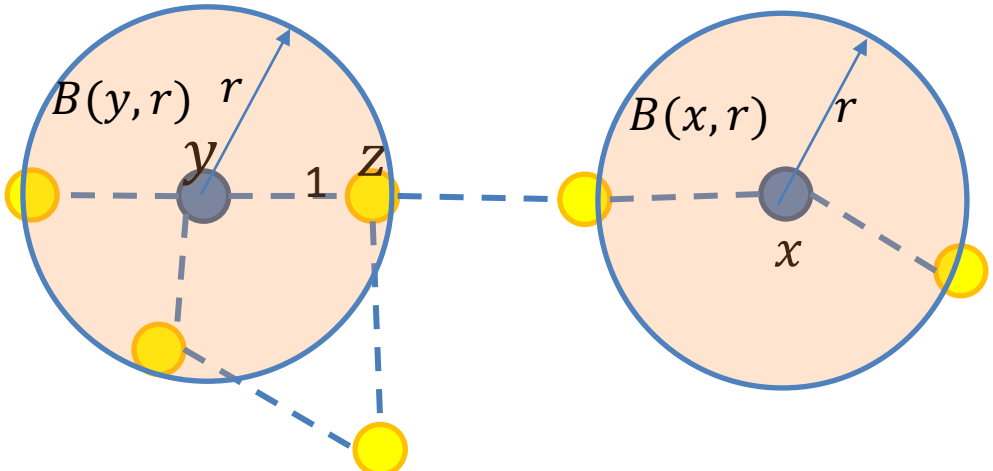
Equivalent Definitions

- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set
$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$
- A code is capable of detecting r errors if
for every codeword $y \in Im(C)$, $B(y,r) \cap Im(C) = \{y\}$



Equivalent Definitions

- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set
$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$
- A code C is capable of detecting r errors if
for every codeword $y \in Im(C)$, $B(y,r) \cap Im(C) = \{y\}$
- A code C is capable of correcting r errors if
for every codewords $y \neq x \in Im(C)$, $B(y,r) \cap B(x,r) = \emptyset$



Equivalent Definitions

- Let $y \in \{0,1\}^n$
- The ball of radius r around y is the set
$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$
- A code C is capable of detecting r errors iff
for every codeword $y \in Im(C)$, $B(y,r) \cap Im(C) = \{y\}$
- A code C is capable of correcting r errors iff
for every codewords $y \neq x \in Im(C)$, $B(y,r) \cap B(x,r) = \emptyset$

Do these definitions make sense to you?

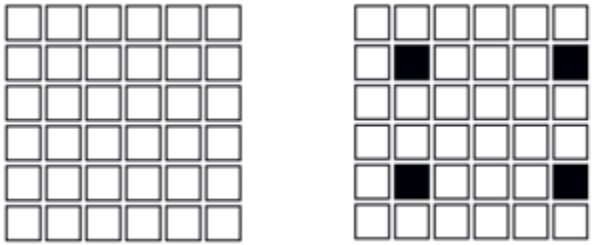
Minimum Distance of the ID Code

- $k = 8$ (length of message)
- $n = 9$ (length of codeword)
- The minimum distance of the ID code is $d = 2$.

  - Therefore, ID code is capable of detecting any single digit error.
  - But a single digit error cannot be corrected, because there are two codewords at Hamming distance 1 from it.
  - There are combinations of two digit errors it cannot detect.

Minimum Distance of the "Card Magic" Code

- $k = 25$
- $n = 36$
- The minimum distance of the ID code is $d = 4$.



- This code is capable of detecting ≤ 3 errors
- This code is capable of correcting 1 error

Minimum Distance of Codes

We will now see 2 additional simple codes:

- Repetition code (the case of 3 copies), where $d = 3$
- Parity (1-dimensional) check code, where $d = 2$

And one more sophisticated code:

- The Hamming (7,4,3) code, where $d = 3$

Repetition Code

- Original message of k bits
- t is the repetition parameter
  - Each bit is repeated t times
- Each codeword is of length $n = k \cdot t$

Repetition Code

An example with $k = 2, t = 3$:

In the following code, the original messages are two bits long. The encoder repeats each original bit, two more times.

original		encoded
2 bits		6 bits
00	$\longrightarrow$	000000
01	$\longrightarrow$	000111
10	$\longrightarrow$	111000
11	$\longrightarrow$	111111

This code can correct any single error. For example, suppose the decoder recieves 001000. The single flipped bit can only be 001000, leading to the codeword 000000 and original message 00.

Remark: Of course, the channel is not aware of a difference between "regular" (original) bits and "blue" (duplicated) bits.

Hamming Distances in the Repetition Code with repetition parameter $t = 3$

- The minimum distacnce of the Repetition Code is 3.
- So the Repetition Code is capable of correcting any single digit error.
- But some combinations of two digit errors can also be corrected: when one of the errors is in the first 3 bits, and one in the last 3 bits.
- Other combinations of two digit errors cannot be corrected - they look just like a single error!
- So it's best to treat each block of 3 bits separately.
- What is the minimum distance of a Repetition code for a single bit source (3 bit codewords)?

Decoding algorithm for Repetition Codes

So will decode blocks of 3 bits.

received signal		decoded message
3 bits		1 bit
$000, 100, 010, 001$	$\longrightarrow$	0
$111, 011, 101, 110$	$\longrightarrow$	1

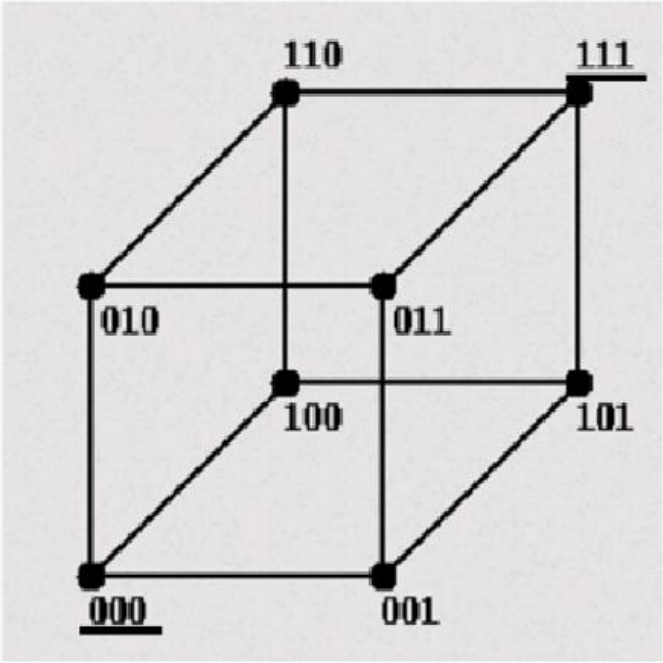
Decoding rule: If the received 3 bit signal is at Hamming distance 0 or 1 from one of the two codewords, the decoded message is the original message encoded by this codeword. This covers all possible cases.

So if the decoder recieves 001100. It can identify the codeword as 000000 and the original message is 00.

Geometric Interpretation of the Repetition Code with $k = 1,\ t = 3$

k=1 (length of message)
n=3 (length of codeword)
d=3

Guaranteed: detect 2 errors, correct 1



Codewords are underlined.

Decoding the Repetition Code in Python

The input to the decoding function is a `reception` and the repetition parameter `t` of the code

```
def repetition_decode(reception, t):
    res = ""
    for i in range(0, len(reception), t):
        block = reception[i:i + t]
        # Select the majority per block
        if block.count("1") > block.count("0"):
            res += "1"
        else:
            res += "0"
    return res
```

Assuming that at most $\left\lfloor \frac{t-1}{2} \right\rfloor$ errors have occurred, `repetition_decode` returns the correct original message.

Parity Check Code

- Original message of k bits
- Add a parity bit
- Each codeword is of length $n = k + 1$

Parity Check Codes

In the following code, the original messages are two bits long. The encoder xors the two original bit (*i.e.* adds them modulo 2). The resulting bit is appended to the original message.

original		encoded
2 bits		3 bits
00	$\longrightarrow$	000
01	$\longrightarrow$	011
10	$\longrightarrow$	101
11	$\longrightarrow$	110

Parity Check Codes: Encoding

original		encoded
2 bits		3 bits
00	$\longrightarrow$	000
01	$\longrightarrow$	011
10	$\longrightarrow$	101
11	$\longrightarrow$	110

This code can detect any single error. But it cannot correct a single error. For example, suppose the decoder receives 001. The single flipped bit could be either 001 (encoded signal 000) or 001 (encoded signal 011) or 001 (encoded signal 101) .

Decoding Parity Check Codes

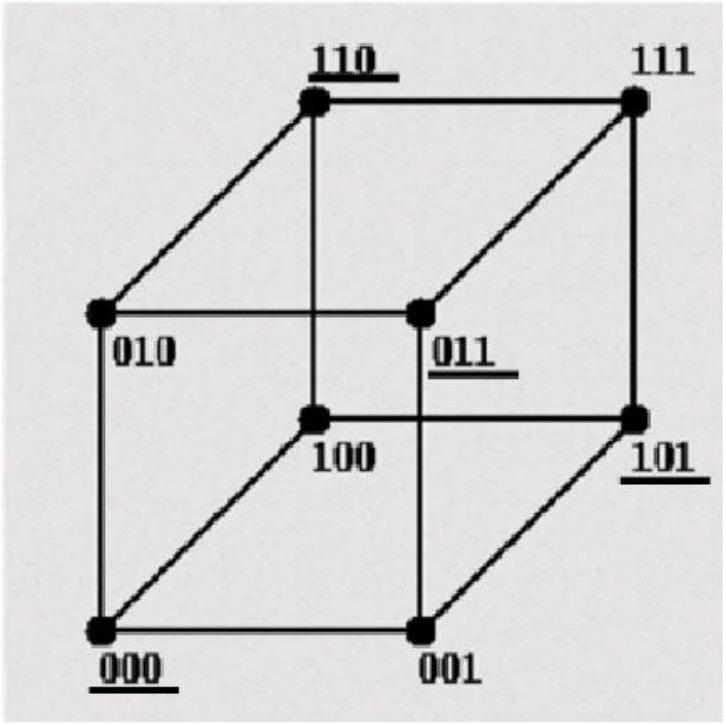
received signal		decoded message
3 bits		2 bits
000	$\longrightarrow$	00
001	$\longrightarrow$	"error"
010	$\longrightarrow$	"error"
011	$\longrightarrow$	01
100	$\longrightarrow$	"error"
101	$\longrightarrow$	10
110	$\longrightarrow$	11
111	$\longrightarrow$	"error"

- Decoding rule: If the received signal is one of the four codewords, decoded message is the original message encoded by this codeword. Otherwise, return error.
- In this simple example too, a full decoding dictionary is possible. But how could we avoid it?

Geometric Interpretation of Parity Bit Code

k=2 (length of message)
n=3 (length of codeword)
d=2 (in fact all HD are 2 here)

Can detect 1 error, correct 0



Codewords are underlined.

Repetition Code and Parity Check

- The repetition code we saw is hardly ever used – it expands messages threefold.
- The parity check code is in use, but it cannot correct even one error.
- Why can't the parity check code correct even a single error?
- What is the minimal distance of the parity check code?

- Next, we will see a more effective code, named the Hamming (7,4,3) code.
  - n is much smaller than that of the repetition code

The Hamming (7, 4, 3) Code

Let $C: \{0,1\}^k \rightarrow \{0,1\}^n$ be an encoding function and let d be the minimum distance of the corresponding code.

We say that this code is an (n, k, d) code.

We will now see the Hamming (7,4,3) code.

Hamming code is actually a family of codes. There are Hamming codes with different parameters, such as (15, 11, 3).

Let's start with an example

- Benny wants to send the message $msg =$"0011" to Rani (length 4)
- The codeword that Benny sends after encoding the message is $x =$"1000011" (length 7)
  - How is x computed?
    - 1 = (0 + 0 + 1) % 2 ⇒ (1+ 0 + 0 + 1) % 2 = 0
    - 0 = (0 + 1 + 1) % 2 ⇒ (0+ 0 +1 + 1) % 2 = 0
    - 0 = (0 + 1 + 1) % 2 ⇒ (0+ 0 +1 + 1) % 2 = 0

Let's start with an example

- Benny wants to send the message $msg =$"0011" to Rani (length 4)
- The codeword that Benny sends after encoding the message is $x =$"1000011" (length 7)
- Suppose that a single error has occurred (the 6th bit was flipped). Rani received the following transmission: $y =$ "100001"
- Could Rani detect that the transmission he received contains an error? Or perhaps even correct the error? Yes, as $d = 3$.

A decoding algorithm

We will now describe a closest-codeword decoding algorithm for Hamming (7,4,3) assuming that a single error has occurred.

Let's start with an example

- Rani received the noisy $y =$ "1000001"
- Rani computes:

  $b_1 = (1+0+0+1)\%2 = 0$

  $b_2 = (0+0+0+1)\%2 = 1$

  $b_3 = (0+0+0+1)\%2 = 1$

  $i = (b_3 b_2 b_1)_2 = (110)_2 = (6)_{10}$

  Decode:

  - Correct:

    $i \neq 0 \Rightarrow$ there is an error in the ith bit$\rightarrow$ flip it.

    $1000001 \xrightarrow{\text{correct}} 1000011$

  - Extract the message: 0011

Another example

- Benny wants to send the message $msg =$"0011" to Rani (length 4)
- The codeword that Benny sends after encoding the message is $x =$"1000011" (length 7)
- Suppose that a single error has occurred (the 2nd bit was flipped). Rani received the following transmission: $y =$ "1100011"

Another example

- Rani received the noisy $y =$ "1100011"
- Rani computes:

  $b_1 = (1+0+0+1)\%2 = 0$

  $b_2 = (1+0+1+1)\%2 = 1$

  $b_3 = (0+0+1+1)\%2 = 0$

  $i = (b_3 b_2 b_1)_2 = (010)_2 = (2)_{10}$

  Decode:

  - Correct:

    $i \neq 0 \Rightarrow$ there is an error in the ith bit$\rightarrow$ flip it.

    $1100011 \xrightarrow{\text{correct}} 1000011$

  - Extract the message: 0011

The Hamming (7, 4, 3) Code

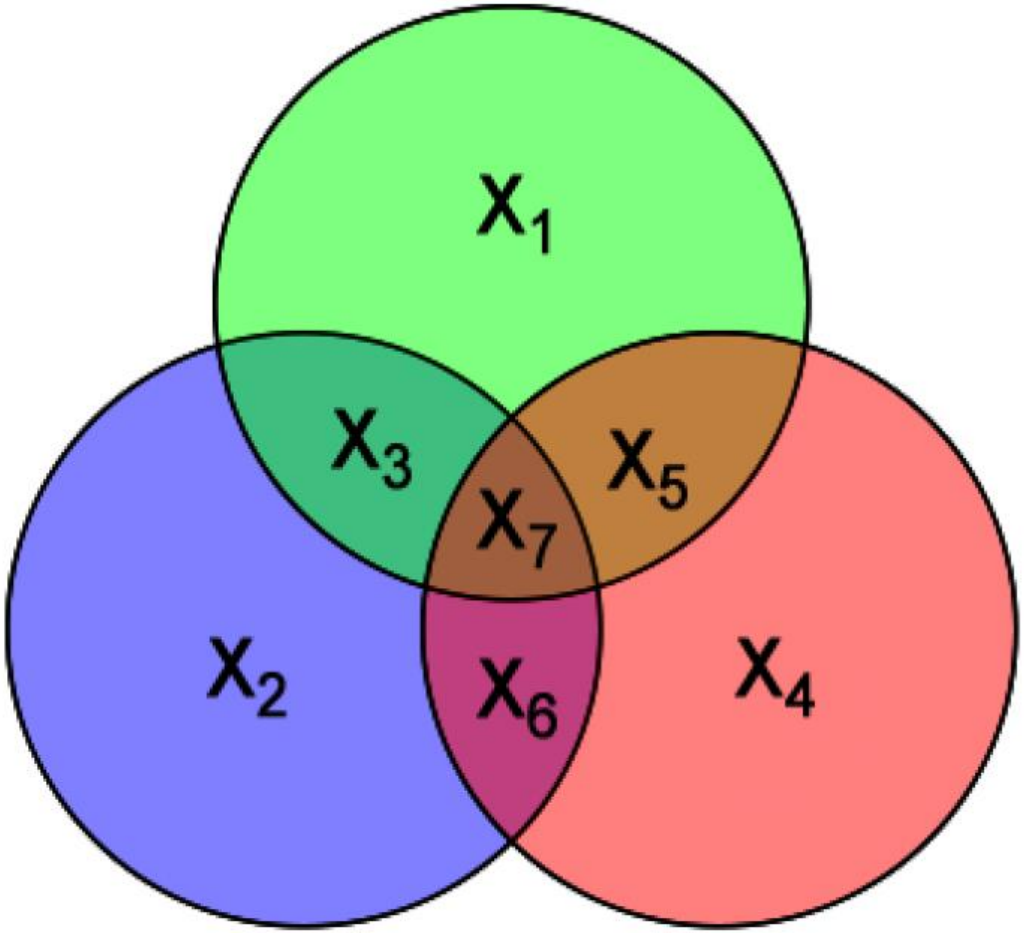
The Hamming encoder gets an original message consisting of 4 bits, and produces a 7 bit long codeword. For reasons to be clarified soon, we will number the bits in the original message in a rather unusual manner. For $(x_3, x_5, x_6, x_7) \in Z_2^4 = \{0, 1\}^4$,

$$(x_3, x_5, x_6, x_7) \longrightarrow \quad (x_1, x_2, x_3, x_4, x_5, x_6, x_7) \ ,$$

where x_1, x_2, x_4 are parity check bits, computed (modulo 2) as following:

$$\begin{aligned} x_1 &= x_3 + x_5 + x_7 \\ x_2 &= x_3 + x_6 + x_7 \\ x_4 &= x_5 + x_6 + x_7 \end{aligned}$$

The Hamming (7, 4, 3) Code



(modified from Wikipedia image)

Hamming Encoding in Python

This Hamming code has $2^4 = 16$ codewords. This is small enough to enable us to describe the encoding procedure using a table lookup, which in Python typically means a dictionary.

But as the size of the code grows, table lookup becomes less attractive.

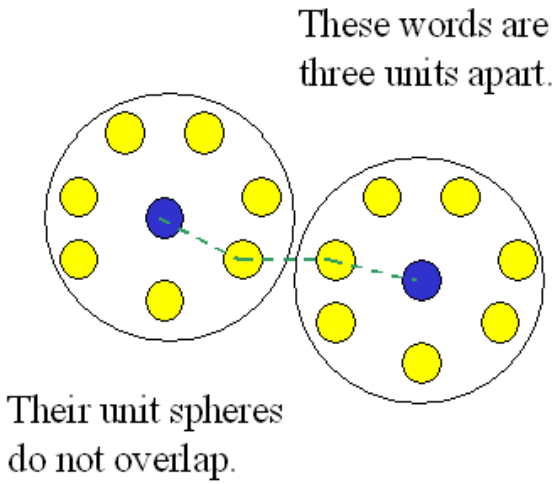
```
def hamming_encode(x3,x5,x6,x7):
    x1= (x3+x5+x7) % 2
    x2= (x3+x6+x7) % 2
    x4= (x5+x6+x7) % 2
    return (x1,x2,x3,x4,x5,x6,x7)
>>> hamming_encode(0,0,0,0)
(0, 0, 0, 0, 0, 0, 0)
>>> hamming_encode(1,0,0,0)
(1, 1, 1, 0, 0, 0, 0)
>>> hamming_encode(0,1,0,0)
(1, 0, 0, 1, 1, 0, 0)
>>> hamming_encode(0,0,1,0)
(0, 1, 0, 1, 0, 1, 0)
```

Geometry of Hamming (7, 4, 3) Code

Let $\mathcal{C}_{\mathcal{H}}$ be the set of $2^4 = 16$ codewords in the Hamming code. A simple computation shows that $\mathcal{C}_{\mathcal{H}}$ equals

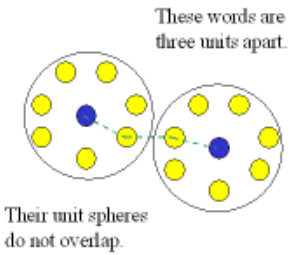
$\{ (0,0,0,0,0,0,0), (1,1,0,1,0,0,1), (0,1,0,1,0,1,0), (1,0,0,0,0,1,1)$
$(1,0,0,1,1,0,0), (0,1,0,0,1,0,1), (1,1,0,0,1,1,0), (0,0,0,1,1,1,1)$
$(1,1,1,0,0,0,0), (0,0,1,1,0,0,1), (1,0,1,1,0,1,0), (0,1,1,0,0,1,1)$
$(0,1,1,1,1,0,0), (1,0,1,0,1,0,1), (0,0,1,0,1,1,0), (1,1,1,1,1,1,1) \}.$

By inspection, the Hamming distance between two codewords is ≥ 3. Therefore the unit spheres around different codewords do not overlap.



(figure from course EE 387, by John Gill, Stanford University, 2010.)

Closest Codeword Decoding of Hamming (7, 4, 3)



This implies that if we transmit a codeword in $\{0,1\}^7$ and the channel changes at most one bit in the transmission (corresponding to a single error), the received word is at distance 1 from the original codeword, and its distance from any other codeword is ≥ 2.

Thus, decoding the received message by taking the closest codeword to it, guarantees to produce the original message, provided at most one error occurred.

Questions

1. How do we find the closest codeword (for our Hamming code).
2. What happens if more than a single error occurs?

Decoding Hamming (7, 4, 3) Code

k = 4 is small enough that we can still decode exhaustively, by a table lookup (using dict in Python). But there is a much cooler way to decode this specific code.

Let $(y_1, y_2, y_3, y_4, y_5, y_6, y_7)$ be the 7 bit signal received by the decoder. Assume that at most one error occurred by the channel. This means that the sent message, $(x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ differs from the received signal in at most one location (bit).

Let the bits b_1, b_2, b_3 be defined (modulo 2) as follows:

$$b_1 = y_1 + y_3 + y_5 + y_7$$
$$b_2 = y_2 + y_3 + y_6 + y_7$$
$$b_3 = y_4 + y_5 + y_6 + y_7$$

Writing the three bits (from left to right) as $b_3 b_2 b_1$, we get the binary representation of an integer ℓ in the range $\{0,1, \dots, 7\}$.

Decoding Hamming (7, 4, 3) Code

Let the bits b_1, b_2, b_3 be defined (modulo 2) as follows:

$$b_1 = y_1 + y_3 + y_5 + y_7$$
$$b_2 = y_2 + y_3 + y_6 + y_7$$
$$b_3 = y_4 + y_5 + y_6 + y_7$$

Writing the three bits (from left to right) as $b_3 b_2 b_1$, we get the binary representation of an integer ℓ in the range $\{0,1,\ldots,7\}$.

- Interpret $\ell = 0$ as "no error" and return bits 3,5,6,7 of the received signal.
- Interpret other values as "error in position ℓ ", and flip y_ℓ. Return bits 3,5,6,7 of the result.

Decoding Hamming (7, 4, 3) Code: Why Does It Work?

Recall the bits b_1, b_2, b_3 be defined (modulo 2) as following

$$\begin{aligned} b_1 &= y_1 + y_3 + y_5 + y_7 \\ b_2 &= y_2 + y_3 + y_6 + y_7 \\ b_3 &= y_4 + y_5 + y_6 + y_7 \end{aligned}$$

- If there was no error (for all i, $x_i = y_i$) then from the definition of x_1, x_2, x_4, it follows that $b_3 b_2 b_1$ is zero, and we correct nothing.
- If there is an error in one of the parity check bits, say $x_2 \neq y_2$. Then only the corresponding bit is non zero ($b_2 = 1$ in this case). The position ℓ ($010 \to 2$ in this case) points to the bit to be corrected.
- If there is an error in one of the original message bits, say $x_5 \neq y_5$. Then the bits of the binary representation of this location will be non zero ($b_1 = b_3 = 1$ in this case). The position ℓ ($101 \to 5$ in this case) points to the bit to be corrected.

Decoding Hamming (7, 4, 3) Code: Binary Representation

Recall the Hamming encoding

$$\begin{aligned} x_1 &= x_3 + x_5 + x_7 \\ x_2 &= x_3 + x_6 + x_7 \\ x_4 &= x_5 + x_6 + x_7 \end{aligned}$$

The locations of the parity bits x_1, x_2, x_4, are all powers of two.

- x_1 corresponds to indices 3,5,7 having a 1 in the first (rightmost) position of their binary representations 011, 101, 111.
- x_2 corresponds to indices 3,6,7 having a 1 in the second (middle) position of their binary representations 011, 110, 111.
- x_4 corresponds to indices 5,6,7 having a 1 in the last (leftmost) position of their binary representations 101, 110, 111.

Decoding Hamming (7, 4, 3) Code: Python

```
def hamming_decode(recep):
    # We start indexing from 1 so we add a dummy bit
    recep = [0] + list(recep)

    # Compute parity bits
    b1 = (recep[1] + recep[3] + recep[5] + recep[7]) % 2
    b2 = (recep[2] + recep[3] + recep[6] + recep[7]) % 2
    b3 = (recep[4] + recep[5] + recep[6] + recep[7]) % 2

    # Interpret as int
    err = int(str(b3)+str(b2)+str(b1), 2)

    # Flip error bit, if err == 0, no effect
    recep[err] = 1 - recep[err]

    # Return original message bits
    return [recep[i] for i in [3, 5, 6, 7]]

>>> hamming_encode(0,0,1,1)
(1, 0, 0, 0, 0, 1, 1)
>>> hamming_decode((1, 0, 0, 0, 0, 1, 1))
[0, 0, 1, 1]
>>> hamming_decode((1, 0, 0, 0, 0, 0, 1))  #contains an error
[0, 0, 1, 1]
```

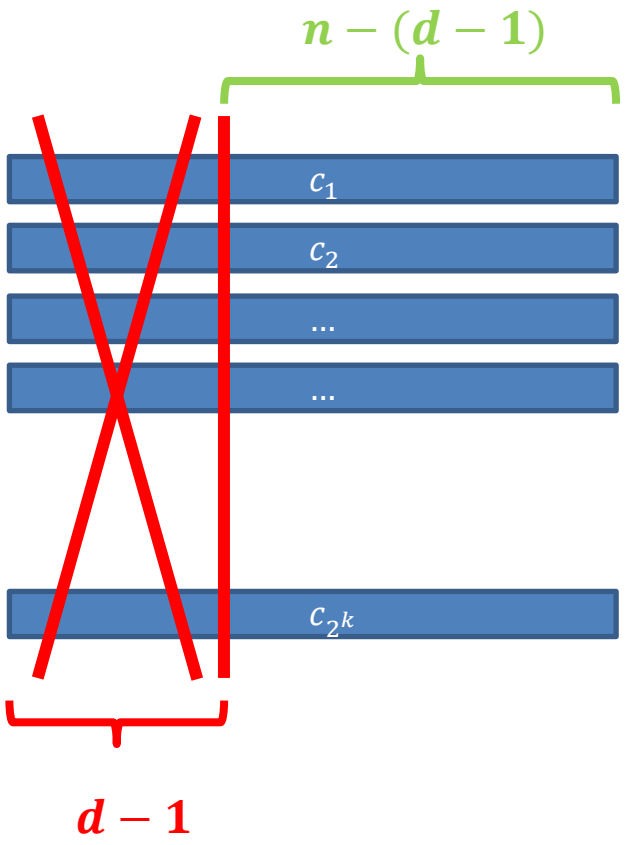
Relation between k,n,d

- Given k (length of message), we wish to:
  - Maximize d – codewords are "**far apart**"
  - Minimize n – codewords are "**short**"
- Intuitively, contradictory goals
- Provably as well

Why $\min n$, $\max d$ is impossible

- Write down all 2^k codewords $C(x)$
- They each **differ** in **at least** d coordinates
- Erase the first $d-1$ coordinates
- All words still differ (why?)
- New words have length $n-(d-1)$
- At most $2^{n-(d-1)}$ such words
- $2^k \leq 2^{n-(d-1)}$ (why?)
- In other words:

$$k \leq n-(d-1)$$
$$\Downarrow$$
$$\boldsymbol{d \leq n-k+1}$$



Balls in $\{0,1\}^n$ and their Volume

Recall that the ball of radius r around $y \in \{0,1\}^n$ is the set

$$B(y,r) = \{z \in \{0,1\}^n \mid \Delta(y,z) \leq r\}$$

The volume of this ball is defined as the number of elements of $\{0,1\}^n$ it contains.

For $r = 1$: The number of elements from $\{0,1\}^n$ at distance exactly 1 from y is n. Obviously, y is also in the ball $B(y,1)$. Thus, the volume of the unit ball $B(y,1)$ in $\{0,1\}^n$ is $n+1$.

For any natural $r \geq 1$:

There are $\binom{n}{h}$ elements at distance exactly h from y.

Therefore, the volume of $B(y,r)$ is $\sum_{h=0}^{r}\binom{n}{h}$.

Volume Bound for $(n, k, 3)$ Codes

- Let C be a $(n, k, 3)$ code.
- This implies that balls of radius 1 around codewords are disjoint.
- Each such ball contains exactly $n + 1$ points (why?).
- There are 2^k such balls (one around each codeword).
- Since the balls are disjoint, no point in $\{0, 1\}^n$ appears twice in their union.
- Thus $2^k \cdot (n + 1) \leq 2^n$.
- The repetition code we saw is a $(6, 2, 3)$ code.
  And indeed, $2^2 \cdot (6 + 1) = 28 < 2^6 = 64$.

Volume Bound for General (n, k, d) Codes

- Let C be a (n, k, d) code. Then $2^k \cdot \sum_{h=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{h} \leq 2^n$.
- This is called the volume, sphere packing, or Hamming, bound.
- Example: The card magic code is a $(36, 25, 4)$ code. Indeed,

$$2^k \cdot \sum_{h=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{h} = 2^{25}(1 + 36) = 1,241,513,984$$

$$< 2^{36} = 68,719,476,736 \ .$$

- Proof idea: Spheres at radius $\lfloor (d-1)/2 \rfloor$ around the 2^k codewords are all disjoint. Thus the "volume" of their union cannot be greater than the "volume" of the whole space, which is 2^n.

(n,k,d) Codes: Rate and Relative Distance

The rate of an (n,k,d) code is k/n. It measures the ratio between the number of information bits, k, and the transmitted bits, n.

The relative distance of the code is d/n.

- The repetition code we saw is a $(6,2,3)$ code. Its rate is $\frac{2}{6} = \frac{1}{3}$. Its relative distance is $\frac{3}{6} = \frac{1}{2}$.
- The parity check code is a $(3,2,2)$ code. Its rate and relative distance are both $\frac{2}{3}$.
- The card magic code is a $(36,25,4)$ code. Its rate is $\frac{25}{36}$, and its relative distance is $\frac{4}{36} = \frac{1}{9}$.

Goal in Code Design

- Large rate, the closer to 1 the better (smaller number of additional bits, and thus smaller communication overhead).
- Large relative distance (related to lower error in decoding).
- Efficient encoding.
- Efficient decoding (computationally).

Codes Used in RAID (for reference only)

RAID (an acronym for redundant array of independent disks) is a storage technology that combines multiple disk drive components into a logical unit. Data is distributed across the drives in one of several ways called "RAID levels", depending on what level of redundancy and performance (via parallel communication) is required (text from Wikipedia).

RAID architectures are also concerned with errors, of course. A relatively frequent error is that one out of $n + 1$ disks crashes. Notice that in such a case we know which is the crashed disk. In our notation, this corresponds to a signal with $n + 1$ bits, one of which is missing. An easy generalization of our parity check code can be employed to recover the crashed bit (disk).

Remark: Different error protection schemes are used in various RAID architectures.