8 Error-Correcting Codes

28 April 2021

Sebastian Wild

version `2021-04-28 10:46`

Outline

8 Error-Correcting Codes

8.1 Introduction

Noisy Communication

- most forms of communication are "noisy"
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

Noisy Communication

- most forms of communication are "noisy"
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary

Noisy Communication

- most forms of communication are “noisy”
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary

- But how is it possible (for us)
  to decode a message in the presence of noise & errors?

Bcaesue it semes taht ntaurul lanaguge has a lots fo ***redundancy*** *bilt itno it!*

Noisy Communication

- most forms of communication are "noisy"
  - humans: acoustic noise, unclear pronunciation, misunderstanding, foreign languages

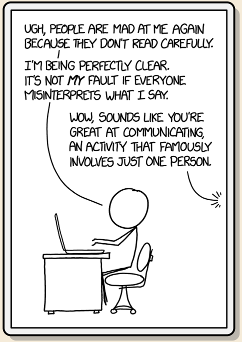
- How do humans cope with that?
  - slow down and/or speak up
  - ask to repeat if necessary

- But how is it possible (for us)
  to decode a message in the presence of noise & errors?

*Bcaesue it semes taht ntaurul lanaguge has a lots fo **redundancy** bilt itno it!*

$\rightsquigarrow$ We can

1. **detect errors** "This sentence has aao pi dgsdho gioasghds."
2. **correct** (some) **errors** "Tiny errs ar corrrected automaticly."
   (sometimes too eagerly as in the Chinese Whispers / Telephone)



Noisy Channels

- computers: copper cables &
  electromagnetic interference
- transmit a binary string
- but occasionally bits can "flip"

⇝ want a robust code

Noisy Channels

- computers: copper cables &
  electromagnetic interference
- transmit a binary string
- but occasionally bits can “flip”

$\leadsto$ want a robust code

- We can aim at
  1. **error detection** $\leadsto$ can request a re-transmit
  2. **error correction** $\leadsto$ avoid re-transmit for common types of errors

Noisy Channels

- computers: copper cables &
  electromagnetic interference
- transmit a binary string = message
- but occasionally bits can "flip"

⇝ want a robust code

- We can aim at
  1. **error detection** ⇝ can request a re-transmit
  2. **error correction** ⇝ avoid re-transmit for common types of errors
- This will require *redundancy*: sending *more* bits than plain message
  ⇝ **goal:** robust code with lowest redundancy

that's the opposite of compression!

Clicker Question

What do you think, how many extra bits do we need to **detect** a **single bit error** in a message of 100 bits?

sli.do/comp526

Click on "Polls" tab

Clicker Question

What do you think, how many extra bits do we need to **correct** a **single bit error** in a message of 100 bits?

`sli.do/comp526`

Click on “Polls” tab

8.2 Lower Bounds

Block codes

- **model:**
  - want to send message $S \in \{\mathtt{0},\mathtt{1}\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** ($\mathtt{0} \to \mathtt{1}$ resp. $\mathtt{1} \to \mathtt{0}$)
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{\mathtt{0},\mathtt{1}\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)
  - $\rightsquigarrow$ what errors can be detected and/or corrected?

Block codes

- **model:**
  - want to send message $S \in \{0,1\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** $(0 \to 1$ resp. $1 \to 0)$
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{0,1\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)
  - ⇝ what errors can be detected and/or corrected?
- all codes discussed here are ***block codes***
  - divide S into *messages* $m \in \{0,1\}^k$ of k bits each (k = *message length*)
  - encode each message (separately) as $C(m) \in \{0,1\}^n$ (n = *block length*, $n \geq k$)
  - ⇝ can analyze everything block-wise

Block codes

- **model:**
  - want to send message $S \in \{0,1\}^\star$ (bitstream) across a *(communication) channel*
  - any bit transmitted through the channel might ***flip*** $(0 \to 1$ resp. $1 \to 0)$
    no other errors occur (no bits lost, duplicated, inserted, etc.)
  - instead of S, we send *encoded bitstream* $C \in \{0,1\}^\star$
    sender *encodes* S to C, receiver *decodes* C to S (hopefully)
  - ⇝ what errors can be detected and/or corrected?
- all codes discussed here are ***block codes***
  - divide S into *messages* $m \in \{0,1\}^k$ of k bits each (k = *message length*)
  - encode each message (separately) as $C(m) \in \{0,1\}^n$ (n = *block length*, $n \geq k$)
  - ⇝ can analyze everything block-wise
- between 0 and n bits might be flipped
  - invalid code
  - how many flipped bits can we definitely **detect**?
  - how many flipped bits can we **correct** without retransmit?
  - i. e. decoding m still possible

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{\texttt{0},\texttt{1}\}^k \longrightarrow \{\texttt{0},\texttt{1}\}^n$

Code distance

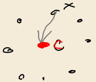
- each block code is an *injective* function $C : \{0,1\}^k \to \{0,1\}^n$

  $m \neq m' \implies C(m) \neq C(m')$

- define $\mathcal{C}$ = set of all codewords = $C(\{0,1\}^k)$ $= \{ b \in \{0,1\}^n : \exists\, m \in \{0,1\}^k : b = C(m) \}$

$\rightsquigarrow$ $\mathcal{C} \subseteq \{0,1\}^n$ $\boxed{|\mathcal{C}| = 2^k \text{ out of } 2^n\ n\text{-bit strings are valid codewords}}$

- <u>decoding</u> = finding closest valid codeword



c

$x \in \{0,1\}^n \setminus \mathcal{C}$

$c \in \mathcal{C}$

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{\mathtt{0}, \mathtt{1}\}^k \to \{\mathtt{0}, \mathtt{1}\}^n$
- define $\mathcal{C}$ = set of all codewords = $C(\{\mathtt{0}, \mathtt{1}\}^k)$

$\rightsquigarrow$ $\mathcal{C} \subseteq \{\mathtt{0}, \mathtt{1}\}^n$ $\boxed{|\mathcal{C}| = 2^k \text{ out of } 2^n\ n\text{-bit strings are valid codewords}}$

- decoding = finding closest valid codeword

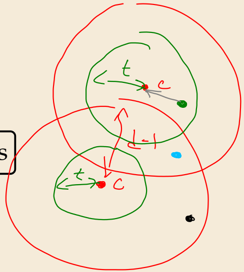
- ***distance of code:***
d = minimal Hamming distance of any two codewords = $\min_{x,y \in \mathcal{C}} d_H(x, y)$

Code distance

$m \neq m' \implies C(m) \neq C(m')$

- each block code is an *injective* function $C : \{0,1\}^k \to \{0,1\}^n$
- define $\mathcal{C}$ = set of all codewords = $C(\{0,1\}^k)$

$\rightsquigarrow$ $\mathcal{C} \subseteq \{0,1\}^n$ $\boxed{|\mathcal{C}| = 2^k \text{ out of } 2^n\ n\text{-bit strings are valid codewords}}$



- decoding = finding closest valid codeword

- ***distance of code:***
  d = minimal Hamming distance of any two codewords = $\min\limits_{x,y \in \mathcal{C}} d_H(x,y)$

Implications for codes

1. Need distance d to **detect** all errors flipping up to $d-1$ bits.
2. Need distance d to **correct** all errors flipping up to $\left\lfloor \frac{d-1}{2} \right\rfloor \overset{\leq t}{}$ bits.

⟹ for detecting 1 bit errors ⤳ need distance 2

for correcting 1 bit errors ⤳ need distance 3

Lower Bounds

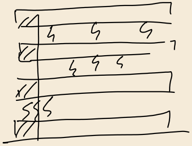
- Main advantage of concept of code distance: can *prove* lower bounds on block length

Lower Bounds

- Main advantage of concept of code distance:
  can *prove* lower bounds on block length

$k = 100$
$d = 2$
$n \geq 101$

- ***Singleton bound:*** $2^k \leq 2^{n-(d-1)} \rightsquigarrow \boxed{n \geq k + d - 1}$
  - *proof sketch:* We have 2^k codeswords with distance d
    after deleting the first $d - 1$ bits, all are still distinct
    but there are only $2^{n-(d-1)}$ such shorter bitstrings.

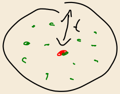


$\mathcal{C}$

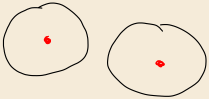
Lower Bounds

- Main advantage of concept of code distance:
  can *prove* lower bounds on block length

- ***Singleton bound:*** $2^k \le 2^{n-(d-1)} \quad \rightsquigarrow \quad n \ge k + d - 1$
  - *proof sketch:* We have 2^k codeswords with distance d
    after deleting the first $d-1$ bits, all are still distinct
    but there are only $2^{n-(d-1)}$ such shorter bitstrings.



- ***Hamming bound:*** $2^k \le \dfrac{2^n}{\sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}}$



  - *proof idea:* consider "balls" of bitstrings around codewords
    count bitstrings with Hamming-distance $\le t = \lfloor (d-1)/2 \rfloor$
    correcting t errors means all these balls are disjoint
    so $2^k \cdot \text{ball size} \le 2^n$

$\rightsquigarrow$ We will come back to these.

8.3 Hamming Codes

Parity Bit

- simplest possible error-detecting code: add a **parity bit**

`0 1 1 0 1 1 1 1 0 | 0`

$$= \begin{cases} 0 & \text{if number of ones is even} \\ 1 & \text{if number of ones is odd} \end{cases}$$

$$0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

XOR

Parity Bit

- simplest possible error-detecting code: add a **parity bit**

`0 1 1 0 1 1 1 1 0 | 0`

$$0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

XOR

$$= \begin{cases} 0 & \text{if number of ones is even} \\ 1 & \text{if number of ones is odd} \end{cases}$$

⇝ code distance 2

- can detect any single-bit error (actually, any odd number of flipped bits)
- used in many hardware (communication) protocols
  - PCI buses, serial buses
  - caches
  - early forms of main memory

Parity Bit

- simplest possible error-detecting code: add a **parity bit**

`0 1 1 0 1 1 1 1 0 | 0`

$$= \begin{cases} 0 & \text{if number of ones is even} \\ 1 & \text{if number of ones is odd} \end{cases}$$

$$0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

XOR

⇝ code distance 2

- can detect any single-bit error (actually, any odd number of flipped bits)
- used in many hardware (communication) protocols
  - PCI buses, serial buses
  - caches
  - early forms of main memory

👍 very simple and cheap

👎 cannot correct any errors

Clicker Question

What do you think, how many extra bits do we need to **detect** a **single bit error** in a message of 100 bits?

`sli.do/comp526`

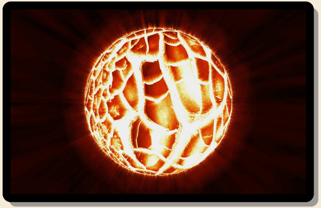
Click on “Polls” tab

Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!

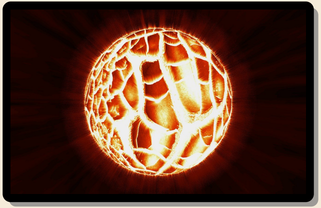
    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2

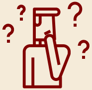


Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    `https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2`

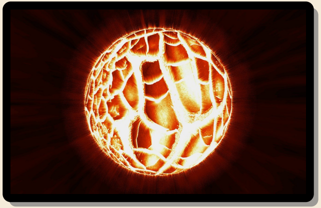




Can we **correct** a bit error without knowing where it occurred? How?

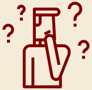
Error-correcting codes

any downtime is expensive!



- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!

    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2



Can we **correct** a bit error without knowing where it occurred? How?

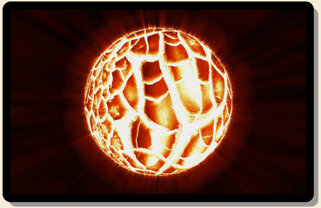
- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win

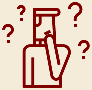
Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!

    https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2





Can we **correct** a bit error without knowing where it occurred? How?

- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win

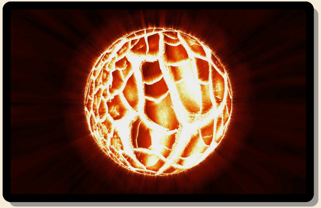
 *triples* the cost!

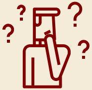
You want WHAT!?!

Error-correcting codes

any downtime is expensive!

- typical application: heavy-duty server RAM
  - bits can randomly flip (e. g., by cosmic rays)
  - individually very unlikely,
    but in always-on server with lots of RAM, it happens!
    `https://blogs.oracle.com/linux/attack-of-the-cosmic-rays-v2`





Can we **correct** a bit error without knowing where it occurred? How?

- Yes! store every bit *three times!*
  - upon read, do majority vote
  - if only one bit flipped, the other two (correct) will still win

triples the cost!

You want WHAT!?!

instead of 200% (!)

Can do it with 11% extra memory!