8 Error-Correcting Codes

28 April 2021

Sebastian Wild

8 Error-Correcting Codes

8.1 Introduction

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
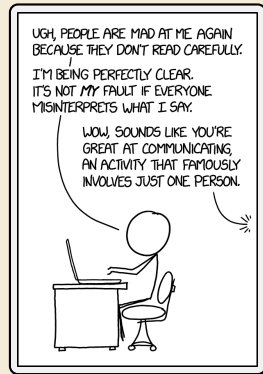   - ▶ ask to repeat if necessary

- ▶ But how is 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!*

   ⤳ 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)

UGH, PEOPLE ARE MAD AT ME AGAIN
BECAUSE THEY DON'T READ CAREFULLY.

I'M BEING PERFECTLY CLEAR.
IT'S NOT *MY* FAULT IF EVERYONE
MISINTERPRETS WHAT I SAY.

WOW, SOUNDS LIKE YOU'RE
GREAT AT COMMUNICATING,
AN ACTIVITY THAT FAMOUSLY
INVOLVES JUST ONE PERSON.

Noisy Channels

- ▶ computers: copper cables &
  electromagnetic interference
- ▶ transmit a binary string
- ▶ 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!

8.2 Lower Bounds

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

3

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 = $C(\{0,1\}^k)$

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

▶ 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** errors flipping up to $d-1$ bits

2. need distance d to **correct** errors flipping up to $\left\lfloor \frac{d-1}{2} \right\rfloor$ bits
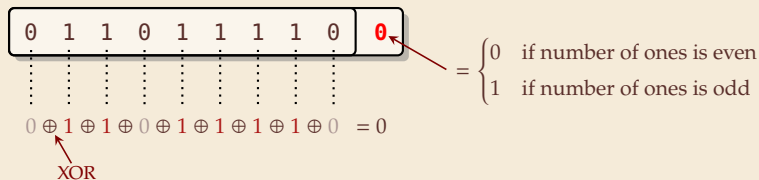

4

Lower Bounds

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

▶ *Singleton bound:* $2^k \leq 2^{n-(d-1)} \rightsquigarrow n \geq k + d - 1$

  ▶ *proof sketch:* We have 2^k codewords 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:* $2^k \leq \dfrac{2^n}{\sum_{f=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{f}}$

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

$\rightsquigarrow$ We will come back to these.

8.3 Hamming Codes

Parity Bit

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

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

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

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
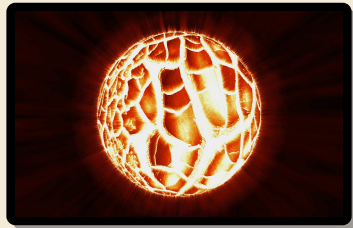   ▶ early forms of main memory

👍 very simple and cheap

👎 cannot correct any errors

6

Error-correcting codes

- ▶ 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
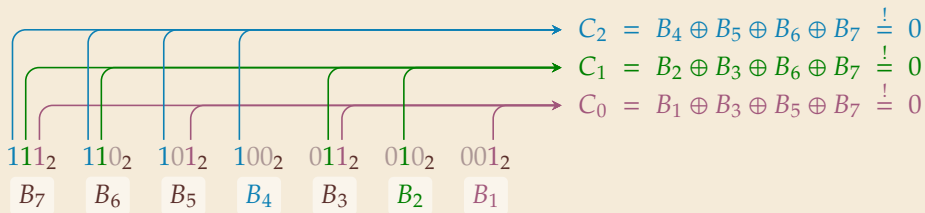 - 👎 *triples* the cost!

You want WHAT!?!

instead of 200% (!)

Can do it with 11% extra memory!

How to locate errors?

- ▶ **Idea:** Use several parity bits
  - ▶ each covers a **subset** of bits
  - ▶ clever subsets $\rightsquigarrow$ violated/valid parity bit pattern narrows down error
  - ⚠ flipped bit can be one of the parity bits!

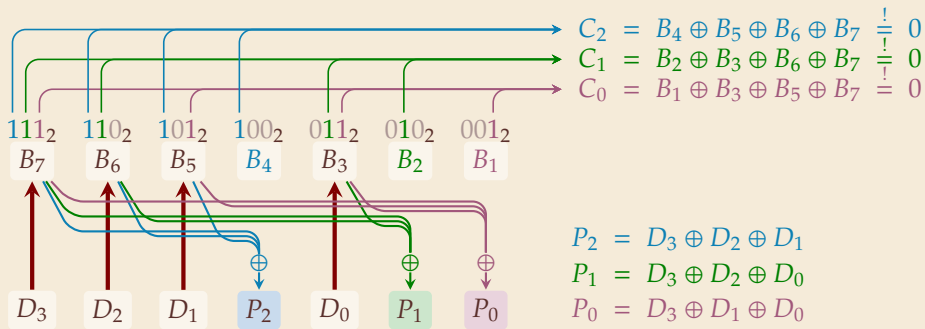- ▶ Consider $n = 7$ bits $B_1, \ldots, B_7$ with the following constraints:

$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \overset{!}{=} 0$$

111_2	110_2	101_2	100_2	011_2	010_2	001_2
B_7	B_6	B_5	B_4	B_3	B_2	B_1

Observe:
- ▶ No error (all 7 bits correct) $\rightsquigarrow$ $C = C_2 C_1 C_0 = 000_2 = 0$ ✓
- ▶ What happens if (exactly) 1 bit, say B_i flips?

$\boxed{C_j = 1 \text{ iff } j\text{th bit in binary representation of } i \text{ is } 1}$ $\rightsquigarrow$ *C encodes **position of error!***

4+3 Hamming Code

▶ *How can we turn this into a code?*

$$C_2 = B_4 \oplus B_5 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_1 = B_2 \oplus B_3 \oplus B_6 \oplus B_7 \overset{!}{=} 0$$
$$C_0 = B_1 \oplus B_3 \oplus B_5 \oplus B_7 \overset{!}{=} 0$$

$$P_2 = D_3 \oplus D_2 \oplus D_1$$
$$P_1 = D_3 \oplus D_2 \oplus D_0$$
$$P_0 = D_3 \oplus D_1 \oplus D_0$$

▶ B_4, B_2 and B_1 occur only in one constraint each ⇝ **define** them based on rest!

▶ **4 + 3 *Hamming Code* – Encoding**

1. **Given:** message $D_3 D_2 D_1 D_0$ of length $k = 4$
2. copy $D_3 D_2 D_1 D_0$ to $B_7 B_6 B_5 B_3$
3. compute $P_2 P_1 P_0 = B_4 B_2 B_1$ so that $C = 0$
4. send $D_3 D_2 D_1 P_2 D_0 P_1 P_0$

4+3 Hamming Code – Decoding

▶ **4 + 3** *Hamming Code* **– Decoding**

1. **Given:** block $B_7 B_6 B_5 B_4 B_3 B_2 B_1$ of length $n = 7$
2. compute C (as above)
3. if $C = 0$ no (detectable) error occurred
   otherwise, flip B_C (the Cth bit was twisted)
4. return 4-bit message $B_7 B_6 B_5 B_3$

▶ **Properties**

▶ can *correct* any 1-bit error
▶ How about 2-bit errors?
   ▶ We can *detect* that *something* went wrong.
   ▶ **But:** above decoder mistakes it for a (different!) 1-bit error and "corrects" that

Hamming Codes – General recipe

- ▶ construction can be generalized:
  - ▶ Start with $n = 2^\ell - 1$ bits for $\ell \in \mathbb{N}$ (we had $\ell = 3$)
  - ▶ use the ℓ bits whose index is a power of 2 as parity bits
  - ▶ the other $n - \ell$ are data bits

- ▶ Choosing $\ell = 7$ we can encode entire word of memory (64 bit) with 11% overhead (using only 64 out of the 120 possible data bits)

👍 simple and efficient coding / decoding

👍 fairly space-efficient