Hamming Codes

Sam Ruggerio

Outline

Review

Hamming Codes

Hamming Bound

Section 1

Review

Error Correction and Detection

- Sending messages between parties might be subject to noise and errors.
- How do we:
  - Detect Errors?
  - Correct Errors?
- The goal is to send a message that minimizes the necessary redundancy to achieve both goals.

Error Correction and Detection

- Easy solution: send k bits for every bit. Use majority voting to determine true bit.
- For example, the message 1011:
  - I send: 11111000001111111111 (i.e. $k = 5$)
  - You receive 11011001100111111111
  - You determine the correct message by voting 1011
- Recall the *rate* of a code is $\frac{len(\text{original message})}{len(\text{code word})}$
- Good robustness, but our rate is 20% (4 extra bits for every 1 bit of information).

Section 2

Hamming Codes

Better Codes

- Can we do better?
- *Claim:* For 11 bits, we can correct 1 error, and detect 2 errors, using only 5 bits, to make "nice" 16 bit blocks
  - $11/16 = 68.75\%$ rate!

Block Messages

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

- Consider a 16-bit block

Block Messages

0	0	0	0
0	1	1	0
0	1	0	0
1	0	1	1

- Consider a 16-bit block
- We want to fill it with the message 01101001011

Block Messages

0	0	0	0
0	1	1	0
0	1	0	0
1	0	1	1

- Consider a 16-bit block
- We want to fill it with the message 01101001011
- We'll use the remaining 5 bits to mark parity of certain regions of the block.

Hamming Codes

0	1	0	0
0	1	1	0
0	1	0	0
1	0	1	1

- We use the parity bit to keep the marked region even parity (ignoring parity bits themselves).

Hamming Codes

0	1	1	0
0	1	1	0
0	1	0	0
1	0	1	1

- We use the parity bit to keep the marked region even parity (ignoring parity bits themselves).

Hamming Codes

0	1	1	0
1	1	1	0
0	1	0	0
1	0	1	1

- We use the parity bit to keep the marked region even parity (ignoring parity bits themselves).

Hamming Codes

0	1	1	0
1	1	1	0
0	1	0	0
1	0	1	1

- We use the parity bit to keep the marked region even parity (ignoring parity bits themselves).

Hamming Codes

0	1	1	0
1	1	1	0
0	1	0	0
1	0	1	1

- What about the bit in the zero position?
- We'll set it to keep the parity of the entire table.
- This allows us to detect a second error, should one exist.

Hamming Codes

- We can detect the error by checking each parity bit, and fix it!

0	1	1	0
1	1	0	0
0	1	0	0
1	0	1	1

0	1	1	0
1	1	0	0
0	1	0	0
1	0	1	1

0	1	1	0
1	1	0	0
0	1	0	0
1	0	1	1

0	1	1	0
1	1	0	0
0	1	0	0
1	0	1	1

Questions?

Making More Codes

- What we described was a (15,11) Extended Hamming Code. (11 bits of message, 4 bit EC parity, 1 bit detection)
- You can easily make any $(2^n - 1, 2^n - (n+1))$ Hamming Code the same way.
  - Place each parity bit in the row/column corresponding to powers of 2
- In a 2^n block, we can efficiently detect 2 errors, and correct 1 error.

In Practice

- You can implement hamming code processing in hardware, or in software
- When sending info, errors tend to happen in *bursts*.
- Interleave blocks to spread out the errors that could happen.

Section 3

Hamming Bound

Hamming Bound

- Exactly how efficient can we get with error correcting codes?
- We have a lower bound called the Hamming Bound
- It gives us the efficiency of how well any error correcting code can utilize the space within the entire code word.
- Codes that achieve this bound are called Perfect Codes

Hamming Bound

Let $\Sigma = \{0, 1\}$. Let $A_\Sigma(n, d)$ be the maximum possible size of a block code C of length n, with minimum hamming distance d between elements of the block code.

Then,

$$A_\Sigma(n, d) \leq \frac{|\Sigma|^n}{\sum_{k=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{k} (|\Sigma| - 1)^k}$$

(Recall that hamming distance is the number of flips you need to reach another string)

Hamming Bound

- We're effectively that over all strings of length n ($|\Sigma|^n$),
- If we can make at most $t = \lfloor (d-1)/2 \rfloor$ errors,
- Then our code C covers the sum over all possible errors up to $k \leq t$, choosing k bits to flip to some other bit.

Hamming Bound

Let $\Sigma = \{0, 1\}$. Let $A_\Sigma(n, d)$ be the maximum possible size of a block code C of length n, with minimum hamming distance d between elements of the block code.

Then,

$$A_2(n, d) \leq \frac{2^n}{\sum_{k=0}^{\lfloor (d-1)/2 \rfloor} \binom{n}{k}}$$

(Recall that hamming distance is the number of flips you need to reach another string)

What does $A_\Sigma(n, d)$ *mean?*

- We have a q-ary language ($q = |\Sigma|$), of which we consider some string of length n
- *How many strings can we make from at most d changes?*

Computing $A_\Sigma(n, d)$ for Hamming Codes

- Our basic (15,11) Hamming code (no 0 bit), requires *three* flips to go from one valid code to another
- While we can detect errors of 2 flips or less, once we go above 2, distinguishing between one valid message to another is impossible.
- This means that the number of strings we can represent with our (15,11) Hamming code, with a minimum hamming distance of 3 is $2^{11} = 2048$

Computing $A_\Sigma(n, d)$ for Hamming Codes

- So let's compute the Hamming bound for $A_\Sigma(15, 3)$

$$\frac{2^{15}}{\sum_{k=0}^{1}\binom{15}{k}(2-1)^k} = \frac{2^{15}}{16} = 2048$$

- So our (15,11) Hamming code matches our hamming bound, thus it is considered a Perfect Code

Practical, not Perfect

- Our *extended* Hamming code includes that 0 bit to detect *one more error.*
- This means that in a 16 bit code word, we have a minimum hamming distance of 4
- $A_\Sigma(16, 4) \leq 3855.06$, but we still can only represent 2048 messages.
- Even though its not *perfect*, this is the mechanism still used in error correction within RAM on your computers. Nice powers of 2 are easy to send around.

The purpose of computing is insight, not numbers.

— Richard Hamming, PhD UIUC (1960s)