CS 229r Essential Coding Theory, Lecture 1 Jan 24, 2017

Hamming Codes, Distance, Examples, Limits, and Algorithms

Instructor: Madhu Sudan *Scribes: Aloni Cohen*

1 Course Information

This course will explore the topic of error-correcting codes. Broadly, the material will be approached from the three interconnected directions of (1) constructing codes and understanding their properties, (2) proving theoretical limits on the space of possible codes, and (3) devising efficient encoding and decoding algorithms. Additionally, applications of coding theory to seemingly unrelated problems will be presented in lecture and through exercises.

The course webpage includes a detailed syllabus, description of course policies, and links to additional resources including the text book, Piazza and Canvas sites, previous incarnations of the course, and contact information: `http://madhu.seas.harvard.edu/courses/Spring2017/`.

2 The Problem

Today's lecture is based on a paper of Hamming [Ham50]. Roughly, we want to figure out how to store digital bits (0s and 1s) on an magnetic storage device with data divided into blocks of 63 bits (the choice of 63 will be convenient for us, but we will consider more general block lengths at the end of the lecture). Due to engineering limitations, the data may become corrupted, with a 0 flipping to a 1 or vice-versa. For simplicity let us assume that at most a single bit of data is corrupted in each 63 bit block.

Addressing the broad questions of the course : (1) How can we store a message perfectly reliably on such a medium? (2) What is the maximum amount of information (i.e., the length of the message) that we can store per block of memory? (3) How can we efficiently detect and correct errors? The process by which a message is converted into a (hopefully) error-correctable form is called *encoding*, and the (uncorrupted) result of encoding a message is called a *codeword*.

3 A Naive Solution: Repetition

Our first approach is to repeat the data 3 times. In this scheme, a 63 bit codeword can store a 21 bit message. To decode a bit of the message from the (possibly corrupted) received word, compute the majority of the corresponding bits in the received word. For example:

Message: x	1	1	0	...
Encoding: repeat thrice		↓		
Codeword: y	111	111	000	...
Corruption		↓		
Received word: z	111	1**0**1	000	...
Decoding: take majority		↓		
	1	1	0	...

To see why this decoding algorithm is correct, observe that a 0 bit of the message may be transformed (by encoding and subsequent corruption) into one of the following 4 possibilities: 000, 001, 010, 100. Similarly,

a 1 may become one of: 111, 110, 101, 011. In each of these 8 possibilities, computing the majority of the received bits correctly recovers the message bit. (A perhaps more satisfying argument for this scheme's correctness stems from the fact that every two distinct codewords differ on at least 3 bits, as we will see later.)

Observe that this repetition code is:

- correct against arbitrary single-bit errors;
- simple, both conceptually and in terms of the concrete efficiency of encoding and decoding; and
- has a *rate* of 1/3, namely encodes 21 message bits into 63 codeword bits.

Observe also that while this code can correct some many-bit error patterns (e.g., a single error in each 3 bit sub-block), it cannot correct arbitrary 2-bit errors.

4 Hamming's Code: version 1

Is it possible to improve upon the rate of the repetition code? We will now analyze a more complex scheme which we will generalize later in the lecture. The previous encoded each bit of the message into 3 bits of the codeword; this scheme will encode 4 message bits into 7 codeword bits. The resulting 63 bit codeword will contain 9 such 7-bit sub-blocks, encoding a total of 36 message bits and yielding a rate of $36/63 = 4/7$. In this and all subsequent sections, all operations are done over $\mathbb{F}_2$, the finite field of 2 elements. Messages will be denoted by $x \in \mathbb{F}_2^4$, codewords by $y \in \mathbb{F}_2^7$, and (possibly corrupted) received words by $z \in \mathbb{F}_2^7$.

Define the following matrix $G \in \mathbb{F}_2^{4\times 7}$, called the *generator* matrix of the code:

$$G = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1}$$

The encoding of a message x is $y = xG$.

4.1 Correcting 1 bit errors

The crucial property we will use for error-correction is that the *distance of the code* is 3. That is:

Claim 1. *For any two distinct messages x and x', $y = xG$ and $y' = x'G$ differ on at least 3 coordinates.*

This coordinate-wise notion of distance is called the *Hamming distance*, and will be denoted $\Delta(y, y')$. Δ is a metric over $\mathbb{F}_2^7$, and in particular satisfies the Triangle Inequality.

Before proving Claim 1, let's see why it suffices for error-correction. Let y, and let z be any possible received message differing from y by at most a single bit. Then for every codeword $y' \neq y$, $\Delta(z, y') \geq 2$ (and therefore z can uniquely be decoded to y).

This follows from the Triangle Inequality. Suppose for contradiction that $\Delta(z, y') \leq 1$:

$$3 \leq \Delta(y, y') \leq \Delta(y, z) + \Delta(z, y') = 1 + \Delta(z, y') \leq 2. \tag{2}$$

This argument may be conceptualized by imagining the space $\mathbb{F}_2^7$ of all possible received words. Each codeword in this space is at the center of a ball of radius 1 that contains the received words that may result from corrupting this codeword. Because the distance of the code (i.e., the distance between two centers) is 3, these balls are non-overlapping.

4.2 The distance of the code

We now prove Claim 1. Let $H \in \mathbb{F}_2^{7\times 3}$ be the following matrix (called the *parity-check matrix*):

$$H = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{3}$$

Two properties of H are important for us:

- each row of H is unique, and
- $GH = 0$.

Suppose for contradiction that there exist distinct messages x and x' and corresponding codewords $y = xG$ and $y' = x'G$ such that $\Delta(y, y') \leq 2$. This implies that $(x - x')G$ has at most 2 nonzero entries. Therefore, the vector $(x - x')GH = 0$ is the result of adding at most two rows of H together. But since the rows of H are distinct, such a sum can never be 0, yielding a contradiction.

4.3 The rate

As previously mentioned, the rate of this code is $4/7$. Observe that codewords $xG \in \mathbb{F}_2^7$ are in the kernel of the columns of H, which has dimension at most (in fact, exactly) 4.

5 Hamming's Code: version 2

In the preceding construction, error-correction was possible because the rows of H were unique. The length of codewords corresponded to the number of rows in H, and the length of messages corresponded to the difference between the number of rows and columns (that is, the dimension of the kernel of H_3). Let us now generalize this construction to make better use of the 63 bits available to us and achieve a better rate.

Let $H' \in \mathbb{F}_2^{63\times 6}$ be the matrix whose ith row contains the binary expansion of $i + 1$.

$$H' = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ & & \vdots & & & \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{4}$$

The code consists of all vectors in the kernel of H': $\{y \in \mathbb{F}_2^{63} : yH' = 0\}$. We could construct a generator matrix G' spanning $\ker(H')$, and the codeword corresponding to a message x would be $y = xG'$ as before.

How many message bits can be encoded with this code? Stated another way, what is the dimension of $\ker(H')$?

Exercise *Show that* $\dim(\ker(H')) = 57$.

5.1 The Packing Bound

The rate of this code is $57/61 = 19/21$, a significant improvment over $4/7$. Is this the best rate possible with 63-bit codewords? Hamming proved that indeed it is. The Hamming constructed above contains 2^{57} codewords, each 63 bits long. Hamming shows that it is in fact impossible to have a code containing even a single additional codeword which can also correct every 1 bit error.

Each codeword may be corrupted in any single bit location or none at all, yielding 64 possible received words from each codeword. In order to correct all errors, these sets of $(64 \cdot \#\text{codewords})$ words must not intersect; otherwise there would exist an ambiguity.

$$\begin{aligned} 64 \cdot \#\text{ codewords} &\leq \#\text{ 64-bit words} = 2^{64} \\ \implies \#\text{ codewords} &\leq 2^{57} \end{aligned}$$

Every word is a codeword or within distance 1 of a codeword.

6 Generalized Hamming Code

We now generalize the Hamming code to support different message and codeword lengths. Let $n = 2^\ell - 1$ be the desired codeword length. Construct the parity-check matrix $H_\ell \in \mathbb{F}_2^{n \times \ell}$ as before, with the ith row containing the binary representation of $i + 1$. As before, the code consists of all vectors in $\ker(H_\ell)$, a space with dimension $n - \ell$. The same packing bound argument proves that this code achieves optimal rate among all codes correcting a single corruption, for n of this form. Encoding a message $x \in \mathbb{F}_2^{n-\ell}$ consists of computing xG_ℓ, where the generator matrix $G_\ell \in \mathbb{F}_2^{n-\ell \times n}$ consists of a basis for $\ker(H_\ell)$.

6.1 Efficient Decoding

It is easy to perform encoding, but given a received word z, how may we efficiently detect and correct an error? Of course, one could perform a brute-force search over all 2^{57} codewords, checking whether they differ from z at a single coordinate.

To improve over this approach, we first observe an important property of our code: $zH = 0$ if and only if z is an (uncorrupted) codeword. Let $e_i \in \mathbb{F}_2^{63}$ be the vector with a 1 in the ith coordinate and 0s elsewhere. A faster method to find the error is as follows:

- If $zH_\ell = 0$, return ``no error''.
- For $i = 1, \ldots, n$:
  - If $(z + e_i)H_\ell = 0$, return `i`.

This method takes $O(n^2)$.

As they say, an ounce of mathematics is worth a pound of programming. Notice that for any codeword y,

$$(y + e_i)H_\ell = e_i H_\ell = i\text{th row of } H_\ell = i \text{ in binary.} \tag{5}$$

Thus, simply multiplying $z = y + e_i$ by H_ℓ enables error-correction in time $O(n)$.

References

[Ham50] Richard W Hamming. Error detecting and error correcting codes. *Bell Labs Technical Journal*, 29(2):147–160, 1950.