Math 342. Notes for Lecture 1. Tuesday, January 4, 2018

Math 342 is a course in Algebra and Coding Theory.

Abstract algebra: theory developed for solving polynomial equations; turned out to have surprising applications to modern technology, such as error-correcting codes.

Coding: Will develop theory along with applications, emphasizing both algorithms and proofs.

Textbook: A First Course in Coding Theory, by Raymond Hill

Course website; http://www.math.ubc.ca/~reichst/342S18syll.html (or link through undergraduate tab of Math Dept website or my personal website)

All administrative information (including homework) will be posted there.

For this course, CODING THEORY = ALGEBRAIC ERROR-CORRECTION

CODING THEORY

– There other typse of codes (cryptographic codes, data compression codes) not covered in this course. Cryptography is covered in Math 312.
– There are two approaches to error-correcting codes, algebraic and probabilistc. We will focus on the algebraic approach.

Basic framework for communication over a noisy channel”

**SEND** → NOISY CHANNEL → **RECEIVER**

Message → Received Message

Information to be transmitted can be data, images, voice

Examples of Noisy Channels:
Communications (from here to there): telephone, cell Phone, TV, radio, satelitte communication (orbital and deep space), internet communication

Recording or storage (from now to then): Hard disk drives for computers, flashstick, CD, DVD, IPOD, digital dameras, bar codes for product identification (read by scanners), face recognition

Sources of noise: electronics noise, mechanical imperfections, human error, atmospheric disturbances, signal power loss over large distances, interference from structures or cosmic rays, etc.

Errors are inevitable!

Simple Mathematical Channel Model: *Binary Symmetric Channel (BSC)* and diagram:

— binary inputs 0,1
— binary outputs 0,1

— a “channel error” means that an input bit is flipped (because of noise), i.e., a transmitted 0 is received as a 1 or a transmitted 1 is received as a 0.

— error (i.e., flipped bit) with channel error probability or crossover probability $p$: think of $p$ as small, say $p \approx 10^{-4}$.

Q: How to detect or correct errors?

A: Add redundancy (extra digits) to each message in order to detect or correct (a limited number of) errors.

Example: 2-repetition code

Encoding:

0 → 00
1 → 11
Decoding:
00 → 0
11 → 1
01 or 10 → Declare an error and request re-transmission.

Performance, within each 2-bit codeword:
– If neither bit is in error, then transmission is fine.
– If only one of the two bits is in error, then error is detected.
– If both bits are in error, then the decoder mis-corrects.

We say that the code is 1-error-detecting.

Trade-off: Message Transmission rate is cut in half: it takes two coded bits to represent one message bit.

Other problems with error detection:
– Need a reliable “return channel” to request re-transmission
– Sometimes transmitted information is gone immediately after transmission
– Re-transmission causes additional delay in reception of information

It would be much better if we can actually correct errors rather than merely detect errors, in a manner that it transparent to the receiver. We can!

– Example: 3-repetition code:
Encoding:
0 → 000
1 → 111

Decoding scheme (majority vote):
000 or 100 or 010 or 001 $\rightarrow$ 0
111 or 011 or 101 or 110 $\rightarrow$ 1

Performance within each 3-bit codeword:
- If no errors are made, transmission is fine.
- If only 1 error is made, then error is corrected.
- If 2 or 3 errors are made, then the decoder will mis-correct.

We say that the code is 1-error-correcting.
In contrast, the 2-repetition code cannot correct any errors.

Trade-off: Message transmission rate is cut by a factor of three: it takes three coded bits to represent one message bit.

Let us calculate the probability of error, using a 3-repetition code.
- The probability of error in each position is $p$. Assume channel errors are made independently. By symmetry may assume that the word 000 is sent.

— This message is decoded correctly (i.e., 000 is understood to be 000) if the received word is 000, 100, 010, or 001.

000 is received with probability $(1 - p)^3$,
100 is received with probability $p(1 - p)^2$,
010 is received with probability $p(1 - p)^2$,
001 is received with probability $p(1 - p)^2$.

— The message is decoded incorrectly (i.e., 000 is understood to be 111) if the received word is

011, the probability of this happening is $p^2(1 - p)$ or
101, with probability $p^2(1 - p)$ or
110, with probability $p^2(1 - p)$ or
111 is received with probability $p^3$.

— The total probability of an error is thus

$$3p^2(1 - p) + p^3 = 3p^2 - 2p^3.$$

If $p$ is small, this number is much smaller than $p$. For example, if $p = 10^{-4}$, then $3 \cdot 10^{-8} - 2 \cdot 10^{-12} < 10^{-7}$.

We have thus achieved reduction in error probability by several orders of magnitude at the cost of slowing the message transmission rate by a factor of 3.

Is the trade-off between error correction and message transmission rate worthwhile?

The answer depends on the circumstances. In some situations, one can cope with a small number of message errors (e.g., images), and transmission speed becomes a predominant factor. In other situations, (e.g., high definition video or audio), one wants to use a strong error-correcting code (correcting more than one error), at a slower rate of transmission.

In summary, when we design an error-correcting code, we want:

(1) well-separated codewords, so that they are hard to confuse with each other

(2) On the other hand, we want to transmit lots of distinct messages by distinct codewords and so there should be lots of codewords.

These are the two main objectives of a “good” code. They are in conflict.

Notation: for a finite set $S$, $|S|$ denotes the number of elements in $S$. 
Definitions:

*Code alphabet:* Any finite set of \( q \geq 2 \) elements, written \( F_q = \{a_1, a_2, \ldots, a_q\} \). Usually, \( F_q = \{0, 1, \ldots, q - 1\} \).

- Main example: \( F_2 = \mathbb{Z}_2 = \{0, 1\}, q = 2 \) (the binary case)

*\( q \)-ary word of length \( n \) over \( F_q \):* a sequence (string) \( \bar{x} = x_1x_2 \ldots x_n \), where each \( x_i \in F_q \).

A *\( q \)-ary code* is a nonempty set \( C \) of \( q \)-ary words all of the same length \( n \).

- \( q = 2 \): We say “binary” instead of 2-ary, ternary instead of 3-ary.

*Codeword:* an element of a code \( C \).

In summary: A CODE IS A SET OF WORDS, CALLED CODE-WORDS.

The *length* of a code \( C \) is \( n \): the (common) length of the codewords in \( C \).

The *size* of a code \( C \), denoted \( M = |C| \): the number of codewords in \( C \).

An *\( (n, M) \)-code:* a code of length \( n \) and size \( M \)

Examples: \( q = 2 \):

- 2-repetition code \( (n, M) = (2, 2) \)
- \( \{aa, bb\} \) \( (n, M) = (2, 2) \)
- 3-repetition code \( (n, M) = (3, 2) \)
- \( n \)-repetition code \( (n, M) = (n, 2) \)

\[ C_1 = \{00, 01, 10, 11\}, \quad (n, M) = (2, 4) \]
\[ C_2 = \{000, 011, 101, 110\} \quad (n, M) = (3, 4) \]
\[ C_3 = \{00000, 01101, 10110, 11011\} \quad (n, M) = (5, 4) \]

\[ q = 3 \text{ (ternary)}: \]
\[ C_4 = \{000000, 111111, 222222\} \quad (n, M) = (6, 3) \]

We need a way to measure distance between words of the same length.

Definition: Hamming distance \( d(x, y) \) between words \( x = x_1 \ldots x_n, y = y_1 \ldots y_n \) of the same length, is the number of positions where \( x_i \) and \( y_i \) disagree:
\[
d(x, y) = |\{1 \leq i \leq n : x_i \neq y_i\}|.
\]
- Example: \( d(0111, 1001) = 3, d(12345, 14134) = 4; \)
- Note: magnitude of difference is irrelevant.

Properties of Hamming distance:
Proposition: \( d(x, y) \) is a metric, i.e.,

(0) \( d(x, y) \geq 0 \)
(1) \( d(x, y) = 0 \) iff \( x = y \)
(2) \( d(x, y) = d(y, x) \)
(3) \( d(x, z) \leq d(x, y) + d(y, z) \) (triangle inequality)

Proof: (0), (1) and (2) follow directly from the definition. See the book, p. 5 for the proof of (3).