6.02, Fall 2014 **George Verghese**

Lecture 3

So far:

1. Source, emitting symbols s_1 to s_N with probabilities p_1 to p_N

Entropy H as measure of uncertainty and expected information

Biggest H when symbols are equally likely, so $p_i = 1/N$, which gives $H = \log_2 N$

2. Binary code for transmission on a channel that can be in one of two states (1 and 0)

Symbol s_i mapped to codeword c_i of length l_i

Codewords associated with leaves of binary tree, distance l_i from root (why not internal nodes? → prefix-free code for instantaneous decodability)

Kraft (in)equality, quick proof by conservation of mass: start with unit mass at the root of the binary tree, push half down each bifurcation. You end up with $(1/2)^{(l_i)}$ as the mass at the end of the i-th leaf.

We want each binary digit of code to carry as much information as possible on average, so we want short code words, but constrained by $L \geq H$, because a binary digit cannot carry more than 1 bit of information.

This is **source coding**

3. Easy proofs of a couple of the above results (**optional**):

First show that $\ln(x) \leq x-1$, with equality if and only if (iff) x=1. Equality at x=1 is clear, then simply note that the slope of ln(x) is larger for x<1 and smaller for x>1.

Now let $\{p_i\}$ and $\{q_i\}$ for 1=1 to N be nonnegative integers that sum to 1 (i.e., probability distributions), and let H be the entropy of $\{p_i\}$. Then

$$\sum p_i \ln(q_i/p_i) \leq \sum p_i [(q_i /p_i)-1] = 0$$

so

$$\sum p_i \log_2(q_i/p_i) \leq 0$$

or

$$H \leq -\sum p_i \log\_2(q_i)$$

Now choosing $q_i = 1/N$ shows that $\mathbf{H \leq \log_2 N}$

and choosing $\log_2 q_i = l_i$ shows that $\mathbf{H \leq L}$

4. With the information now tightly packaged thanks to source coding, we are ready to ship the stream of binary digits to the receiver. (We tend now to use "bits" and "binary digits" interchangeably, because we will be saying less about binary information units, mostly just binary digits.)

Problem: communication on a physical channel typically requires conversion to analog signals, with degradation. Bits may occasionally arrive flipped from 1 to 0, or the other way.

Abstraction: Memoryless binary symmetric channel (BSC), with probability p of a bit flip.

So now information-laden bits get corrupted.

Ideas for doing better?

5. Repetition – but rate goes down

Parity bit – ok to detect single error: Draw picture of going from the square in a 2-binary digit code to the cube for this code with an even parity check. What has happened? Moved to a bigger space, to put more room between codewords.

We'd like to do a whole lot better than repetition or a single parity bit can do.

6. Notion of Hamming distance. By using the parity bit, we have increased the HD between codewords from 1 to 2. To compute HD algebraically for two (equal-length) bit strings, take the *entry-by-entry "GF(2)" sum* (ex-or) and determine the *weight* of the result.

7. Extending the parity idea: take k "information bits" captured in k binary digits after source coding, and systematically (re)introduce redundancy to embed in a space of n binary digits, i.e., go from codewords of length k after source coding to codewords of length n. So n-k "parity" bits. This is **channel coding**.

(Systematic re-introduction of redundancy in this way is better than trying to exploit the variable source-dependent redundancy, especially when there may be

multiple sources feeding to the same transmitter, or when source characteristics can change. That's an important reason for first doing source coding to compress whatever source is feeding the transmitter, then channel coding to expand.)

Rate of information transfer: **R=k/n** binary information units per binary digit

But have to pick these parity bits smartly.

e.g., putting in a second overall parity bit is a non-starter. Other ideas? Use parity checks on subsets of the information bits! We'll learn more about specific codes next week.

8. Shannon's insight and key result: It is possible to embed 2^k codewords in n-space (i.e., choose 2^k corners of the 2^n in the hypercube), where n > k, and to get a specified rate R (=k/n < 1) binary information units per binary digit transmitter, with *arbitrarily low* probability of error, *provided* n is large enough and R < C, where C is a number that can be computed for the specific channel, and is knows as its **capacity**.

Intuitively, if k = Rn, then there are $2^n/2^{Rn}$ or $2^{(1-R)n}$ words of length n per codeword, so the number of n-bit words surrounding each codeword increases exponentially with n. However, on the BSC, the number of bit flips on an n-bit codeword will be around np, i.e., will increase linearly in n, so more space is needed around each codeword (a neighborhood of HD around np) to prevent confusion with another codeword. Shannon's result is obtained by carefully working out this tradeoff and showing that you win as n → infinity, provided R < C.

*Shannon's very general definition of C is on the slides, intuitively interpreted, and there's also a plot of the specific C for the BSC. But this is **optional** reading.*

Penalty for the low error probability: long latency in decoding.

Current codes come close to realizing the Shannon limit with reasonable latency. Among the best are "low-density parity check" (LDPC) codes invented by Prof. Robert Gallager in our department in his MIT PhD thesis, but only recognized much later as exceptionally effective. Also a class known as turbo codes. We will see examples of simple but widely used codes in the next few lectures. For now, just a bit of terminology: A **linear code** is one in which the sum of any two codewords is again a codeword.

9. A figure of merit for a code: the *minimum HD between codewords*, denote by d, indicates how much room one has for error. So codes are often described as (n,k,d) codes: n bits long, with 2^k codewords chosen from 2^n possible positions, and minimum HD of d between codewords.

Suppose I want to reliably **detect the occurrence of up to t_D errors**, **AND** to reliably (detect and) **correct up to t_C ($\leq t_D$) errors**, *under the assumption that no more than t_D errors occur*.

A necessary and sufficient condition is

$$\mathbf{d \geq t_D + t_C + 1}$$

Say **d=3**:

O X X O

Can have $t_D = 2$ and $t_C = 0$ (up to double-error detection, but then no error correction possible):

→ →
O X X O

or $t_D = 1$ and $t_C = 1$ (can detect and correct single errors):

→ ←
O X X O

d=5:

O X X X X O

$t_D = 4$ and $t_C = 0$ (can detect up to 4 errors, but not correct errors)

→ → → →
O X X X X O

$t_D = 3$ and $t_C = 1$ (can detect up to 3 errors, and correct single errors)

→ → → ←
O X X X X O

$t_D = 2$ and $t_C = 2$ (can detect and simultaneously correct up to 2 errors)

→ → ← ←
O X X X X O

Again, if the actual number of errors exceeds t_D, all bets are off!