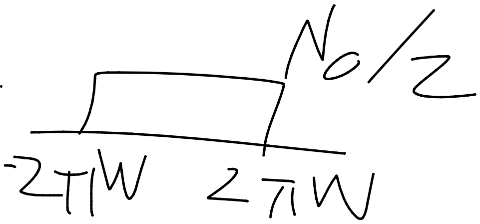
Tuesday, November 17, 2020 6:14 PM

$$C = \frac{1}{2}\log\left(1 + P/P_n\right)$$

usual case:



$P_N = W N_0$, so

$$C = \frac{1}{2}\log\left(1 + \frac{P}{WN_0}\right) = \frac{1}{2}\log(1 + SNR)$$

bits/use

uses/sec $= f_s = 2W$

$$C = W\log\left(1 + \frac{P}{N_0 W}\right) \text{ bits/sec}$$

$$\approx W\log(SNR) \text{ if } SNR \gg 1$$

<u>Ex</u> I send signal w/ BW

Ex I send signal w/ BW
W = 20 kHz, SNR 60 dB

What is max bitrate for reliable comm?

$60\,dB = 10\log_{10} X$

$X = 10^6$

$C = (20000)\log_2(1 + 10^6)$

$\approx 398.6\ kb/s$

$= 49.8\ kB/s$

---

consider

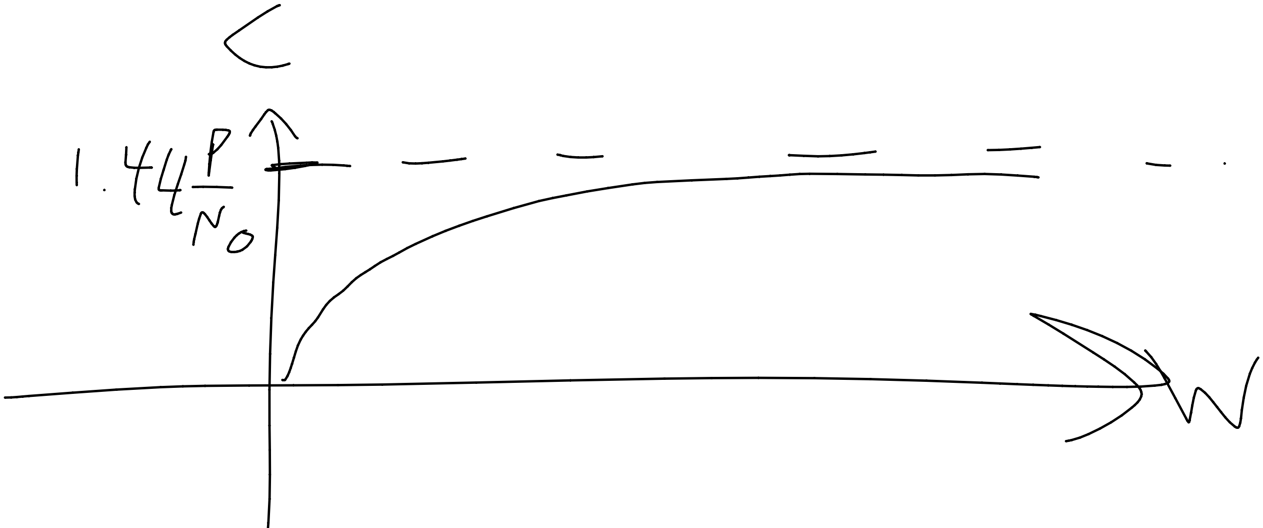
$$\lim_{W\to\infty} W\log_2\left(1+\frac{P}{N_0W}\right) = ?$$

$$\log_2\left(1+\frac{P}{N_0W}\right) = \frac{\ln(1+P/N_0W)}{\ln(2)}$$

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

$$\lim_{W\to\infty} C = \frac{1}{\ln 2}\lim_{W\to\infty} W\left(\frac{P}{N_0W} - \frac{P^2}{2N_0^2W^2} + \cdots\right)$$

$$= \frac{1}{\ln 2}\frac{P}{N_0} \approx 1.44\frac{P}{N_0}$$



to increase C, $\nearrow W$ only does "so much"

must increase $\boxed{P}/N_0$

Power/BW where you can't always compensate for low P w/ high W.

Say we transmit $R < C$

the spectral bitrate is

$$\boxed{r = R/W}$$

so $\quad r < \frac{C}{W} = \log\left(1 + \frac{P}{N_0 W}\right)$

R is # bits/sec, P is $\frac{\text{Energy}}{\text{sec}}$

$$\mathcal{E}_b = P/R > \frac{P}{C} > \frac{P}{Wr}$$

So $$\frac{\mathcal{E}_b}{N_0} r > \frac{P}{WN_0}$$

So $$\log\left(1+\frac{P}{WN_0}\right) < \log\left(1+\frac{\mathcal{E}_b r}{N_0}\right)$$

So $$r < \log\left(1 + r\frac{\mathcal{E}_b}{N_0}\right)$$

power eff.

$$\frac{2^r - 1}{r} \le \mathcal{E}_b$$

$$\frac{\ }{r} \; \cancel{\le} \; \frac{\mathcal{E}_b}{N_0}$$

BW-efficiency

if $\mathcal{E}_b/N_0 < \frac{2^r - 1}{r}$

$\rightarrow R > C \Rightarrow$ no reliable comm!

Information Transmission THM

To transmit a source U reliably over a channel w/ capacity C, must have

$$\boxed{H(U) < C}$$

Coding Theory

Tuesday, November 17, 2020 6:40 PM

trading ultimate efficiency for SAFETY

Linear Block Code

Def. An (n, k) block code is a collection of $M = 2^k$ binary sequences, each of length n, called codewords

A block code or "code book" is written

$$\mathcal{C} = \{c_1, c_2, \ldots, c_M\},$$ where c_i is a length n binary sequence

Ex. $\left\{\begin{matrix} 001, \\ 100, \\ 110, \\ 111 \end{matrix}\right\}$ 4 codewords of length 3, $M = 4 = 2^k$, $k = 2$

this is a (3, 2) block code

Def. the <u>rate</u> of a block code is $R := k/n$

for code above, rate is $2/3$

each codeword can encode at most 2 bits of data

$$\left\{\begin{matrix}001, \\ 100, \\ 110, \\ 111\end{matrix}\right\} \to \left\{\begin{matrix}00 \\ 01 \\ 10 \\ 11\end{matrix}\right\}$$

each codeword uses 3 bits to encode 2, $R<1$,

the larger R codes are more efficient

Def A block code is <u>linear</u> if the <u>sum</u> of any two codewords is itself a codeword.

<u>But</u> what is the sum of two binary sequences?

Define $c_1 \oplus c_2$ by elementwise modulo 2 addition

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

<u>Ex.</u>

$$\begin{array}{r} 11010100 \\ \oplus\ 01110110 \\ \hline 10100010 \end{array}$$

$\to \oplus$ <u>commutes</u>

a linear block code (LBC) is a block code whose codebook is <u>closed</u> under addition.

Ex $\mathcal{C} = \{1011, 0100, 1111, 0000\}$

an (n, k) block code $\rightarrow n = 4,\ k = 2$

$(4, 2)$ block code

is it linear? $c_1 \oplus c_2 = 1111 = c_3$

$c_1 \oplus c_3 = 0100 = c_2$

$c_1 \oplus c_4 = c_1$

$c_2 \oplus c_3 = 1011 = c_1,\ c_2 \oplus c_4 = c_2,\ c_3 \oplus c_4 = c_3$

so <u>yes</u> $\mathcal{C}$ is linear

Say each codeword maps to a k-bit sequence

$\mathcal{C} = \{1011, 0100, 1111, 0000\}$

$\updownarrow$

$\mathcal{I} = \{\underset{i_1}{10}, \underset{i_2}{01}, \underset{i_3}{11}, \underset{i_4}{00}\}$

then

$i_1 \oplus i_2 = i_3$

$c_1 \oplus c_2 = c_3$

can choose a nice mapping so that the <u>data</u> has the same <u>algebra</u> as the code

A code that isn't linear will **not** allow this feature

Generators – facilitate an efficient representation of LBCs

I have (n, k) LBC

$$\#\text{data words} = 2^k \quad \left\{\begin{array}{l} \underbrace{00\ldots0}_{} \\ \vdots \\ \underbrace{11\ldots1}_{k\text{-bits}} \end{array}\right. \begin{array}{c} \to \\ \\ \\ \end{array} \begin{array}{c} c_1 \\ \vdots \\ \underbrace{c_M}_{n\text{-bits}} \end{array} \qquad \mathcal{C} = \{c_1, \ldots, c_M\}$$

Define

$$e_1 = \overbrace{100\ldots0}^{k \text{ bits}} \to g_1 \in \mathcal{C}$$

$$e_2 = 010\ldots0 \to g_2$$

$$\vdots \qquad\qquad \vdots$$

$$e_k = 000\ldots01 \to g_k$$

any data sequence $X = X_1 X_2 \ldots X_k$ ← each X_i is a **bit**

can be written as $X = \sum_{i=1}^{k} X_i e_i$, where $1 \cdot v = v$
$0 \cdot v = 0$

ex. $\underset{X_1X_2X_3X_4X_5}{10101} = 1 \cdot 10000 + 0 \cdot 01000 + 1 \cdot 00100$
$+ 0 \cdot 00010 + 1 \cdot 00001$

Ex. $\underset{x_1 x_2 x_3 x_4 x_5}{10101} = 1 \cdot 10000 + 0 \cdot 01000 + 1 \cdot 00100 + 0 \cdot 00010 + 1 \cdot 00001$

$= x_1 e_1 + x_2 e_2 + x_3 e_3 + x_4 e_4 + x_5 e_5$

we have linearity, so we can have codewords follow same algebraic structure as data, so

if $x \to c$ then

$$c = \sum_{i=1}^{k} x_i g_i$$

although we have $2^k >> k$ codewords, they are spanned by k generators

Def. The generator matrix for an LBC (as described above) is given by

$$G = \begin{pmatrix} g_1 \\ g_2 \\ \vdots \\ g_k \end{pmatrix} = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1n} \\ g_{21} & g_{22} & \cdots & g_{2n} \\ g_{31} & g_{32} & \cdots & g_{3n} \\ \vdots & \vdots & & \vdots \\ g_{k1} & g_{k2} & \cdots & g_{kn} \end{pmatrix} \in M_{k \times n}(\{0,1\})$$

each g_i is a row vector, $(g_{i1}, g_{i2}, \ldots, g_{in})$

So we have arithmetic elementwise $(\oplus, \cdot)$ so we can define matrix multiplication as usual using these operations

c is a $1 \times n$ row vector, (codeword)
$x \to c$, x is a $1 \times k$ row vector

$$xG = (x_1 \ldots x_k) \begin{pmatrix} g_1 \\ \vdots \\ g_k \end{pmatrix} = \sum x_i g_i = c$$

i.e. $\boxed{x \to c \iff c = xG}$

$\longrightarrow$ code is totally determined by $\underline{G}$

Ex.

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \quad k=3, \ n=7$$

$2^3 = 8$ data points, 8 codewords $\to$ 3 elementary vectors, 3 generators

$e_1 = (100) \to g_1 = (100)G = 1001011$ (1st row of G)

$g_2 = (010)G = 0100101$
$g_3 = (001)G = 0011111$

What does 111 map to?

$111 = e_1 \oplus e_2 \oplus e_3$

$\hookrightarrow g_1 \oplus g_2 \oplus g_3 = 1110001$

<u>Def.</u> <u>Systematic Code</u> is one in which each codeword <u>begins</u> with the data it represents

i.e. $x = x_1 \dots x_k \rightarrow c = c_1 \dots c_n \Rightarrow c_1 \dots c_k = x_1 \dots x_k$

to ensure for (n, k) LBC:

$$G = (I_k \,\vdots\, P) \quad \text{for some } P \;\; k \times (n-k)$$

I_k: $k \times k$ identity

$$\rightarrow \quad xG = (xI_k, xP)$$

$$= (x, xP) = c, \text{ starts w/ } x$$

we call the last $n-k$ bits xP the "paritycheck" bits

If I know G and I receive $c_1 c_2 \dots c_n$

I would decode as $x = c_1 \dots c_k$

then check $xP = c_{k+1} \dots c_n$

if it <u>doesn't</u>, then an error has occurred

ideally, $xG = c$

ex.

$$G = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

I receive $\tilde{c} = 010\ 0111$

I know my code is systematic, so I assume that

$\tilde{x} = 010$

now check: $\tilde{x} G = 010\ 0101 \neq \tilde{c}$ so an error occurred!,

this codeword $\tilde{c} \notin \mathcal{C}$

otoh if I receive $0011111 = \tilde{c}$

$\tilde{x} = 001$, $\tilde{x} G = \tilde{c}$ so hopefully safe!

this is an error detection method

Def the parity check matrix $H = (P^T \,\vdots\, I_{n-k})$

H is an $(n-k) \times n$ matrix.

So for any codeword c, we have $x \to c$,

$$cH^T = xGH^T = (xI_k \,\vdots\, xP)\begin{pmatrix} P \\ \hdashline I_{n-k} \end{pmatrix}$$

$$= xI_kP + xPI_{n-k}$$

$$= \underbrace{xP}_{n\text{-vector}} + x\underbrace{P = \vec{0}}_{n\text{-vector}}$$

error detection

So if I receive m and $mH^T \neq \vec{0}$, then an error occurred as $cH^T = \vec{0}$ $\forall c \in \mathcal{C}$.

There exist errors that <u>can't</u> be detected

<u>ex.</u> $G = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$

$x = 100 \longrightarrow c = 1001101$

↓ channel, corrupts

<u>Error 1</u> $\underline{0}001101$ → not a codeword so <u>error detected</u>

<u>Error 2</u> $C \xrightarrow{\text{corrupt}} 10011\underline{1}1$ → not a codeword, error detected

<u>Error 3</u> $C \to \underline{0}0\underline{1}1\underline{0}\underline{1}\underline{0}$ → <u>is</u> a codeword & error is <u>not</u> det

required <u>6</u> bit errors!

a good code can <u>detect</u> <u>likely</u> errors

↳ few bit errors

<u>Special Case:</u> <u>Hamming Code</u> $n = 2^m - 1$, $K = 2^m - m - 1$

Special Case: Hamming Code $n = 2\ -1$, $k = 2\ -m-1$

for some $m \geq 3$.

P given by every length m sequence except the all-zero or a sequence containing a single 1

P is a $k \times (n-k)$ matrix, so it contains $k = 2^m - m - 1$ sequences of length $n-k = m$

2^m = # sequences of m bits

1 = # of all-zero sequences

m = # of seq. containing a single 1

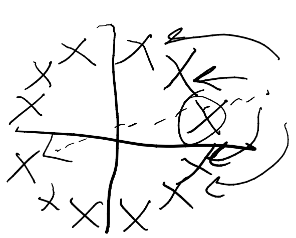
so this is possible ✓

Ex. $m=3 \rightarrow n=7,\ k=4$ (7,4) Hamming Code

$$P = \begin{pmatrix} 0&1&1 \\ 1&0&1 \\ 1&1&0 \\ 1&1&1 \end{pmatrix} \rightarrow G = \left(\begin{array}{cccc|ccc} 1&0&0&0&0&1&1 \\ 0&1&0&0&1&0&1 \\ 0&0&1&0&1&1&0 \\ 0&0&0&1&1&1&1 \end{array}\right)$$

$$\hookrightarrow H = \left(P^T \,\middle|\, I_{n-k}\right) = \left(\begin{array}{cccc|ccc} 0&1&1&1&1&0&0 \\ 1&0&1&1&0&1&0 \\ 1&1&0&1&0&0&1 \end{array}\right)$$

(there are $k!$ distinct Hamming Codes for any given m)



"nearness" of symbols

$\rightarrow$ how mistakable are symbols for another

Want to extend this idea to codewords

Def the Hamming Distance between two codewords c_i and c_j is the # of components in which they disagree.

denoted $d(c_i, c_j)$

Ex. $c_i = (1010)$, $c_j = (1101)$ differ at 2, 3, 4

so $d(c_i, c_j) = 3$

$c_i = (1010)$, $c_j = (1011) \Rightarrow d(c_i, c_j) = 1$

Def. The minimum distance of a code $\mathcal{C}$ is the min. Hamming distance between any two codewords in $\mathcal{C}$.

Denoted: d_{min}

→ quantifies the # bit errors required to mistake one codeword for another

Fact: $d_{min} = 3$ for any Hamming code

Def The hamming weight of a codeword is the ~~# of~~ 1s in the codeword (analogous to the "norm" of a vector) Deno $w(c)$

Def. The minimum weight of a code is the min. Hamming weight among

Def. The minimum weight of a code is the min. Hamming weight among nonzero codewords in $\mathcal{C}$, denoted w_{min}.

THM In a linear code, $d_{min} = w_{min}$.

Proof If c is a codeword in $\mathcal{C}$, $w(c) = d(c, \vec{0})$

If $c_i, c_j \in \mathcal{C}$, so too is $c_i \oplus c_j$ (linearity)

$d(c_i, c_j) = w(c_i \oplus c_j)$ as $c_i \oplus c_j$ is only 1 in positions where c_i and c_j disagree.

So — if there exists a codeword with weight w, then there exist two codewords with distance w, and vice versa.

$\rightarrow d_{min} = w_{min}$ //

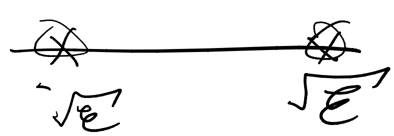
Soft-Decision Decoding

recall that all codes $\rightarrow$ constellation pts in some mod. scheme

$\rightarrow$ analog signals

The Euclidean distance between my symbols (either in the constellation or as L^2 analog signals) depends on the Hamming distance!

Ex. BPSK



I send a codeword $c_i = [c_{i1} c_{i2} \cdots c_{in}]$ as n transmissions of BPSK

I receive $c_j = (c_{j1}, \dots, c_{jn})$ corrupted by noise, guessed via some decision method

If $c_{ik} = c_{jk}$ → contributes 0 to the Hamming distance $d(c_i, c_j)$

else $c_{ik} \neq c_{jk}$ → contributes 1 to the Hamming distance

that happens if the symbol is read as being $2\sqrt{\mathcal{E}}$ away from the true signal, contributing $4\mathcal{E}$ to the Euclidean square distance

Hamming

$d_{c_{ij}}^{H}$ = # bits mistaken

$(d_{ij}^{E})^2 = 4\mathcal{E} \cdot$ # bits mistaken

$$\rightarrow \boxed{d_{ij}^{E} = 2\sqrt{\mathcal{E} d_{ij}^{H}} \text{ for BPSK}}$$

now for BPSK we had $P_{error}^{\text{index}} = Q\left(\frac{d^E}{\sqrt{2N_0}}\right)$

$$= Q\left(\sqrt{\frac{2d_{ij}^{H}\mathcal{E}}{N_0}}\right) \leq Q\left(\sqrt{\frac{2d_{min}\mathcal{E}}{N_0}}\right)$$

there are $M-1$ possible codebook errors

so by union bound:

$$P^{\text{codeword}} \leq (M-1)\, Q\left(\sqrt{2d_{min}\mathcal{E}} \right.$$

$$P_{error}^{codeword} \le (M-1)\, Q\left(\sqrt{\frac{2 d_{min} \mathcal{E}}{N_0}}\right)$$

$\mathcal{E}$ = Energy in a symbol, so $n\mathcal{E}$ is the energy in a codeword

→ k bits of data, so $n\mathcal{E} = k\mathcal{E}_b$

$$\rightarrow \mathcal{E}_b = \frac{n}{k}\mathcal{E} = \mathcal{E}/R$$

$$\Rightarrow P_{error}^{codeword} \le (M-1)\, Q\left(\sqrt{\frac{2\mathcal{E}_b R d_{min}}{N_0}}\right)$$

Soft-decision decoding — uses minimum-Euclidean distance codeword to decode via MF componentwise

→ doesn't take the $\mathcal{C}$ into account at all

Ex. using soft-decision, transmit 101
rx 111

but $111 \notin \mathcal{C}$
is possible

naive → yield errors

Hard-decision decoding uses first componentwise, then picks nearest codeword in $\mathcal{C}$ (in a Hamming sense)

↳ how do you do that??
(next lecture)

$\Rightarrow$ [illegible]

(next lecture)