L17 Coding: Linear Codes

17.1 Channel Coding Introduction

- channel coding introduces redundancy into your data
- thus the data or bit stream that you send in inter-related in either a block-by-block or sliding-window fashion
- thus channel coding introduces memory into the signalling process (what I get now is somewhat related to what I sent earlier)
- as a result of the redundancy the number of different sequences that are produced by the receiver are less than would otherwise be generated
- codes can be viewed as a mapping

MESSAGES (sequence of length, k) $\Rightarrow$ CODEWORDS (sequence of length, n)

$$k < n$$

$(n-k)$: redundant bits
parity bits
check bits

$$\text{REDUNDANCY} = \frac{(n-k)}{k} \quad \left(\frac{\#\text{ of redundant bits}}{\#\text{ of data bits}}\right)$$

portion of codeword that carries information

$$\text{CODE RATE} = \frac{k}{n} \quad \left(\frac{\#\text{ data bits}}{\text{total }\#\text{ of bits}}\right)$$

- we have 2^n possible codewords, but only 2^k are legal

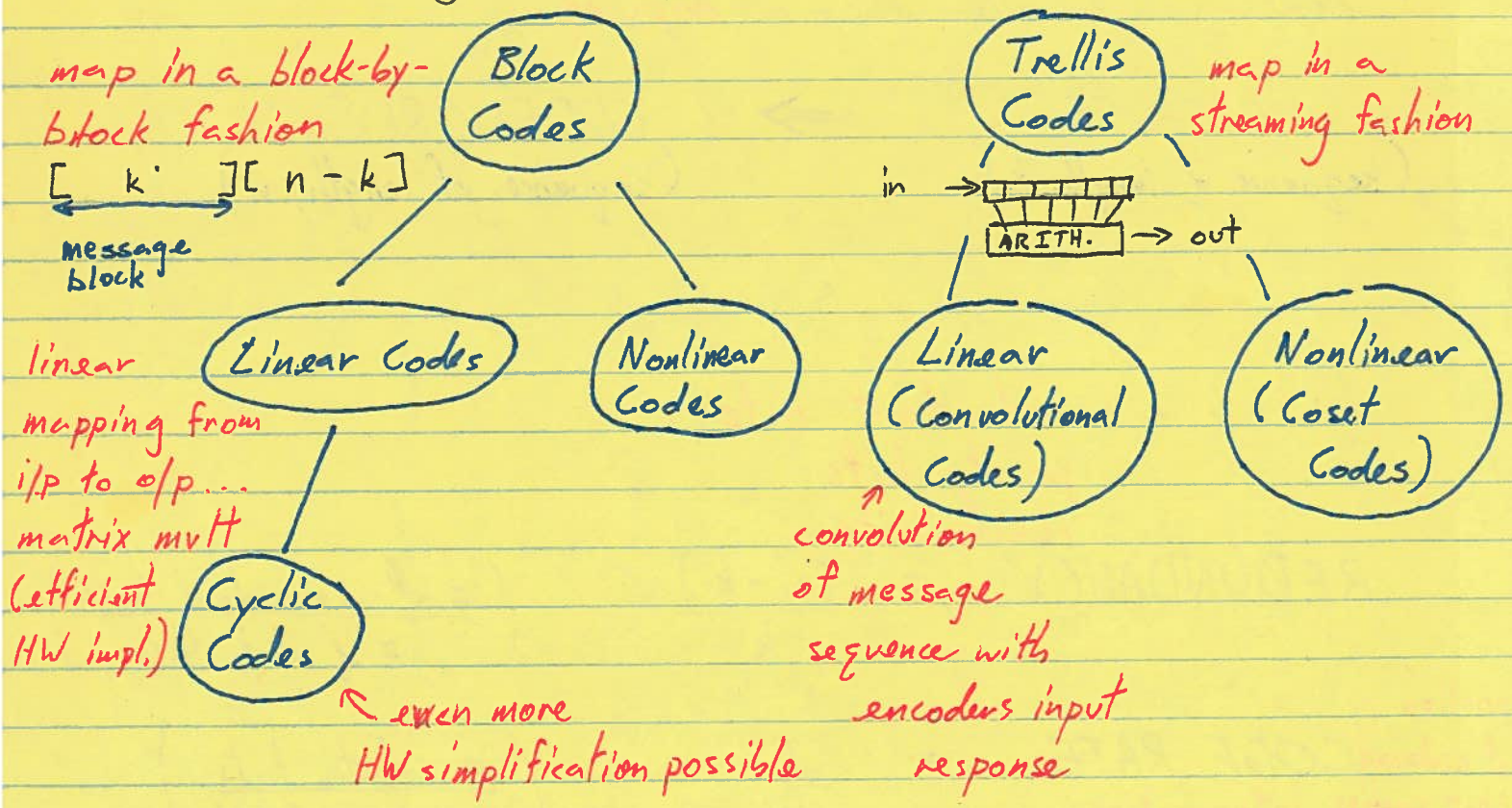
∴ a very small proportion $\frac{2^k}{2^n} = \frac{1}{2^{n-k}}$ of my messages will be in terms of legal codewords

- this gives the RX a chance to spot AND correct errors
- should be able to detect $(2^n - 2^k)$ error patterns

(2^n: total possible patterns; 2^k: undetectable patterns (look like legal codewords))

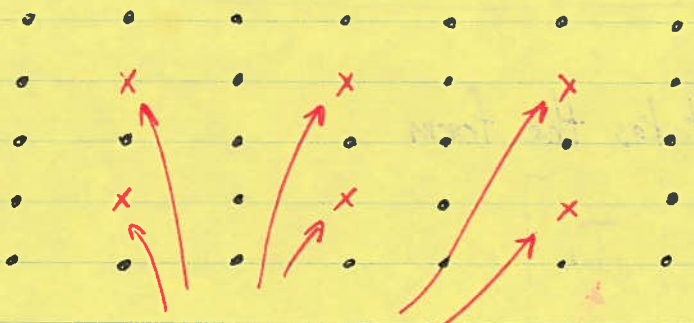
- and correct roughly 2^{n-k} error patterns ← proportion

channel coding taxonomy



17.2 Linear Block Code Spaces

- map one of 2^k message k-tuples $\bar{m}$ to a unique codeword n-tuple $\bar{U}$
- a subspace of 2^k signals/sequences in a space of 2^n n-tuples $\bar{V}_i$



2^k n-tuples $\bar{U}$ among 2^n n-tuples $\bar{V}_i$

"codewords"

- $\bar{U}$ are special such that any (key properties of a "subspace")

$$\bar{U}_i \oplus \bar{U}_j \in \bar{U}$$ ①

($\oplus$: mod 2 sum ("XOR"); $\in$: belongs to $\bar{U}$)

} "closure property of a subspace" [also requires all zeros vector in $\bar{U}$ (to be a subspace)] ②

17.3 Systematic Block Code Generation

- a $\bar{U}$ n-tuple can be obtained by multiplying $\bar{m}$ k-tuple (message) by "GENERATOR MATRIX"

$$\underset{[1\times n]}{\bar{U}} = \underset{[1\times k]}{\bar{m}} \cdot \underset{[k\times n]}{\bar{G}}$$

} i.e. don't need to remember 2^k n-bit codewords (1 for ea. 2^k message), just a $k \times n$ matrix $\bar{G}$

- in general $\bar{G} = \begin{bmatrix} \bar{V}_1 \\ \vdots \\ \bar{V}_k \end{bmatrix}$ } k linearly independent n-tuples that SPAN THE SUBSPACE (a BASIS)

- in a SYSTEMATIC mapping: $\bar{U} = [\underbrace{p_1, \ldots, p_{n-k}}_{\text{parity bits}}, \underbrace{m_1, \ldots, m_k}_{\text{message bits}}]$

(i.e. message appears directly in codeword)

generator matrix takes the form

$$\underset{[k \times n]}{\bar{G}} = \left[\bar{P} \,\vdots\, \bar{I}_k \right]$$

$(k) \times (n-k)$ parity array

$(k \times k)$ identity matrix directly mapping message bits into codeword

NOTE: in our matrix operations...
element-by-element multiply is bitwise AND "$\cdot$"
" " " add " " XOR "$\oplus$"

17.4 Parity Check Matrix

- how does the RX check for errors... a matrix that enables decoding of received vectors

$$\bar{H} = \left[\bar{I}_{n-k} \,\vdots\, \bar{P}^T \right]$$

$(n-k) \times (n)$ — $(n-k) \times (n-k)$ — $(n-k) \times k$

a matrix that sniffs out errors

rows of $\bar{H}$ orthogonal to rows of $\bar{G}$ $\Rightarrow G\bar{H}^T = \emptyset$

$[k \times n]$ $[n \times (n-k)]$ $[k \times (n-k)]$

G's rows correlated with H's rows
(ea. row has n elements)

$$\bar{H}^T = \begin{bmatrix} \bar{I}_{n-k} \\ \hline \bar{P} \end{bmatrix} \quad \text{so} \quad G\bar{H}^T = [\bar{P} \,\vdots\, \bar{I}_k] \begin{bmatrix} \bar{I}_{n-k} \\ \hline \bar{P} \end{bmatrix}$$

$(k \times n) \qquad n \times (n-k)$

- linking back to our codewords

$$\bar{U} = \bar{m}\,G$$

$$\therefore \quad \bar{U}\bar{H}^T = \bar{m}\,\underbrace{G\bar{H}^T}_{[k\times(n-k)]}_{[1\times k]} = \bar{\emptyset}_{[1\times(n-k)]} \qquad \text{again...}$$

$$\dots \quad \underset{[1\times n]}{\bar{U}}\,\underset{[n\times(n-k)]}{\bar{H}^T} = \underset{[1\times(n-k)]}{\bar{\emptyset}} \qquad \text{expanded...}$$

$$[\underbrace{p_1, p_2, \dots p_{n-k}}_{n-k}, \underbrace{m_1, \dots m_k}_{k}] \begin{bmatrix} \bar{I}_{n-k} \\ \hline \bar{P} \end{bmatrix} \quad \begin{matrix} \leftarrow n-k \rightarrow \\ \\ \updownarrow n-k \\ \updownarrow k \end{matrix}$$

ℓ columns of $[\bar{P}_1], [\bar{P}_2]$

$$= p_1 \oplus ([m_1, \dots m_k][\bar{P}_1]) = p_1 \oplus p_1$$

$$p_2 \oplus ([m_1, \dots m_k][\bar{P}_2]) = p_2 \oplus p_2$$

$$\vdots$$

$$p_{n-k} \oplus ([m_1, \dots, m_k][\bar{P}_{n-k}]) = p_{n-k} \oplus p_{n-k}$$

$\epsilon = 0$

17.5 Syndrome Testing

- ... and if $\bar{U}\bar{H}^T \neq \bar{\emptyset}$... can we find the errors?

$\bar{r} = \bar{U} \oplus \bar{e}$

$[1 \times n]$ received vector — $[1 \times n]$ bitwise mod-2 sum — $[1 \times n]$ ERROR VECTOR: $e_i = 1$ when error in ith position

- evaluate...

$[1 \times n] \cdot [n \times (n-k)] = [1 \times (n-k)]$

$$\bar{r}\bar{H}^T = (\bar{U} \oplus \bar{e})\bar{H}^T = \bar{\emptyset} \oplus \bar{e}\bar{H}^T = e\bar{H}^T = \bar{S}$$

- result of our parity check.... SYNDROME of $\bar{r}$
- if $\bar{r}$ has errors $\bar{S} \neq \bar{\emptyset}$ $[1 \times (n-k)]$
- NOTE: $\bar{r}\bar{H}^T$ and $\bar{e}\bar{H}^T$ produce same $\bar{S}$!

What's the significance of this?!?!: If $\bar{S}$ is related to $\bar{e}$ then when $\bar{S}$ is derived from $\bar{r}$ (as it always is in the RX) it can "somehow" be used to identify $\bar{e}$ and ∴ correct for it

-OR-

$\bar{S}$ is generated from $\bar{r}$, but it clearly can be generated from $\bar{e}$ ∴ knowing $\bar{S}$ and $\bar{H}^T$ we should be able to identify $\bar{e}$ and CORRECT IT!!!

17.6 Error Detecting & Correcting Capability

Hamming weight $w(\bar{U})$ = # of 1's in the codeword

Hamming distance $d(\bar{U}, \bar{V})$ = # of different elements

$$d(01011, 11001) = 2$$

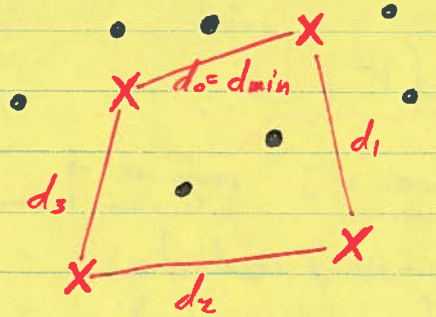
$$d(\bar{U}, \bar{V}) = w(\bar{U} \oplus \bar{V})$$

∴ distance between two codewords $\bar{U}_i$ and $\bar{U}_j$ is just the Hamming weight of their sum

- but remember! $\bar{U}_i \oplus \bar{U}_j = \bar{U}_k$ ← another valid codeword (in the case of linear codes)

∴ $d_{min} = w_{min}$ (excluding all zeros $\bar{U}$)

geometrically...



for ... $\bar{U}_i$ ← d_{min} → $\bar{U}_j$ .... what's

d_{min}: the minimum number of bit flips needed to turn one code word into another (and hence have no chance of telling if error occurred)

the largest # of errors a codeword can sustain such that we RECOGNIZE (DETECT) a problem? $d_{min} - 1 = e$: error detection capability

- what is the largest number of errors that can be CORRECTER?

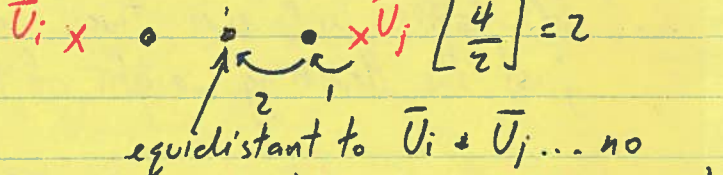
$$t = \left\lfloor \frac{d_{min} - 1}{2} \right\rfloor$$

"error correction capability"

$$\left\lfloor \frac{5-1}{2} \right\rfloor = 2$$

- hence if you want a code with a certain guaranteed correction ability you need to set the Hamming weight for the t you want

if we just used $\left\lfloor \frac{d_{min}}{2} \right\rfloor$ we'd have a problem e.g.



equidistant to $\bar{U}_i$ & $\bar{U}_j$ ... no way to tell which is the correct codeword

- you might also want to consider probabilistic factors

- for a channel with a probability p for flipping any one bit, the chance that my codeword will have an error I can't correct is

$$P_M \lesssim \sum_{j=t+1}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

- $\lesssim$: approx. (can correct some errors where codes exceed t spacing)
- $\sum_{j=t+1}^{n}$: only count those errors exceeding your guaranteed correction level, t
- $\binom{n}{j}$: how many combinations of j errors out of n
- p^j: prob. of j errors
- $(1-p)^{n-j}$: prob. rest of bits aren't in error

BER approximation $P_B \approx \frac{1}{n} \sum_{j=t+1}^{n} j \binom{n}{j} p^j (1-p)^{n-j}$

- probability of undetected error

$$P_{nd} = \sum_{j=1}^{n} A_j p^j (1-p)^{n-j}$$

A_j ← number of codewords with Hamming weight j

$\binom{n}{j}$ replaced by A_j

e.g. code with $(n,k) = (7,4)$ where $A_0 = 1$, $A_1 = A_2 = 0$, $A_3 = 7$, $A_4 = 7$, $A_5 = A_6 = 0$, $A_7 = 1$

$$P_{nd} = 7p^3(1-p)^{7-3} + 7p^4(1-p)^{7-4} + 1 \cdot p^7$$

17.7 Simultaneous Error Correction & Detection

- using a code to do a bit of both

$$d_{min} \geq \alpha + \beta + 1 \qquad \beta \geq \alpha$$

α: how many bits you correct; β: how many bits you can detect

e.g. $d_{min} = 7$

β	α
6	0 } can detect up to 6 bits but correct none
⋮	⋮
4	2 } can detect up to 4 errors & if there are only 2 errors I can correct them too (from my subset of 4)
3	3 } correct & detect 3 errors (obviously if you correct 3 you detect at least 3)