Information Theory and Networks

Lecture 26: Coding with Noise

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

October 9, 2013

Part I

Coding with Noise

There is a recorded case of a two-word military signal which suffered a processing delay of 150 years. The message, deciphered at the Pentagon in 1972, simply read "Send Reinforcements". It was sent on 1830 from Little Bighorn by General Custer.

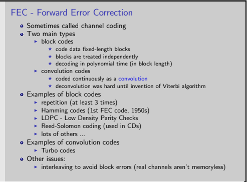
The Alice and Bob After Dinner Speech,
John Gordon, 1984

Section 1

Error Correction

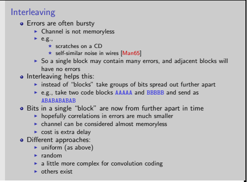
FEC - Forward Error Correction

- Sometimes called channel coding
- Two main types
  - block codes
    - code data fixed-length blocks
    - blocks are treated independently
    - decoding in polynomial time (in block length)
  - convolution codes
    - coded continuously as a convolution
    - deconvolution was hard until invention of Viterbi algorithm
- Examples of block codes
  - repetition (at least 3 times)
  - Hamming codes (1st FEC code, 1950s)
  - LDPC - Low Density Parity Checks
  - Reed-Solomon coding (used in CDs)
  - lots of others ...
- Examples of convolution codes
  - Turbo codes
- Other issues:
  - interleaving to avoid block errors (real channels aren't memoryless)


FEC - Forward Error Correction
- Sometimes called channel coding
- Two main types
  - block codes
    - code data fixed-length blocks
    - blocks are treated independently
    - decoding in polynomial time (in block length)
  - convolution codes
    - coded continuously as a convolution
    - deconvolution was hard until invention of Viterbi algorithm
- Examples of block codes
  - repetition (at least 3 times)
  - Hamming codes (1st FEC code, 1950s)
  - LDPC - Low Density Parity Checks
  - Reed-Solomon coding (used in CDs)
  - lots of others ...
- Examples of convolution codes
  - Turbo codes
- Other issues:
  - interleaving to avoid block errors (real channels aren't memoryless)

Interleaving

- Errors are often bursty
  - Channel is not memoryless
  - e.g.,
    - scratches on a CD
    - self-similar noise in wires [Man65]
  - So a single block may contain many errors, and adjacent blocks will have no errors
- Interleaving helps this:
  - instead of "blocks" take groups of bits spread out further apart
  - e.g., take two code blocks `AAAAA` and `BBBBB` and send as `ABABABABAB`
- Bits in a single "block" are now from further apart in time
  - hopefully correlations in errors are much smaller
  - channel can be considered almost memoryless
  - cost is extra delay
- Different approaches:
  - uniform (as above)
  - random
  - a little more complex for convolution coding
  - others exist


Interleaving
- Errors are often bursty
  - Channel is not memoryless
  - e.g.,
    - scratches on a CD
    - self-similar noise in wires [Man65]
  - So a single block may contain many errors, and adjacent blocks will have no errors
- Interleaving helps this:
  - instead of "blocks" take groups of bits spread out further apart
  - e.g., take two code blocks AAAAA and BBBBB and send as ABABABABAB
- Bits in a single "block" are now from further apart in time
  - hopefully correlations in errors are much smaller
  - channel can be considered almost memoryless
  - cost is extra delay
- Different approaches:
  - uniform (as above)
  - random
  - a little more complex for convolution coding
  - others exist

Hamming Codes

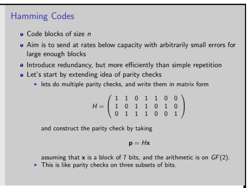
- Code blocks of size n
- Aim is to send at rates below capacity with arbitrarily small errors for large enough blocks
- Introduce redundancy, but more efficiently than simple repetition
- Let's start by extending idea of parity checks
  - lets do multiple parity checks, and write them in matrix form

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

    and construct the parity check by taking

$$\mathbf{p} = H\mathbf{x}$$

    assuming that $\mathbf{x}$ is a block of 7 bits, and the arithmetic is on $GF(2)$.
  - This is like parity checks on three subsets of bits.


Hamming Codes
- Code blocks of size n
- Aim is to send at rates below capacity with arbitrarily small errors for large enough blocks
- Introduce redundancy, but more efficiently than simple repetition
- Let's start by extending idea of parity checks
  - lets do multiple parity checks, and write them in matrix form

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

and construct the parity check by taking

$$\mathbf{p} = H\mathbf{x}$$

assuming that $\mathbf{x}$ is a block of 7 bits, and the arithmetic is on $GF(2)$.
  - This is like parity checks on three subsets of bits.

Hamming Codes

- The parity check is constructed using

$$\mathbf{p} = H\mathbf{x}$$

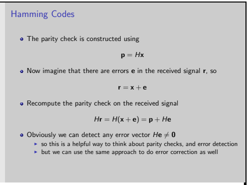
- Now imagine that there are errors $\mathbf{e}$ in the received signal $\mathbf{r}$, so

$$\mathbf{r} = \mathbf{x} + \mathbf{e}$$

- Recompute the parity check on the received signal

$$H\mathbf{r} = H(\mathbf{x} + \mathbf{e}) = \mathbf{p} + H\mathbf{e}$$

- Obviously we can detect any error vector $H\mathbf{e} \neq \mathbf{0}$
  - so this is a helpful way to think about parity checks, and error detection
  - but we can use the same approach to do error correction as well


Hamming Codes
- The parity check is constructed using

$$\mathbf{p} = H\mathbf{x}$$

- Now imagine that there are errors $\mathbf{e}$ in the received signal $\mathbf{r}$, so

$$\mathbf{r} = \mathbf{x} + \mathbf{e}$$

- Recompute the parity check on the received signal

$$H\mathbf{r} = H(\mathbf{x} + \mathbf{e}) = \mathbf{p} + H\mathbf{e}$$

- Obviously we can detect any error vector $H\mathbf{e} \neq \mathbf{0}$
  - so this is a helpful way to think about parity checks, and error detection
  - but we can use the same approach to do error correction as well

Vectors with $H\mathbf{e} = \mathbf{0}$

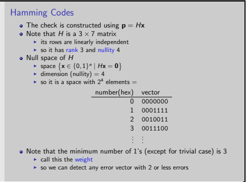
number(hex)	vector
0	0000000
1	0001111
2	0010011
3	0011100
4	0100101
5	0101010
6	0110110
7	0111001
8	1000110
9	1001001
A	1010101
B	1011010
C	1100011
D	1101100
E	1110000
F	1111111

Hamming Codes

- The check is constructed using $\mathbf{p} = H\mathbf{x}$
- Note that H is a 3×7 matrix
  - its rows are linearly independent
  - so it has rank 3 and nullity 4
- Null space of H
  - space $\{\mathbf{x} \in \{0,1\}^n \mid H\mathbf{x} = \mathbf{0}\}$
  - dimension (nullity) = 4
  - so it is a space with 2^4 elements =

number(hex)	vector
0	0000000
1	0001111
2	0010011
3	0011100
⋮	⋮

- Note that the minimum number of 1's (except for trivial case) is 3
  - call this the weight
  - so we can detect any error vector with 2 or less errors


Hamming Codes
The check is constructed using $\mathbf{p} = H\mathbf{x}$
Note that H is a 3×7 matrix
its rows are linearly independent
so it has rank 3 and nullity 4
Null space of H
space $\{\mathbf{x} \in \{0,1\}^n \mid H\mathbf{x} = \mathbf{0}\}$
dimension (nullity) = 4
so it is a space with 2^4 elements =
number(hex) vector
0 0000000
1 0001111
2 0010011
3 0011100
⋮ ⋮
Note that the minimum number of 1's (except for trivial case) is 3
call this the weight
so we can detect any error vector with 2 or less errors

Definition (Matrix rank)

The column (row) rank of a matrix is defined to be the maximum number of linearly independent columns (rows) of the matrix.

The column rank of a matrix A is the dimension of the column space of the matrix, i.e., the space of all linear combinations of its columns
Likewise for row rank.
It turns out that the column and row rank are equal. Also, importantly here,

$$rank(A) + nullity(A) = n$$

where

- n is the number of columns
- the nullity is the dimension of the null space of A, i.e., the space $\{\mathbf{x} \mid A\mathbf{x} = \mathbf{0}\}$

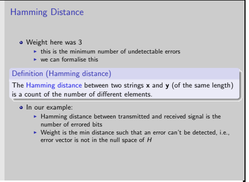
Hamming Distance

- Weight here was 3
  - this is the minimum number of undetectable errors
  - we can formalise this

Definition (Hamming distance)

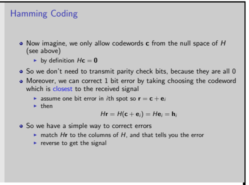
The Hamming distance between two strings $\mathbf{x}$ and $\mathbf{y}$ (of the same length) is a count of the number of different elements.

- In our example:
  - Hamming distance between transmitted and received signal is the number of errored bits
  - Weight is the min distance such that an error can't be detected, i.e., error vector is not in the null space of H


Hamming Distance
Weight here was 3
this is the minimum number of undetectable errors
we can formalise this
Definition (Hamming distance)
The Hamming distance between two strings $\mathbf{x}$ and $\mathbf{y}$ (of the same length) is a count of the number of different elements.
In our example:
Hamming distance between transmitted and received signal is the number of errored bits
Weight is the min distance such that an error can't be detected, i.e., error vector is not in the null space of H

Hamming Coding

- Now imagine, we only allow codewords $\mathbf{c}$ from the null space of H (see above)
  - by definition $H\mathbf{c} = \mathbf{0}$
- So we don't need to transmit parity check bits, because they are all 0
- Moreover, we can correct 1 bit error by taking choosing the codeword which is closest to the received signal
  - assume one bit error in ith spot so $\mathbf{r} = \mathbf{c} + \mathbf{e}_i$
  - then
$$H\mathbf{r} = H(\mathbf{c} + \mathbf{e}_i) = H\mathbf{e}_i = \mathbf{h}_i$$
- So we have a simple way to correct errors
  - match $H\mathbf{r}$ to the columns of H, and that tells you the error
  - reverse to get the signal

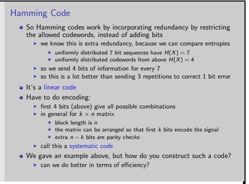

Hamming Coding
- Now imagine, we only allow codewords $\mathbf{c}$ from the null space of H (see above)
  - by definition $H\mathbf{c} = \mathbf{0}$
- So we don't need to transmit parity check bits, because they are all 0
- Moreover, we can correct 1 bit error by taking choosing the codeword which is closest to the received signal
  - assume one bit error in ith spot so $\mathbf{r} = \mathbf{c} + \mathbf{e}_i$
  - then
$$H\mathbf{r} = H(\mathbf{c} + \mathbf{e}_i) = H\mathbf{e}_i = \mathbf{h}_i$$
- So we have a simple way to correct errors
  - match $H\mathbf{r}$ to the columns of H, and that tells you the error
  - reverse to get the signal

We mean closest in the sense that it minimises the Hamming distance.

$\mathbf{e}_i$ is the vector with 0s except in the ith spot which is 1

Hamming Code

- So Hamming codes work by incorporating redundancy by restricting the allowed codewords, instead of adding bits
  - we know this is extra redundancy, because we can compare entropies
    - ★ uniformly distributed 7 bit sequences have $H(X) = 7$
    - ★ uniformly distributed codewords from above $H(X) = 4$
  - so we send 4 bits of information for every 7
  - so this is a lot better than sending 3 repetitions to correct 1 bit error
- It's a linear code
- Have to do encoding:
  - first 4 bits (above) give all possible combinations
  - in general for $k \times n$ matrix
    - ★ block length is n
    - ★ the matrix can be arranged so that first k bits encode the signal
    - ★ extra $n - k$ bits are parity checks
  - call this a systematic code
- We gave an example above, but how do you construct such a code?
  - can we do better in terms of efficiency?


Hamming Code
- So Hamming codes work by incorporating redundancy by restricting the allowed codewords, instead of adding bits
  - we know this is extra redundancy, because we can compare entropies
    - uniformly distributed 7 bit sequences have $H(X) = 7$
    - uniformly distributed codewords from above $H(X) = 4$
  - so we send 4 bits of information for every 7
  - so this is a lot better than sending 3 repetitions to correct 1 bit error
- It's a linear code
- Have to do encoding:
  - first 4 bits (above) give all possible combinations
  - in general for $k \times n$ matrix
    - block length is n
    - the matrix can be arranged so that first k bits encode the signal
    - extra $n - k$ bits are parity checks
  - call this a systematic code
- We gave an example above, but how do you construct such a code?
  - can we do better in terms of efficiency?

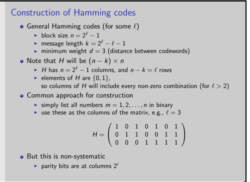
symbol	codeword	encoding
0	0000	0000000
1	0001	0001111
2	0010	0010011
3	0011	0011100
4	0100	0100101
5	0101	0101010
6	0110	0110110
7	0111	0111001
8	1000	1000110
9	1001	1001001
A	1010	1010101
B	1011	1011010
C	1100	1100011
D	1101	1101100
E	1110	1110000
F	1111	1111111

Construction of Hamming codes

- General Hamming codes (for some ℓ)
  - block size $n = 2^\ell - 1$
  - message length $k = 2^\ell - \ell - 1$
  - minimum weight $d = 3$ (distance between codewords)
- Note that H will be $(n-k) \times n$
  - H has $n = 2^\ell - 1$ columns, and $n - k = \ell$ rows
  - elements of H are $\{0, 1\}$,
    so columns of H will include every non-zero combination (for $\ell > 2$)
- Common approach for construction
  - simply list all numbers $m = 1, 2, \ldots, n$ in binary
  - use these as the columns of the matrix, e.g., $\ell = 3$

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

- But this is non-systematic
  - parity bits are at columns 2^i


Construction of Hamming codes
General Hamming codes (for some ℓ)
block size $n = 2^\ell - 1$
message length $k = 2^\ell - \ell - 1$
minimum weight $d = 3$ (distance between codewords)
Note that H will be $(n-k) \times n$
H has $n = 2^\ell - 1$ columns, and $n - k = \ell$ rows
elements of H are $\{0, 1\}$,
so columns of H will include every non-zero combination (for $\ell > 2$)
Common approach for construction
simply list all numbers $m = 1, 2, \ldots, n$ in binary
use these as the columns of the matrix, e.g., $\ell = 3$
$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$
But this is non-systematic
parity bits are at columns 2^i

$\ell = 2$ (special case)

$$H = [1\,1\,1]$$

$\ell = 3$ we get

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

$\ell = 4$ we get

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

Construction of Hamming Codes

- We also need a $k \times n$ code generator matrix G such that

$$HG^T = \mathbf{0}$$

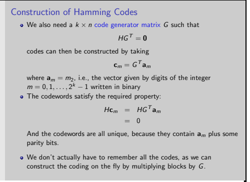
  codes can then be constructed by taking

$$\mathbf{c}_m = G^T \mathbf{a}_m$$

  where $\mathbf{a}_m = m_2$, i.e., the vector given by digits of the integer $m = 0, 1, \ldots, 2^k - 1$ written in binary
- The codewords satisfy the required property:

$$\begin{aligned} H\mathbf{c}_m &= HG^T\mathbf{a}_m \\ &= 0 \end{aligned}$$

  And the codewords are all unique, because they contain $\mathbf{a}_m$ plus some parity bits.
- We don't actually have to remember all the codes, as we can construct the coding on the fly by multiplying blocks by G.


Construction of Hamming Codes
We also need a $k \times n$ code generator matrix G such that
$HG^T = \mathbf{0}$
codes can then be constructed by taking
$\mathbf{c}_m = G^T \mathbf{a}_m$
where $\mathbf{a}_m = m_2$, i.e., the vector given by digits of the integer $m = 0, 1, \ldots, 2^k - 1$ written in binary
The codewords satisfy the required property:
$H\mathbf{c}_m = HG^T\mathbf{a}_m = 0$
And the codewords are all unique, because they contain $\mathbf{a}_m$ plus some parity bits.
We don't actually have to remember all the codes, as we can construct the coding on the fly by multiplying blocks by G.

$\ell = 2$

$$H = [1\,1\,1]$$

has possible

$$G = [1\,1\,1]$$

Then the code words based on $a_0 = 0$ and $a_1 = 1$

$$\begin{aligned} (0,0,0)^T &= G^T 0 \\ (1,1,1)^T &= G^T 1 \end{aligned}$$

Hence the simplest Hamming code is the repetition code.

Construction of Systematic Hamming Codes

- We can turn this into a systematic form (as above) by
  - permuting the columns
  - performing row operations (creating linear combinations) of the rows
- Use these to put it into the form

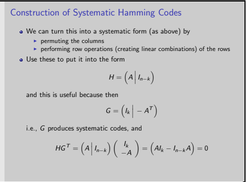
$$H = \left(A \,\middle|\, I_{n-k} \right)$$

and this is useful because then

$$G = \left(I_k \,\middle|\, -A^T \right)$$

i.e., G produces systematic codes, and

$$HG^T = \left(A \,\middle|\, I_{n-k} \right) \begin{pmatrix} I_k \\ -A \end{pmatrix} = \left(AI_k - I_{n-k}A \right) = 0$$


Construction of Systematic Hamming Codes
We can turn this into a systematic form (as above) by
permuting the columns
performing row operations (creating linear combinations) of the rows
Use these to put it into the form
$H = (A | I_{n-k})$
and this is useful because then
$G = (I_k | -A^T)$
i.e., G produces systematic codes, and
$HG^T = (A | I_{n-k}) \begin{pmatrix} I_k \\ -A \end{pmatrix} = (AI_k - I_{n-k}A) = 0$

$\ell = 3$ the systematic version of H is

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

and so

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

and then

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

NB: Matlab has a function `hammgen` that creates the H matrix, but seems to do it as $H = (I_{n-k} \mid A)$.

Block Codes in General

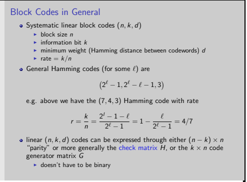
- Systematic linear block codes (n, k, d)
  - block size n
  - information bit k
  - minimum weight (Hamming distance between codewords) d
  - rate $= k/n$
- General Hamming codes (for some ℓ) are

$$(2^\ell - 1, 2^\ell - \ell - 1, 3)$$

e.g. above we have the $(7, 4, 3)$ Hamming code with rate

$$r = \frac{k}{n} = \frac{2^l - 1 - \ell}{2^\ell - 1} = 1 - \frac{\ell}{2^\ell - 1} = 4/7$$

- linear (n, k, d) codes can be expressed through either $(n-k) \times n$ "parity" or more generally the check matrix H, or the $k \times n$ code generator matrix G
  - doesn't have to be binary


Block Codes in General
Systematic linear block codes (n, k, d)
block size n
information bit k
minimum weight (Hamming distance between codewords) d
rate $= k/n$
General Hamming codes (for some ℓ) are
$(2^\ell - 1, 2^\ell - \ell - 1, 3)$
e.g. above we have the $(7,4,3)$ Hamming code with rate
$r = \frac{k}{n} = \frac{2^l - 1 - \ell}{2^\ell - 1} = 1 - \frac{\ell}{2^\ell - 1} = 4/7$
linear (n, k, d) codes can be expressed through either $(n-k) \times n$ "parity" or more generally the check matrix H, or the $k \times n$ code generator matrix G
doesn't have to be binary

Hamming codes

ℓ	n	rate
2	3	$1 - 2/3 \simeq 0.333$
3	7	$1 - 3/7 \simeq 0.571$
4	15	$1 - 4/15 \simeq 0.733$
5	31	$1 - 5/31 \simeq 0.839$
6	63	$1 - 6/63 \simeq 0.905$

so we can see the rate converges to 1 quite quickly (but block sizes also increase quite quickly).

Actually, to detect 2 bit errors, we have to assume we aren't doing error correction as well, as $d = 3$ means that two bit errors in one codeword could look like 1 bit error in another.

Apparently popular is the $(127, 120, 3)$ code, with an extra parity bit so that single errors can be corrected, and double errors detected at the same time.

Analysis of Block Codes

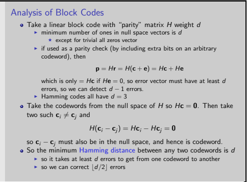
- Take a linear block code with "parity" matrix H weight d
  - minimum number of ones in null space vectors is d
    - except for trivial all zeros vector
  - if used as a parity check (by including extra bits on an arbitrary codeword), then

$$\mathbf{p} = H\mathbf{r} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e}$$

    which is only $= H\mathbf{c}$ if $H\mathbf{e} = 0$, so error vector must have at least d errors, so we can detect $d - 1$ errors.
  - Hamming codes all have $d = 3$
- Take the codewords from the null space of H so $H\mathbf{c} = \mathbf{0}$. Then take two such $\mathbf{c}_i \neq \mathbf{c}_j$ and

$$H(\mathbf{c}_i - \mathbf{c}_j) = H\mathbf{c}_i - H\mathbf{c}_j = \mathbf{0}$$

  so $\mathbf{c}_i - \mathbf{c}_j$ must also be in the null space, and hence is codeword.
- So the minimum Hamming distance between any two codewords is d
  - so it takes at least d errors to get from one codeword to another
  - so we can correct $\lfloor d/2 \rfloor$ errors


Analysis of Block Codes
- Take a linear block code with "parity" matrix H weight d
  - minimum number of ones in null space vectors is d
    - except for trivial all zeros vector
  - if used as a parity check (by including extra bits on an arbitrary codeword), then $\mathbf{p} = H\mathbf{r} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{c} + H\mathbf{e}$ which is only $= H\mathbf{c}$ if $H\mathbf{e} = 0$, so error vector must have at least d errors, so we can detect $d - 1$ errors.
  - Hamming codes all have $d = 3$
- Take the codewords from the null space of H so $H\mathbf{c} = \mathbf{0}$. Then take two such $\mathbf{c}_i \neq \mathbf{c}_j$ and $H(\mathbf{c}_i - \mathbf{c}_j) = H\mathbf{c}_i - H\mathbf{c}_j = \mathbf{0}$ so $\mathbf{c}_i - \mathbf{c}_j$ must also be in the null space, and hence is codeword.
- So the minimum Hamming distance between any two codewords is d
  - so it takes at least d errors to get from one codeword to another
  - so we can correct $\lfloor d/2 \rfloor$ errors

Intuitively, we take our set of symbols, and map them into a larger space (in this case a higher dimensional space), where they are further apart.

If we choose codewords that are distance d apart, and correct errors by assuming the closest codeword is correct, then we will map back to the correct codeword if the received word is less than $d/2$ from the original.

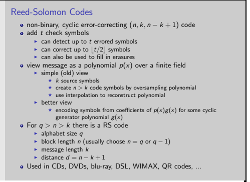
Definition (Cyclic code)

A cyclic code over $GF(q)$ is a code with the special property that any cyclic shift of a codeword is another codeword.

That is, if $(x_1, x_2, \ldots, x_{n-1}, x_n)$ is a codeword, then $(x_2, x_3, \ldots x_n, x_1)$ is a codeword (as are other cyclic shifts) [Gal68, p.221]. Cyclic codes have a generator polynomial (as in CRCs), from which all the codewords can be derived. So CRCs are actually just a special case of block error coding – and in general linear error detection and error correction codes are related.

Reed-Solomon Codes

- non-binary, cyclic error-correcting $(n, k, n - k + 1)$ code
- add t check symbols
  - can detect up to t errored symbols
  - can correct up to $\lfloor t/2 \rfloor$ symbols
  - can also be used to fill in erasures
- view message as a polynomial $p(x)$ over a finite field
  - simple (old) view
    - k source symbols
    - create $n > k$ code symbols by oversampling polynomial
    - use interpolation to reconstruct polynomial
  - better view
    - encoding symbols from coefficients of $p(x)g(x)$ for some cyclic generator polynomial $g(x)$
- For $q > n > k$ there is a RS code
  - alphabet size q
  - block length n (usually choose $n = q$ or $q - 1$)
  - message length k
  - distance $d = n - k + 1$
- Used in CDs, DVDs, blu-ray, DSL, WIMAX, QR codes, ...


Reed-Solomon Codes
- non-binary, cyclic error-correcting $(n, k, n - k + 1)$ code
- add t check symbols
  - can detect up to t errored symbols
  - can correct up to $\lfloor t/2 \rfloor$ symbols
  - can also be used to fill in erasures
- view message as a polynomial $p(x)$ over a finite field
  - simple (old) view
    - k source symbols
    - create $n > k$ code symbols by oversampling polynomial
    - use interpolation to reconstruct polynomial
  - better view
    - encoding symbols from coefficients of $p(x)g(x)$ for some cyclic generator polynomial $g(x)$
- For $q > n > k$ there is a RS code
  - alphabet size q
  - block length n (usually choose $n = q$ or $q - 1$)
  - message length k
  - distance $d = n - k + 1$
- Used in CDs, DVDs, blu-ray, DSL, WIMAX, QR codes, ...

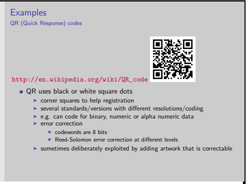
- CDs use interleaved $(32, 28, 5)$ and $(28, 24, 5)$ Reed-Solomon codes
  - can correct bursts of up to 4000 errors
  - needed because errors in CDs often caused by scratches

Examples

QR (Quick Response) codes

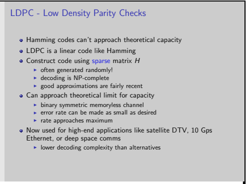
http://en.wikipedia.org/wiki/QR_code

- QR uses black or white square dots
  - corner squares to help registration
  - several standards/versions with different resolutions/coding
  - e.g. can code for binary, numeric or alpha numeric data
  - error correction
    - codewords are 8 bits
    - Reed-Solomon error correction at different levels
  - sometimes deliberately exploited by adding artwork that is correctable


Examples
QR (Quick Response) codes
http://en.wikipedia.org/wiki/QR_code
- QR uses black or white square dots
  - corner squares to help registration
  - several standards/versions with different resolutions/coding
  - e.g. can code for binary, numeric or alpha numeric data
  - error correction
    - codewords are 8 bits
    - Reed-Solomon error correction at different levels
  - sometimes deliberately exploited by adding artwork that is correctable

LDPC - Low Density Parity Checks

- Hamming codes can't approach theoretical capacity
- LDPC is a linear code like Hamming
- Construct code using sparse matrix H
  - often generated randomly!
  - decoding is NP-complete
  - good approximations are fairly recent
- Can approach theoretical limit for capacity
  - binary symmetric memoryless channel
  - error rate can be made as small as desired
  - rate approaches maximum
- Now used for high-end applications like satellite DTV, 10 Gps Ethernet, or deep space comms
  - lower decoding complexity than alternatives

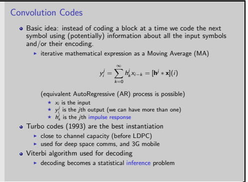

LDPC - Low Density Parity Checks
- Hamming codes can't approach theoretical capacity
- LDPC is a linear code like Hamming
- Construct code using sparse matrix H
  - often generated randomly!
  - decoding is NP-complete
  - good approximations are fairly recent
- Can approach theoretical limit for capacity
  - binary symmetric memoryless channel
  - error rate can be made as small as desired
  - rate approaches maximum
- Now used for high-end applications like satellite DTV, 10 Gps Ethernet, or deep space comms
  - lower decoding complexity than alternatives

Convolution Codes

- Basic idea: instead of coding a block at a time we code the next symbol using (potentially) information about all the input symbols and/or their encoding.
  - iterative mathematical expression as a Moving Average (MA)

$$y_i^j = \sum_{k=0}^{\infty} h_k^j x_{i-k} = [\mathbf{h}^j * \mathbf{x}](i)$$

  (equivalent AutoRegressive (AR) process is possible)
    - x_i is the input
    - y_i^j is the jth output (we can have more than one)
    - h_k^j is the jth impulse response
- Turbo codes (1993) are the best instantiation
  - close to channel capacity (before LDPC)
  - used for deep space comms, and 3G mobile
- Viterbi algorithm used for decoding
  - decoding becomes a statistical inference problem



Convolution examples:
Simple AR process

$$y(n) = \sum_{i=1}^{p} a(i) y(n-i) + b(0) x(n)$$

Simple (causal) MA process

$$y(n) = \sum_{i=0}^{q} b(i) x(n-i)$$

Equivalence, e.g., EWMA (Exponentially Weighted Moving Average)

$$AR : y(n) = a y(n-1) + (1-a) x(n)$$

is equivalent to

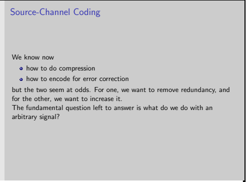
$$MA : y(n) = (1-a) \sum_{i=0}^{\infty} a^i x(n-i)$$

Source-Channel Coding

We know now

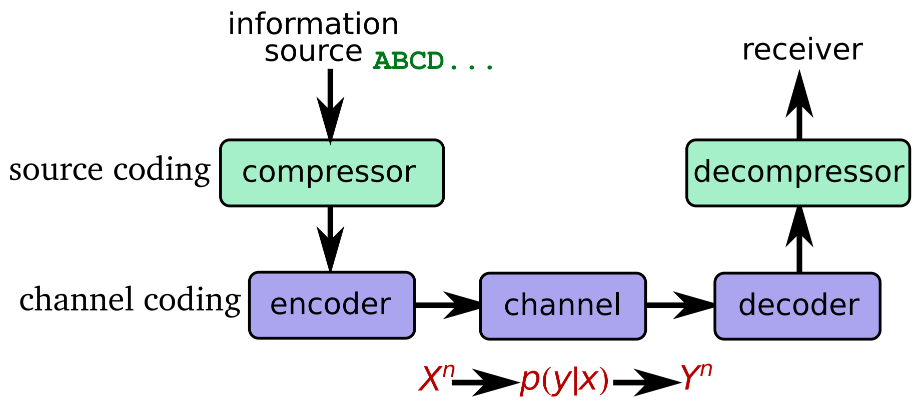
- how to do compression
- how to encode for error correction

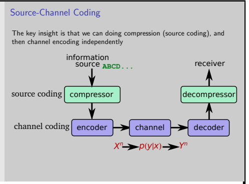
but the two seem at odds. For one, we want to remove redundancy, and for the other, we want to increase it.
The fundamental question left to answer is what do we do with an arbitrary signal?



Source-Channel Coding

The key insight is that we can doing compression (source coding), and then channel encoding independently



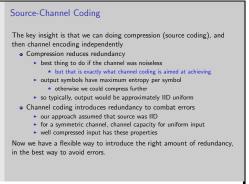

Source-Channel Coding
The key insight is that we can doing compression (source coding), and then channel encoding independently

Source-Channel Coding

The key insight is that we can doing compression (source coding), and then channel encoding independently

- Compression reduces redundancy
  - best thing to do if the channel was noiseless
    - but that is exactly what channel coding is aimed at achieving
  - output symbols have maximum entropy per symbol
    - otherwise we could compress further
  - so typically, output would be approximately IID uniform
- Channel coding introduces redundancy to combat errors
  - our approach assumed that source was IID
  - for a symmetric channel, channel capacity for uniform input
  - well compressed input has these properties

Now we have a flexible way to introduce the right amount of redundancy, in the best way to avoid errors.


Source-Channel Coding
The key insight is that we can doing compression (source coding), and then channel encoding independently
Compression reduces redundancy
best thing to do if the channel was noiseless
but that is exactly what channel coding is aimed at achieving
output symbols have maximum entropy per symbol
otherwise we could compress further
so typically, output would be approximately IID uniform
Channel coding introduces redundancy to combat errors
our approach assumed that source was IID
for a symmetric channel, channel capacity for uniform input
well compressed input has these properties
Now we have a flexible way to introduce the right amount of redundancy, in the best way to avoid errors.

Also in real systems, the two parts are often completely separate (see layering discussions). But also note that not all IP traffic is compressed, so channel coding may be making assumptions about input that aren't true.

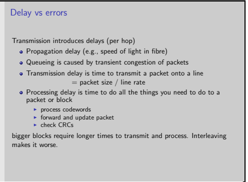
We can construct cases where the above doesn't work, or where two-stage encoders such as the above aren't optimal. In other cases, we might like to use the human ability to decode signals such as speech (which we are very good at), and hence we don't want to compress it, and try to detect errors algorithmically.

Delay vs errors

Transmission introduces delays (per hop)

- Propagation delay (e.g., speed of light in fibre)
- Queueing is caused by transient congestion of packets
- Transmission delay is time to transmit a packet onto a line
  = packet size / line rate
- Processing delay is time to do all the things you need to do to a packet or block
  - process codewords
  - forward and update packet
  - check CRCs

bigger blocks require longer times to transmit and process. Interleaving makes it worse.


Delay vs errors

Transmission introduces delays (per hop)
- Propagation delay (e.g., speed of light in fibre)
- Queueing is caused by transient congestion of packets
- Transmission delay is time to transmit a packet onto a line
  = packet size / line rate
- Processing delay is time to do all the things you need to do to a packet or block
  - process codewords
  - forward and update packet
  - check CRCs

bigger blocks require longer times to transmit and process. Interleaving makes it worse.

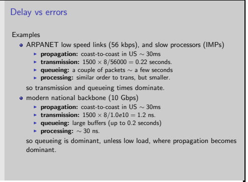
Delay vs errors

Examples

- ARPANET low speed links (56 kbps), and slow processors (IMPs)
  - **propagation:** coast-to-coast in US $\sim$ 30ms
  - **transmission:** $1500 \times 8/56000 = 0.22$ seconds.
  - **queueing:** a couple of packets $\sim$ a few seconds
  - **processing:** similar order to trans, but smaller.

  so transmission and queueing times dominate.
- modern national backbone (10 Gbps)
  - **propagation:** coast-to-coast in US $\sim$ 30ms
  - **transmission:** $1500 \times 8/1.0e10 = 1.2$ ns.
  - **queueing:** large buffers (up to 0.2 seconds)
  - **processing:** $\sim$ 30 ns.

  so queueing is dominant, unless low load, where propagation becomes dominant.


Delay vs errors

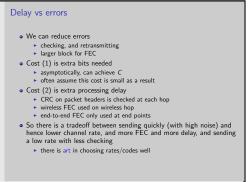
Examples
- ARPANET low speed links (56 kbps), and slow processors (IMPs)
  - propagation: coast-to-coast in US ~ 30ms
  - transmission: 1500 × 8/56000 = 0.22 seconds.
  - queueing: a couple of packets ~ a few seconds
  - processing: similar order to trans, but smaller.

  so transmission and queueing times dominate.
- modern national backbone (10 Gbps)
  - propagation: coast-to-coast in US ~ 30ms
  - transmission: 1500 × 8/1.0e10 = 1.2 ns.
  - queueing: large buffers (up to 0.2 seconds)
  - processing: ~ 30 ns.

  so queueing is dominant, unless low load, where propagation becomes dominant.

Delay vs errors

- We can reduce errors
  - checking, and retransmitting
  - larger block for FEC
- Cost (1) is extra bits needed
  - asymptotically, can achieve C
  - often assume this cost is small as a result
- Cost (2) is extra processing delay
  - CRC on packet headers is checked at each hop
  - wireless FEC used on wireless hop
  - end-to-end FEC only used at end points
- So there is a tradeoff between sending quickly (with high noise) and hence lower channel rate, and more FEC and more delay, and sending a low rate with less checking
  - there is art in choosing rates/codes well



Delay vs errors

- We can reduce errors
  - checking, and retransmitting
  - larger block for FEC
- Cost (1) is extra bits needed
  - asymptotically, can achieve C
  - often assume this cost is small as a result
- Cost (2) is extra processing delay
  - CRC on packet headers is checked at each hop
  - wireless FEC used on wireless hop
  - end-to-end FEC only used at end points
- So there is a tradeoff between sending quickly (with high noise) and hence lower channel rate, and more FEC and more delay, and sending a low rate with less checking
  - there is art in choosing rates/codes well

Error detection and feedback

- Maybe we can do better with error detection, and feedback than FEC?
  - channel capacity theorem assumed no feedback
  - error detection seems more efficient than correction
- Turns out that feedback capacity is just the same
  - see [CT91, Theorem 8.12.1, p.213]
- But error detection and feedback may be
  - easier to implement
  - more efficient for finite codes
- On the other hand
  - feedback requires more RTTs
  - could require implementation of timeout timers

Further reading I

Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley and Sons, 1991.

Robert G. Gallager, *Information theory and reliable communication*, John Wiley and Sons, 1968.

David J. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, 2011.

Benoit Mandelbrot, *Self-similar error clusters in communication systems and the concept of conditional stationarity*, IEEE Transactions on Communications Technology (1965), 71–90.