Lecture 9A: Error Detection and Correction

John M Pauly

March 2, 2026

Error Protection: Detection and Correction

Communication channels are subject to noise.

Noise distorts analog signals.

Noise can cause digital signals to be received as different values.

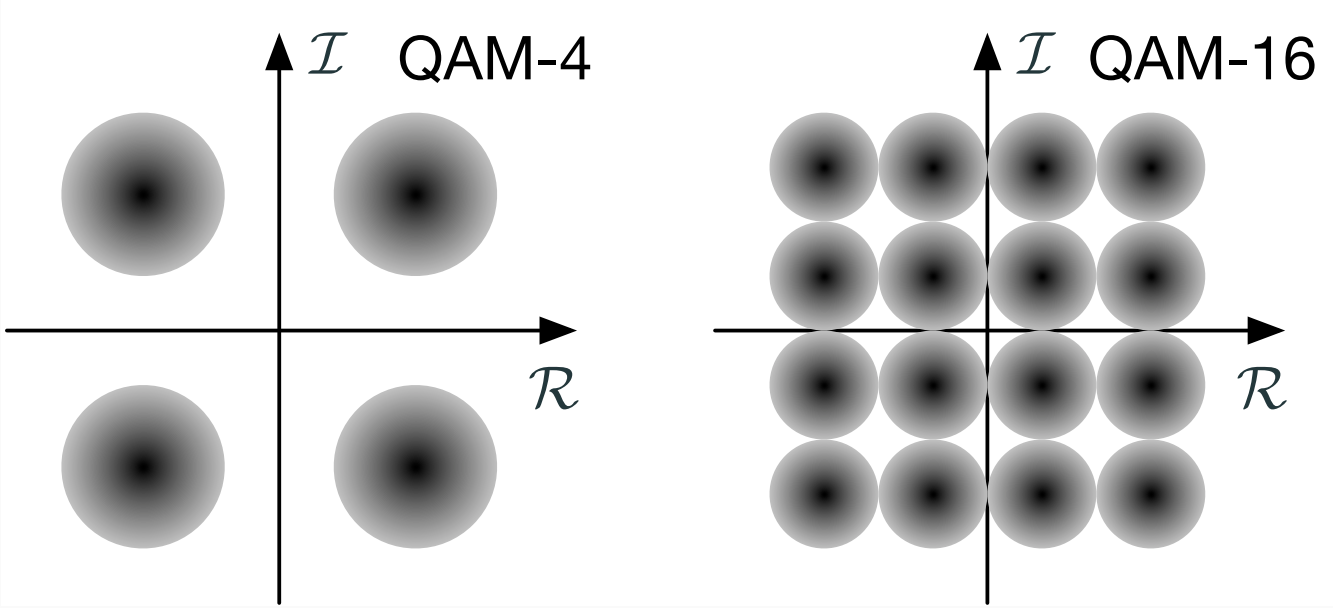
- Bits can be flipped
- Points in a signal constellation can be shifted.

Changes in a digital signal are called *errors*.

Based on notes from John Gill and Miki Lustig

Effects of Noise

Noise causes the constellation to blur,



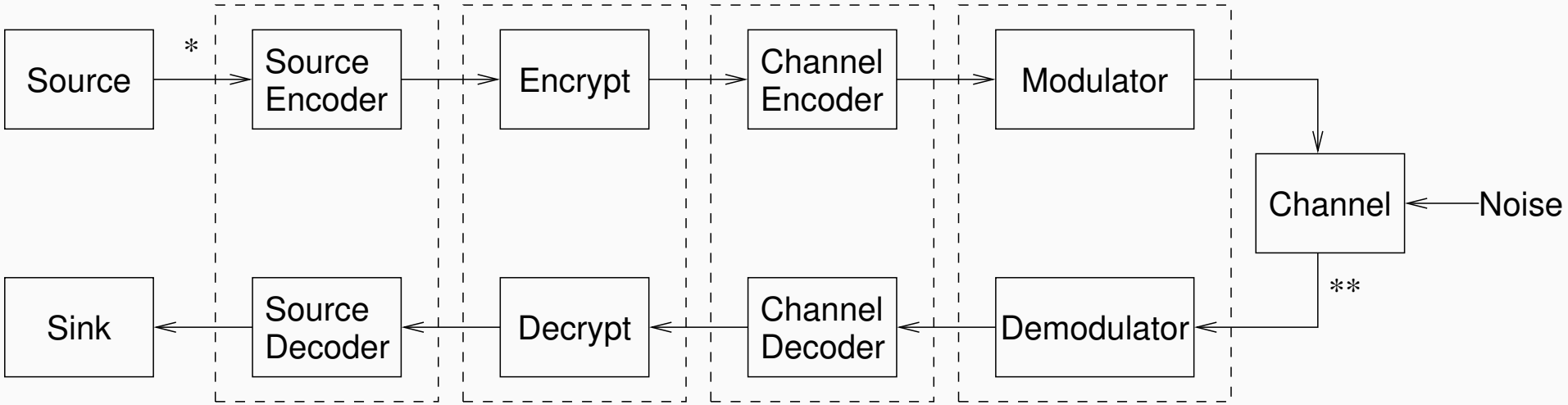
With enough noise we will get decoding errors

We'd like to be able to detect, and perhaps correct, these errors

To do this, we'll add some redundant information

Communication Systems

Recall the communication system block diagram:



We have concentrated on the modulator/demodulator blocks.

Error protection involves channel encoder and decoder.

Types of Error Protection

Error detection

- Goal: avoid accepting faulty data.
- Lost data may be unfortunate; wrong data may be disastrous.
- Solution: *checksums* are included (packets, frames, sectors).
  If any part of the message is altered, then the checksum is *not* valid (with high probability).

Error correction

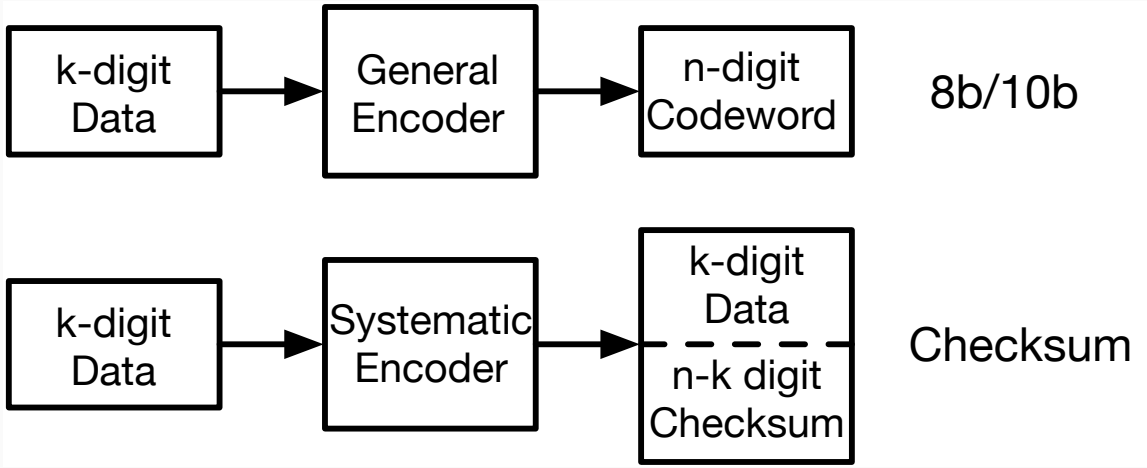
- Add redundancy to the message so that we can correct errors
- On reception determine which is the most likely message

Block Codes

k-digit blocks of information digits are encoded into n-digit codewords ($n \geq k$) by adding $p = n - k$ redundant check digits.

There is no memory between blocks. The encoding of each data block is independent of past and future blocks.

An encoding where digits are unchanged in the codewords is *systematic*.



Block Codes: Simple Parity-Check Codes

Append one check bit to data bits so that all codewords have the same overall parity — either even or odd.

Even-parity codewords are defined by a single *parity-check equation*:

$$c_1 \oplus c_2 \oplus \cdots \oplus c_n \;=\; (c_1 + c_2 + \cdots + c_n) \bmod 2 \;=\; 0\,,$$

where $\oplus$ denotes the exclusive-or operation.

If we XOR c_n to both sides we obtain an *encoding equation*:

$$c_n \;=\; c_1 \oplus c_2 \oplus \cdots \oplus c_{n-1}\,.$$

The *check bit* c_n is computed from the *data bits* $c_1, \ldots, c_{n-1}$.

Any single bit error (or any odd number of errors) can be detected.

Note: Any bit c_i can be considered to be the check bit because it can be computed from the other $n-1$ bits.

Polynomial Division

- We can consider parity as the remainder after polynomial division.
- For data bits $c_1, \cdots, c_7$ the parity bit is the remainder after dividing

$$c_1x^6 + c_2x^5 + c_3x^4 + c_4x^3 + c_5x^2 + c_6x + c_7$$

by $x + 1$.

- Example: $c_1, \cdots, c_7 = 1010001$ divided by 11

Encoding

```
          1 1 0 0 0 1 R 1
      ┌─────────────
1 1   │ 1 0 1 0 0 0 1
        1 1
        ─────
          1 1 0 0 0 1
          1 1
          ─────
          0 0 0 0 0 1
                  1 1
                  ─────
                    1   Remainder
```

Decoding

```
          1 1 0 0 0 1 0
      ┌─────────────
1 1   │ 1 0 1 0 0 0 1 [1]   Parity Bit
        1 1
        ─────
          1 1 0 0 0 1 1
          1 1
          ─────
          0 0 0 0 0 1 1
                  1 1
                  ─────
                    0   Remainder
```

Addition and subtraction are without carry (XOR)

Cyclic Redundancy Check (CRC)

If we use a longer divisor polynomial we can detect which bit is in error for a much longer bit stream

For an optimal polynomial of order n we can

- Detect any single or double bit error in $2^n - 1$ bits
- With a factor $1 + x$, detect any error in an odd number of bits

Lots of different polynomials are possible. Two very common ones are

- CRC-16-IBM : $x^{16} + x^{15} + x^2 + 1$
- CRC-CCITT : $x^{16} + x^{12} + x^5 + 1$

Other CRC codes from CRC-3 to CRC-64

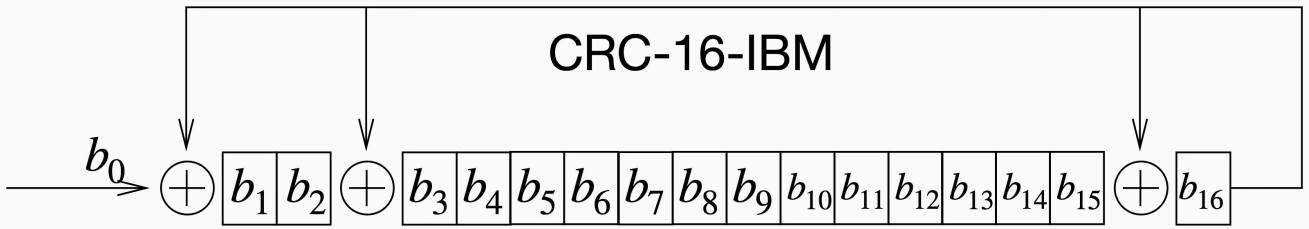
Can be applied to any length block less than $2^n - 1$

Used in Bluetooth, USB, GSM, X.25, ACARS (ACARS-16-ARINC)

CRC Implementation

Polynomial division can be implemented using linear feedback shift registers. This does what we do with long division, clocking in one new bit each cycle.

CRC-16-IBM: $g(D) = 1 + D^2 + D^{15} + D^{16}$



Every delay is a power of 2.

As each new bit is clocked in:

- Bits 0, 2, 15 are xored with the previous output, bit 16
- Everything shifts right
- Repeat for each new bit.

CRC Implementation, (con't)

At Transmit:

- Whole packet of bits is input serially to the generator
- The state at the end of the packet is: $\mathbf{b} = (b_1, b_2, \cdots b_{16})$
- Transmit 16 CRC bits at the end of the packet as a designated part (link layer)

On Receive:

- Decode the packet
- Run the decoded bits through the $g(D)$, the CRC generator
- Compare against CRC part of the packet.

CRC Implementation, (con't)

Lots of practical subtleties

Bits over the air are generally least significant bit first (LSB)

However, the CRC:

- Is often just protects the message itself, not the packet
- Usually computed using the most significant bit first (MSB)
- May only include part of the message
- May be applied after the message has been decoded (not include parity bits, for example)

Sometimes there is an initial value set for the register to start things off

Sometimes the result is xored with a specified value at the end

Can by tedious to debug.

CRC Analysis

Miss Detection

There are errors, but CRC checks out

For a K bit CRC code(CRC-K):

$$\begin{aligned} P[\text{miss}] &\approx 2^{-K} \\ &\approx 2^{-16} = 5 \times 10^{-5} \end{aligned}$$

All ethernet standards use a 32 bit CRC at the hardware level

$$\Rightarrow P[\text{miss}] \approx 2^{-32} = 2.3 \times 10^{-10}$$

If an error is missed, then packet is assumed correct.

This is obviously a problem given current files sizes.

CRC Analysis

False Alarm

No packet errors, but CRC bits have errors

For CRC-K with a $P_e = 10^{-5}$

$$\begin{aligned} P[\text{False alarm}] &= 1 - P[\text{all received are correct}] \\ &= 1 - (1 - P_e)^K \approx KP_e \\ &= -(1 - 10^{-5})^{16} \\ &= 1.6 \times 10^{-4} \end{aligned}$$

False alarm often leads to retransmission

Large K, exponential decrease in miss detection, but linear increase in false alarms

Usually detecting errors has priority

Error Correction

So far we've been focusing on detecting errors. That is valuable, but it generally leads to retransmissions

Ideally we'd like to correct errors

To do this we need to add extra information. We'll look at two approaches

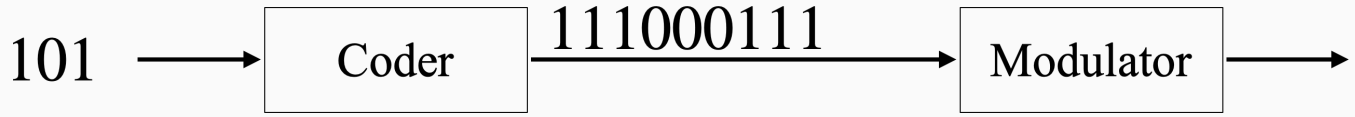
- Convolutional codes: extra bits are added to the bit stream
- Block codes: extra bits are added to each block of data

Repetition Codes

For forward error correction, the idea is to transmit extra bits

These are described by their rate $r = m/n$, where m is the number of input bits that are used to produce n output bits

A simple code repeats the same symbol m times, give a $r = 1/3$ code



Then if there is an error, take the majority vote

$$101 \to 1, \quad 001 \to 0$$

Very simple. Used in early deep space probes where bandwidth was abundant.

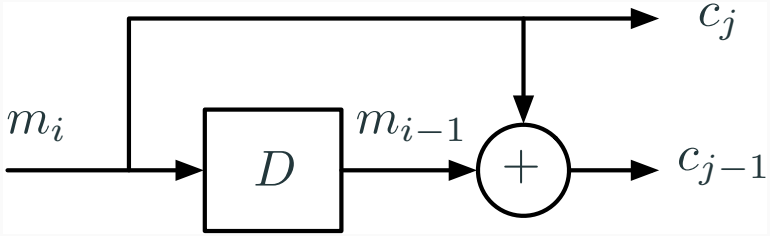
Improves SNR and reduces probability of error, but triples the bandwidth.

Convolutional Codes

For convolutional codes, a filter adds additional bits to the bit stream

Each n-bit codeword block depends on the current codeword and on the previous m codewords. m is the memory order

This is a rate $1/2$ convolutional code, $m = 1$ and $n = 2$ used in GSM:



This generates two output bits for every input bit, the current data bit, and the XOR with the previous bit.

Convolutional codes are widely used in digital communications. They don't need a fixed block size. Examples are Viterbi codes.

Convolutional Codes

A set of modulo two filters convolved with input stream

- Add correlation between output bits
- Correlation can be used to correct errors at the receiver

A sliding parity bit calculation

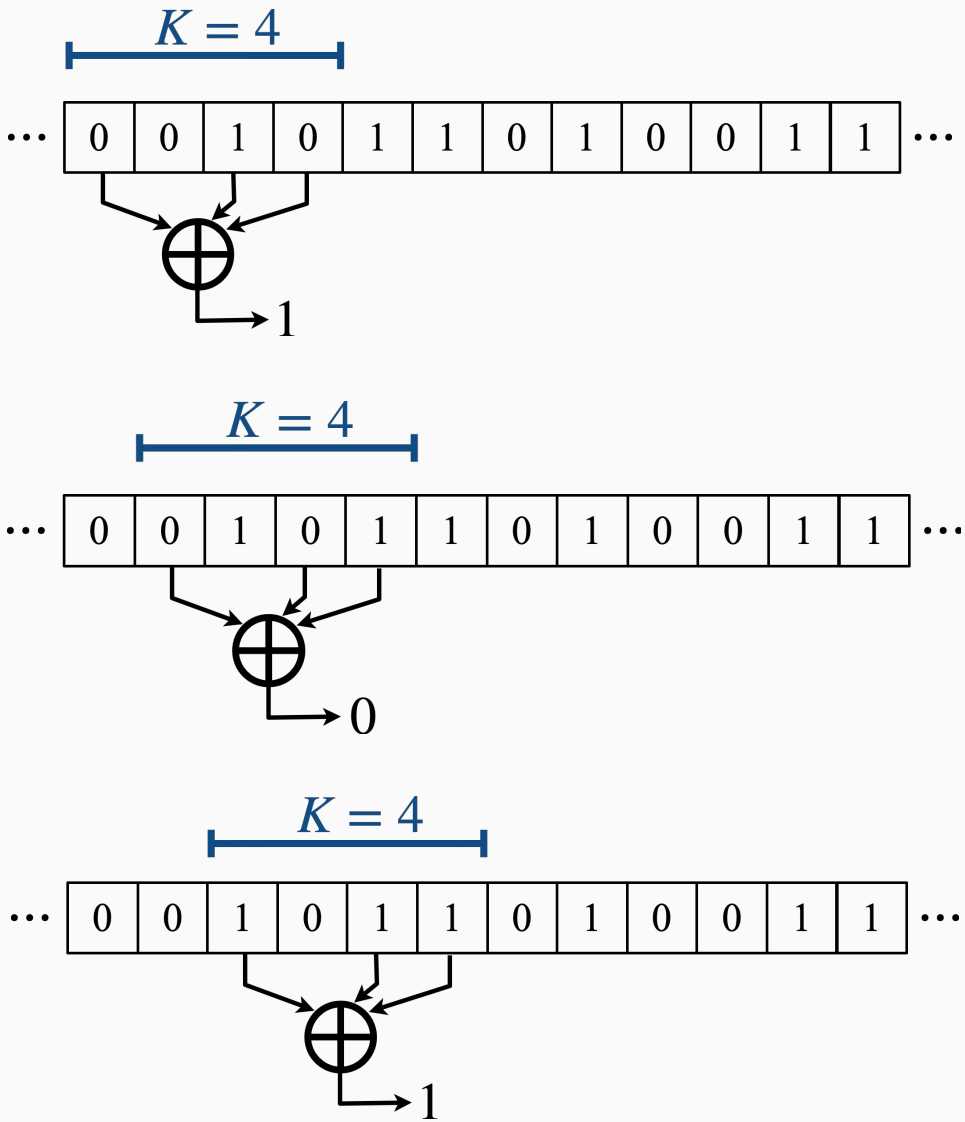
- Transmit only the parity bits

Consider the case where

- We use a sliding window of the last four input bits
- We compute two output bits for each input bit

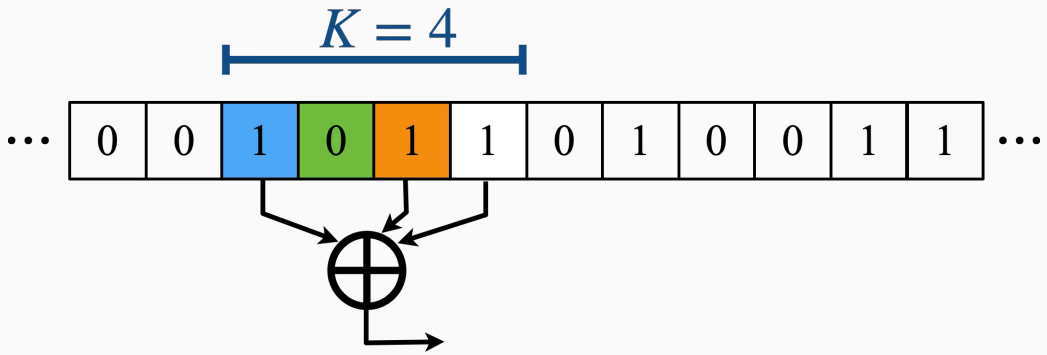
Convolutional Code Mechanics

For the first parity equation

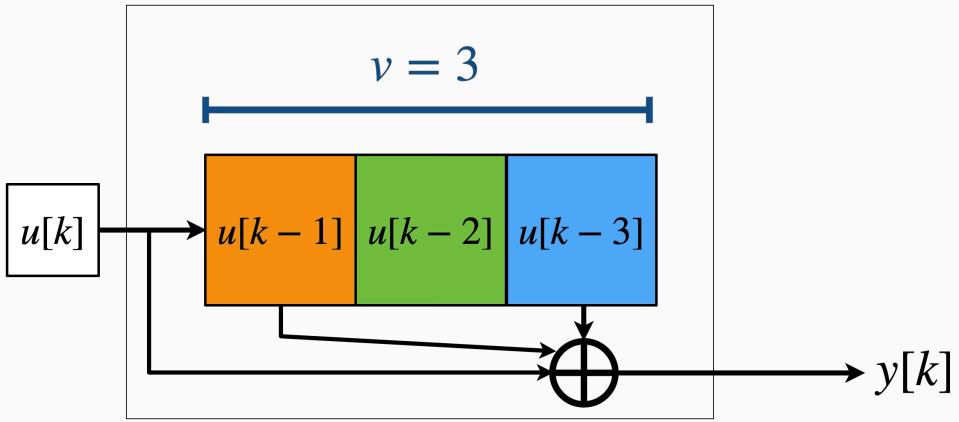


Convolutional Codes

The discrete convolution



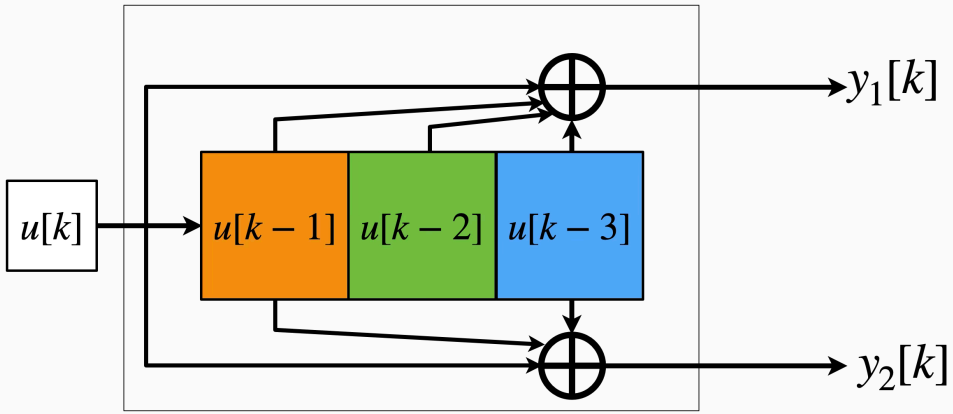
Can be implemented as a shift register



This gives us one of the two output bits for each input bit

Convolutional Codes

We'll also include a second, different polynomial to give a second set of bits



What we transmit is then

$$[\cdots\cdots y_1[k-1], y_2[k-1], y_1[k], y_2[k], y_1[k+1], y_2[k+1], \cdots\cdots]$$

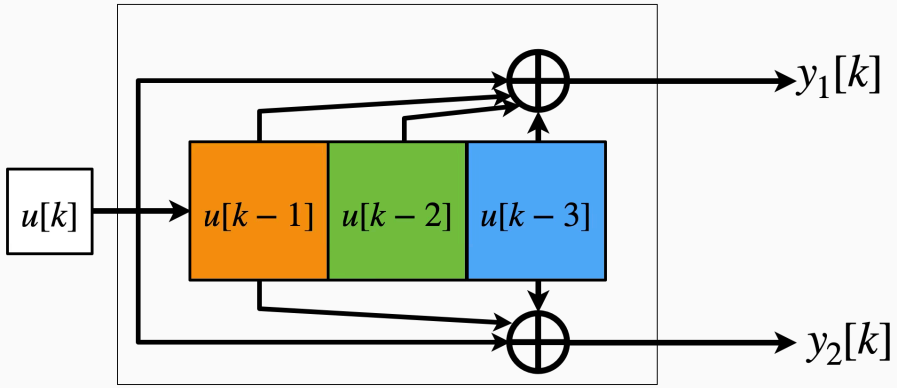
This doubles the number of bits transmitted

However, each bit depends on multiple input bits, so if there is an error in detection, I have some hope of detecting and correcting it.

Convolutional Codes: Encoding Example

Compute the parity equations for each input

k	$u[k]$	$s[k]$	$y_1[k]$	$y_2[k]$
0	1	(0,0,0)	1	1
1	0	(1,0,0)	1	1
2	1	(0,1,0)	0	1
3	1	(1,0,1)	1	1
4	0	(1,1,0)	0	1
5	1	(0,1,1)	1	0



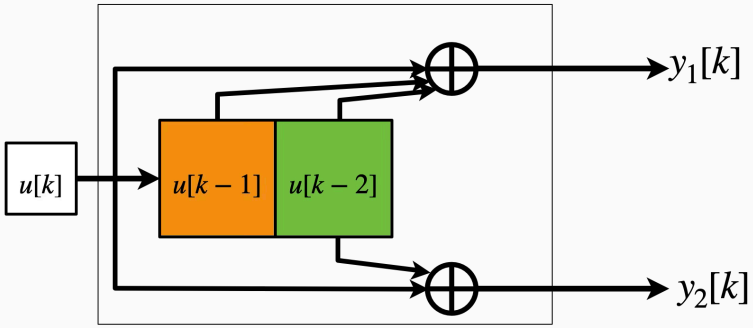
$$g_{11}(D) = 1111_{(2)}$$
$$g_{12}(D) = 1101_{(2)}$$

$$[u[k]\ s[k]] \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} = [y_1[k]\ \ y_2[k]]$$

Convolutional Codes: Decoding Example

For each possible input, find the output that is closest to the actual received bits

Message	Coded bits	Distance
“000”	“000000”	4
“001”	“000011”	4
“010”	“001110”	1
“011”	“001101”	3
“100”	“111011”	3
“101”	“111000”	3
“110”	“110101”	4
“111”	“110110”	2



received: $\hat{y} = 101110$

Q: Which 3 bits were transmitted?

A: Compute hamming distance:

$$d_{\text{hamming}} = \sum_{i=0}^{5} \hat{y}[i] \oplus y[i]$$

Convolutional Codes: Decoding Example

Decoding becomes more difficult as the number of bits increases,

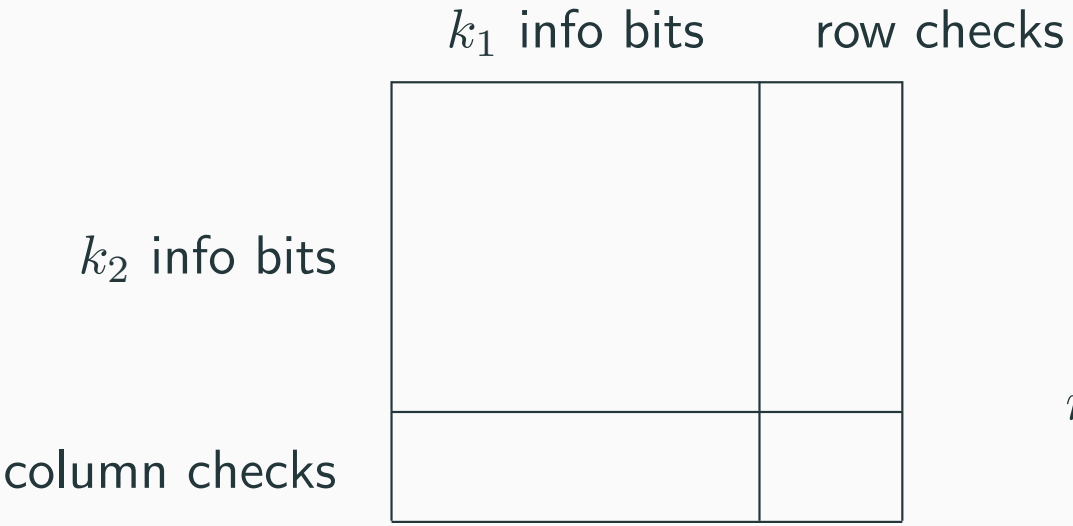
Depends on the previously output bits, which may be in error

Recursive algorithms (Vitterbi, and many variants) very efficiently compute the most likely input sequence for a given output

Widely used, particularly for satellite communications and cell phones

Error Correction: Simple Product Codes

Arrange data bits in a two-dimensional array. Append parity check bits at the end of each row and column.



$$\begin{aligned} n &= (k_1+1)(k_2+1) \\ k &= k_1 k_2 \\ n-k &= k_1 + k_2 + 1 \end{aligned}$$

Single error causes failure of one row equation and one column equation. Incorrect bit is located at the intersection of the bad row and bad column.

Double errors can be detected — two rows or two columns (or both) have the wrong parity — but cannot be corrected.

Some triple errors cause miscorrection. Which?

Error Correction: Hamming Codes

Simple product codes are simple but inefficient:

- a failed parity-check equation locates row or column of error
- however, a satisfied equation gives little information

An "efficient" equation gives one bit of information about the error location. It "looks" at half the codeword bits and is "independent" of other equations.

Basic idea:

- We will send a block of size $n = 2^m - 1$
- We'll use m bits for parity, and the rest for data
- Each parity bit corresponds to half of the block
- A single error and its location can be identified by which parity equations fail. This pattern is called the *error syndrome*

Hamming Codes

To determine the parity equations,

- Index each output bit from one to $2^n - 1$, and represent it in binary
- Any index that has a single bit is used for parity (i.e. 00100)
- All other indexed outputs are sequentially filled with input bits
- The parity bit is computed for all indexes that have the same index bit set (i.e all the odd samples for the first parity bit)

Example: $2^n - 1 = 7$

- Parity bits are 001, 010, and 100
- Data bits are the rest, 011, 101, 110, and 111
- We'll have 3 parity bits, 4 data bits, and seven bits total
- This is called a (7,4) code

Hamming Codes

- The following table defines a $(7, 4)$ Hamming parity-check code.

	c_1	c_2	c_3	c_4	c_5	c_6	c_7
	p_1	p_2	d_1	p_4	d_2	d_3	d_4
p_1	1	0	1	0	1	0	1
p_2	0	1	1	0	0	1	1
p_4	0	0	0	1	1	1	1

} 3 parity-check equations

- Each column is the binary representation of that index
- Each row defines one parity bit equation
- The 1's indicate which codeword bits affect which parity-check equations.
- We can double the block size and only need one more parity bit, for a (15,11) code. This gets very efficient as n gets large.

Hamming Code Error Detection and Correction

The parity equations are

$$c_1 \oplus c_3 \oplus c_5 \oplus c_7 = 0$$
$$c_2 \oplus c_3 \oplus c_6 \oplus c_7 = 0$$
$$c_4 \oplus c_5 \oplus c_6 \oplus c_7 = 0$$

where $c_1 = p_1$, $c_2 = p_2$, and $c_4 = p_4$ are parity bits.

Parity bit error: Each parity bit occurs in just one equation, so only that equation will have a parity error. A single parity equation error is an error in that parity bit

Data bit error: Each data bit occurs in multiple equations, each of which will have a parity error. The data bit error is the bit shared by those multiple equations

Either way, a single bit error can be detected and corrected.

Hamming Code Error Detection and Correction

For larger blocks, we just add more parity bits.

For example, the parity-check matrix for (15,11) is

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

- There are now four parity equations, with parity bits p_1, p_2, p_4, and p_8
- The data bits appear in 2, 3, or 4 equations.
- A single parity equation error is an error in the parity bit
- Multiple parity equation errors is an error in the bit shared by these equations

All single bit errors can be detected and corrected.

Error Detection Conclusion

A single overall parity-check equation detects single errors.

Hamming codes use m equations to correct one error in $2^m - 1$ bits.

- Most useful when the error rate is low. ECC RAM uses Hamming codes
- More sophisticated algorithms needed when multiple errors likely in a block

Extensions

- We can use nonbinary equations if we create *symbols* from sequences of bits. E.g., four bits can represent $0, 1, \ldots, 15$.
- Nonbinary check equations can use more advanced arithmetic, such as mod 16.
- We can add other types of equations, $c_1 + 2c_2 + 3c_3 + \cdots$ and $c_1 + \alpha c_2 + \alpha^2 c_3 + \cdots$ to be able to detect and correct multiple bit errors.

Conclusion

Noise and errors are a given in communications

The theoretical limit is the *Shannon Capacity*

$$C = B \log_2 (1 + SNR)$$

where C is the capacity in bits/s.

Published by Claude Shannon in 1948.

With modern error correction, there are many systems that operate with 0.5 dB of this theoretical limit.

This is pretty astonishing!