

3F4: Data Transmission

Handout 4: Properties of Convolutional Codes and Turbo Codes

Jossy Sayir

Probability, Systems, Information and Inference Lab Ψ^2
Department of Engineering
js851@cam.ac.uk

Lent Term 2026

1 / 17

Warning: Catastrophic encoders

Consider the $(6, 5)_8$ encoder shown,
with input $\underline{u} = 11111111 \dots$

The code sequence generated is

$$\underline{x} = 11 \mathbf{01} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \dots$$

Suppose the received sequence is

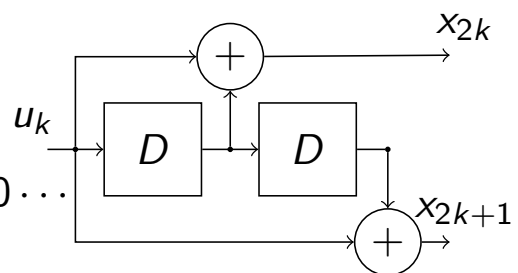
$$\underline{y} = 11 \mathbf{10} \mathbf{01} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \mathbf{00} \dots$$

This is itself a codeword, corresponding to source string

$$\underline{u}' = \mathbf{100000} \dots$$

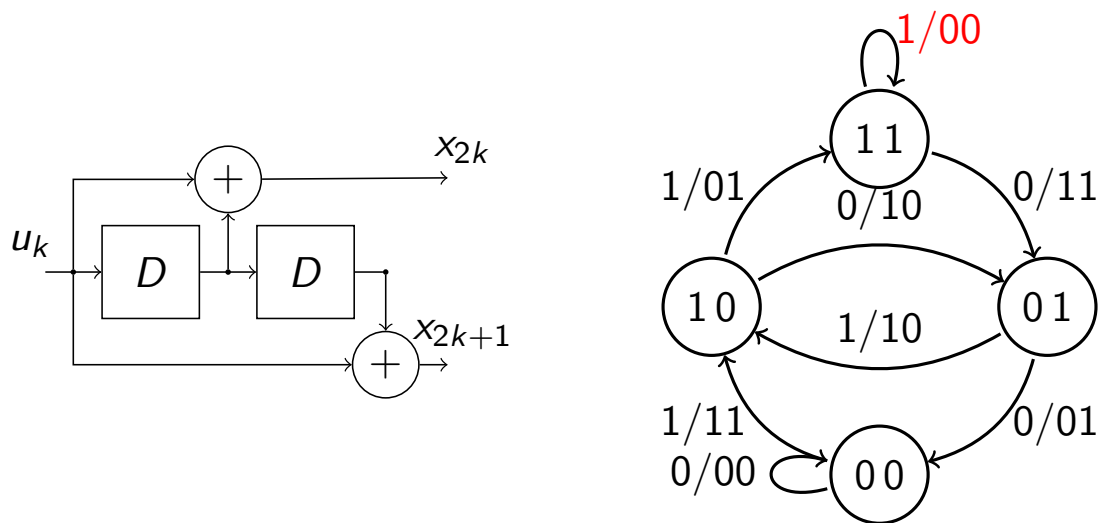
\Rightarrow A finite number of channel errors (only three, in fact)
led to infinitely many decoding errors!

Encoders with this very undesirable property are called *catastrophic*



2 / 17

What made this encoder catastrophic?



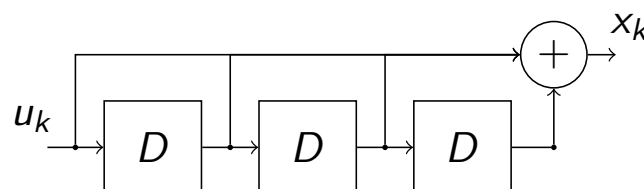
The root of the problem is that we can go from state 11 to state 11, with an input bit of weight 1 and an output string 00 that has weight 0!!

but beware! Not all catastrophic encoders are so easy to identify...

3 / 17

Catastrophic Encoders

Consider the rate $R = 1$ convolutional encoder



- The infinite weight input sequence $u = 1, 1, 1, \dots$ results in the weight 2 output sequence $x = 1, 0, 1, 0, 0, 0, 0, \dots$. How did we know this?
- Encoding of the sequence $u = 1, 1, 1, \dots$ can be expressed in the D domain as

$$\begin{aligned} X(D) &= c(D)u(D) \\ &= (1 + D + D^2 + D^3)(1 + D + D^2 + D^3 + D^4 + \dots) \end{aligned}$$

- Noting that $c(D) = (1 + D)(1 + D^2)$ and $u(D) = \frac{1}{1+D}$ (sum of a geometric sequence), we have

$$X(D) = \frac{(1 + D)(1 + D^2)}{1 + D} = 1 + D^2$$

4 / 17

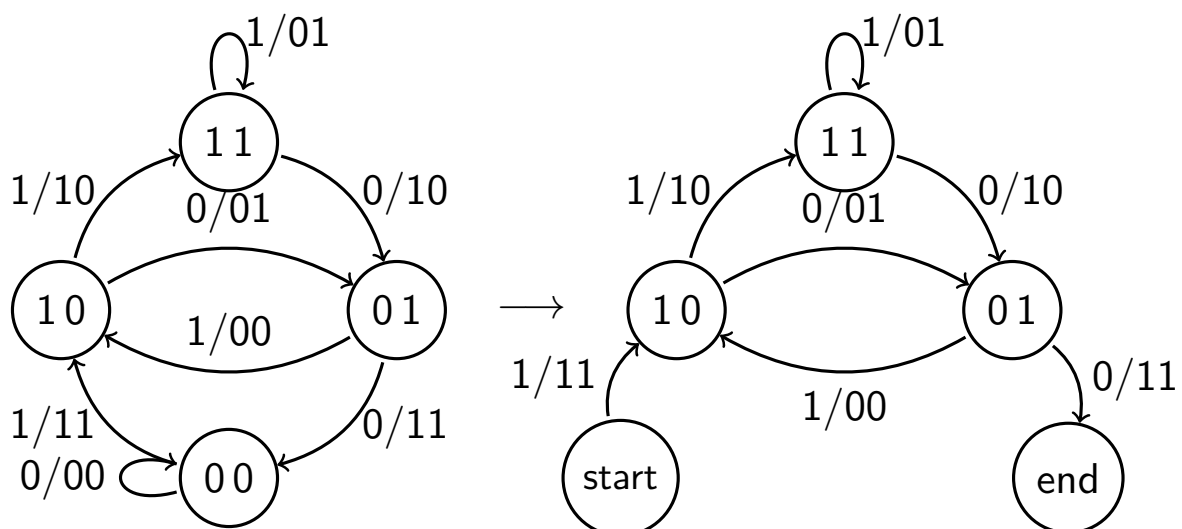
The significance of detours

- Clearly the performance of the Viterbi algorithm and/or BCJR depends on the number and properties of detours in the trellis
- We would like to answer questions such as:
 - what is the detour of minimum output (code sequence) weight? (this is known as the **free distance** d_{free})
 - how many input bits would be decoded wrong if a codeword on a detour was selected?
 - how many detours of weights $d = 1, 2, 3, 4, \dots$ are there, how many stages does each have, and what are the weights of the corresponding input sequences?
- All questions are concerned with the scenario where the all-zero codeword is transmitted and the noise pushes us to consider detours from the transmitted all-zero path
- Since the code is linear and the channel is a BSMC, measuring decoder performance when the all-zero codeword is transmitted is sufficient: the performance for any other transmitted codeword is the same

7 / 17

Counting detours

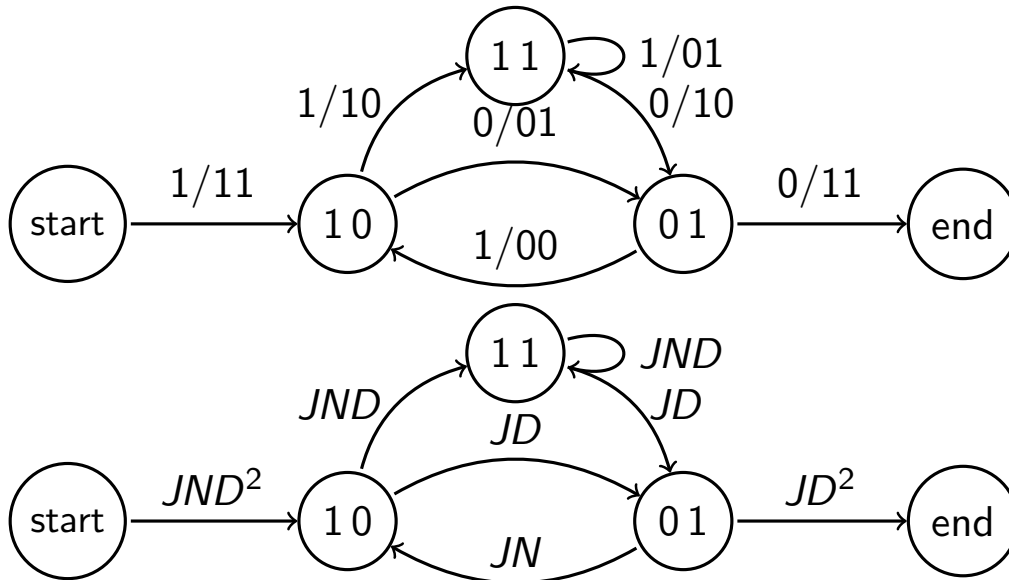
Consider the state diagram of the $(5, 7)_8$ encoder with the zero state split into a “start” and “end” state (the start and end of a detour):



8 / 17

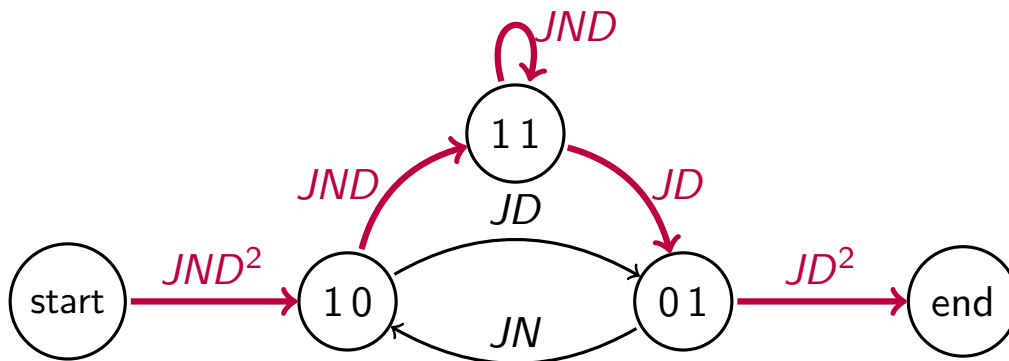
Counting detours with dummy variables J, N, D

- We will (once again) harness the power of polynomials (actually, multinomials in this case) to count detours
- Label each branch with $JN^{w_i}D^{w_o}$ where w_i and w_o are the weights of the input bit and output bits along the branch, respectively



9 / 17

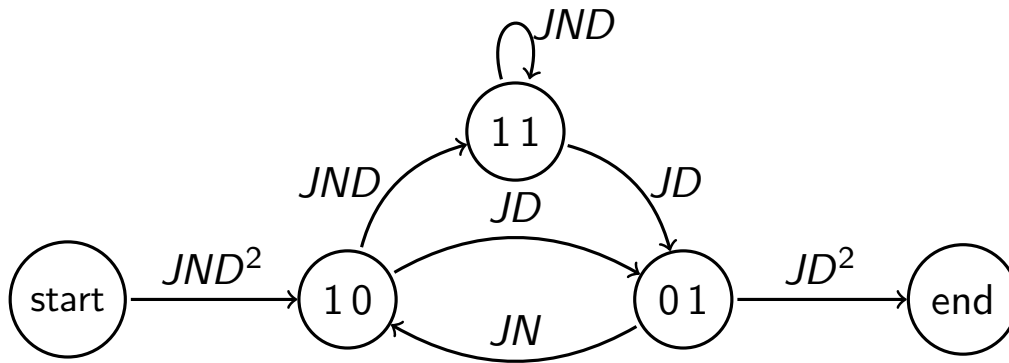
Counting detours with dummy variables J, N, D



- Consider the highlighted detour assuming that we go n times around the self-loop in state 11
- The product of the branch labels along this path is $J^4 N^2 D^6 (JND)^n$
- These paths comprise $4 + n$ trellis stages, corresponding to input sequences of weight $2 + n$ and output sequences of weight $6 + n$
- The degrees of the multinomials answer all our questions
- We need the **graph transfer function** $T_{\text{start} \rightarrow \text{end}}(J, D, I)$

10 / 17

Computing the transfer function



- Define a variable for every state and set up the system of equations corresponding to the graph

$$\begin{cases} x_{10} &= JND^2 x_{\text{start}} + JN x_{01} \\ x_{11} &= JND x_{11} + JND x_{10} \\ x_{01} &= JD x_{10} + JD x_{11} \\ x_{\text{end}} &= JD^2 x_{01} \end{cases}$$

- Solve (analytically!) and determine $T(J, N, D) = \frac{x_{\text{end}}}{x_{\text{start}}}$

11 / 17

Converting back to a power series

- In the example,

$$T(J, N, D) = \frac{J^3 ND^5}{1 - (1 + J)JND}$$

- This is a tedious calculation! (I'll soon teach you a trick to make it easier...)
- Using the sum of a geometric sequence

$$\begin{aligned} T(J, N, D) &= J^3 ND^5 (1 + \alpha + \alpha^2 + \dots) \text{ where } \alpha = (1 + J)JND \\ &= J^3 ND^5 + J^4 N^2 D^6 + J^5 N^2 D^6 + J^5 N^3 D^7 \\ &\quad + 2J^6 N^3 D^7 + J^7 N^5 D^7 + J^6 N^4 D^8 + \dots \end{aligned}$$

- The expression says:
 - There is one path of 3 inputs (including one 1) of output weight 5 (hence $d_{\text{free}} = 5$)
 - There is a path of 4 inputs (incl. two 1s) of output weight 6
 - etc...

12 / 17

Summary so far

- Split the zero state into a start and end state
- Set up a system of equations and solve analytically to obtain $T(J, N, D)$
- Convert back to power series using partial fractions, sum of geometric sequence or **polynomial division**
- If only interested in output weight, you can ignore J and N and evaluate

$$T(D) = T(1, 1, D) = D^5 + 2D^6 + 4D^7 + \dots$$

i.e., there is one detour of weight 5, two of weight 6, 4 of weight 7, etc.

- You can also compute the transfer function $T(D)$ directly labeling branches only with D^{w_o} but **be careful**: this doesn't always work (e.g. if there is a weight zero self-loop)
- Transfer function can be used to bound the error probability, such as the for the Viterbi algorithm on the BIAWGN,

$$P_e \leq \sum_k T(D)|_{c_k} Q\left(\sqrt{2k\text{SNR}}\right)$$

13 / 17

Mason's gain formula: how to avoid messy equations

- The transfer function formula is

$$T = \frac{x_{\text{out}}}{x_{\text{in}}} = \frac{\sum_{k=1}^N G_k \Delta_k}{\Delta}$$

- Δ is the graph determinant

$$\Delta = 1 - \sum_i L_i + \sum_{i,j} L_i L_j - \sum_{i,j,k} L_i L_j L_k + \dots$$

- G_k is the gain of the k -th forward path (from in to out)
- L_i is the gain of the i -th loop
- $L_i L_j$ is the product of the **non-intersecting** loops i and j
- loops are said to intersect if they have at least one common node
- Δ_k is the determinant of the graph with loops intersecting with the k -th forward path removed

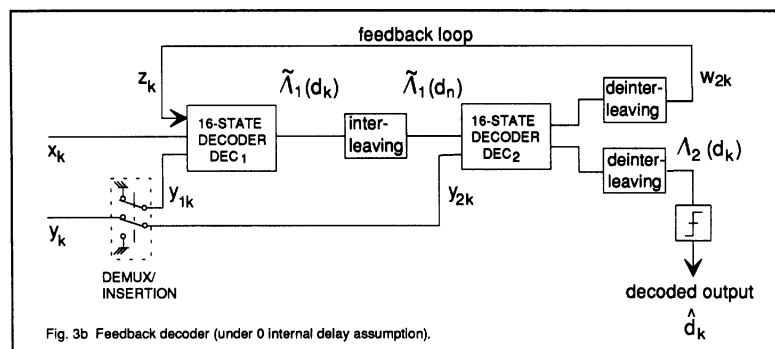
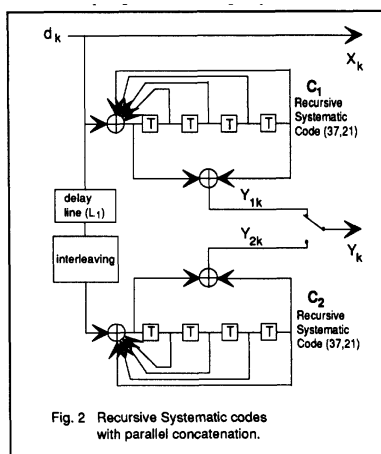
14 / 17

Mason's gain formula: background

- This rule is derived from Cramer's rule to invert a matrix using the determinant and co-factors
- It is useful when computing the transfer function of a **signal flow graph** with not too many non-intersecting loops
- Signal flow graphs have applications in control theory (block diagrams), coding (convolutional coding) and electronics (circuit analysis) and possibly in other fields

15 / 17

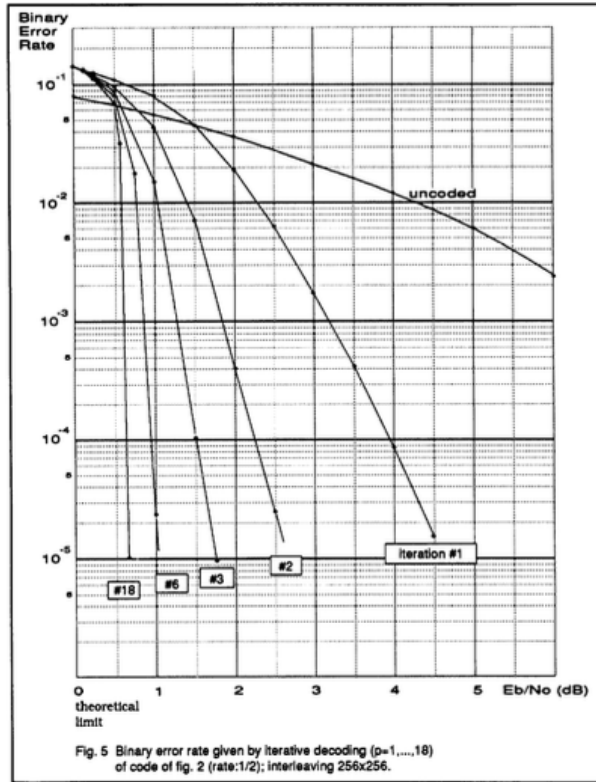
Turbo Codes



From: "Near Shannon limit error-correcting coding and decoding: Turbo-codes" by C. Berrou & al., ICC 1993.

16 / 17

Turbo Codes



From: "Near Shannon limit error-correcting coding and decoding: Turbo-codes" by C. Berrou & al., ICC 1993.