

## Convolutional Codes

is  
in Block Codes' data encoded using a combinational circuit. That is, a circuit with only logic gates and no memory.

Convolutional Codes on the other hand, have encoders that are sequential circuits.

A Convolutional encoder receives  $k$  bits as the input and generates  $n \geq k$  output symbols based on the input at a given time and past inputs (or outputs) still in the memory. A convolutional code usually has  $m$  memory units resulting in  $2^m$  states.  $m$  is called the constraint length. Some times  $m$  is called the constraint length taking into account the present <sup>input</sup> and  $m$  bits in the memory as the bits affecting the output.  $k$  and  $n$  are usually small integers. Of particular interest are the codes with  $k=1$  resulting in code rate  $\frac{1}{n}$ .

8-1

For example, if  $k=1$  and  $n=2$ , we have a code of rate  $\frac{k}{n} = \frac{1}{2}$ .

It is important to note the fact that while  $k=1$  or 2 or some other small number, the input and output are streams of bits.

Assume that  $L$  symbols enter the encoder.

This means  $kL$  input <sup>bits</sup> and  $nL$  output bits

we need also to flush the encoder to make it ready for next block of data, e.g., by feeding  $km$  bits. So, the output will actually be  $n(L+m)$  bits long and the rate is

$$\frac{kL}{n(L+m)} = \frac{k}{n} \cdot \frac{L}{L+m} \rightarrow \frac{k}{n}$$

when  $m \ll L$ .

---

Since the codewords of a convolutional code are generated using a Finite State Machine (FSM), i.e., a sequential circuit, the decoder can be a scheme that finds the best match for the received sequence (based on minimum distance), by going

through all possible outputs the FSM (the encoder could have generated). The scheme used is the travelling salesman algorithm. It is called the Viterbi Algorithm (VA) in coding literature as it was first used for decoding of Convolutional Codes by Andrew Viterbi (the relationship between VA and travelling salesman problem was later discovered. So, in fact, Viterbi re-invented the algorithm).

VA finds a solution that is optimal over the whole received sequence and not necessarily having lower probability of error for each symbol.

Another scheme called BCJR (Bahl, Cocke, Jelinek and Raviv) algorithm is another decoding technique that works based on the maximum a posteriori (MAP) probability taking into account the a priori probability of bits.

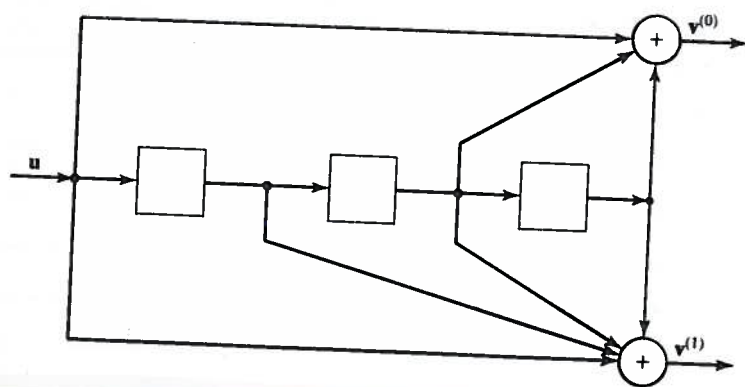
It is very useful in <sup>iterative</sup> decoding of Turbo Codes.

## Example

Let's start with a non-Systematic  
Feed forward rate  $\frac{1}{2}$  Code.

The code is non-Systematic as there is no direct connection between the input and any of the outputs.

Also, we call the code feed forward as the outputs are not fed back to the input side.



Let the input sequence to be  $\underline{u} = (u_0, u_1, u_2, \dots)$

The output sequences will be

$$\underline{v}^{(0)} = (v_0^{(0)}, v_1^{(0)}, v_2^{(0)}, \dots),$$

and

$$\underline{v}^{(1)} = (v_0^{(1)}, v_1^{(1)}, v_2^{(1)}, \dots).$$

These outputs can be obtained by convolving  
 $\underline{u}$  with the impulse responses of two branches.

To find the impulse response let

$$\underline{u} = (1 \ 0 \ 0 \ \dots)$$

and observe the outputs. Since the memory order is  $m$ , the impulse response can last at most  $m+1$  time units. So,

$$\underline{g}^{(0)} = (g_0^{(0)}, g_1^{(0)}, \dots, g_m^{(0)})$$

and

$$\underline{g}^{(1)} = (g_1^{(1)}, g_2^{(1)}, \dots, g_m^{(1)})$$

For this example

$$\underline{g}^{(0)} = (1 \ 0 \ 1 \ 1)$$

and

$$\underline{g}^{(1)} = (1 \ 1 \ 1 \ 1)$$

These are also called generator sequences.

Now,

$$\underline{v}^{(0)} = \underline{u} \otimes \underline{g}^{(0)}$$

and

$$\underline{v}^{(1)} = \underline{u} \otimes \underline{g}^{(1)}$$

where  $\otimes$  denotes convolution:

$$v_l^{(j)} = \sum_{i=0}^m u_{l-i} g_i^{(j)} = u_l g_0^{(j)} + u_{l-1} g_1^{(j)} + \dots + u_{l-m} g_m^{(j)}$$

for  $j=0$  and  $1$ .

For this example,

$$v_l^{(0)} = u_l + u_{l-2} + u_{l-3}$$

$$v_l^{(1)} = u_l + u_{l-1} + u_{l-2} + u_{l-3}$$

and

$$\underline{v} = (v_0^{(0)}, v_0^{(1)}, v_1^{(0)}, v_1^{(1)}, v_2^{(0)}, v_2^{(1)}, \dots)$$

Assume  $\underline{u} = (10111)$

then

$$\underline{v}^{(0)} = (10111) \otimes (1011) = (10000001)$$

$$\underline{v}^{(1)} = (10111) \otimes (1111) = (1101101)$$

and

$$\underline{v} = (11, 01, 00, 01, 01, 01, 00, 11)$$

When the number of bits encoded is large, we can view the operation of the convolutional encoder as a block encoder by defining the generator matrix:

matrix:

$$G = \begin{bmatrix} g_0^{(0)} & g_0^{(1)} & g_1^{(0)} & g_1^{(1)} & g_2^{(0)} & g_2^{(1)} & \dots & g_m^{(0)} & g_m^{(1)} \\ & g_0^{(0)} & g_0^{(1)} & g_1^{(0)} & g_1^{(1)} & \dots & g_{m-1}^{(0)} & g_{m-1}^{(1)} & g_m^{(0)} & g_m^{(1)} \\ & & g_0^{(0)} & g_0^{(1)} & \dots & g_{m-2}^{(0)} & g_{m-2}^{(1)} & g_{m-1}^{(0)} & g_{m-1}^{(1)} & g_m^{(0)} & g_m^{(1)} \\ & & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix}$$

Then  $\underline{v} = \underline{u} G$ .

For input

$\underline{u} = (10111)$ , we

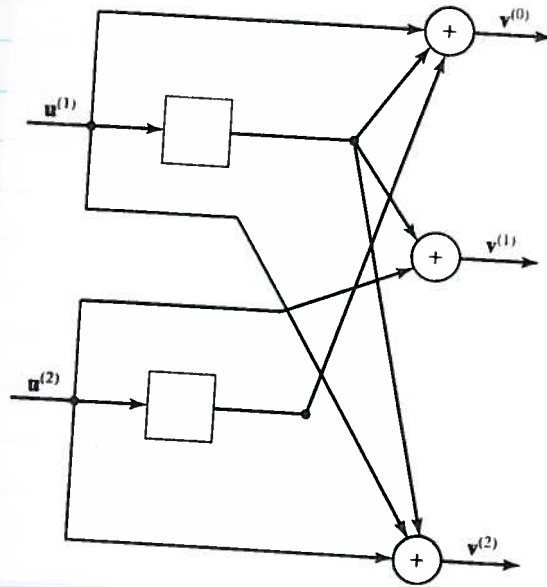
have  $\rightarrow$

$$\underline{v} = \underline{u} G$$

$$= (10111) \begin{bmatrix} 11 & 01 & 11 & 11 \\ & 11 & 01 & 11 & 11 \\ & & 11 & 01 & 11 & 11 \\ & & & 11 & 01 & 11 & 11 \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

$$= (11, 01, 00, 01, 01, 01, 00, 11),$$

Example: A Rate  $\frac{2}{3}$  Non Systematic  
Feed forward Convolutional Code:



Let  $g_i^{(j)}$  represent the generator sequence corresponding to input  $i$  and output  $j$ , we have:

$$\begin{aligned} g_1^{(0)} &= (1 \ 1) & g_1^{(1)} &= (0 \ 1) & g_1^{(2)} &= (1 \ 1), \\ g_2^{(0)} &= (0 \ 1) & g_2^{(1)} &= (1 \ 0) & g_2^{(2)} &= (1 \ 0), \end{aligned}$$

Then:

$$\begin{aligned} v^{(0)} &= u^{(1)} \otimes g_1^{(0)} + u^{(2)} \otimes g_2^{(0)} \\ v^{(1)} &= u^{(1)} \otimes g_1^{(1)} + u^{(2)} \otimes g_2^{(1)} \\ v^{(2)} &= u^{(1)} \otimes g_1^{(2)} + u^{(2)} \otimes g_2^{(2)}. \end{aligned}$$

So,

$$\begin{aligned} v_l^{(0)} &= u_l^{(1)} + u_{l-1}^{(1)} + u_{l-1}^{(2)}, \\ v_l^{(1)} &= u_l^{(2)} + u_{l-1}^{(1)}, \\ v_l^{(2)} &= u_l^{(1)} + u_l^{(2)} + u_{l-1}^{(1)}, \end{aligned}$$

While <sup>rate</sup>  $\frac{k}{n}$  Codes with  $k \neq 1$  can be used, it is easier to use a Code  $\frac{1}{n}$  rate to generate Codes  $\frac{k}{n}$ . This is done using puncturing.

Assume that we have a rate  $\frac{1}{2}$  Code, we can feed it two bits at a time and out of 4 bits we get at the output through out one to get a rate  $\frac{2}{3}$  Code. Or input 3 bits and get 6 bits out and through out 2 bits to get  $\frac{3}{4}$  Code.

Example: Industry Standard Code

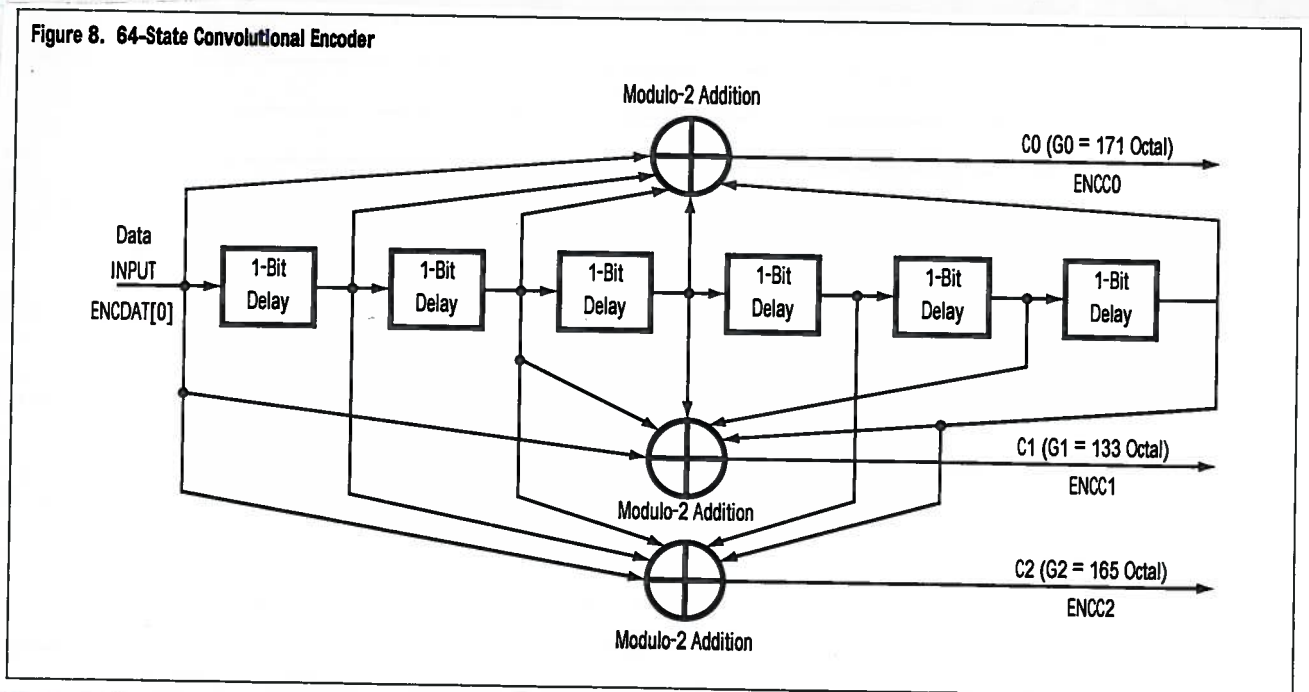




Figure 9a. Punctured Coding for Rate 3/4

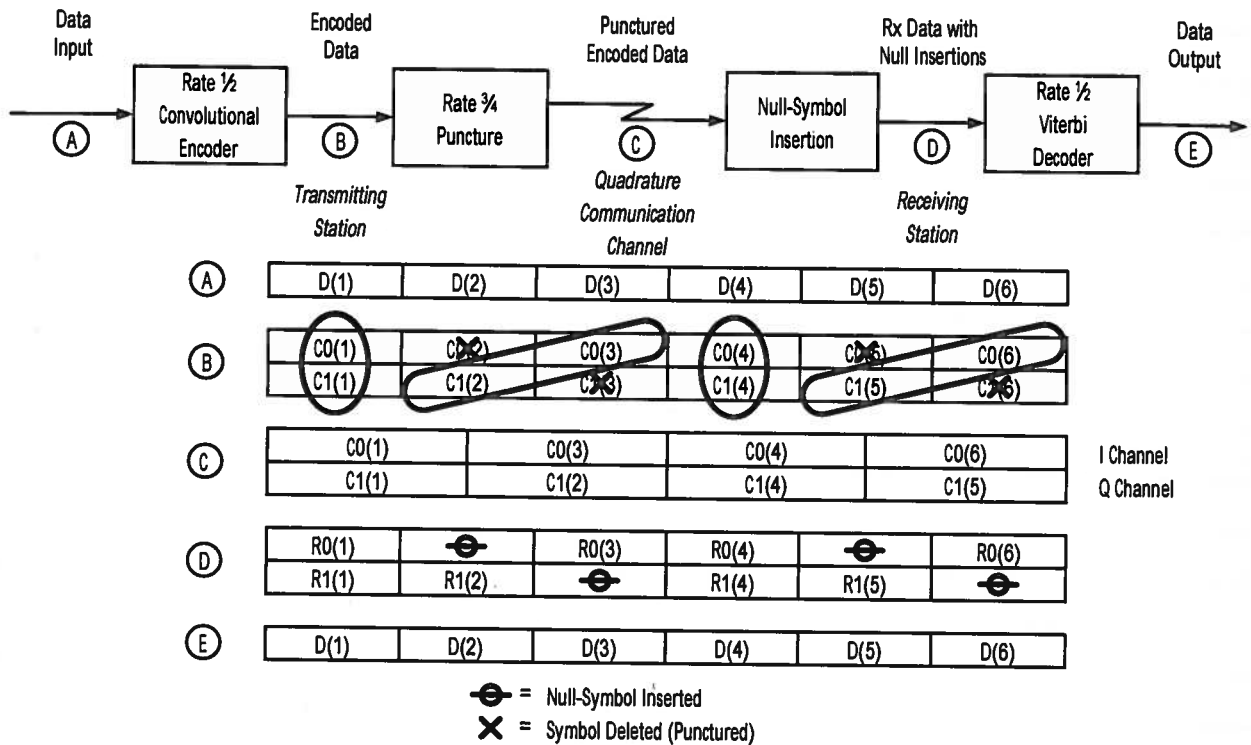
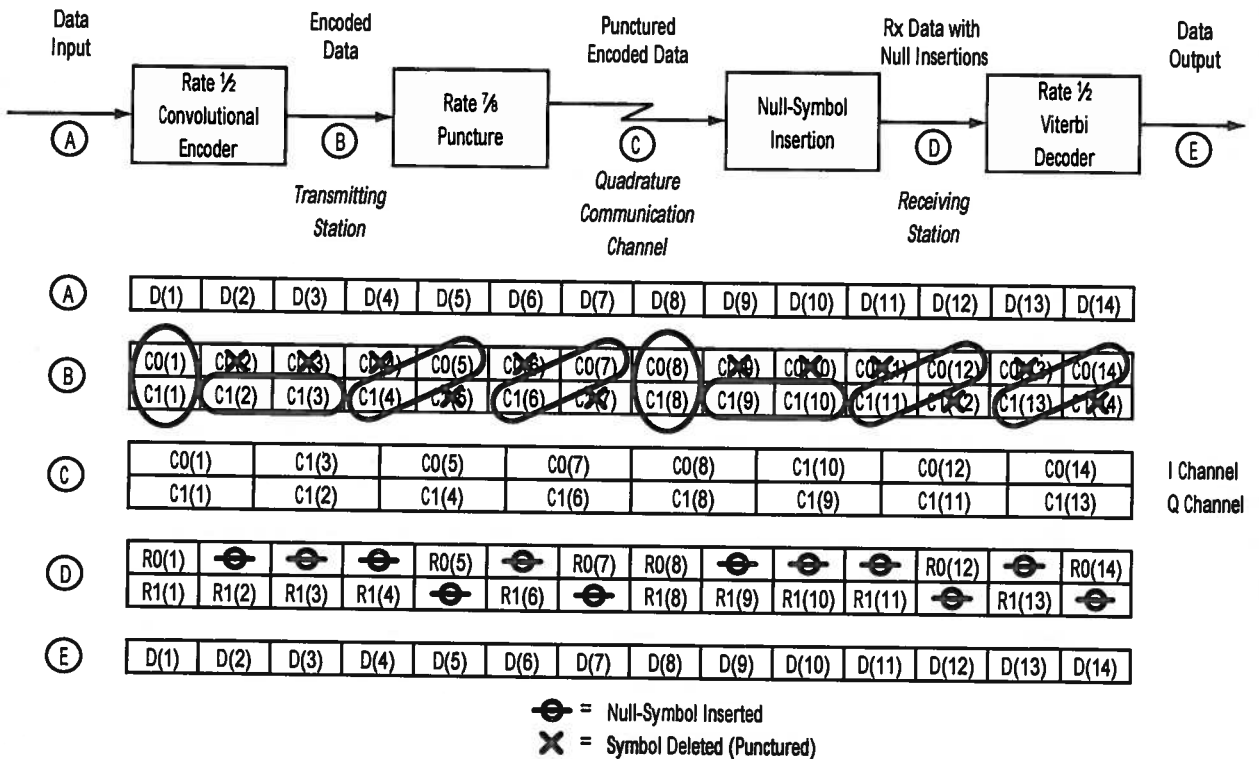


Figure 9b. Punctured Coding for Rate 7/8

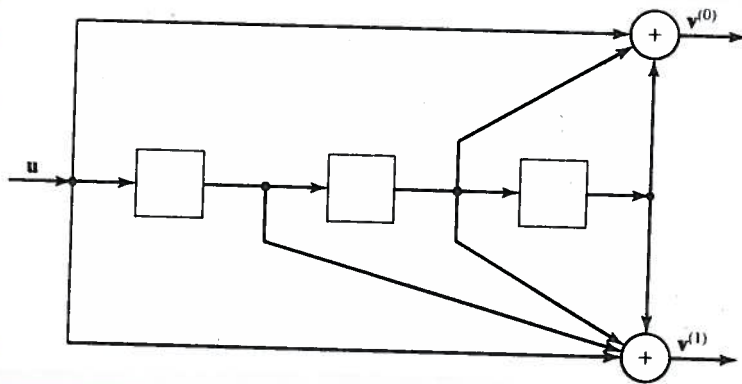


## Polynomial Domain representation:

Similar to block codes, we can transform the time domain representation by transforming the generating sequences to generating polynomials and similarly input and output sequences to input and output polynomials.

For example for the (2,1,3) Code, i.e., rate  $\frac{1}{2}$  memory = 3 Code, instead of  $g^{(0)} = (1011)$  and  $g^{(1)} = (1111)$ , we can use:

$$g^{(0)}(D) = 1 + D^2 + D^3 \quad \text{and} \quad g^{(1)}(D) = 1 + D + D^2 + D^3$$



If we denote the input sequence as,

$$u(D) = u_0 + u_1 D + u_2 D^2 + \dots$$

Then  $v^{(0)}(D) = u(D) g^{(0)}(D)$

and,  $v^{(1)}(D) = u(D) g^{(1)}(D).$

So

$$V(D) = [v^{(0)}(D), v^{(1)}(D)] = v^{(0)}(D^2) + D v^{(1)}(D^2)$$

Assume that  $u = 10111 \Rightarrow u(D) = 1 + D^2 + D^3 + D^4$ .

Then,

$$v^{(0)}(D) = (1 + D^2 + D^3 + D^4)(1 + D^2 + D^3) = 1 + D^7$$

$$\begin{aligned} v^{(1)}(D) &= (1 + D^2 + D^3 + D^4)(1 + D + D^2 + D^3) = \\ &= 1 + D + D^3 + D^4 + D^5 + D^7 \end{aligned}$$

and

$$\begin{aligned} v(D) &= [1 + D^7, 1 + D + D^3 + D^4 + D^5 + D^7] \\ &= 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15}. \end{aligned}$$

As an exercise try to get 11010001010100 by feeding 10111 to the encoder.

Equivalently, we could use

$$g(D) = g^{(0)}(D^2) + Dg^{(1)}(D^2)$$

and find the output using

$$v(D) = u(D^2)g(D)$$

For the above example:

$$g(D) = 1 + D^4 + D^6 + D[1 + D^2 + D^4 + D^6]$$

$$= 1 + D^4 + D^6 + D + D^3 + D^5 + D^7$$

$$= 1 + D + D^3 + D^4 + D^5 + D^6 + D^7$$

and

$$\begin{aligned} v(D) &= u(D^2)g(D) = (1 + D^4 + D^6 + D^8)(1 + D + D^3 + D^4 + D^5 + D^6 + D^7) \\ &= 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15} \end{aligned}$$

In general for a code with  $k$  inputs and  $n$  outputs, we have

$$g_i(D) = g_i^{(0)}(D^n) + D g_i^{(1)}(D^n) + \dots + D^{n-1} g_i^{(n-1)}(D^n)$$

for  $1 \leq i \leq k$

and

$$V(D) = \sum_{i=1}^k u^{(i)}(D^n) g_i(D).$$

## Graphical representation of Convolutional codes

Trees, Trellises and FSM

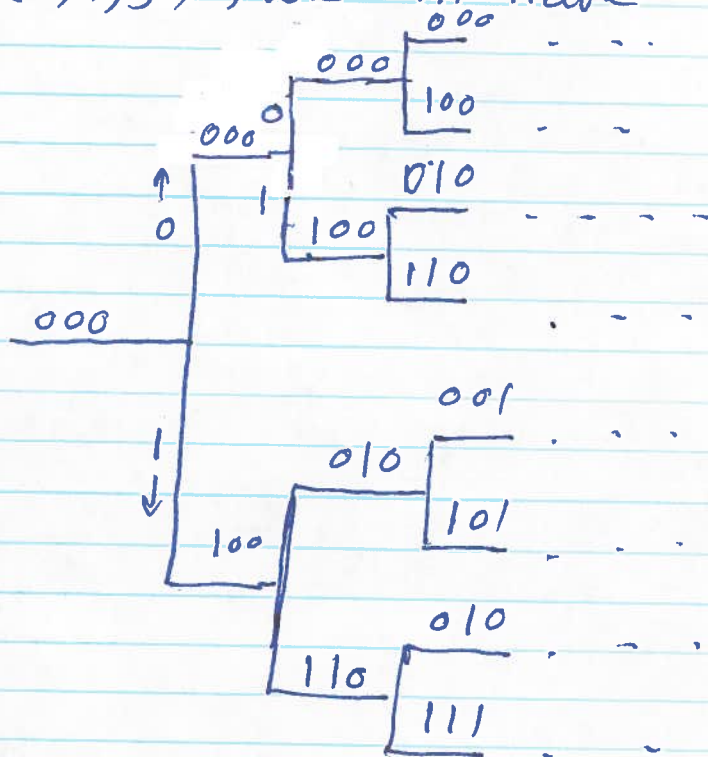
Consider a code with  $m$  memory elements, each bit takes the contents of shift register from one of the  $2^m$  possible values to another value. For simplicity take the case of feedforward,  $k=1$  code. Let's start from all zero content for  $m$  memory elements i.e., start from

$\underbrace{00 \dots 0}_m$ . If the bit entering the encoder is zero it moves to the same state.

Other wise goes to  $100 \dots 0$ .

Next bit takes it to  $010 \dots 0$  or  $110 \dots 0$ .

For example for the 3-bit memory encoder (2,1,3), we will have

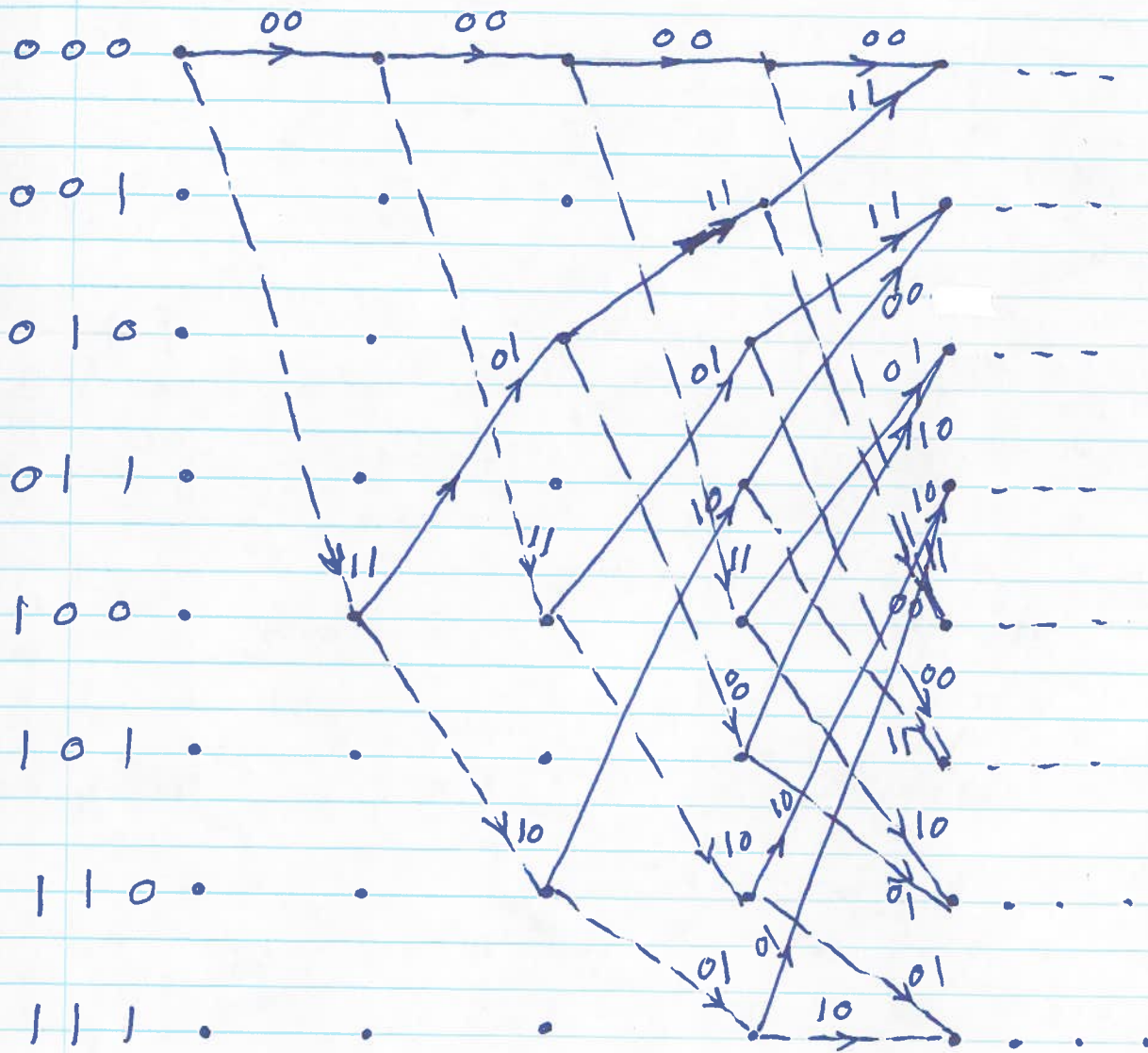


Note that after entering 3 bits, we get to 8 terminal nodes with all possible binary values. If we continue, we get trees with 16, 32, 64, ... of terminal nodes. But node labels will still be one of the eight patterns 000, 001, ..., 111.

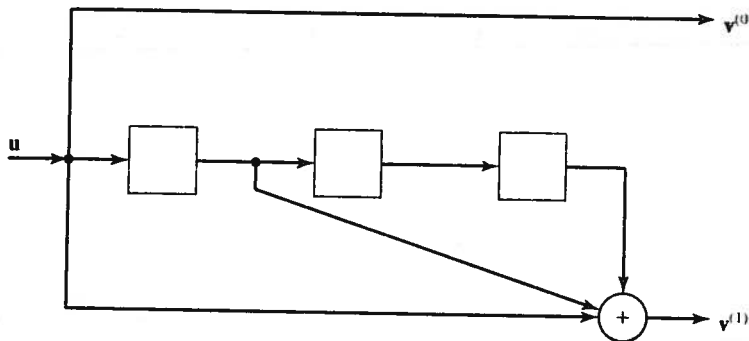
So, instead of a tree, we can use a trellis. For the above code, we have the following trellis. We label the nodes by state values and the arcs by the input and



input by thickness or shape of line, e.g., using solid line for zero and dashed line for one

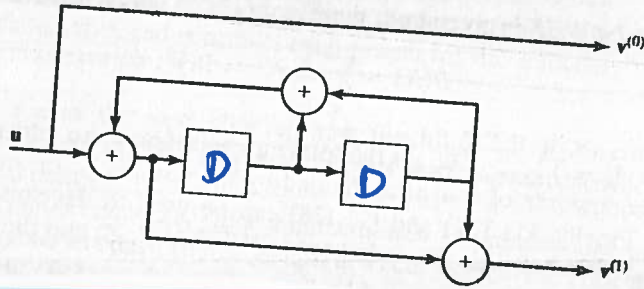


A code can be systematic, e.g.,



A (2, 1, 3) binary systematic feedforward convolutional encoder.

Convolutional encoders can also have feedback. Following is an example of  $(2, 1, 2)$  Systematic feedback encoder.



These codes are defined in terms of their feedback and feed forward transfer functions (or generator functions). The above code has  $g_0(D) = 1$  and  $g_1(D) = \frac{1 + D^2}{1 + D + D^2}$  or the generator matrix is

$$G(D) = [1, (1 + D^2)/(1 + D + D^2)]$$

Trellis Diagrams list all possible codewords.

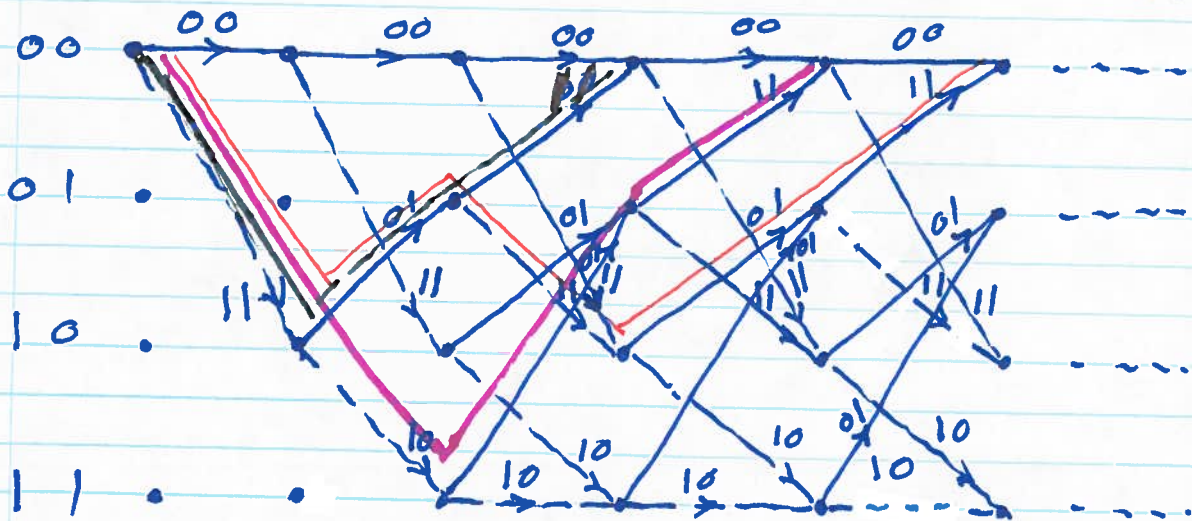
For example, for a Convolutional code with binary input, after entering  $K$  bits, i. e., passing through  $K$  stages, we have  $2^K$  paths to consider. Note that for a rate  $\frac{1}{n}$  after entering  $K$  bits



we have an  $nK$  bit sequence. So, the search is for one of the  $2^K$  trellis paths among  $2^{nK}$  possibilities.

We will have an error if the transmitted sequence belongs to one path of the trellis and we decide <sup>in favour of</sup> another path. That is, if the noise make the path diverge at one node and converge at another node. To see this, let's try a simple example.

Consider the  $(2, 1, 2)$  feedback code above.



Assume, for example, that input the encoder is all zero sequence. Then the top line is the path taken by the output of the encoder.

Now assume that at some point there is an error and instead of straight (solid line), the decoder takes the dashed line. Unlike uncoded case, the next output is not unconstrained since the first divergence has resulted in 00 being 11, then the next two bits are either 01 or 10. This continues until the two paths merge. Any convergence of two diverged paths is called an error event. The shortest error event defines, the minimum or free distance of the code. In this case the free distance is 5 and is a result of 00, 00, 00 being changed to 11, 01, 00. That means decoding 100 instead of 000.

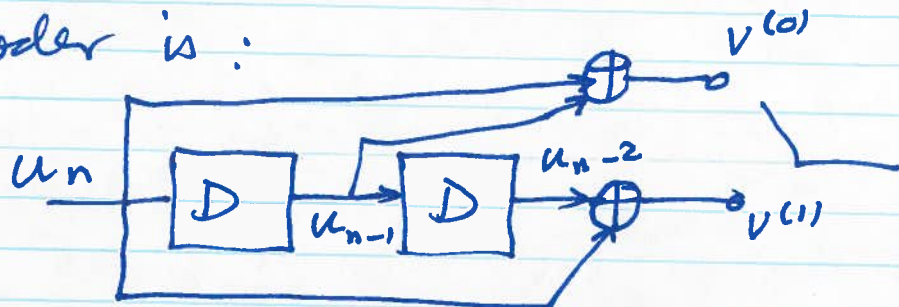
### Catastrophic Convolutional Encoders

A convolutional encoder, is one that creates a trellis diagram in which an stream with an infinite number of errors appears as

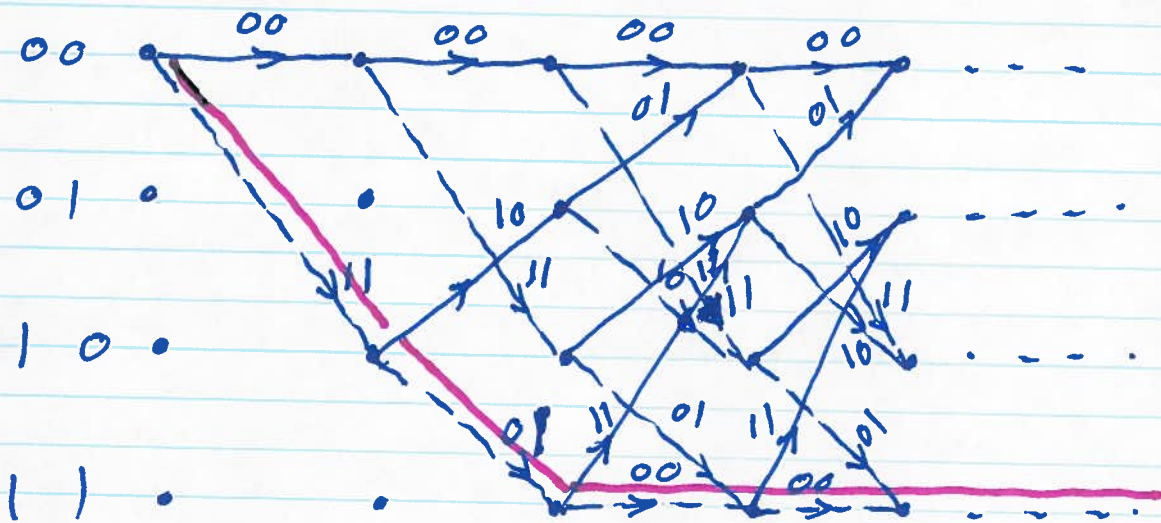
finite  
 having a distance from the original  
 sequence.

Example: Take the Convolutional Code  
 with  $G(D) = [1+D, 1+D^2]$

The encoder is:



The trellis for this encoder is:



Now assume that  $00 \dots 0$  be encoded, i.e.,  
 $00, 00, \dots 00$  (upper path) be transmitted.

The distance between  $11, 01, 00, 00, \dots 00$   
 (the violet. path) and the correct path is 3.

But one decodes to all 0 and the other to  
 all one.

Let's analyze the situation.

Note that  $1+D$  is a common divisor to

$$g_0(D) = 1+D \quad \text{and} \quad g_1(D) = 1+D^2 = (1+D)(1+D)$$

So, the encoder can be considered as an  $m=1$  encoder  $[1, 1+D]$  and a multiplication (say prior to encoding) by  $1+D$ .

Note that

$$\frac{1}{1+D} = 1 + D + D^2 + D^3 + \dots$$

or  $(1 + D + D^2 + D^3 + \dots)(1+D)$

So, if we feed the all one sequence to the  $1+D$ , we get a single 1 and all zero.

The same is true for any encoder whose constituent  $g_i(D)$  have a common divisor other than unity (or a shift of unity say  $D^l$ ).

A  $\frac{1}{n}$  code is not catastrophic if and only if

$$\text{GCD}[g_0(D), g_1(D), \dots, g_{n-1}(D)] = D^l \text{ for some integer } l.$$

## Performance of Convolutional Codes

The performance of convolutional codes is computed based on their distance profile (spectrum)

Assume that in the trellis of a code there are <sup>error event</sup>

$A_d$  paths of distance  $d$ , for all possible  $d$ .

Then

$$A(X) = \sum_{d=d_{\text{free}}}^{\infty} A_d X^d$$

Assuming that Probability of error <sup>event</sup> for a path of distance  $d$  is  $P_d$ . then:

$$P(E) \leq \sum_{d=d_{\text{free}}}^{\infty} A_d P_d$$

$P(E)$  can be approximated as (upper bound approximation for BSC):

$$P(E) < \sum_{d=d_{\text{free}}}^{\infty} A_d [2\sqrt{p(1-p)}]^d = A(X) \Big|_{X=2\sqrt{p(1-p)}}$$

For low  $p$ , i.e., high SNR, we have the path with distance  $d_{\text{free}}$  as dominant so:

$$P(E) \approx A_{d_{\text{free}}} [2\sqrt{p(1-p)}]^{d_{\text{free}}}$$

Instead of  $A_d$  which is the number of error events of weight  $d$ , we may use  $B_d$  which is the sum of numbers of non-zero bits on all  $d$  paths divided by the number of information bits  $k$ .

Then

$$P_b(E) < \sum_{d=d_{\text{free}}}^{\infty} B_d P_d = \sum_{d=d_{\text{free}}}^{\infty} B_d [2p\sqrt{1-p}]^d$$

Again, taking  $d_{\text{free}}$  path as the dominant path:

$$P(E) \approx B_{d_{\text{free}}} [2p\sqrt{1-p}]^{d_{\text{free}}}$$

$$\approx B_{d_{\text{free}}} 2^{d_{\text{free}}} p^{d_{\text{free}}}$$

Letting  $p = Q\left(\sqrt{\frac{2E_s}{N_0}}\right) \approx \frac{1}{2} e^{-E_s/N_0}$ , i.e.,

assuming BPSK:

$$P_b(E) \approx B_{d_{\text{free}}} 2^{d_{\text{free}}/2} e^{-\left(\frac{d_{\text{free}}}{2}\right)\left(\frac{E_s}{N_0}\right)}$$

Note that  $E_b = \frac{E_s}{R}$ , so

$$P_b(E) \approx B_{d_{\text{free}}} 2^{d_{\text{free}}/2} e^{-\left(\frac{R d_{\text{free}}}{2}\right)\left(\frac{E_b}{N_0}\right)}$$

Comparing this with uncoded BPSK, i.e.,

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \approx \frac{1}{2} e^{-E_b/N_0}$$

We find that there is a power gain of  $\frac{R \text{ degree}}{2}$ .

This in decible is called the asymptotic coding gain:

$$\gamma \triangleq 10 \log_{10} \left( \frac{R \text{ degree}}{2} \right) \text{ dB.}$$

### Soft Decoding:

If we do not do demodulation prior to decoding, we use Euclidean distance.

Then:

$$P(E) \leq \sum_{d=d_{\text{free}}}^{\infty} A_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right)$$

or

$$P_b(E) < \sum_{d=d_{\text{free}}}^{\infty} B_d Q\left(\sqrt{\frac{2dRE_b}{N_0}}\right)$$

We can approximate the  $Q(\cdot)$  function so that,

$$P_b(E) < \sum_{d=d_{\text{free}}}^{\infty} B_d e^{-\frac{dRE_b}{N_0}}$$

or

$$P_b(E) \approx B_{d_{\text{free}}} e^{-\frac{d_{\text{free}} R E_b}{N_0}}$$

Comparing with uncoded BPSK, i.e.,

$$P_b \approx e^{-\frac{E_b}{N_0}}$$

we get coding gain (asymptotic):

$$\gamma \triangleq 10 \log_{10} (R d_{\text{free}}) \text{ dB}$$

which is 3 dB better than hard decision decoding.



TABLE 12.2: Optimum rate  $R = 1/2$  quick-look-in convolutional codes.

| $\nu$ | $g^{(0)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|------------|----------------|---------------|
| 2     | 5         | 5          | 1              | 3.98          |
| 3     | 15        | 6          | 1              | 4.77          |
| 4     | 31        | 7          | 2              | 5.44          |
| 5     | 55        | 8          | 2              | 6.02          |
| 6     | 151       | 9          | 4              | 6.53          |
| 7     | 215       | 9          | 1              | 6.53          |
| 8     | 455       | 10         | 1              | 6.99          |
| 9     | 1335      | 11         | 3              | 7.40          |
| 10    | 3055      | 12         | 3              | 7.78          |
| 11    | 6055      | 13         | 8              | 8.13          |
| 12    | 14135     | 14         | 10             | 8.45          |
| 13    | 34731     | 14         | 3              | 8.45          |
| 14    | 60545     | 15         | 6              | 8.75          |
| 15    | 171045    | 16         | 11             | 9.03          |
| 16    | 341225    | 16         | 2              | 9.03          |
| 17    | 613151    | 17         | 5              | 9.29          |
| 18    | 1422255   | 18         | 6              | 9.54          |
| 19    | 3007451   | 18         | 2              | 9.54          |
| 20    | 6153605   | 19         | 4              | 9.78          |
| 21    | 14565371  | 20         | 7              | 10.00         |
| 22    | 32720445  | 20         | 1              | 10.00         |
| 23    | 63347465  | 21         | 3              | 10.21         |
| 24    | 147373045 | 22         | 7              | 10.41         |

TABLE 12.1(a)<sup>3</sup>: Optimum rate  $R = 1/4$  convolutional codes.

| $\nu$ | $g^{(0)}$ | $g^{(1)}$ | $g^{(2)}$ | $g^{(3)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|-----------|-----------|-----------|------------|----------------|---------------|
| 1     | 1         | 1         | 3         | 3         | 6          | 1              | 1.76          |
| 2     | 5         | 5         | 7         | 7         | 10         | 1              | 3.98          |
| 3     | 13        | 13        | 15        | 17        | 13         | 2              | 5.12          |
| 4     | 25        | 27        | 33        | 37        | 16         | 4              | 6.02          |
| 5     | 45        | 53        | 67        | 77        | 18         | 3              | 6.53          |
| 6     | 117       | 127       | 155       | 171       | 20         | 2              | 6.99          |
| 7     | 257       | 311       | 337       | 355       | 22         | 1              | 7.40          |
| 8     | 533       | 575       | 647       | 711       | 24         | 1              | 7.78          |
| 9     | 1173      | 1325      | 1467      | 1751      | 27         | 3              | 8.29          |

TABLE 12.1(b): Optimum rate  $R = 1/3$  convolutional codes.

| $\nu$ | $g^{(0)}$ | $g^{(1)}$ | $g^{(2)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|-----------|-----------|------------|----------------|---------------|
| 1     | 1         | 3         | 3         | 5          | 1              | 2.22          |
| 2     | 5         | 7         | 7         | 8          | 2              | 4.26          |
| 3     | 13        | 15        | 17        | 10         | 3              | 5.22          |
| 4     | 25        | 33        | 37        | 12         | 5              | 6.02          |
| 5     | 47        | 53        | 75        | 13         | 1              | 6.36          |
| 6     | 117       | 127       | 155       | 15         | 3              | 6.99          |
| 7     | 225       | 331       | 367       | 16         | 1              | 7.27          |
| 8     | 575       | 623       | 727       | 18         | 1              | 7.78          |
| 9     | 1167      | 1375      | 1545      | 20         | 3              | 8.23          |
| 10    | 2325      | 2731      | 3747      | 22         | 7              | 8.65          |
| 11    | 5745      | 6471      | 7553      | 24         | 13             | 9.03          |
| 12    | 2371      | 13725     | 14733     | 24         | 5              | 9.03          |

TABLE 12.1(c): Optimum rate  $R = 1/2$  convolutional codes.

| $\nu$ | $g^{(0)}$ | $g^{(1)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|-----------|------------|----------------|---------------|
| 1     | 3         | 1         | 3          | 1              | 1.76          |
| 2     | 5         | 7         | 5          | 1              | 3.98          |
| 3     | 13        | 17        | 6          | 1              | 4.77          |
| 4     | 27        | 31        | 7          | 2              | 5.44          |
| 5     | 53        | 75        | 8          | 1              | 6.02          |
| 6     | 117       | 155       | 10         | 11             | 6.99          |
| 7     | 247       | 371       | 10         | 1              | 6.99          |
| 8     | 561       | 753       | 12         | 11             | 7.78          |
| 9     | 1131      | 1537      | 12         | 1              | 7.78          |
| 10    | 2473      | 3217      | 14         | 14             | 8.45          |
| 11    | 4325      | 6747      | 15         | 14             | 8.75          |
| 12    | 10627     | 16765     | 16         | 14             | 9.03          |
| 13    | 27251     | 37363     | 16         | 1              | 9.03          |

TABLE 12.1(d): Optimum rate  $R = 2/3$  convolutional codes.

| $\nu$ | $h^{(2)}$ | $h^{(1)}$ | $h^{(0)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|-----------|-----------|------------|----------------|---------------|
| 2     | 3         | 5         | 7         | 3          | 1              | 3.01          |
| 3     | 17        | 15        | 13        | 4          | 1              | 4.26          |
| 4     | 23        | 31        | 27        | 5          | 3              | 5.23          |
| 5     | 71        | 57        | 73        | 6          | 7              | 6.02          |
| 6     | 123       | 147       | 121       | 7          | 17             | 6.69          |
| 7     | 313       | 227       | 241       | 8          | 43             | 7.27          |
| 8     | 555       | 631       | 477       | 8          | 6              | 7.27          |
| 9     | 1051      | 1423      | 1327      | 9          | 17             | 7.78          |
| 10    | 2621      | 2137      | 3013      | 10         | 69             | 8.24          |

TABLE 12.1(e): Optimum rate  $R = 3/4$  convolutional codes.

| $\nu$ | $h^{(3)}$ | $h^{(2)}$ | $h^{(1)}$ | $h^{(0)}$ | $d_{free}$ | $A_{d_{free}}$ | $\gamma$ (dB) |
|-------|-----------|-----------|-----------|-----------|------------|----------------|---------------|
| 2     | 2         | 5         | 7         | 6         | 3          | 6              | 3.52          |
| 3     | 11        | 13        | 15        | 12        | 4          | 10             | 4.77          |
| 4     | 33        | 25        | 37        | 31        | 4          | 2              | 4.77          |
| 5     | 47        | 73        | 57        | 75        | 5          | 7              | 5.74          |
| 6     | 107       | 135       | 133       | 141       | 6          | 27             | 6.53          |
| 7     | 211       | 341       | 315       | 267       | 6          | 5              | 6.53          |
| 8     | 535       | 757       | 733       | 661       | 7          | 27             | 7.20          |
| 9     | 1475      | 1723      | 1157      | 1371      | 8          | 136            | 7.78          |