Convolutional Codes

In Block Codes data is 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. Sometimes $m+1$ is called the constraint length taking into account the present input 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 $\frac{1}{n}$.

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

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

when $m \ll L$.

Since the codewords of a convolutional code are generate 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.

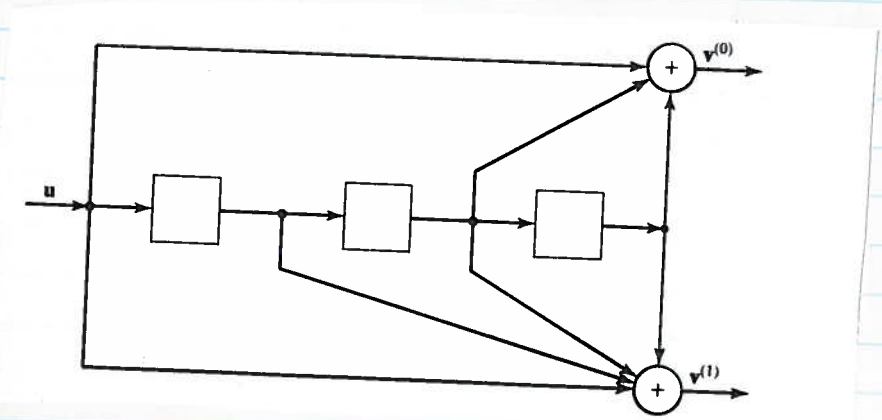
Another scheme called BCJR (Bahl, Cocke, Jelinek and Raviv) algorithm is another decoding technique that works based on the maximum a posterior (MAP) probability taking into account the a priori probability of bits. It is very useful in iterative decoding of Turbo Codes.

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 $\underline{u} = (u_0, u_1, u_2, \ldots)$

The output sequences will be

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

and

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

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

To find the impulse response let

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

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

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

and

$$\underline{g}^{(1)} = (g_1^{(0)}, g_1^{(1)}, \cdots, g_m^{(1)})$$

For this example

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

and

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

These are also called <u>generator sequences</u>.

Now,

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

and

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

where $\circledast$ denotes convolution:

$$v_\ell^{(j)} = \sum_{i=0}^{m} u_{\ell-i}\, g_i^{(j)} = u_\ell g_0^{(j)} + u_{\ell-1} g_1^{(j)} + \cdots + u_{\ell-m} g_m^{(j)}$$

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

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

Assume $\underline{u} = (1\,0\,1\,1\,1)$

Then

$$\underline{v}^{(0)} = (1\,0\,1\,1\,1) \otimes (1\,0\,1\,1) = (1\,0\,0\,0\;0\,0\,0\,1)$$

$$\underline{v}^{(1)} = (1\,0\,1\,1\,1) \otimes (1\,1\,1\,1) = (1\,1\,0\,1\,1\;1\,0\,1)$$

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:

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

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

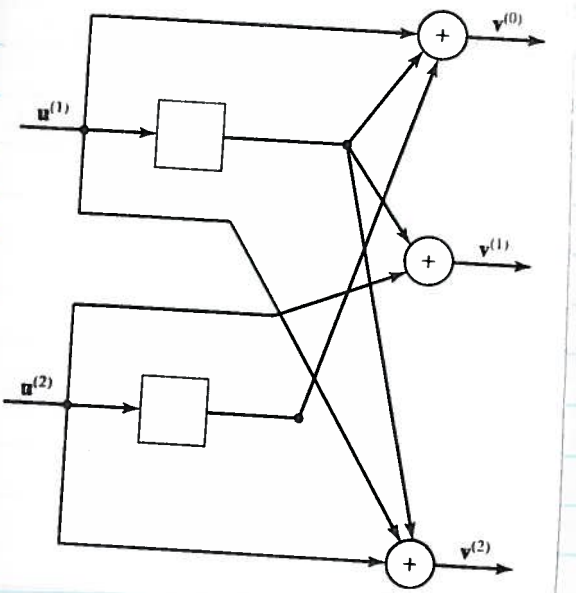
For input $\underline{u} = (1\,0\,1\,1\,1)$, we have

$$\mathbf{v} = \mathbf{uG}$$

$$= (1\,0\,1\,1\,1) \begin{bmatrix} 11 & 01 & 11 & 11 & & & \\ & 11 & 01 & 11 & 11 & & \\ & & 11 & 01 & 11 & 11 & \\ & & & 11 & 01 & 11 & 11 \\ & & & & 11 & 01 & 11 & 11 \end{bmatrix}$$

$$= (1\,1, 0\,1, 0\,0, 0\,1, 0\,1, 0\,1, 0\,0, 1\,1),$$

Example: A Rate 2/3 Nonsystematic Feed forward Convolutional Code:



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

$$\mathbf{g}_1^{(0)} = (1\ 1) \qquad \mathbf{g}_1^{(1)} = (0\ 1) \qquad \mathbf{g}_1^{(2)} = (1\ 1),$$
$$\mathbf{g}_2^{(0)} = (0\ 1) \qquad \mathbf{g}_2^{(1)} = (1\ 0) \qquad \mathbf{g}_2^{(2)} = (1\ 0),$$

Then:

$$\mathbf{v}^{(0)} = \mathbf{u}^{(1)} \circledast \mathbf{g}_1^{(0)} + \mathbf{u}^{(2)} \circledast \mathbf{g}_2^{(0)}$$
$$\mathbf{v}^{(1)} = \mathbf{u}^{(1)} \circledast \mathbf{g}_1^{(1)} + \mathbf{u}^{(2)} \circledast \mathbf{g}_2^{(1)}$$
$$\mathbf{v}^{(2)} = \mathbf{u}^{(1)} \circledast \mathbf{g}_1^{(2)} + \mathbf{u}^{(2)} \circledast \mathbf{g}_2^{(2)}.$$

So,

$$v_l^{(0)} = u_l^{(1)} + \quad + u_{l-1}^{(1)} + u_{l-1}^{(2)},$$
$$v_l^{(1)} = \quad u_l^{(2)} + u_{l-1}^{(1)},$$
$$v_l^{(2)} = u_l^{(1)} + u_l^{(2)} + u_{l-1}^{(1)},$$

rate

While $\frac{k}{n}$ 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 8. 64-State Convolutional Encoder

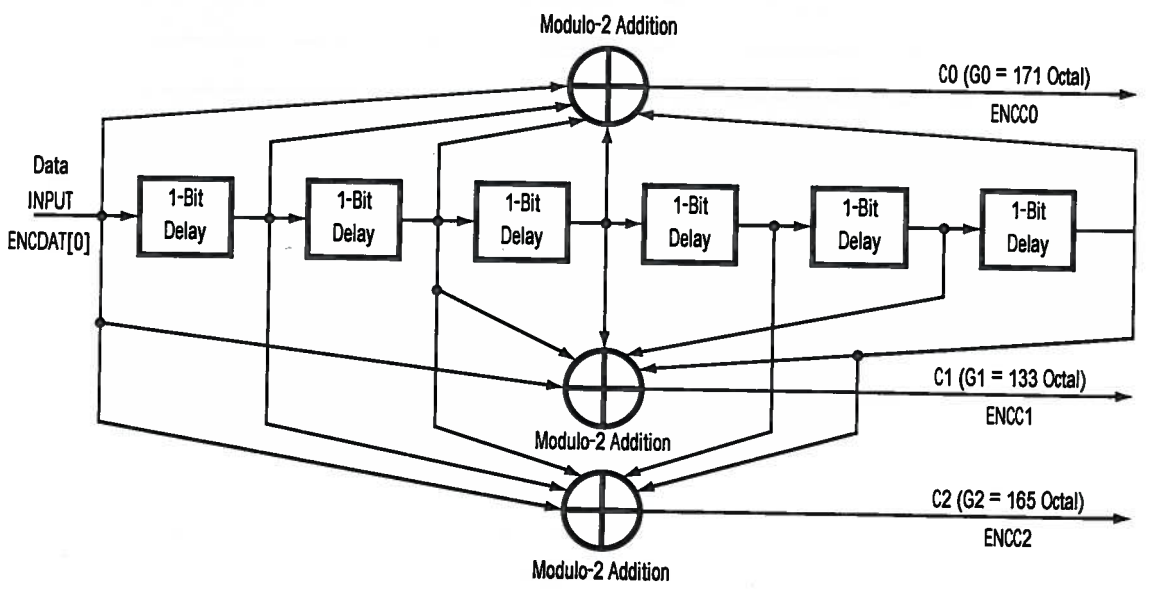


Figure 9a. Punctured Coding for Rate 3/4

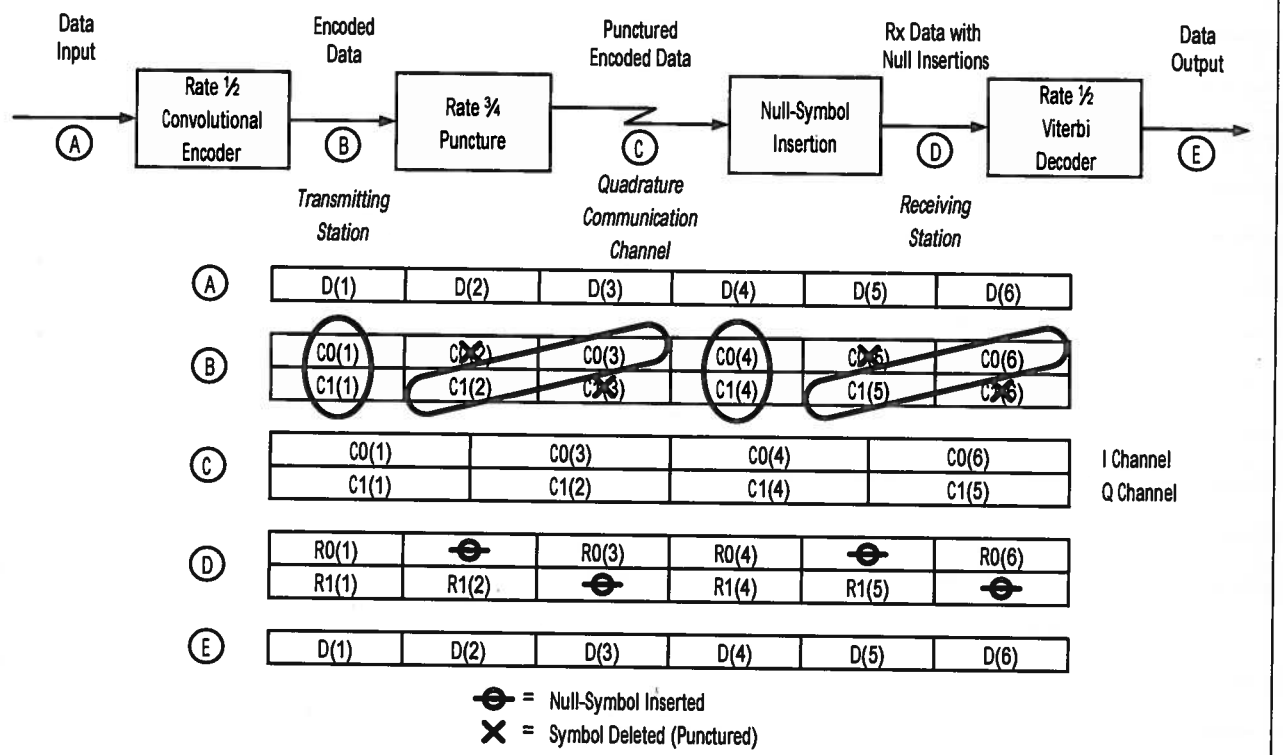
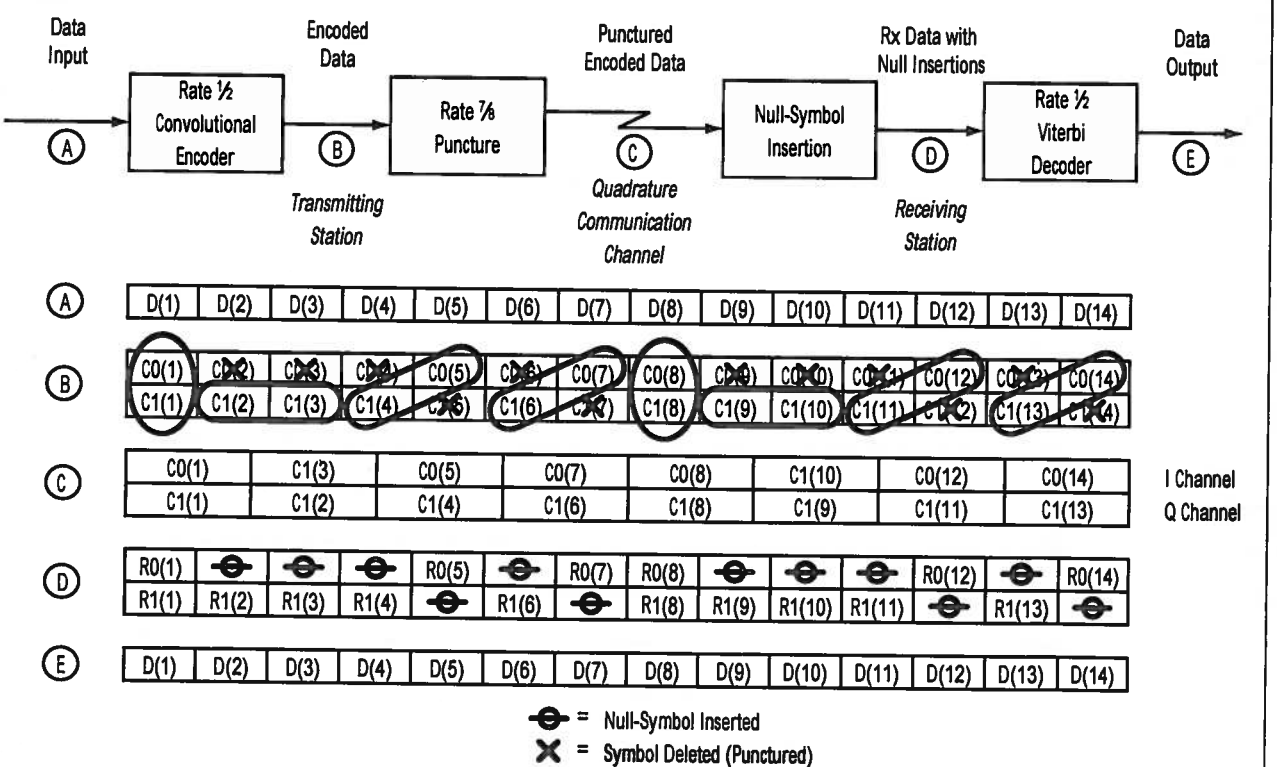


Figure 9b. Punctured Coding for Rate 7/8

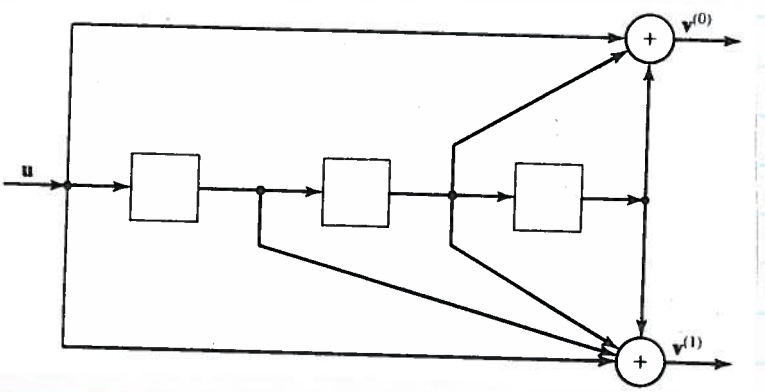


Polynomial Domain representation:

Similar to block codes, we can transform the time domain representation by transforming the generating sequences to generating polynomial 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:

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



If we denote the input sequence as,

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

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 $\underline{u} = 10111 \Rightarrow u(D) = 1 + D^2 + D^3 + D^4$.

Then,

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

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

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

and

$$v(D) = [1 + D^7,\ 1 + D + D^3 + D^4 + D^5 + D^7]$$

$$= 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15}.$$

As an exercise try to get 1101000101010 0 by feeding 10111 to the encoder.

Equivalently, we could use

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

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

$$v(D) = u(D^2)\, g(D) = (1 + D^4 + D^6 + D^8)(1 + D + D^3 + D^4 + D^5 + D^6 + D^7)$$

$$= 1 + D + D^3 + D^7 + D^9 + D^{11} + D^{14} + D^{15}$$

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

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

for $1 \le i \le k$

and

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

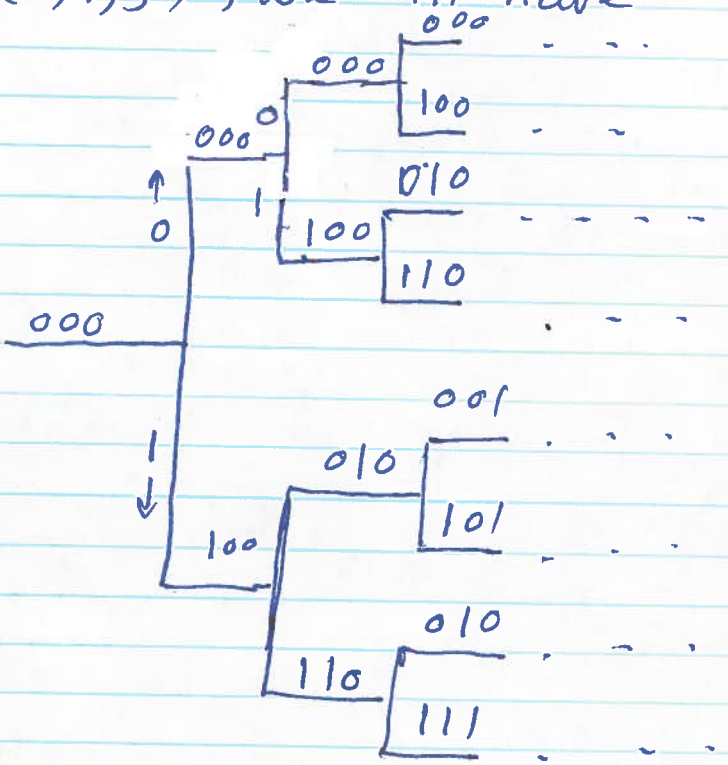
Graphical representation of Convolutional Codes

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 $\underbrace{0\,0\,\ldots\,0}_{m}$. If the bit entering the encoder is zero it moves to the same state. Otherwise goes to $1\,0\,0\cdots0$.

Next bit takes it to $0\,1\,0\cdots0$ or $1\,1\,0\cdots0$.

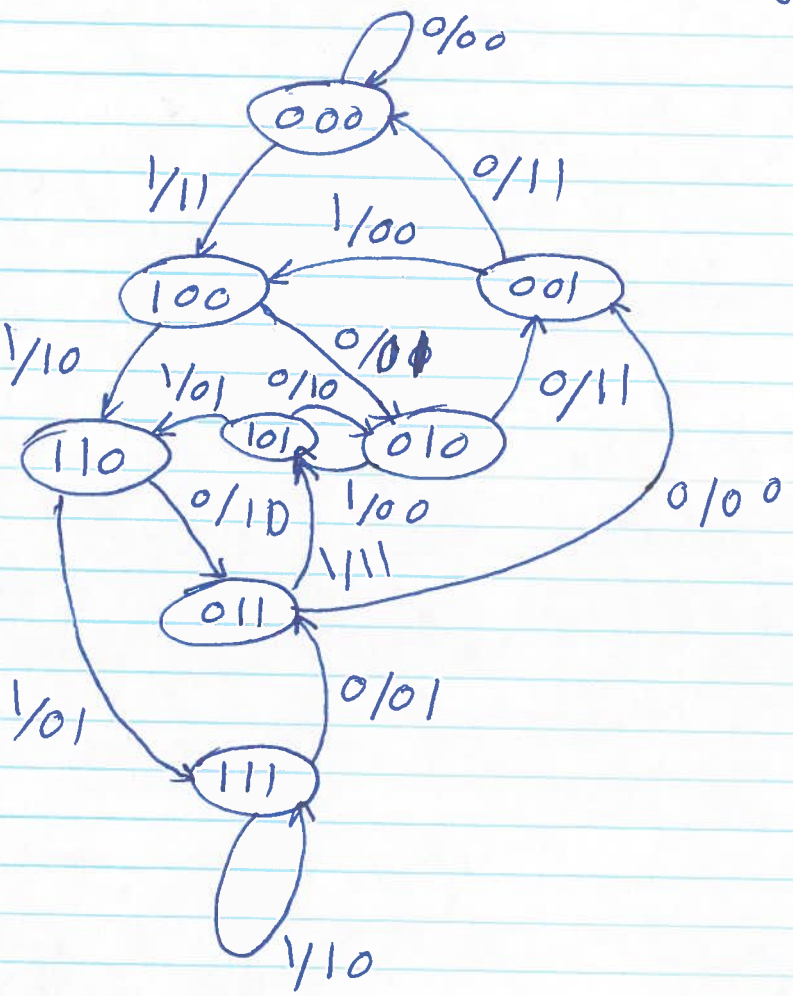
For example for the 3-bit memory encoder (2,1,3), we will have



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.

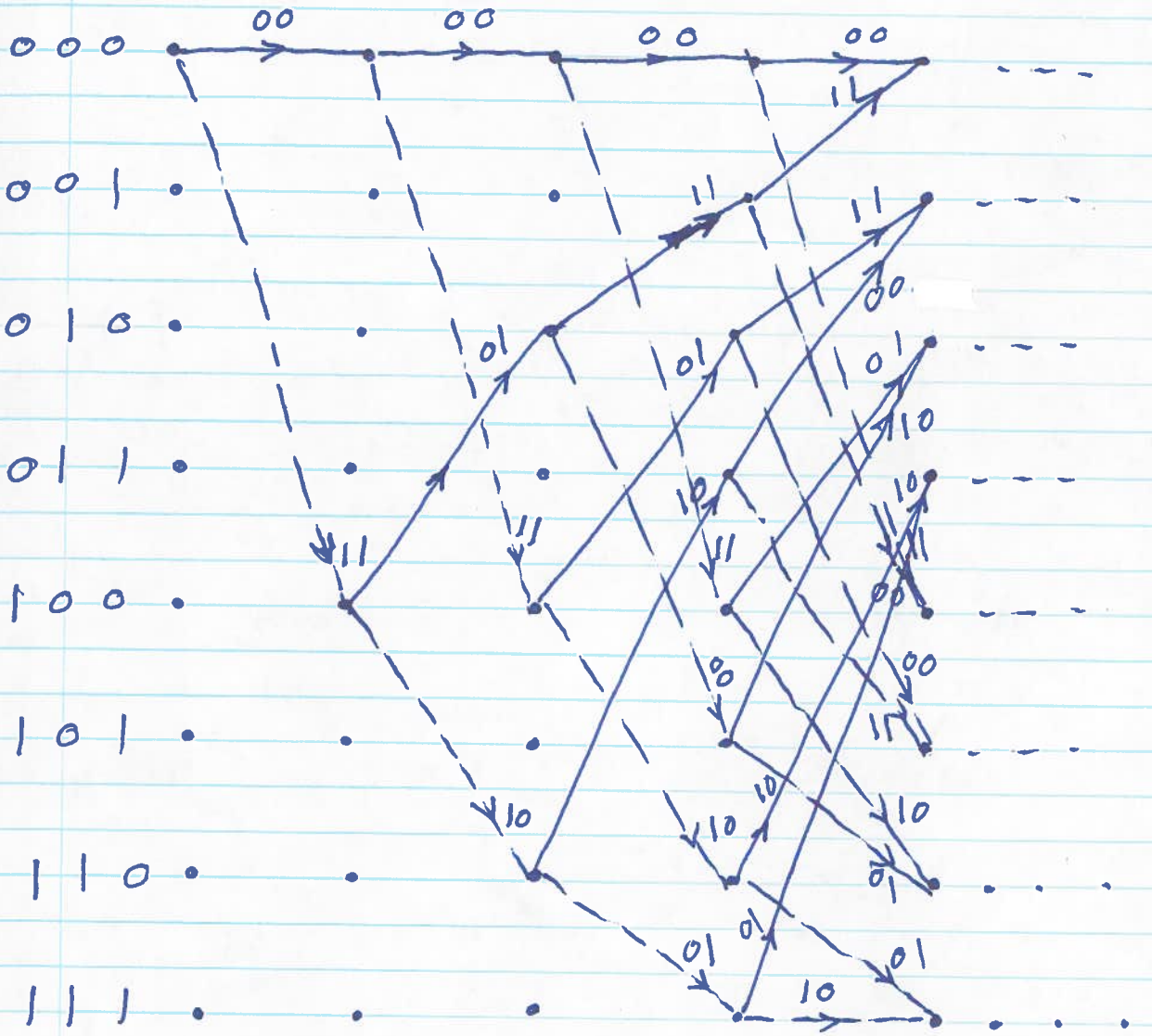
So, instead of a tree, we can use a trellis. For the above code, we have the following trellis. We label the nodes by state values and the arcs by the input and

Let's first draw the state-diagram:

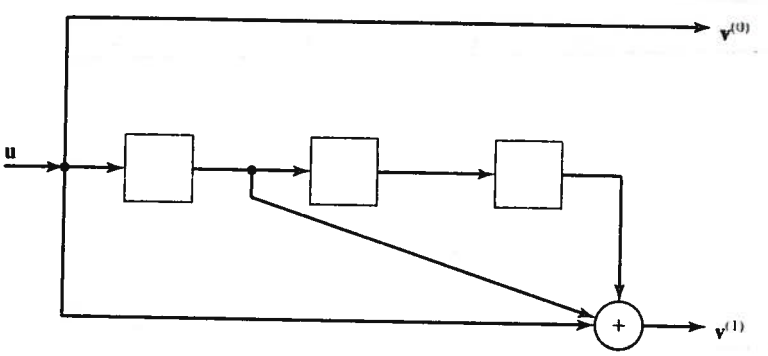


Each state is labeled with the content of the memory elements and each transition is labeled with the input and output. So, An input 1 takes the system from state 000 to state 100 generating output 11, So, 1/11 is the label of the transition from 000 to 1000. Since this is a binary (one bit) input code, we can represent the

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

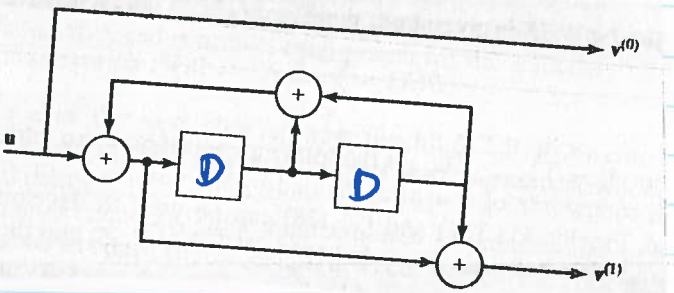


A code can be systematic, e.g.,



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 feedforward transfer functions (or generator functions). The above code has $g_0(D) = 1$

and $g_1(D) = \dfrac{1+D^2}{1+D+D^2}$

or the generator matrix is

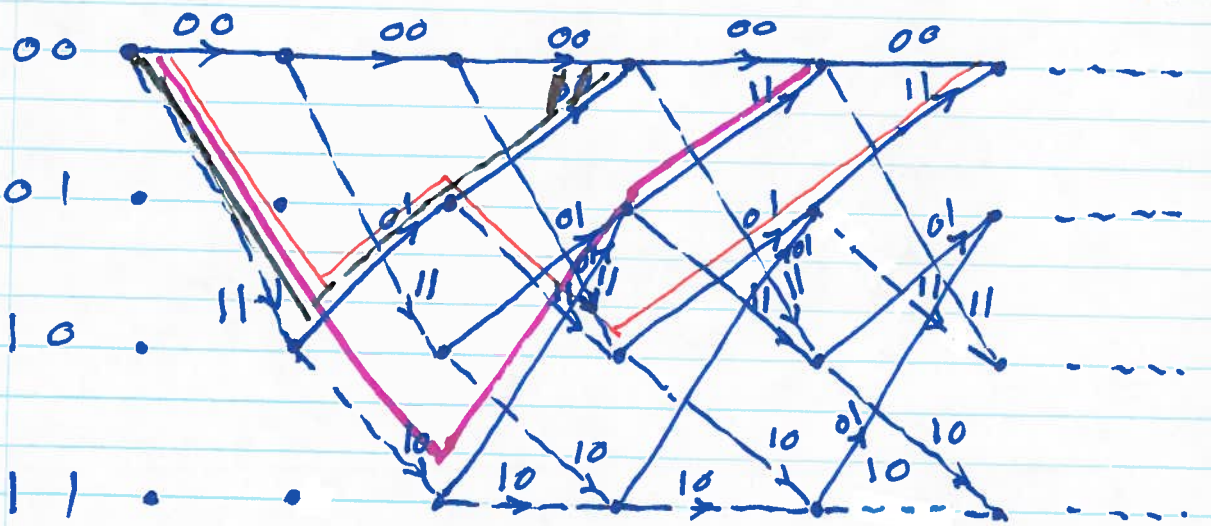
$$G(D) = \left[1 \;,\; (1+D^2)/(1+D+D^2)\right]$$

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 pathes among 2^{nK} possibilities.

We will have an error if the transmitted sequence belongs to one path of the trellis and we decide in favour of 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 un-constrai 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.

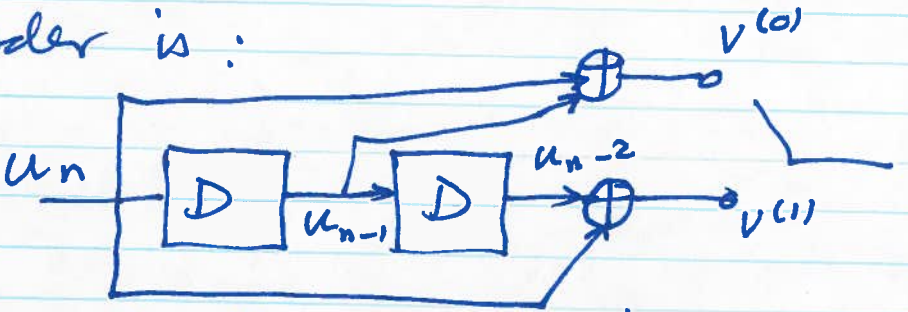
Catastrophic Convolutional Encoders

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

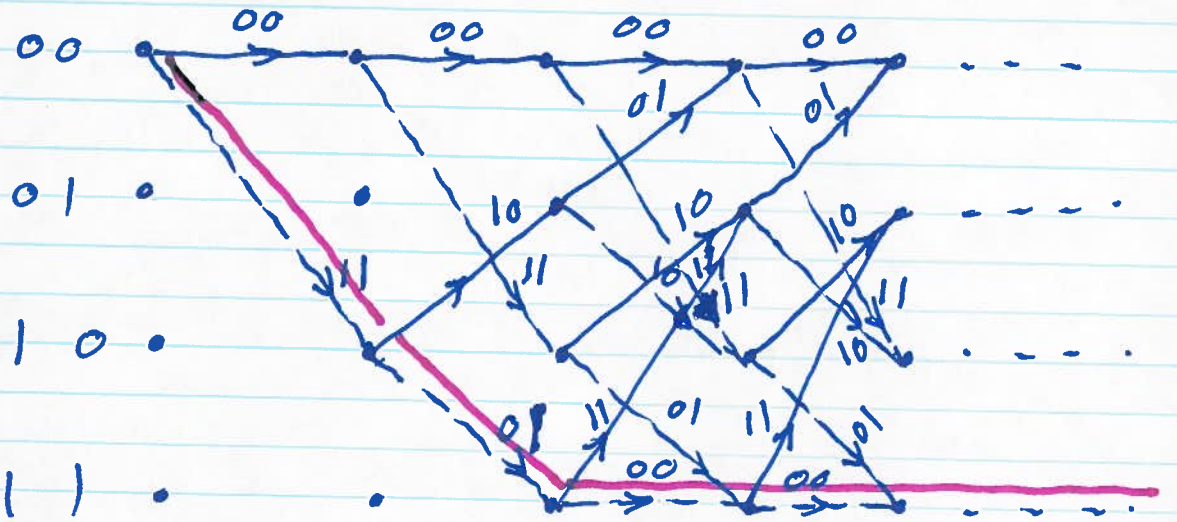
Finite
having d_f distance from the original sequence.

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 0 0 ... 0 be encoded, i.e., 00, 00, ... 00 (upper path) be transmitted. The distance between 11, 01, 00, 00, ... 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

$g_0(D) = 1+D$ and $g_1(D) = 1+D^2 = (1+D)(1+D)$

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

Note that

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

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

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

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

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 A_d error event paths of distance d, for all possible d.

Then

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

Assuming that Probability of error event for a path of distance d is P_d. then:

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

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

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

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

$$P(E) \simeq A_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}}.$$

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

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

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

$$P(E) \simeq B_{d_{free}} \left[2\sqrt{p(1-p)}\right]^{d_{free}}$$

$$\simeq B_{d_{free}} 2^{d_{free}} p^{d_{free}}$$

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

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

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

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

Comparing this with uncoded BPSK, i.e.,

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

We find that there is a power gain of

$$\frac{R\, d_{free}}{2}.$$

This in decible is called the asymptotic coding gain:

$$\gamma \triangleq 10 \log_{10}\left(\frac{R\, d_{free}}{2}\right) \quad dB.$$

Soft Decoding:

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

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d\, Q\left(\sqrt{\frac{2d\,R E_b}{N_o}}\right)$$

or

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

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

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

or

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

Comparing with uncoded BPSK, i.e.,

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

we get coding gain (asymptotic):

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

which is 3 dB better than hard decision decoding.

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

ν	$\mathbf{g}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (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)[3]: Optimum rate $R = 1/4$ convolutional codes.

ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbf{g}^{(2)}$	$\mathbf{g}^{(3)}$	d_{free}	$A_{d_{free}}$	γ (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.

ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	$\mathbf{g}^{(2)}$	d_{free}	$A_{d_{free}}$	γ (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.

ν	$\mathbf{g}^{(0)}$	$\mathbf{g}^{(1)}$	d_{free}	$A_{d_{free}}$	γ (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.

ν	$\mathbf{h}^{(2)}$	$\mathbf{h}^{(1)}$	$\mathbf{h}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (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.

ν	$\mathbf{h}^{(3)}$	$\mathbf{h}^{(2)}$	$\mathbf{h}^{(1)}$	$\mathbf{h}^{(0)}$	d_{free}	$A_{d_{free}}$	γ (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