Convolutional Codes

- Like the block codes discussed earlier, send parity bits computed from blocks of message bits
  - Unlike block codes, don’t send message bits, send only the parity bits!
  - The code rate of a convolutional code tells you how many parity bits are sent for each message bit. We’ll mostly be talking about rate $1/r$ codes.
  - Use a sliding window to select which message bits are participating in the parity calculations. The width of the window (in bits) is called the code’s constraint length.

0101100101100011…
  $p_0[n] = x[n] + x[n-1] + x[n-2]$
  \[ p_0[n] = x[n] + x[n-1] + x[n-2] \]

Shift-register View

- One often sees convolutional encoders described with a block diagram like the following:
  - Think of this as “black box”: message in, parity out
    - Input bits arrive one-at-a-time from the left
    - The box computes the parity bits using the incoming bit and the $K-1$ previous message bits
    - At the end of the bit time, the saved message bits are shifted right by one, and the incoming bit moves into the left position.
Example: Transmit message 1011

\[\begin{align*}
\text{Processing } x[0] & : 1 \\
\text{Processing } x[1] & : 0 \\
\text{Processing } x[2] & : 1 \\
\text{Processing } x[3] & : 0
\end{align*}\]

\[\begin{align*}
p_i[n] &= x[n] + x[n-1] + x[n-2] \\
p_i[n] &= x[n] + x[n-2]
\end{align*}\]

Parity Bit Equations

- A convolutional code generates sequences of parity bits from sequences of message bits:
  \[p_i[n] = \sum_{j=0}^{K-1} g_i[j] x[n-j] \mod 2\]

- \(K\) is the constraint length of the code
  - The larger \(K\) is, the more times a particular message bit is used when calculating parity bits
  - better error correction possibilities (in general)

- \(g_i\) is the \(K\)-element generator polynomial for parity bit \(p_i\)
  - Each element \(g_i[n]\) is either 0 or 1
  - More than one parity sequence can be generated from the same message; the simplest choice is to use 2 generator polynomials

Convolutional Codes (cont’d.)

- We’ll transmit the parity sequences, not the message itself
  - As we’ll see, we can recover the message sequences from the parity sequences
  - Each message bit is “spread across” \(K\) elements of each parity sequence, so the parity sequences are better protection against bit errors than the message sequence itself

- If we’re using multiple generators, construct the transmit sequence by interleaving the bits of the parity sequences:
  \[\text{xmit} = p_0[0], p_0[1], p_0[2], p_1[1], p_2[1], p_1[2] \ldots\]

- Code rate is \(1/\text{number of generators}\)
  - 2 generator polynomials → rate = \(1/2\)
  - Engineering tradeoff: using more generator polynomials improves bit-error correction but decreases rate of the code (the number of message bits/s that can be transmitted)

Example

- Using two generator polynomials:
  - \(g_0 = 1, 1, 0, 0, \ldots\) abbreviated as 110 for \(K\)-3 code
  - \(g_1 = 1, 1, 0, 0, \ldots\) abbreviated as 111 for \(K\)-3 code

- Writing out the equations for the parity sequences:
  - \(p_0[n] = (x[n] + x[n-1] + x[n-2])\mod 2\)
  - \(p_1[n] = (x[n] + x[n-2])\mod 2\)

- Let \(x[n] = [1,0,1,1,\ldots]\); \(x[n]=0\) when \(n<0\):
  - \(p_0[0] = (1 + 0 + 0) \mod 2 = 1\)
  - \(p_0[1] = (1 + 0 + 0) \mod 2 = 1\)
  - \(p_0[2] = (1 + 0 + 1) \mod 2 = 0\)
  - \(p_0[3] = (1 + 0 + 1) \mod 2 = 0\)

- Transmit: 1, 1, 0, 0, 0, 1, ...
State-Machine View

- Example: $K=3$, rate-$\frac{1}{2}$ convolutional code
- States labeled with $x[n-1] x[n-2]$
- Arcs labeled with $x[n]/p_i$

**msg** = 101100; **xmit** = 11 11 01 00 01 10

The state machine is the same for all $K$-3 codes. Only the $p_i$ labels change depending on number and values for the generator polynomials.

Using Convolutional Codes

- **Transmitter**
  - Beginning at starting state, processes message bit-by-bit
  - For each message bit: makes a state transition, sends $p_i$
  - Optionally: pad message with $K$-1 zeros to ensure return to starting state

- **Receiver**
  - Doesn’t have direct knowledge of transmitter’s state transitions; only knows (possibly corrupted) received parity bits, $p_i$
  - Must find most likely sequence of transmitter states that could have generated the received parity bits, $p_i$
  - If BER < $\frac{1}{2}$, more errors < fewer errors
  - When BER < $\frac{1}{2}$, maximum-likelihood message sequence is the one that generated the codeword (here, sequence of parity bits) with the smallest Hamming distance from the received codeword (here, parity bits)
  - I.e., find nearest valid codeword closest to the received codeword

Example

- For the code $p_0 = x[n]+x[n-1]+x[n-2]$
  $p_1 = x[n] + x[n-1]$

- Received:
  11101100110

- Some errors have occurred...

- What’s the 4-bit message?

- Look for message whose codeword is closest to rcvd bits

Most likely: 1011

<table>
<thead>
<tr>
<th>Msg</th>
<th>Codeword</th>
<th>Received</th>
<th>Hamming distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>00000000000000</td>
<td>11101100110</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>00000011111111</td>
<td>11101100110</td>
<td>2</td>
</tr>
<tr>
<td>0010</td>
<td>00000111111100</td>
<td>11101100110</td>
<td>5</td>
</tr>
<tr>
<td>0011</td>
<td>00001101010110</td>
<td>11101100110</td>
<td>8</td>
</tr>
<tr>
<td>0100</td>
<td>00111110000000</td>
<td>11101100110</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>00111101001000</td>
<td>11101100110</td>
<td>8</td>
</tr>
<tr>
<td>0110</td>
<td>00110100100000</td>
<td>11101100110</td>
<td>3</td>
</tr>
<tr>
<td>0111</td>
<td>00111110001010</td>
<td>11101100110</td>
<td>8</td>
</tr>
<tr>
<td>1000</td>
<td>11111111001100</td>
<td>11101100110</td>
<td>3</td>
</tr>
<tr>
<td>1001</td>
<td>11111111010110</td>
<td>11101100110</td>
<td>8</td>
</tr>
<tr>
<td>1010</td>
<td>11111101110110</td>
<td>11101100110</td>
<td>3</td>
</tr>
<tr>
<td>1011</td>
<td>11111110110100</td>
<td>11101100110</td>
<td>5</td>
</tr>
<tr>
<td>1100</td>
<td>11110001100000</td>
<td>11101100110</td>
<td>5</td>
</tr>
<tr>
<td>1101</td>
<td>11110010110110</td>
<td>11101100110</td>
<td>5</td>
</tr>
<tr>
<td>1110</td>
<td>11110101111010</td>
<td>11101100110</td>
<td>5</td>
</tr>
<tr>
<td>1111</td>
<td>11110110111101</td>
<td>11101100110</td>
<td>5</td>
</tr>
</tbody>
</table>

Decoding: Finding the Maximum-Likelihood (ML) Path

Given the received parity bits, the receiver must find the most-likely sequence of transmitter states, i.e., the path through the trellis that minimizes the Hamming distance between the received parity bits and the parity bits the transmitter would have sent had it followed that state sequence.

*One solution: Viterbi decoding*
Key Concept: A Trellis

- Example: $K=3$, rate-$\frac{1}{2}$ convolutional code
  - $G_0 = 111$: $p_0 = 1^{*}x[n] \odot 1^{*}x[n-1] \odot 1^{*}x[n-2]$
  - $G_1 = 110$: $p_1 = 1^{*}x[n] \odot 1^{*}x[n-1] \odot 0^{*}x[n-2]$
- States labeled with $x[n-1]$ $x[n-2]$
- Arcs labeled with $x[n] / p_0 p_1$

Trellis View at Transmitter

- States labeled with $x[n-1]$ $x[n-2]$
- Arcs labeled with $x[n] / p_0 p_1$