MAP decoding: The BCJR algorithm

- Maximum a posteriori probability (MAP) decoding
  - Baum-Welch algorithm (1963?)*

Decoder inputs:
- Received sequence $r$ (soft or hard)
- A priori $L$-values $L_a(u_l) = \ln(P(u_l = 1)/P(u_l = -1))$

Decoder outputs:
- A posteriori probability (APP) $L$-values $L(u_l) = \ln(P(u_l = 1| r)/P(u_l = -1| r))$
  - $> 0$: $u_l$ is most likely to be 1
  - $< 0$: $u_l$ is most likely to be -1
BCJR (cont.)

\[
L(u_l) \equiv \ln \left[ \frac{P(u_l = +1 | r)}{P(u_l = -1 | r)} \right]
\]

\[
P(u_l = +1 | r) = \frac{p(u_l = +1, r)}{P(r)} = \frac{\sum_{u \in U^+_l} p(r|v) P(u)}{\sum_u p(r|v) P(u)}
\]

\[
L(u_l) = \ln \left[ \frac{\sum_{u \in U^+_l} p(r|v) P(u)}{\sum_{u \in U^-_l} p(r|v) P(u)} \right]
\]

\[
P(u_l = +1 | r) = \frac{p(u_l = +1, r)}{P(r)} = \frac{\sum_{(s',s) \in \Sigma^+_l} p(s_l = s', s_{l+1} = s, r)}{P(r)}
\]
$L(u_l) = \ln \left\{ \frac{\sum_{(s',s) \in \Sigma_l^+} p(s_l = s', s_{l+1} = s, r)}{\sum_{(s',s) \in \Sigma_l^-} p(s_l = s', s_{l+1} = s, r)} \right\}$

$p(s', s, r) = p(s', s, r_{t<l}, r_l, r_{t>l})$,

$p(s', s, r) = p(r_{t>l} | s', s, r_{t<l}, r_l) p(s', s, r_{t<l}, r_l) = p(r_{t>l} | s', s, r_{t<l}, r_l) p(s, r_l | s', r_{t<l}) p(s', r_{t<l}) = p(r_{t>l} | s) p(s, r_l | s') p(s', r_{t<l})$. 

BCJR (cont.)
BCJR (cont.)

\[\alpha_l(s') \equiv p(s', r_{t<l})\]

\[\gamma_l(s', s) \equiv p(s, r_l | s')\]

\[\beta_{l+1}(s) \equiv p(r_{t>l} | s),\]

\[p(s', s, r) = \beta_{l+1}(s) \gamma_l(s', s) \alpha_l(s').\]

\[\alpha_{l+1}(s) = p(s, r_{t<l+1}) = \sum_{s' \in \sigma_l} p(s', s, r_{t<l+1})\]

\[= \sum_{s' \in \sigma_l} p(s, r_l | s', r_{t<l}) p(s', r_{t<l})\]

\[= \sum_{s' \in \sigma_l} p(s, r_l | s') p(s', r_{t<l})\]

\[= \sum_{s' \in \sigma_l} \gamma_l(s', s) \alpha_l(s'),\]

\[\alpha_0(s) = \begin{cases} 1, & s = \emptyset \\ 0, & s \neq \emptyset \end{cases}\]
**BCJR (cont.)**

\[
\beta_l(s') = \sum_{s \in \sigma_{l+1}} \gamma_l(s', s) \beta_{l+1}(s)
\]

\[
\beta_K(s) = \begin{cases} 
1, & s = 0 \\
0, & s \neq 0
\end{cases}
\]

\[
\gamma_l(s', s) = p(s, r_l | s') = \frac{p(s', s, r_l)}{P(s')}
\]

\[
= \left[ \frac{P(s', s)}{P(s')} \right] \left[ \frac{p(s', s, r_l)}{P(s', s)} \right]
\]

\[
= P(s | s') p(r_l | s', s) = P(u_l) p(r_l | v_l)
\]

\[
\gamma_l(s', s) = P(u_l) p(r_l | v_l) = P(u_l) \left( \sqrt{\frac{E_s}{\pi N_0}} \right)^n e^{-\frac{E_s}{N_0} ||r_l - v_l||^2}
\]

**AWGN**
MAP algorithm

- Initialize forward and backward recursions $\alpha_0(s)$ and $\beta_N(s)$
- Compute branch metrics $\{\gamma_l(s', s)\}$
- Carry out forward recursion $\{\alpha_{l+1}(s)\}$ based on $\{\alpha_l(s)\}$
- Carry out backward recursion $\{\beta_{l-1}(s)\}$ based on $\{\beta_l(s)\}$
- Compute APP $L$-values
- Complexity: Approximately $3 \times \text{Viterbi}$
- Requires detailed knowledge of SNR
  - Viterbi just maximizes $r \cdot v$, and does not require exact knowledge of SNR
BCJR (cont.)

\[ \gamma_l(s', s) = P(u_l) e^{-E_s/N_0 \|r_l - v_l\|^2} \]

\[ P(u_l = \pm 1) = \frac{[P(u_l = +1)/P(u_l = -1)]^\pm 1}{1 + [P(u_l = +1)/P(u_l = -1)]^\pm 1} \]
\[ = \frac{e^{\pm L_a(u_l)}}{1 + e^{\pm L_a(u_l)}} \]
\[ = \frac{e^{-L_a(u_l)/2}}{1 + e^{-L_a(u_l)}} e^{u_l L_a(u_l)/2} \]
\[ = A_l e^{u_l L_a(u_l)/2}, \]

\[ \gamma_l(s', s) = A_l e^{u_l L_a(u_l)/2} e^{-(E_s/N_0)\|r_l - v_l\|^2} \]
\[ = A_l e^{u_l L_a(u_l)/2} e^{(2E_s/N_0)(r_l \cdot v_l) - \|r_l\|^2 - \|v_l\|^2} \]
\[ = A_l e^{-(\|r_l\|^2 + n)} e^{u_l L_a(u_l)/2} e^{(L_c/2)(r_l \cdot v_l)} \]
\[ = A_l B_l e^{u_l L_a(u_l)/2} e^{(L_c/2)(r_l \cdot v_l)}, \ l = 0, 1, \ldots, h - 1, \]

\[ \gamma_l(s', s) = P(u_l) e^{-(E_s/N_0)\|r_l - v_l\|^2} \]
\[ = e^{-(E_s/N_0)\|r_l - v_l\|^2} \]
\[ = B_l e^{(L_c/2)(r_l \cdot v_l)}, \ l = h, h + 1, \ldots, K - 1, \]
\[ \max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|}) \]

\[ \gamma^*_l(s', s) \equiv \ln \gamma_l(s', s) = \begin{cases} \frac{u_l L_0(u_l)}{2} + \frac{L_c}{2} r_l \cdot v_l, & l = 0, 1, \ldots, h - 1, \\ \frac{L_c}{2} r_l \cdot v_l, & l = h, h + 1, \ldots, K - 1. \end{cases} \]

\[ \alpha^*_{l+1}(s) \equiv \ln \alpha_{l+1}(s) = \ln \sum_{s' \in \sigma_l} \gamma^*_l(s', s) \alpha^*_l(s') \]
\[ = \ln \sum_{s' \in \sigma_l} e^{[\gamma^*_l(s', s) + \alpha^*_l(s')]}, \quad l = 0, 1, \ldots, K - 1 \]

\[ \alpha^*_0(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases} \]

\[ \beta^*_l(s') \equiv \ln \beta_l(s') = \ln \sum_{s \in \sigma_{l+1}} \gamma_l(s', s) \beta_{l+1}(s) \]
\[ = \ln \sum_{s \in \sigma_{l+1}} e^{[\gamma^*_l(s', s) + \beta^*_l(s)]}, \quad l = K - 1, K - 2, \ldots, 0 \]

\[ \beta^*_K(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = 0 \\ -\infty, & s \neq 0 \end{cases} \]
BCJR (cont.)

\[ p(s', s, r) = e^{\beta^*_l + 1(s) + \gamma^*_l(s', s) + \alpha^*_l(s')} \]

\[
L(u_l) = \ln \left\{ \sum_{(s', s) \in \Sigma_l^+} e^{\beta^*_l + 1(s) + \gamma^*_l(s', s) + \alpha^*_l(s')} \right\} - \ln \left\{ \sum_{(s', s) \in \Sigma_l^-} e^{\beta^*_l + 1(s) + \gamma^*_l(s', s) + \alpha^*_l(s')} \right\}
\]

\[
\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z]
\]

\[
L(u_l) = \max^*_{(s', s) \in \Sigma_l^+} [\beta^*_l + 1(s) + \gamma^*_l(s', s) + \alpha^*_l(s')] - \max^*_{(s', s) \in \Sigma_l^-} [\beta^*_l + 1(s) + \gamma^*_l(s', s) + \alpha^*_l(s')]
\]

\[ L(u_l) = \max^* (\beta^*_{l+1} + \gamma^*_l + \alpha^*_l \text{ for solid lines}) - \max^* (\beta^*_{l+1} + \gamma^*_l + \alpha^*_l \text{ for dashed lines}) \]
Log-MAP algorithm

- Initialize forward and backward recursions $\alpha_0*(s)$ and $\beta_N*(s)$
- Compute branch metrics $\{\gamma_l*(s', s)\}$
- Carry out forward recursion $\{\alpha_{l+1}*(s)\}$ based on $\{\alpha_l*(s)\}$
- Carry out backward recursion $\{\beta_{l-1}*(s)\}$ based on $\{\beta_l*(s)\}$
- Compute APP $L$-values
- Advantages over MAP algorithm:
  - Easier to implement
  - Numerically more stable
Max-log-MAP algorithm

\[
\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})
\]

- Replace max* by max, i.e., remove table look-up correction term
- Advantage: Simpler and much faster
- Forward and backward passes are equivalent to a Viterbi decoder
- Disadvantage: Less accurate, but the correction term is limited in size by \(\ln(2)\)
- Can improve accuracy by scaling with an SNR-(in)dependent scaling factor*
Example: log-MAP

\[ G(D) = [1 \quad 1/(1 + D)] \]
Example: log-MAP

- Assume $E_s/N_0 = 1/4 = -6.02$ dB
- $R = 3/8$, so $E_b/N_0 = 2/3 = -1.76$ dB

\[ \gamma_0^*(S_0, S_0) = \frac{-1}{2} L_a(u_0) + \frac{1}{2} r_0 \cdot v_0 = \frac{1}{2} (-0.8 - 0.1) = -0.45 \]

\[ \alpha_1^*(S_0) = [\gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] = -0.45 + 0 = -0.45 \]

\[ \alpha_2^*(S_0) = \max\{[\gamma_1^*(S_0, S_0) + \alpha_1^*(S_0)], [\gamma_1^*(S_1, S_0) + \alpha_1^*(S_1)]\} \]
\[ = \max\{[(-0.25) + (-0.45)], [(0.75) + (0.45)]\} \]
\[ = \max\{(-0.70, +1.20) = 1.20 + \ln(1 + e^{-|-1.9|}) = 1.34 \]
Example: log-MAP

• Assume $E_b/N_0 = 1/4 = -6.02$ dB
• $R = 3/8$, so $E_b/N_0 = 2/3 = -1.76$ dB

\[
L(u_0) = [\beta^*_1(S_1) + \gamma^*_0(S_0, S_1) + \alpha^*_0(S_0)] - [\beta^*_1(S_0) + \gamma^*_0(S_0, S_0) + \alpha^*_0(S_0)]
\]

\[
= (3.47) - (2.99) = +0.48
\]

\[
= \max^*(-0.70, +1.20) = 1.20 + \ln(1 + e^{-|-1.9|}) = 1.34
\]
Example: Max-log-MAP

- Assume $E_s/N_0 = 1/4 = -6.02$ dB
- $R = 3/8$, so $E_b/N_0 = 2/3 = -1.76$ dB

\[
\gamma_0^*(S_0, S_0) = \frac{1}{2} L_a(u_0) + \frac{1}{2} r_0 \cdot v_0
\]

\[
= \frac{1}{2} (0.8 - 0.1) = -0.45
\]

\[
\alpha_1^*(S_0) = -0.45 + 0 = -0.45
\]

\[
\alpha_2^*(S_0) = \max(-0.70, +1.20) = 1.20
\]
Example: Max-log-MAP

- Assume $E_s/N_0 = 1/4 = -6.02$ dB
- $R = 3/8$, so $E_b/N_0 = 2R E_s/N_0 = 1.8$

\[ L(u_0) = [\beta_1^*(S_1) + \gamma_0^*(S_0, S_1) + \alpha_0^*(S_0)] - [\beta_1^*(S_0) + \gamma_0^*(S_0, S_0) + \alpha_0^*(S_0)] \]

\[ = (2.79) - (2.86) = -0.07 \]
Punctured convolutional codes

- Recall that an \((n,k)\) convolutional code has a decoder trellis with \(2^k\) branches going into each state
- More complex decoding
- Solutions:
  - Bit-level encoders
  - Syndrome trellis decoding (Riedel)*
  - **Punctured codes**
    - Start with low-rate convolutional *mother* code (rate \(1/n\)?)
    - Puncture (delete) some code bits according to a predetermined pattern
    - Punctured bits are not transmitted. Hence, the code rate is increased, but the free distance of the code could be reduced
    - Decoder inserts dummy bits with neutral metrics contribution
Example: Rate 2/3 punctured from rate 1/2

The punctured code is also a convolutional code

\( d_{\text{free}} = 3 \)
Example: Rate 3/4 punctured from rate 1/2

\[ d_{\text{free}} = 3 \]
More on punctured convolutional codes

• Rate-compatible punctured convolutional (RCPC) codes:
  • Used for applications that need to support several code rates, e.g., adaptive coding or hybrid ARQ
  • Sequence of codes is obtained by repeated puncturing
  • Advantage: One decoder can decode all codes in the family
  • Disadvantage: Resulting codes may be sub-optimum

• Puncturing patterns:
  • Usually periodic puncturing patterns
  • Found by computer search
  • Care must be exercised to avoid catastrophic encoders
Best punctured codes

<table>
<thead>
<tr>
<th>Mother Code</th>
<th>Punctured Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ν</td>
<td>g(0)</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>65</td>
</tr>
<tr>
<td>6</td>
<td>155</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mother Code</th>
<th>Punctured Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>ν</td>
<td>g(0)</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>4</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>65</td>
</tr>
<tr>
<td>6</td>
<td>155</td>
</tr>
</tbody>
</table>
Tailbiting convolutional codes

- Purpose: Avoid the terminating tail (rate loss) and maintain a uniform level of protection
  - Note: Cannot avoid distance loss completely unless the length is not too short. When the length gets larger, the minimum distance approaches the free distance of the convolutional code
- Codewords can start in any state
  - This gives $2^\nu$ as many codewords
  - However, each codeword must end in the same state that it started from. This gives $2^{-\nu}$ as many codewords
  - Thus, the code rate is equal to the encoder rate
- Tailbiting codes are increasingly popular for moderate length purposes
- Some of the best known linear block codes are tailbiting codes
- Tables of optimum tailbiting codes are given in the book
- DVB: Turbo codes with tailbiting component codes
Feedforward encoder: Always possible to find an information vector that ends in the proper state (inspect the last $m$ $k$-bit input tuples)
Example: Feedback encoder

- Feedback encoder: Not always possible, for every length, to construct a tailbiting code
- For each u: Must find unique starting state
- \( L^* = 6 \) not OK
- \( L^* = 5 \) OK
- In general, \( L^* \) should not have the length of a zero input-weight cycle as a divisor
Decoding of tailbiting codes:
- Try all possible starting states (multiplies complexity by $2^\nu$), i.e., run the Viterbi algorithm for each of the $2^\nu$ subcodes and compare the best paths from each subcode.
- Suboptimum Viterbi: Initialize an arbitrary state at time 0 with zero metric and find the best ending state. Continue "one round" from there with the best subcode.
- MAP: Similar