Simplified Implementation of the MAP Decoder

Shouvik Ganguly

ECE 259B
Final Project Presentation
Introduction: MAP Decoder

- \( \hat{u}_k = \arg \max_{i \in \{0,1\}} \Pr[u_k = i | R_1^N] \)

- LAPPR

\[
\Lambda_k = \log \frac{\Pr[u_k = 1 | R_1^N]}{\Pr[u_k = 0 | R_1^N]} = \log \frac{\Pr[u_k = 1, R_1^N]}{\Pr[u_k = 0, R_1^N]}
\]

- MAP Rule:

\[
\hat{u}_k = \begin{cases} 
1, & \Lambda_k \geq 0 \\
0, & \text{otherwise}
\end{cases}
\]
BCJR Algorithm [2], [3]

\[ \Lambda_k = \log \frac{\sum_{m' \in S} \sum_{m \in S} \alpha_{k-1}(m') \gamma_k^1(m', m) \beta_k(m)}{\sum_{m' \in S} \sum_{m \in S} \alpha_{k-1}(m') \gamma_k^0(m', m) \beta_k(m)} \]

\( S = \) set of states of trellis
\( S_k = \) state of the trellis after \( k^{th} \) input.

\( \alpha_k(m) = \Pr[S_k = m, R_1^k] \)
\( \beta_k(m) = \Pr[R_{k+1}^N | S_k = m] \)
\( \gamma_i^k(m', m) = \Pr[u_k = i, S_k = m, R_k | S_{k-1} = m'] \)
BCJR Algorithm: Forward and backward recursions

\[ \alpha_k(m) = \sum_{m' \in S} \sum_{i=0}^{1} \alpha_{k-1}(m') \gamma_i^k(m', m) \]

\[ \beta_k(m) = \sum_{m' \in S} \sum_{i=0}^{1} \beta_{k+1}(m') \gamma_i^{k+1}(m, m') \]

\[ \gamma_i^k(m', m) = \Pr[u_k = i] \Pr[S_k = m | S_{k-1} = m', u_k = i] \Pr[R_k | S_{k-1} = m', u_k = i, S_k = m] \]
BCJR Algorithm: Computation and Memory Requirements

- $O(|S|^2)$ multiplications and additions for computing the metrics for each $k$
- $O(|S|^2)$ multiplications and additions for computing the LAPPR for each $k$
- Need to store forward metric for every $k$

Problematic for large block-length and codes with higher memory
A New ‘Maximum’ Function

- \( \max^*(x, y) \triangleq \log(e^x + e^y) = \max(x, y) + \log(1 + e^{-|y-x|}) \)

- **Key insight:** \( \max^* \) can be approximated by \( \max \)

- \( \max^*(x, y, z) \triangleq \max^*(\max^*(x, y), z) = \log(e^x + e^y + e^z) \)

And so on...
BCJR Algorithm : Simplification of Computation [1]

\[ \tilde{a}_k(m) \triangleq \log \alpha_k(m) \]
\[ \tilde{b}_k(m) \triangleq \log \beta_k(m) \]
\[ \tilde{c}_{i,k}(m', m) \triangleq \log \gamma_{k}^i(m', m) \]

\[ \Lambda_k \approx \max_{m, m' \in S} \left[ \tilde{a}_{k-1}(m') + \tilde{c}_{1,k}(m', m) + \tilde{b}_k(m) \right] - \max_{m, m' \in S} \left[ \tilde{a}_{k-1}(m') + \tilde{c}_{0,k}(m', m) + \tilde{b}_k(m) \right] \]

\[ \tilde{a}_k(m) \approx \max_{m' \in S, i \in \{0, 1\}} \left[ \tilde{a}_{k-1}(m') + \tilde{c}_{i,k}(m', m) \right] \]

\[ \tilde{b}_j(m) \approx \max_{m' \in S, i \in \{0, 1\}} \left[ \tilde{b}_{j+1}(m') + \tilde{c}_{i,j+1}(m', m) \right] \]
BCJR Algorithm: Simplification of Computation

\( \tilde{c}_{i,k}(m', m) = \log \Pr[u_k = i] + \log \Pr[S_k = m | S_{k-1} = m', u_k = i] + \log \Pr[R_k | S_{k-1} = m', u_k = i, S_k = m] \)

- Initializations

\[ \tilde{a}_0(m) = \begin{cases} 0, & m = s_0 \\ -\infty, & \text{otherwise.} \end{cases} \]

\[ \tilde{b}_N(m) = \begin{cases} 0, & m = s_N \\ -\infty, & \text{otherwise.} \end{cases} \]
BCJR Algorithm: Simplification of Computation

- Forward and backward metrics can be computed similarly to VA
- Problem of normalization at each step solved

Memory requirement problems
BCJR Algorithm : Reducing Memory Requirements [1]

- Status till now:

  Forward metric stored for all stages till $N$

  Backward metric stored for one stage at a time for ‘dual-maxima’ process

  Decoded vector output only after time $N$ (after the whole input is seen)

- Larger block-length increases memory requirement
BCJR Algorithm : Reducing Memory Requirements

- Key idea: Behavior of VA nearly independent of initial conditions beyond a few constraint lengths
- Use two backward decoders in tandem
- $L = 'learning period'

Received symbols delayed by $2L$
BCJR Algorithm: Reducing Memory Requirements

- Forward decoder starts at branch 0 at time $2L$
- Forward decoder stores every branch metric for each time
- Time $2L$: first backward decoder starts backwards from branch $2L$ and stores only the most recent metric till branch $L$
- Time $3L$: first backward decoder meets the computed forward metric at branch $L$
BCJR Algorithm: Reducing Memory Requirements

- Time $3L$ to time $4L$: first backward decoder moves till branch $0$ and dual maxima processor outputs soft decisions for first $L$ branches.

- Time $3L$: second backward decoder starts backwards from branch $3L$ and stores only the most recent metric till branch $2L$.

- Time $4L$: second backward decoder meets the computed forward metric at branch $2L$. 
BCJR Algorithm: Reducing Memory Requirements

- Time $4L$ to time $5L$: second backward decoder moves till branch $L$ and dual maxima processor outputs soft decisions for branches $L$ through $2L$.

- Two backward processors hop forward $4L$ branches every time $2L$ sets of backward state metrics have been generated.

- Time-sharing of dual-maxima processor.
BCJR Algorithm: Reducing Memory Requirements

- State metrics for only $2L$ branches stored by first decoder
- Soft decisions for first $2L$ branches generated at time $5L$
- Four times the complexity of a simple VA for the same convolutional code
Schematic Representation

**Figure:** Scheduling [1]
Results

Block length 10,000

Performance of rate 1/3 Turbo Code (Viterbi's Method)
A. Viterbi.  
*An Intuitive Justification and a simplified implementation of the MAP decoder for Convolutional Codes.*  

*Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate.*  