Maximum Likelihood (ML) Decoding

**Definition**

A maximum likelihood decoder chooses as decoding output the input vector $\hat{u}$ corresponding to the codeword $\hat{v}$ that maximizes the likelihood function $p(r|v)$, i.e.,

$$\hat{v} = \arg \max_{v \in C} p(r|v)$$

- This decoding rule minimizes the block error probability $P_B$.
- For channels without memory we can write

$$p(r|v) = \prod_{i=1}^{N} p(r_i|v_i)$$

- BSC ($\epsilon < 0.5$):

$$\arg \max_{v \in C} p(r|v) = \arg \max_{v \in C} \left( \frac{\epsilon}{1-\epsilon} \right)^{d_H(r,v)} = \arg \min_{v \in C} d_H(r,v)$$

A Posteriori Probability (APP) Decoding

**Definition**

An a posteriori probability decoder delivers at its output the posterior probabilities $p(u_i|r)$ for all input symbols $u_i \in u$, i.e.,

$$p(u_i = x|r) = \sum_{u_i = x} p(u|r) = \sum_{u_i = x} \frac{p(r|u)p(u)}{p(r)} , \quad x \in \{0,1\}$$

- The posterior probabilities can be used as soft output and passed to other components in a receiver (see Chapter 4).
- A symbol-by-symbol maximum a posteriori (s/s-MAP) decoder chooses as decoder output the input symbols

$$\hat{u}_i = \max_{u_i \in \{0,1\}} p(u_i|r) , \quad i = 1,\ldots,K$$

- This decoding rule minimizes the bit error probability $P_b$.

Efficient ML decoding with the Viterbi algorithm
Example

\[ G(D) = (1 + D + D^2, 1 + D^2), \quad d_{\text{free}} = 5 \]

\[ u = (1, 1, 0, 1), \quad v = (11, 01, 01, 00, 10, 11) \]

Decode \( r = (11, 11, 01, 10, 10, 11) \) (two errors)

\[ \text{BSC: } \max p(r|v) \Leftrightarrow \min d_H(r,v) \]

Task: find code sequence in trellis that is closest to \( r \).

\[ b = (11, 11, 01, 10, 10, 11) \]

\[ \Rightarrow \text{both errors corrected} \]

Maximal Likelihood (ML) Decoding

- Consider a channel without memory (e.g., AWGN, BSC)
- An ML decoder has to find the codeword \( v \) that maximizes
  \[ p(r|v) = \prod_{i=1}^{N} p(r_i|v_i) \]
  for a given \( r \), or, equivalently, maximizes
  \[ \Lambda \overset{\text{def}}{=} \log p(r|v) = \sum_{i=1}^{N} \log p(r_i|v_i) = \sum_{i=1}^{N} \lambda (r_i|v_i) \]

Example (Single parity-check (SPC) code)

\[ N = 3, \quad K = 2 \]

\[ G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \]

- Can we efficiently reuse the branch metric values \( \lambda (r_i|v_i) \)?

Efficient representation of codeword metric

- Idea:
  1. merge branches with equal metric \( \lambda (r_i|v_i) \) together
  2. locally compute cumulative metric of codeword paths

- We arrive at a trellis representation of the SPC code

Question:

- How can we construct a trellis of an arbitrary code?

The syndrome trellis of a block code

Definition

For a code \( C \) with parity-check matrix \( H = (h_1, h_2, \ldots, h_N) \) and a given vector \( v \) the partial syndrome at level \( t = 1, \ldots, N \) is defined as

\[ \sigma_t(v) = v_1 h_1 + v_2 h_2 + \cdots + v_t h_t \]

For \( t = 0 \) it is defined as \( \sigma_0(v) = 0 \).

- From \( vH^T = 0 \) it follows that \( \sigma_t(v) = 0 \) for any \( v \in C \)
- A syndrome trellis of a code \( C \) is constructed by using the partial syndromes at level \( r \) as trellis states for all \( v \in C \)
- The trellis is constructed recursively by expanding all states at each level \( r \) with new symbols \( v_{r+1} = 0 \) and \( v_{r+1} = 1 \) to a full trellis
- Then all paths that do not lead to the state \( \sigma_N(v) = 0 \) are expurgated from the full trellis
The Viterbi algorithm

**Input:** received vector $r$ and a trellis representation of the code

- Initialize $\Lambda_0(\sigma_0) = 0$ and $\Lambda(\sigma_0) = -\infty$ for all $\sigma \neq 0$
- For all trellis levels $i = 1, \ldots, N$:
  1. For each state $\sigma_i$ at level $i$ identify the set $S = \{\sigma_{i-1} \mid \sigma_{i-1} \xrightarrow{w_{i-1}} \sigma_i\}$ of predecessor states that are connected to it by a branch
  2. Compute and store the state metric
     \[ \Lambda_i(\sigma_i) = \max_{\sigma_{i-1}} \left( \Lambda_{i-1}(\sigma_{i-1}) + \lambda(\sigma_{i-1} \xrightarrow{w_{i-1}} \sigma_i) \right) \]
     where $\lambda(\sigma_{i-1} \xrightarrow{w_{i-1}} \sigma_i) = \lambda(r_i | v_i)$ is the corresponding branch metric
  3. Remove those incoming branches that do not give the maximum (equivalently we can store the state $\sigma_{i-1}$ that results in $\Lambda_i(\sigma_i)$)
- The maximum likelihood path has the metric $\Lambda_N(\sigma_N) = 0$ and can be found by tracing back the surviving branches

**Output:** Decoded code sequence $\hat{v}$ and input sequence $\hat{u}$

Efficient ML decoder implementation

- We want to compute
  \[ \hat{v} = \arg \max_{v \in \mathbb{C}} \sum_{i=1}^{N} \lambda(r_i | v_i), \quad \lambda(r_i | v_i) = \log p(r_i | v_i) \]
- The branch metric values $\lambda(r_i | v_i)$ can be efficiently reused in a trellis representation of the code $C$
- The path metric $\Lambda(v) = \sum_{i=1}^{N} \lambda(r_i | v_i)$ of a codeword can be computed recursively in the trellis
- **Key observation:** the maximization can be performed locally

From hard decision to soft decision

- For the BSC we use the metric
  \[ \lambda(r_i, v_i) = d_H(r_i, v_i) \]
  and replace the maximization by minimization in each stage
- For the AWGN channel we have
  \[ p(r | v) = \prod_{i=1}^{N} p(r_i | v_i), \quad p(r_i | v_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{(r_i - x_i)^2}{2\sigma^2} \right) \]
  where $x_i = +1$ if $v_i = 0$ and $x_i = -1$ if $v_i = 1$
- We want to maximize
  \[ \sum_{i=1}^{N} \log p(r_i | v_i) = -\text{const.} \sum_{i=1}^{N} (r_i - x_i)^2 \]
- This is equivalent to minimizing the Euclidean distance
  \[ d_E(r, x) = \sum_i (r_i - x_i)^2 \]
  using $\lambda(r_i | v_i) = d_E(r_i, x_i)$
- Simpler to implement: maximize $\lambda(r_i | v_i) = r_i \cdot x_i$
Quantization of the AWGN channel

- The AWGN channel with continuous output is often quantized into a discrete memoryless channel (DMC)

Example: 8 level quantization

\[ r_i \in \{ 0_4, 0_3, 0_2, 0_1, 1_1, 1_2, 1_3, 1_4 \} \]

Transition probabilities:

\[
\begin{array}{ccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

- It is convenient to shift and scale the metric according to

\[
\lambda(r_i|v) = c_2 \left( \log p(r_i|v) + c_1 \right)
\]

Example: \( c_1 = -\min \{ \log p(r_i|0), \log p(r_i|1) \} \) and \( c_2 = 1.5 \) with rounding to integers yields:

\[
\begin{array}{ccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Pairwise Error Probability

- Assume that \( v \in C \) is transmitted and consider ML decoding of \( r \)

- A decoding error occurs if for some \( v' \in C \) at distance \( d = d_H(v,v') \) we have

\[
p(r|v') > p(r|v)
\]

- The probability that a code word at distance \( d \) is more likely than \( v \) is called the pairwise error probability \( p_d \)

BSC: \( r = v + e, e = w_H(e) \) errors at positions where \( v \neq v' \)

\[
p_d < \sum_{e \in \{ \pm 1 \}^d} \left( \begin{array}{c} d \\ d/2 \end{array} \right) e^d (1-e)^{d-e}, \quad d \text{ odd}
\]

- For even \( d \) the term for \( e = d/2 \) adds with probability 1/2

- Both cases can be upper bounded by the Bhattacharyya parameter \( B \):

\[
p_d < \left( 2\sqrt{e(1-e)} \right)^{d/2} B^{d/2}
\]

ML Decoding Error Probability

Union Bound

- The \( E(v \rightarrow v') \) denote the event that \( v' \) is selected instead of \( v \)

- The block error probability is the union of such events

\[
P_B = P \left( \bigcup_{v' \in C} E(v \rightarrow v') \right) \leq \sum_{v' \in C} P(E(v \rightarrow v'))
\]

- Using the pairwise error probability \( p_d \) we obtain

\[
P_B \leq \sum_{d=d_{\text{min}}}^{N} A_d p_d
\]

where \( A_d \) denotes the number of codewords of weight \( d \)

- Using the weight enumerator function

\[
A(X) = A_1 X + A_2 X^2 + \cdots + A_N X^N
\]

(which excludes the zero codeword, i.e., \( A_0 = 1 \)) we can write

\[
P_B \leq A(X) \bigg|_{X=p_d}
\]