Lecture 8: Convolutional Codes
Introducing memory

• A binary linear block encoder is a linear transformation $\mathbb{F}_2^k \rightarrow \mathbb{F}_2^n$.

• What about using a Linear time-invariant linear system for encoding?

• Convolutional codes consider small $k$ and $n$, but introduce memory into the encoding process of a sequence of consecutive blocks.

• We may see this as the convolution of a sequence of information blocks $\{u_i\}$ with a matrix $G$ of impulse responses, in order to generate a sequence of coded blocks $\{c_i\}$. 
A \( k \times n \) Moving Average (MA) system is defined by:

\[
\begin{align*}
  c_i^{(1)} &= \sum_{\ell=0}^{m_{1,1}} g^{(1,1)}_{\ell} u_{i-\ell} + \cdots + \sum_{\ell=0}^{m_{k,1}} g^{(k,1)}_{\ell} u_{i-\ell} \\
  c_i^{(2)} &= \sum_{\ell=0}^{m_{1,2}} g^{(1,2)}_{\ell} u_{i-\ell} + \cdots + \sum_{\ell=0}^{m_{k,2}} g^{(k,2)}_{\ell} u_{i-\ell} \\
  \vdots \\
  c_i^{(n)} &= \sum_{\ell=0}^{m_{1,n}} g^{(1,n)}_{\ell} u_{i-\ell} + \cdots + \sum_{\ell=0}^{m_{k,n}} g^{(k,n)}_{\ell} u_{i-\ell}
\end{align*}
\]
• Defining a vector output sequence $c_0, c_1, c_2, \ldots$ such that $c_i = (c_i^{(1)}, \ldots, c_i^{(n)})$ and a vector input sequence $u_0, u_1, u_2, \ldots$ such that $u_i = (u_i^{(1)}, \ldots, u_i^{(k)})$, we can write

$$c_i = \sum_{\ell=0}^{m} u_{i-\ell} G_{\ell}$$

where we let $m = \max\{m_{(i,j)}\}$.

• We obtain a block-Toeplitz notation

$$\begin{bmatrix}
G_0 & G_1 & \cdots & G_m & 0 & \cdots \\
0 & G_0 & & G_{m-1} & G_m & \\
\vdots & 0 & \ddots & \vdots & G_{m-1} & \ddots \\
G_0 & & & 0 & G_0 & \ddots \\
0 & \cdots & & G_0 & & \ddots
\end{bmatrix}$$

$$\begin{align*}
(c_0, c_1, c_2, c_3, \ldots) &= (u_0, u_1, u_2, u_3, \ldots)
\end{align*}$$

• The impulse responses $g^{(i,j)}$ are called the code generators.
$D$-transform

- $D$-transform domain

\[ u_i \rightarrow u(D) = \sum_i u_i D^i \quad \text{(Laurent series)} \]

- Convolutional encoding in the $D$-transform domain:

\[ c(D) = u(D)G(D) \]

or, equivalently,

\[
(c_1(D), \ldots, c_n(D)) = (u_1(D), \ldots, u_k(D)) \begin{bmatrix}
g_{1,1}(D) & g_{1,2}(D) & \cdots & g_{1,n}(D) \\
g_{2,1}(D) & g_{2,2}(D) & \cdots & g_{2,n}(D) \\
\vdots & \vdots & \ddots & \vdots \\
g_{k,1}(D) & g_{k,2}(D) & \cdots & g_{k,n}(D)
\end{bmatrix}
\]
Example: a $(2, 1)$ convolutional code

\[ C_i^{(1)} = u_i + u_{i-2} \]

\[ C_i^{(2)} = u_i + u_{i-1} + u_{i-2} \]

- Generator matrix:

\[ G(D) = [1 + D^2, 1 + D + D^2] \]
Encoder canonical forms

- A code $C$ is defined as the set of all output sequences (code sequences).

- As for block codes, a convolutional code $C$ may have several input-output encoder implementations.

- We seek encoders in canonical form: a general problem in system theory is for a given system, defined as the ensemble of all its output sequences, what is the minimal canonical realization?

- State-space representation (ABCD):

  $$s_{i+1} = s_i A + u_i B, \quad c_i = s_i C + u_i D$$

  a minimal representation is a representation with the minimum number of state variables.
More on the \((2,1)\) example

- In the \((2,1)\) example of before, the state is defined as the content of the memory elements,

\[ s_i = (u_{i-1}, u_{i-2}) \]

therefore we have

\[ s_{i+1} = s_i \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} + u_i \begin{bmatrix} 1 & 0 \\ \end{bmatrix} \]

and

\[ c_i = s_i \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} + u_i \begin{bmatrix} 1 & 1 \end{bmatrix} \]
Generalization ...

- A convolutional code can be seen as a block code defined on the field $\mathbb{F}_q(D)$ of rational functions over $\mathbb{F}_q$.
- Roughly speaking: rational functions are to polynomials as rationals $\mathbb{Q}$ to the integers $\mathbb{Z}$.
- Generalizing what seen before, we can consider $\mathbf{G}(D)$ with rational elements $g_{i,j}(D)$.
- In system theory, this corresponds to AR-MA linear systems.
- The code is preserved by elementary row operations.
- It follows that for any $\mathbf{G}(D)$, we can find a systematic generator matrix in the form
  \[ \mathbf{G}(D) = [\mathbf{I} | \mathbf{P}(D)] \]
  where $\mathbf{I}$ is the $k \times k$ identity, and $\mathbf{P}(D)$ is a $k \times (n - k)$ matrix of rational functions.
• For the \((2, 1)\) example with memory \(m = 2\) and generator matrix:

\[
G(D) = [1 + D^2, 1 + D + D^2]
\]

we have:

\[
G'(D) = \left[ 1, \frac{1 + D + D^2}{1 + D^2} \right]
\]
A state-space realization with $m$ binary state variables is a finite-state machine (FSM) with a state space $\Sigma = \mathbb{F}_2^m$.

In general, a FSM is described by its state transition diagram, i.e., by a graph with $|\Sigma|$ vertices, corresponding to all possible state configurations, and edges connecting those states for which a transition is possible.

Each edge $(s, s') \in \Sigma \times \Sigma$ is labeled by input and output vectors $b \in \mathbb{F}_2^k$ and $c \in \mathbb{F}_2^n$, corresponding to the state transition between $s$ and $s'$. 
Example: the \((2, 1)\) 4-state code

- The input-output labels of the state diagram depend on the encoding function: this example corresponds to the feedforward (non-systematic) encoder for the \((2, 1)\) example seen before.
• An alternative representation consists of the trellis section, i.e., by a bipartite graph with $|\Sigma|$ state vertices on the left and $|\Sigma|$ state vertices on the right.

• Left vertices represent the possible states at time $i$, and right vertices represent the possible states at time $i + 1$. Edges represent the possible state transitions corresponding to input $u_i$ and output $c_i$.

• The trellis representation follows from the state transition diagram by introducing the time axis.

• A trellis diagram for a convolutional code consists of the concatenation of an infinite number of trellis sections.

• Given an initial state at time $i = 0$, an input sequence $u(D)$ determines an output sequence $c(D)$ and a state sequence $s(D)$ that correspond to a path in the trellis.
Example: the $(2, 1)$ 4-state code

\[
\begin{align*}
(0, 0) & \quad 0/00 \quad 1/11 \\
(1, 0) & \quad 0/11 \quad 1/00 \\
(0, 1) & \quad 1/01 \quad 0/01 \\
(1, 1) & \quad 0/10 \quad 1/01 
\end{align*}
\]
The Factor Graph for a convolutional code

- Three types of variable nodes: information bitnodes $u_i$, coded bitnodes $c_i$ and states nodes $s_i$.

- The function nodes correspond to the state and output mappings

$$s_{i+1} = s_i A + u_i B, \quad c_i = s_i C + u_i D$$
Finding $G(D)$ from the state equation

• Consider the ABCD minimal state space realization:

$$s_{i+1} = s_i A + u_i B, \quad c_i = s_i C + u_i D$$

• By applying $D$-transform we obtain

$$D^{-1}s(D) = s(D)A + u(D)B, \quad c(D) = s(D)C + u(D)D$$

• Eliminating the state $s(D)$, we arrive at $c(D) = u(D)G(D)$ with

$$G(D) = B \left( D^{-1}I + A \right)^{-1} C + D$$
MAP decoding of convolutional codes

- We consider the transmission of the block code $C_N$ obtained by trellis termination of a convolutional code, over a memoryless channel with input $c_i \in \mathbb{F}_2^n$ and output $y_i \in \mathcal{Y}$.

- Block MAP decoding rule (aka, Maximum Likelihood (ML) decoding rule):

$$\hat{c} = \arg \max_{c \in C_N} \sum_{i=0}^{N-1} \log P_{Y|X}(y_i|c_i)$$

- Since $|C_N| = 2^{k(N-\nu_{\text{max}})}$ and $N - \nu_{\text{max}}$ is generally large, the brute-force evaluation of the MAP decoding rule is intractable.
The Viterbi Algorithm

- The ML decision metric is additive and that the codewords are represented by paths in the code trellis.

- In this case, the ML decoding rule can be computed efficiently by the Viterbi Algorithm (VA).

- For a state transition \((s', s)\), let 
  \[c(s', s) = (c^{(1)}(s', s), \ldots, c^{(n)}(s', s))\]
  denote the corresponding block of coded symbols.

- We define the branch metric at trellis section \(i\) as
  \[
  M_i(c) = \alpha \log P_{Y|X}(y_i|c) - \beta, \quad c \in \mathbb{F}_2^n
  \]
  where \(\alpha > 0\) and \(\beta\) are suitable constant.
• The VA maintains one path metric for each surviving path terminating at each state, and recursively updates the path metrics as follows:

1. Initialization: \( M_0(0) = 0, M(s) = -\infty \) for all \( s \neq 0 \).
2. Add Compare and Select (ACS) recursion: for \( i = 1, 2, \ldots, N \), and for all states \( s \in \Sigma \), let

\[
M_i(s) = \max_{s' \in P(s)} \{ M_{i-1}(s') + M_{i-1}(c(s', s)) \}
\]

where \( P(s) \) is the set of “parent” states of \( s \), i.e., the set of states \( s' \in \Sigma \) for which there exists a transition \( (s', s) \) in the code state diagram.
3. The \( n \)-tuple of symbols \( c(s', s) \) achieving the maximum in the ACS step is added to the survivor path terminating in state \( s \) at time \( i \).
4. Final decision (trace-back): the maximum of the total metric is obtained as \( M_N(0) \). The MAP decoded codeword \( \hat{c} \) is the corresponding survivor path.
Example: the \((2, 1)\) 4-state code over the BSC
Example: the \((2, 1)\) 4-state code over the BSC
Practical implementation issues

• **Path metric re-normalization:** the output of the VA does not change if at each step the maximum path metric is subtracted from all path metrics.

• **Minimizing instead of maximizing:** the output of the VA does not change if we choose $\alpha < 0$ and min instead of max in the ACS step.

• **Register exchange vs. traceback:** .... better explained through an example ...

• **Handling puncturing:** the contribution to the branch metric of a punctured bit is zero.

• **Handling ties:** if two competing candidate paths terminating in the same state have the same metric, the VA takes an arbitrary, or random, decision, with no impact on the average error probability.
Optimality of the VA

**Theorem 13.** The final surviving path in the VA is the ML-decoded codeword.

\[\square\]

Proof:

- Claim: if \( M_-(b) < M_-(a) \) then the semi-path \( b_- \) cannot be part of the MAP codeword and can be dropped from the metric count.
• Compare the 4 path metrics

\[ M_-(b) + M_+(b), \quad M_-(b) + M_+(a), \quad M_-(a) + M_+(b), \quad M_-(a) + M_+(a) \]

• If \( M_-(b) < M_-(a) \) then

\[
\max \{ M_-(b) + M_+(b), M_-(b) + M_+(a) \} < \max \{ M_-(a) + M_+(b), M_-(a) + M_+(a) \}
\]
End of Lecture 8