Week 5 : Session 1.

In the last section, we talked about convolutional code. Convolutional code is a sequential circuit, so it has memory and it can be represented using a state diagram and Trellis diagram.

In this section, we'll talk about how to decode a convolutional code. We'll exploit the Trellis diagram of the convolutional code to decode a convolutional code.

This is also known as Viterbi Decoding, named after Andrew Viterbi.

We'll consider a rate $R = \frac{1}{n}$ convolutional code which has memory m. (ie) Viterbi decoding of $(n, 1, m)$ code. We'll use this example to illustrate the decoding of convolutional code.

Example : Viterbi decoding of $(2, 1, 2)$ convolutional code ($k = 1$) on BSC.

Refer 8 (i) of Digital Communication System 1.

Week 5 : Session 2

We'll continue the discussion on decoding of convolutional codes. In this section, we're going to talk about MAP decoding algorithm for convolutional code.

We've seen in case of viterbi decoding, the output is just 0's and 1's. We only get an estimate on the code sequence or the information sequence, but we do not get information about how likely those bits are going to be 0 and 1.

So, we are going to talk about an algorithm, which is known as BCJR algorithm, which will give us soft estimates on the information sequence. So, we will not only know whether the bits are 0's and 1's, but we'll also get an information about how much likely it's going to be 0 or 1.

BCJR Algorithm for Convolutional Codes.

We are interested in minimizing the BER. If the information bit u_k is not equal to the estimate $\hat{u}_k$, then there is an error. The BER will tell us how many such bits / what fraction of information bits are in error.

To minimize the bit error rate (BER), the a-posteriori probability (Probability that an information bit u_k is correctly decoded) (ie) $P(\hat{u}_k = u_k \mid r)$ must be maximized.

If we are able to maximize this, we'll be minimizing the BER. An algorithm that maximizes $P(\hat{u}_k = u_k \mid r)$ is called maximum a-posteriori probability (MAP) decoder.

In 1974, Bahl, Cocke, Jelinek and Raviv from Israel introduced a MAP decoder that can be applied to any linear code. This is known as BCJR algorithm.

The BCJR algorithm computes the a-posteriori L-values (APP L-value) of each information bit.

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

The decoder output is given by

$$\hat{u}_l = \begin{cases} +1 & \text{if } L(u_l) > 0 \\ -1 & \text{if } L(u_l) < 0 \end{cases}, \quad l = 0, 1, \ldots, k-1.$$

---------- (2)

The APP value $P(u_l = +1|r)$ as follows:

$$P(u_l = +1|r) = \frac{P(u_l = +1, r)}{P(r)} = \frac{\sum_{u \in U_l^+} p(r|v) \cdot P(u)}{\sum_u p(r|v) P(u)}$$

---------- (3)

where,

- U_l^+ is the set of all information sequences u such that $u_l = +1$.
- v is the transmitted codeword corresponding to the information sequence u.
- $p(r|v)$ is the pdf of the received sequence r given v.

Similarly,

$$P(u_l = -1|r) = \frac{P(u_l = -1, r)}{P(r)} = \frac{\sum_{u \in U_l^-} p(r|v) \cdot P(u)}{\sum_u p(r|v) \cdot P(u)}$$

The expression in equation (1) for the APP L-value becomes

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

---------- (4)

where U_l^- is the set of all information sequences u such that $u_l = -1$.

For short constraint length convolutional codes, equation ④ can be simplified by employing a recursive computational procedure based on the trellis structure of the code.

Equation ③ can be rewritten as

$$P(u_k = +1|r) = \frac{P(u_k = +1, r)}{P(r)} = \frac{\sum_{(s',s)\in\Sigma_k^+} P(s_k = s', s_{k+1} = s, r)}{P(r)} \quad \text{⑤}$$

where Σ_k^+ is the set of all state pairs $s_k = s'$ and $s_{k+1} = s$ that correspond to the input bit $u_k = +1$ at time k.

Similarly, equation ④ can be written as

$$L(u_k) = \ln\left\{\frac{\sum_{(s',s)\in\Sigma_k^+} P(s_k = s', s_{k+1} = s, r)}{\sum_{(s',s)\in\Sigma_k^-} P(s_k = s', s_{k+1} = s, r)}\right\} \quad \text{⑥}$$

where Σ_k^- is the set of all state pairs $s_k = s'$ and $s_{k+1} = s$, that correspond to the input bit $u_k = -1$ at time k.

The Joint PDFs $P(s', s, r)$ in equation ⑥ can be evaluated recursively.

$$\begin{aligned} P(s', s, r) &= P(s', s, r_{t<k}, r_k, r_{t>k}) \\ &= P(r_{t>k} | s', s, r_{t<k}, r_k) \cdot P(s', s, r_{t<k}, r_k) \\ &= P(r_{t>k} | s', s, r_{t<k}, r_k) \cdot \\ &\quad P(s, r_k | s', r_{t<k}) \cdot P(s', r_{t<k}) \\ &= P(r_{t>k} | s) \cdot P(s, r_k | s') \cdot P(s', r_{t<k}) \end{aligned} \quad \text{⑦}$$

where $r_{t<k}$ represents the portion of the received sequence r before time k.

$r_{t>l}$ represents the portion of the received sequence r after time l.

Defining,

$$\alpha_l(s') \equiv p(s', r_{t<l}) \quad \text{—— (8)}$$

$$\gamma_l(s', s) \equiv p(s, r_l | s') \quad \text{—— (9)}$$

$$\beta_{l+1}(s) \equiv p(r_{t>l} | s) \quad \text{—— (10)}$$

Equation (7) can be written as

$$p(s', s, r) = \beta_{l+1}(s)\ \gamma_l(s', s)\ \alpha_l(s') \quad \text{—— (11)}$$

The expression for the probability $\alpha_{l+1}(s)$ can now be rewritten as

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

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

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

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

$$= \sum_{s' \in \sigma_l} \gamma_l(s', s)\ \alpha_l(s') \quad \text{(Recursive)} \quad \text{—— (12)}$$

where, σ_l is the set of all states at time l.

Similarly, expression of the probability $\beta_k(s')$ can be written as

$$\beta_k(s') \equiv p(r_{t>(k-1)} \mid s')$$

$$= \sum_{s \in \sigma_{k+1}} p(r_{t>(k-1)}, s \mid s')$$

$$= \sum_{s \in \sigma_{k+1}} p(r_{t>k}, r_k, s \mid s')$$

$$= \sum_{s \in \sigma_{k+1}} p(r_{t>k} \mid s', s, r_k)\, p(s, r_k \mid s')$$

$$= \sum_{s \in \sigma_{k+1}} p(r_{t>k} \mid s)\, p(s, r_k \mid s')$$

$$= \sum_{s \in \sigma_{k+1}} \beta_{k+1}(s)\, \gamma_k(s', s)$$

(Recursive) (13)

where σ_{k+1} is the set of all states at time $k+1$.

The branch metric $\gamma_k(s', s)$ can be written as

$$\gamma_k(s', s) = p(s, r_k \mid s') = \frac{p(s', s, r_k)}{p(s')}$$

$$= \left[\frac{p(s', s)}{p(s')}\right] \left[\frac{p(s', s, r_k)}{p(s', s)}\right]$$

$$= p(s \mid s')\, p(r_k \mid s', s) = p(u_k)\, p(r_k \mid v_k) \quad (14)$$

where u_k is the input bit and v_k the output bits corresponding to the state transition $s' \to s$ at time k.

For a continuous output AWGN channel, if $s' \to s$ is a valid transition.

$$\gamma_t(s', s) = P(u_t)\, p(r_t | v_t) = P(u_t) \left(\sqrt{\frac{E_s}{\pi N_0}}\right)^n e^{-\frac{E_s}{N_0}\|r_t - v_t\|^2} \quad (15)$$

where, $\|r_t - v_t\|^2$ is the squared Euclidean distance between the (normalized by $\sqrt{E_s}$) received branch r_t and the transmitted branch v_t at time t.

On the other hand, if $s' \to s$ is not a valid state transition, $P(s|s')$ and $\gamma_t(s', s)$ are both zero.

Initial conditions for recursion

⊙ Forward recursion:

$$\alpha_0(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases} \quad (16)$$

⊙ Backward recursion:

$$\beta_K(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases} \quad (17)$$

Step 1 : Initialize the forward and backward metrics $\alpha_0(s)$ and $\beta_K(s)$ using equation (16) and (17).

Step 2 : Compute the branch metrics $\gamma_t(s', s)$, $t = 0, 1, \ldots, K-1$ using equation (14)

Step 3 : Compute the forward metrics $\alpha_{t+1}(s)$, $t = 0, 1, \ldots, K-1$ using equation (12)

Step 4 : Compute the backward metrics $\beta_t(s')$, $t = K-1, K-2, \ldots, 0$ using equation (13)

Step 5 : Compute the APP L-values $L(u_t)$ using equations (6) and (11).

Step 6 : Compute the hard decisions $\hat{u}_t$ using equation (2)

Example:

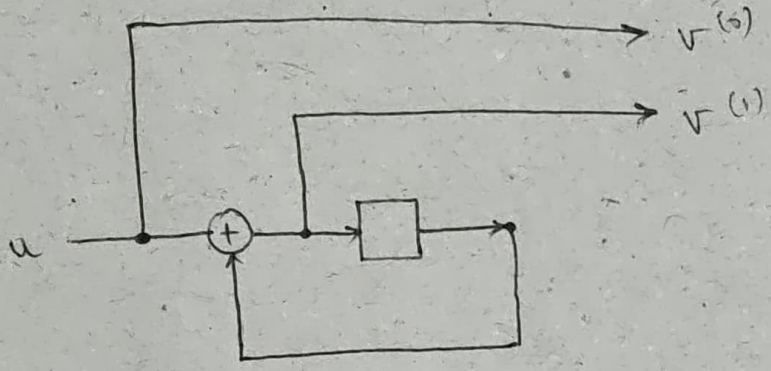
Consider the (2,1,1) systematic recursive convolutional code with generator matrix

$$G(D) = \left[1 \quad \frac{1}{1+D}\right]$$

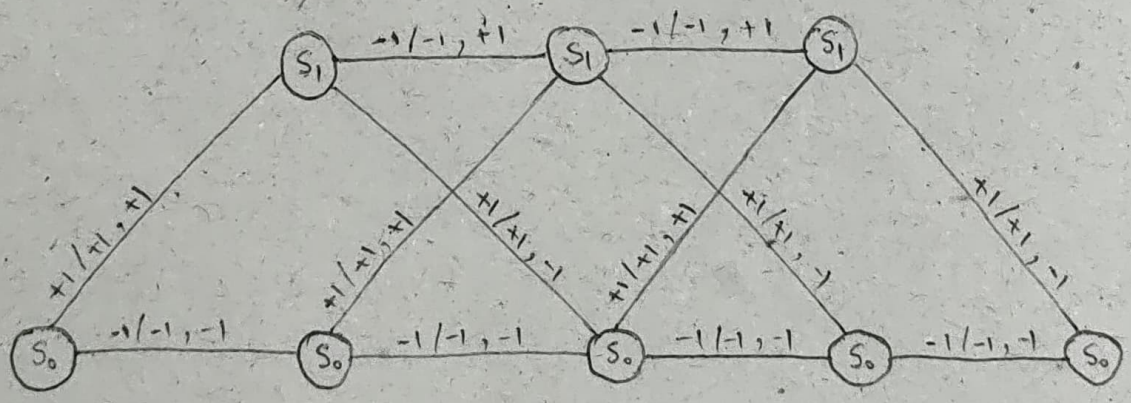
We assume an AWGN channel with SNR of $\frac{E_s}{N_0} = \frac{1}{4}$ (−6.02 dB). The received vector (normalized by $\sqrt{E_s}$) is given by

$$r = (r_0, r_1, r_2, r_3) = \left(r_0^{(0)}, r_0^{(1)}; r_1^{(0)}, r_1^{(1)}; r_2^{(0)}, r_2^{(1)}; r_3^{(0)}, r_3^{(1)}\right)$$

$$= (+0.8, +0.1; +1.0, -0.5; -1.8, +1.1; +1.6, -1.6).$$



(a) Encoder diagram



(b) Trellis diagram

Step 1: Initialize the forward and backward metrics $\alpha_0(s)$ and $\beta_K(s)$ using equations (16) and (17)

Initial conditions for recursion.

⊙ Forward recursion:

$$\alpha_0(s) = \begin{cases} 1, & s = 0 \\ 0, & s \neq 0 \end{cases}$$

⊙ Backward recursion:

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

Step 2: Compute the branch metrics $\gamma_k(s', s)$, $k = 0, 1, \ldots, K-1$ using equation (14).

$$\gamma_0(S_0, S_0) = e^{-0.45} = 0.6376$$
$$\gamma_0(S_0, S_1) = e^{0.45} = 1.5683$$
$$\gamma_1(S_0, S_0) = e^{-0.25} = 0.7788$$
$$\gamma_1(S_0, S_1) = e^{0.25} = 1.2840$$
$$\gamma_1(S_1, S_1) = e^{-0.75} = 0.4724$$
$$\gamma_1(S_1, S_0) = e^{0.75} = 2.1170$$
$$\gamma_2(S_0, S_0) = e^{0.35} = 1.4191$$
$$\gamma_2(S_0, S_1) = e^{-0.35} = 0.7047$$
$$\gamma_2(S_1, S_1) = e^{1.45} = 4.2631$$
$$\gamma_2(S_1, S_0) = e^{-1.45} = 0.2346$$
$$\gamma_3(S_0, S_0) = e^{0} = 1.0$$
$$\gamma_3(S_1, S_0) = e^{1.6} = 4.9530$$

Step 3 : Compute the forward metrics $\alpha_{t+1}(s)$, $t = 0, 1, \ldots, K-1$ using equation (12)

$$\alpha_1(s_0) = \alpha_0(s_0)\,\gamma_0(s_0, s_0) = 0.6376 \quad (0.2890)$$

$$\alpha_1(s_1) = \alpha_0(s_0)\,\gamma_0(s_0, s_1) = 1.5683 \quad (0.7110)$$

$$\alpha_2(s_0) = \alpha_1(s_0)\,\gamma_1(s_0, s_0) + \alpha_1(s_1)\,\gamma_1(s_1, s_0)$$
$$= 3.8167 \quad (0.7099)$$

$$\alpha_2(s_1) = \alpha_1(s_0)\,\gamma_1(s_0, s_1) + \alpha_1(s_1)\,\gamma_1(s_1, s_1)$$
$$= 1.5595 \quad (0.2901)$$

$$\alpha_3(s_0) = \alpha_2(s_0)\,\gamma_2(s_0, s_0) + \alpha_2(s_1)\,\gamma_2(s_1, s_0)$$
$$= 5.7821 \quad (0.3824)$$

$$\alpha_3(s_1) = \alpha_2(s_0)\,\gamma_2(s_0, s_1) + \alpha_2(s_1)\,\gamma_2(s_1, s_1)$$
$$= 9.3379 \quad (0.6176).$$

Step 4 : Compute the backward metrics $\beta_t(s)$, $t = K-1, K-2, \ldots, 0$ using equation (13).

$$\beta_3(s_0) = \beta_4(s_0)\,\gamma_3(s_0, s_0) = 1.0 \quad (0.1680)$$

$$\beta_3(s_1) = \beta_4(s_0)\,\gamma_3(s_1, s_0) = 4.9530 \quad (0.8320)$$

$$\beta_2(s_0) = \beta_3(s_0)\,\gamma_2(s_0, s_0) + \beta_3(s_1)\,\gamma_2(s_0, s_1)$$
$$= 4.9095 \quad (0.1870)$$

$$\beta_2(s_1) = \beta_3(s_0)\,\gamma_2(s_1, s_0) + \beta_3(s_1)\,\gamma_2(s_1, s_1)$$
$$= 21.3497 \quad (0.8130)$$

$$\beta_1(s_0) = \beta_2(s_0)\,\gamma_1(s_0, s_0) + \beta_2(s_1)\,\gamma_1(s_0, s_1)$$
$$= 31.2365 \quad (0.6040)$$

$$\beta_1(s_1) = \beta_2(s_0)\,\gamma_1(s_1, s_0) + \beta_2(s_1)\,\gamma_1(s_1, s_1)$$
$$= 20.4790 \quad (0.3960)$$

Step 5: Compute the APP L-values $L(u_k)$, using equations (6) and (11).

$$L(u_0) = \ln\left\{\frac{\alpha_0(s_0)\,\gamma_0(s_0,s_1)\,\beta_1(s_1)}{\alpha_0(s_0)\,\gamma_0(s_0,s_0)\,\beta_1(s_0)}\right\} = 0.4778$$

$$L(u_1) = \ln\left\{\frac{\alpha_1(s_0)\,\gamma_1(s_0,s_1)\,\beta_2(s_1) + \alpha_1(s_1)\,\gamma_1(s_1,s_0)\,\beta_2(s_0)}{\alpha_1(s_0)\,\gamma_1(s_0,s_0)\,\beta_2(s_0) + \alpha_1(s_1)\,\gamma_1(s_1,s_1)\,\beta_2(s_1)}\right\}$$

$$= 0.6154$$

$$L(u_2) = \ln\left\{\frac{\alpha_2(s_0)\,\gamma_2(s_0,s_1)\,\beta_3(s_1) + \alpha_2(s_1)\,\gamma_2(s_1,s_0)\,\beta_3(s_0)}{\alpha_2(s_0)\,\gamma_2(s_0,s_0)\,\beta_3(s_0) + \alpha_2(s_1)\,\gamma_2(s_1,s_1)\,\beta_3(s_1)}\right\}$$

$$= -1.0301$$

Step 6: Compute the hard decisions $\hat{u}_k$ using equation (2)

$$\hat{u} = (+1, +1, -1)$$

The beauty of this algorithm is, we are not only getting estimates on bits 0's and 1's, but the magnitudes (L-values) will tell us how much confidence we have / what is the likelihood of this being +1 or -1.

If the magnitude values are large (either towards $-\infty$ or $-\infty$), then we have lot more confidence.