

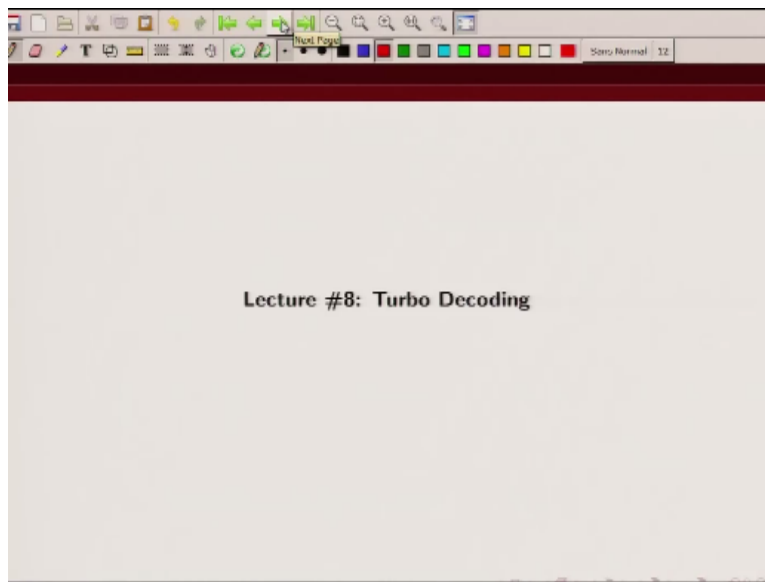
Indian Institute of Technology Kanpur
National Programme on Technology Enhanced Learning (NPTEL)
Course Title
Error Control Coding: An Introduction to Convolutional Codes

Lecture-8
Turbo Decoding

by
Prof. Adrish Banerjee
Dept. Electrical Engineering, IIT Kanpur

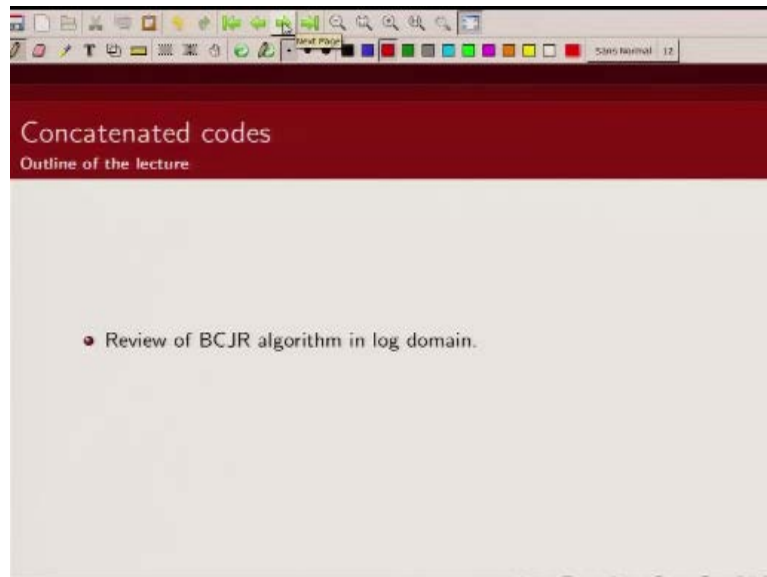
Welcome to the course on error control coding, an introduction to convolutional codes.

(Refer Slide Time: 00:21)



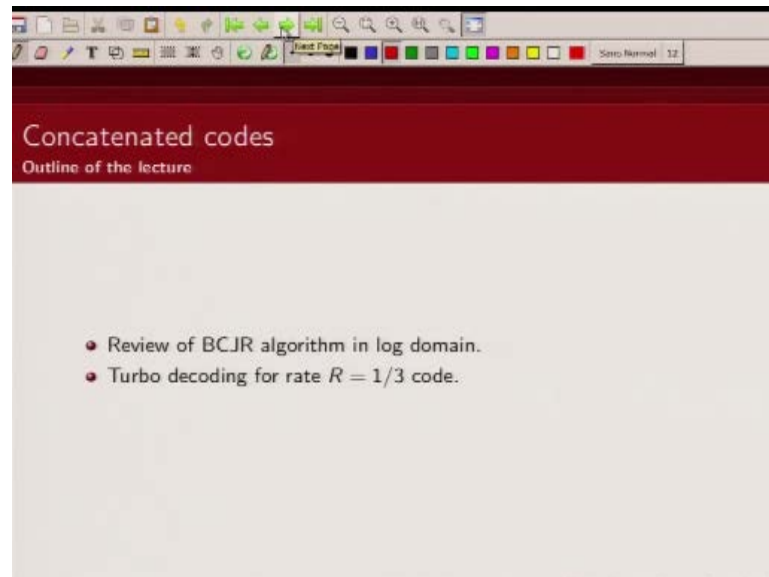
Today we are going to discuss about decoding of turbo codes.

(Refer Slide Time: 00:26)



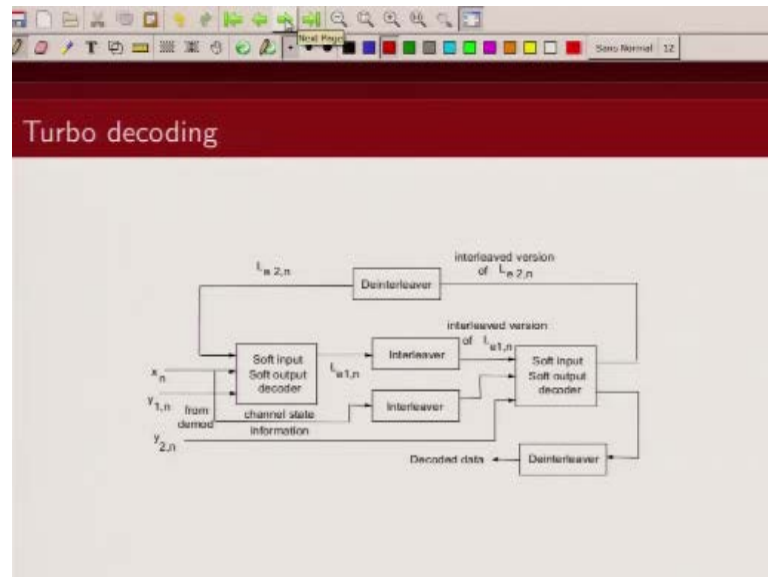
So to do that we will first review our BCJR algorithm in the log domain. We have talked about BCJR algorithm in the probability domain. We will very quickly review the metrics that are updated in BCJR algorithm and how they are implemented in the log domain. And then we will talk about turbo decoding.

(Refer Slide Time: 00:50)



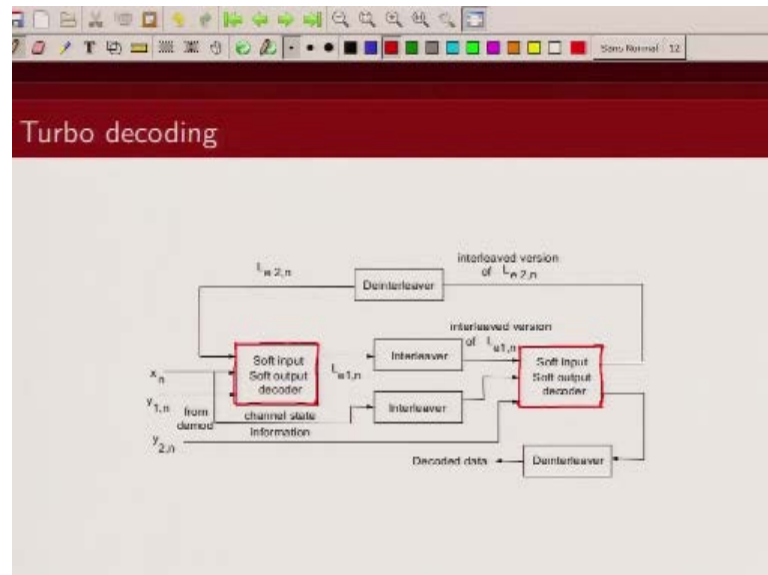
We will take an example of a rate $1/3$ code.

(Refer Slide Time: 00:55)



So this is the block diagram of a turbo decoder. As you can see the main blocks here are – so recall a turbo code consist of parallel concatenation of two recursive convolutional encoder.

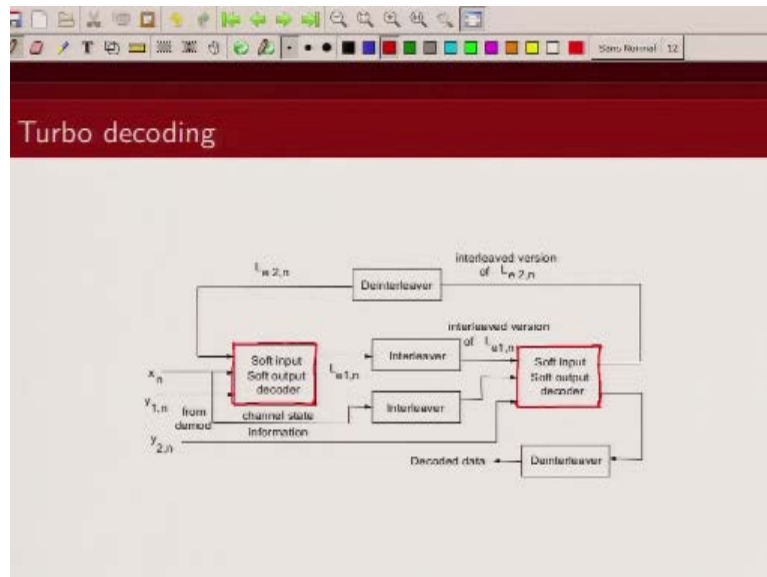
(Refer Slide Time: 01:14)



So here we have two decoders, it is an iterative decoding process, we have two decoders corresponding to the two encoders. So this is one decoder, this is second decoder. Now note I have written soft-input, soft-output decoder, now what do I mean by soft-input, soft-output decoder.

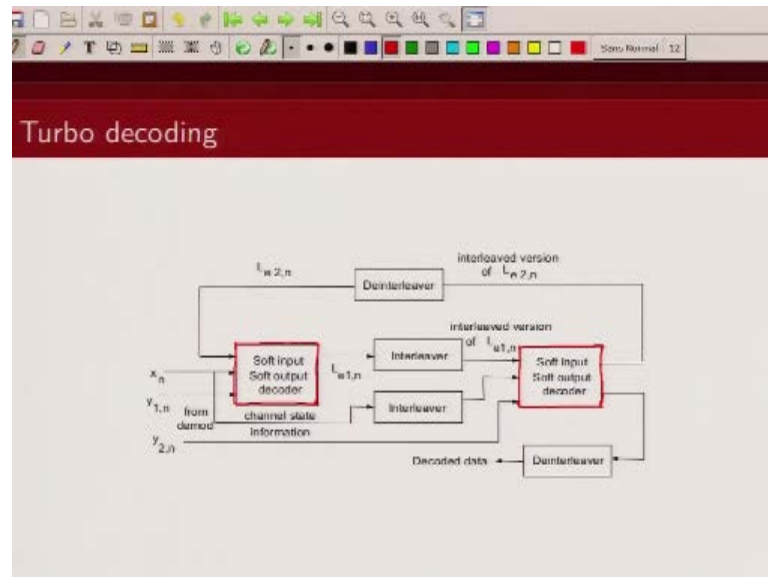
So the input that this decoder receives are the real channel received values. These are not quantized to zeros or ones. As opposed to getting zeros and ones in case of a hard quantized decoder.

(Refer Slide Time: 01:57)



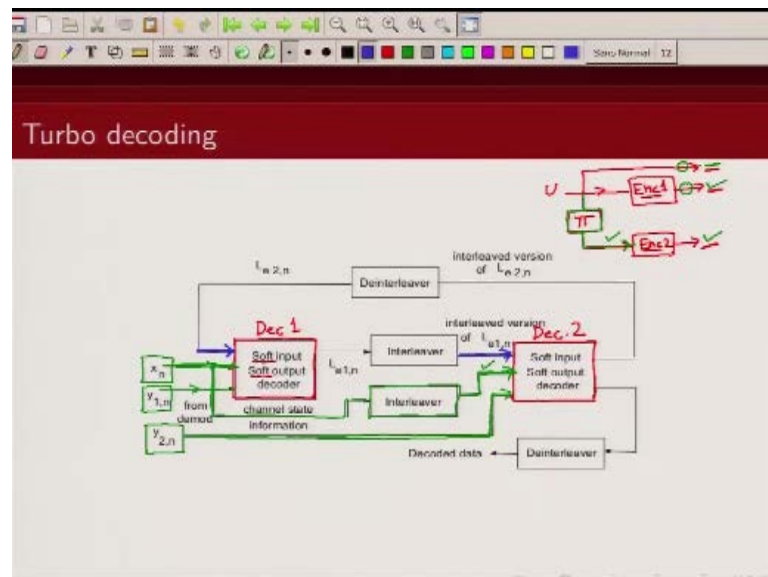
Here we are getting the received – noisy received sequence from the – directly from the channel and that what we will call – we are calling as soft input bits.

(Refer Slide Time: 02:07)



The output is also soft, what do we mean by soft-output? Now the decoder will output not only the decision about whether the bit it thinks is one or zero, but it will also give us some information about with what probability it thinks the bit is zero or one. So not only we are getting information about the decision of whether the bit is zero or one, but we are also getting some information about how likely the bit is going to be zero or one.

(Refer Slide Time: 02:42)



That is why the input here is also soft and the output is also soft. As opposed to a hard decoder where the output would have been just zeros and ones. So if you look at this decoder structure, now recall our encoder diagram for a turbo code. So what we had was, we had one encoder right, and this was an information sequence.

Now this information sequence was permuted using an interleaver, I am denoting the interleaver by π and then this interleaved signal was sent to another encoder, and the three outputs were first was this information bits, second was this parity bit coming out of the first encoder and the third is the parity bit coming out of the second encoder.

So these were the three outputs of a turbo decoder. Now after these bits passed through the channel what you are going to receive is noisy version of the information sequence, noisy version of this parity bit and noisy version of this second parity bit. Now as I said we are using two decoders, one decoder to decode this convolutional encoder, the second decoder to decode this convolutional encoder.

So this was my decoder one, if I want to call it and this was my decoder two. So what are the inputs of decoder one, now decoder one I should be sending in the received information bits and the received parity bits. So that is what I am sending here, you see here I have written, X_n is actually my received information bit and Y_{1n} is my received parity bit corresponding to the encoder one. So these are the two input to the decoder one.

Now what are the inputs to decoder two, now decoder two is an interleaved version of information that has been coming to encoder one. Now here I am receiving information bit directly, so the information received information bit that I will feed to the decoder two, because this information bit is getting interleaved before being sent to encoder one. What I am going to do is, so whatever – so this is my received information sequence.

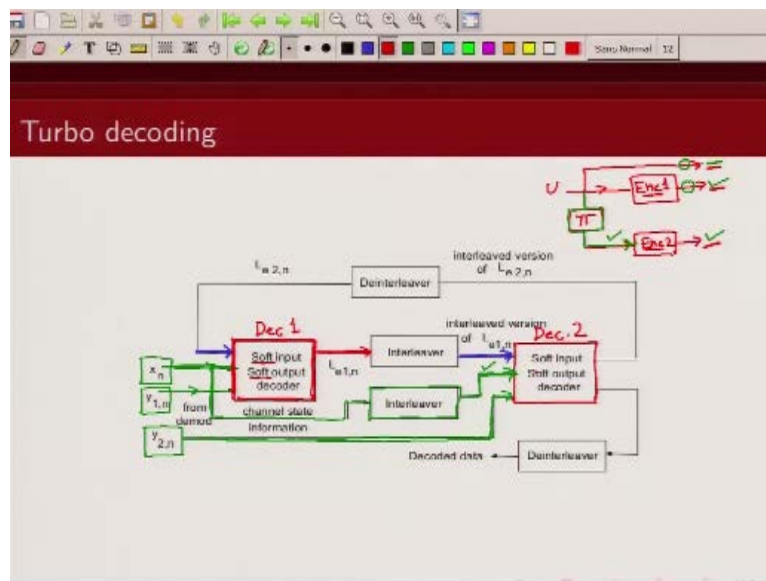
So what I do is to the second decoder before I feed this information sequence I am going to interleave this. So that the order of information bits that is coming here and what is being fed to decoder two is same. So we do not send the interleave information sequence in turbo code from this received sequence by interleaving, I can get back the information sequence that is being fed to this decoder.

So the information sequence input to this decoder is nothing but the interleaved version of the received information sequence. Now what is the second input with this decoder, this received parity bit? And that I am denoting by Y_{2n} , fine. So these are the two inputs which I am receiving from the channel, which I am feeding to decoder. So the input to decoder one are these two inputs and input to the decoder two is interleaved version of the information sequence and this particular parity bit.

So these are the two inputs to the decoders from the channel, what else is input to this decoder? There is a third input to this decoder which is this one and this one. And what is this input; this is a-priori knowledge about the information sequence. What is the a-priori knowledge about this information sequence that is being fed here as third input? So these third input that you see to decoder one is the a-priori knowledge about the information sequence.

Now how do we get this a-priori knowledge, now initially when we start decoding we do not have any a-priori knowledge about the information bit. So we would assume that the information bit is equally likely to be zero or one. So the likelihood of the information being zero or one that is one, basically is equally likely whether the bit is zero or one.

(Refer Slide Time: 08:08)



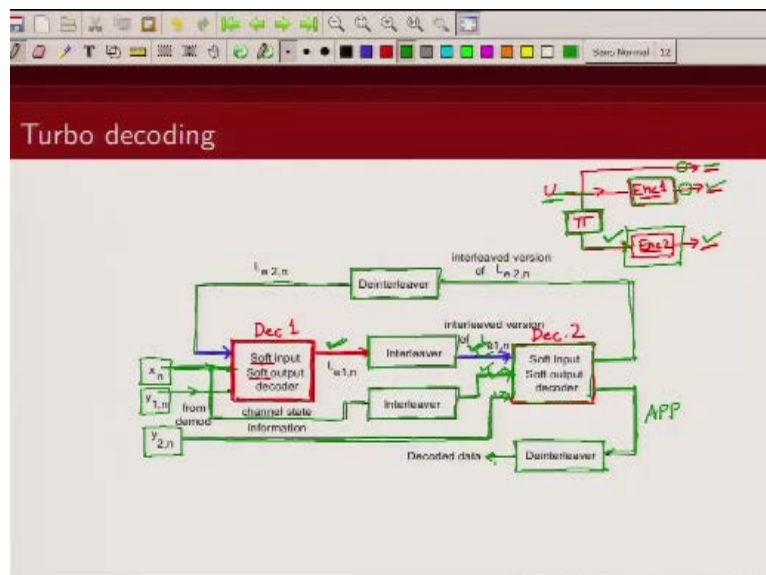
So that would be our initial a-priori information. Now once we feed these three inputs to this decoder one, now let us pay attention to the outputs of decoder one. So decoder one is giving us two information, one what we call extrinsic information. Now this is information which this decoder computes based on the trellis structure of this convolutional code and this extrinsic information is being passed to the decoder one and this input is fed as a-priori knowledge to the second decoder.

So you can think of it like this, so there are two decoders who are working independently, but then one decoder once it decodes information sequence it passes some information to the other decoder saying, hey I think the information bit is likely to be zero with this probability and information bit is one with this probability. So the other decoder will take that as an input and

then recomputes the probability and then it will again compute some new probabilities of bit being zero and one.

And it will pass that information back to the first decoder saying no, I think it is likely to be zero with this probability and likely to be one with this probability. And this information exchange, you know happens in a regular fashion in this, until they converge to this particular decision we may stop iteration after fixed number of iteration or we could use some sort of a stopping criteria to do that.

(Refer Slide Time: 10:01)



So the third input is the a-priori input, the other two inputs are the input received from the channel. Now what this decoder one does it, it tries to find some estimate about the information sequence pass it on to the second decoder. Now this input is fed as a-priori input. Now what is the use of this interleaver, now note that the order of information bit here and the order of information bit here, now the information bit that is being fed to second encoder which is this is interleaved version of the information bit that goes to encoder one.

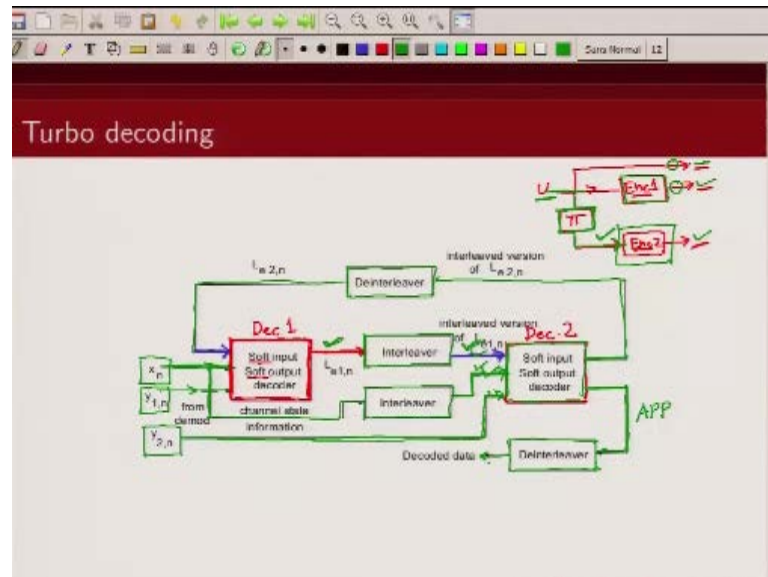
So the estimates that encoder one gives about the information bits that is being interleaved and sent to decoder one, this is to maintain the same order of information sequence as being received by decoder two. So, because the information sequence here is interleaved version of information sequence here, so we are going to use this interleaver. Now similarly this decoder two, what does this decoder two does, it takes these two channel values and it takes this a-priori information which it has received from decoder one.

And then it will try to decode this code and it will form some estimate, some extrinsic information and that information is fed back to first decoder and note there is a deinterleaver in between, because the order of information sequence here is deinterleaved version of information that is being fed to encoder two. So this deinterleaver is done, so that the order of information estimates that we are feeding to encoder one is in same order as these other inputs are.

So this is an iterative process which goes on and after some fixed number of iteration or you can do some stopping criteria you can use some stopping criteria after that finally you take some decision so I can take decision let us say from the second decoder. So this is my a-posteriori probability. Now this information is driven by the information sequence ordering at encoder two is interleaved version of the information sequence of encoder one.

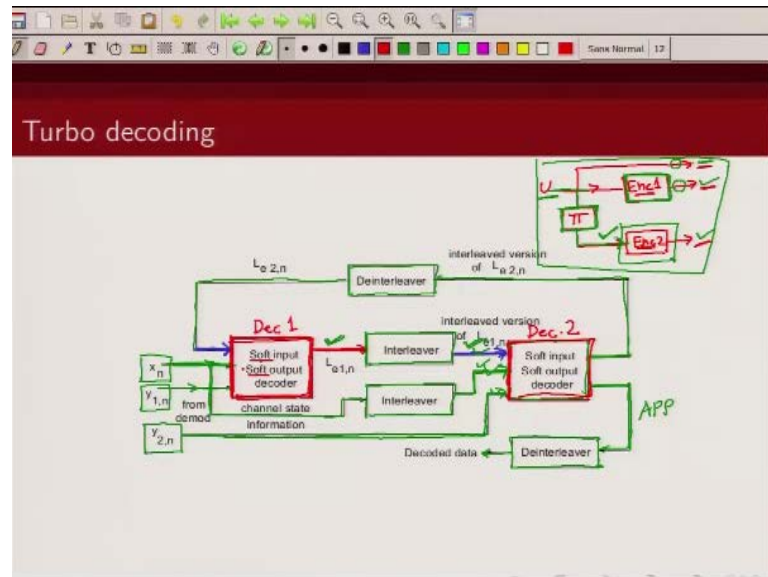
So if you want to know the order of information sequence here we need to deinterleaved this data and here, then we will take a hard decision and take a decision whether the bits are zero or one.

(Refer Slide Time: 13:00)



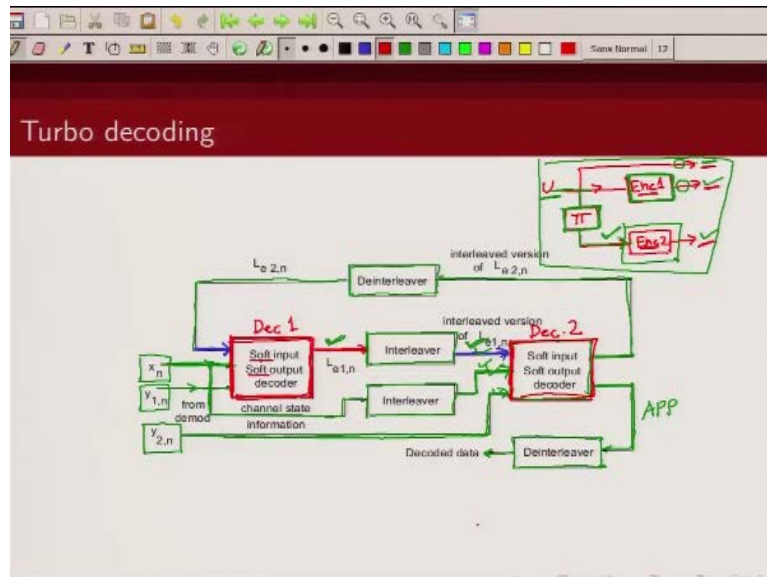
So you can see this is an iterative decoding algorithm where instead of decoding this whole code at one go. What we are trying to do is we are trying to decode first encoder one and then encoder one is passing some information to encoder 2, encoder 2 receives some information from the channel and it receives some information from encoder 1, using this it tries to form some opinion about information bits which it passes it on to decoder1 and then decoder1 uses that information so one iteration is

(Refer Slide Time: 13:34)



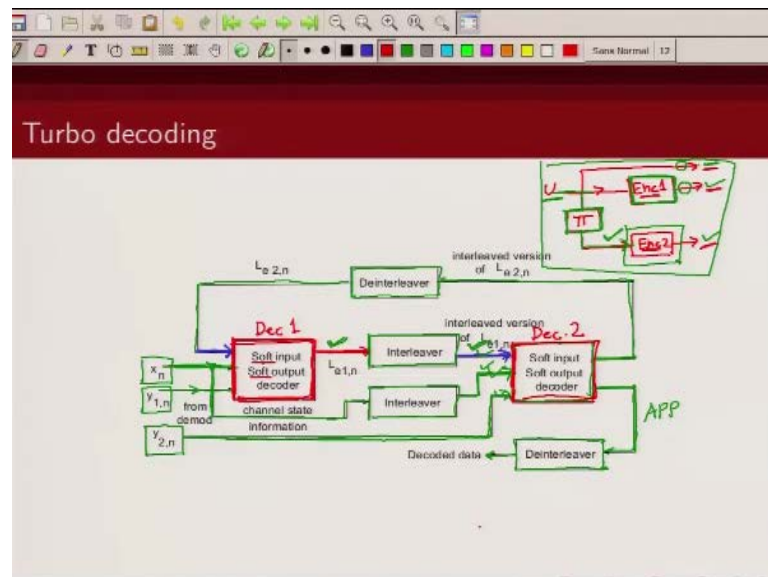
When this decoder1 has finished decoding and decoder2 has also finished decoding, so that is my one full iteration. Now what is algorithm which is used inside of this I said soft input and soft output decoder I said input is soft output is soft but what sort of algorithm we can use? Now we can use any algorithm which would, which can take soft inputs and which can give soft outputs and one of the most commonly used algorithm is our BCJR algorithm. Now in lecture 5 we have talked about decoding of BCJR algorithm for convolutional code so this is

(Refer Slide Time: 14:23)



Precisely what these 2 decoders that you see here they are going to use, they are going to use BCJR algorithm and

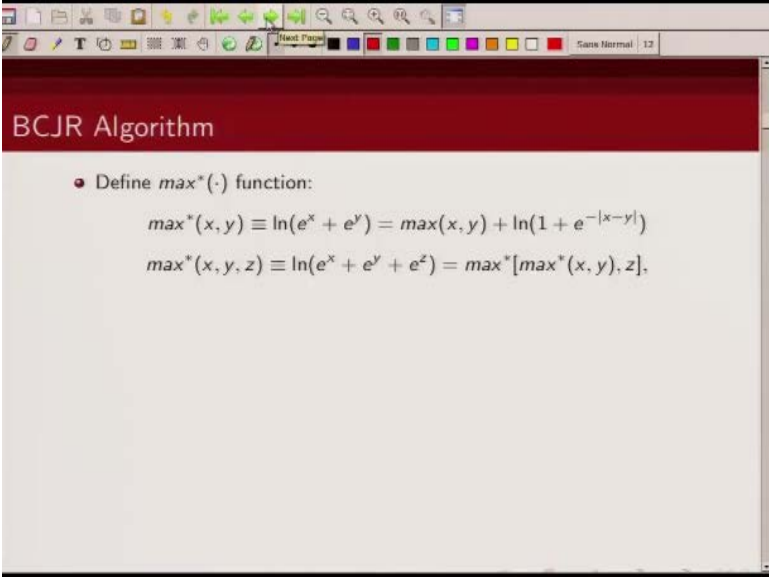
(Refer Slide Time: 14:32)



We are going to slightly modify this algorithm to get these extrinsic information's from the decoder and that we will show in subsequent slide. So this is the basic block diagram of your turbo decoder you have to remember this and again I repeat each of this decoders are independently working in the sense there is channel input which has been fed back and there are some a-priori information which is coming from the other decoder which has been fed to these decoder and these decoder take these 3 inputs.

And they compute 2 values what is this a-posteriori probability another is extrinsic information ,extrinsic information is passed on to the other decoder as a-priori value and the a-posteriori probability is used when we want to take the final decision about the bits.

(Refer Slide Time: 15:33)



BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

So let us just review BCJR algorithm very quickly, now we have already derived the expressions for the channel matrix, the matrix that we need to compute in BCJR algorithm so we are just going to directly write the expression for BCJR algorithm now I just

(Refer Slide Time: 15:54)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

Handwritten red notes above the equations:

$$\ln(e^x(1 + e^{y-x})) = x + \ln(1 + e^{y-x})$$

$$\ln(e^y(1 + e^{x-y})) = y + \ln(1 + e^{x-y})$$

Want to introduce an operator which I called max* operator now what is a max* operator, max* operator of X and Y is basically defined as log of $E^x + E^y$. Now this log of $E^x + E^y$ this can be written as log of this $E^x \times 1 + E^y$ so $Y - X$. We can write it this way correct, now this can be written as log of $e^x + \log$ of $1 + E$ to power $Y - X$, or I can also write the same thing as log of E^y $1 + E^{x-y}$ log of E^{y+} log of $1 + E^{x-y}$ and what is this log of E^x ? That is just X so this can be written as, so this can be just written as X + this and Y + this or I can write this max* XY as maximum of X and Y + the actual log of $1 + e^{-\text{absolute value of } X - Y}$, right?

So, so whenever I have to take log of terms of this form I can actually implement it simply like maximum of this 2 operator X and Y plus some correction.

(Refer Slide Time: 18:05)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

$$\max^*(x, y, z) = \ln(e^x + e^y + e^z) - \max^*(\max^*(x, y), z)$$

Handwritten red notes on the slide show the derivation of the \max^* function:

$$\ln(e^x(1 + e^{y-x})) = x + \ln(1 + e^{y-x})$$

$$\ln(e^y(1 + e^{x-y})) = y + \ln(1 + e^{x-y})$$

Term which can be implemented with a table look of kind of thing and this operate, operator can be extended for more than two operations, so if you want to

(Refer Slide Time: 18:18)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

$$\ln(e^x(1 + e^{y-x})) = x + \ln(1 + e^{y-x})$$

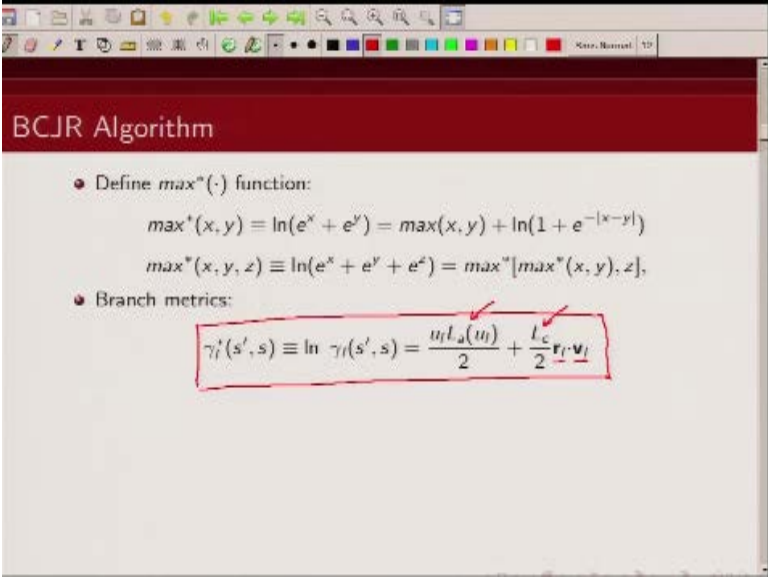
$$\ln(e^y(1 + e^{x-y})) = y + \ln(1 + e^{x-y})$$

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

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

Find \max^* of X , Y and Z then I can write it as the natural log of $E^x + E^y + Z$ and I can do \max^* of XYZ and then max then I take the \max^* of \max^* of XY and Z, so I can iteratively apply this \max^* operator to compute quantities of this form okay, which is log of summation terms. Now where do we encounter log of summation terms, I will come to that.

(Refer Slide Time: 18:57)

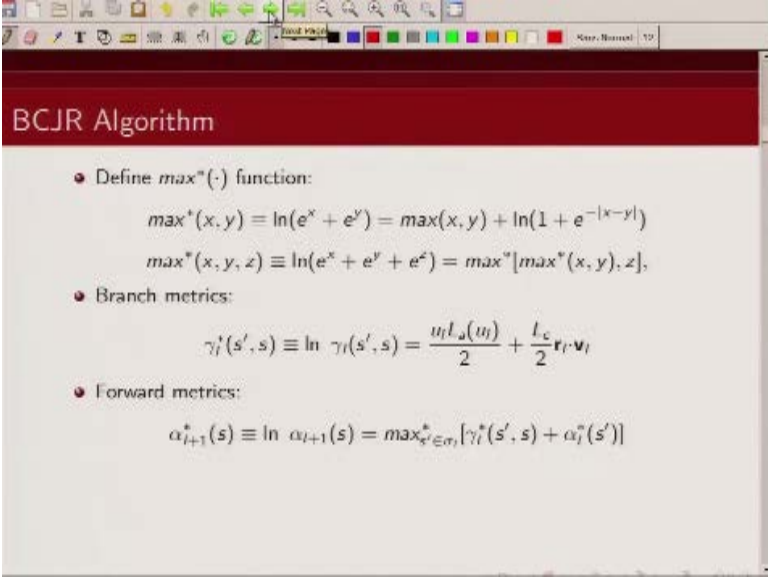


BCJR Algorithm

- Define $\max^*(\cdot)$ function:
$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$$
$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z],$$
- Branch metrics:
$$\gamma_l^*(s', s) \equiv \ln \gamma_l(s', s) = \frac{u_l l_a(u_l)}{2} + \frac{l_c}{2} \underline{r}_l \cdot \underline{v}_l$$

Now this branch metric, this we have derived in one of the lectures is given by this expression this is a-priori, information this information bit, this is depends on channel SNR, this is receive sequence, transmitted code receive sequence, transmitted code word so this you already are familiar with because we have derived this expression before.

(Refer Slide Time: 19:22)



BCJR Algorithm

- Define $\max^*(\cdot)$ function:
$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$$
$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*(\max^*(x, y), z),$$
- Branch metrics:
$$\gamma_l^*(s', s) \equiv \ln \gamma_l(s', s) = \frac{u_l l_a(u_l)}{2} + \frac{l_c}{2} \mathbf{r}_l \cdot \mathbf{v}_l$$
- Forward metrics:
$$\alpha_{l+1}^*(s) \equiv \ln \alpha_{l+1}(s) = \max_{s' \in \sigma_l} [\gamma_l^*(s', s) + \alpha_l^*(s')]$$

Now remember, recall there were 3 quantities that we need to evaluate when we want to apply BCJR algorithm and what are those 3 quantities? One was this alpha's which was the forward recursion, second was this betas which was the backward recursion, and then we had this channel matrix, branch matrix right and gammas. Now if you

(Refer Slide Time: 19:51)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

$$\max^*(x, y, z) = \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z],$$
- Branch metrics:

$$\gamma_l^*(s', s) \equiv \ln \gamma_l(s', s) = \frac{u_l l_a(u_l)}{2} + \frac{l_c}{2} \mathbf{r}_l \cdot \mathbf{v}_l$$
- Forward metrics:

$$\alpha_{l+1}^*(s) \equiv \ln \alpha_{l+1}(s) = \max_{s' \in \sigma_l} [\gamma_l^*(s', s) + \alpha_l^*(s')]$$

Handwritten notes on the slide include:

$$\ln(a_1 b_1 + a_2 b_2) = \alpha_{L-1}(a) \gamma_L(a, a) + \alpha_{L-1}(b) \gamma_L(b, a)$$

$$\ln(e^{a_1} e^{b_1} + e^{a_2} e^{b_2}) = \alpha_{L-1}(a) \gamma_L(a, a) + \alpha_{L-1}(b) \gamma_L(b, a)$$

A small trellis diagram is shown on the right side of the slide, illustrating the state transitions and branch metrics.

Recall what was alpha's, alpha bit so if you just draw simple trellis diagram, let us just draw simple trellis diagram, two state trellis and this is alpha at time, let us say $L - 1$ this is L 5 time L , let us just call it a state zero, let us call it state one then what is, let us say I want to compute alpha at time L for state zero. What is it equal to, recall this was equal to so there are 2 states, there are 2 branches which are terminating here one is this one, other is this one so the alpha zero will be alpha $L - 1$ zero times branch metric corresponding to this.

Which we gamma zero, zero plus alpha $L - 1$, 1 times this branch metric which is gamma one zero. So if you recall we had terms of the form summation alpha $L - 1$ times some gamma, so those were our terms. Now if we take log of that so these were our alphas, we are defining a new operator alpha * so that is a log of these alphas, so what is going to happen here? So you have log of summation terms right, so if you can think of it as so here you have terms of the form like this.

$A_1, B_1 + A_2, B_2$ and we are taking log of this, now we can also write this in terms of let us say $E^{a_1} E^{b_1} + E^{a_2} E^{b_2}$ right? So if we take log of this then this will become because this is like E of $E^x + 1$ so this when we take log of these summation terms we get this \max^* operator so we

will get this \max^* operator and this product term and we take log they will become summation. So this forward metric will become when we implement it in we take log of this, this will become a \max^* operator and this 2 terms inside which is this gamma term.

And this alpha terms, this will be plus, so the forward metric here in log domain will be given by this metric, \max^* operator the summing over all the branches that are terminating at this state and this will be gamma plus alpha *, so this will be our forward metric in the log domain, now recall how did we initialize the alphas?

(Refer Slide Time: 23:22)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

$$\max^*(x, y, z) = \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z]$$
- Branch metrics:

$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i l_a(u_i)}{2} + \frac{l_c}{2} r_i v_i$$
- Forward metrics:

$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$

Handwritten red notes and diagram:

Diagram: A state transition diagram showing two states, s_{i-1} and s_i , with transitions labeled a_1, b_1 and a_2, b_2 . The diagram is annotated with α_{i-1} and α_i .

Handwritten red equations:

$$\ln(a_1 b_1 + a_2 b_2) = \alpha_{i-1}(a_1) \gamma_i(a_1, a_2) + \alpha_{i-1}(a_2) \gamma_i(a_2, a_1)$$

$$\ln(e^{a_1} e^{b_1} + e^{a_2} e^{b_2}) = \alpha_{i-1}(a_1) \gamma_i(a_1, a_2) + \alpha_{i-1}(a_2) \gamma_i(a_2, a_1)$$

When we were working in probability domain we said that the encoder starts from all zero state so at a time zero it will be at all zero states, the probability of its being in all zero state at time zero is one for all other it is zero, so when we take log of that then

(Refer Slide Time: 23:47)

BCJR Algorithm

- Define $\max^*(\cdot)$ function:

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

$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*(\max^*(x, y), z),$$
- Branch metrics:

$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i l_a(u_i)}{2} + \frac{l_c}{2} r_i \cdot v_i$$
- Forward metrics:

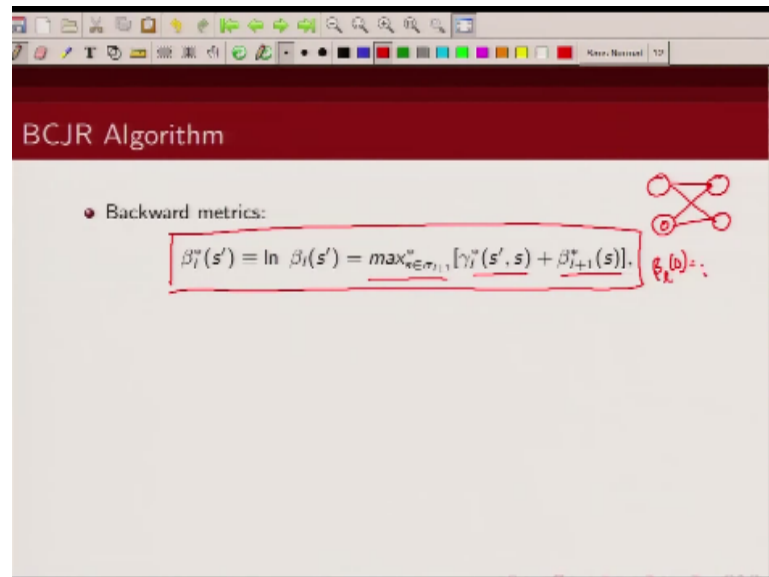
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \mathcal{S}_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:

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

The initialization will become for state initial state zero then this will become log of one will become zero and for all probability of its being all other state this will then become minus infinity okay. So if we are writing our recursion in this fashion using \max^* operation then the initialization should be done like this, when it is in state zero the initialization $\alpha_0^* (0)$ will be zero and it will be minus infinity for all other cases. Now if you are wondering why am I switching from probability.

Domain to this log domain you can see that we have shown this \max^* operator can be very easily implemented because this is just maximum of XY + some correction term, so this can be easily implemented and that is why we are rewriting our forward recursion, backward recursion, gamma in terms of log domain so that our, in our expression where we had the summation now we, that has been replaced by \max^* operator, wherever we had multiplication that has been now replace by addition. So following the same logic

(Refer Slide Time: 25:23)



BCJR Algorithm

• Backward metrics:

$$\beta_l^*(s') = \ln \beta_l(s') = \max_{s \in \mathcal{S}_{j+1}} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)].$$

$\xi_L(b)$

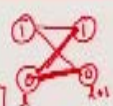
We can write the backward metric in log domain in this particular fashion, so this max* operator and this is sum of this branch metric in the log domain and the betas in the log domain, and again if I, if you can recall how were we computing betas, so let us say you had some this thing, so if you are interested in computing beta for beta at L zero you will be

(Refer Slide Time: 26:02)

BCJR Algorithm

• Backward metrics:

$$\beta_l^*(s') \equiv \ln \beta_l(s') = \max_{s \in \mathcal{S}_{l+1}} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)]$$



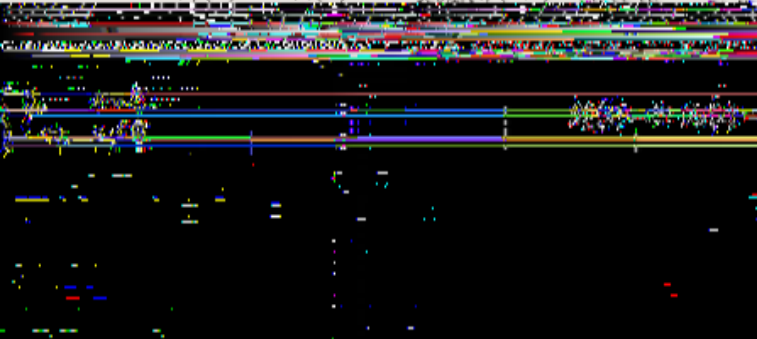
$\beta_L(0) = \beta_{L+1}(0)$
 $\gamma_L(0,0)$
 $+ \beta_{L+1}(1)$
 $\gamma_L(0,1)$

You will be β and this is let us say this is time l , time $l+1$, β time l this is β at time $l+1$, β at zero will be, so these are the two branches that are terminating here so β_{l+1} zero times branch metric of this which is $\gamma_L(0,0) + \beta_{L+1}$ this is state zero this is state one, state one $\beta_l(1)$ times $\gamma_l(0,1)$ this we have already studied when we did BCJR algorithm we are just re-writing the expression in terms of log so that our addition term here becomes max star operator and the product term here that you see becomes addition term.

(Refer Slide Time: 27:02)

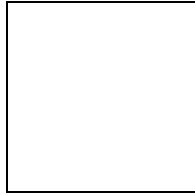
BCJR Algorithm

• Backward metrics:

$$\beta_l^*(s') \equiv \ln \beta_l(s') = \max_{s \in \mathcal{Q}_M} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)].$$


And similar to alpha bit initialization

(Refer Slide Time: 27:08)



If we are terminating our convolutional encoder then β_k at $s=0$ will be 0 and for all other state it will be minus infinity.

(Refer Slide Time: 27:23)

BCJR Algorithm

- Backward metrics:

$$\beta_l^*(s') \equiv \ln \beta_l(s') = \max_{s \in \mathcal{S}_{l+1}} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)],$$

- Backward metrics initialization:

$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0}. \end{cases}$$

- APP L-value:

$$L(u_l) = \max_{(s', s) \in \Sigma_l^+} [\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')] \\ - \max_{(s', s) \in \Sigma_l^-} [\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')].$$

And of course if we are not terminating it then its probability of being any state is basically same, and if you recall our APP log likelihood value this was again Σ of product of three terms α at time l , β at time $l+1$ and γ .

(Refer Slide Time: 27:48)

BCJR Algorithm

- Backward metrics:

$$\beta_l^*(s') \equiv \ln \beta_l(s') = \max_{s \in \mathcal{S}_{l+1}} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)],$$
- Backward metrics initialization:

$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0}. \end{cases}$$
- APP L-value:

$$L(u_l) = \max_{\{s', s\} \in \mathcal{S}_l} [\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')] - \max_{\{s', s\} \in \mathcal{S}_l} [\beta_{l+1}^*(s) + \gamma_l^*(s', s) + \alpha_l^*(s')].$$

So that and there were two terms one term in the numerator corresponding to all those transitions belonging to information bit being +1 and in the denominator we had some terms related to transitions that belong to information bit being -1.

(Refer Slide Time: 28:09)

BCJR Algorithm

- Backward metrics:

$$\beta_l^*(s') \equiv \ln \beta_l(s') = \max_{s \in \mathcal{O}_{l+1}} [\gamma_l^*(s', s) + \beta_{l+1}^*(s)],$$
- Backward metrics initialization:

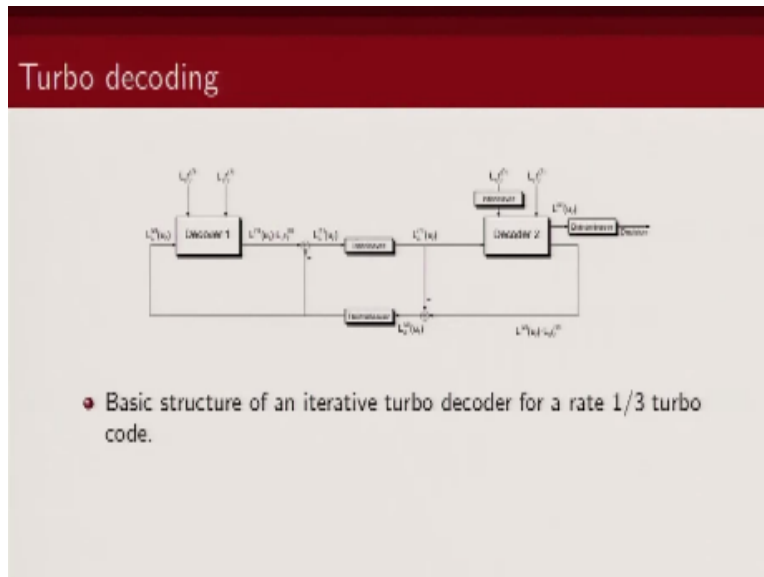
$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0}. \end{cases}$$
- APP L-value:

$$L(u_l) = \frac{\max_{(s', s) \in \Sigma_+} [\beta_{l+1}^*(s) \boxplus \gamma_l^*(s', s) \boxplus \alpha_l^*(s')]}{\max_{(s', s) \in \Sigma_-} [\beta_{l+1}^*(s) \boxplus \gamma_l^*(s', s) \boxplus \alpha_l^*(s')]}.$$

So this term that you see here is the term corresponding to the numerator term, again the Σ term has been replaced by this max star operator and this product term has been replaced by these addition terms okay, and similarly this corresponds to all those transitions where my information bit is +1 and the denominator we had this so we take log of them so become minus of this and the denominator corresponds to all those transitions which belong to information bit being -1.

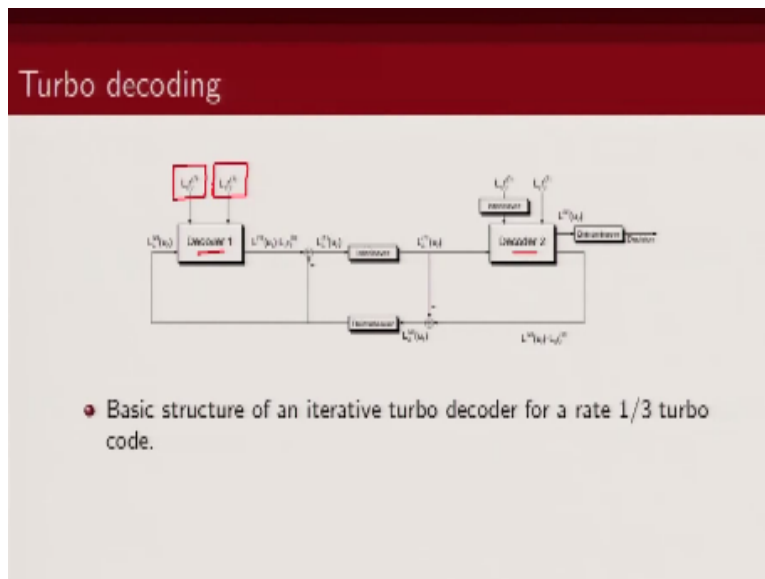
And again the addition term that we had in the probability domain description of BCJR that is now max star operator and the product terms that we had in the BCJR algorithm are now addition term. So this is re-writing the expressions for forward recursion, backward recursion, and log likelihood ratio a-posteriori probability l value computation for the BCJR algorithm.

(Refer Slide Time: 29:29)



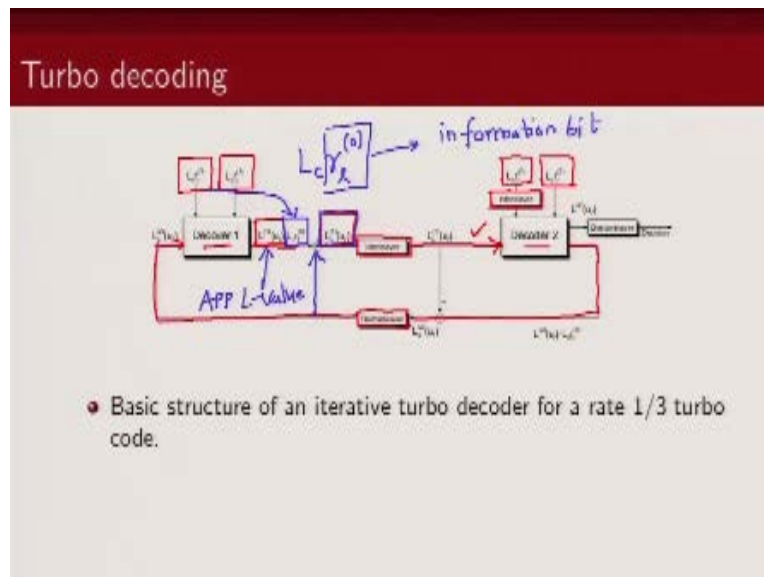
So again we will go back to this decoder diagram that we have shown you before, I am reproducing the decoder for rate 1/3rd turbo code.

(Refer Slide Time: 29:49)



This is my decoder 1 corresponding to encoder 1, this is my decoder 2, these two inputs that you see are my inputs corresponding to the received information bit and the corresponding parity bit.

(Refer Slide Time: 30:08)



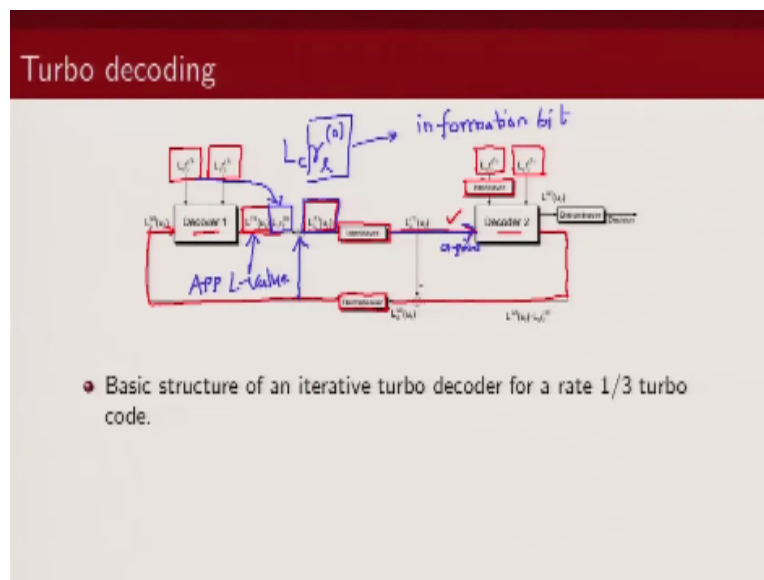
This is my information received information bit which is interleaved before being sent to decoder 2 and this is the received, this is an bit corresponding to the received parity, second parity bit. I also said there is an third input which is the a-priori value, note this a-priori value is coming from the other decoder and since the order of information bit here is deinterleaved version of the order or information bit at second decoder so you de-interleave it and send this information here and the a-priori value that you send here is coming from this decoder.

And I am interleaving this because the order of information bit at decoder 2 is interleaved version of the order of information bit at decoder 1. There is one more thing I am computing here and I have talked about extrinsic value, right? So what is my extrinsic value and this is what I am computing here so let us pay some attention to this, this is the extrinsic value so what is this extrinsic value? So note I am getting this APP L- value computed that is this, from there I am subtracting the contribution of the information bit; this term is same as this bit.

This is, this L_c term depends on received SNR I will come to that, this term is this and this is the received value corresponding to the information bit and this is the term corresponding to APP L- value, so what I am doing is I am subtracting from the APP L- value the contribution of this

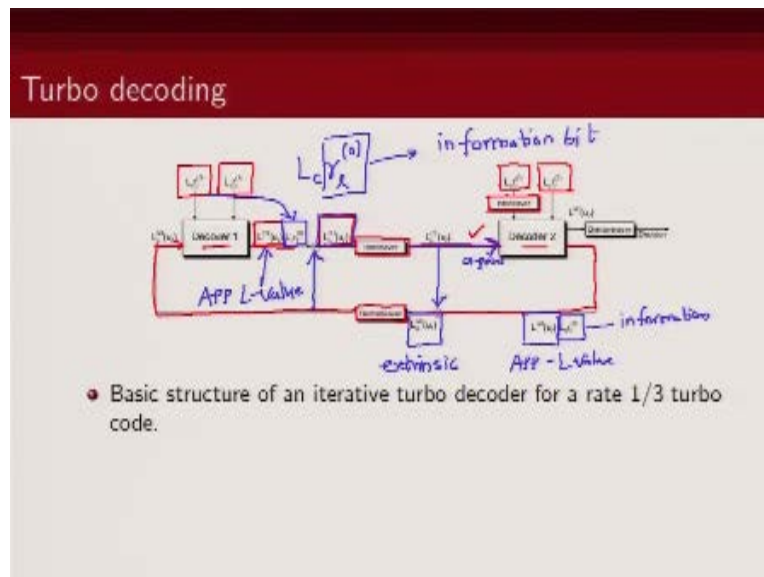
received bit. I am also subtracting from this contribution this a-priori information so what I get is my extrinsic information. So you can of it as some additional information about the information bits that has been derived from the structure of the convolutional encoder and how am I computing this extrinsic information from the APP L-value? I am subtracting the contribution of the received channel bit.

(Refer Slide Time: 32:57)



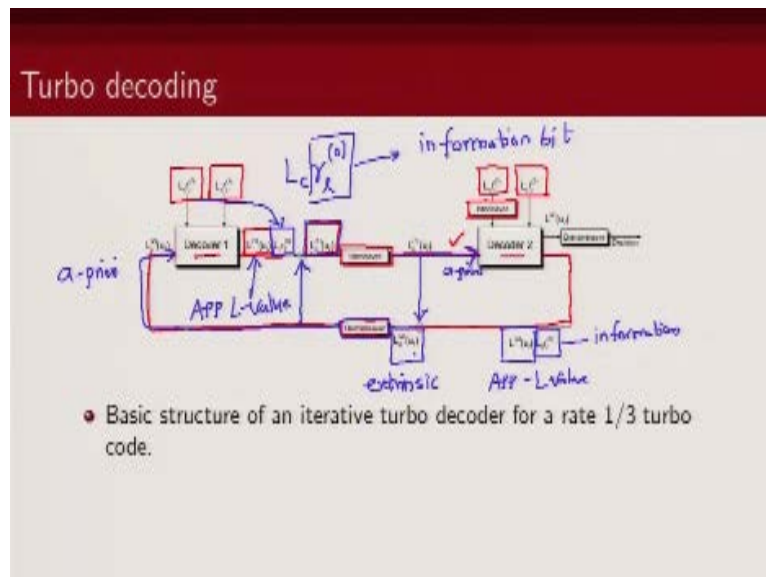
I am subtracting the contribution of the a-priori value, what is left is extrinsic information and this information as I said is being passed, is interleaved because the order of the information bit in the second decoder is interleaved version of the order of bits in decoder 1 so I interleave it and feed this as a-priori information so this is my a-priori information. So this is how I compute the extrinsic information for decoder 1, now how do I compute extrinsic information for decoder 2, the same procedure.

(Refer Slide Time: 33:41)



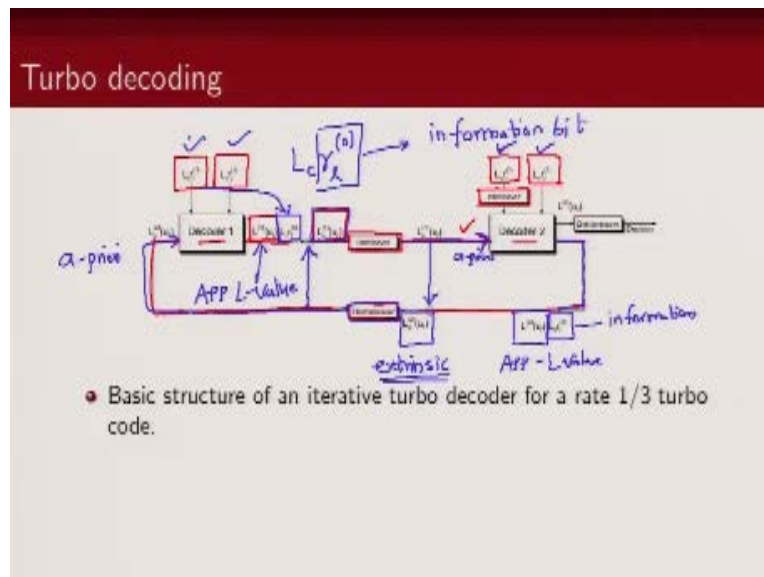
This is my APP L-value, correspond to the information bit. I subtract from there the contribution of information bit, this is my L_{C1} zero term and then I am subtracting this contribution of the a-priori information so what is left is this information which is my extrinsic information, so for the second decoder in the similar fashion I compute the extrinsic information. I subtract from the APP value the contribution of the information bit and contribution of the a-priori knowledge, what is left is my extrinsic information.

(Refer Slide Time: 34:33)



Now I need to de-interleave this information before feeding it back as a-priori information so this is my a-priori value for decoder 1, and I need to de-interleave that because the order of information bit for decoder 1 is de-interleave version of the order of information for decoder 2, so this is how my iterative decoding algorithm is working, again I will do a quick recap so using the BCJR algorithm.

(Refer Slide Time: 35:14)

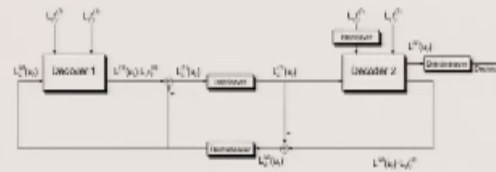


So I have some received value from the channel which is this and this, initially I do not have any a-priori knowledge about the information bit so I assume they are equally likely to be 1(0) this decoder 1 will apply BCJR algorithm and it will compute the APP L-values and it will compute the extrinsic L-values. Now this extrinsic information is passed as a-priori information to decoder 2, in addition decoder 2 has this received parity bit which is this one and interleaved version of the information sequence as input, so it will again compute APP L – value.

And it will subtract the contribution of the information bit and the a-priori value and then we will get back extrinsic information, so you can see this extrinsic information is getting passed from one decoder to another okay, and ideally what we would like is this extrinsic information should grow in a manner so that it pushed the decision in favor of either information bit being +1 or -1, that is when we say the decoder is converging with iteration.

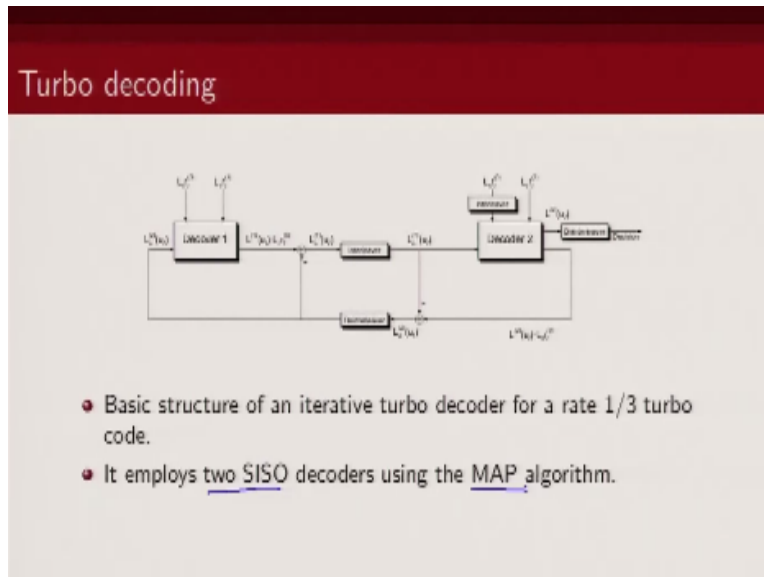
(Refer Slide Time: 36:32)

Turbo decoding



- Basic structure of an iterative turbo decoder for a rate 1/3 turbo code.
- It employs two SISO decoders using the MAP algorithm.

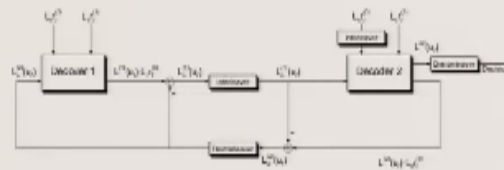
(Refer Slide Time: 36:35)



So we have already explained that there are two decoders and each of them are using map algorithm, this BCJR algorithm that we talked about.

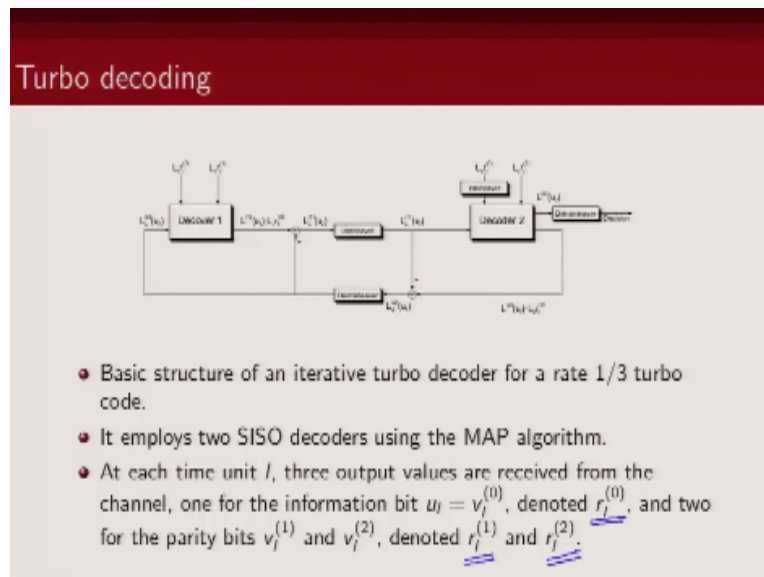
(Refer Slide Time: 36:41)

Turbo decoding



- Basic structure of an iterative turbo decoder for a rate 1/3 turbo code.
- It employs two SISO decoders using the MAP algorithm.
- At each time unit l , three output values are received from the channel, one for the information bit $u_l = v_l^{(0)}$, denoted $r_l^{(0)}$, and two for the parity bits $v_l^{(1)}$ and $v_l^{(2)}$, denoted $r_l^{(1)}$ and $r_l^{(2)}$.

(Refer Slide Time: 36:46)



There are three inputs received from the channel 1 corresponds this $r_L^{(0)}$, corresponds to the information bit $r_L^{(1)}$, corresponds to the parity bit corresponding to encoder 1, and $r_L^{(2)}$ the parity bit corresponding to encoder 2.

(Refer Slide Time: 37:08)

Turbo decoding

- The $3K$ -dimensional received vector is denoted by

$$\mathbf{r} = \left(r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \dots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)} \right).$$

- Assume 0 is mapped to -1 and 1 to $+1$.
- Then for an AWGN channel, we define the log-likelihood ratio (L-value) $L(u_i | r_i^{(0)})$ (before decoding) of a transmitted information bit u_i given the received value $r_i^{(0)}$ as

$$\begin{aligned} L(u_i | r_i^{(0)}) &= \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})} \\ &= \ln \frac{P(r_i^{(0)} | u_i = +1) P(u_i = +1)}{P(r_i^{(0)} | u_i = -1) P(u_i = -1)} \end{aligned}$$

(Refer Slide Time: 37:14)

Turbo decoding

- The $3K$ -dimensional received vector is denoted by

$$\mathbf{r} = \left(\underbrace{r_0^{(0)}, r_0^{(1)}, r_0^{(2)}}_{\text{first parity bit}}, \underbrace{r_1^{(0)}, r_1^{(1)}, r_1^{(2)}}_{\text{information bit}}, \dots, \underbrace{r_{K-1}^{(0)}, r_{K-1}^{(1)}, r_{K-1}^{(2)}}_{\text{second parity bit}} \right).$$
- Assume 0 is mapped to -1 and 1 to +1.
- Then for an AWGN channel, we define the log-likelihood ratio (L-value) $L(u_i | r_i^{(0)})$ (before decoding) of a transmitted information bit u_i given the received value $r_i^{(0)}$ as

$$\begin{aligned} \underline{L(u_i | r_i^{(0)})} &= \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})} \\ &= \ln \frac{P(r_i^{(0)} | u_i = +1) P(u_i = +1)}{P(r_i^{(0)} | u_i = -1) P(u_i = -1)} \end{aligned}$$

So at each time instance, so this is my time index time 0, time 1, time k-1, so each time instance I am getting this three bit information which is the information bit par, first parity bit and the second parity bit, remember this is a rate $1/3^{\text{rd}}$ code so if you have a information block of k you will get $3k$ coded bits, zeros map to in this case -1+1 is 1, now let us compute the log likelihood ratio so probability of u_L given r_L is zero is given by what is the probability that u_L is +1 given this received sequence r_L divided by probability of u_L being -1 given receive sequence r_L this we can write as probability of r given u multiplied by probability of u and so that is how we have written it.

(Refer Slide Time: 38:20)

Turbo decoding

$$\begin{aligned}
 L(u_l | r_l^{(0)}) &= \ln \frac{P(r_l^{(0)} | u_l = +1)}{P(r_l^{(0)} | u_l = -1)} + \ln \frac{P(u_l = +1)}{P(u_l = -1)} \\
 &= \ln \frac{e^{-\frac{E_s}{N_0}(r_l^{(0)} - 1)^2}}{e^{-\frac{E_s}{N_0}(r_l^{(0)} + 1)^2}} + \ln \frac{P(u_l = +1)}{P(u_l = -1)}, \\
 &= -\frac{E_s}{N_0} \left\{ (r_l^{(0)} - 1)^2 - (r_l^{(0)} + 1)^2 \right\} + \ln \frac{P(u_l = +1)}{P(u_l = -1)} \\
 &= 4 \frac{E_s}{N_0} r_l^{(0)} + \ln \frac{P(u_l = +1)}{P(u_l = -1)} \\
 &= L_c r_l^{(0)} + L_a(u_l),
 \end{aligned}$$

where E_s/N_0 is the channel SNR, u_l and $r_l^{(0)}$ have both been normalized by a factor of $\sqrt{E_s}$, $L_c = 4(E_s/N_0)$ is the channel reliability factor and

Now we can separate out this into two terms

(Refer Slide Time: 38:22)

Turbo decoding

- The $3K$ -dimensional received vector is denoted by

$$\mathbf{r} = \left(\underbrace{r_0^{(0)} r_0^{(1)} r_0^{(2)}}_{\text{for } u_0}, \underbrace{r_1^{(0)} r_1^{(1)} r_1^{(2)}}_{\text{for } u_1}, \dots, \underbrace{r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)}}_{\text{for } u_{K-1}} \right).$$
- Assume 0 is mapped to -1 and 1 to +1.
- Then for an AWGN channel, we define the log-likelihood ratio (L-value) $L(u_i | r_i^{(0)})$ (before decoding) of a transmitted information bit u_i given the received value $r_i^{(0)}$ as

$$\begin{aligned} \underline{L(u_i | r_i^{(0)})} &= \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})} \\ &= \ln \left(\frac{P(r_i^{(0)} | u_i = +1) \cancel{P(u_i = +1)}}{P(r_i^{(0)} | u_i = -1) \cancel{P(u_i = -1)}} \right) \end{aligned}$$

So this is one term we have and this is another term we have so $\log(a)$ times $b \log(a) + \log(b)$

(Refer Slide Time: 38:31)

Turbo decoding

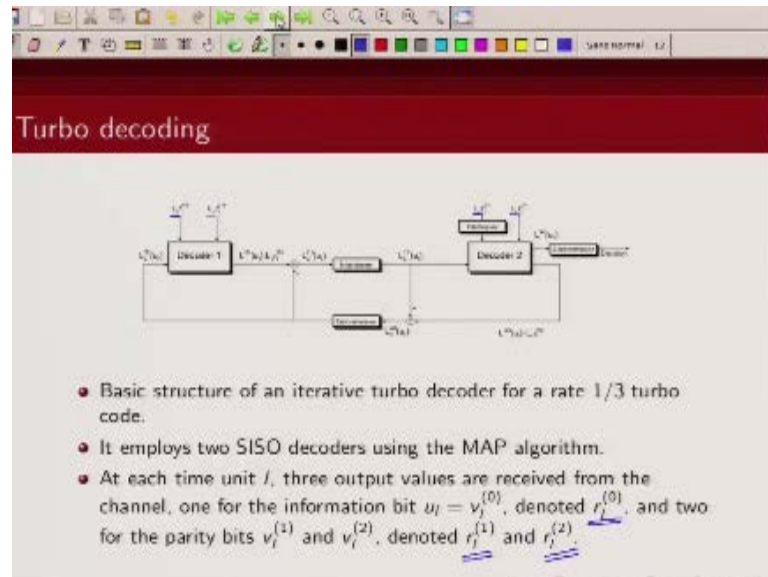
$$\begin{aligned}
 L(u_i | r_i^{(0)}) &= \ln \frac{P(r_i^{(0)} | u_i = +1)}{P(r_i^{(0)} | u_i = -1)} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= \ln \frac{e^{-\frac{E_s}{N_0}(r_i^{(0)} - 1)^2}}{e^{-\frac{E_s}{N_0}(r_i^{(0)} + 1)^2}} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= \frac{E_s}{N_0} \left\{ (r_i^{(0)} - 1)^2 - (r_i^{(0)} + 1)^2 \right\} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= \boxed{4 \frac{E_s}{N_0} r_i^{(0)}} + \boxed{\ln \frac{P(u_i = +1)}{P(u_i = -1)}} \\
 &= L_c r_i^{(0)} + L_a(u_i),
 \end{aligned}$$

where E_s/N_0 is the channel SNR, u_i and $r_i^{(0)}$ have both been normalized by a factor of $\sqrt{E_s}$, $L_c = 4(E_s/N_0)$ is the channel reliability factor and

So we can write it as this is one term and this is another term, now we are talking about additive white Gaussian noise channel so we can find out what is the likelihood ratio so we plug that value in here and after simplification what we get is a term of the form this, so there are two terms one is this and another is this, now what is this is $4(E_s/N_0) r_s^0$, r_s^0 is my information received information sequence.

So this you can $4 E_s/n_0$ I am writing as L_c and calling it a channel reliability factor. So it depends on the signal to noise ratio of the channel and this you can see \log of $P(u_i = +1)/P(u_i = -1)$ this is a-priori knowledge about the information bit being +1 or -1. So there are two inputs, one coming from the channel which is multiplied by this factor L_c and other is a-priori knowledge. And that is why you noticed here in the diagram where I had

(Refer Slide Time: 39:51)



I had L_c times $r_s^{(0)}$ L_c times $r_l^{(1)}$, this was this channel here at L_c times $r_l^{(0)}$ L_c times r_l so these values received values was scaled by this channel reliability factor as you have just derived here.

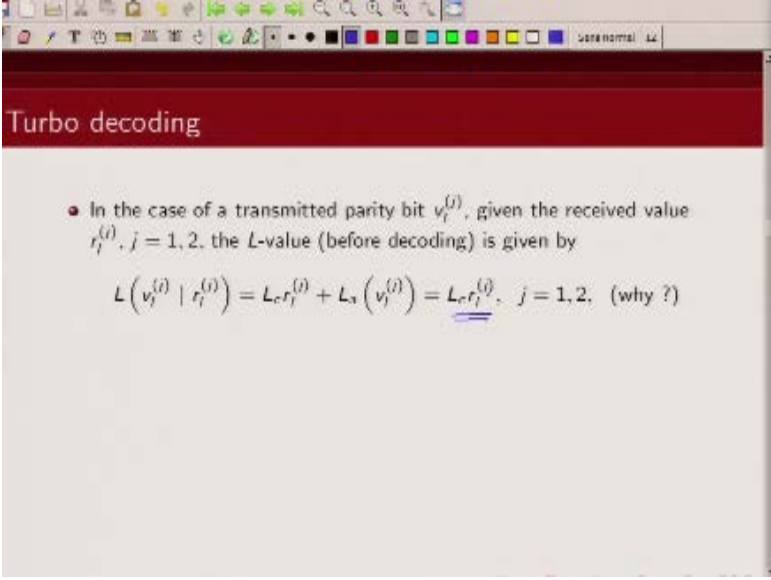
(Refer Slide Time: 40:10)

Turbo decoding

$$\begin{aligned}
 L(u_i | r_i^{(n)}) &= \ln \frac{P(r_i^{(n)} | u_i = +1)}{P(r_i^{(n)} | u_i = -1)} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= \ln \frac{e^{-\frac{E_s}{N_0}(r_i^{(n)} - 1)^2}}{e^{-\frac{E_s}{N_0}(r_i^{(n)} + 1)^2}} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= -\frac{E_s}{N_0} \left\{ (r_i^{(n)} - 1)^2 - (r_i^{(n)} + 1)^2 \right\} + \ln \frac{P(u_i = +1)}{P(u_i = -1)} \\
 &= \boxed{4 \frac{E_s}{N_0} r_i^{(n)}} + \boxed{\ln \frac{P(u_i = +1)}{P(u_i = -1)}} \\
 &= \underline{L_c r_i^{(n)}} + \underline{L_a(u_i)},
 \end{aligned}$$

where E_s/N_0 is the channel SNR, u_i and $r_i^{(n)}$ have both been normalized by a factor of $\sqrt{E_s}$, $L_c = 4(E_s/N_0)$ is the channel reliability factor and

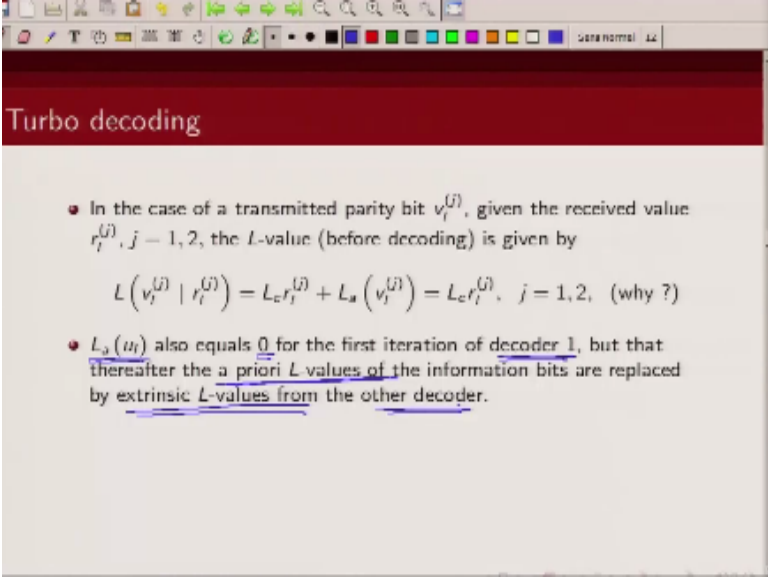
(Refer Slide Time: 40:13)



The image is a screenshot of a presentation slide. At the top, there is a red header bar with the text "Turbo decoding" in white. Below the header, the slide contains a bullet point and a mathematical equation. The bullet point states: "In the case of a transmitted parity bit $v_i^{(j)}$, given the received value $r_i^{(j)}$, $j = 1, 2$, the L -value (before decoding) is given by". Below this, the equation is displayed:
$$L(v_i^{(j)} | r_i^{(j)}) = L_{cr_i^{(j)}} + L_s(v_i^{(j)}) = \underline{L_{cr_i^{(j)}}}, \quad j = 1, 2, \quad (\text{why ?})$$

Now in case of transmitted parity bit we do not have any a-priori knowledge so in those case this will be just L_{cr_1} or 2 depending on whether we are considering parity bit 1 or parity 2.

(Refer Slide Time: 40:33)



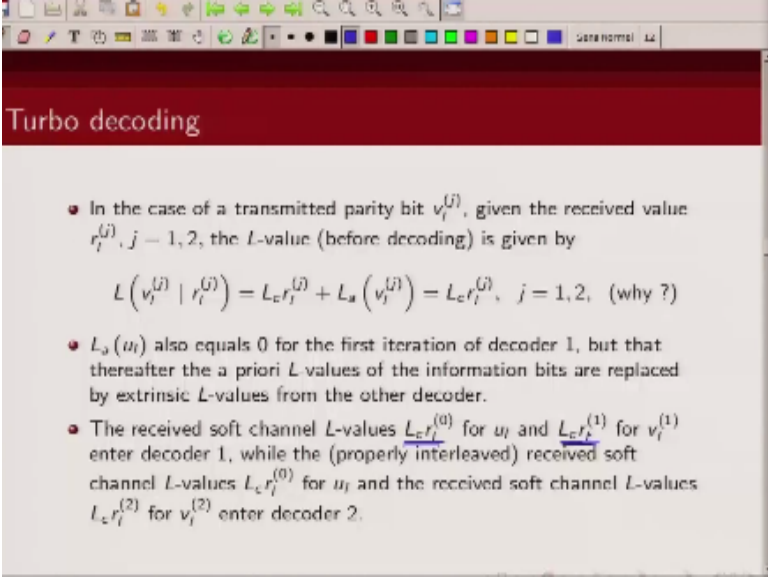
The image is a screenshot of a presentation slide titled "Turbo decoding". The slide has a red header bar with the title in white. The main content area is white with black text. It contains two bullet points and a mathematical equation. The first bullet point discusses the L-value for a transmitted parity bit given a received value. The second bullet point discusses the initial L-value for information bits and how it is updated. The equation shows the relationship between the total L-value, the channel L-value, and the a-priori L-value for parity bits.

Turbo decoding

- In the case of a transmitted parity bit $v_i^{(j)}$, given the received value $r_i^{(j)}$, $j = 1, 2$, the L -value (before decoding) is given by
$$L\left(v_i^{(j)} \mid r_i^{(j)}\right) = L_c r_i^{(j)} + L_a\left(v_i^{(j)}\right) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_a(u_i)$ also equals 0 for the first iteration of decoder 1, but that thereafter the a priori L values of the information bits are replaced by extrinsic L -values from the other decoder.

As I said initially we do not have any a-priori knowledge about whether the bit is +1 or -1. So we would assume this equally likely +1 and -1, so the a-priori log likelihood value 1 value will be considered as 0 for the first decoder. But thereafter this a-priori values will be replaced by the extrinsic values received from the other decoder right.

(Refer Slide Time: 41:15)



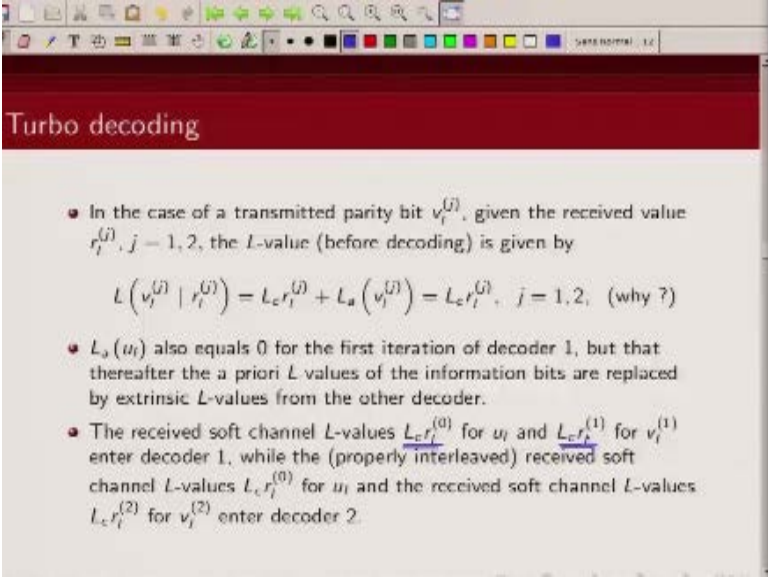
The image is a screenshot of a presentation slide titled "Turbo decoding". The slide has a red header bar with the title in white. The main content area is white with black text. It contains three bullet points and a mathematical equation. The first bullet point discusses the L-value for a transmitted parity bit given a received value. The second bullet point discusses the L-value for information bits. The third bullet point discusses the flow of L-values between two decoders. The equation shows the relationship between the total L-value, the channel L-value, and the extrinsic L-value for a parity bit.

Turbo decoding

- In the case of a transmitted parity bit $v_i^{(j)}$, given the received value $r_i^{(j)}$, $j = 1, 2$, the L -value (before decoding) is given by
$$L\left(v_i^{(j)} \mid r_i^{(j)}\right) = L_c r_i^{(j)} + L_u\left(v_i^{(j)}\right) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_u(u_i)$ also equals 0 for the first iteration of decoder 1, but that thereafter the a priori L values of the information bits are replaced by extrinsic L -values from the other decoder.
- The received soft channel L -values $L_c r_i^{(0)}$ for u_i and $L_c r_i^{(1)}$ for $v_i^{(1)}$ enter decoder 1, while the (properly interleaved) received soft channel L -values $L_c r_i^{(0)}$ for u_i and the received soft channel L -values $L_c r_i^{(2)}$ for $v_i^{(2)}$ enter decoder 2.

And again the received values that we got from the channel $L_c r_1^{(0)}$ and $r_1^{(1)}$ and $r_1^{(2)}$ remember to feed them in proper order when you are feeding to decoder 1 and decoder 2.

(Refer Slide Time: 41:35)



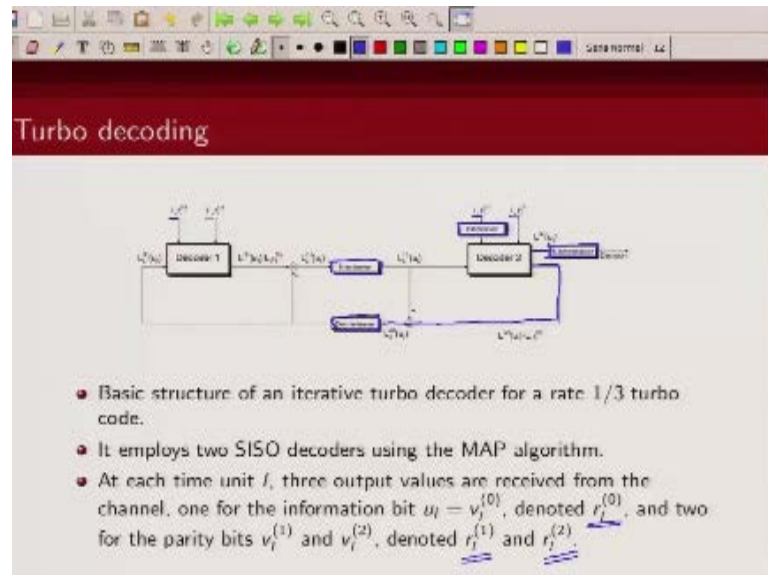
The image is a screenshot of a presentation slide titled "Turbo decoding". The slide has a red header bar with the title in white. Below the header, there is a list of three bullet points and a mathematical equation. The first bullet point discusses the L-value for a transmitted parity bit given a received value. The second bullet point mentions that L_a(u_i) equals 0 for the first iteration of decoder 1. The third bullet point describes the flow of L-values between two decoders. The equation shows the relationship between the total L-value, the extrinsic L-value, and the a priori L-value for a parity bit.

Turbo decoding

- In the case of a transmitted parity bit $v_i^{(j)}$, given the received value $r_i^{(j)}$, $j = 1, 2$, the L -value (before decoding) is given by
$$L\left(v_i^{(j)} \mid r_i^{(j)}\right) = L_e r_i^{(j)} + L_a\left(v_i^{(j)}\right) = L_e r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_a(u_i)$ also equals 0 for the first iteration of decoder 1, but that thereafter the a priori L values of the information bits are replaced by extrinsic L -values from the other decoder.
- The received soft channel L -values $L_e r_i^{(0)}$ for u_i and $L_e r_i^{(1)}$ for $v_i^{(1)}$ enter decoder 1, while the (properly interleaved) received soft channel L -values $L_e r_j^{(0)}$ for u_i and the received soft channel L -values $L_e r_i^{(2)}$ for $v_i^{(2)}$ enter decoder 2.

And this we have explained also earlier you can see.

(Refer Slide Time: 41:40)



When we feed the extrinsic information from decoder 1 to decoder 2 we are interleaving it. Similarly the information bit that we are feeding to decoder 2 that is the interleaved version of information coming from decoder 1. Similarly from decoder 2 if we are feeding something back to decoder 1 we are doing D interleaver and when we are taking decision from decoder 2 we are also again doing D interleaving. So this interleaving D interleaving is done so as the order of the information bit is preserved in the fashion they are entering encoder 1 and encoder 2

(Refer Slide Time: 42:16)

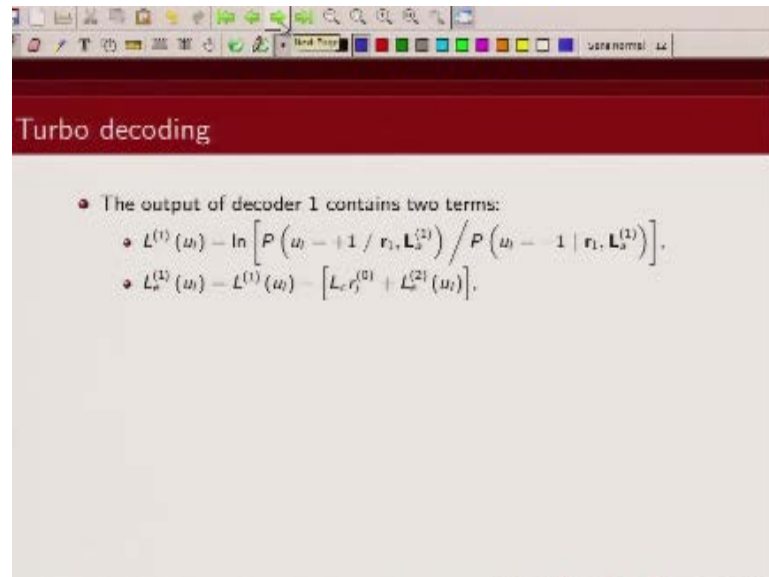
Turbo decoding

- The $3K$ -dimensional received vector is denoted by

$$\mathbf{r} = \left(\underline{r_0^{(0)}}, \underline{r_0^{(1)}}, \underline{r_0^{(2)}}, \underline{r_1^{(0)}}, \underline{r_1^{(1)}}, \underline{r_1^{(2)}}, \dots, \underline{r_{K-1}^{(0)}}, \underline{r_{K-1}^{(1)}}, \underline{r_{K-1}^{(2)}} \right).$$
- Assume 0 is mapped to -1 and 1 to $+1$.
- Then for an AWGN channel, we define the log-likelihood ratio (L-value) $L(u_i | r_i^{(0)})$ (before decoding) of a transmitted information bit u_i given the received value $r_i^{(0)}$ as

$$\begin{aligned} \underline{L(u_i | r_i^{(0)})} &= \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})} \\ &= \ln \left(\frac{P(r_i^{(0)} | u_i = +1)}{P(r_i^{(0)} | u_i = -1)} \right) \left(\frac{P(u_i = +1)}{P(u_i = -1)} \right) \end{aligned}$$

(Refer Slide Time: 42:19)



Turbo decoding

- The output of decoder 1 contains two terms:
 - $L^{(1)}(u_i) = \ln \left[\frac{P(u_i = +1 | r_i, L_a^{(2)})}{P(u_i = -1 | r_i, L_a^{(2)})} \right]$.
 - $L_a^{(1)}(u_i) = L^{(1)}(u_i) + [L_c(r_i^{(0)}) + L_a^{(2)}(u_i)]$.

So as I said there are three inputs to this decoder, this channel received value corresponding to information sequence and parity bit and a-priori value and there are two output, one is this.

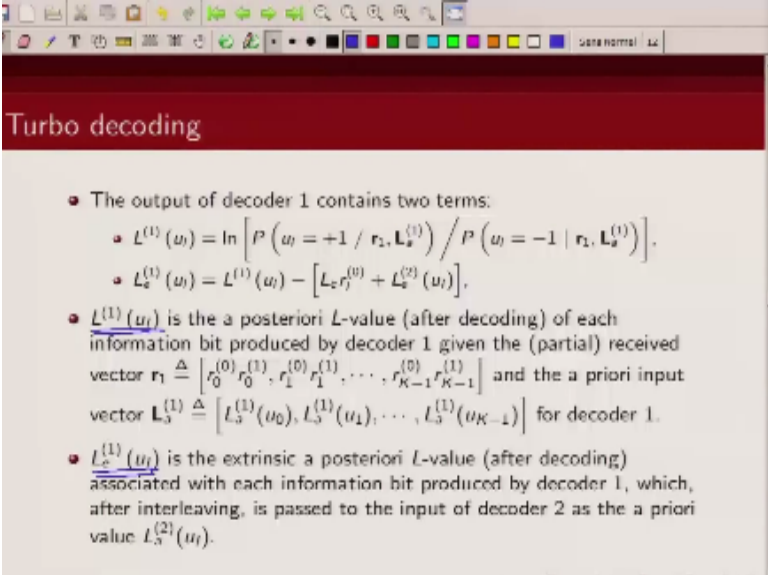
(Refer Slide Time: 42:37)

Turbo decoding

- The output of decoder 1 contains two terms:
 - $L^{(1)}(u_i) = \ln \left[\frac{P(u_i = +1 / r_i, L_a^{(2)})}{P(u_i = -1 / r_i, L_a^{(2)})} \right]$
 - $L_a^{(2)}(u_i) = L^{(1)}(u_i) - [L_c r_i^{(2)}] + [L_a^{(1)}(u_i)]$

APP L value that we have computed and the second is extrinsic value and how are we computing extrinsic value from the APP L value, we are subtracting the contribution of the received channel value and we are subtracting the contribution of a-priori value which is nothing but extrinsic value of the other decoder.

(Refer Slide Time: 43:07)

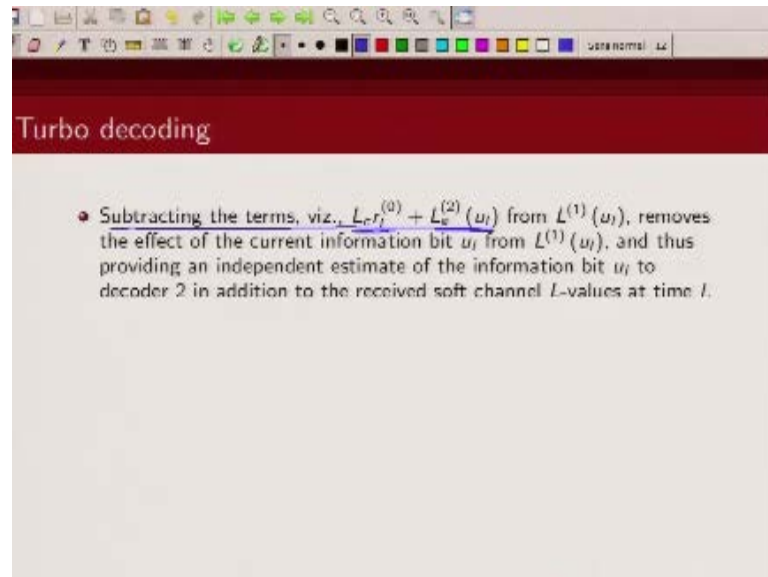


Turbo decoding

- The output of decoder 1 contains two terms:
 - $L^{(1)}(u_i) = \ln \left[\frac{P(u_i = +1 \mid r_1, \mathbf{L}_s^{(1)})}{P(u_i = -1 \mid r_1, \mathbf{L}_s^{(1)})} \right]$.
 - $L_s^{(1)}(u_i) = L^{(1)}(u_i) - [L_s^{(0)} + L_e^{(2)}(u_i)]$.
- $L^{(1)}(u_i)$ is the a posteriori L -value (after decoding) of each information bit produced by decoder 1 given the (partial) received vector $\mathbf{r}_1 \triangleq [r_0^{(0)} r_0^{(1)}, r_1^{(0)} r_1^{(1)}, \dots, r_{K-1}^{(0)} r_{K-1}^{(1)}]$ and the a priori input vector $\mathbf{L}_s^{(1)} \triangleq [L_s^{(1)}(u_0), L_s^{(1)}(u_1), \dots, L_s^{(1)}(u_{K-1})]$ for decoder 1.
- $L_e^{(1)}(u_i)$ is the extrinsic a posteriori L -value (after decoding) associated with each information bit produced by decoder 1, which, after interleaving, is passed to the input of decoder 2 as the a priori value $L_s^{(2)}(u_i)$.

Okay, so this we have explained, so this is the APP L value and this is the extrinsic L value.

(Refer Slide Time: 43:20)



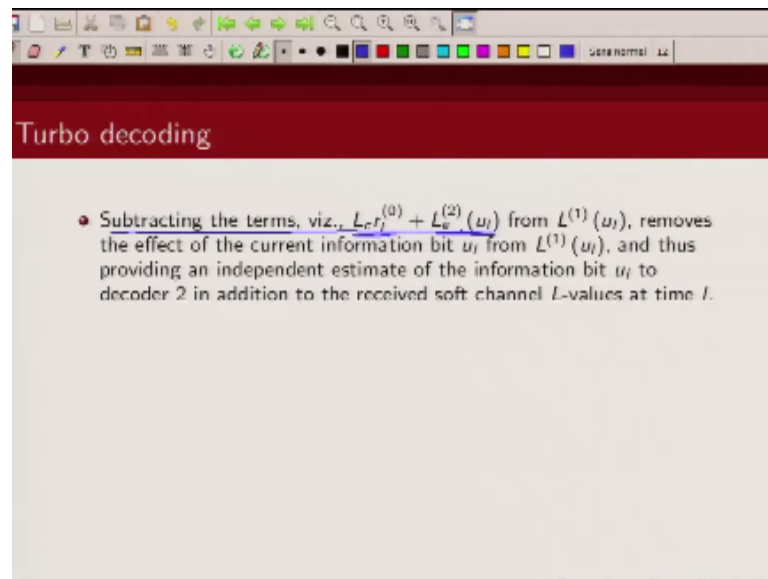
The image is a screenshot of a presentation slide. At the top, there is a red header bar with the text "Turbo decoding" in white. Below the header, the slide has a light gray background. A single bullet point is centered on the slide. The text of the bullet point describes the process of subtracting certain terms from a value to provide an independent estimate of an information bit.

Turbo decoding

- Subtracting the terms, viz., $L_{r,l}^{(a)} + L_v^{(2)}(u_l)$ from $L^{(1)}(u_l)$, removes the effect of the current information bit u_l from $L^{(1)}(u_l)$, and thus providing an independent estimate of the information bit u_l to decoder 2 in addition to the received soft channel L -values at time l .

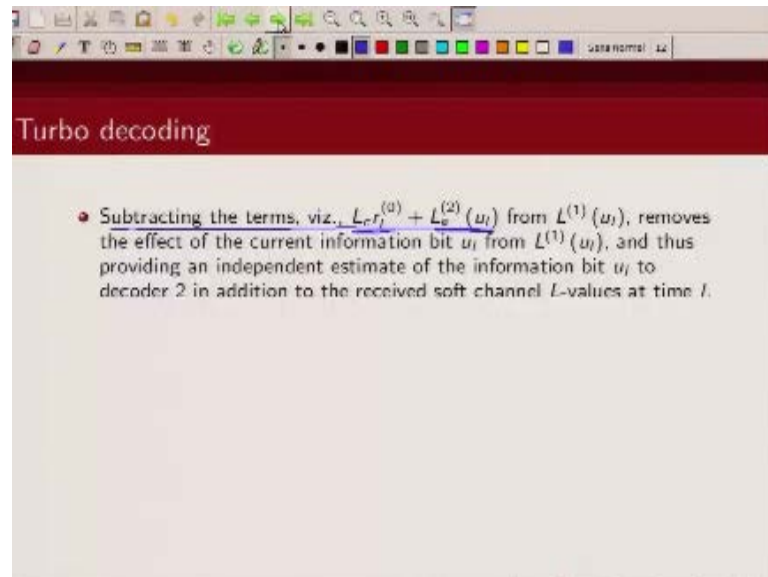
So why are we subtracting these terms this information bit term and the a-priori term, so we essentially are trying to remove the effect of the current bit in some sense from the APP values so in some sense we are trying to provide some independent estimate because this

(Refer Slide Time: 43:46)



Anywhere this a-priori information has come from the extrinsic information from the previous decoder, so there is no point feeding the same information back to the same decoder. And the received values are already received by the decoder so we are trying to some in one way trying to send some independent estimate about what we think the bits are to the other decoder and that is the whole idea behind this iterative process that you want to give some sort of independent estimate. If you try to feed the same information back then it will become a positive feedback system and unstable.

(Refer Slide Time: 44:22)



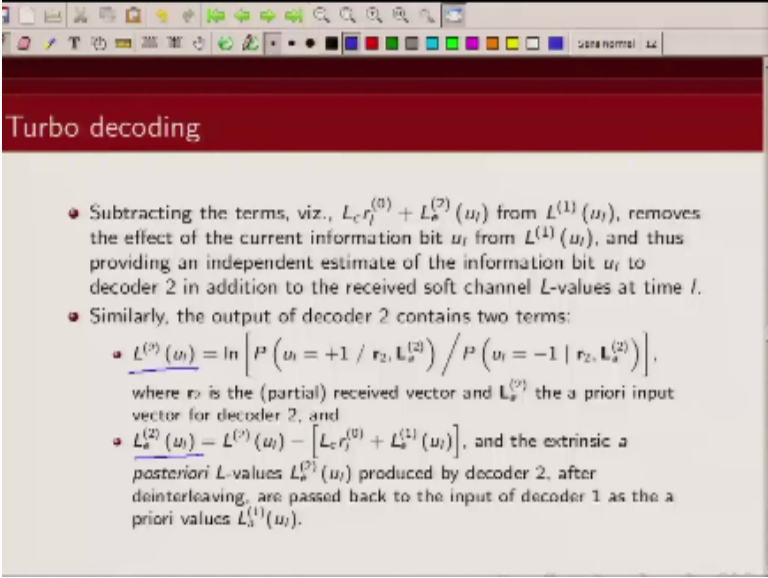
The image is a screenshot of a presentation slide. At the top, there is a red header bar with the text "Turbo decoding" in white. Below the header, the slide has a light beige background. A single bullet point is centered on the slide. The text of the bullet point describes the process of subtracting terms from $L^{(1)}(u_i)$ to provide an independent estimate of the information bit u_i to decoder 2. The slide is displayed within a window that has a standard toolbar at the top and a title bar that says "data format 12".

Turbo decoding

- Subtracting the terms, viz., $L_{r_i}^{(d)} + L_v^{(2)}(u_i)$ from $L^{(1)}(u_i)$, removes the effect of the current information bit u_i from $L^{(1)}(u_i)$, and thus providing an independent estimate of the information bit u_i to decoder 2 in addition to the received soft channel L -values at time i .

The decoder may not converge, so that we do not want.

(Refer Slide Time: 44:24)



Turbo decoding

- Subtracting the terms, viz., $L_c r_i^{(2)} + L_s^{(2)}(u_i)$ from $L^{(1)}(u_i)$, removes the effect of the current information bit u_i from $L^{(1)}(u_i)$, and thus providing an independent estimate of the information bit u_i to decoder 2 in addition to the received soft channel L -values at time i .
- Similarly, the output of decoder 2 contains two terms:
 - $L^{(2)}(u_i) = \ln \left[\frac{P(u_i = +1 | r_2, L_s^{(2)})}{P(u_i = -1 | r_2, L_s^{(2)})} \right]$, where r_2 is the (partial) received vector and $L_s^{(2)}$ the a priori input vector for decoder 2, and
 - $L_s^{(2)}(u_i) = L^{(2)}(u_i) - [L_c r_i^{(2)} + L_s^{(1)}(u_i)]$, and the extrinsic a posteriori L -values $L_e^{(2)}(u_i)$ produced by decoder 2, after deinterleaving, are passed back to the input of decoder 1 as the a priori values $L_s^{(1)}(u_i)$.

And in the same fashion the decoder 2 will also have two terms, one is this APP value and other is this extrinsic value.

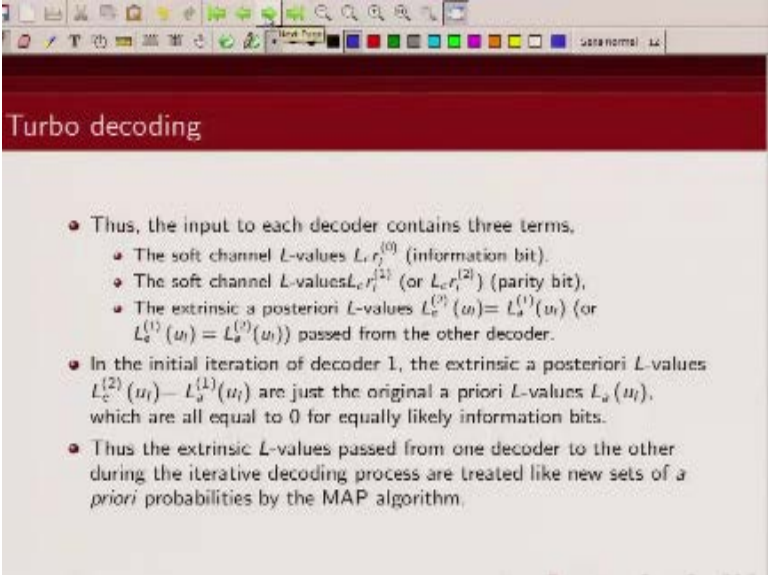
(Refer Slide Time: 44:35)

Turbo decoding

- Thus, the input to each decoder contains three terms.
 - The soft channel L -values $L_{cr_i}^{(0)}$ (information bit).
 - The soft channel L -values $L_{cr_i}^{(1)}$ (or $L_{cr_i}^{(2)}$) (parity bit).
 - The extrinsic a posteriori L -values $L_e^{(2)}(u) - L_e^{(1)}(u)$ (or $L_e^{(1)}(u) - L_e^{(2)}(u)$) passed from the other decoder.

I have already explained what are the inputs to the decoder. I will again repeat one term corresponding to the received information bit, one term corresponding to the received parity bit and one term corresponding to the a-priori information which have been fed from the second decoder. You can see basically a-priori information for decoder 1 is nothing but extrinsic information coming from decoder 2, after proper interleaving D interleaving. Because you want to ensure that the order of the information bit is same.

(Refer Slide Time: 45:23)

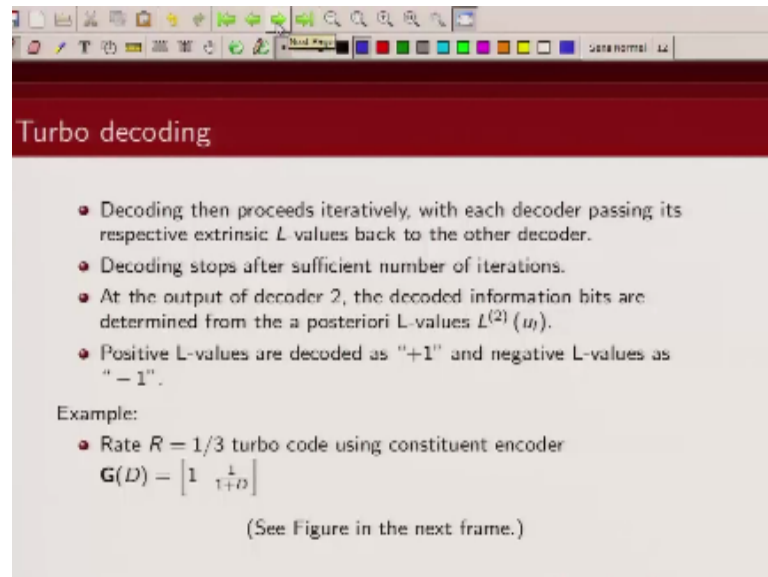


Turbo decoding

- Thus, the input to each decoder contains three terms.
 - The soft channel L -values $L_c r_i^{(0)}$ (information bit).
 - The soft channel L -values $L_c r_i^{(2)}$ (or $L_c r_i^{(1)}$) (parity bit).
 - The extrinsic a posteriori L -values $L_e^{(0)}(u_i) = L_a^{(1)}(u_i)$ (or $L_e^{(1)}(u_i) = L_a^{(2)}(u_i)$) passed from the other decoder.
- In the initial iteration of decoder 1, the extrinsic a posteriori L -values $L_e^{(2)}(u_i) - L_a^{(1)}(u_i)$ are just the original a priori L -values $L_a(u_i)$, which are all equal to 0 for equally likely information bits.
- Thus the extrinsic L -values passed from one decoder to the other during the iterative decoding process are treated like new sets of a priori probabilities by the MAP algorithm.

Okay this I have explained in the initial iteration extrinsic information is basically 0 and subsequently a-priori information is 0, and subsequently the a-priori value will be nothing but the extrinsic information.

(Refer Slide Time: 45:43)



Turbo decoding

- Decoding then proceeds iteratively, with each decoder passing its respective extrinsic L values back to the other decoder.
- Decoding stops after sufficient number of iterations.
- At the output of decoder 2, the decoded information bits are determined from the a posteriori L -values $L^{(2)}(u_i)$.
- Positive L -values are decoded as "+1" and negative L -values as "-1".

Example:

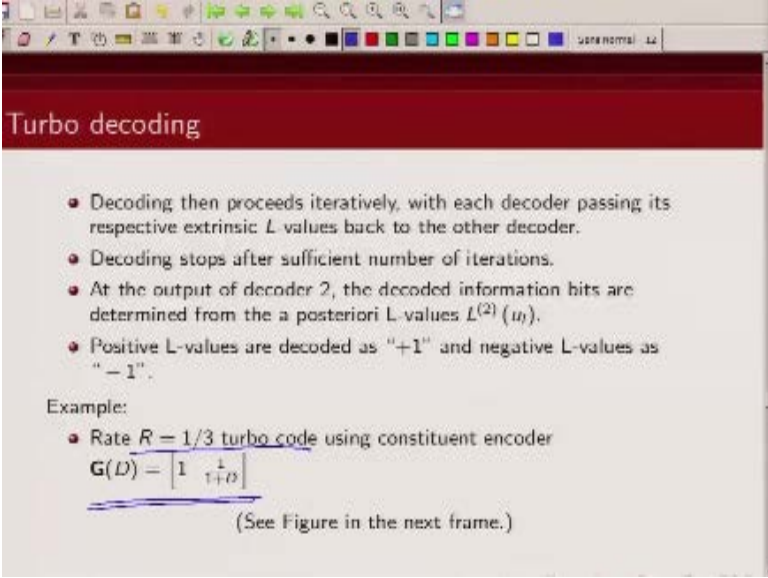
- Rate $R = 1/3$ turbo code using constituent encoder

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 1 \\ 1 & 1+D \end{bmatrix}$$

(See Figure in the next frame.)

So this process as I said goes on repeatedly iterative fashion. So let us take an example and see how this works.

(Refer Slide Time: 45:58)



The image is a screenshot of a presentation slide titled "Turbo decoding". The slide has a red header bar with the title in white. Below the header, there is a list of four bullet points describing the iterative decoding process. The text is in a standard sans-serif font. The slide is displayed within a window that has a standard operating system toolbar at the top.

Turbo decoding

- Decoding then proceeds iteratively, with each decoder passing its respective extrinsic L values back to the other decoder.
- Decoding stops after sufficient number of iterations.
- At the output of decoder 2, the decoded information bits are determined from the a posteriori L -values $L^{(2)}(u)$.
- Positive L -values are decoded as "+1" and negative L -values as "-1".

Example:

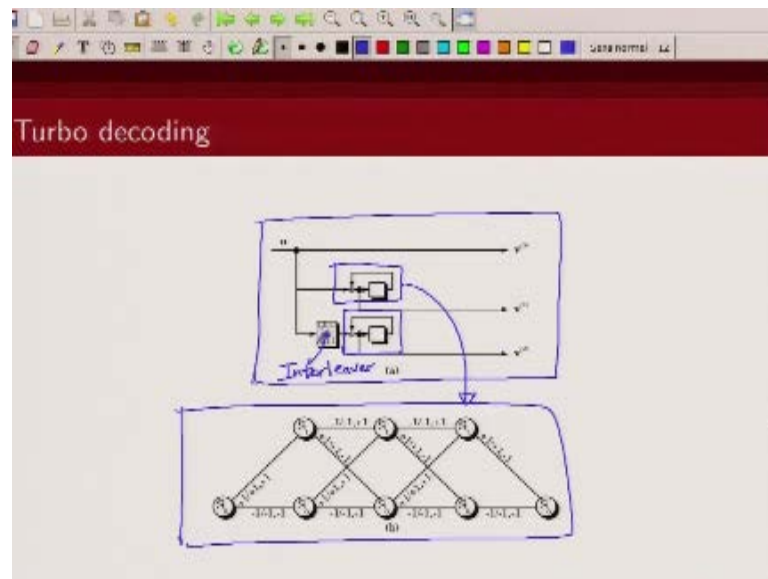
- Rate $R = 1/3$ turbo code using constituent encoder

$$\mathbf{G}(D) = \begin{bmatrix} 1 & 1+D \end{bmatrix}$$

(See Figure in the next frame.)

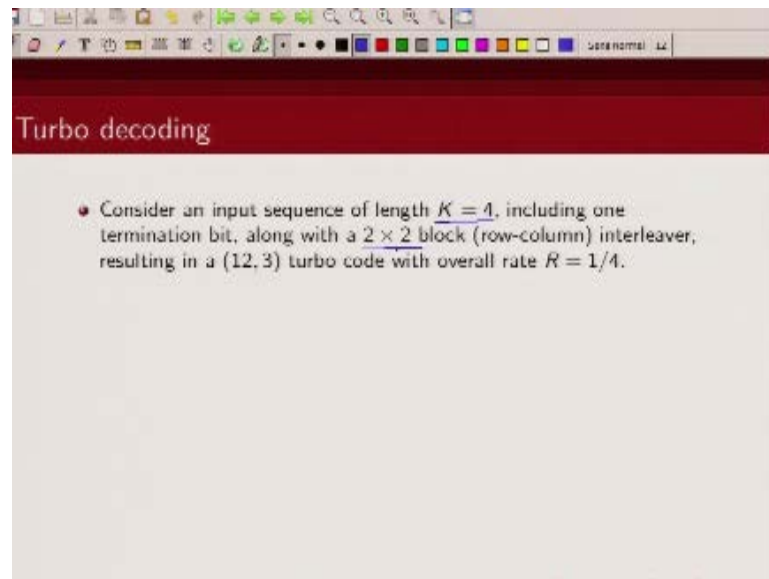
So we are considering a rate 1/3 turbo code where the constituent encoder is this two state encoder.

(Refer Slide Time: 46:08)



So our turbo code is this, each one of them is using this two state recursive convolutional encoder. This is my, this is my interleaver and this is the state diagram corresponding to these convolutional encoder okay.

(Refer Slide Time: 46:40)

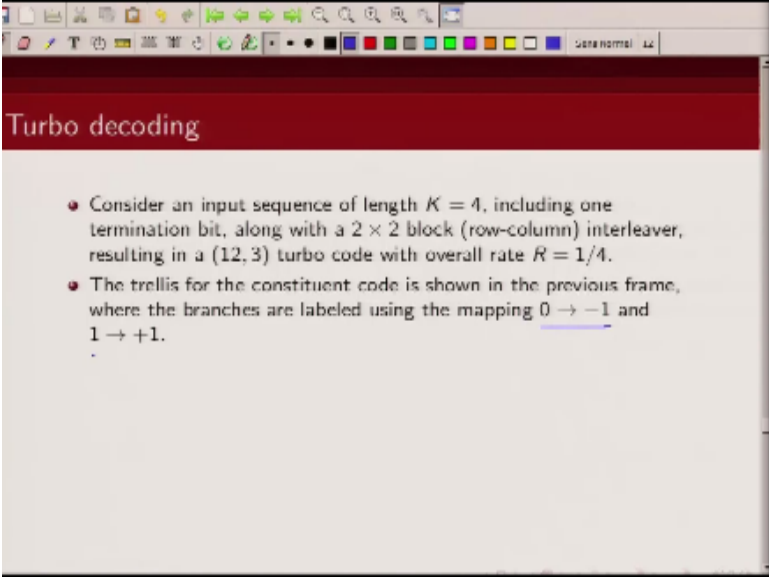


Turbo decoding

- Consider an input sequence of length $K = 4$, including one termination bit, along with a 2×2 block (row-column) interleaver, resulting in a $(12, 3)$ turbo code with overall rate $R = 1/4$.

Consider a information bit length of 4 and let us say I am doing block interleaving so I am feeding the data block wise and I am reading it.

(Refer Slide Time: 46:55)

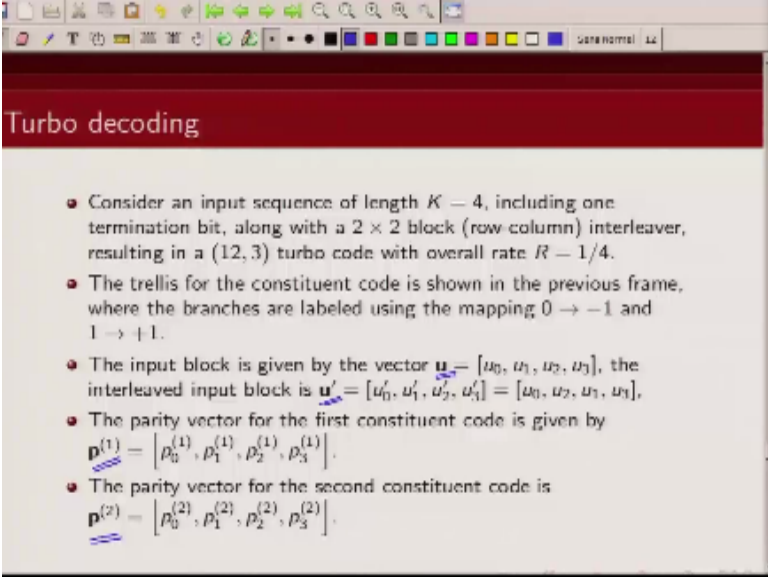


Turbo decoding

- Consider an input sequence of length $K = 4$, including one termination bit, along with a 2×2 block (row-column) interleaver, resulting in a $(12, 3)$ turbo code with overall rate $R = 1/4$.
- The trellis for the constituent code is shown in the previous frame, where the branches are labeled using the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$.
Column

Column wise, so as I said I am mapping 0 to -1 here and 1 to +1.

(Refer Slide Time: 47:04)



Turbo decoding

- Consider an input sequence of length $K = 4$, including one termination bit, along with a 2×2 block (row column) interleaver, resulting in a $(12, 3)$ turbo code with overall rate $R = 1/4$.
- The trellis for the constituent code is shown in the previous frame, where the branches are labeled using the mapping $0 \rightarrow -1$ and $1 \rightarrow +1$.
- The input block is given by the vector $\mathbf{u} = [u_0, u_1, u_2, u_3]$, the interleaved input block is $\mathbf{u}' = [u'_0, u'_1, u'_2, u'_3] = [u_0, u_2, u_1, u_3]$.
- The parity vector for the first constituent code is given by $\mathbf{p}^{(1)} = [p_0^{(1)}, p_1^{(1)}, p_2^{(1)}, p_3^{(1)}]$.
- The parity vector for the second constituent code is $\mathbf{p}^{(2)} = [p_0^{(2)}, p_1^{(2)}, p_2^{(2)}, p_3^{(2)}]$.

So if this is my input block the interleave block is given by \mathbf{u}' and corresponding parity for the first encoder is given by this and the parity due to second encoder is given by this. So I use notation $\mathbf{p}^{(1)}$ to denote the parity coming from the first encoder $\mathbf{p}^{(2)}$ to denote the parity coming from the second encoder, used my information sequence \mathbf{u}' is the interleaved version of information sequence which has been fed to encoder 2.

(Refer Slide Time: 47:44)

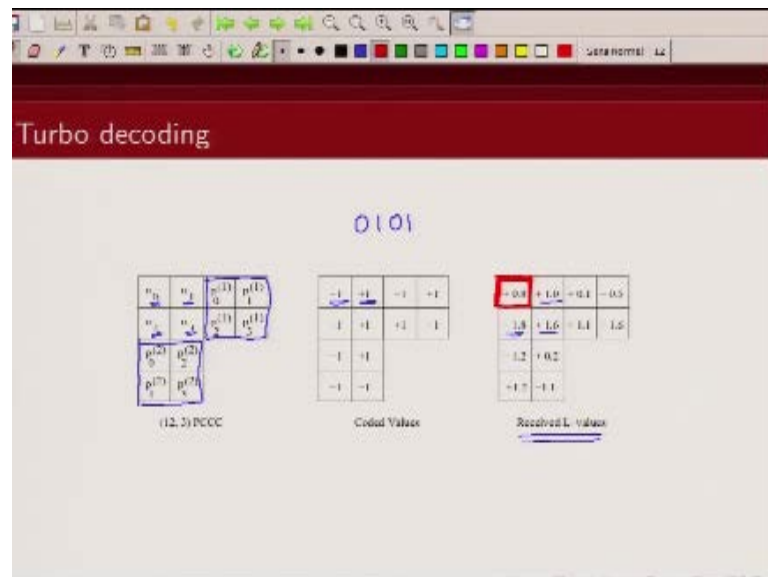
Turbo decoding

- The 12 transmitted bits are represented in a rectangular array, as shown in Figure in the next frame, where the input vector \mathbf{u} determines the parity vector $\mathbf{p}^{(1)}$ in the first two rows and the interleaved input vector \mathbf{u}^i determines the parity vector $\mathbf{p}^{(2)}$ in the first two columns.
- For purposes of illustration, we assume the particular bit values shown in Figure.
- We also assume a channel SNR of $E_s/N_0 = 1/4$ (-6.02dB), so that the received channel L -values corresponding to the received vector $\mathbf{r} = [r_0^{(0)} r_0^{(1)} r_0^{(2)} r_1^{(0)} r_1^{(1)} r_1^{(2)} r_2^{(0)} r_2^{(1)} r_2^{(2)} r_3^{(0)} r_3^{(1)} r_3^{(2)}]$ are given by

$$\underline{L_c r_i^{(j)}} = 4 \left(\frac{E_s}{N_0} \right) r_i^{(j)} = \underline{r_i^{(j)}}, \quad i = 0, 1, 2, 3, \quad j = 0, 1, 2. \quad (1)$$

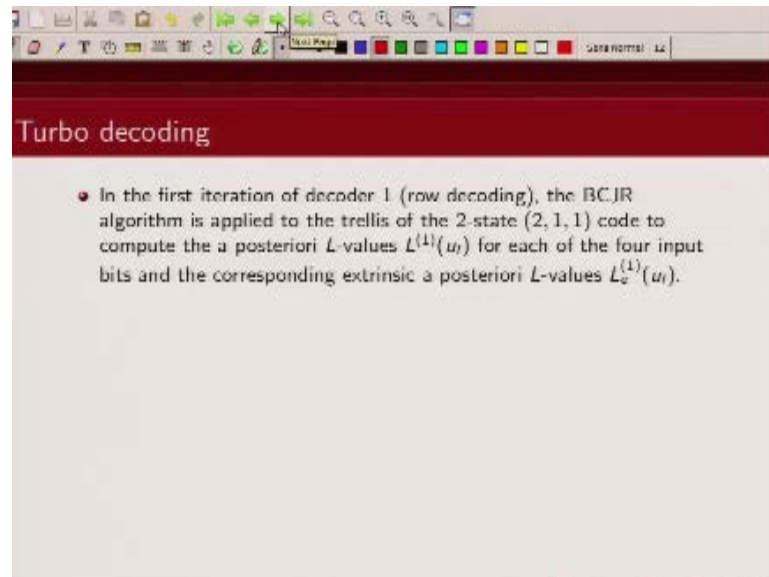
So I will just show you basically I will just, and I am assuming a channel $\frac{1}{4}$ so the likelihood channel like reliability factor L_c will be basically 1. So $L_c r_i$ will be in my case would be same as a received value for this particular signal to noise ratio. This is just a toy example to illustrate how this decoding works.

(Refer Slide Time: 48:15)



So these are my information bits $u_{(0)}, u_{(1)}, u_{(2)}, u_{(3)}$, so as I said if it is a 0 I am mapping it to -1 if it is a + I am mapping to +1. So the information sequence here $u_{(0)}$ is 0, $u_{(1)}$ is 1, $u_{(2)}$ is 0 and $u_{(3)}$ is 1. These are the corresponding parities for this information sequence from the encoder 1 and these are the corresponding parities from the encoder 2. Now what I have here is the received values. So you can see here, so this was transmitted as + I received a +, this transmitted as + I received a +, this was transmitted -1, this was received as -. But here the sign has changed, note here this was transmitted as -1, but what I received is + point 8. So this bit is received in error. Now let us see using this turbo decoding how we are able to correct this error.

(Refer Slide Time: 49:39)

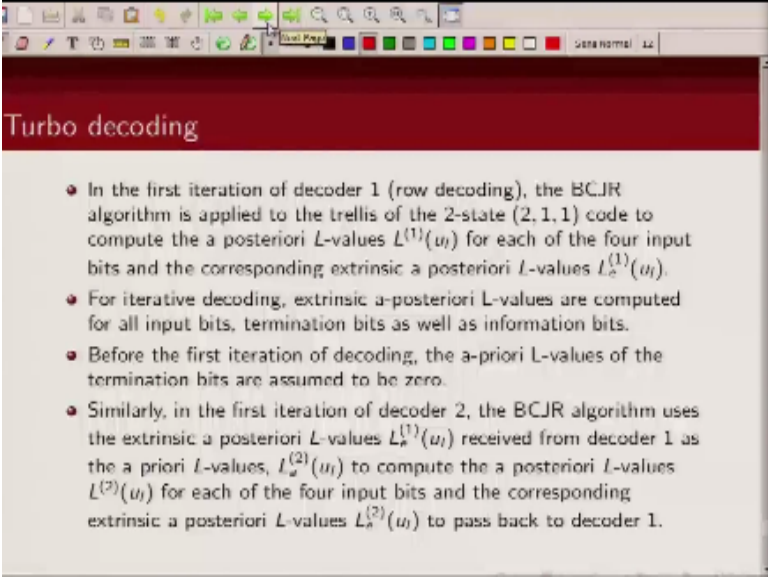


Turbo decoding

- In the first iteration of decoder 1 (row decoding), the BCJR algorithm is applied to the trellis of the 2-state (2, 1, 1) code to compute the a posteriori L -values $L^{(1)}(u_i)$ for each of the four input bits and the corresponding extrinsic a posteriori L -values $L_e^{(1)}(u_i)$.

So of course, the first half of decoding will be I will decode, I will have decoder 1 work first and then decoder 2.

(Refer Slide Time: 49:50)



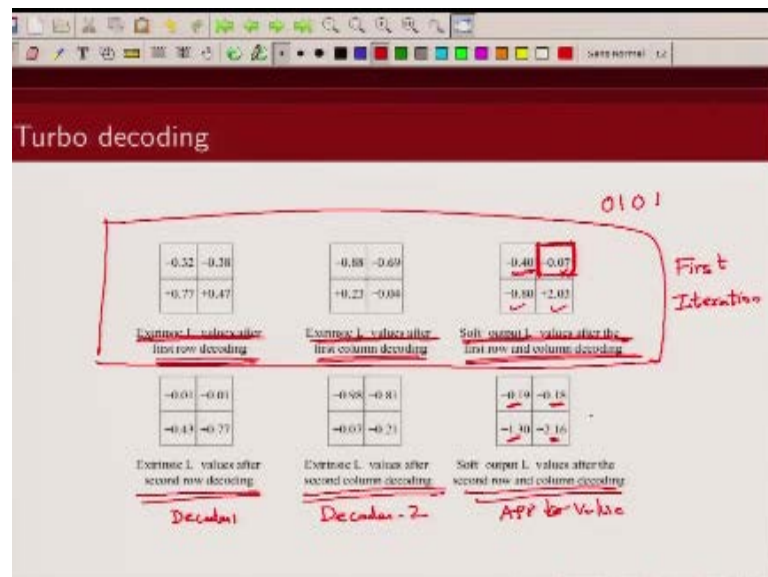
The image is a screenshot of a presentation slide titled "Turbo decoding". The slide is displayed within a window that has a standard Windows-style title bar and a toolbar. The title bar includes icons for file operations (like save, print, copy, paste) and a status bar at the bottom right showing "Slide 10/11" and a zoom level of "1.2". The slide content is as follows:

Turbo decoding

- In the first iteration of decoder 1 (row decoding), the BCJR algorithm is applied to the trellis of the 2-state (2, 1, 1) code to compute the a posteriori L -values $L^{(1)}(u_i)$ for each of the four input bits and the corresponding extrinsic a posteriori L -values $L_e^{(1)}(u_i)$.
- For iterative decoding, extrinsic a-posteriori L -values are computed for all input bits, termination bits as well as information bits.
- Before the first iteration of decoding, the a-priori L -values of the termination bits are assumed to be zero.
- Similarly, in the first iteration of decoder 2, the BCJR algorithm uses the extrinsic a posteriori L -values $L_e^{(1)}(u_i)$ received from decoder 1 as the a priori L -values, $L_a^{(2)}(u_i)$ to compute the a posteriori L -values $L^{(2)}(u_i)$ for each of the four input bits and the corresponding extrinsic a posteriori L -values $L_e^{(2)}(u_i)$ to pass back to decoder 1.

So I will just directly come to the values because I have already explained the whole procedure, I will just show you

(Refer Slide Time: 49:59)



The extrinsic information value and the APP values at the end of each iteration. So this is the extrinsic information after I have decoded using decoder 1. This is the extrinsic information after decoder 2, and this is the APP value after decoder 1. Note that I have transmitted 0101 so there the sign is okay, there the sign is not okay, this is after first decoding this is in error, this sign is okay, this sign is okay.

Now next so this whole thing was my first iteration, this is my first iteration, fine? Now what about second iteration so, so again I will first compute the extrinsic values this is after I do decoder 1 this is extrinsic information after I have done decoder 2 and this is the APP value L value, APP L value after decoder 2 and note here this was 0, this is 1, this is 0, this is 1. So I, after 2 iteration I am able to correct the transmission error. So with this I am going to conclude our discussion on turbo decoding. Thank you.

Acknowledgement

Ministry of Human Resource & Development

Prof. Satyaki Roy

Co-ordinator, NPTEL IIT Kanpur

NPTEL Team

Sanjay Pal

Ashish Singh

Badal Pradhan

Tapobrata Das

Ram Chandra

Dilip Tripathi

Manoj Shrivastava

Padam Shukla

Sanjay Mishra

Shubham Rawat

Shikha Gupta

K. K. Mishra

Aradhana Singh

Sweta

Ashutosh Gairola

Dilip Katiyar

Sharwan

Hari Ram

Bhadra Rao

Puneet Kumar Bajpai

Lalty Dutta

Ajay Kanaujia

Shivendra Kumar Tiwari

an IIT Kanpur Production

©copyright reserved