

**An Introduction to Coding Theory**  
**Professor Adrish Banerji**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Kanpur**  
**Module 07**  
**Lecture Number 28**  
**Turbo Decoding**

(Refer Slide Time 00:14)



Lecture #16A: Turbo Decoding



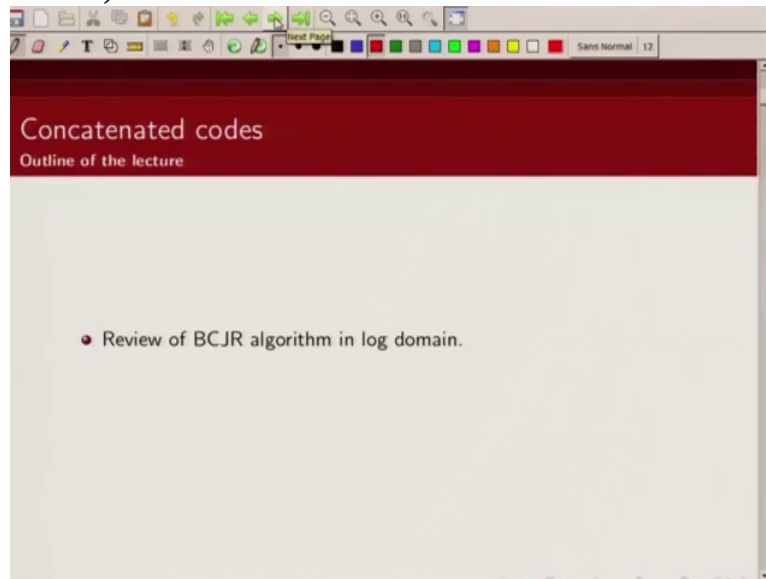
Today we are going to discuss about

(Refer Slide Time 00:16)



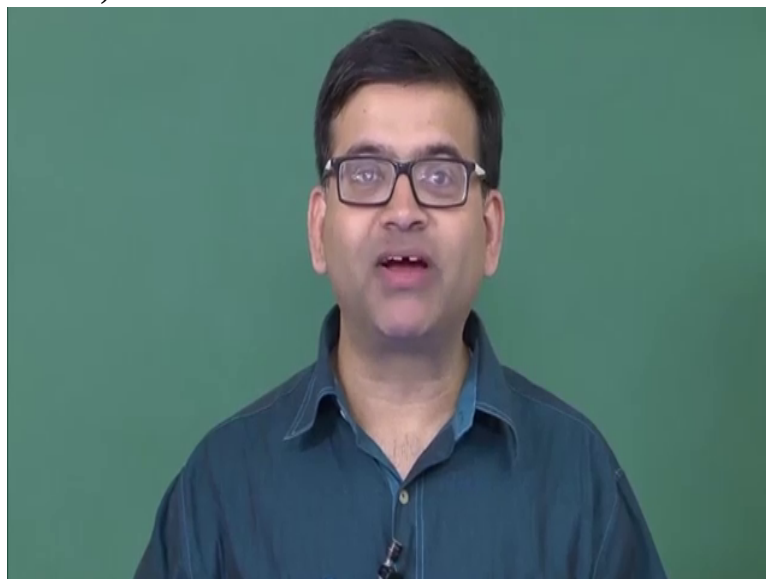
decoding of turbo codes.

(Refer Slide Time 00:19)



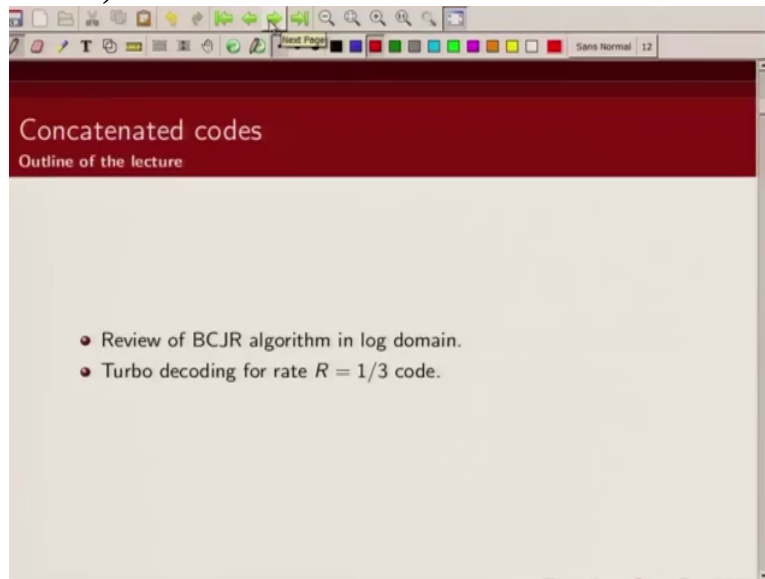
So to do that we will first review our B C J R algorithm in the log domain. We have talked about B C G R algorithm in the probability domain. We will very quickly

(Refer Slide Time 00:30)



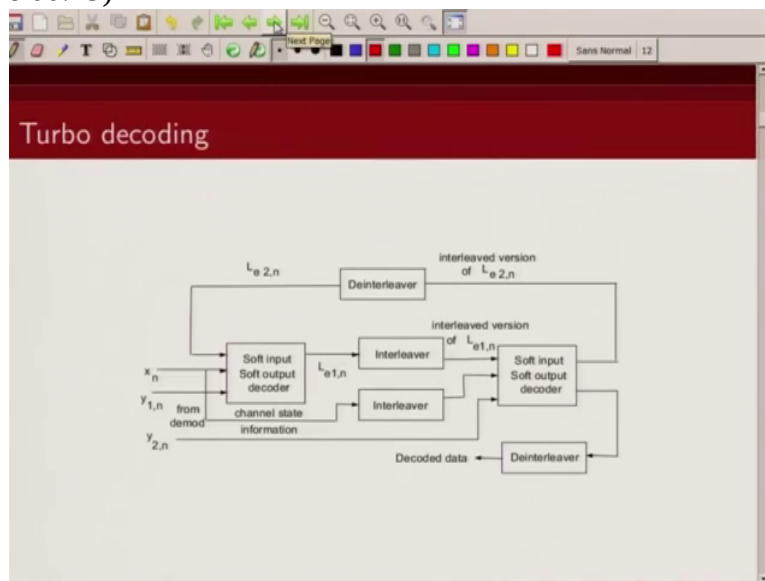
review the metric that are updated in B C G R algorithm and how they are implemented in the log domain. And then we will talk about turbo decoding.

(Refer Slide Time 00:42)



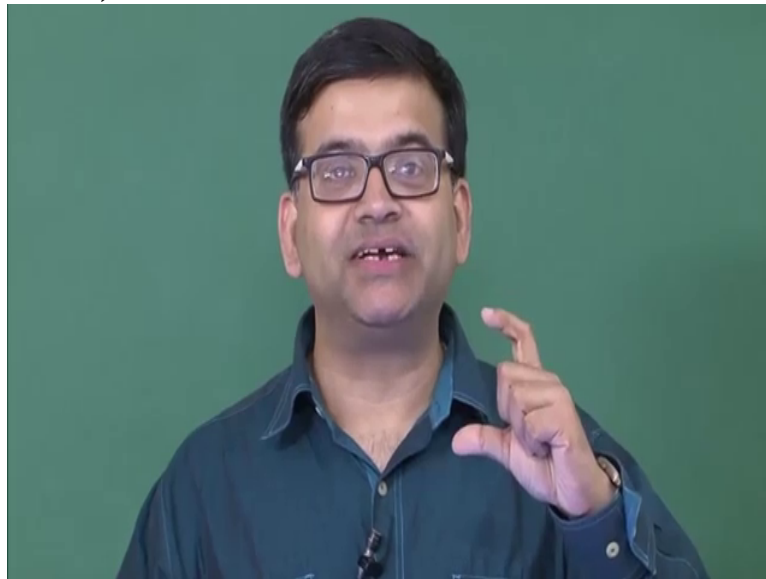
We will take an example of a rate one third code.

(Refer Slide Time 00:49)



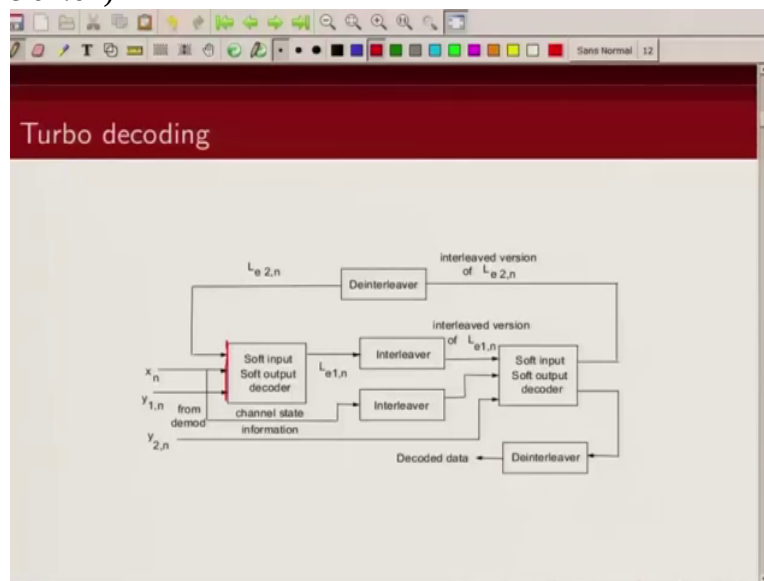
So this is a block diagram of turbo decoder. As you can see, the main blocks are, recall a turbo code consists of parallel concatenation

(Refer Slide Time 01:01)



of two recursive convolutional encoder. So here we have

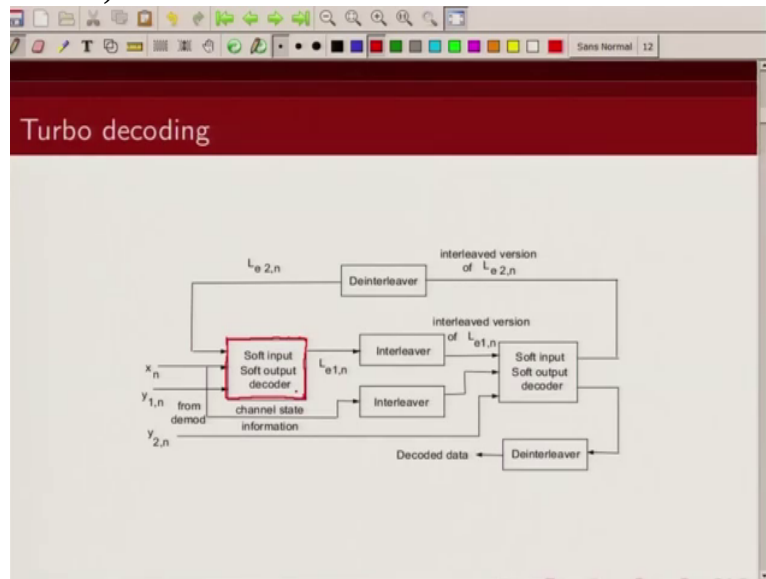
(Refer Slide Time 01:07)



2 decoders, it is an iterative decoding process. We have 2 decoders corresponding to

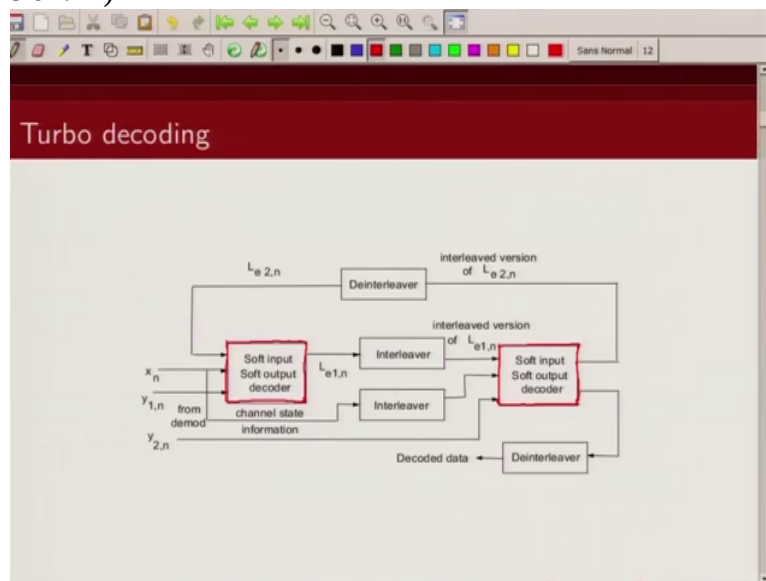


(Refer Slide Time 01:13)



the 2 encoders. So this is one decoder, this is second decoder.

(Refer Slide Time 01:24)



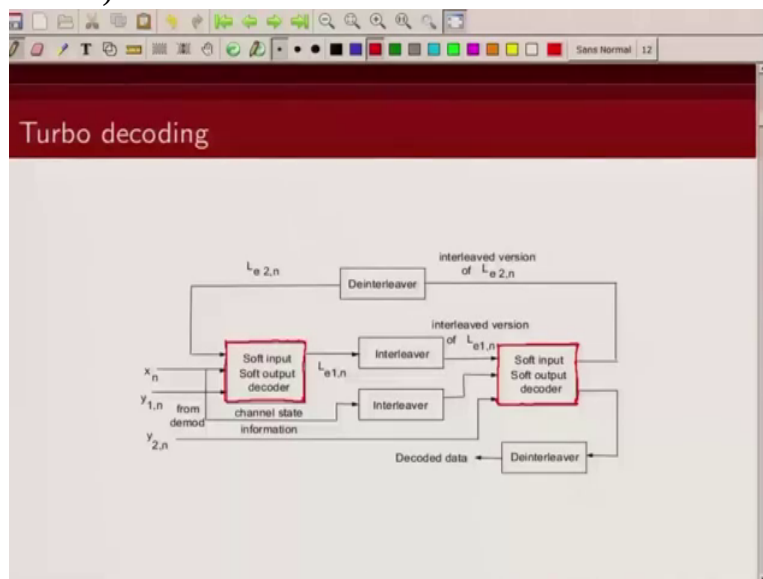
Now note I have written soft inputs soft output decoder. Now what do I mean by soft inputs soft output decoder? So the input that this decoder receives are the real

(Refer Slide Time 01:38)



channel received values. These are not quantized to 0s or 1s, as opposed to getting 0s or 1s in case of a hard quantized decoder here we are getting

(Refer Slide Time 01:50)



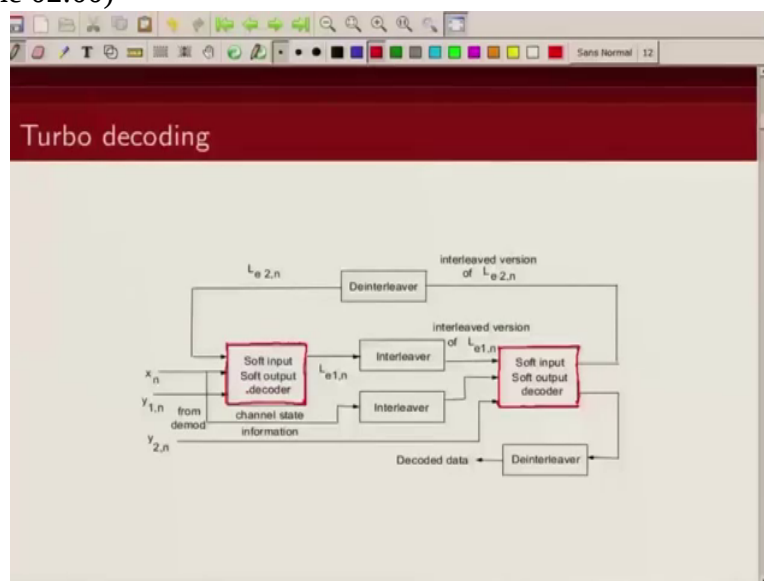
the received, noisy received sequence from the, directly from the channel. And that's what we

(Refer Slide Time 01:56)



are calling as soft input bits. Now the output is

(Refer Slide Time 02:00)



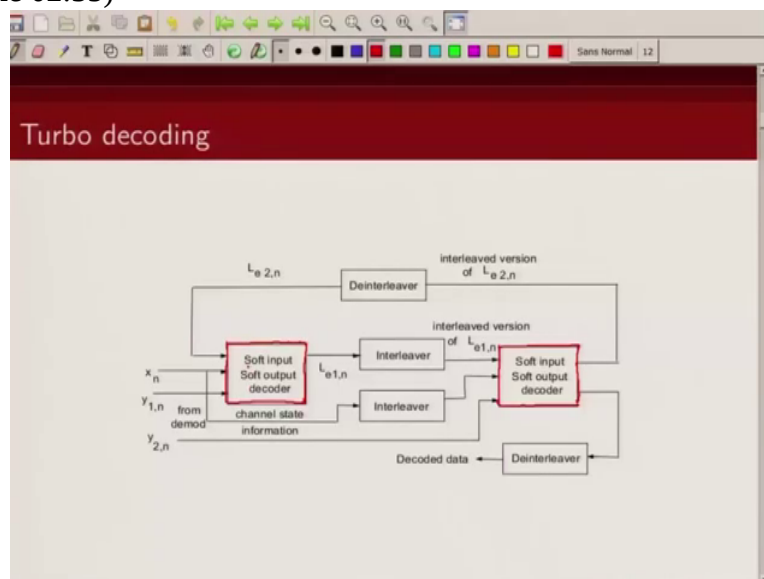
also soft. What do we mean by soft output? Now the decoder will output not only the decision about whether the bit it thinks is 1 or 0 but it will also give

(Refer Slide Time 02:13)



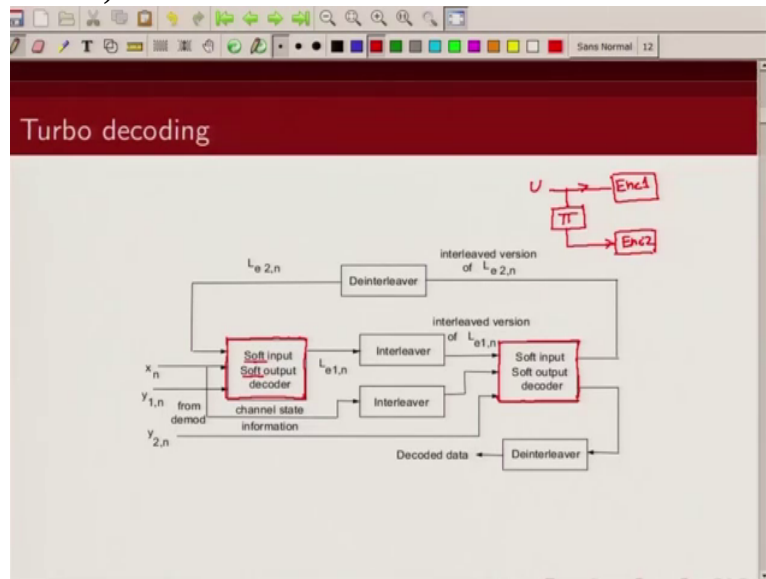
us some information about with what probability it thinks the bit is 0 or 1. So not only we are getting information about the decision of whether the bit is 0 or 1 but we are also getting information about how likely the bit is going to be 0 or 1. That's why

(Refer Slide Time 02:35)



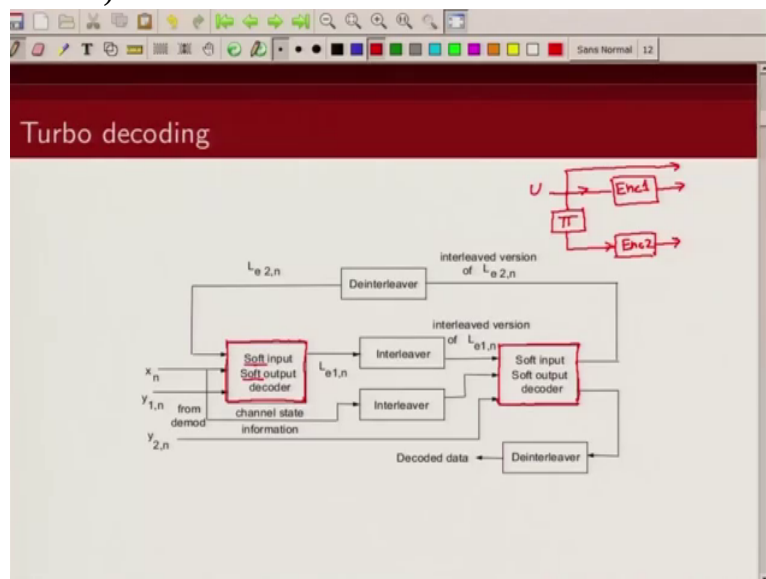
the input here is also soft and the output is also soft as opposed to hard decoder where the output would have been just 0s and 1s. So if you look at the decoder structure, now recall our encoder diagram for the turbo code. What we had was we had one encoder, right? And this was your information sequence. This information sequence was permuted using an interleaver. I am denoting the interleaver by  $\pi$ . And this interleaved signal was sent to another encoder and the

(Refer Slide Time 03:22)



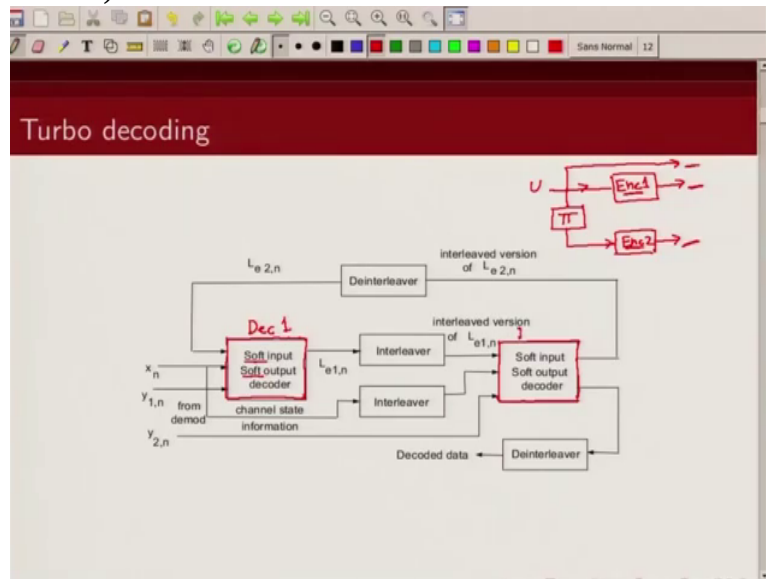
3 outputs were, first was the this information bits, second was the parity bit coming out from the first encoder. And the third

(Refer Slide Time 03:34)



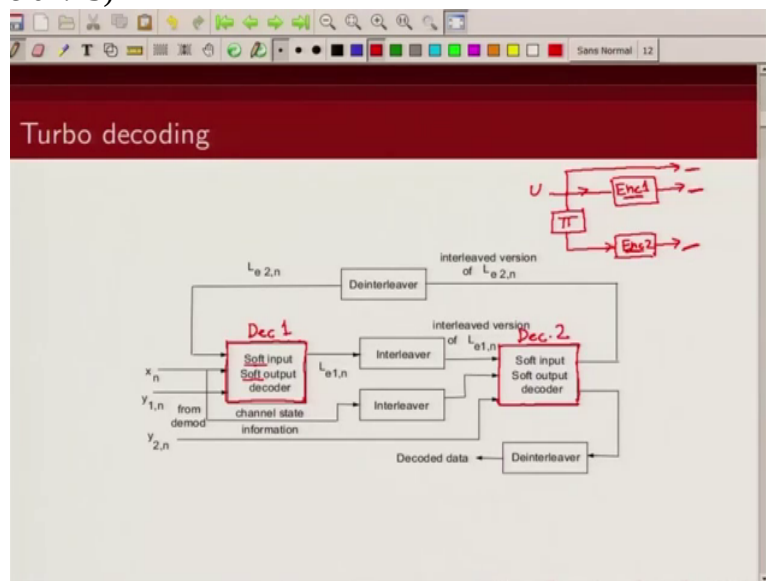
is the parity bit coming out of the second encoder. So these were the 3 outputs of a turbo decoder. Now after these bits pass through the channel, what you are going to receive is the noisy version of the information sequence, noisy bit of this parity bit and the noisy version of the second parity bit. Now as I said we are using 2 decoders. One decoder to decode this convolutional encoder, the second decoder to decode this convolutional encoder. So this was my decoder 1 if I want to call it and this was my

(Refer Slide Time 04:21)



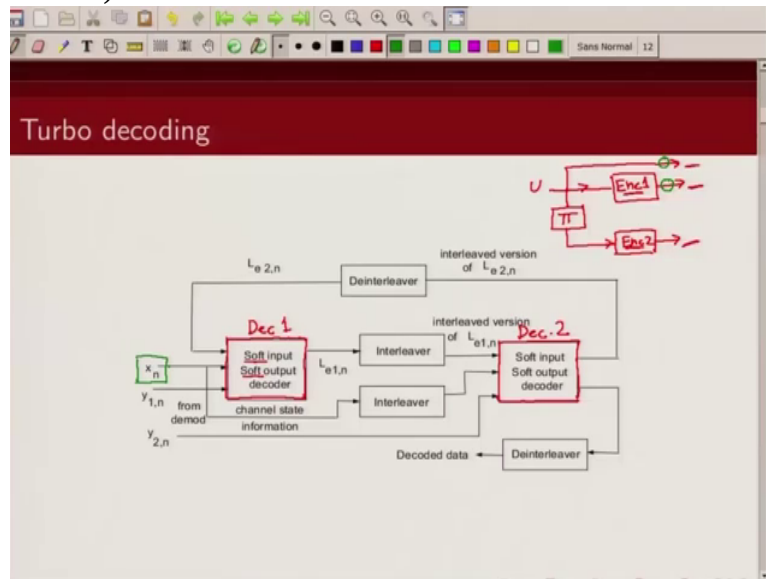
decoder 2.

(Refer Slide Time 04:23)



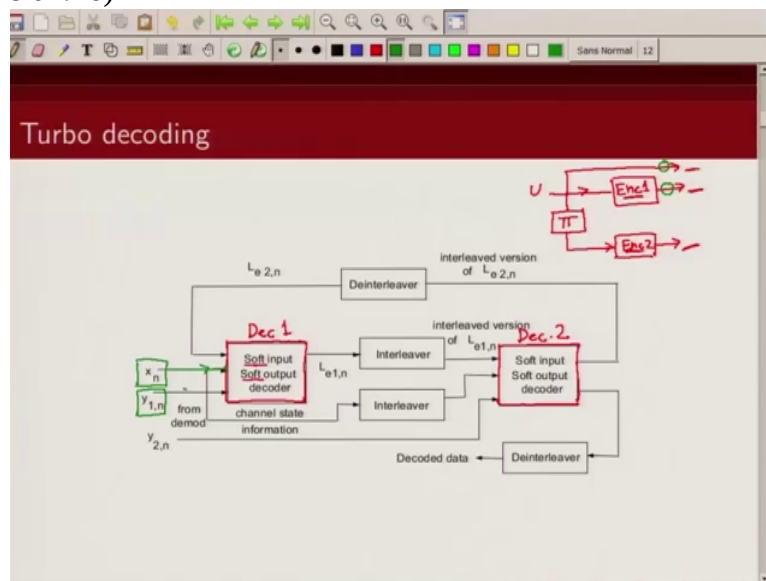
So what are the inputs of decoder 1? Now decoder 1, I should be sending in the received information bits and the received parity bits. So that's what I am sending here. You see here I have written  $x_n$  is actually my

(Refer Slide Time 04:39)



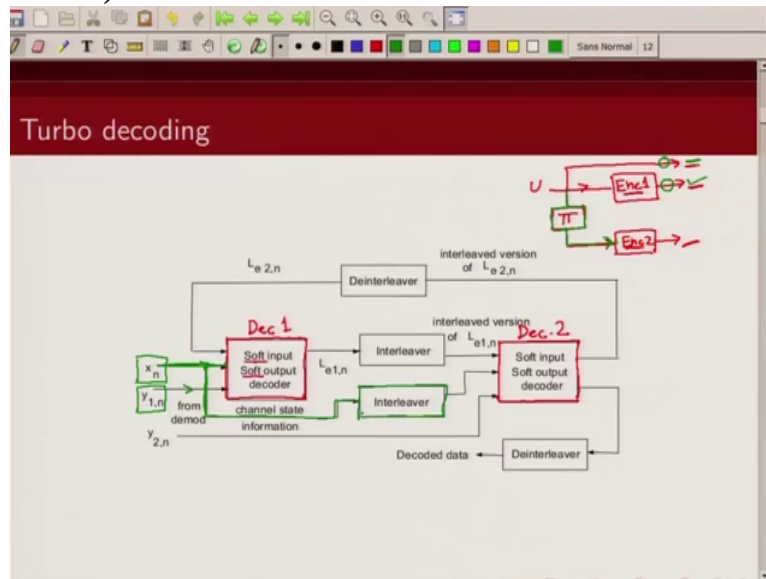
received information bit and  $y_{1,n}$  is

(Refer Slide Time 04:46)



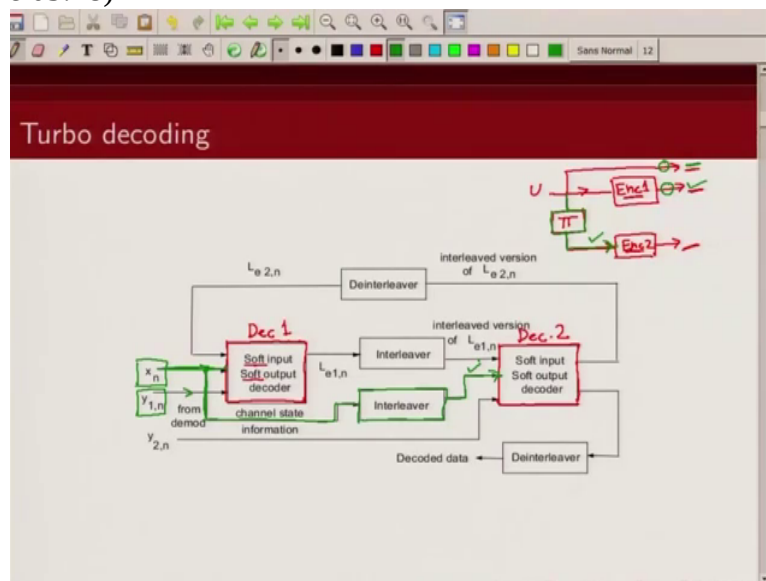
my received parity bit corresponding to the encoder 1. So these are the 2 input to the decoder 1. Now what are the inputs to the decoder 2? Decoder 2 is the interleaved version of information that has been coming to encoder 1. Now here, I am receiving information bit directly so the information, received information bit that I will feed to the decoder 2, because this information bit is getting interleaved before being to sent to encoder 1. So 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 my information sequence, I am going to interleave this

(Refer Slide Time 05:39)



so that the order of information bits that is coming here and what is being fed

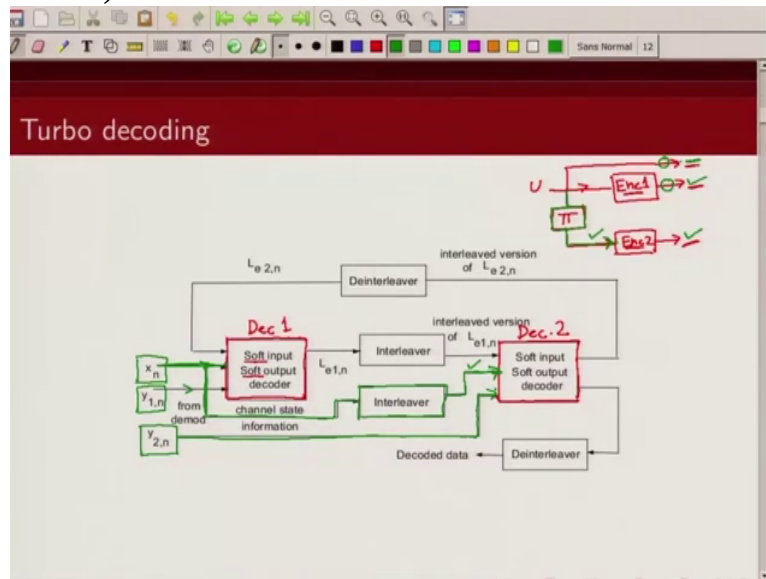
(Refer Slide Time 05:48)



to decoder 2 is same. So we do not send interleaved 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 the interleaved version of the received information sequence. Now what is the second input to the decoder, this received parity bit and that I am denoting by  $y_{2,n}$ , fine? So these are the 2 inputs which I am receiving

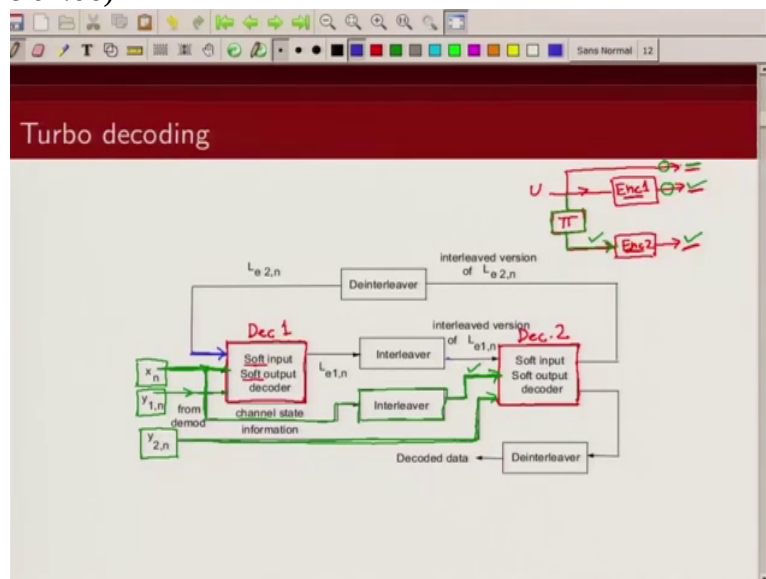


(Refer Slide Time 06:33)



from the channel, which I am feeding to decoder. So the input to decoder 1 are these 2 inputs, and input to the decoder 2 is interleaved version of information sequence and this particular parity bit. So these are the two inputs from the decoder to the channel. What is this input to the decoder? There is a third input to this decoder which is this one and

(Refer Slide Time 07:06)



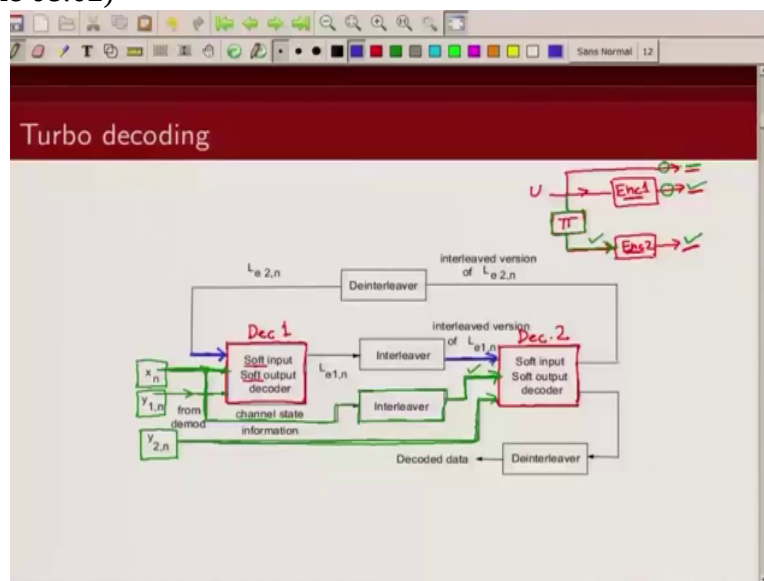
this one. And what is this input? This is a priori knowledge about the information sequence. What is the a priori knowledge of the information sequence that is being fed here as third input? So this third input that you see to decoder 1 is the a priori knowledge about the information sequence. Now how do get the a priori knowledge? Now initially when we start decoding we do not have any a priori knowledge about the information bit. So we could assume that

(Refer Slide Time 07:45)



the information bit is equally likely to be zero or 1. So the likelihood of the information being 0 or 1, that's 1. It's equally likely whether the bit is 0 or 1. So that would be our

(Refer Slide Time 08:02)



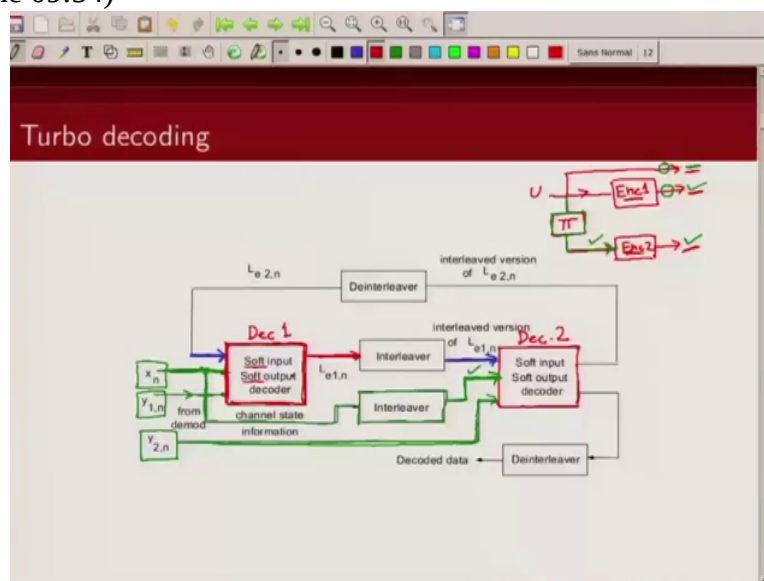
initial a priori information. Now once we feed these 3 inputs to the decoder 1, now let's pay attention to outputs of decoder 1. So decoder 1 is giving us two information. One what we call extrinsic information, now this is information which decoder computes on the Trellis structure of the convolutional code and this extrinsic information is passed to the decoder 1, and this input is fed as a priori knowledge to the second decoder. So you can think of it like this. There are 2 decoders

(Refer Slide Time 08:57)



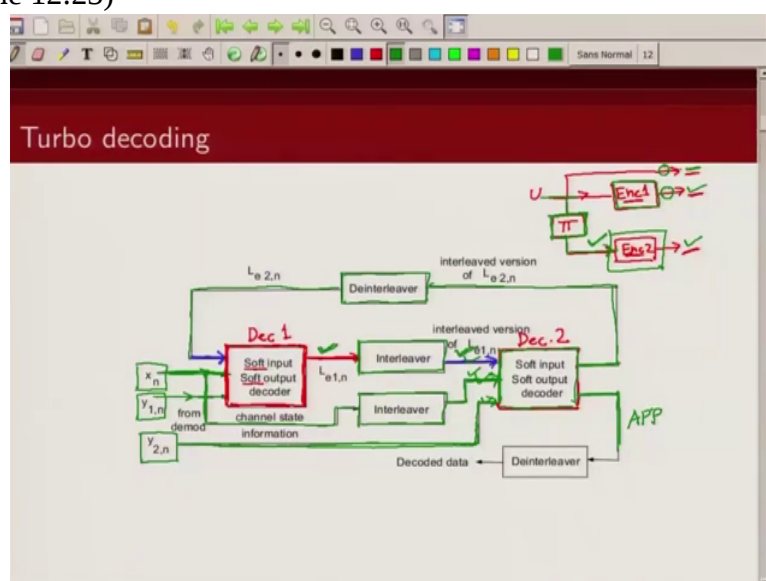
who are working independently but one decoder, once it decodes, once it decodes information sequence it passes some information to the other decoder saying hey I think information bit is likely to be zero with this probability and information bit is 1 with this probability. So the other decoder will take that as an input and then recomputes its probability and then it will again compute some new probabilities of bit being 0 and 1 and it will pass that information back to the first decoder saying no I think it is likely to be 0 with this probability and likely to be 1 with this probability. And this information exchange you know happens in a regular fashion in this until they converge to particular decision. We may stop iteration after fixed number of iterations or we could use some sort of a stopping criteria to do

(Refer Slide Time 09:54)



that. So the third input is the a priori input. The other two inputs are the inputs received from the channel. Now what this decoder 1 does, 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? Note that the order of information bit here and the order of information bit here, now the information bit that has been fed to the second encoder which is this is interleaved version of the information bit that goes to encoder 1, so the estimates that encoder gives about information bits that is being interleaved and sent to decoder 1. This is to maintain the same order of information sequence as being received by decoder 2. So, because information sequence here is interleaved version of information sequence here, so we are going to use this interleaved version. Now similarly this decoder 2, what does this decoder 2 does? It takes these 2 channel values and it takes a priori values information which it has received from decoder 1. And then it will try to decode this code and it will form some estimate, some expensing information and that information is fed back to first decoder and note there is a deinterleaver in between because the order of information sequence is deinterleaved version of information that is being fed to encoder 2. So this deinterleaver is done so that the order of information estimates that we are feeding to encoder 1 is in same order as these other inputs are. So this is an iterator process which goes on and after some fixed number of iterations 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's say from the second decoder so this is my a posteriori probability.

(Refer Slide Time 12:23)



So I can take decision, let's say from the second decoder so this is my a posteriori probability. This information, remember the information sequence ordering at encoder 2 is interleaved

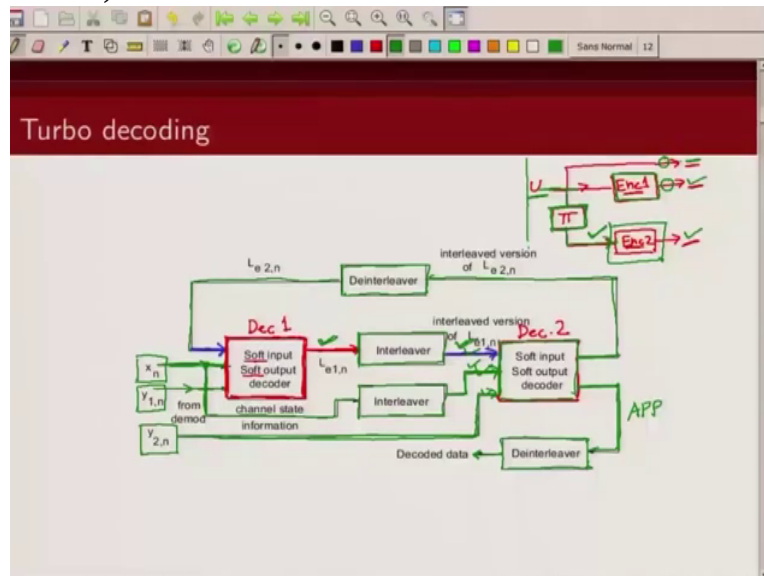
version of the information sequence of encoder 1. So if you want to know the order of information sequence here, we need to de-interleave this data and here we will take a hard decision and take the decision whether the bits are 0 or 1. So you can see it is an iterative

(Refer Slide Time 12:54)



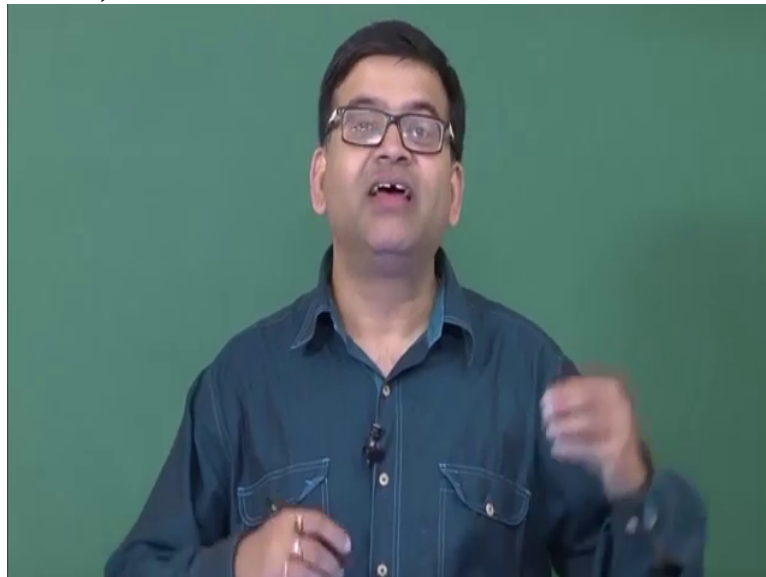
decoding algorithm where instead of decoding this whole

(Refer Slide Time 12:58)



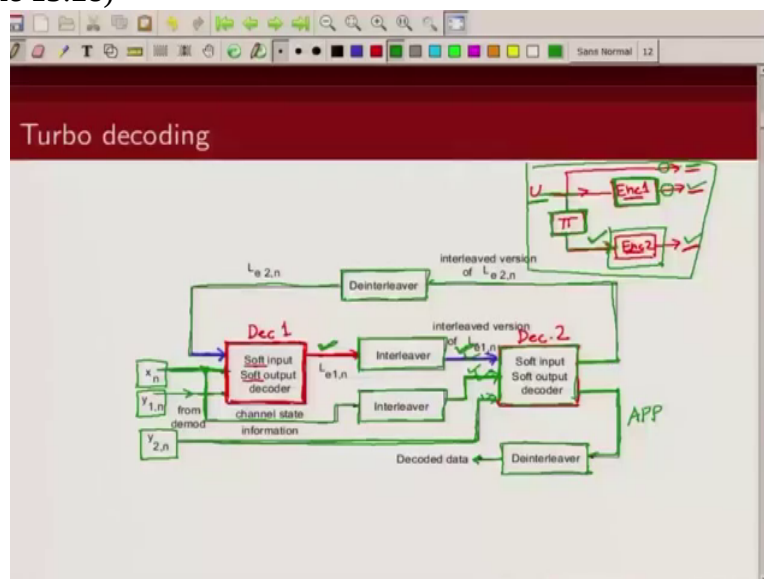
code at one go, what we are trying to do is we are trying to decode first encoder 1 and then encoder 1 is passing some information to encoder 2. Encoder 2 receives some information from the channel

(Refer Slide Time 13:13)



and it receives some information from encoder 1. Using this it tries to find some opinion about information bits which it passes it on to decoder 1 and decoder 1 uses that information. So one iteration is when

(Refer Slide Time 13:28)



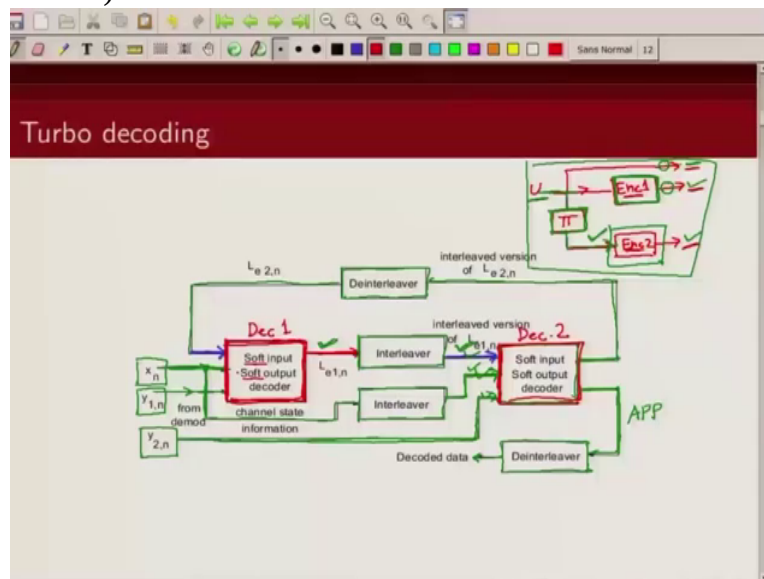
this decoder 1 has finished decoding and decoding 2 has also finished decoding. So that's my one full iteration. Now what is the algorithm which is used inside of this? I said soft input and soft output decoder. I said input is soft, output is soft but what sort of algorithm we can use? Now we can use any algorithm which can take

(Refer Slide Time 13:58)



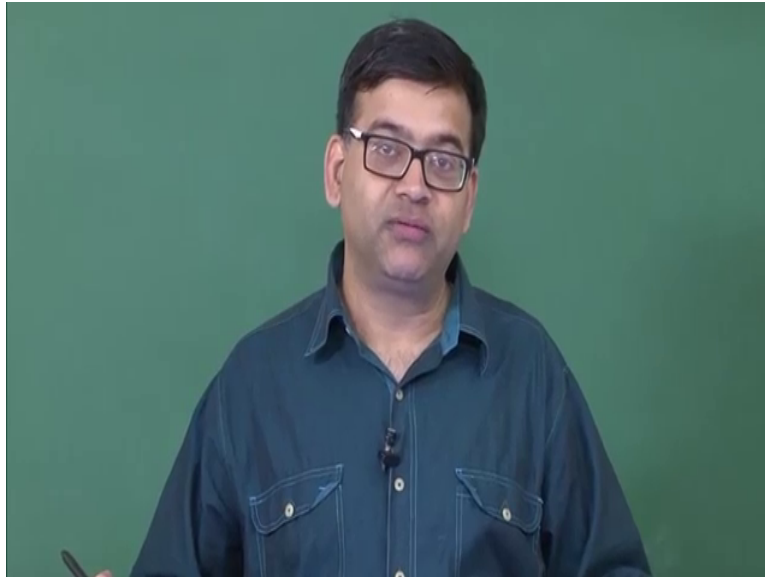
soft inputs and which can give soft outputs. And one of the most commonly used algorithm is our B C J R algorithm. Now in lecture 5 we have talked about decoding of B C J R algorithm for convolutional code so this is precisely

(Refer Slide Time 14:16)



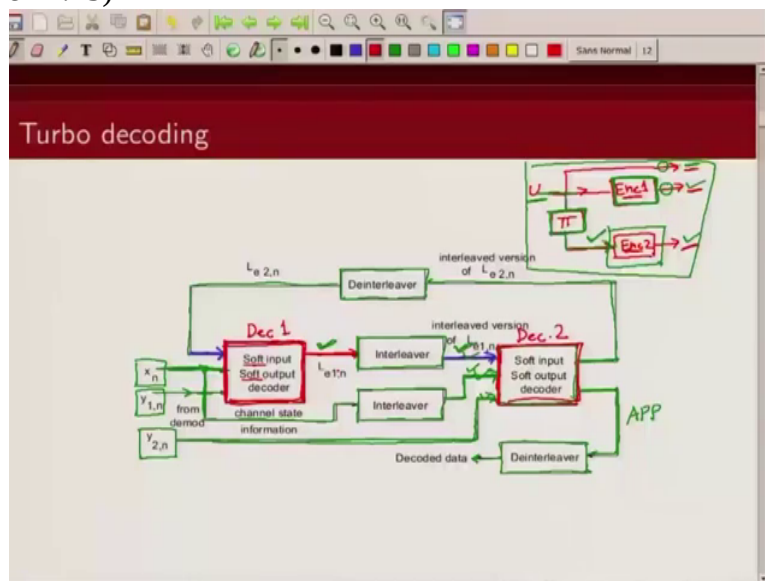
what these two decoders that you see here they are going to use.

(Refer Slide Time 14:21)



They are going to use B C J R algorithm

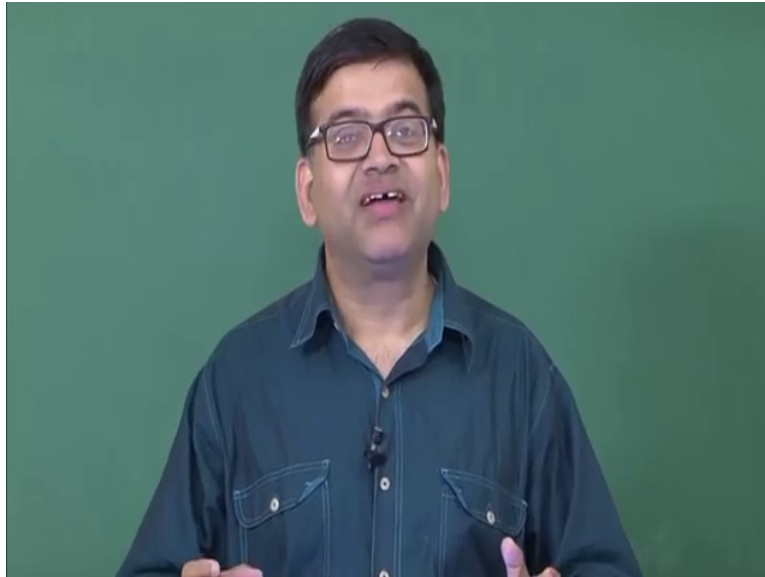
(Refer Slide Time 14:25)



and we are going to slightly modify this algorithm to get this extrinsic information from the decoder and that we will show in subsequent slide. So this is



(Refer Slide Time 14:41)



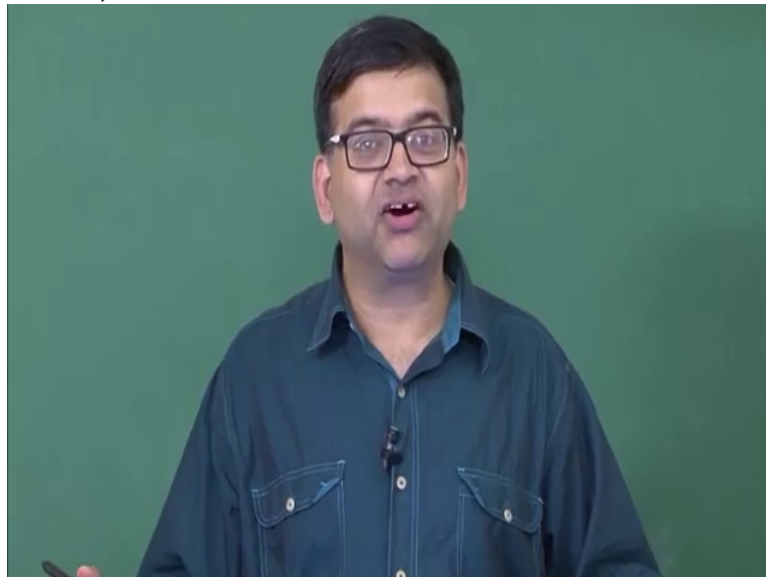
the basic block diagram of your turbo decoder. You have to remember this and again I repeat each of these decoders are independently working uh in the sense there is channel input which is being fed back and there is some a priori information which is coming from other decoder which is being fed to these decoders and these decoder take these three inputs and they compute these 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. So

(Refer Slide Time 15:27)

A screenshot of a presentation slide. The slide has a dark red header with the text "BCJR Algorithm" in white. Below the header, there is a bullet point: "Define  $\max^*(\cdot)$  function:". Underneath this, two mathematical equations are displayed:
$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$$
$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z].$$
The slide is shown within a window that has a standard toolbar at the top.

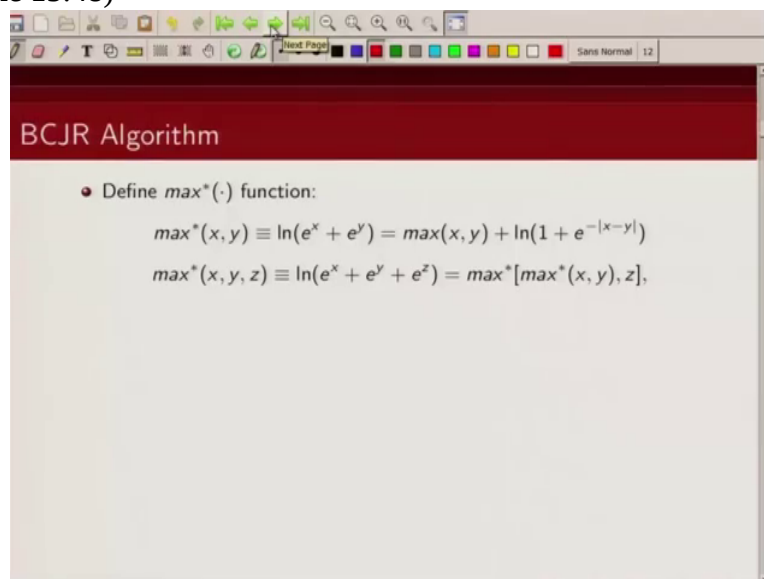
let us just review B C J R algorithm

(Refer Slide Time 15:30)



very quickly. Now we have already derived the expressions for the channel metric, the metric that we need to compute in B C J R algorithm. So we are just going to directly write the expression for B C J R algorithm. Now I just want to

(Refer Slide Time 15:48)



introduce an operator which I call max star operator. Now what is a max star operator? max star operator of x and y, it is basically defined as log of e to power x plus e to power y. Now this log of e to power x plus e to power y, this can be written as log of let's say e to power x into 1 plus e to y minus x. You can write it this way,

(Refer Slide Time 16:31)

BCJR Algorithm

- Define  $\max^*(\cdot)$  function:  $\ln(e^x(1 + e^{y-x}))$

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

correct? Now this can be written as log of e to power x plus log of 1 plus e to power y minus x.

(Refer Slide Time 16:46)

BCJR Algorithm

- Define  $\max^*(\cdot)$  function:  $\ln(e^x(1 + e^{y-x})) = \ln e^x + \ln(1 + e^{y-x})$

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

Or I can also write the same thing as log of e to power y, 1 plus e to power x minus y log of e to power y plus log of 1 plus e to power x minus y.

(Refer Slide Time 17:09)

BCJR Algorithm

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

Handwritten notes in red ink:  
 $\ln(e^x(1 + e^{y-x})) = \ln e^x + \ln(1 + e^{y-x})$   
 $\ln(e^y(1 + e^{x-y})) = \ln e^y + \ln(1 + e^{x-y})$

And what is this log of e to power x, that's just x. So this can be written as, so this can be just written as x plus this and y plus this.

(Refer Slide Time 17:26)

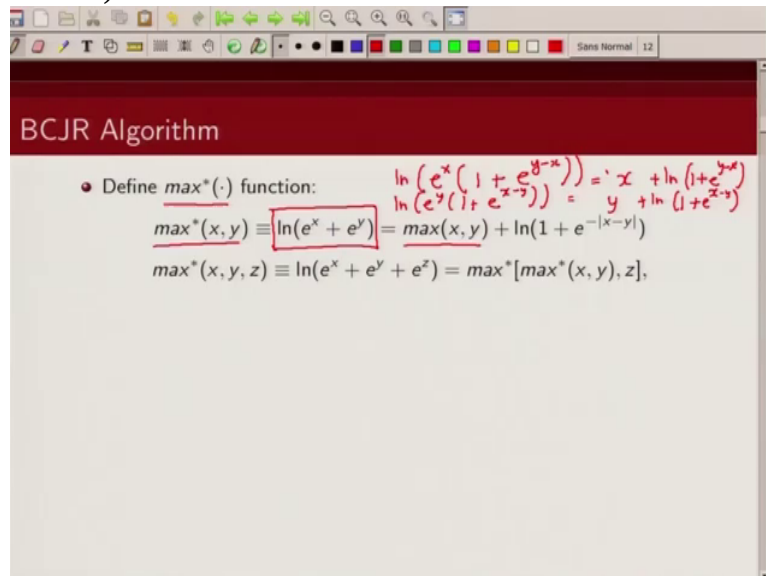
BCJR Algorithm

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

Handwritten notes in red ink:  
 $\ln(e^x(1 + e^{y-x})) = x + \ln(1 + e^{y-x})$   
 $\ln(e^y(1 + e^{x-y})) = y + \ln(1 + e^{x-y})$

Or I can write this max star x y as maximum of x and y plus natural log of 1 plus e raised to power minus absolute value of x minus y, right? So, so whenever I have to take log of terms of this form,

(Refer Slide Time 17:49)



BCJR Algorithm

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

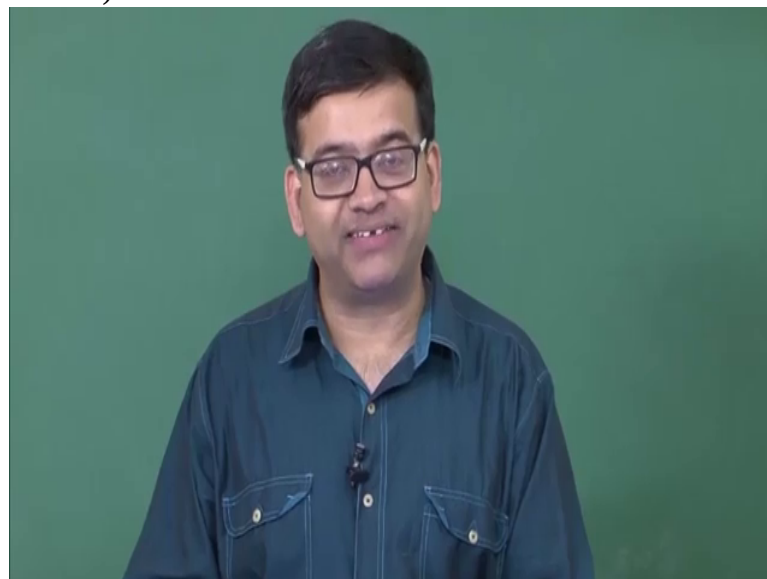
$$\max^*(x, y) \equiv \ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{-|x-y|})$$
$$\max^*(x, y, z) \equiv \ln(e^x + e^y + e^z) = \max^*[\max^*(x, y), z],$$

Handwritten notes in red:

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

I can actually implement it simply like maximum of these two operator x

(Refer Slide Time 17:55)



and y plus some

(Refer Slide Time 17:58)

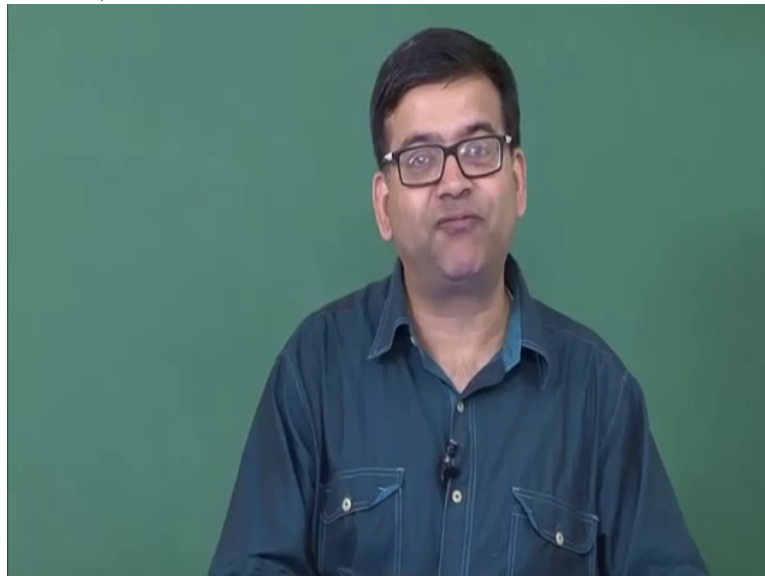
BCJR Algorithm

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

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

correction term which can be implemented with a table lookup kind of thing. And this operator can be extended

(Refer Slide Time 18:07)



for more than 2 operations, so

(Refer Slide Time 18:11)

**BCJR Algorithm**

- Define  $\max^*(\cdot)$  function:
 
$$\ln(e^x(1+e^{y-x})) = x + \ln(1+e^{y-x})$$

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

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

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

if you want to find max star of x, y and z then I can write it as natural log of e x plus e y plus z. And I can do max star of x, y and z and then I take the max star of, max star of x, y and z. So I can iteratively apply this max star operator to compute quantities of this form,

(Refer Slide Time 18:38)

**BCJR Algorithm**

- Define  $\max^*(\cdot)$  function:
 
$$\ln(e^x(1+e^{y-x})) = x + \ln(1+e^{y-x})$$

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

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

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

Ok which is log of summation terms; now where do we encounter log of summation terms. I will come to that.

(Refer Slide Time 18:49)

**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:  
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$

Now this branch metric, this we have derived in one of the lectures, is given by this

(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:  
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$

expression. This is a priori information, this is information bit, this depends on channel s n r, this is received sequence, transmitted codeword, so this you already are familiar with, because we have derived this expression before. Now recall



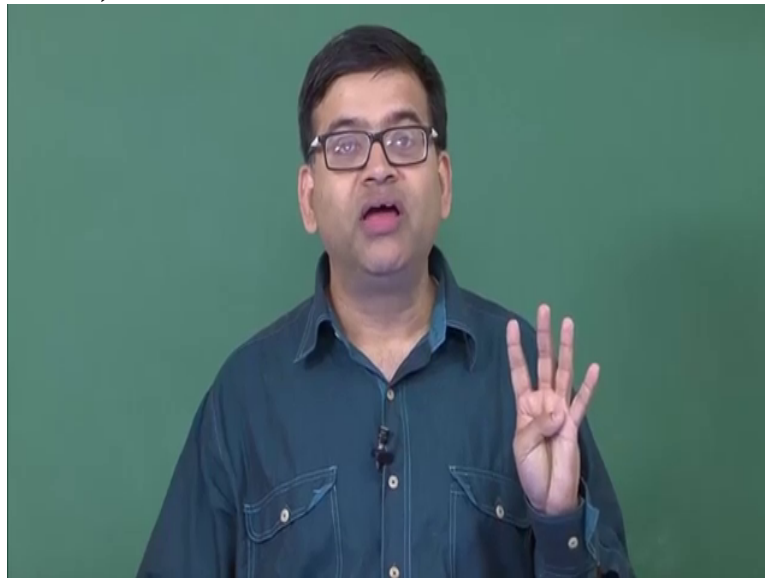
(Refer Slide Time 19:16)

**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:  
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:  
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$

there were three quantities we need to

(Refer Slide Time 19:20)



evaluate when we need to apply B C J R algorithm. And what were those quantities? One was these alphas which was the forward recursion. Second was the betas which was the backward recursion. And then we had the channel metric, branch metric, right, gammas. Now

(Refer Slide Time 19:44)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$

if you recall what was alphas, alpha, so if you just draw simple Trellis diagram, you just draw simple Trellis diagram, two state Trellis and this is alpha at time let's say  $l-1$ , this is alpha at time  $l$ . Let's call it a state zero. Let's call it state 1. Then what is, let's say I want to compute alpha at time  $l$  for state 0. What is it equal to? Recall this was equal to  $(\cdot)$  so there are 2 states. There are 2 branches which are terminating here. One is this one, other is this one. So the alpha 0 will be alpha  $l-1$  zero times branch metric corresponding to this which will be gamma 0 0 plus alpha  $l-1$ , 1 times this branch metric which is gamma 1 0. So if you

(Refer Slide Time 20:53)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- 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 notes and diagram:

- A small trellis diagram with two states at time  $i-1$  and two states at time  $i$ . Red circles and arrows indicate transitions.
- Equation:  $\alpha_i(0) = \alpha_{i-1}(0) \gamma_i(0,0) + \alpha_{i-1}(1) \gamma_i(1,0)$

recall we had terms of the form summation alpha  $l-1$  times some gamma. So those were our terms. Now if you take log of that, so these were our alphas. We are defining a new

operator alpha star, that's a log of these alphas. So what's going to happen here? You have log of summation terms, right. So if you can think of it as, so here you terms of the form like this, a 1 b 1 plus a 2 b 2. And we are taking log of this. We can

(Refer Slide Time 21:39)

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

$\ln(a_1 b_1 + a_2 b_2) = \alpha_{x-1}(0) \gamma_x(0,0) + \alpha_{x-1}(1) \gamma_x(1,0)$

*Handwritten notes:*  $\ln(a_1 b_1 + a_2 b_2) = \alpha_{x-1}(0) \gamma_x(0,0) + \alpha_{x-1}(1) \gamma_x(1,0)$

*Diagram:* A trellis diagram with two nodes at each time step, labeled  $\alpha_{x-1}$  and  $\alpha_x$ .

also write this in terms of let's say e raised to power a 1 dash, e raised to power b 1 dash plus e raised to power a 2 dash, e raised to power b 2 dash,

(Refer Slide Time 21:59)

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

$\ln(e^{a_1} e^{b_1} + e^{a_2} e^{b_2}) = \alpha_{x-1}(0) \gamma_x(0,0) + \alpha_{x-1}(1) \gamma_x(1,0)$

*Handwritten notes:*  $\ln(e^{a_1} e^{b_1} + e^{a_2} e^{b_2}) = \alpha_{x-1}(0) \gamma_x(0,0) + \alpha_{x-1}(1) \gamma_x(1,0)$

*Diagram:* A trellis diagram with two nodes at each time step, labeled  $\alpha_{x-1}$  and  $\alpha_x$ .

right? So if we take log of this, then this will become, because this is like e of e x plus 1, so when take log of these summation terms we get this max star operator. So we get these max star operator and this product term when we take log they will become summation. So this forward metric will become, when we implement it

(Refer Slide Time 22:31)

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

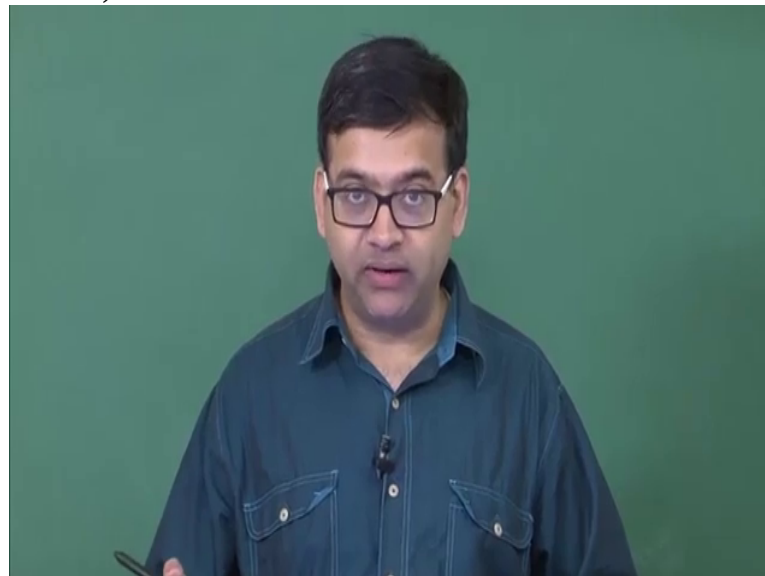
*Handwritten notes:*

$$\ln(a_1 b_1 + a_2 b_2) = \alpha_{i-1}(0) \gamma_i(0,0) + \alpha_{i-1}(1) \gamma_i(1,0)$$

$$\ln(e^{a_1} e^{b_1} + e^{a_2} e^{b_2})$$

in, we take log of this, this will become a max star operator and these two 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 star operator we are summing over all the branches that are terminating at this state and this would be gamma plus alpha star. So this will be the forward metric in the log domain.

(Refer Slide Time 23:10)



Now recall how did we initialize the alphas,

(Refer Slide Time 23:16)

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

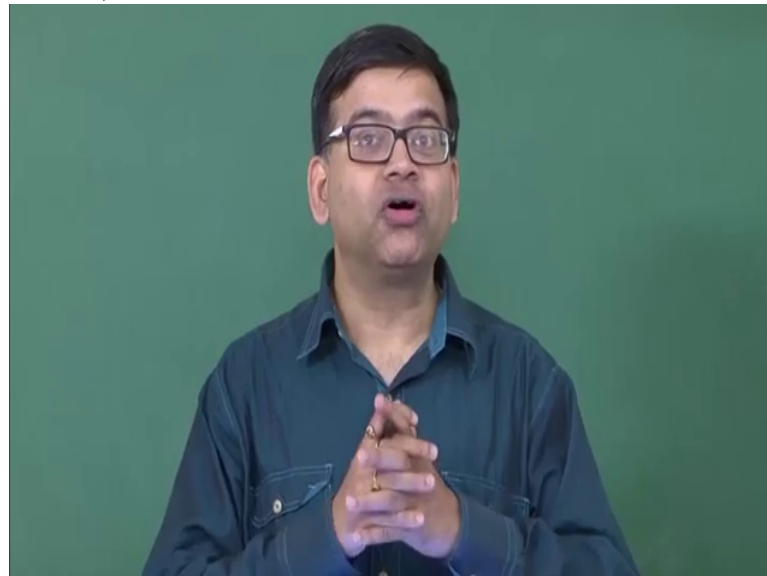
Handwritten notes below the forward metrics equation:

$$\ln(a_1 b_1 + a_2 b_2) = \alpha_{i-1}^*(0) \gamma_i^*(0,0) + \alpha_{i-1}^*(1) \gamma_i^*(1,0)$$

A trellis diagram on the right shows two states at time  $i-1$  (labeled  $\alpha_{i-1}^*$ ) and two states at time  $i$  (labeled  $\alpha_i^*$ ). Red arrows indicate transitions between states.

when we were working in probability domain. We said that the encoder starts at all zero state so

(Refer Slide Time 23:26)



at, at time zero it will be at all zero state. The probability of it being in all zero state at time zero is 1, for all other it is 0. So when we take log of that then, the

(Refer Slide Time 23:41)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:
 
$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0} \end{cases}$$

initialization will become for state,

(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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:
 
$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0} \end{cases}$$

initial state zero, then this will become, log of 1 will become 0. And for all, probability of its being all other states, this will then become

(Refer Slide Time 23:58)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:
 
$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0} \end{cases}$$

minus infinity. Ok so if we are writing our recursion in this fashion using

(Refer Slide Time 24:09)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:
 
$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0} \end{cases}$$

max star operation then the initialization should be done like this. When it is in state zero the initialization zero star zero will be zero and it will be minus infinity for all other cases.

Now if you are wondering why I am switching from probability domain to log domain you can see that we have shown this max star operator can be very easily implemented. Because this is just maximum of

(Refer Slide Time 24:42)

**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:
 
$$\gamma_i^*(s', s) \equiv \ln \gamma_i(s', s) = \frac{u_i L_a(u_i)}{2} + \frac{L_c}{2} \mathbf{r}_i \cdot \mathbf{v}_i$$
- Forward metrics:
 
$$\alpha_{i+1}^*(s) \equiv \ln \alpha_{i+1}(s) = \max_{s' \in \sigma_i} [\gamma_i^*(s', s) + \alpha_i^*(s')]$$
- Forward metrics initialization:
 
$$\alpha_0^*(s) \equiv \ln \alpha_0(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0} \end{cases}$$

x, y plus some correction term. So this can be easily implemented, that's why we are rewriting our

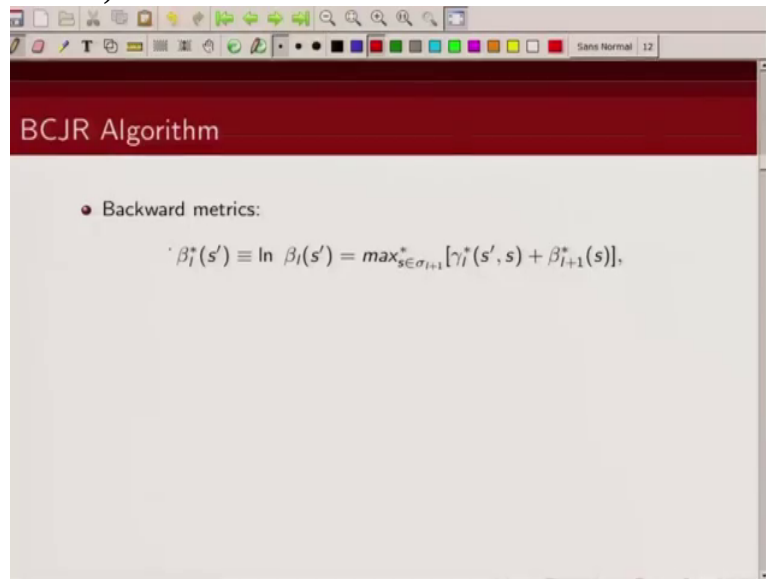
(Refer Slide Time 24:52)



forward recursion, backward recursion, gamma in terms of log domain so that our expression, where we had the summation. Now that's been replaced by max star operator. Wherever we had multiplication that's been now replaced by addition. So following the same logic we can

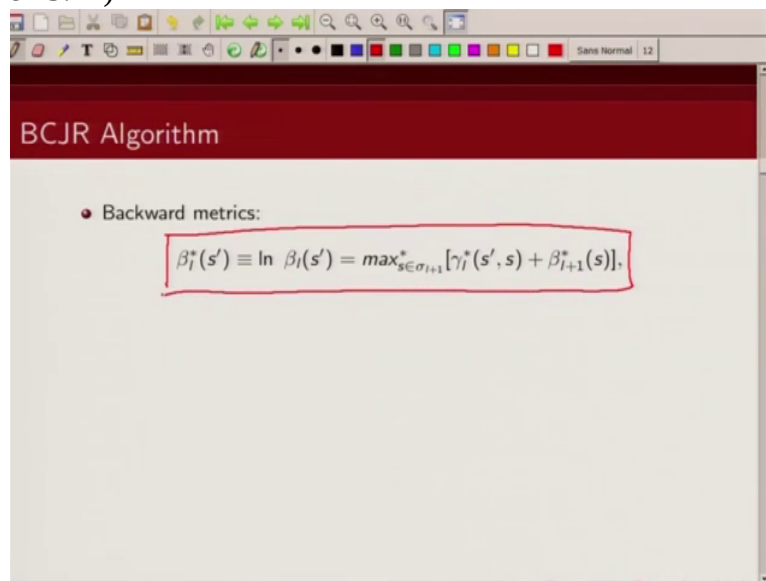


(Refer Slide Time 25:17)



write the backward metric in log domain in this

(Refer Slide Time 25:22)



particular fashion so this max star operator and this is sum of this branch metric in the log domain and the betas in the log domain. And again if, if I, if you can recall how were we computing betas so let's say you had some this thing

(Refer Slide Time 25:44)

BCJR Algorithm

- Backward metrics:

$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$

so if you are interested in computing this thing, computing beta for beta at l 0, you will be beta and this is, let's say this is time l, time l plus 1, beta this is time l, this is beta time l plus 1, beta 0 will be, so these are the two branches that are terminating here. So beta l plus 1 zero times branch metric of this which is gamma l 0 0 plus beta l plus 1, this is stat 0, this is state 1, state 1, beta l 1 times gamma l 0 1. This we have already studied when we did B C J R algorithm.

(Refer Slide Time 26:39)

BCJR Algorithm

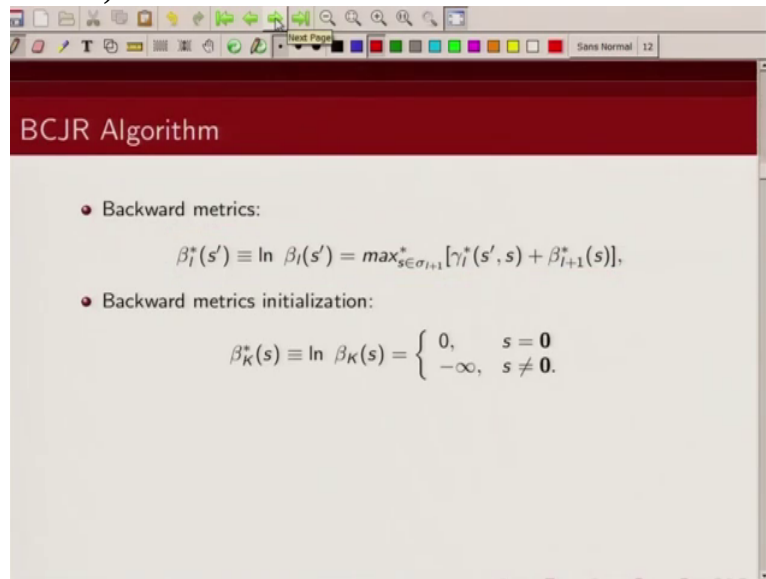
- Backward metrics:

$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$

$\beta_i^*(b) = \beta_{i+1}^*(a)$   
 $\gamma_i^*(a, a)$   
 $+ \beta_{i+1}^*(i)$   
 $\gamma_i^*(a, i)$

We are just rewriting the expression in terms of log so that our addition term here becomes the max star operator and the product term here that you see here becomes addition term. And

(Refer Slide Time 26:56)

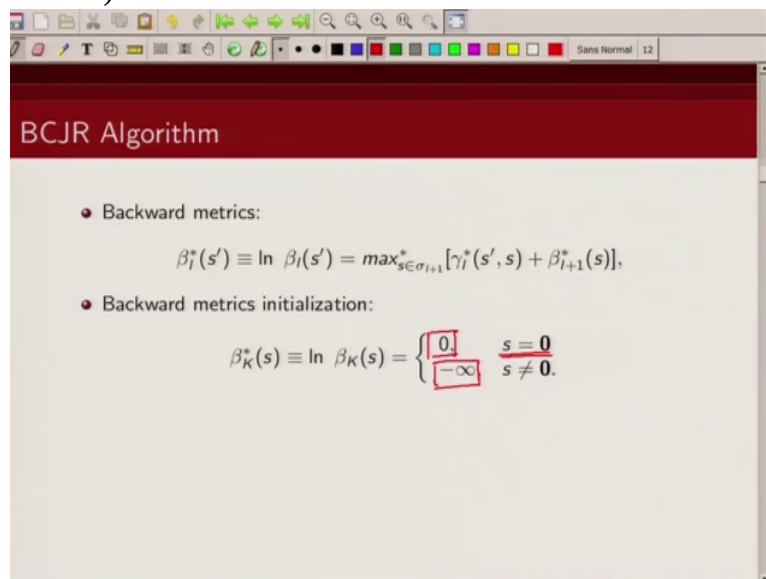


BCJR Algorithm

- Backward metrics:
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- Backward metrics initialization:
$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0}. \end{cases}$$

similar to alpha bit initialization if we are terminating our convolutional encoder then beta k at s equal to 0 will be 0 and for all other state, it will be minus

(Refer Slide Time 27:14)

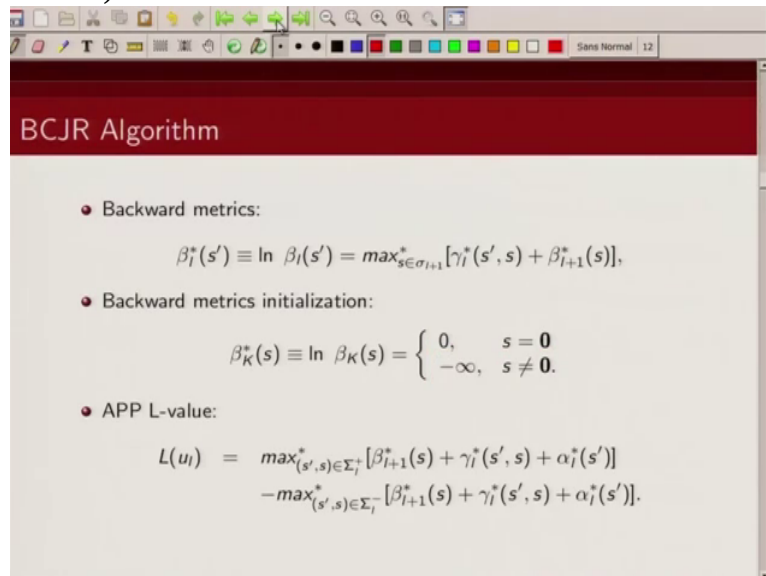


BCJR Algorithm

- Backward metrics:
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- Backward metrics initialization:
$$\beta_K^*(s) \equiv \ln \beta_K(s) = \begin{cases} 0, & s = \mathbf{0} \\ -\infty, & s \neq \mathbf{0}. \end{cases}$$

infinity. And

(Refer Slide Time 27:17)



**BCJR Algorithm**

- Backward metrics:
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
$$L(u_i) = \max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')] - \max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')].$$

of course if we are not terminating it, then its probability of being any state is basically same. And if you recall our a p p log likelihood value this was again summation of product of three terms, alpha at time l, beta at time l plus 1 and gamma so that and there were 2 terms, one term in the numerator corresponding to all those transitions

(Refer Slide Time 27:48)



belonging to information bit being plus 1 and in the denominator we had some terms related to, transitions that belong to information bit being minus 1. So this

(Refer Slide Time 28:02)

**BCJR Algorithm**

- Backward metrics:
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}}^* [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}{-\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}.$$

term that you see here is the

(Refer Slide Time 28:05)

**BCJR Algorithm**

- Backward metrics:
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}}^* [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}{-\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}.$$

term corresponding to the numerator term, again the summation term has been replaced by this max star operator and this product term has been replaced by these

(Refer Slide Time 28:19)

**BCJR Algorithm**

- Backward metrics:
 
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
 
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}{-\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}.$$

addition terms, Ok. And similarly this corresponds to all those transitions where my information bit is plus 1 and then denominator we had this, so we take log of them, so it will become minus

(Refer Slide Time 28:38)

**BCJR Algorithm**

- Backward metrics:
 
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
 
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}{-\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) + \gamma_i^*(s', s) + \alpha_i^*(s')]}.$$

of this. And the denominator corresponds to all those transitions

(Refer Slide Time 28:42)

**BCJR Algorithm**

- Backward metrics:
 
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
 
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}{\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}.$$

which belong to information bit being minus 1 and again the addition term that we had in the probability domain description of B C J R, that's now max star operator and the product terms that we had in the B C J R algorithm

(Refer Slide Time 28:59)

**BCJR Algorithm**

- Backward metrics:
 
$$\beta_i^*(s') \equiv \ln \beta_i(s') = \max_{s \in \sigma_{i+1}} [\gamma_i^*(s', s) + \beta_{i+1}^*(s)],$$
- 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:
 
$$L(u_i) = \frac{\max_{(s', s) \in \Sigma_i^+} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}{\max_{(s', s) \in \Sigma_i^-} [\beta_{i+1}^*(s) \boxplus \gamma_i^*(s', s) \boxplus \alpha_i^*(s')]}.$$

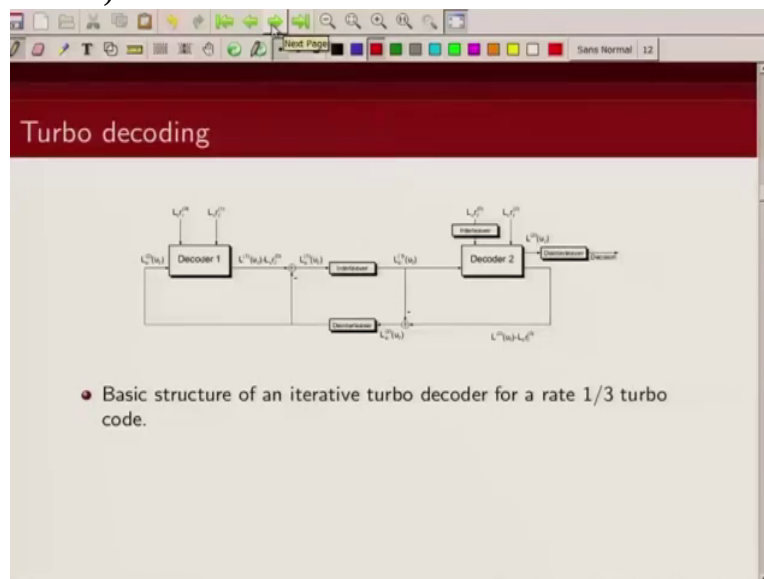
are now addition term. So this is

(Refer Slide Time 29:06)



rewriting the expressions for forward recursion, backward recursion and log likelihood ratio a posteriori probability L value computation for the B C J R algorithm.

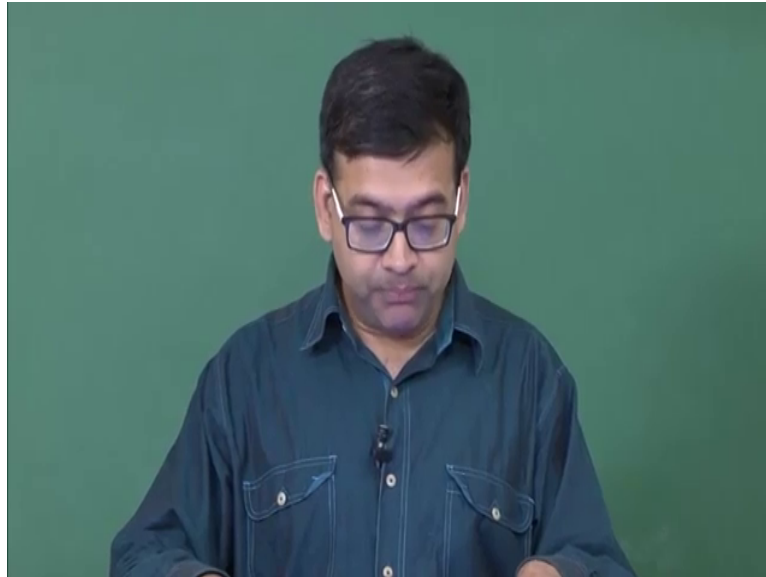
(Refer Slide Time 29:22)



So again we will go back to the decoder diagram that we have shown you before. I am reproducing the decoder for rate one third turbo

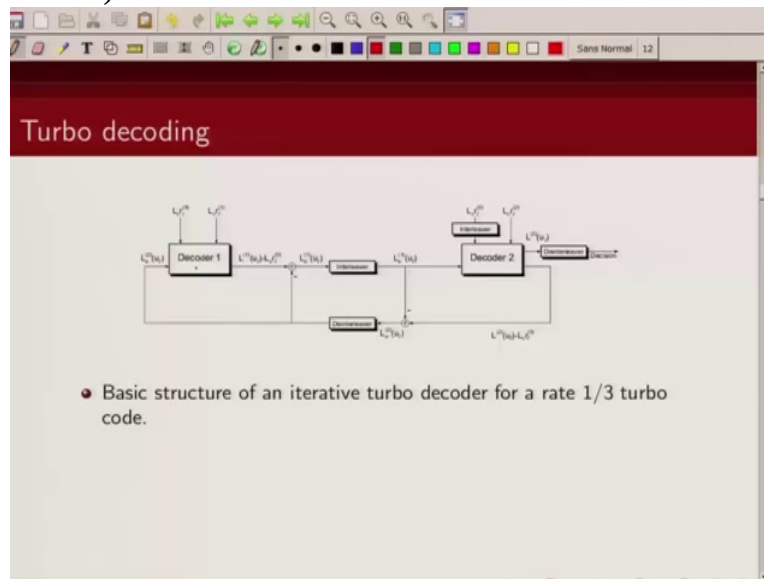


(Refer Slide Time 29:40)



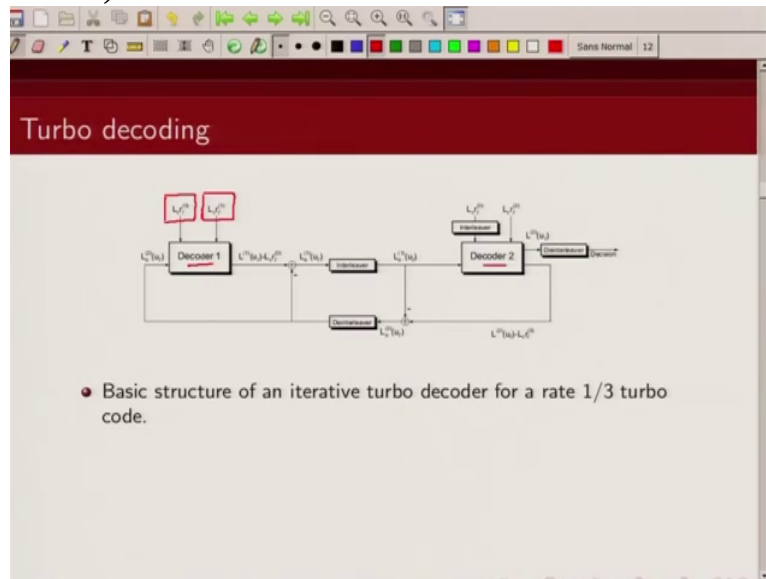
code. This is

(Refer Slide Time 29:42)



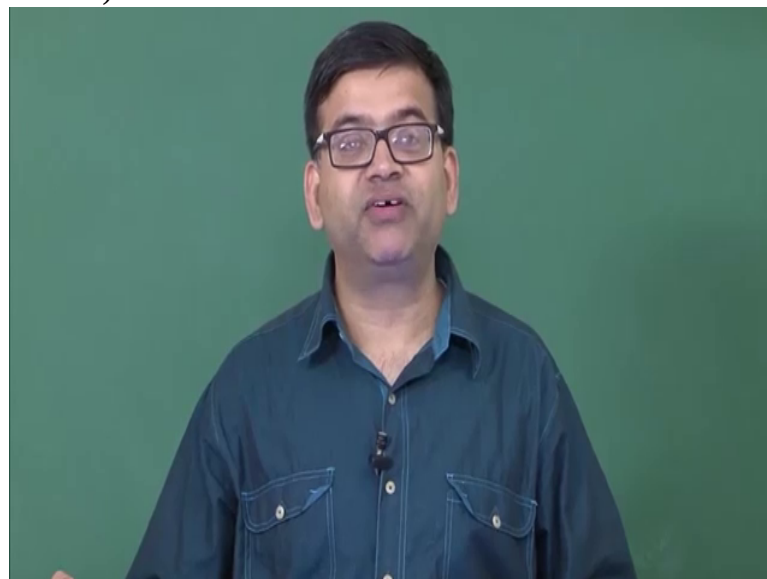
my decoder 1 corresponding to encoder 1. This is my decoder 2. These 2 inputs that you see are my inputs corresponding

(Refer Slide Time 29:55)



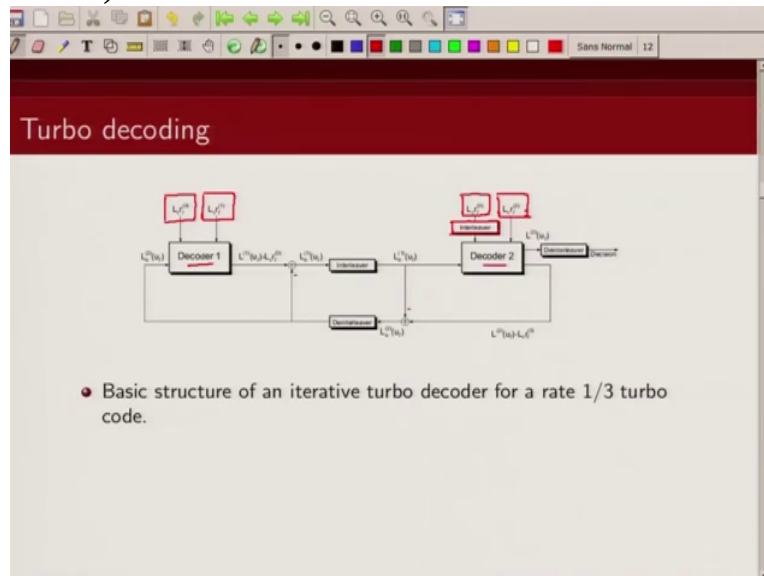
to the received

(Refer Slide Time 29:57)



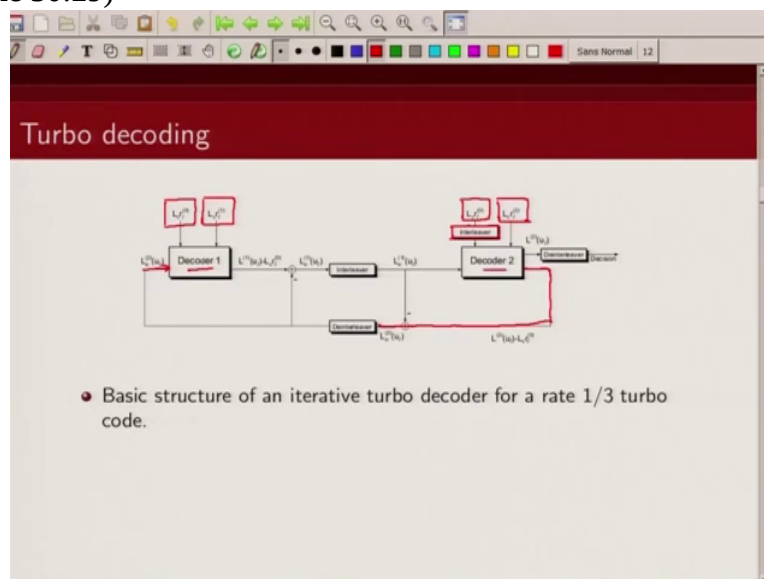
information bit and the corresponding parity bit. This is my information, received information which is interleaved before being sent to decoder 2 and this is the received,

(Refer Slide Time 30:13)



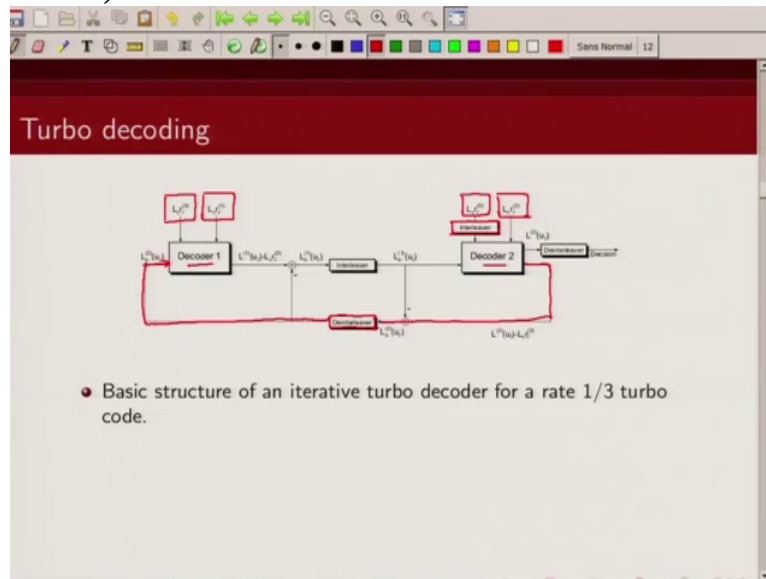
this is the bit corresponding to the received parity, second parity bit. I also said there is a third input which is a a priori value. Note this a priori value is coming from the other decoder and since the order

(Refer Slide Time 30:29)



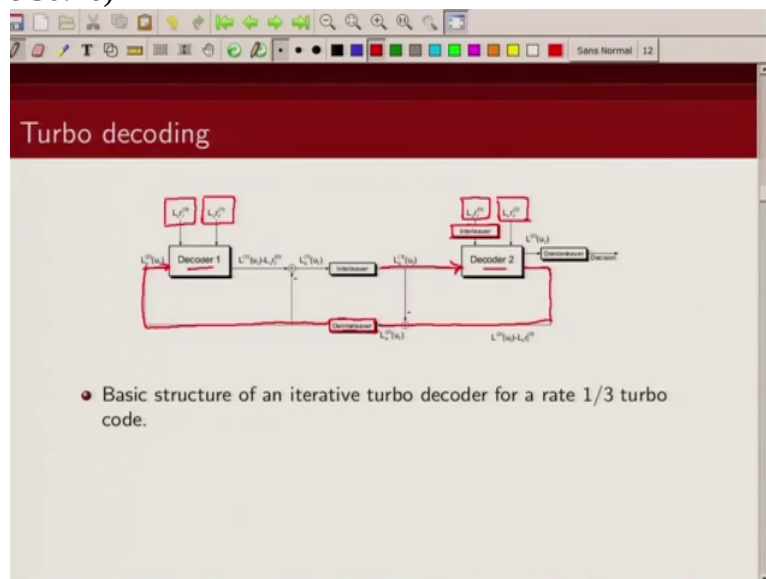
of information is de-interleaved version of the order information bit at second decoder so you de-interleave it and send the information here

(Refer Slide Time 30:41)



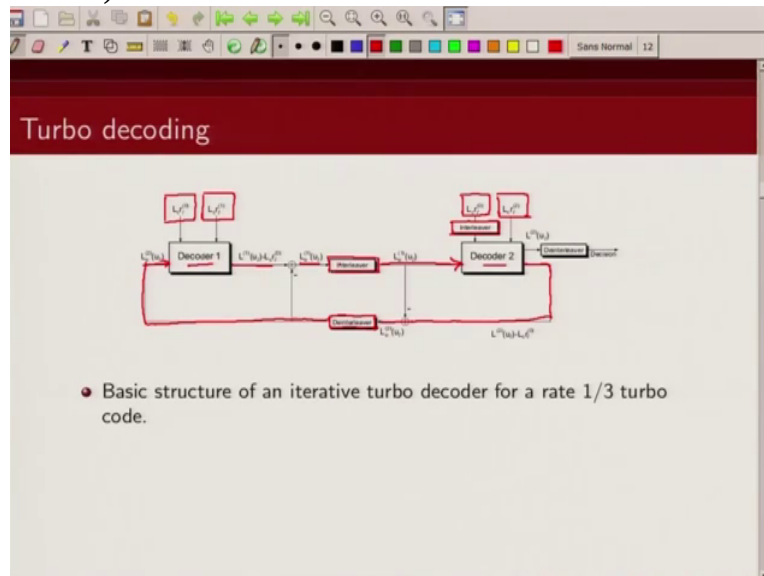
and the a priori value that you send

(Refer Slide Time 30:46)



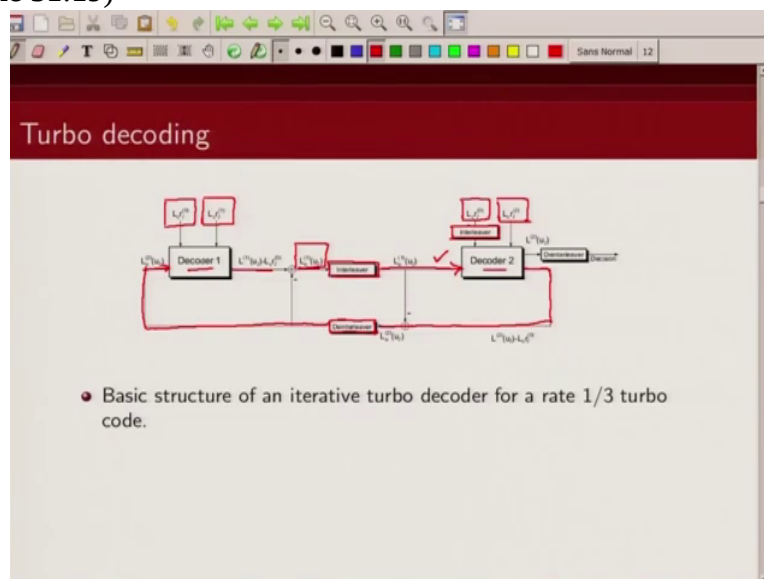
here is coming from this decoder. And I am interleaving this because the order of

(Refer Slide Time 30:54)



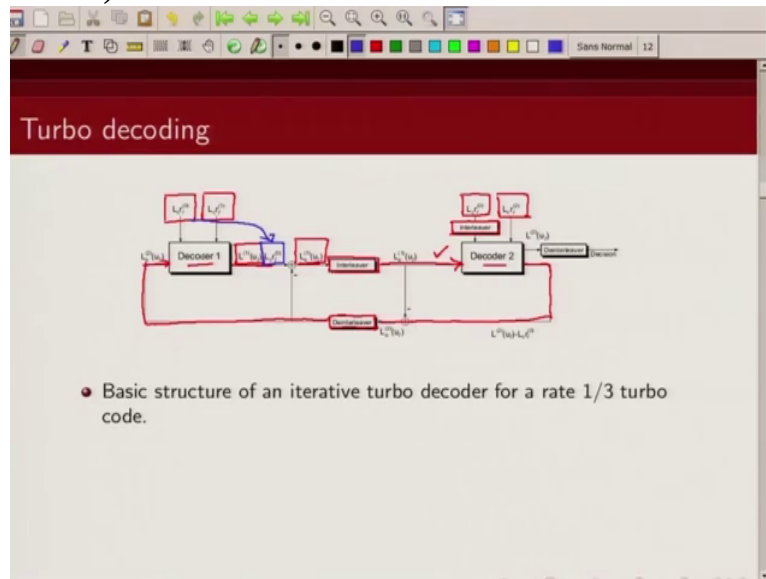
information bit at decoder 2 is interleaved version of the order of information bit at decoder 1. This one more thing I am computing here and I talked about extrinsic value, right? So what is my extrinsic value? This is what I am computing here. So let's pay some attention

(Refer Slide Time 31:19)



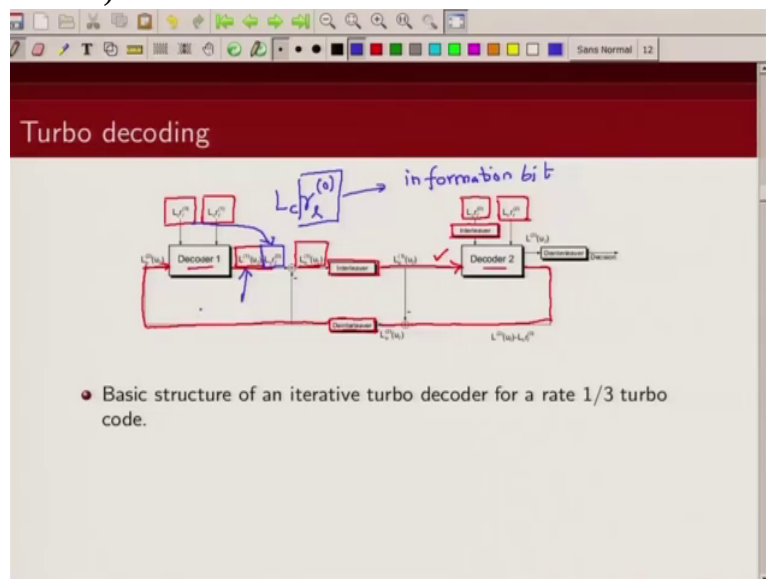
to, this is the extrinsic value. So what is this extrinsic value? So note I am getting this a p p L value computed, that is this. From there I am subtracting the contribution of information bit. This term is same as this bit. This is,

(Refer Slide Time 31:42)



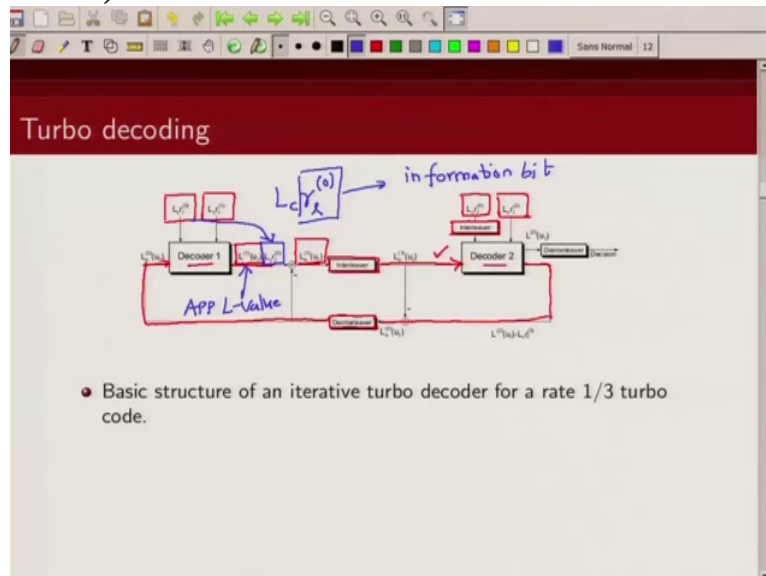
this L C term depends on received s n r. I will come to that. This term is this and this is received value corresponding to

(Refer Slide Time 32:02)



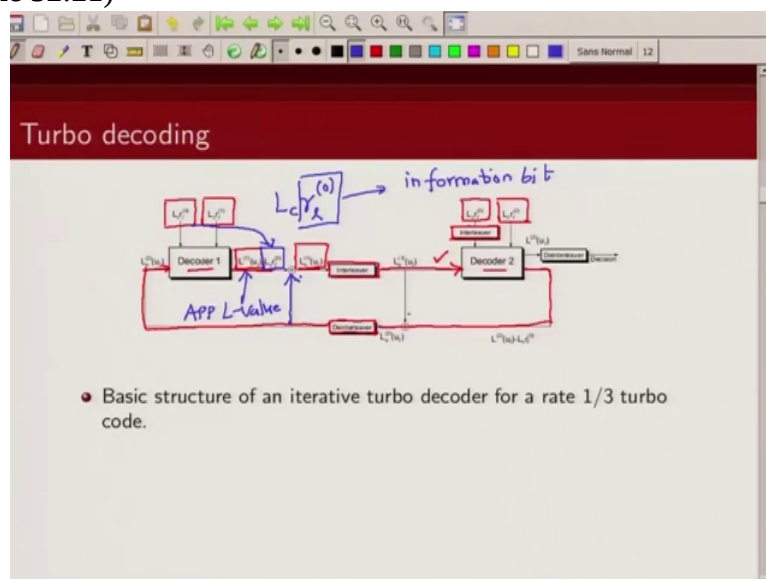
the information bit. And this is the term corresponding to the a p p L value. So what I am doing is I am

(Refer Slide Time 32:09)



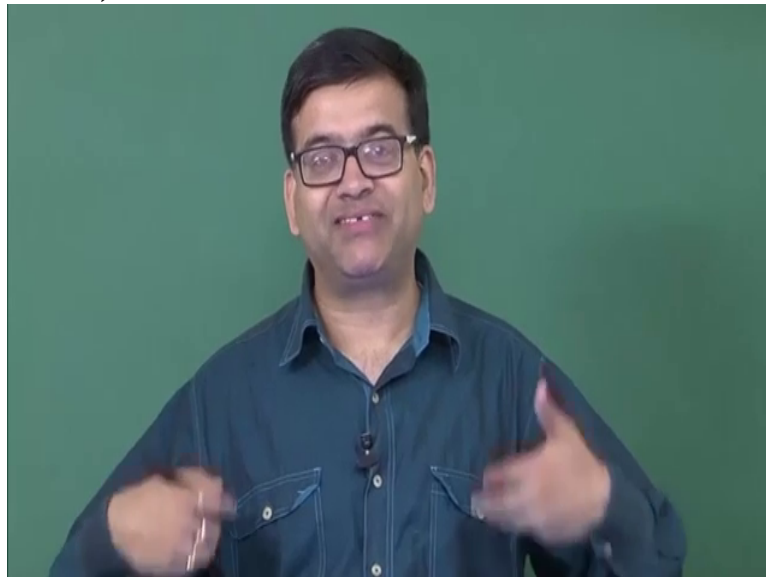
subtracting from the a p p L value the contribution of the received bit. I am also subtracting from this contribution this a priori

(Refer Slide Time 32:21)



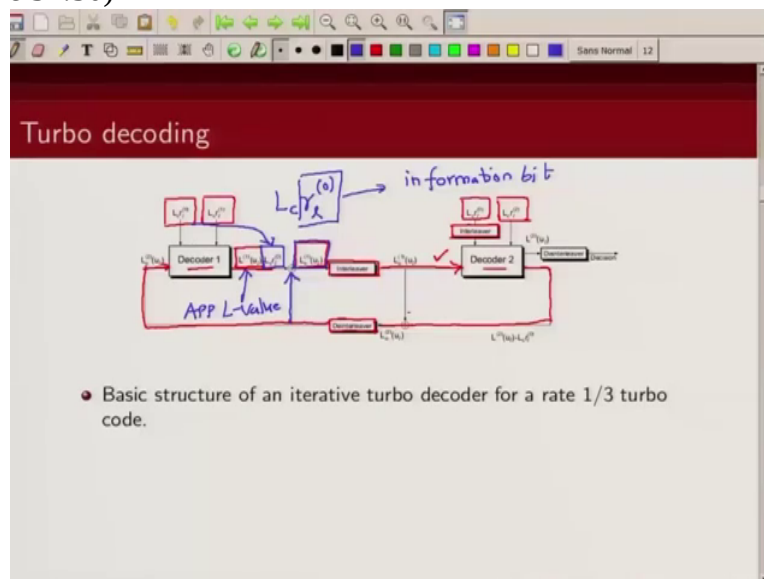
information so what I get is my extrinsic information. So it's like, you can think of it like some,

(Refer Slide Time 32:29)



some additional information about the information bits, that has been derived from the structure of the convolutional encoder. And how I am computing this extrinsic information. From the a p p L value I am subtracting the contribution of the received channel, I am subtracting the

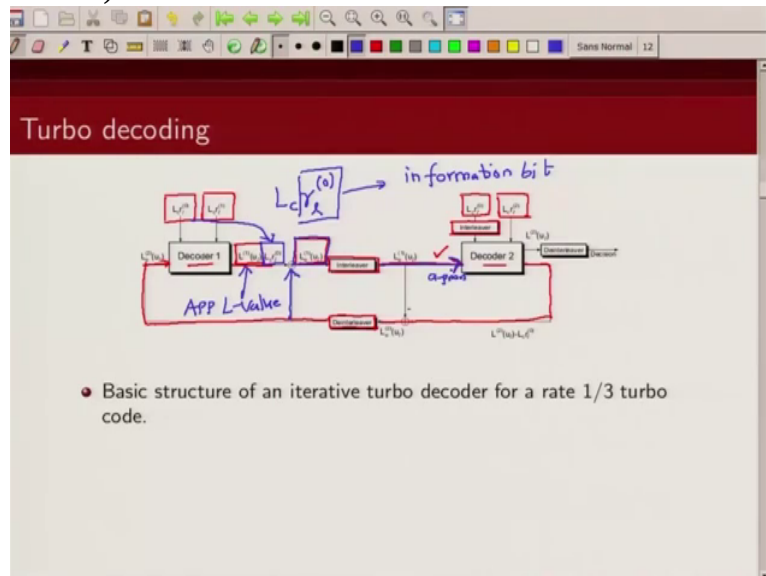
(Refer Slide Time 32:50)



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 a priori information. So this is my a priori information. So



(Refer Slide Time 33:20)



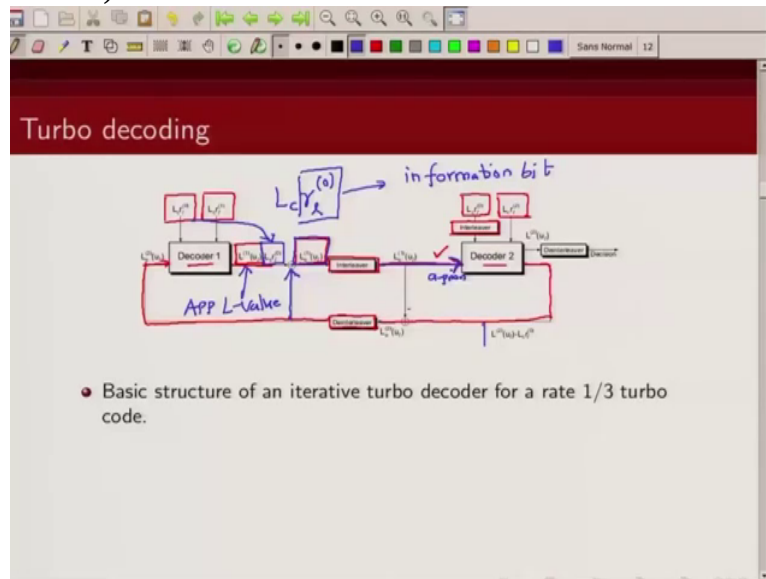
this is how I compute the extrinsic

(Refer Slide Time 33:23)



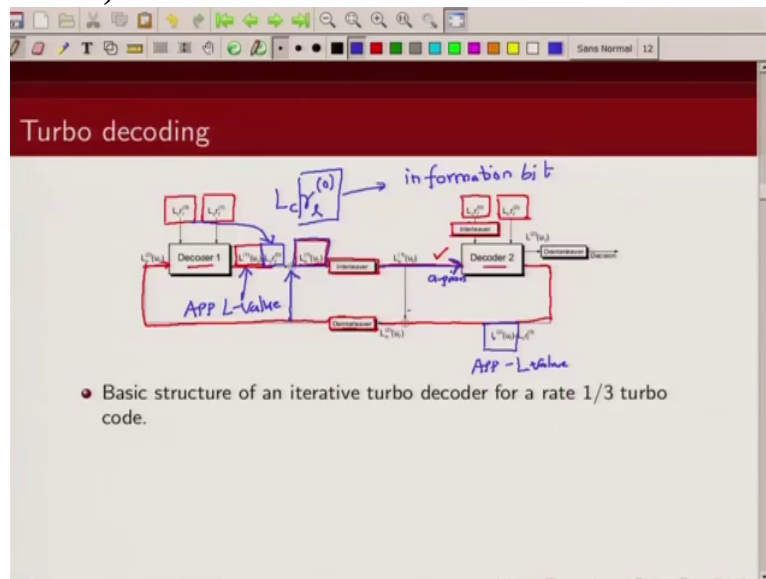
information for decoder 1. Now do I compute extrinsic information for decoder 2, the same procedure? This is

(Refer Slide Time 33:34)



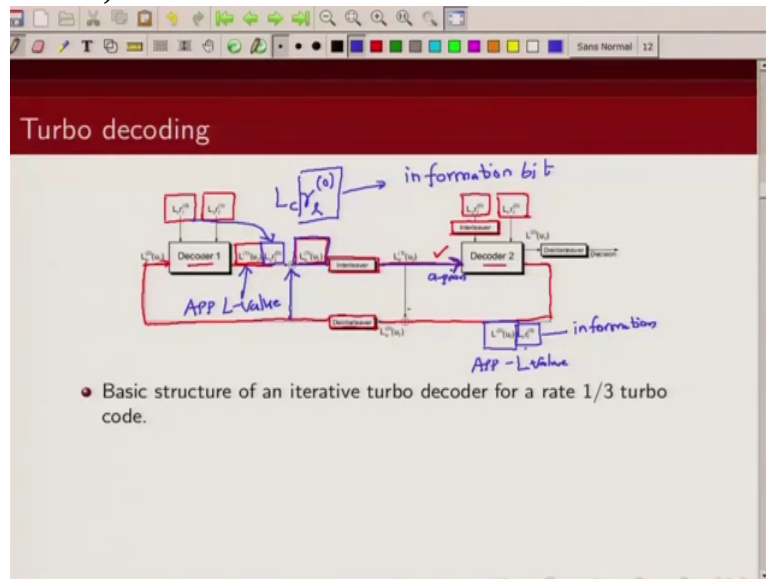
This is my a p p L value correspond to information bit.

(Refer Slide Time 33:43)



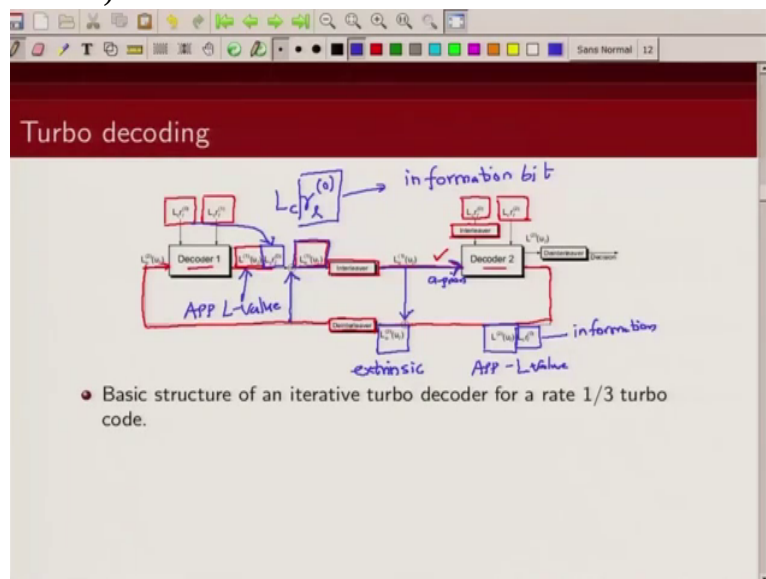
I subtract from there the contribution of information bit. This is my

(Refer Slide Time 33:51)



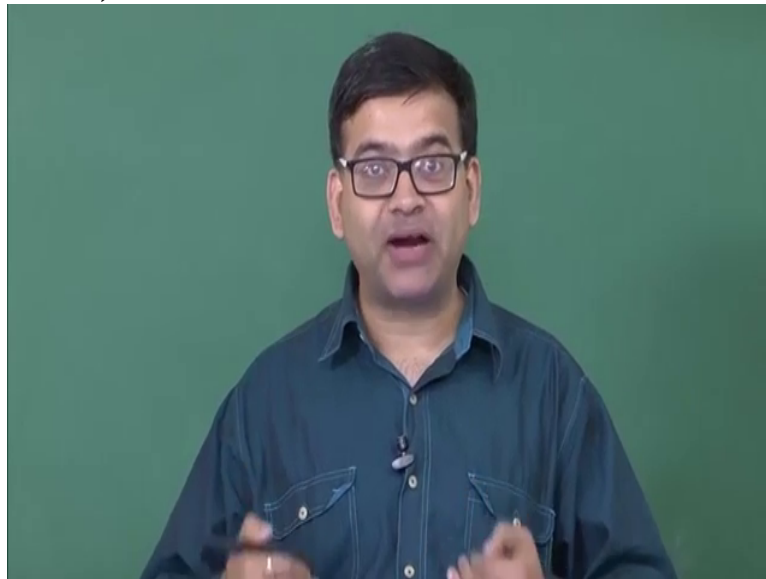
L C r L r 0 term. And then I am subtracting this information of a priori information. So what is left is this information which is my extrinsic information. So for the second decoder in the

(Refer Slide Time 34:09)



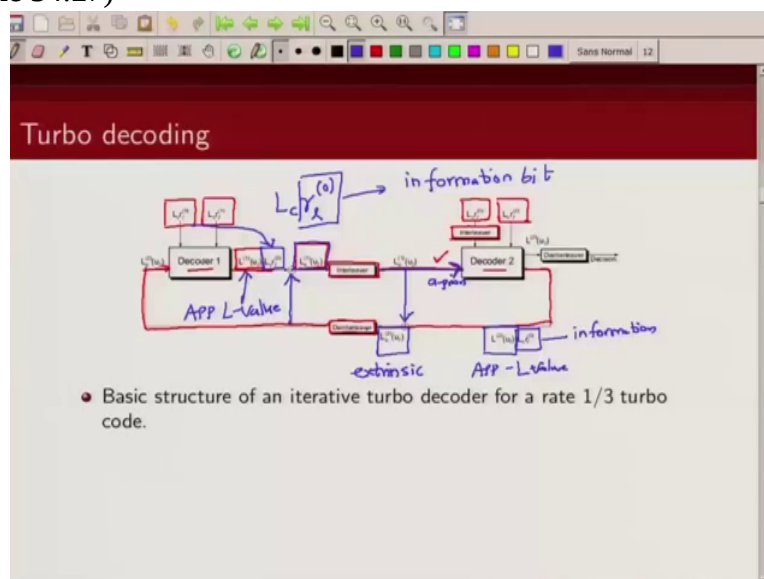
similar fashion I compute the extrinsic information. I subtract

(Refer Slide Time 34:14)



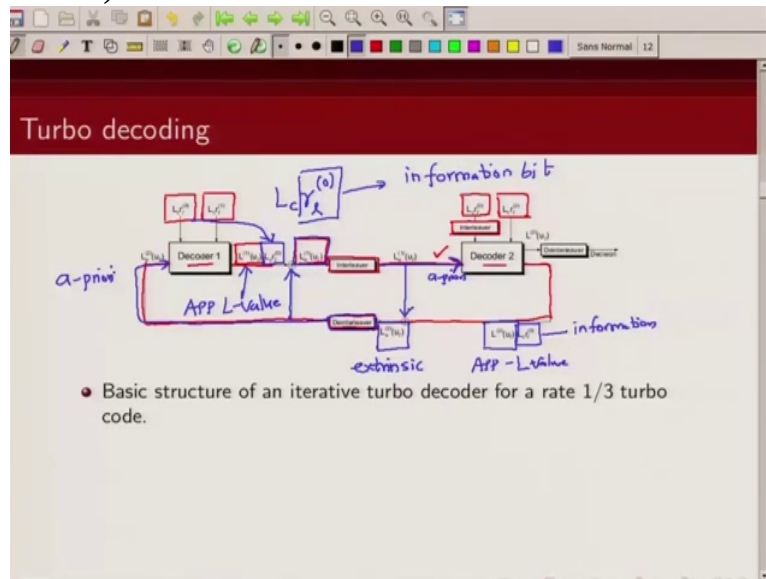
from the a p p value, the contribution of the information bit and the contribution of the a priori knowledge. What is left with is my extrinsic information. Now I

(Refer Slide Time 34:27)



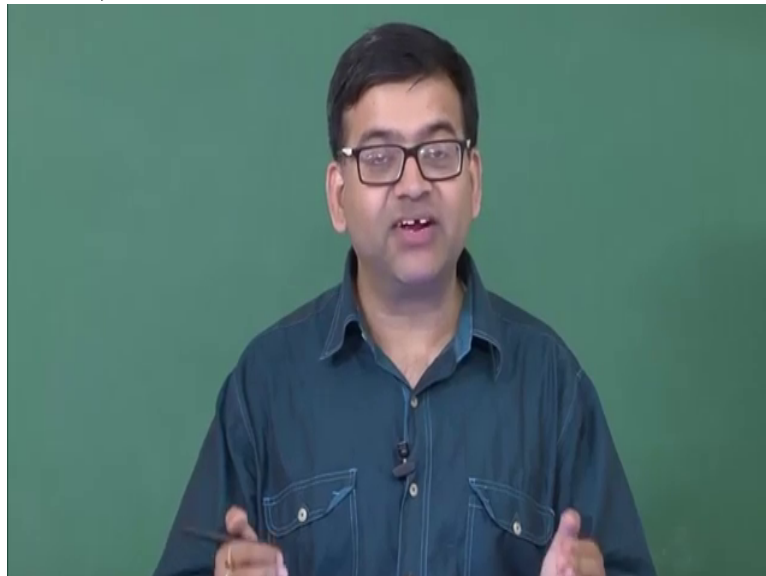
need to de-interleave this information before feeding it back as a priori information. So this is my a priori value for decoder

(Refer Slide Time 34:38)



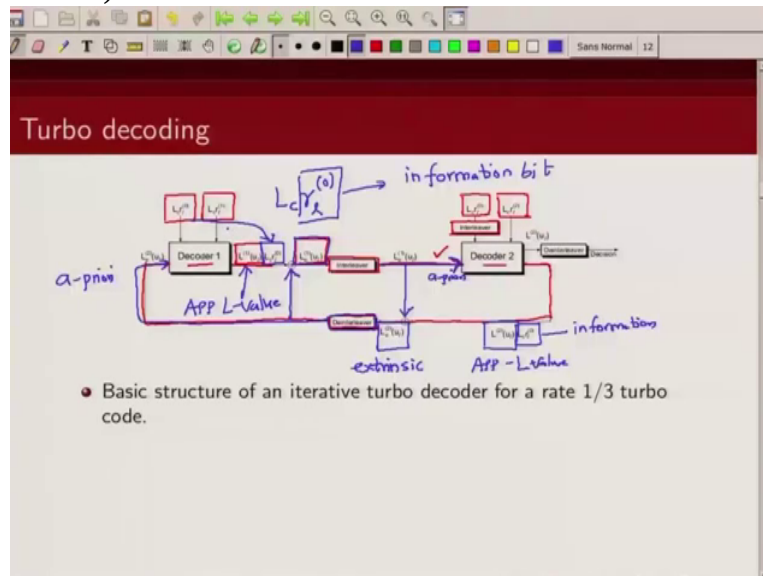
1 and I need to de-interleave because the order of information bit for decoder 1 is de-interleaved version of the order of information for decoder 2. So this is how my iterative

(Refer Slide Time 34:58)



decoder algorithm is working. Again I will do a quick recap. So using the B C J R algorithm so I have

(Refer Slide Time 35:08)



some received values from the channel which is this and this, initially I do not have any a priori knowledge about the information bits so I assume equally likely to be 1 or 0, this decoder of 1 will apply B C J R algorithm and it will compute the a p p 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 the interleaved version of the information sequence as input. So it will again compute a p p 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

(Refer Slide Time 36:04)



one decoder to another, Ok. And ideally what we would like is this that extrinsic information should grow in a manner that it pushed the decision in favor of either information bit being plus 1 or minus 1. That's when we say the decoder is converging with iteration. So we have

(Refer Slide Time 36:25)

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

already explained that there are 2 decoders and each of them are using MAP algorithm this BCJR algorithm that we talked about.

(Refer Slide Time 36:34)

- 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  $i$ , three output values are received from the channel, one for the information bit  $u_i = v_i^{(0)}$ , denoted  $r_i^{(0)}$ , and two for the parity bits  $v_i^{(1)}$  and  $v_i^{(2)}$ , denoted  $r_i^{(1)}$  and  $r_i^{(2)}$ .

There are 3 inputs received from the channel, one corresponds this  $r_i^{(0)}$ , corresponds to the information bit  $r_i^{(1)}$ , corresponds to the parity bit corresponding to encoder 1 and  $r_i^{(2)}$  the parity bit corresponding to encoder 2.

(Refer Slide Time 37:02)

**Turbo decoding**

- The  $3K$ -dimensional received vector is denoted by
 
$$\mathbf{r} = (r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \dots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)}).$$
- 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
 
$$L(u_i | r_i^{(0)}) = \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})}$$

$$= \ln \frac{P(r_i^{(0)} | u_i = +1) P(u_i = +1)}{P(r_i^{(0)} | u_i = -1) P(u_i = -1)}$$

So at each time instance, so this is my time index, time 0, time 1, time k minus 1, so each time instance I am getting this 3 bit information which is information bit, first parity bit and the second parity bit. Remember this is a rate one third code. If you have a information block of k you will get 3 k coded bits. 0 is mapped to in this case, minus 1, plus 1 is 1. Now let's compute the log likelihood ratio. So probability of u L given r L is zero is given by what's the probability that u L is plus 1 given this received sequence r L divided by probability of u L being minus 1 given received sequence we can write as probability of r given u multiplied by probability of u. And that's how we have written it. Now we can separate out into 2 terms. So this is one term we have and this is another term we have. So log of a

(Refer Slide Time 38:21)

**Turbo decoding**

- The  $3K$ -dimensional received vector is denoted by
 
$$\mathbf{r} = (r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \dots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)}).$$
- 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
 
$$L(u_i | r_i^{(0)}) = \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})}$$

$$= \ln \left( \frac{P(r_i^{(0)} | u_i = +1) P(u_i = +1)}{P(r_i^{(0)} | u_i = -1) P(u_i = -1)} \right)$$

times b, log a plus log b,



(Refer Slide Time 38:24)

**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

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

(Refer Slide Time 38:49)

**Turbo decoding**

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

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

and other is this.

(Refer Slide Time 38:53)

**Turbo decoding**

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

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

Now what is this? This is 4 times  $E_s$  by  $N_0$  times  $r_i^{(0)}$ .  $r_i^{(0)}$  is my information, received information sequence. So this you can, four  $E_s$  by  $N_0$  I am writing as  $L_c$  and calling it a channel reliability factor. It depends on the signal to noise ratio of the channel. And this you can see, log of probability of  $u_i$  being plus 1 divided by  $u_i$  being minus 1, this is a priori knowledge of the information bit being plus 1 or minus 1. So there are 2 inputs, one coming from that channel which is multiplied by this factor  $L_c$  and other is a priori knowledge. And that's why you noticed here in the diagram where I had,

(Refer Slide Time 39:45)

**Turbo decoding**

The diagram shows an iterative turbo decoder structure. It consists of two SISO decoders, Decoder 1 and Decoder 2, connected in series. Decoder 1 receives three inputs:  $L_c^{(0)}$ ,  $L_c^{(1)}$ , and  $L_c^{(2)}$ . It outputs  $L_c^{(1)}$  and  $L_c^{(2)}$ . These are then fed into Decoder 2, which outputs  $L_c^{(0)}$ . The outputs of Decoder 2 are fed back into Decoder 1. The overall output is  $L_c^{(0)}$ .

- 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  $i$ , three output values are received from the channel, one for the information bit  $u_i = v_i^{(0)}$ , denoted  $r_i^{(0)}$ , and two for the parity bits  $v_i^{(1)}$  and  $v_i^{(2)}$ , denoted  $r_i^{(1)}$  and  $r_i^{(2)}$ .

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

(Refer Slide Time 40:04)

**Turbo decoding**

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

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

(Refer Slide Time 40:07)

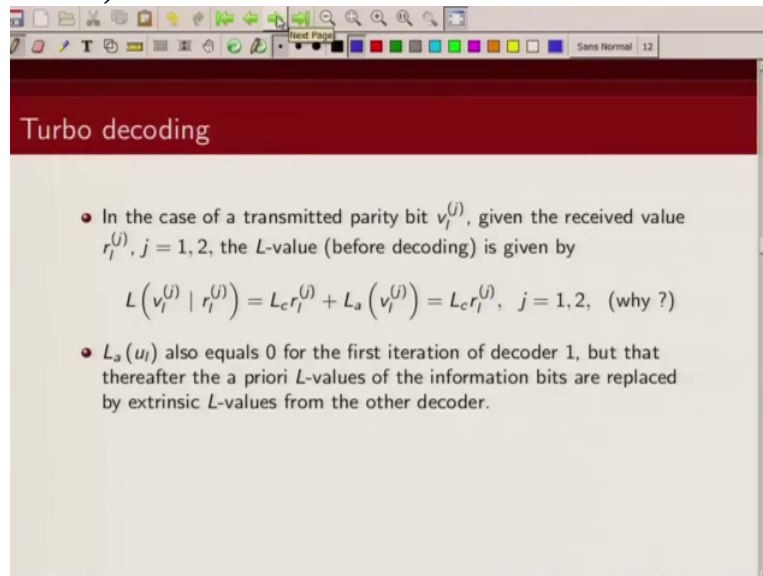
**Turbo decoding**

- 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

$$L(v_i^{(j)} | r_i^{(j)}) = L_c r_i^{(j)} + L_a(v_i^{(j)}) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$

Now in case of transmitted parity bit, we do not have any a priori knowledge so in those case this will be just  $L_c r_i^{(j)}$  depending on whether we are considering parity bit 1 or parity bit 2.

(Refer Slide Time 40:27)

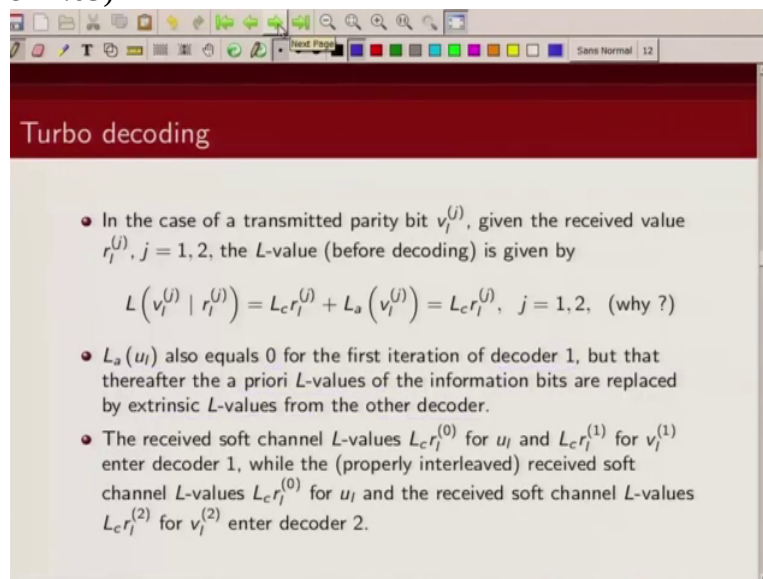


The slide is titled "Turbo decoding" and contains the following text:

- 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
$$L(v_i^{(j)} | r_i^{(j)}) = L_c r_i^{(j)} + L_a(v_i^{(j)}) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_a(u_i)$  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 don't have any a priori knowledge about whether the bit is plus 1 or minus 1, so we would assume it's equally likely to be plus 1 and minus 1. So the a priori log likelihood value  $L$  value will be considered as zero for the first decoder. But thereafter these a priori values will be replaced by the extrinsic values received from the other decoder, right?

(Refer Slide Time 41:09)



The slide is titled "Turbo decoding" and contains the following text:

- 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
$$L(v_i^{(j)} | r_i^{(j)}) = L_c r_i^{(j)} + L_a(v_i^{(j)}) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_a(u_i)$  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.
- 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  $L_c r_i^{(0)}$  and  $r_i^{(1)}$  and  $r_i^{(2)}$ , remember to feed them in proper order

(Refer Slide Time 41:23)



when you are feeding to decoder 1 and decoder 2 and this we have

(Refer Slide Time 41:29)

**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
$$L(v_i^{(j)} | r_i^{(j)}) = L_c r_i^{(j)} + L_a(v_i^{(j)}) = L_c r_i^{(j)}, \quad j = 1, 2, \quad (\text{why?})$$
- $L_a(u_i)$  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.
- 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.

explained also earlier. You can see when we feed

(Refer Slide Time 41:34)

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  $i$ , three output values are received from the channel, one for the information bit  $u_i = v_i^{(0)}$ , denoted  $r_i^{(0)}$ , and two for the parity bits  $v_i^{(1)}$  and  $v_i^{(2)}$ , denoted  $r_i^{(1)}$  and  $r_i^{(2)}$ .

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's 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 de-interleaver and when we are taking decision from

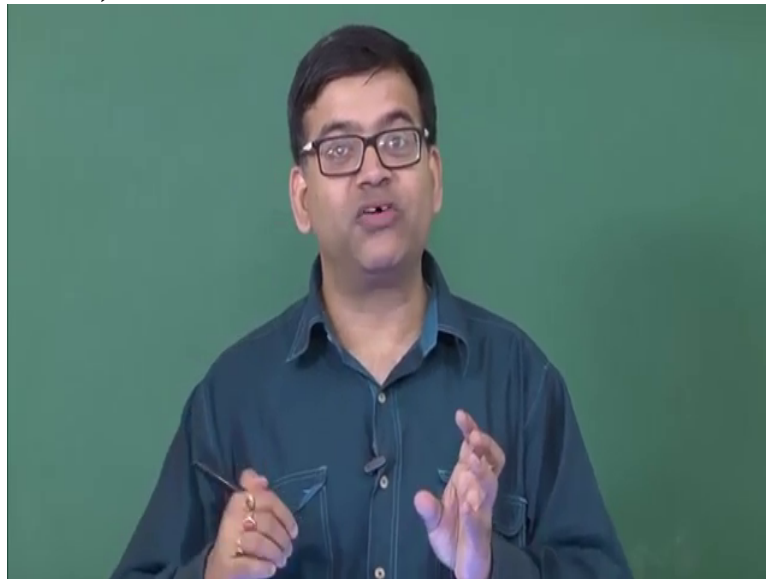
(Refer Slide Time 41:56)

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  $i$ , three output values are received from the channel, one for the information bit  $u_i = v_i^{(0)}$ , denoted  $r_i^{(0)}$ , and two for the parity bits  $v_i^{(1)}$  and  $v_i^{(2)}$ , denoted  $r_i^{(1)}$  and  $r_i^{(2)}$ .

decoder 2 we are also again doing de-interleaving. So this interleaving, de-interleaving is done so

(Refer Slide Time 42:04)



as the order of the information bit is preserved in the fashion they are entering encoder 1 and

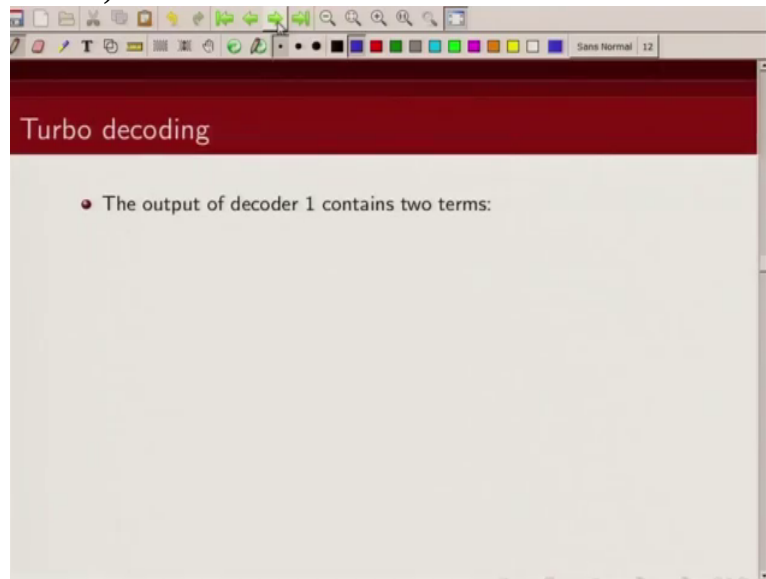
(Refer Slide Time 42:10)

**Turbo decoding**

- The  $3K$ -dimensional received vector is denoted by
$$\mathbf{r} = (r_0^{(0)} r_0^{(1)} r_0^{(2)}, r_1^{(0)} r_1^{(1)} r_1^{(2)}, \dots, r_{K-1}^{(0)} r_{K-1}^{(1)} r_{K-1}^{(2)}).$$
- 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
$$\begin{aligned} L(u_i | r_i^{(0)}) &= \ln \frac{P(u_i = +1 | r_i^{(0)})}{P(u_i = -1 | r_i^{(0)})} \\ &= \ln \left( \frac{P(r_i^{(0)} | u_i = +1)}{P(r_i^{(0)} | u_i = -1)} \right) \frac{P(u_i = +1)}{P(u_i = -1)} \end{aligned}$$

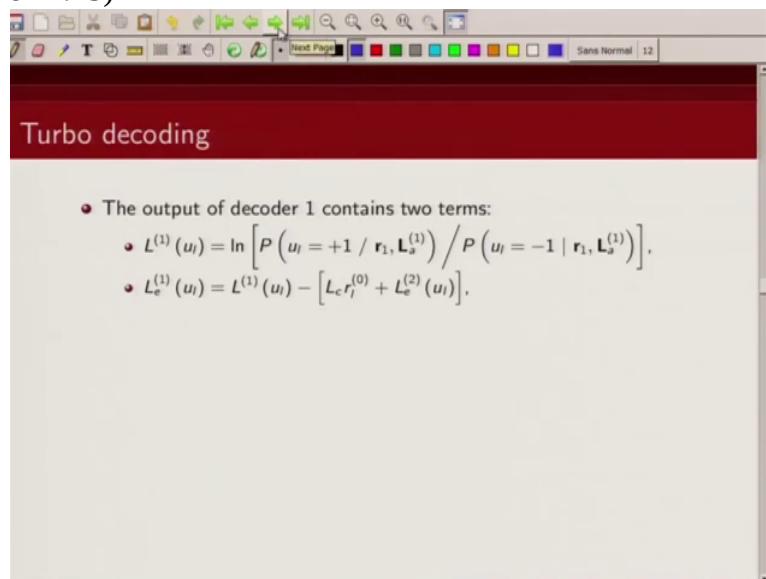
encoder 2.

(Refer Slide Time 42:13)



So as I said

(Refer Slide Time 42:15)



there are 3 inputs to the decoder.

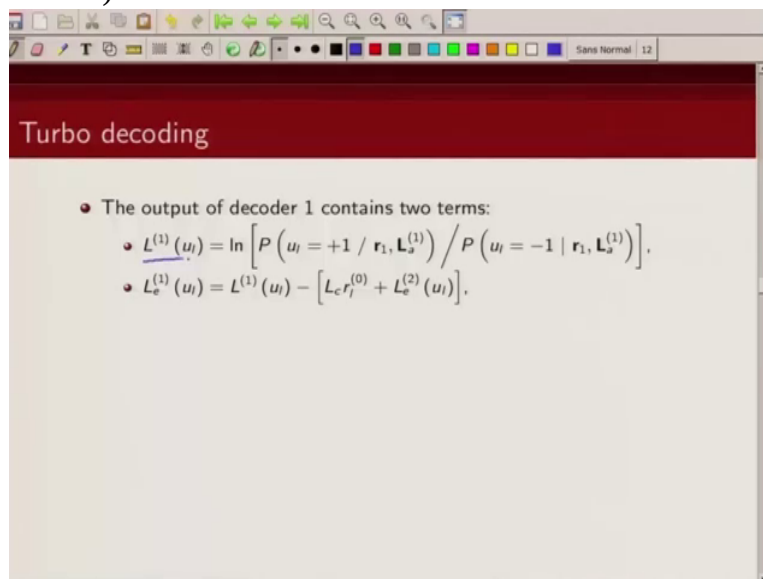


(Refer Slide Time 42:20)



Channel received value corresponds to information sequence and parity bit and a priori value and there are 2 output, one is this

(Refer Slide Time 42:30)



a p p L value that we have computed and the second is extrinsic value and how are we computing extrinsic value, from the a p p L value, we are subtracting the contribution

(Refer Slide Time 42:44)

Turbo decoding

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

of the received channel value and we are subtracting the contribution of

(Refer Slide Time 42:51)

Turbo decoding

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

a priori value which is nothing but extrinsic value of the other decoder,

(Refer Slide Time 43:01)

**Turbo decoding**

- The output of decoder 1 contains two terms:
  - $L^{(1)}(u_i) = \ln \left[ \frac{P(u_i = +1 | \mathbf{r}_1, \mathbf{L}_a^{(1)})}{P(u_i = -1 | \mathbf{r}_1, \mathbf{L}_a^{(1)})} \right]$ ,
  - $L_e^{(1)}(u_i) = L^{(1)}(u_i) - [L_c r_i^{(0)} + L_e^{(2)}(u_i)]$ ,
- $L^{(1)}(u_i)$  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  $\mathbf{L}_a^{(1)} \triangleq [L_a^{(1)}(u_0), L_a^{(1)}(u_1), \dots, L_a^{(1)}(u_{K-1})]$  for decoder 1.

Ok. So this we have explained.

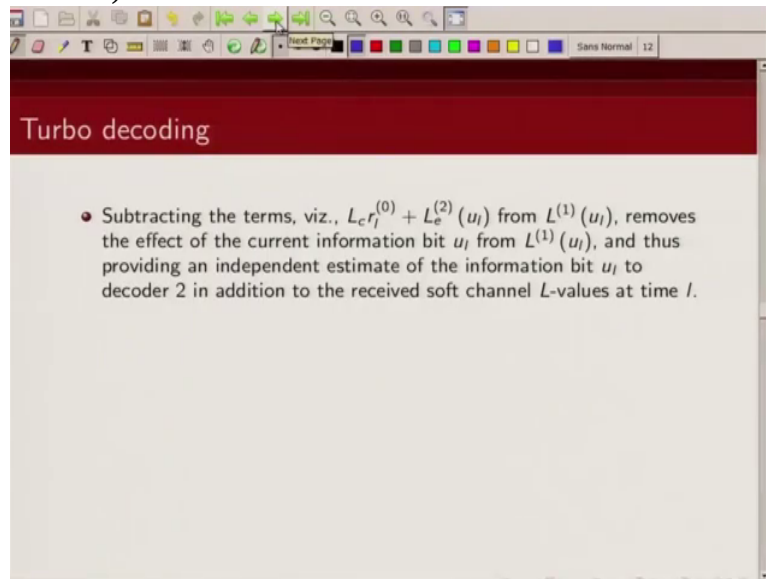
(Refer Slide Time 43:06)

**Turbo decoding**

- The output of decoder 1 contains two terms:
  - $L^{(1)}(u_i) = \ln \left[ \frac{P(u_i = +1 | \mathbf{r}_1, \mathbf{L}_a^{(1)})}{P(u_i = -1 | \mathbf{r}_1, \mathbf{L}_a^{(1)})} \right]$ ,
  - $L_e^{(1)}(u_i) = L^{(1)}(u_i) - [L_c r_i^{(0)} + L_e^{(2)}(u_i)]$ ,
- $L^{(1)}(u_i)$  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  $\mathbf{L}_a^{(1)} \triangleq [L_a^{(1)}(u_0), L_a^{(1)}(u_1), \dots, L_a^{(1)}(u_{K-1})]$  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  $L_a^{(2)}(u_i)$ .

This is the a p L value and this is the extrinsic L value.

(Refer Slide Time 43:14)



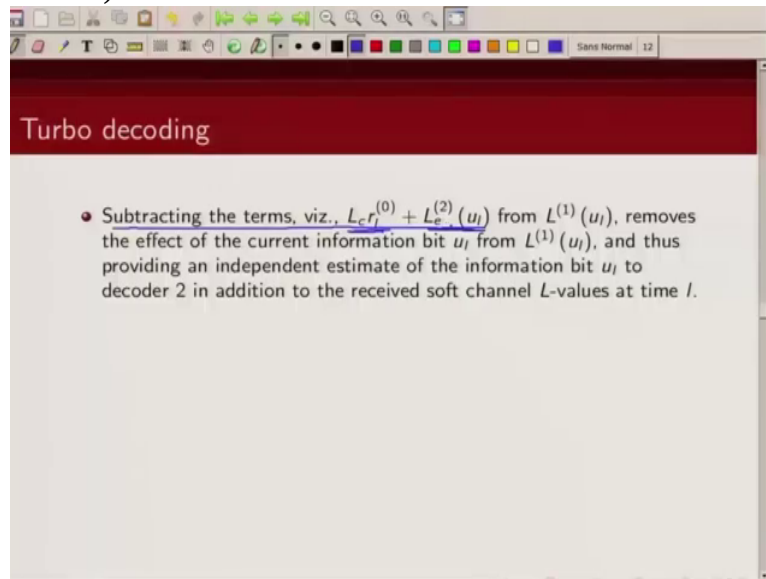
So why are we subtracting these terms, these information bit term and the a priori term? So we essentially are trying to remove

(Refer Slide Time 43:27)



the effect of the current bit in some sense from the a p p value so in some sense we are trying to provide some independent estimate because

(Refer Slide Time 43:39)



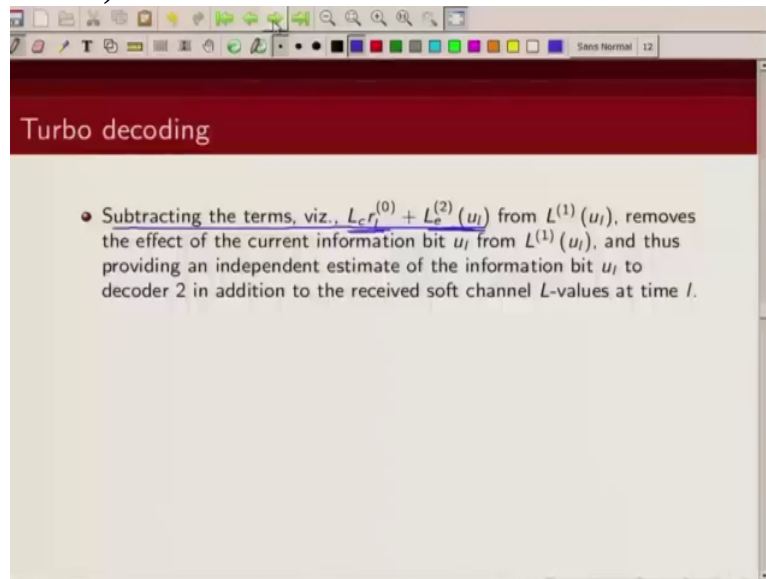
anyway this a priori information has come from the previ, extrinsic information from the previous decoder. So there is no point feeding the same information back to the same

(Refer Slide Time 43:49)



decoder. And the received values are already received by the decoder. So we are trying to, in some one way, trying to send some independent estimate about what we think the bits are to the other decoder. And that's the whole idea behind uh iterative process that you want to give some sort of independent estimate. You try to feed the same information back then it will become a positive feedback system and unstable, the decoder

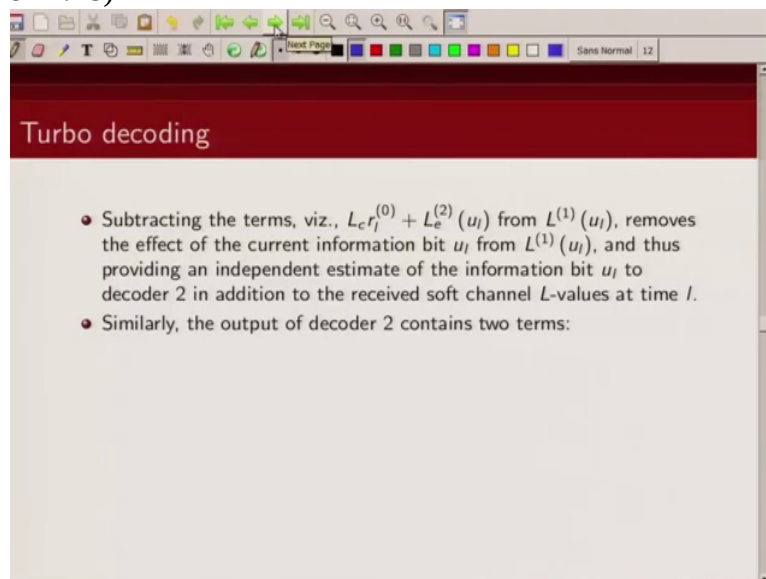
(Refer Slide Time 44:15)



The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there is a single bullet point. The text of the bullet point is: "Subtracting the terms, viz.,  $L_e r_l^{(0)} + L_e^{(2)}(u_l)$  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  $l$ ."

may not convert. So that we don't want.

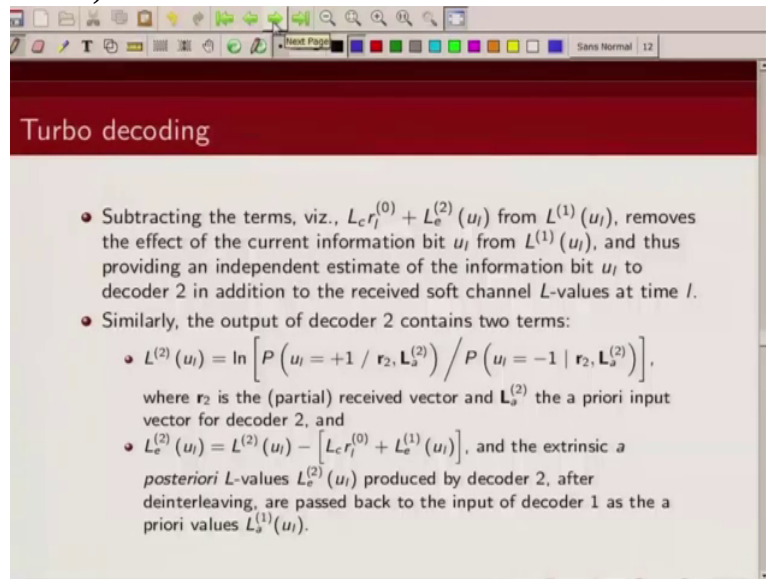
(Refer Slide Time 44:18)



The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there are two bullet points. The text of the first bullet point is: "Subtracting the terms, viz.,  $L_e r_l^{(0)} + L_e^{(2)}(u_l)$  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  $l$ ." The text of the second bullet point is: "Similarly, the output of decoder 2 contains two terms:"

And in the same fashion the decoder 2 will also have 2 terms.

Refer Slide Time 44:23)

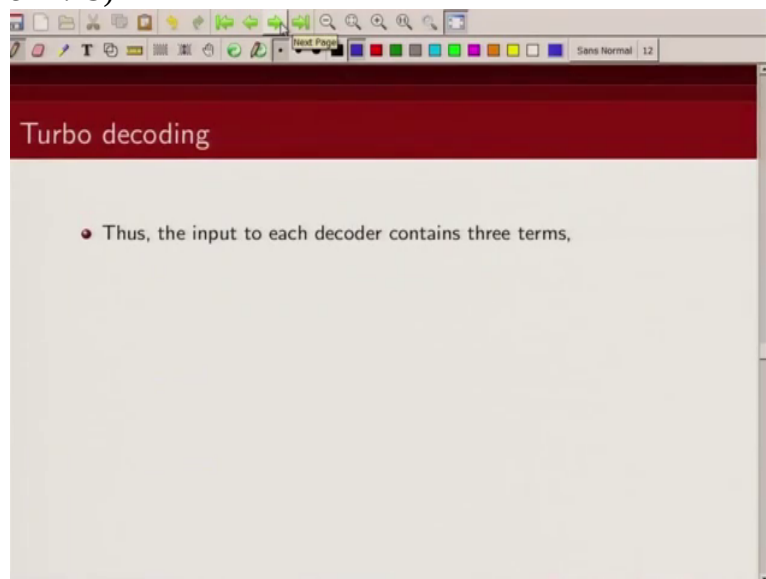


The slide is titled "Turbo decoding" and contains the following text:

- Subtracting the terms, viz.,  $L_c r_l^{(0)} + L_e^{(2)}(u_l)$  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  $l$ .
- Similarly, the output of decoder 2 contains two terms:
  - $L^{(2)}(u_l) = \ln \left[ \frac{P(u_l = +1 | \mathbf{r}_2, \mathbf{L}_s^{(2)})}{P(u_l = -1 | \mathbf{r}_2, \mathbf{L}_s^{(2)})} \right]$ , where  $\mathbf{r}_2$  is the (partial) received vector and  $\mathbf{L}_s^{(2)}$  the a priori input vector for decoder 2, and
  - $L_e^{(2)}(u_l) = L^{(2)}(u_l) - [L_c r_l^{(0)} + L_e^{(1)}(u_l)]$ , and the extrinsic a posteriori  $L$ -values  $L_e^{(2)}(u_l)$  produced by decoder 2, after deinterleaving, are passed back to the input of decoder 1 as the a priori values  $L_s^{(1)}(u_l)$ .

One is this a p p value and other is this extrinsic value.

(Refer Slide Time 44:29)

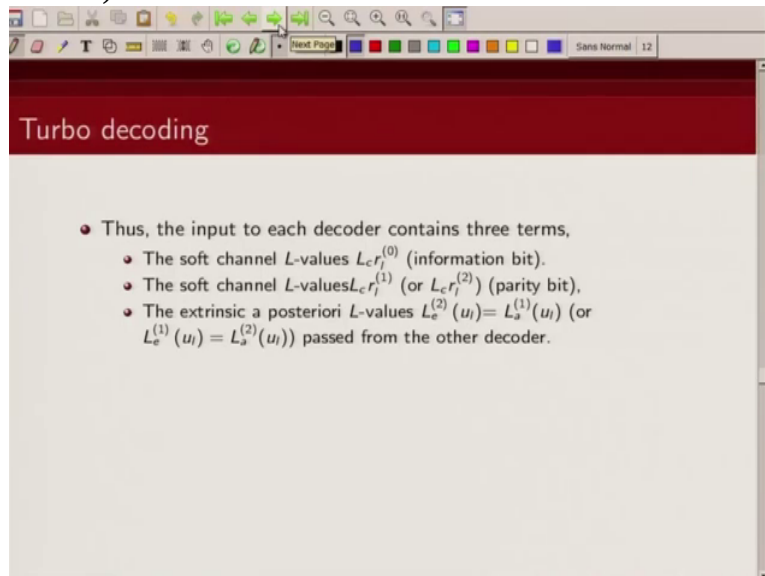


The slide is titled "Turbo decoding" and contains the following text:

- Thus, the input to each decoder contains three terms,

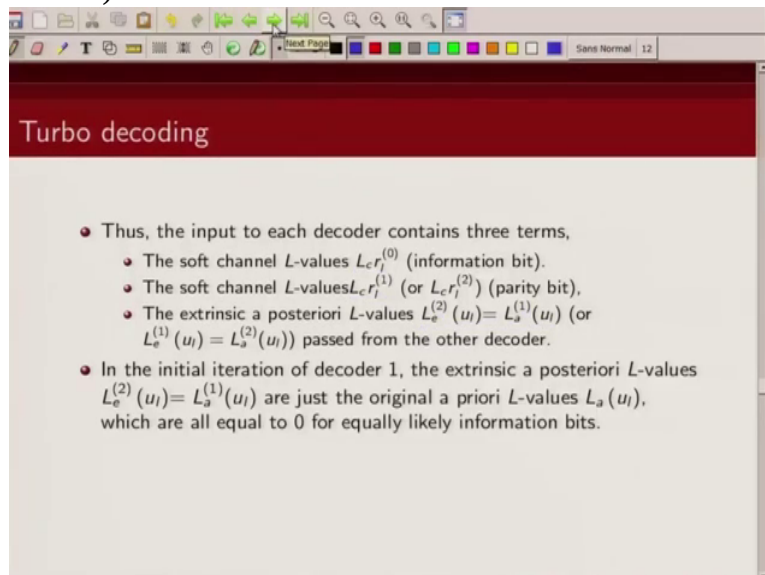
I have already

(Refer Slide Time 44:35)



received information bit, one term corresponding to the received parity bit and one term corresponding to the a priori information which is being 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 de-interleaving. Because you want to ensure the order of the information bit is same.

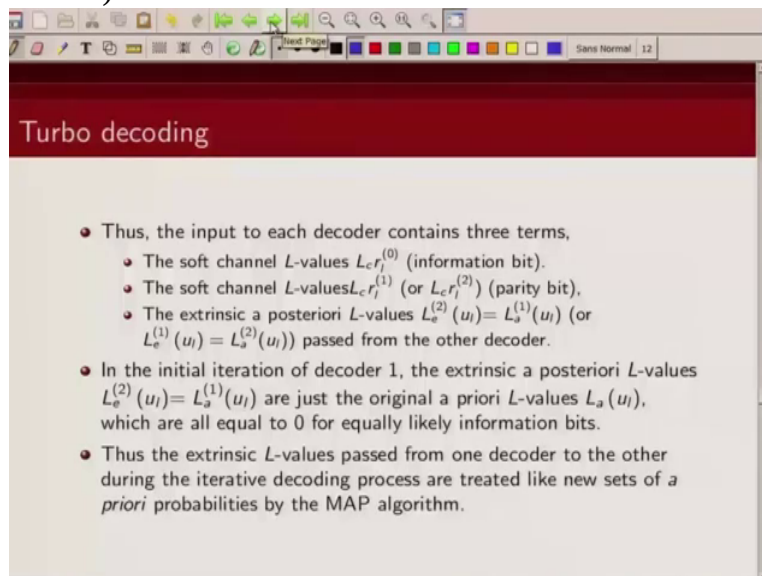
(Refer Slide Time 45:17)



Ok this I have explained, in initial iteration extrinsic information is basically zero and subsequently,



(Refer Slide Time 45:26)

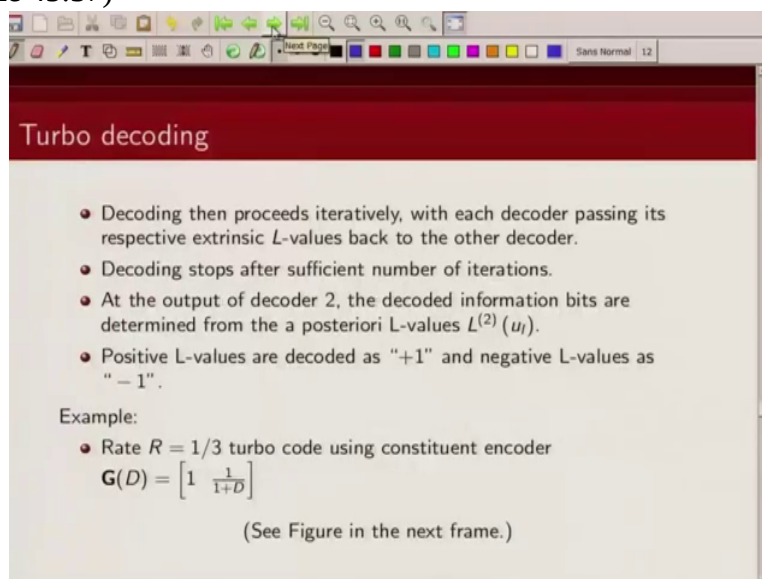


The slide is titled "Turbo decoding" and contains the following text:

- 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  $L_c r_i^{(1)}$  (or  $L_c r_i^{(2)}$ ) (parity bit),
  - The extrinsic a posteriori  $L$ -values  $L_e^{(2)}(u_i) = L_s^{(1)}(u_i)$  (or  $L_e^{(1)}(u_i) = L_s^{(2)}(u_i)$ ) passed from the other decoder.
- In the initial iteration of decoder 1, the extrinsic a posteriori  $L$ -values  $L_e^{(2)}(u_i) = L_s^{(1)}(u_i)$  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.

a priori information is zero and subsequently the a priori value will be nothing but the extrinsic information.

(Refer Slide Time 45:37)



The slide is titled "Turbo decoding" and contains the following text:

- 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

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

(See Figure in the next frame.)

So this process as I said goes on repeatedly in iterative fashion. So let's take an example

(Refer Slide Time 45:47)



and see how this works.

(Refer Slide Time 45:51)

A screenshot of a presentation slide. The slide has a red header with the title "Turbo decoding". Below the header, there are four bullet points describing the iterative decoding process. An "Example:" section follows, describing a rate 1/3 turbo code and showing the generator polynomial  $G(D) = \begin{bmatrix} 1 & \frac{1}{1+D} \end{bmatrix}$ . The slide also includes a note to see a figure in the next frame. The slide is displayed in a window with a standard toolbar and a font size of 12.

**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:

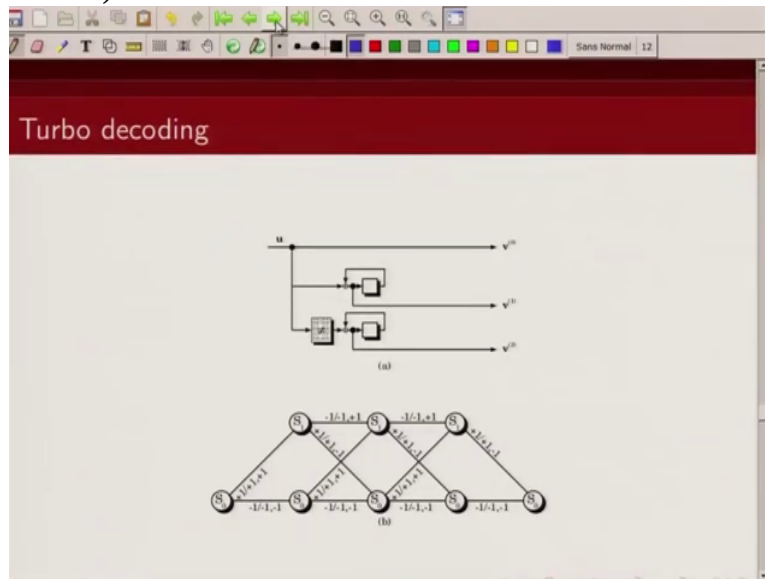
- Rate  $R = 1/3$  turbo code using constituent encoder

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

(See Figure in the next frame.)

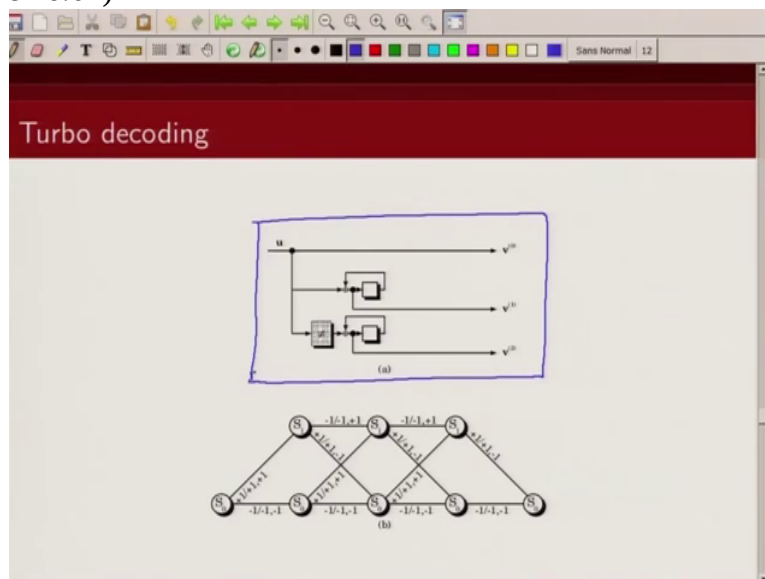
So we are considering a rate one third turbo code where the constituent encoder is this 2 state encoder. So our

(Refer Slide Time 46:02)



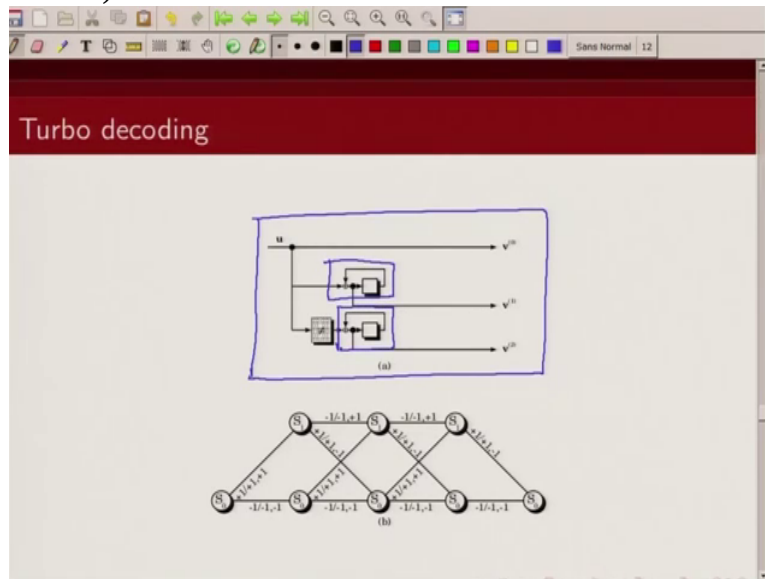
turbo code is this.

(Refer Slide Time 46:07)



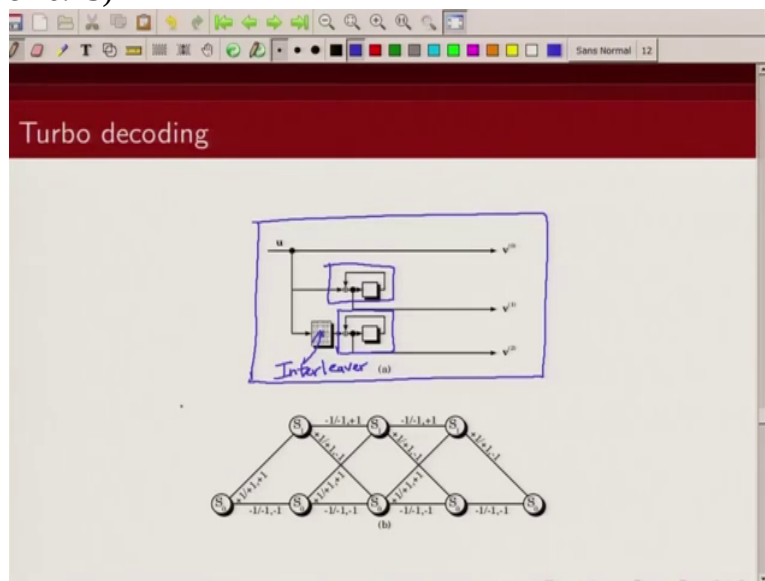
Each one of them is using this 2 state recursive convolutional

(Refer Slide Time 46:14)



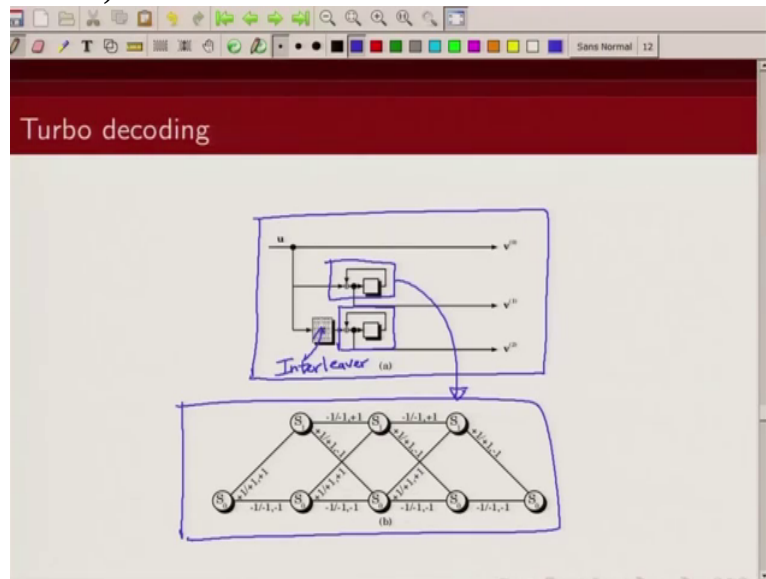
encoder. This is my, this is my interleaver. And this

(Refer Slide Time 46:23)



is the state diagram corresponding to these convolutional encoders,

(Refer Slide Time 46:32)



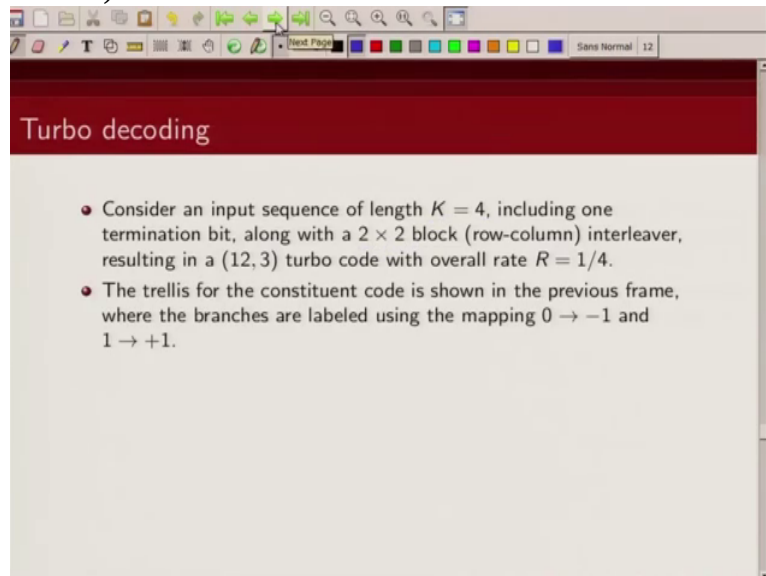
Ok.

(Refer Slide Time 46:35)

The slide, titled "Turbo decoding", contains a single bullet point: "Consider an input sequence of length  $K = 4$ , including one termination bit, along with a  $2 \times 2$  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's say I am doing block interleaving. So I am fitting the data block wise and reading it

(Refer Slide Time 46:48)

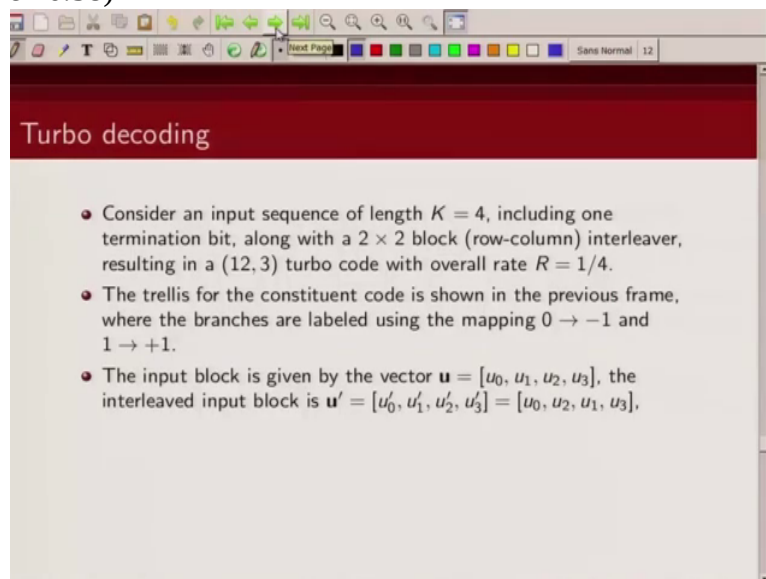


The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there are two bullet points:

- Consider an input sequence of length  $K = 4$ , including one termination bit, along with a  $2 \times 2$  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 wise. So as I have said I am mapping 0 to minus 1 here and 1 to plus 1.

(Refer Slide Time 46:58)

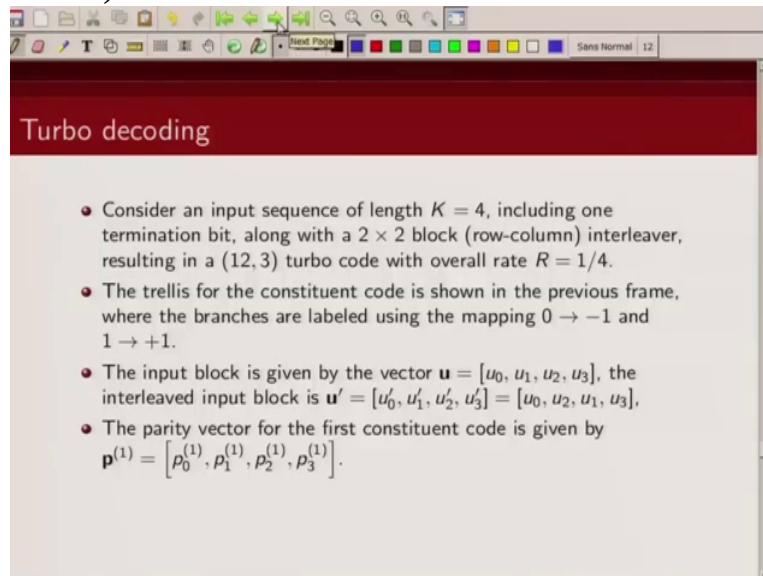


The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there are three bullet points:

- Consider an input sequence of length  $K = 4$ , including one termination bit, along with a  $2 \times 2$  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  $\mathbf{u}' = [u'_0, u'_1, u'_2, u'_3] = [u_0, u_2, u_1, u_3]$ ,

So this is my input block. The interleaved block is given by u hat

(Refer Slide Time 47:06)

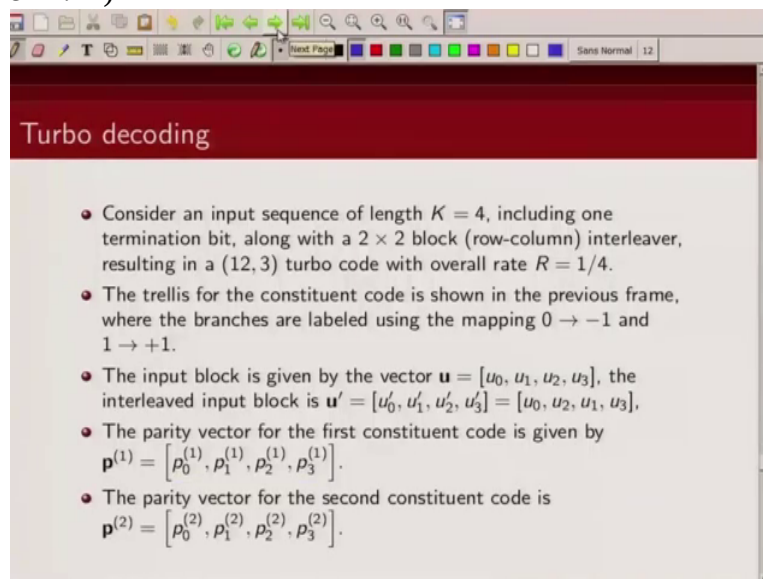


**Turbo decoding**

- Consider an input sequence of length  $K = 4$ , including one termination bit, along with a  $2 \times 2$  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  $\mathbf{p}^{(1)} = [p_0^{(1)}, p_1^{(1)}, p_2^{(1)}, p_3^{(1)}]$ .

and corresponding parity for the first encoder is given by this and parity

(Refer Slide Time 47:14)

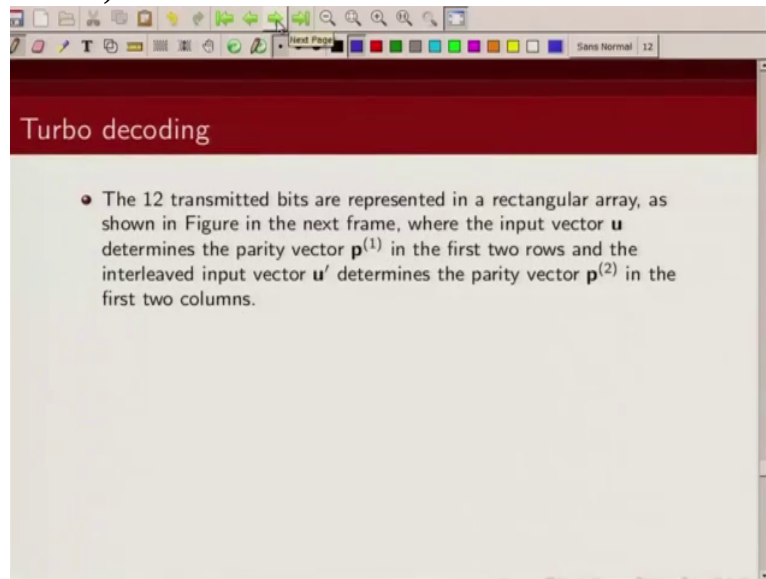


**Turbo decoding**

- Consider an input sequence of length  $K = 4$ , including one termination bit, along with a  $2 \times 2$  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  $\mathbf{p}^{(2)} = [p_0^{(2)}, p_1^{(2)}, p_2^{(2)}, p_3^{(2)}]$ .

due to second encoder is given by this. So I use notation p 1 to denote parity coming from the first encoder, p 2 to denote parity coming from the second encoder. u is my information sequence. u hat is the interleaved version of information sequence which is being fed to encoder 2.

(Refer Slide Time 47:37)

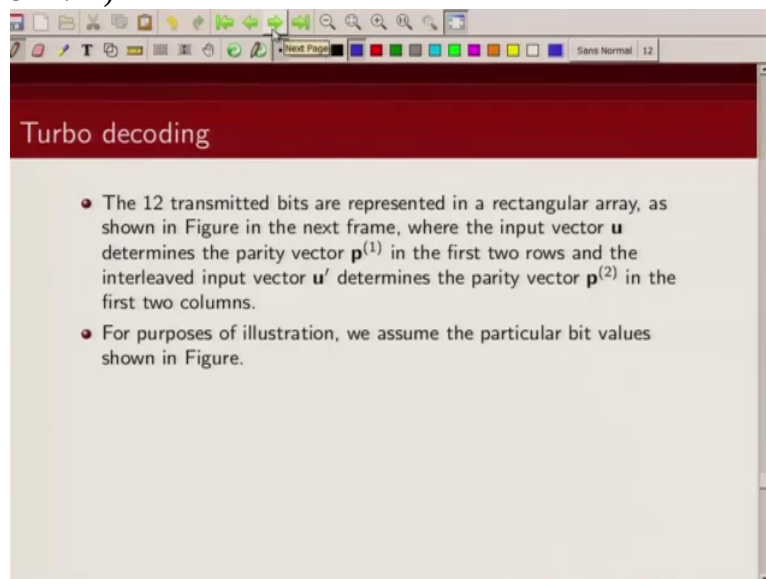


The screenshot shows a presentation slide with a dark red header containing the text "Turbo decoding". Below the header, there is a single bullet point:

- 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  $\mathbf{u}'$  determines the parity vector  $\mathbf{p}^{(2)}$  in the first two columns.

So I will just show you

(Refer Slide Time 47:41)



The screenshot shows a presentation slide with a dark red header containing the text "Turbo decoding". Below the header, there are two bullet points:

- 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  $\mathbf{u}'$  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.

basically I will just,



(Refer Slide Time 47:43)

**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  $\mathbf{u}'$  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

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

and I am assuming a channel 1 by 4 so the likelihood channel reliability factor I will see will be basically 1. So  $L_c r_l^{(j)}$  will be in my case would be same as the received value for this particular signal to noise ratio. This is just a toy example to illustrate how this decoding works.

(Refer Slide Time 48:08)

**Turbo decoding**

$u_0$	$u_1$	$p_0^{(1)}$	$p_1^{(1)}$
$u_2$	$u_3$	$p_2^{(1)}$	$p_3^{(1)}$
$p_0^{(2)}$	$p_2^{(2)}$		
$p_1^{(2)}$	$p_3^{(2)}$		

(12, 3) PCCC

-1	+1	-1	+1
-1	+1	+1	-1
-1	+1		
-1	-1		

Coded Values

+0.8	+1.0	+0.1	-0.5
-1.8	+1.6	+1.1	-1.6
-1.2	+0.2		
+1.2	-1.1		

Received L-values

So these are my information bits  $u_0, u_1, u_2, u_3$ . So as I said, if it is a zero, I am mapping it to minus 1. If it is plus, I am mapping to plus 1. So the information sequence here is  $u_0$  is 0,  $u_1$  is 1,  $u_2$  is 0 and  $u_3$  is 1. These are the corresponding

(Refer Slide Time 48:36)

The slide displays the following components:

- (12, 3) PCCC:** A trellis diagram showing the encoder structure with input bits  $u_0, u_1, u_2$  and parity bits  $p_0^{(1)}, p_1^{(1)}, p_2^{(1)}, p_3^{(1)}$  and  $p_0^{(2)}, p_1^{(2)}, p_2^{(2)}, p_3^{(2)}$ .
- Coded Values:** A 4x4 grid of values:
 

-1	+1	-1	+1
-1	+1	+1	-1
-1	+1		
-1	-1		
- Received L-values:** A 4x4 grid of values:
 

+0.8	+1.0	+0.1	-0.5
-1.8	+1.6	+1.1	-1.6
-1.2	+0.2		
+1.2	-1.1		

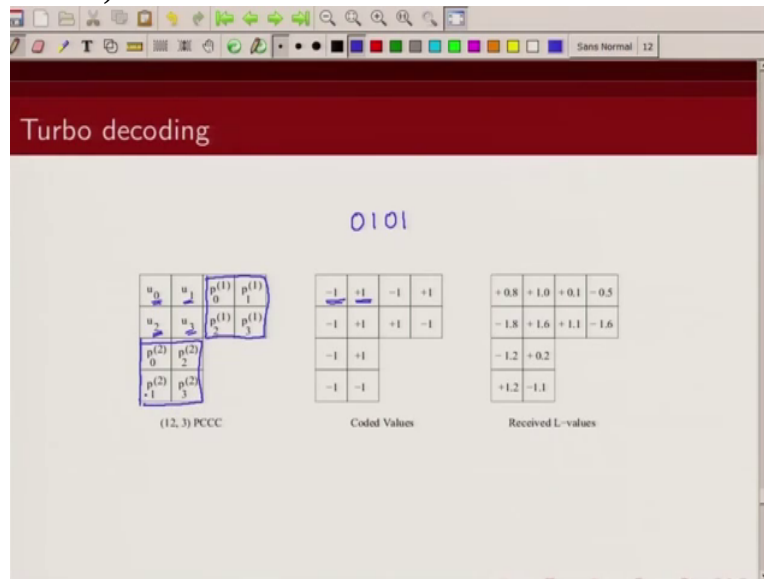
parities for

(Refer Slide Time 48:40)

The slide is identical to slide 48:36, but with a blue box highlighting the first two rows of the (12, 3) PCCC diagram, specifically the input bits  $u_0, u_1$  and parity bits  $p_0^{(1)}, p_1^{(1)}$ .

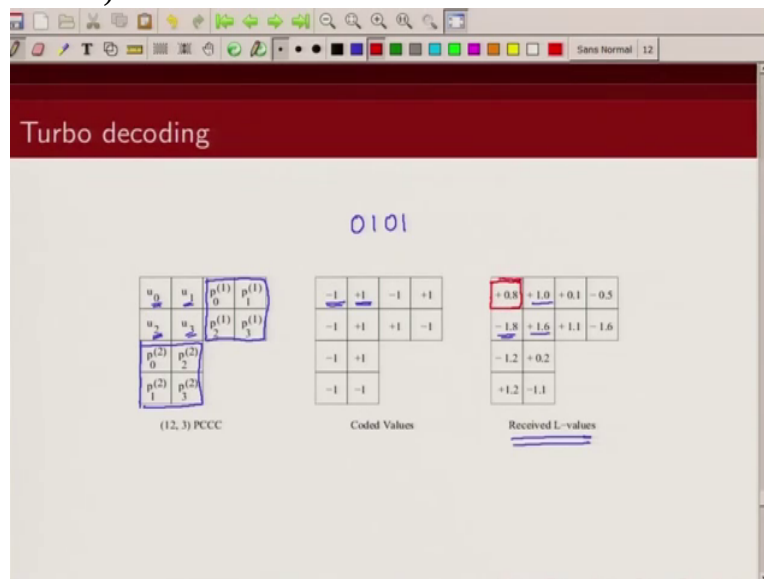
this information sequence from the encoder 1 and these are the corresponding parities from

(Refer Slide Time 48:49)



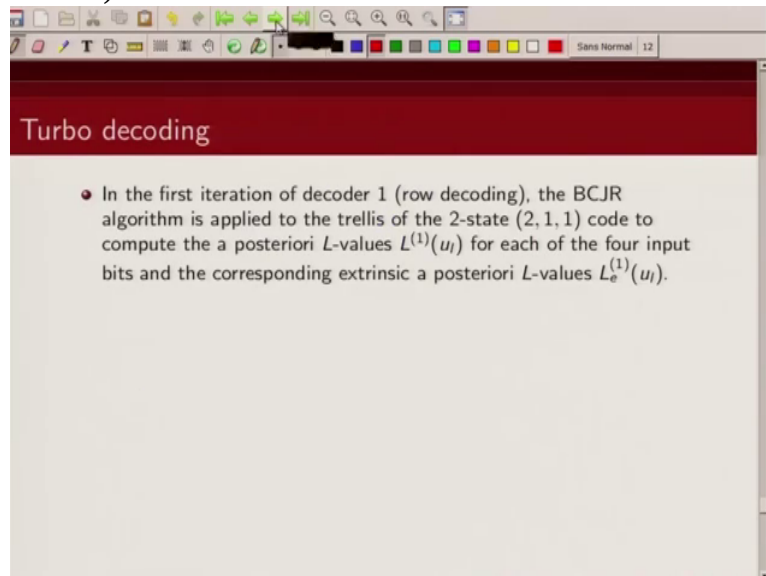
the encoder 2. Now what I have here is the received values. So you can see here, this was transmitted as plus, I received a plus. This transmitted a plus, I received a plus, this was transmitted minus 1, this was received minus but here the sign has changed. Note here this

(Refer Slide Time 49:13)



was transmitted as minus 1 but what I received is plus point 8 so this bit is received in error. Now let's see using this turbo decoding how we are able to correct this error.

(Refer Slide Time 49:32)



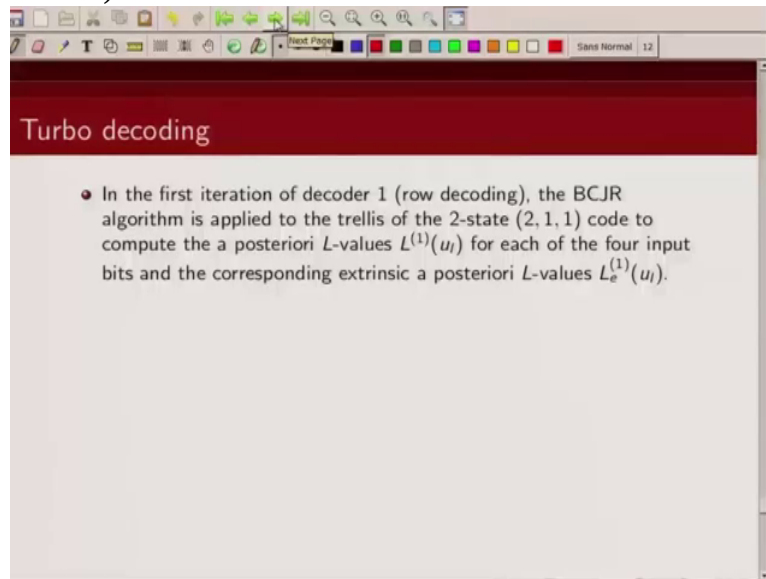
So of course, the first half of decoding

(Refer Slide Time 49:36)



will be, I will decode, I will have decoder 1 work first and then decoder 2.

(Refer Slide Time 49:44)

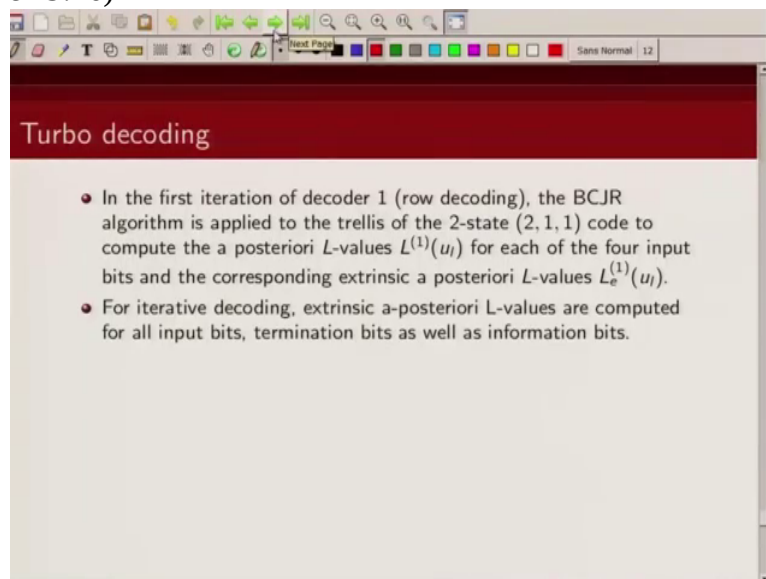


The screenshot shows a presentation slide with a dark red header containing the text "Turbo decoding". Below the header, there is a single bullet point describing the first iteration of decoder 1 (row decoding).

- 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 I will just

(Refer Slide Time 49:46)

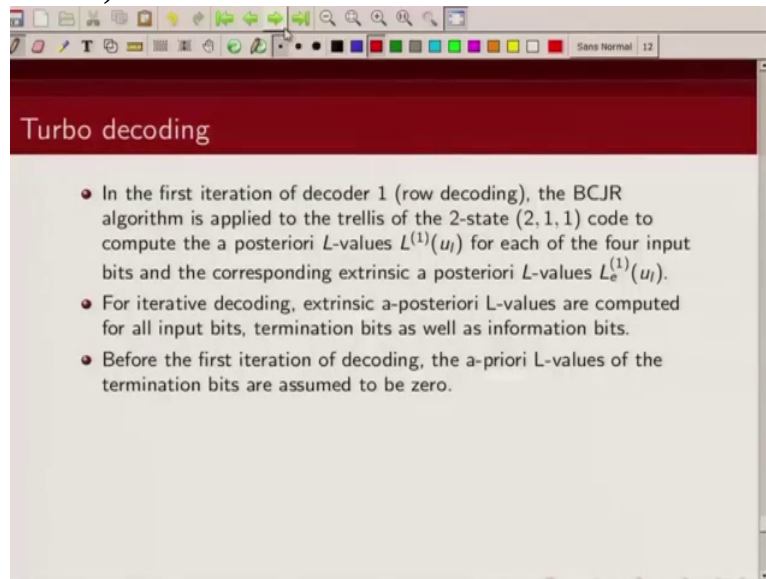


The screenshot shows a presentation slide with a dark red header containing the text "Turbo decoding". Below the header, there are two bullet points describing the first iteration of decoder 1 (row decoding) and the process for iterative decoding.

- 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.

directly come to the values

(Refer Slide Time 49:48)

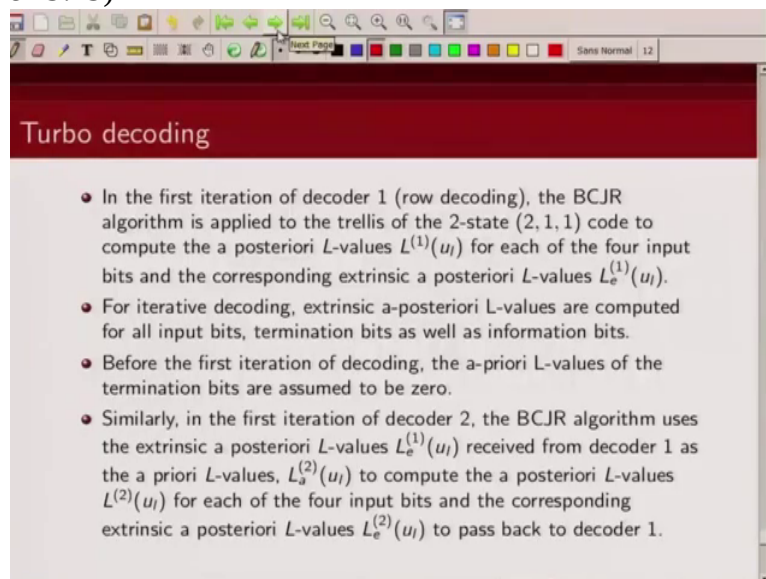


The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there is a list of three bullet points. The first bullet point describes the BCJR algorithm's application to a 2-state (2, 1, 1) code trellis to compute a posteriori L-values and extrinsic a posteriori L-values. The second bullet point discusses iterative decoding and the computation of extrinsic a-posteriori L-values for all input, termination, and information bits. The third bullet point states that before the first iteration, the a-priori L-values of termination bits are assumed to be zero.

- 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.

because I have already

(Refer Slide Time 49:49)



The screenshot shows a presentation slide with a red header containing the text "Turbo decoding". Below the header, there is a list of four bullet points. The first three bullet points are identical to those in the previous slide. The fourth bullet point describes the first iteration of decoder 2, where the BCJR algorithm uses the extrinsic a posteriori L-values from decoder 1 as a priori L-values to compute a posteriori L-values and extrinsic a posteriori L-values to pass back to decoder 1.

- 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,  $L_s^{(2)}(u_i)$  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.

explained the whole procedure. I will just show you

(Refer Slide Time 49:52)

**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.
- Decoding proceeds iteratively in this fashion.

the

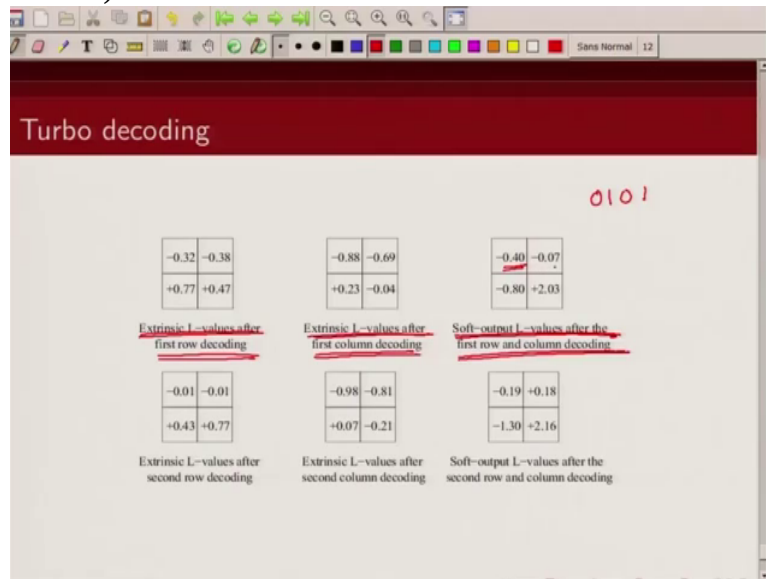
(Refer Slide Time 49:53)

**Turbo decoding**

$\begin{bmatrix} -0.32 & -0.38 \\ +0.77 & +0.47 \end{bmatrix}$ <p>Extrinsic L-values after first row decoding</p>	$\begin{bmatrix} -0.88 & -0.69 \\ +0.23 & -0.04 \end{bmatrix}$ <p>Extrinsic L-values after first column decoding</p>	$\begin{bmatrix} -0.40 & -0.07 \\ -0.80 & +2.03 \end{bmatrix}$ <p>Soft-output L-values after the first row and column decoding</p>
$\begin{bmatrix} -0.01 & -0.01 \\ +0.43 & +0.77 \end{bmatrix}$ <p>Extrinsic L-values after second row decoding</p>	$\begin{bmatrix} -0.98 & -0.81 \\ +0.07 & -0.21 \end{bmatrix}$ <p>Extrinsic L-values after second column decoding</p>	$\begin{bmatrix} -0.19 & +0.18 \\ -1.30 & +2.16 \end{bmatrix}$ <p>Soft-output L-values after the second row and column decoding</p>

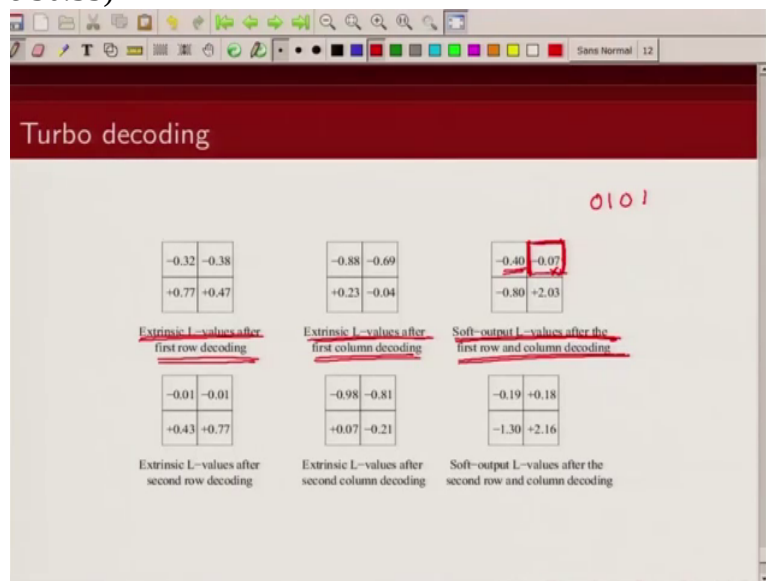
extrinsic information value and the a p p values at the end of each iteration. So this is the extrinsic information after I have decoded using decoder 1. This is the extrinsic information after decoder 2. And this is the a p p value after decoder 1. Note that I have transmitted 0 1 0 1. So there the sign is Ok, there

(Refer Slide Time 50:28)



the sign is not Ok. This is after first decoding, this is in error.

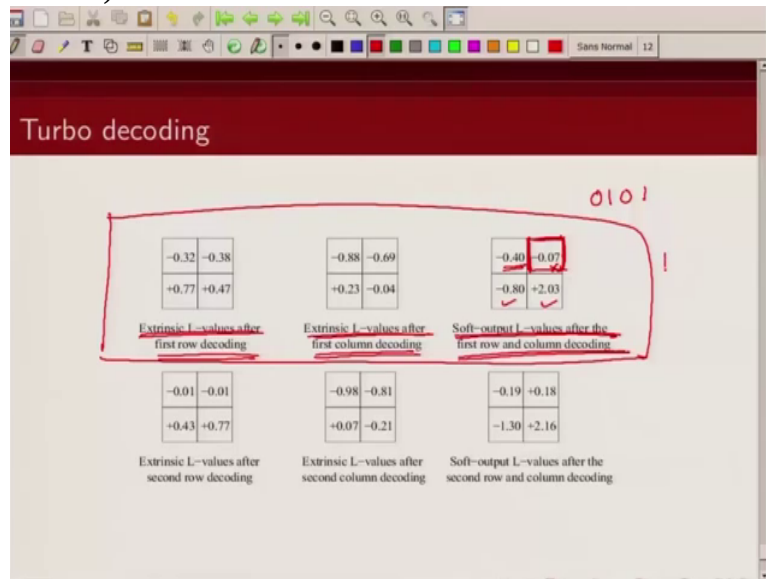
(Refer Slide Time 50:35)



This sign is ok. This sign is ok. Now next, so this whole thing was my first iteration. This is my first iteration,

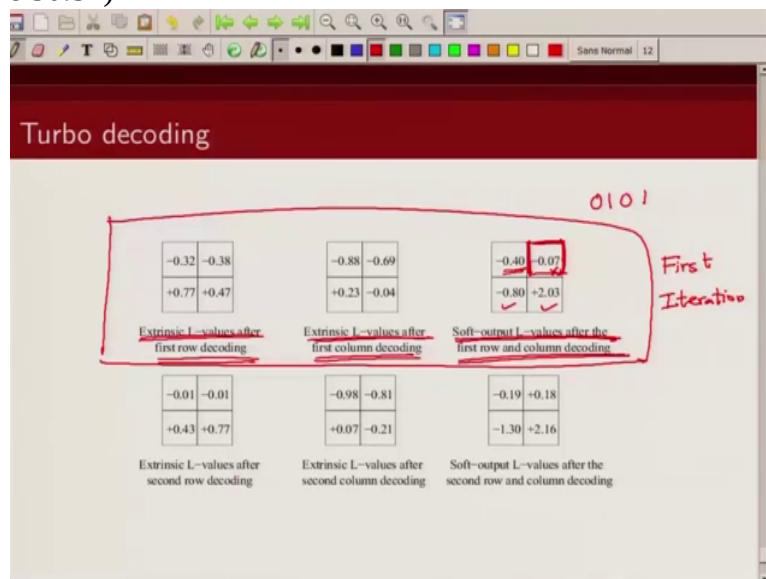


(Refer Slide Time 50:46)



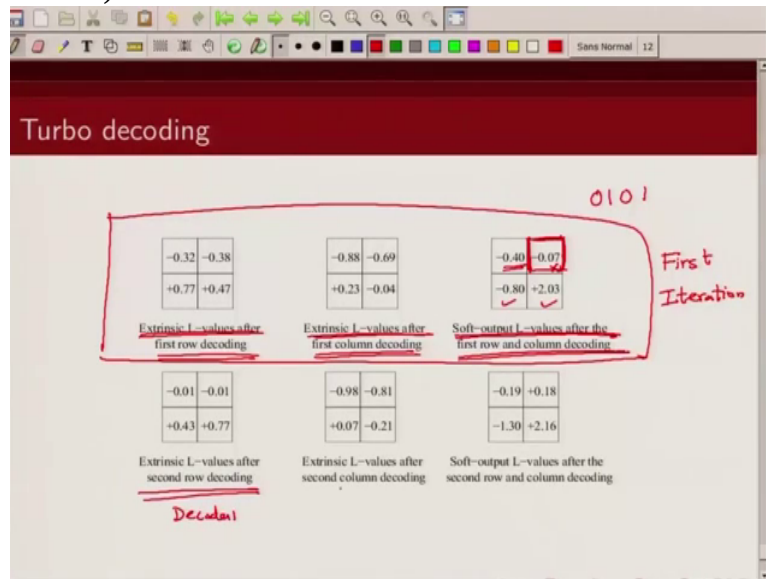
fine?

(Refer Slide Time 50:52)



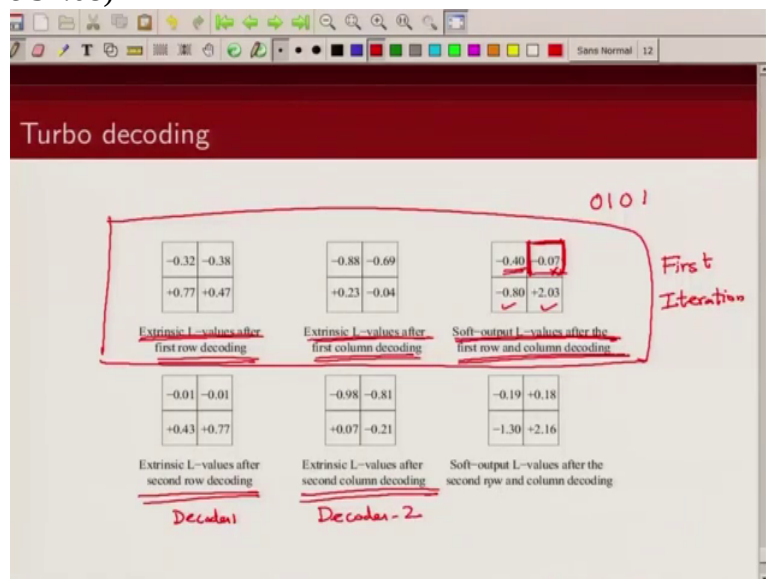
Now what about second iteration? So, so again I compute the extrinsic values, this is after I do decoder 1,

(Refer Slide Time 51:03)



this is extrinsic information after I have done decoder 2

(Refer Slide Time 51:08)



and this is the a p p value, L value, a p p L value after decoder 2 and note here this was 0, this is 1, this is 0, this is 1. So after 2 iterations I am able to correct the transmission error. So with this

(Refer Slide Time 51:17)

**Turbo decoding**

0101

-0.32	-0.38
+0.77	+0.47

Extrinsic L-values after first row decoding

-0.88	-0.69
+0.23	-0.04

Extrinsic L-values after first column decoding

-0.40	-0.07
-0.80	+2.03

Soft-output L-values after the first row and column decoding

First Iteration

-0.01	-0.01
+0.43	+0.77

Extrinsic L-values after second row decoding

Decoder-1

-0.98	-0.81
+0.07	-0.21

Extrinsic L-values after second column decoding

Decoder-2

-0.19	+0.18
-1.30	+2.16

Soft-output L-values after second row and column decoding

APP for value

I am going to conclude our discussion on

(Refer Slide Time 51:41)



turbo decoding, thank you