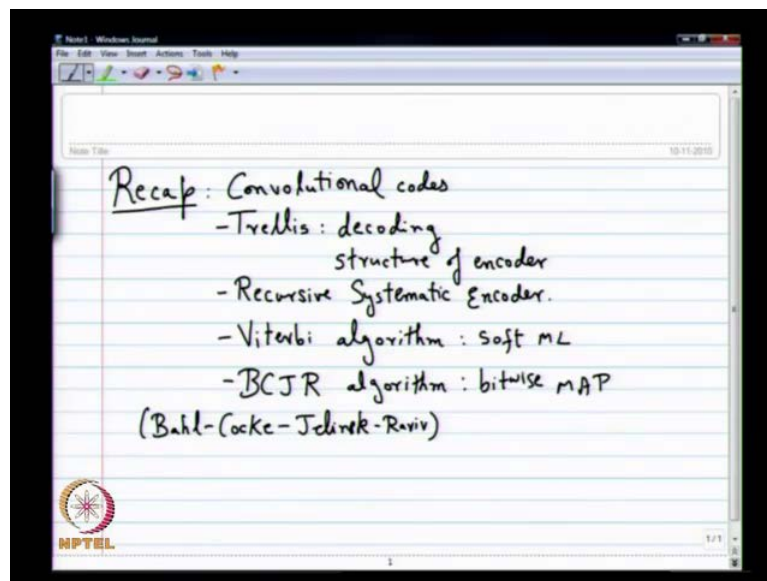


Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 34
BCJR Decoder

So, let us begin once again with a quick recap. So, the last thing we spoke about is probably the viterbi decoding algorithm, tail biting maybe collection of miscellaneous topics around convolution codes. Couple of things I want to highlight is this notion of trellis of course, trellis plays an important role. So, the trellis is an important idea so if you if you think about convolutional codes, you have to immediately think about the trellis that is the main idea.

(Refer Slide Time: 01:15)



Everything of around the convolutional code revolves around the trellis. Everything about the convolutional codes is both the trellis. It plays an important role in decoding of course; it also plays an important role in understanding the structure of the encoder. And we did not really spend too much time on this, but things like for instance minimum distance. The way I loosely define minimum distance is definitely quick to when we can find that out using the trellis in a much faster way than you would in any other way.

So, things like that are interesting and another interesting idea, which is gained a lot of prominence because of turbo code, is this recursive systematic encoding. And there are

several encoders, all of them are equivalent in the sense that the same set of code words are produced anywhere. So, they do not make any difference to the decoding performance. If you look at convolutional codes on its own 1 entity, when it turns out in turbo codes and other constructions, you use convolutional codes in concatenation.

When you use them in concatenation the structure of the encoder makes a difference and we will see that soon enough, but for that reason the recursive systematic encoder is particularly crucial. So, the properties that are interesting when you go to recursive systematic encoding is low weight code words. Low weight messages do not necessarily always give you low weight code words. So, in particular weight 1 message definitely gives you a very long code word and weight two also most of the time gives you a long code.

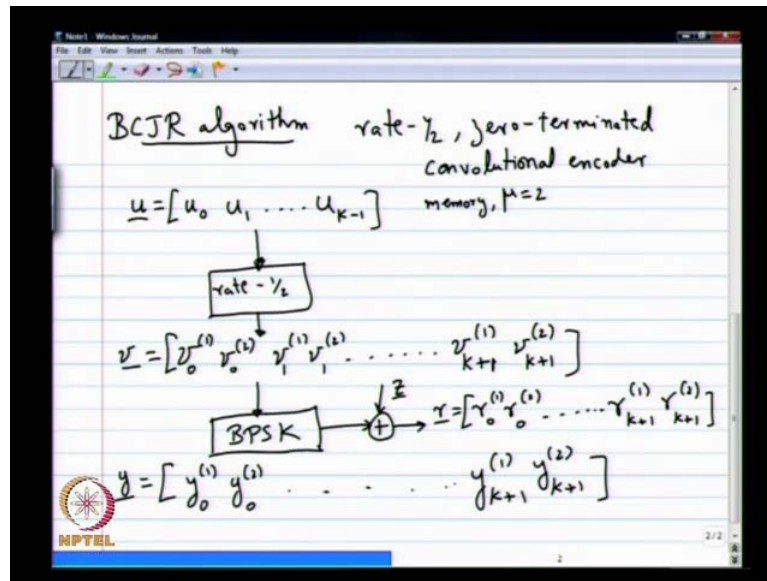
Only in some cases it gets stuck into a multiple of the polynomial that you use for the connection and then that case it works to a low weight code word. So, that is the main ideas that we spoke about and then of course, the viterbi algorithm. This is the ML decoder for convolutional codes. And what we are going to see next is called BCJR algorithm. So, let me try and see to way remember the names of Jelinik maybe EK and Raviv. So, Viterbi algorithm is the soft ML decoder, this is the bitwise MAP decoder.

So, it is named after the four inventors and we are going to describe the BCJR algorithm in this lecture. So, just like the Viterbi algorithm the BCJR algorithm also works on the trellis, while the viterbi algorithm makes 1 pass with trellis from the initial stage to the final stage and outputs the most likely code word or path on the trellis; that is what it outputs. What the BCJR algorithm does is, it will go it will make two passes on the trellis, it will go once from first to last and then from last to first.

And for that extra effort what you get in the bargain is not only do you get, you may not necessarily get the best part. What you get is what a post A or E log likelihood ratio for each message width; that is what you get from the BCJR algorithm, so which is what the MAP decoder does. What is the difference between ML and MAP, ML you just find the code word, which is closest to distance and bitwise MAP what should you find out? You get the LLRs for each bit given all the received words, received information. It is a bit more information and that plays an important part in the turbo decoder etcetera, etcetera.

So, if you have to only decode convolutional code and nothing else, what will you go with always? Load beta b you do not need anything else, but if you want decode the convolutional code as part of a larger system, you might want to use a bitwise MAP decoder because it produces what is called soft output. It produces probabilities on the bits not just the likely bits itself and those probabilities can be used further by another part of your decoder, which wants to improve etcetera, etcetera, so that is the idea.

(Refer Slide Time: 06:11)



So, let us talk about the BCJR algorithm, we will pick a rate half 0 terminated convolutional code, convolutional encoder. So, that is what we will do and I mean of course, the BCJR algorithm is more general than just rate half codes 0 terminated. Everybody has 0 terminated, but rate half is not as definitely needed for BCJR algorithm, but it is easy to describe and I will use rate half.

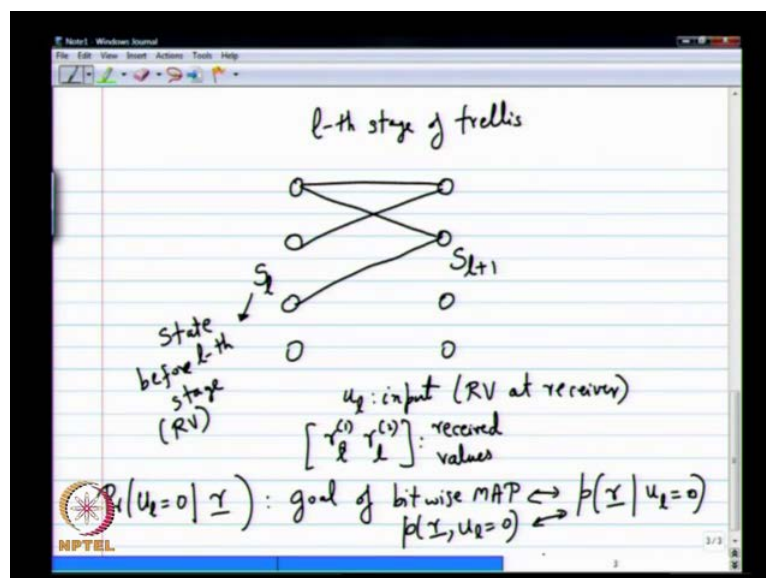
So, what will happen if you use rate half is, your message have been using M or u for message u maybe. So, you have let us say k message bits u_0, u_1 all the way to u_{k-1} . So, suppose you have a message u this is going to get encoded by the convolutional encoder like I said it is rate half. So, you will get code word. So, let us just say code word is c , so I get c_0 . So, I think v is what we used for the code word. We use v or c v maybe v so I will use v again v I probably put $v_1 \ v_1 \ 0$ is that $v_2 \ 0$ this is how it is.

So, this is the first stage and then for $v = 1, 2, \dots$ all the way to v . It is going to be $k + \mu - 1$. So, I will once again take memory to be 2 it is just again convenience of course, it does not work, but of course works for everything else. So, in that case the last stage will be $k + 1$ so it goes up to $k + 1$. So, that is going to be the code word and then we will do BPSK encoding, BPSK modulation and what comes out? I will maybe call y , y once again $y_0, 1$ so on. On way to $y + 1$ is that so what is gonna happen to this y , this will go through an AWGN channel, which will add a noise vector z and you will get a receive vector, which we will call maybe r .

I do not know it does not $r_1, k + 1, r_2, k + 1$. So, this is the setup for the BCJR algorithm that I will be using. So, of course do you can use it in a much larger setup than this BCJR algorithm, just like the viterbi algorithm works on any trellis you have a trellis BCJR algorithm would work. So, I am giving you a much simpler situation. So, for instance a standard place where you might again use it as an equalisation, if you have an ISI channel and you have a trellis corresponding to that you can once again use the BCJR algorithm nothing stops you from doing that.

So, as long as you have a trellis, you can use the BCJR algorithm, but I am going to use this specific example to make my life easier with notation and other things. So, once again the crucial thing is the trellis. So, what the values in the notations in the trellis if I look at the l -th stage of a trellis, there will be a corresponding trellis.

(Refer Slide Time: 10:48)



And that will have four states because our assumed memory is two. If the memory is not two then we will have larger number of states. I am drawing 1 particular example so you can in general have a different situation in the l - t h stage we are going to associate with the l - t h stage several things. First of all u sub all is the input corresponding to the l - t h stage and of course, this is not known at the decoder has to estimate this. And the output corresponding to the l - t h stage is of course, known to the decoder $r_{l,1}$, $r_{l,2}$.

These are the received values corresponding to something went wrong with my values that is better. So, these are the received values corresponding to that l - t h stage and there are some I guess some connections no matter how they look. So, I will just draw some arbitrary stuff here. So, this is how the connections it does not matter how the connections are.

So as far as, the decoder has concerned everything is a random variable u_l is a random variable, it does not know u_l , it only knows $r_{l,1}$ and $r_{l,2}$, it does not know u_l , and this would be a random variable at the zero. So, the goal in a bitwise MAP decoder is to compute probability of u_l equals 0. Let us say this is the goal of bitwise MAP, but usually nobody computes the probability directly. What do you compute? You compute the LLR so that is what you compute. So, well actually I am wrong in saying this, so let me just take this back and then write down exactly what the wants to compute. Of course, you want to compute probability that u_l is 0 given the entire received vector r ; this is what you want to compute.

If you do not give the received vector r of course, it is r there is no big deal in that question. So, given the entire r , but you do not compute this probability directly, what you usually compute? You compute the log likelihood ratio say let us say we will keep it like this for now. It does not matter what it is we will go the LLR later we will try to compute this probability, probability that u_l is 0 given r .

So, u_l like I said this is the input, this is a random variable at the receiver and for that you want to know, compute the compute the probability. So, let us add a few more random variables to describe what is actually happening. So, we will say the state before the l - t h stage so we will call that as some random variable. So, what can be a good notation for that so we will say let us say s_{l-1} it may seem to you. So, we will say the state before the l - t h stage, but then the problem will come when you do 0 t h

stage. So, let us say s_{l-1} is not a good idea we will do s_l , s_l is the state before l -th stage of trellis.

So, what will be s_{l+1} , it will be the state after the l -th stage. So, that will be that will again random variables. So, s_l is definitely a random variable at the receiver it is not known. So, there will be probabilities, there will be probability that f_l equals 0 probability that s_l equals 1 etcetera, etcetera. So, for any particular state there will be a probability. So, what we are going to do is we will try and write this conditional probability in terms of s_l also, we will bring in the states. So, why do we bring in the states?

It is of course, the most natural thing to bring in the trellis we know ultimately that the states contribute to the decoder a lot. So, what happened in the viterbi decoder clearly was because you could retain just 1 part per state at a given stage. So, the states are going to play a crucial role so it turns out likewise here in the bitwise MAP decoder also. The states will play a crucial role in the facilitating the computation so we have to bring that end.

So, before that in case anyways we are interested in the LLR, it is enough if you compute f of r given u_l equals 0. So, the reason is I mean if you do the LLR kind of computation you will see that there will be some priori probability coming in then that you know. You know definitely half ahead of times, so it will going to cancel and this calculation you will be really trying to compute p_d of r given u_l is 0. So, that is also good enough. So, keep that in mind so we might equivalently try and compute. So, I will use p just to make sure that I do not get into trouble here r given u_l equal to 0.

So, in fact this given u_l equal to 0 you can even say this is as good as trying to compute probability of r comma u_l equals 0 because what is the relationship between the conditional 1 you are dividing by probability. So, that is also the same thing so this is again equivalent to compute probability of r comma u_l equals 0. So, we will try to actually compute this guy r comma u_l equal to 0. So, that is the 1 that we will try to compute and then you will see that LLR is basically log of p of r comma u_l equal to 0 divided by p of r comma u_l equal to 1.

So, anyway for LLR you know this is good enough. So, if I do this for u_l equal to 0 and u_l equal to 1 then I am done there is no problem. So, that is what we will do so let us try

and concentrate on this and see how to simplify this and then we will maybe also simplify probability of r given r comma u l equal to 1. We will see that is also easy.

(Refer Slide Time: 17:54)

$$p(u_k=0, r) = \sum_{s' \rightarrow s: \text{labelled with input}=0} p(S_k = s', S_{k+1} = s, r)$$

Main result

$$p(S_k = s', S_{k+1} = s, r) = d_{k-1}(s') \gamma_k(s', s) \beta_k(s),$$

$$d_{k-1}(s') = p(S_k = s', r_0^{k-1})$$

$$\gamma_k(s', s) = P(S_{k+1} = s, r_k^{(0)}, r_k^{(2)} | S_k = s')$$

$$\beta_k(s) = p(r_{k+1}^{k+1} | S_{k+1} = s)$$

So, how do you simplify this guy probability of let me write u l equal to 0 comma r . This now I am going to bring in the states so this is the same as summation over all states s prime comma s such that what? So, I will write that down, but before that I will write this here. So, probability of s l minus 1 equals no s l equals s prime comma s l plus 1 equals s comma r . So, I want to break this expression up in as a summation over possible states s prime comma s and I want to say s l should be s prime and s l plus 1 should be s small s and small s prime.

So, what should s prime and s be? So, let me ask this question. So, basically the condition here will be that the branch s prime to s should be labelled with u l equal to 0 that is all. So, it should be labelled with input 0. So, when I say u l equal to 0 there are out of the eight branches, there will be four branches, which correspond to u l equal to 0 in the trellis. There are eight branches four of them will correspond to u l equal to 0 the other four will correspond to u l equal to 1. So, I am only going to look at those branches where u l equal to 0 happened and then look at the starting state and the ending state.

And then let s prime and s vary over those starting states and ending states and then I can write this probability as summation of this probability. So, if I want to compute this guy what I really need to compute is this fellow, so we will compute this in the BCJR

algorithm. Now, it is also clear what will happen if I want to compute probability of u_l equal to 1 comma r , how will this change? See you have to do a summation for s_{prime} as labelled with input 1 that is all. So, all I really have to do is compute what is inside this for all possible transitions s_{prime} comma s .

If s_{prime} to s is not connected by an edge I do not have to compute this you cannot really compute it will be like 0. So, no problem you do not have to compute it. But if a transition is possible from s_{prime} to s , I have to compute probability that in the l -th stage s_l was s_{prime} , s_{l+1} was s and the vector r was received. So, remember this p that I have written here is will be like a complicated distribution. It will be neither concrete nor continuous kind of distribution. So, it is a little bit you have to be careful when you deal with these things, but usually this is a very simple problem so it is not a big concern.

So, do not ask me what this p is, it is a p d f or a p m f. So, it is some distribution, it is a valid way of writing it we can you can write with c d f if you like, but this is a much simpler way of writing it. I am hopefully it is clear to you. So, this is what we will try to compute probability that in l -th stage, the previous state was s_{prime} , the next state was s and the vector r . So, once you compute this anything else you want to compute is done. So, how do you go about doing that is the question?

So, for that we once again use the trellis property and how in a trellis you naturally have several Markov chains, all your random variables will become Markov chains. So, that is the crucial idea, given that you are in a particular state in a particular stage all the past is totally irrelevant for the present and the future. So, that is the crucial property that is exploited in computing this. So, that is why this becomes easy so we use Markov chain properties of trellis to now simplify this expression. So, there will be a recursion thus developed and that is what will help you.

So, it turns out the main result or theorem or lemma or whatever it is the following relationship. You can show it is a simplification of this, you can show probability that s_l equals s_{prime} , s_{l+1} equals s comma entire vector r can be written as, I make sure I do not make a mistake here. There are enough cases of making mistakes here can be written as a product of three terms and these three terms, it is very common to call them

alpha, gamma and beta. So, what are the three terms, the first term is $\alpha^{l-1} s^{\prime}$, the next term is $\gamma^{l-1} s$ and then the next term is $\beta^{l-1} s$.

So, one can show that it can be written as a product of three terms. So, like I said this is the main result and this result will naturally give you the algorithm. So, of course I will have to tell what this alpha, beta and gamma is. So, just write it down like this and then you can say these two are one and then that is not the point is. You can write down, you can say that this $\alpha^{l-1} s^{\prime}$ this will be probability s^{l-1} equals s^{\prime} comma r , but r between 0 and 1 I have been writing it from 0 to $l-1$.

So, l will not be involved. So, when I raise this 0 to $l-1$ what it means is if I have a vector index from 0 to whatever, if I say 0 to $l-1$ you only take those indices it is like Mat lab notation. It will be r of 0 colon $l-1$ that is not actually Mat lab notation why 0 is not a valid index in Mat lab, but anyway it does not think of it as Mat lab notation in vectors. So, r from 0 to $l-1$ and then what is gamma, $\gamma^{l-1} s^{\prime}$ comma s is once again probability that s^{l-1} equals s comma r^{l-1} .

So, that is hopefully clear what I mean by r^{l-1} that is the output at the l -th point given that s^{l-1} . Did I make a mistake here? I think this is s^{l-1} , I am sorry no it is s^l . Next one will be s^{l+1} given s^l equals s . Thanks for pointing that out s^l equals s^{\prime} and then beta l of s the Americans say beta l of s is going to be probability r . So, 0 to $l-1$ was involved in the alpha expression, l was involved in the gamma expression and in the beta expression you are going to get windows doing something. So, r^{l+1} man it really wants to do something, what did I do?

It wants me to go to the next page. No I want to finish this off and then we will see $l+1$ see it is continues to think that I am writing something. So, I do not know why $l+1$ all the way to the end. So, the end is going to be what $k+1$ given s^{l+1} equals s . I managed that quite well. So, this decomposition is possible because of the Markov properties. So, we will call on a lot of Markov properties to enable this decomposition and you might wonder what the big deal in this decomposition is. You haven't learned anything new I mean it is not like anything has changed.

We had one probability expression and that is some product of three things and each of them involves anyway a long vector of r . May be the second gamma expression does not scare you too much. So, there is only one l that is involved, but alpha and beta both of

them involve long vectors r . So, is there any simplification is the natural question you can ask. So, we will first do this decomposition and then I will show you that it is possible to write down the recursions for alpha l minus 1 in terms of alpha l minus 2 and gamma l minus 1.

And then it is possible to write down the recursions for beta l in terms of gamma l plus 1 and beta l plus 1. So, once you have recursions like that you can recursively compute alpha and beta very efficiently so that is the point. So, this decomposition itself is not anything special it becomes special because of the recursions that are possible by for alpha and beta you might be able to write so many other decompositions using the Markov property. So why not, but this particular one is nice because of the recursions that are possible

Student: ((Refer Time: 28:25))

Which noise?

Student: ((Refer Time: 28:29))

Let me see I will write it down you see we will assume I think so. Yes, for this decomposition yes. So, let us see a quick proof of this, the proof is not too hard I mean it is quite straight forward just that the notation is a bit clumsy, but it is quite easy.

(Refer Slide Time: 28:54)

The image shows a digital whiteboard with handwritten mathematical equations. The equations are as follows:

$$\begin{aligned}
 \text{Pf: } p(S_l = s', S_{l+1} = s, \underline{r}) &= p(s', s, \underline{r}_0^{l-1}, r_l, \underline{r}_{l+1}^{k+1}) \\
 &= p(\underline{r}_{l+1}^{k+1} | s', s, r_l, \underline{r}_0^{l-1}) \cdot p(s', s, r_l, \underline{r}_0^{l-1}) \\
 &= p(\underline{r}_{l+1}^{k+1} | s) \cdot p(s, r_l | s', \underline{r}_0^{l-1}) \cdot p(s', \underline{r}_0^{l-1}) \\
 &= p(S_l = s', \underline{r}_0^{l-1}) \cdot p(S_{l+1} = s, r_l | S_l = s') \\
 &\quad p(\underline{r}_{l+1}^{k+1} | S_{l+1} = s)
 \end{aligned}$$

The whiteboard also features the NPTEL logo in the bottom left corner and a status bar at the bottom with the text '3/5'.

So, s_l equals s_{l+1} and then the entire vector r this is what I want to write down and I want to use some Markov type property. So, the first step is to first write down, so instead of writing s_l equals s_{l+1} equals, I will simply write s_{l+1} comma s_l . So, I will say first thing is s_{l+1} means s_l equals s_{l+1} , second one is s_l means s_{l+1} equals s_l . So, just to make my notations I will first simplify split r as going from 0 to $l-1$ and then the r_l , I will write r_l .

So, r_l is basically these two guys together is denoted r_l all there are two outputs you get in your l -th stage denote it as r_l and then I will have r_{l+1} to $k+1$. Now, you start gathering things, which are in the past and in the future and then do some conditioning and see what happens, that is the idea. So, this you can write as probability of r_{l+1} to $k+1$ given everything else. So, everything else is what? s_{l+1} r_{l+1} r_0 to $l-1$ then you should multiply by all this case s_{l+1} r_{l+1} and r_0 to $l-1$. Now, what is an immediate simplification in the first term?

So, remember like I said noise is IID so noise is not dependant on the past and given that s_{l+1} was equal to small s , all other information you have is totally irrelevant to r_{l+1} to $k+1$. It does not matter what happened in the past as long as you got to this state s and the $l+1$ th stage, I mean after stage l only that matters. So, in the first term you can clearly drop this one, this one and this one, so you can drop these things. So, now let us look at this guy so here I am going to now write it as times. Now once again what is the most recent thing in this r_l , r_l is the most recent.

So, let us write it as and s_{l+1} also, s and r_l happen at the like the last stage anyway from your guess you are all experts in reworking the solution. So, if you look at the solutions you need to get r_l given s_{l+1} r_0 to $l-1$ times probability of s_{l+1} r_0 to $l-1$. Now once again you use your Markov thing and say if at s_{l+1} before the l -th stage all these guys do not matter in the in s_n . So, that is totally determined only by s_{l+1} so I can conditionally make it independent.

Remember these are all conditional independences, not totally independent. Only given all these things, the thing becomes independent. So, now we have the final form and we can easily write it. I will write this last term first because that is what gives me my alpha term $p_{s_l} = s_{l+1}$ comma r_0 to $l-1$ times the s_{l+1} equals s comma r_l

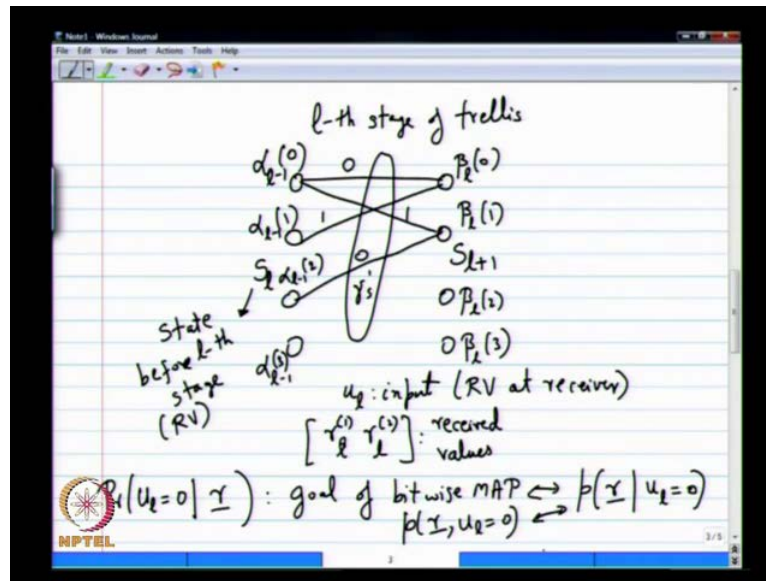
given s_l equals s prime times, what is the last term r_{l+1} to $k+1$ given s_{l+1} equals s .

So, like I said it just uses basic chain rule and probability and the Markov condition with the trellis, the trellis plays the key role as you can see, so it is important. So, actually if you have any Markov chain, this BCJR computation will apply. So, if you have any Markov chain where the state is hidden to you, so you can think of it as a hidden Markov model if you want. The state is hidden to you what you observe is the noisy version of something that denotes the state transition, which gives you probability for states transitions.

So, any such situation you have you can use your BCJR algorithm. So, only property you are using clearly is the Markov property nothing else matters. Of course, the independence of the noise from time to time also is used noise were dependant clearly I cannot throw out all of these things. So, that is the proof for the main result. So, once you have a proof for the main result, the problem of computing this term becomes equivalent to computing $\alpha_{l-1} \gamma_l \beta_l$. So, how can we compute each of those things is the question? γ_l is the easiest thing I will do that first because it involves only values in the state in the l -th state.

α and β we will need recursions so that is a very easy recursion, which you can once again derive for α and β and you can write that down easy. So, it is also good before we do the all those things to see this formula in the trellis. So, how do you see this formula in the trellis, in the l -th stage you are going to have $\alpha_{l-1} \gamma_l \beta_l$.

(Refer Slide Time: 35:16)



This will be $\alpha_{l-1}^{(i)}$, those will be the probabilities that you can associate with those states $\alpha_{l-1}^{(0)}$, $\alpha_{l-1}^{(1)}$, $\alpha_{l-1}^{(2)}$, and $\alpha_{l-1}^{(3)}$. And here with this state what will you associate? Well not alpha, alpha also you can associate, but that does not play any role in the l-th stage. So, what is most natural beta, you can do $\beta_l^{(0)}$, $\beta_l^{(1)}$, $\beta_l^{(2)}$ and $\beta_l^{(3)}$ and then you might have some transitions. So, this might be 0, this might be 1, this will be 1, this will be 0. So, for each transition, each branch you have a gamma.

So, look at the gamma, gamma depends see the alpha is associated with a particular state in the previous state, beta is associated with the next state. What is gamma associated with both s' and s means it is associated with a branch. So, on each branch you can put a gamma. So, the some gammas will come here. So, let me just put the whole thing there and say gammas. So, how do I compute the final probability, this probability how do I compute this probability?

How do I compute this guy on the trellis? If I take a particular branch and multiply the alpha on the left hand side and the beta on the hand side with the gamma and the branch. So, I take the each branch, take the alpha beta multiply them with the gamma on the branch, I get the probability that I want probability of s' comma s comma the entire vector r . So, how will I compute probability of r comma $u_l=0$, I compute this quantity for all the branches corresponding to 0 and then add them. So, it is the same thing for probability of r comma $u_l=1$.

I compute the probabilities the products for all the branches corresponding to input 1 then add them I get. So, clearly the crucial thing is alpha, beta and gamma and they are all naturally associated with the trellis nodes. So, of course you are also there is alpha l on this side also, but then that is for the next stage, so valid only for the next stage. I am not going to use that alpha in this stage so it is most natural to think of alpha from on the left and beta on the and gamma on the branches.

And my basic probability of the branch given the comma, the entire received vector is basically the product of alpha, gamma and beta that is how you think of that product. Product is the probability of that branch comma the entire received vector r. So, I add up all the branches together, which have input 0 I get the probability of r comma u l equal to 0 etcetera, etcetera so that is crucial.

(Refer Slide Time: 38:38)

α, β - recursions

$$\alpha_l(s) = \sum_{s'} \gamma_l(s', s) \alpha_{l-1}(s')$$

S_{l+1} S_l

s' : $s' \rightarrow s$ is a branch

$$p(S_{l+1}=s, Y_0^{l-1}) = \sum_{s'} p(S_{l+1}=s, S_l=s', Y_0^{l-1}, Y_l)$$

$$= \sum_{s'} p(S_{l+1}=s, Y_l | S_l=s', Y_0^{l-1}) \alpha_{l-1}(s')$$

NPTEL

Now how do you efficiently compute alpha and beta? So, you have recursion so the alpha, beta recursions. So, the first formula to speak is this formula for alpha, you can show alpha l s is summation over all S prime, gamma l S prime comma s alpha l minus 1 s prime. Remember this is capital L capital S l l plus 1, so this is S l. So, there is like a shift in the way this notation goes and this guy s prime comma s is from S l to S l plus 1. I am not going to prove this recursion, it is not too difficult. It is basically based on the expression for probability, what is this expression? This is probability of S comma.

Student: ((Refer Time: 40:09))

$s_l + 1$ is s comma r from 0 to 1 that is this expression. So, basically what you have to do is once again use Markov property, I want to use $\alpha_l - 1$. So, what I will say is the first step you can easily write down. Maybe I will quickly do the proof here we do have time. So, you do summation over s prime probability of $s_l + 1 = s$ comma $s_l = s$ prime comma you split that as 0 to $l - 1$ and r_l . So, this is the first step that you do and then what is the next step.

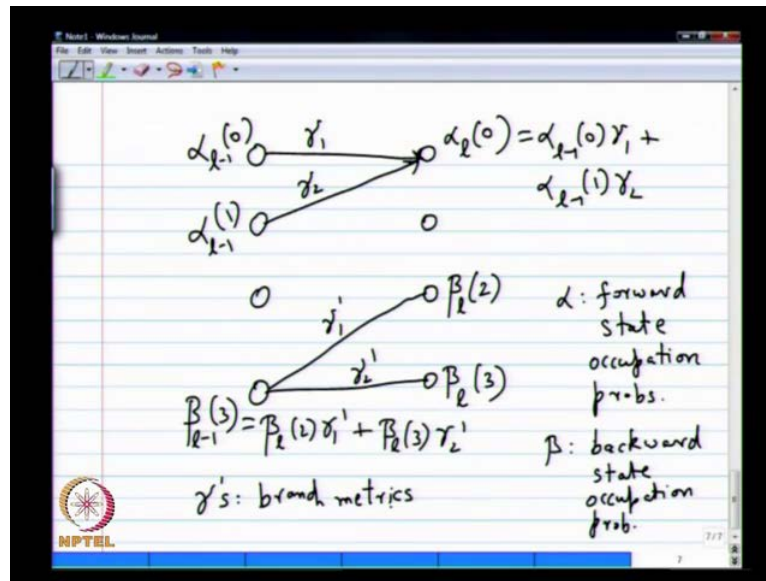
So, you can see that I mean you can easily figure it out from the formula that is given their s prime. I am going to now use Markov property. So, I need an α here so I am going to condition on that, I am going to write given $s_l + 1 = s_l = s$ prime comma r 0 to $l - 1$. So, I have to write down here $s_l + 1 = s$ comma r_l times I would have already got the $\alpha_l - 1$ of s prime that is it. And then this guy is basically once you know s_l is s prime, this you can drop and that becomes the γ that I want.

So, it is basically the Markov property just gives you this recursion. So, once again it is interesting to look at this in terms of the trellis. So, if you interpreted in terms of the trellis, this recursion looks a little bit artificial now it seems like why do I do all this s prime etcetera. Remember if you want to compute α_l of s the only s prime that is relevant is.

Student: ((Refer Time: 42:10))

Ones, which lead to s if you are not connected to the s prime, you do not have to include that here. So, let me put this here s prime such that s prime to s is an edge is a branch. So, it is a valid branch I have to only those things I mean if I could not have to come to s from that previous state, there is not going to be any probability distribution. You only have to take that. So, in terms of the trellis it is particularly easy to understand this.

(Refer Slide Time: 42:38)



Definitely only branches are considered so prime to s is labelled with input potentially it is a branch. So, here once again the trellis interpretation for the alpha recursion is quite easy. If you look at alpha let us say 1 of 0, so this might be connected from here and then from here, they will be a like a gamma 1 here and there will be a gamma 2 here. And this will be alpha 1 minus 1 of 0 and alpha 1 minus of 1 and what will be this alpha 1 of 0? It will simply be alpha 1 minus 1 of 0 times gamma 1 plus alpha 1 minus 1 of 1 times gamma 2.

And I write it as a formula it seems like a very fancy formula, but essentially what I am saying is I know the probability of being in state 0 and state 1. Well when I say it is not the probability of being at state 0 and state 1, the definition is probability of state comma all the received values it is not given. So, it is not given it is comma, it is and, it is the joint probability. So, once I know these probabilities how I find alpha 1 of 0 simply take that by multiplying the corresponding gamma that leads you up there and add it up with the other contributions that you have.

So, it is a kind of like natural recursion that you get on the trellis. There will be a similar natural recursion for beta also, maybe I will do with something like this maybe this goes here and it goes here. So, if you want to look at beta 1 of 2 and then a beta 1 of 3 and if you want to compute beta 1 minus 1 of 3 and if this is like say gamma 1 prime gamma 2

prime, what will this be? This will once again be the natural formula that this is β_1 of $2 \times \gamma_1$ prime plus β_1 of $3 \times \gamma_2$ prime.

So, I can write down the beta recursion more formally also, but this is what you want to remember. So, this is how you can easily remember the recursions. Otherwise if you write down the Markov formula every time also you can do it, but it is a little more confusing, but this is a very easy way to remember the recursion. Once you know the gamma computation the alpha recursions are and beta recursions are easy. So, how do you start the recursion? So, any recursion has a starting point what is the starting point for this recursion. So, if were the 0^{th} stage then what do I know?

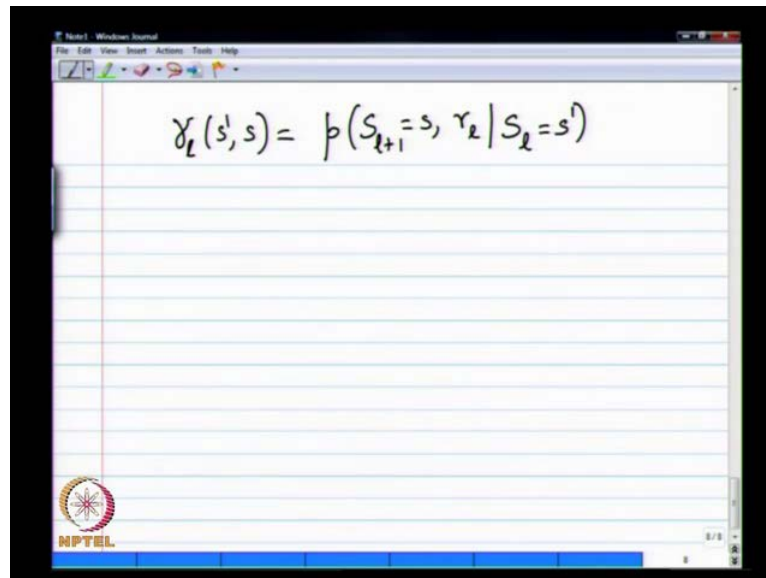
So, you always start at the all 0 state. So, only one state will be probable with probability 1, all the other states will be probability 0. So, then what do you do? At the 0^{th} stage on the left, you only put probability 1 for this and everything else 0. What about beta? Beta likewise I have 0 terminated of the $k - k + 1$ stage only the 0 terminated state 0 state will have probability 1, everything else will have probability 0.

So, these alphas are also called forward state occupation probabilities and this is a good word because it gives you a nice feel for what they actually represent? Forward state occupation probability is what is beta? So, let me not write it there, write it there. What is beta? Beta is backward state occupation probability. So, alpha is like the probability that you are in Maharashtra and beta is the probability that you are in Bihar. So, that is another way to think about this forward state, backward state.

If too many people these kinds of comments when you go in one direction you can compute alpha, you have to go in the forward direction, then computing beta you have to come back in the backward direction. So, that is what you do, what about gamma? Gamma is neither forward nor backward. So, it is a stage so it corresponds to the branch probabilities for each stage. So, gammas are like branch metrics. I do not want to write down a formal formula for beta, you know how it will look?

How will this look? You simply change alpha to beta here and then instead of $1 - \alpha$, you put $1 - \beta$ here and α there that is all, so you get the exact same and s prime and s you have to reverse. So, you are gonna get the same formula. So, the last thing is gamma and that is crucial, that is a little bit important so let us see. I want you to pay some attention here because it plays an important role in the turbo decoder and all that.

(Refer Slide Time: 48:12)


$$\gamma_l(s', s) = p(s_{l+1} = s', r_l | s_l = s)$$

So, what is gamma? Gamma l s prime comma s is probability that what is the formula s l plus 1 equals s prime comma s l equals s.

Student: ((Refer Time: 48:32))

Comma r l and given there is an s prime.

Student: ((Refer Time: 48:42))

Given s l equals s. So, many mistakes that should work out and we have about a minute left. So, let me stop here. So, in the next lecture I will pick up from here, I will talk about how gamma is computed and that will kind of conclude the BCJR algorithm.