



LDPC and Polar codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology, Madras
Modifications to the Decoder: Layered Decoding and Offset

(Refer Slide Time: 0:17)





Decoder Modifications

- Offset and scaling in minsum
 - $a(\text{minsum}) + b$
 - Makes minsum as good as log-MAP
- Layered decoding
 - Faster convergence, fewer iterations
 - Codes in standards often support layering
- Most implementations
 - Offset, scaled minsum with layering

Modifications to the Decoder: Layered Decoding and Offset

Hello, in the previous lecture we saw a toy example an illustration of what happens in the LDPC decoder, we use the storage matrix idea and then did draw operations, column operations. So hopefully you agree with me that the whole thing is very simple and direct there is nothing very complicated or magical in the decoder except the magical aspects are how we use the local structure, very efficiently to do simple operations and improve our believes and how we combine different messages coming from the local structure iteratively in a very smart way using very extensive information to do more and more iterations, okay so these are the two simple ideas which give really powerful performance.

(Refer Slide Time: 0:56)



Decoder Modifications

- Offset and scaling in minsum
 - $a(\text{minsum}) + b$ $\text{minsum} - \text{offset}$
 - Makes minsum as good as log-MAP
- Layered decoding
 - Faster convergence, fewer iterations
 - Codes in standards often support layering
- Most implementations
 - Offset, scaled minsum with layering

PROF. ANDREW THANGARAJ
@THANGARAJ Modifications to the Decoder: Layered Decoding and Offset

But you can see why the sparse structure is important, if the structure is sparse we do not have too much dependency, right as you keep doing iterations and you can do more true updates of your believes and you can do better, okay. So now in this lecture we will see couple of important modifications to the decoder actually one main modification which is a layered decoder along with that we will see a modification to the minsum approximation which is offset and scale, okay.

So it turns out minsum is pretty useful approximation, now to improve that approximation people typically use offset, okay. So instead of just taking the minimum of the two value, you take some constants a and b multiply with that and add b, okay. So typically scaling is not something we will use, we will simply use minimum minus some offset, okay so this is very very popular in practise, you take the so you remember we did this approximation of log 10 hyperbolic function with the min, right the sum of two exponentials approximated as min. It turns out you can subtract a small offset from it and make it more accurate, okay.

So this is something that is done, okay so this is very common modification and in our implementation also may be we will allow for some modification like this, okay it is a small subtraction, okay. The most important modification and practise is this layered decoding, okay. So what we saw before was this row operation and then a column operation and then a decision, okay.


So typically in practise people do layered decoding where you update to the column operation more aggressively, okay so you do not wait for all the rows to finish before doing the column

operation. You take a set of rows or a layer of the parity check matrix once you are done processing that layer we immediately update the column, okay. So it turns out this has faster conversions, fewer iterations (and you have) but you did design the code in a way which will support layering, okay.

So it turns out this proto graph expansion things are very good for that too, a nicely support layering each expansion row each expanded row naturally acts like a layer, okay and that is you can design it so that it supports it better and all standards codes in the standards will support this. So there is some minor instability from the layered decoding but it can be handled it usually does not show up in the kind of error rates we are interested in, okay.

So most implementations will use offset, scale but even scale is not too much, okay offset minsum with layering is very very standard implementation on most decoders for LDPC codes, okay. So let us see how that layered decoding works, once again we will not see any major theory just the example and see how it modifies from the previous one, we will sort of ignore the offset, I will not show you the offset right now I will show you later on in implementation how to add that offset into your decoder.

(Refer Slide Time: 3:50)




Layering in the LDPC matrix

- Rows of parity check matrix grouped into several layers
- Column weight of each layer = 1 (typically)

one block row in 5G LDPC codes

$$H = \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix} \begin{matrix} \text{Layer 1} \\ \\ \text{Layer 2} \end{matrix}$$

- C_1 : Code with H_1 as parity check matrix
- C_2 : Code with H_2 as parity check matrix
- C_1, C_2 : LDPC codes, Code = C_1 intersect C_2




Okay, so how does layering work? So you group rows of a parity check matrix into one layer and after you finish that layer you do the column operation, okay so that is sort of the idea, okay. So when the column weight of each layer is usually kept as 1 it is possible to increase that also and do some more complicated arithmetic there but we will skip that we will only

look at column of each layer as 1 and if you go back to the 5G LDPC matrix specification we will think of each layer as 1 block row in the standard in say the 5G LDPC codes, okay.

So that is how we will think of $(4:33)$ but once again we will not use a 5G code in our example it is just too much it becomes very big I will take a simple toy example. So here is the toy example I have split my parity check matrix into two layers, layer 1 and layer 2 and these are the two layers, okay you can see the first layer has just a single 1 in each column, okay column weight is 1, similarly the second layer also has a column weight 1 these two together specify the overall parity check matrix, okay.

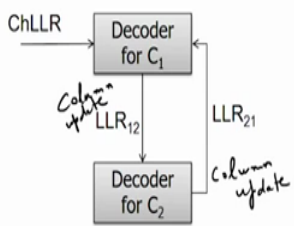
So the way the philosophy behind once again this kind of decoding is you think of the overall parity check matrix as being composed of these two layers, okay so you have a code defined by the first parity check matrix H_1 , another code defined by the second parity check matrix H_2 and you sort of think of them as decoding the first code and the second code and iterating between the two for improving, okay so that is the idea there is slightly there are more involved details here but we will skip that, okay.

(Refer Slide Time: 5:31)




Layered decoding

- Iterate between decoders for each layer



- Decoder for C_1 uses ChLLR only in first iteration
 - LLR_{21} is used afterwards
- Reduces required number of iterations by 1/2
 - Converges faster




PROF. ANDREW THANGARAJ
IIT MADRAS

Modifications to the Decoder: Layered Decoding and Offset

So this is the layered decoder and to understand this better we are going to see a toy example small example of length 7 code with a very small parity check matrix of course when we actually implement the 5G code it will be much larger but the steps are the same and you can study it in the small example and easily extrapolate to the larger one, okay so let us get started.

(Refer Slide Time: 5:52)



Toy Example for Illustration

Initialization


$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Layer 1 → col wt = 1
Layer 2

$r = [0.2 \quad -0.3 \quad 1.2 \quad -0.5 \quad 0.8 \quad 0.6 \quad -1.1]$

$L = \begin{bmatrix} 0.2 & -0.3 & 1.2 & -0.5 & 0.8 & 0.6 & -1.1 \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \end{bmatrix}$

in coming received vector
0's in PC matrix only layer 1



PROF. ANDREW THANGARAJ
@IITMADRAS

Modifications to the Decoder: Layered Decoding and Offset

This is the toy example I will be using this is my parity check matrix we saw this just before this is so we saw this example earlier as well the matrix is divided into two layers, this is layer 1 and this is layer 2, okay so rows 1 and 2 are in layer 1, rows 3 and 4 are in layer 2. So if you notice within a layer the 1's do not overlap as in so you the column weight is 1, okay so you see that that is true that is maintained here.

So everything gets updated in that fashion so that is an important rule to remember in each layer column weight will be 1, okay so there is no question of updating the same bit twice within the layer, okay so in fact there are extensions for those kind of decoders as well but we are not going to look at it in this class, we will only consider a case where each layer has a column weight of at most 1, there may be columns with weight 0 also but in this example it is not there, okay alright.

So this is the parity check matrix, this is the received vector r is the incoming received vector, okay. So one easy way to think of layer decoding is you start with this received vector and start going down the parity check matrix row by row. As you process each row you keep updating this received vector, okay so that is the very simple way of visualizing the layer decoder and you can also think of the received vector is going layer by layer, okay and in each layer some processing occurs and the received vector gets updated and you keep proceeding further and further, so keep this picture in mind so you keep looking at the received vector, then you process row by row and as you process every row you keep updating the received vector and you go forward, okay so that is how it will work, okay.

So I have my received vector and you remember this L matrix the matrix in which we store all the LLR's the message LLR's and if you remember wherever you have 1's in the parity check matrix we store something, wherever you have 0's in the parity check matrix you do not store anything, okay. So what I have done here is I have put blue wherever you have 0's the blue corresponds to 0's in the parity check matrix, okay.

So wherever you have 0's I am not going to store anything, okay so I have just blocked that out and put a blue box there to show that we are not using those locations, only in the other locations we will store it, okay. So the first step is initialization and if you notice here the difference between the non-layered version and the layered version is in the layered version we will only use the first layer in the beginning, okay so we will not use the second layer at all when we start with.

So the initialization just goes down and every position is initialized with the received vector, okay so that is what happens here so you see wherever you have 1 the received vector sort of goes in and sits there, right so that is the way to think of it and since column weight is 1 every received vector will sit in only 1 position, okay so that is something that you can work out very easily, okay so this is how it starts so this is the initialization.

In later iterations there may be already some value stored in L, okay in that case we have to do something different but in the first iteration we just initialize and store like this, okay. So I will comment on this later iteration when we come to it, okay alright. So this is the first step hopefully this is clear and notice once again we do not do any initialization for layer 2, okay so only layer 1, okay the initialization (9:29) has worked only in layer 1 so far and layer 2 we will come to it later, okay.

(Refer Slide Time: 9:36)

Iteration 1, Layer 1

NPTEL

Prof. Andrew Tanigara

Modifications to the Decoder: Layered Decoding and Offset

Overall parity

Minsum on first layer: Rows 1, 2

input bits

output bits

add to update

$$L = \begin{bmatrix} 0.2 & -0.3 & 1.2 & -0.5 & 0.8 & 0.6 & -1.1 \\ - & - & - & - & - & - & - \\ - & - & - & - & - & - & - \end{bmatrix} \begin{matrix} - \\ + \\ - \end{matrix}$$

$$r = \begin{bmatrix} 0.2 & -0.3 & 1.2 & -0.5 & 0.8 & 0.6 & -1.1 \\ -0.3 & 0.2 & -0.2 & -0.6 & -0.2 & 0.5 & -0.5 \\ - & - & - & - & - & - & - \end{bmatrix}$$

$$\text{sum} = \begin{bmatrix} -0.1 & -0.1 & 1.0 & -1.1 & 0.6 & 1.1 & -1.6 \end{bmatrix}$$

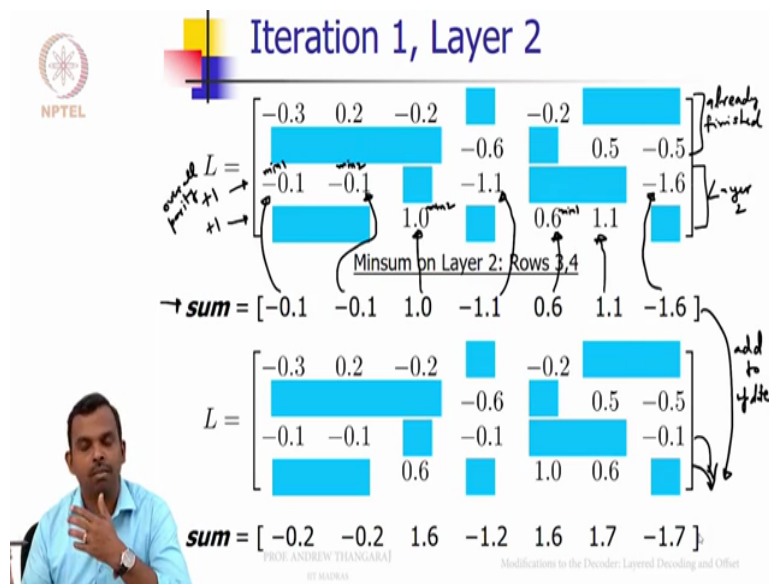
So after we initialize we jump into the iteration this is iteration 1 layer 1 and we saw that L was initialized in this fashion these values went and sat there and now we do row processing, how do we do row processing? We use minsum, okay as always we use minsum. So if you look at the first row this is min 1, this is min 2, okay an absolute value and the overall parity is not satisfied, right so there is minus 1, okay so this is overall parity product of the signs this is not satisfied.

So what do you do? You replace min 1 with absolute value of min 2, okay so min 1 position got replaced with absolute value of min 2 here so min 2 came here and then all the other positions got replaced with min 1 and then all the signs got flipped, okay so because overall parity is minus 1 all the signs got flipped, okay so this is basic minsum. The same thing we do in the remaining in the next row as well min 1 is 0.5, min 2 is 0.6 and then (so we) and overall parity is plus 1, okay so signs do not get flipped so min 1 gets replaced with min 2, the other 2 get replaced with min 1, okay so that is what happens and you are done, okay.

So first step was initialization, second step was this minsum okay which we did in layer 1 for iteration 1, okay and then we immediately update, okay. So after this once you are done you use these values and you update it, okay you add to update, okay. So like I said the received value as the layers get processed the received value will keep getting updated, okay so it is a very easy and simple method I supposed to doing row processing first and then doing column processing, okay the column processing happens as you finish the layers, okay so you do the first two rows then immediately you update, what is update? You just add, okay.

So minus 0.1 you added 0.2 minus 0.3 this became minus 0.1, minus 0.3 plus 0.2 it became minus 0.1, 1.2 minus 0.2 is 1 so on, you just add hopefully I have done the addition right you can check that all the addition here results in the sum. So from now on I am going to call this as sum, this is like the output LLR okay output belief this r is like the input belief, okay and then we will keep updating this belief as we go along, okay so this is what we keep track of, is it okay? So that is what happen in iteration 1 in layer 1 it is quite easy the received values came in, we initialized layer 1, did minsum processing for layer 1 and then updated immediately you are done, okay not done yet we will go to the next step. What is the next step? We will do iteration 1 in layer 2, okay.

(Refer Slide Time: 12:35)



What do we do in layer 2? This is your updated belief, okay and now layer 1 is done so we will not consider this not processed anymore so this is already done we will call it already done as far as iteration 1 is concerned this is already finished. So we will not touch it, so only layer 2 will be in operation here, okay. So what do we do in layer 2? So layer 2 we initialize, so notice this minus 1 goes here, this goes here, this goes here, this minus 1.1 goes there, 0.6 goes here, 1.1 goes here, minus 1.6 goes here, okay.

So it is almost as if the sum became the received vector for layer 2, okay so we had a received vector, we process layer 1, in the same way we are going to process layer 2 as well except that we will think of the sum vector the processed vector from the previous layer as the input to this layer, okay. What do we do you have layer, you have to process it, you have an incoming vector so you do initialization and that is what happens here, all these values go

in and occupy those memory positions in the layer 2, okay the row 3 and row 4 that is what you do here.


And once you occupy the rows you do row processing which is minsum on layer 2, rows 3 and row 4 and if you notice here min 1 and min 2, okay min is this, min 2 is also this, okay and then the overall parity is (minus) plus 1, okay the overall so we process that. So when you have two things equal everything will become the same, right so 0.1, 0.1, 0.1, 0.1, min 1 and min 2 are the same (so) and the sign does not get changed there so it become minus 1.1, minus 1.1, minus 1.1, minus 1.1.

And for row 4 again min 1 is 0.6, min 2 is 1.0 and so that is what happens and again overall parity is also plus 1, okay so nothing gets changed and you can identify min 1 here and min 2 here and that got replaced and you are done, okay. So now what do we do? And we also update, okay so you take these two values and this guy and then you add to update, okay so you can see I have done the update correctly.

Once again remember the first layer is already finished and you sort of ignore the first layer when you process the second layer, okay the second layer is sort of like a new decoder you have an incoming vector you do the same processing as you did in layer 1 and you process layer 2 add and update and move on, okay that is what we do. So you see here I have added here minus 1.1 minus 1.1 0.2, minus 1.1 minus 1.1 minus 0.2, 1 and 0.6 is 1.6, hopefully I have added correctly you can check that and you get the output belief of layer 2 so that is it, this is how you process layer after layer, okay.

The next iteration also I am going to walk you through because there will be one minor difference in the next iteration because already this L matrix will have some value, okay so what do you do in that case? So that is the only thing I am going to show and after that you will see the layer decoding is well set, okay it is a much more easier operation and you just use the storage do the row processing and keep repeating alright.

(Refer Slide Time: 16:02)



Iteration 2, Layer 1: Subtract

sum = [-0.2 -0.2 1.6 -1.2 1.6 1.7 -1.7] *incoming belief*

L = $\begin{bmatrix} -0.3 & 0.2 & -0.2 & \blacksquare & -0.2 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -0.6 & \blacksquare & 0.5 & -0.5 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$ *Layer 1*

Subtraction in Layer 1: Rows 1,2

sum = [0.1 -0.4 1.8 -0.6 1.8 1.2 -1.2] *connected input for layer 1*

L = $\begin{bmatrix} 0.1 & -0.4 & 1.8 & \blacksquare & 1.8 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -0.6 & \blacksquare & 1.2 & -1.2 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$

© 2006 MIT Modifications to the Decoder: Layered Decoding and Offset



So this is iteration 2 layer 1 the first step is what I call subtract, okay so you always have to subtract this is the incoming belief, okay this is the incoming belief and remember I want to process these two layers these two rows, right so this is layer 1, okay. So in the first step you have to subtract, so what do I mean by subtract? You take the incoming belief and do incoming belief minus layer 1, so this is subtraction so incoming belief minus layer 1, so this is sort of the subtraction that you have to do being roughly stating what I am doing here I will illustrate what is going on.

So if you see here I do minus 0.2 minus minus 0.3 I get 0.1, okay so this is already I get this minus 0.2 minus minus 0.3, okay and so on minus 0.2 minus 0.2 is minus 0.4, 1.6 minus minus 0.2 is 1.8, minus 1.2 minus minus 0.6 is minus 0.6, so on. So I do subtraction incoming belief minus layer 1, is it okay? Alright? So this is like the new actual input or some sort of a connected input for layer 1, okay.

You had an incoming belief and before you start processing layer 1 you have to sort of correct it, what are you correcting it for because the incoming belief actually had some input from layer 1 itself in the previous iteration there was some input from layer 1 which was used in updating incoming belief, okay. Now when it comes back again to you, you have to subtract that influence, so what you contributed to the incoming belief in the previous round has to be subtracted out before you do fresh processing, so that you use only the new input that came in in your process, okay.


So this is sort of similar to this rule that we have that we do not resend or reuse information that was already used, okay so that is something that is important so this is the correction that happens here. So this is sort of what happens in the column processing, right you take the incoming belief from the channel and you add all the other beliefs and you subtract out the belief that you had.

So that subtraction happens in the layer decoding in this way. So the incoming belief is subtracted from layer 1 and you get the corrected input for layer 1. Once you had the corrected input for layer 1 everything else is the same, okay. So once you have the corrected input it is everything is very straight forward you repeat exactly what you did before, okay. So you start moving the values into the correct positions and then you do proceed as before, okay.

So once again I want to repeat the incoming beliefs come into a layer, the layer already has some memory stored in. now what you have to do is, you have to do this updating or correction for the incoming belief, you take the incoming belief, subtract it from what is already stored in the layer, okay and then what is already stored in the layer is subtracted and difference is what is retained as the corrected belief and that same difference is also used to update the storage in the layer 1, okay the layer in the storage 1 also gets updated with that difference and the incoming belief also gets corrected in the same process, okay both of these happens.

So take a note of it there are some 2, 3 things happening here and this is this step is very important, if you do not do this correctly lot of instability will happen in your decoder because you will be reusing the information you already used the information that came from rows layer 1 before got added to the incoming beliefs and then they have to be subtracted now and updated and then you have to use so this is the crucial step in the layer decoder, okay. So once you do this that is it.

(Refer Slide Time: 20:08)



Iteration 2, Layer 1: Minsum

$sum = [0.1 \quad -0.4 \quad 1.8 \quad -0.6 \quad 1.8 \quad 1.2 \quad -1.2]$

$L = \begin{bmatrix} 0.1 & -0.4 & 1.8 & \blacksquare & 1.8 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -0.6 & \blacksquare & 1.2 & -1.2 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$

Minsum in Layer 1: Rows 1,2

$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$

PROF. ANDREW THANGARAJ
 Modifications to the Decoder: Layered Decoding and Offset



So the next step is just minsum, okay so we have the incoming belief here and then these are the this is layer 1 and you want to do this is sum 1 layer 1 so this is the beliefs they got into the layer 1 storage and we are doing minsum, okay so minsum is just the same as before you identify min 1, min 2 and the overall parity is minus 1 here. So sign gets flipped and min 1 and min 2 get assigned in the same fashion.

So you see here you got min 1, min 1, min 1 and here you got min 2. I am not going to show you the second one second row you can work it out for yourself but once again I want to point out when you process layer 1, layer 2 storage is completely ignored you are not doing anything with the layer 2 storage it is just kept as it is, okay hopefully this was clear what was going on you do the you process this layer 1 for minsum just like before. So once you got the corrected inputs into layer 1 you process layer 1 just as before.

(Refer Slide Time: 21:14)

Iteration 2, Layer 1: Update

$sum = [0.1 \quad -0.4 \quad 1.8 \quad -0.6 \quad 1.8 \quad 1.2 \quad -1.2]$ *corrected version*

Update in Layer 1: Rows 1,2

$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$$

add to update

$sum = [-0.3 \quad -0.3 \quad 1.7 \quad -1.8 \quad 1.7 \quad 1.8 \quad -1.8]$

subtract → minsum → update

PROF. ANDREW THANGARAJ
@TUMUKAI


Modifications to the Decoder: Layered Decoding and Offset

So once you finish processing layer 1 is just the update step, so how do you do updating? You take these two guys and then add to update okay so these two are added in the update fashion so that is it. So this is the three step process you will have a subtract step in each layer you will have a subtract step and then you will have a minsum step, then you will have an update step, okay.

So remember this subtract is a bit critical if you miss this things will go wrong, when you subtract both the incoming belief the overall total belief and the storage in that particular layer is updated, after you update (what) due to the minsum then after you finish the minsum the incoming beliefs are updated again but that is also only to the corrected incoming belief, so the new incoming belief the corrected version that gets updated so remember that gets added to the what is there in the layer already the processed minsum processed value then you get the new one, okay you can see I have done the addition here correctly you can check if that addition is correct or not and you get the new beliefs.

So once you do this after this every step is the same. So you go into a layer you have some incoming beliefs you correct the beliefs, update the layer memory, do minsum processing and then finally do a overall update for the beliefs, so subtract minsum update subtract minsum update you can carry on iteration after iteration.

(Refer Slide Time: 22:56)



Iteration 2, Layer 2: Subtract

incoming belief

$$\text{sum} = [-0.3 \quad -0.3 \quad 1.7 \quad -1.8 \quad 1.7 \quad 1.8 \quad -1.8]$$

$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare & -0.1 \\ \blacksquare & \blacksquare & 0.6 & \blacksquare & 1.0 & 0.6 & \blacksquare \end{bmatrix}$$


Subtraction in Layer 2: Rows 3,4

corrected belief

$$\text{sum} = [-0.2 \quad -0.2 \quad 1.1 \quad -1.7 \quad 0.7 \quad 1.2 \quad -1.7]$$

$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.2 & -0.2 & \blacksquare & -1.7 & \blacksquare & \blacksquare & -1.7 \\ \blacksquare & \blacksquare & 1.1 & \blacksquare & 0.7 & 1.2 & \blacksquare \end{bmatrix}$$

PROF. ANDREW THIANGARA
IIT MADRAS
Modifications to the Decoder: Layered Decoding and Offset



Iteration 2, Layer 2: Minsum

$$\text{sum} = [-0.2 \quad -0.2 \quad 1.1 \quad -1.7 \quad 0.7 \quad 1.2 \quad -1.7]$$

$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.2 & -0.2 & \blacksquare & -1.7 & \blacksquare & \blacksquare & -1.7 \\ \blacksquare & \blacksquare & 1.1 & \blacksquare & 0.7 & 1.2 & \blacksquare \end{bmatrix}$$

Minsum in Layer 2: Rows 3,4


$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.2 & -0.2 & \blacksquare & -0.2 & \blacksquare & \blacksquare & -0.2 \\ \blacksquare & \blacksquare & 0.7 & \blacksquare & 1.1 & 0.7 & \blacksquare \end{bmatrix}$$

PROF. ANDREW THIANGARA
IIT MADRAS
Modifications to the Decoder: Layered Decoding and Offset

I will show it to you just for iteration 2 layer 2, okay just going through that in the same detail here. So first we do subtraction, okay so this is incoming belief, this is corrected belief so you see that the okay so this is okay this is corrected belief I got a little confused with this layer 1 and layer 2 so remember this layer 1 is ignored ignore layer 1 so things will be complicated when you are looking that so you have to ignore the layer 1 part, only layer 2 matters I am doing layer 2, so the corrected beliefs go and occupy layer 2 so you see all of that has happened correctly so the occupation happens correctly.

And then once the corrected beliefs have been put in layer 2 you do minsum just like before minsum happens here I do not want to talk about this is great detail I have done the minsum find the two minimums, do the overall parity and assign it as before.

(Refer Slide Time: 24:10)




Iteration 2, Layer 2: Update

$sum = [-0.2 \quad -0.2 \quad 1.1 \quad -1.7 \quad 0.7 \quad 1.2 \quad -1.7]$

Update in Layer 2: Rows 3,4

$$L = \begin{bmatrix} -0.4 & 0.1 & -0.1 & \blacksquare & -0.1 & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & -1.2 & \blacksquare & 0.6 & -0.6 \\ -0.2 & -0.2 & \blacksquare & -0.2 & \blacksquare & \blacksquare & -0.2 \\ \blacksquare & \blacksquare & 0.7 & \blacksquare & 1.1 & 0.7 & \blacksquare \end{bmatrix}$$

$sum = [-0.4 \quad -0.4 \quad 1.8 \quad -1.9 \quad 1.8 \quad 1.9 \quad -1.9]$



PROF. ANDREW THANGARAJ
@ IITMADRAS

Modifications to the Decoder: Layered Decoding and Offset

And then once you finish the minsum it is just the update, okay and you have the overall belief, so this is the step that is repeated again and again and again when you want to do more iterations repeat the same thing subtract, minsum, update, okay. So now if you look at this structure of the storage this was a very small storage so you do not have major challenges when you implement.


But how to store this efficiently? How to read from this efficiently? How to write back into it efficiently? How to do that in parallel is the biggest challenge when you implement LDPC decoding and today there are really very very advanced IP codes available from many people they really optimize the optimize this storage and all that very very well and it can run at multiple Gigabits per second, it is easy to get implementations like that today so it is not a very difficult operation but you see that is how it happens.

Also couple of words about how you will go into the 5G code when you move towards 5G each layer will be one block row so we have this block row base matrix, every block row is you remember gets expanded into the ZC. So that one block row will be one layer for us, so we will do this layer processing one block row at a time. So here the layer had only two rows in the 5G code we may have you know 40 rows, 50 rows in a particular layer but still the column weight will only be 1 and you can do the exact same thing there also, okay so hopefully that was clear to you.

But you can also see that there is this because of these blue blobs that are there the storage has to be carefully setup the addressing has to be both column wise and row wise and you

should know what to add, what to subtract, etc when you do row processing and when you do the subtraction of columns and all that, that takes a little bit of you know variables and carefully assigning that when we implement the decoder you will see how that is done, okay.

(Refer Slide Time: 26:07)




Decision

- If $\text{Sum}_j > 0$, Decision on Bit $j = 0$
- If $\text{Sum}_j < 0$, Decision on Bit $j = 1$
 - Assuming BPSK 0 \rightarrow +1 and 1 \rightarrow -1

After two iterations,
 $\text{Sum} = [-0.4 \quad -0.4 \quad 1.8 \quad -1.9 \quad 1.8 \quad 1.9 \quad -1.9]$ ^{belief}

$\text{Dec} = [1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1]$


- For more iterations, continue...



PROF. ANDREW THANGARAJ
 IIT MADRAS
 Modifications to the Decoder: Layered Decoding and Offset


So how do you do decisions okay so that is the last part which I should emphasize, decisions are very simple as just thresholding you take the belief at any stage, belief that you have at any given stage this variable which I called as sum and if it is negative you make it 1, if it is positive you make it 0, okay and if you want to do more iterations keep continuing on and on and on whenever you want to stop you stop make a decision you are done, okay so that is the layer decoder.

(Refer Slide Time: 26:36)



Summary

- LDPC codes provide close-to-capacity performance
- Adopted in several standards
- VLSI architectures and implementations are available today
- Interesting issues
 - Optimized implementations
 - Speed and efficiency
 - Optimal layering?



PROF. ANDREW THANGARAJ
 IIT MADRAS
 Modifications to the Decoder: Layered Decoding and Offset

So let me summarize what I told you about the decoder so far it LDPC codes provide close to capacity performance, adopted in several standards, VLSI architectures implementations are available today. There are quite a few interesting issues and things one can look at but really most problems are already solved today we do not have to worry about doing much more research and improving the decoder of course this more research to be done for lower block length and all that but now for higher block length more or less, okay.

So in this course atleast we will focus on implementation in particular we will try and do a MATLAB implementation for the layer decoder, okay. So we will take one of the parity check matrices in the 5G standard and build a layer decoder for it, okay so just the storage matrices, then you know the received values simulated, let it go through the layer decoder, do the subtract row process and update, okay.

So hopefully you agree with me the operations themselves are quite simple, it is just a question of organizing the memory and moving things around, okay. So that is the end of this lecture and in the next lecture we will see a MATLAB implementation of layer decoder for LDPC codes, okay thank you very much.