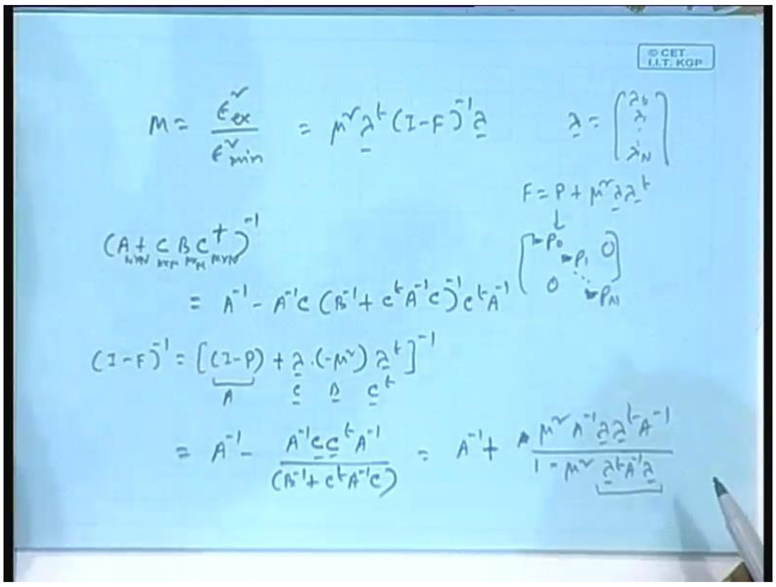
Adaptive Signal Processing

Prof. M. Chakraborty
Department of E and ECE
Indian Institute of Technology, Kharagpur

Lecture -13
Sign LMS Algorithm

Doing with this misadjustment and just a few little bit of that is left, so let us a quickly work that out.

(Refer Slide Time: 01:05)



$$M = \frac{\tilde{\epsilon}_{ex}}{\tilde{\epsilon}_{min}} = \mu^2 \underline{\lambda}^t (I-F)^{-1} \underline{\lambda} \qquad \underline{\lambda} = \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_N \end{bmatrix}$$

$$F = P + \mu^2 \underline{\lambda}\,\underline{\lambda}^t$$

$$P \downarrow \begin{bmatrix} \rho_0 & & 0 \\ & \rho_1 & \\ 0 & & \ddots\ \rho_N \end{bmatrix}$$

$$(\underset{N\times N}{A} + \underset{N\times M}{C}\,\underset{M\times M}{B}\,\underset{M\times N}{C^t})^{-1} = A^{-1} - A^{-1}C(B^{-1} + C^tA^{-1}C)^{-1}C^tA^{-1}$$

$$(I-F)^{-1} = [\underbrace{(I-P)}_{A} + \underbrace{\underline{\lambda}}_{\underline{c}}.\underbrace{(-\mu^2)}_{b}\underbrace{\underline{\lambda}^t}_{\underline{c}^t}]^{-1}$$

$$= A^{-1} - \frac{A^{-1}\underline{c}\,\underline{c}^tA^{-1}}{(b^{-1} + \underline{c}^tA^{-1}\underline{c})} = A^{-1} + \frac{\mu^2 A^{-1}\underline{\lambda}\,\underline{\lambda}^tA^{-1}}{1 - \mu^2 \underline{\lambda}^tA^{-1}\underline{\lambda}}$$

Misadjustment M was defined as this ratio, excess mean square error divided by minimum mean square error, right. Apart from the excess mean square error come with, I told you that in a steepest descent procedure, we are I mean, while going to a LMS algorithm from there we replace r and p by some weird estimates, wild estimates. And that is why the penalty that we pay, the prize that we pay is that the filter based does not converge to the optimal weights directly, but only they converge into terms of mean. As a result if you take the error, filter error d n minus y n, that error is no longer having the minimum variance, it will have some extra components that is the excess mean square error.

As this ratio is misadjustment; obviously, we would like to have this as low as possible, isn't it. The numerator should be much less compared to the denominator. So, this is a very important quantity. Now, to find that out we yesterday correct me if I make making a mistake, this is the expression that we derived actually, okay. Lambda was vector consisting all the Eigen values of R matrix, input autocorrelation matrix. F had 2 component, P plus mu square lambda-lambda transpose, right. Where P was what, P was a diagonal matrix of this form 1 minus rho 0, 1 minus rho 1 dot, dot, dot 1 minus rho n 0 0, sorry it is just rho 0 sorry, sorry, sorry just rho 0, rho, 1 rho N, right.

Now I have to evaluate this quantity, because you see F itself called the, consist of mu square. I have to see how this quantity depends on mu, because remember in everything that we deal with LMS, whenever you deal with LMS algorithm the only parameter that you have in your hand from controlling things is mu, there is you only handle. Even while remaining with the range of convergence, how to adjust other things like this M by adjusting mu. Now, here outside there is mu square, but inside also there is a mu, isn’t it in F matrix you see. So, I have to simplify this further, so that the dependence on mu comes out very clearly.

That is how you need to simplify it further, for that we yesterday quoted that matrix inverse lemma that is, A suppose is the N cross N matrix, C say N cross M, B M M cross M and C transpose. So, N cross M this is N cross M, this is N cross M, this is M cross N, this is N cross N. So, all of these are N cross N matrix suppose it is invertible, the sum I mean wherever I am using inverse of a matrix, please assume that I am assuming that inverse exist, the matrix is invertible. Under that assumption will be I am writing the expression then, the formula that we have here is A inverse minus A inverse C B inverse.

See, C cannot be invertible because C in general square matrix, N cross M, but B is a square matrix, B is M cross M. So, is A here I am assuming that inverse is exist, these are the thing. Now, I want to make use of it to obtain the, I have to simplify this I minus F inverse, F is P plus this. So I minus F, I minus F inverse means what? You replace F by this, so I minus P 1 matrix which is a diagonal matrix. I is a diagonal matrix, P also is a diagonal matrix of this form. So, I minus P is a diagonal matrix typical element is 1 minus rho I, minus this mu square times.

So you can write it that way lambda 1, N cross 1 into minus mu square into lambda transpose, the whole thing inverse, okay. So, this is your C this is your B this is your C transpose, C is N cross 1 only a column vector. So this is M cross M, this 1 cross 1 which is scalar here, that is 1 cross N because it is a transpose. So, I can directly use this formula, why I am doing this you know and I am calling it A, as I told you what is my purpose here, why I want to take A? Because, you see in this inverse expression A inverse comes of N and A is a diagonal matrix getting its inverse is no problem. That is why I want to target and want to take that to be A and simplify, I mean go ahead, isn't it.

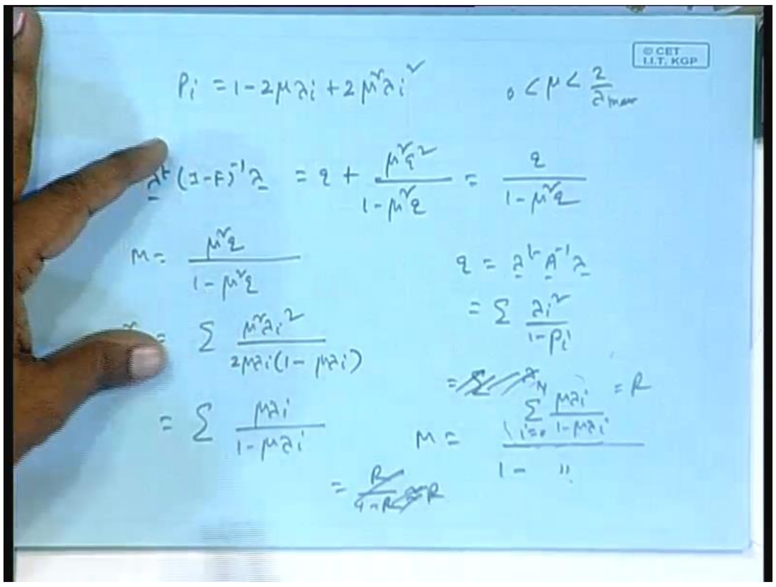
That is why I separated it out and if you do that then, in this case it will be keeping, I mean calling it A itself, I will replace it by i minus P later. A inverse minus look at this quantity, this quantity B inverse plus C transpose A inverse C. Now B is a scalar here, B is what minus mu square. So, B inverse is just 1 by that is a scalar and obviously, this has to be a scalar otherwise, it will not be comfortable scalar plus matrix has no meaning.
Of course is scalar, because square matrix times to column vector, column vector row into column scalar. So, scalar plus scalar is a scalar inverse of that means, 1 by that. So, that I can take out of this matrix product is a scalar number.

So matrix product remains as it is, A inverse C, C transpose A inverse divided by the scalar element. In fact okay, let me write down that scalar element is your, in this what is B inverse C transpose A inverse C and this means A inverse minus. Now what is B, B is minus mu square, B is minus mu square. So, 1 B inverse is 1 by minus mu square. So, mu square will go up, if you put minus 1 by mu square here, this minus and this minus will be make it plus, 1 minus mu square C is lambda. So, lambda transpose and here mu square times A inverse lambda-lambda transpose A inverse. So, this is that is where we stopped here yesterday.

Now, this quantity you know lambda transpose A inverse A, it is the scalar I told you A inverse is matrix, matrix into column vector, column vector rho into column is a scalar, you give it a name. Some name suppose, it is q have I use q somewhere, no, q do not you send q now, isn't it. So this is a scalar q, I can call it q, the scalar only, I am given it a name q. Now, come back to this formula I am interested in this excess mean square here, M. Here I have to multiply by mu square I will do later, but before that lambda transpose times I minus F inverse into lambda. So, this whole expression will be pre multiplied by lambda transpose post multiplied by lambda, is it not?

So, at then mu square will come little later remember, the mu square I will bring in later, step by step, only I am bringing lambda transpose and lambda. If I do that on the, on the left hand side if you multiplied by lambda transpose and from the right hand side lambda. What you get lambda transpose A inverse lambda which is back to q again, isn't it.

(Refer Slide Time: 08:52)



$$p_i = 1 - 2\mu\lambda_i + 2\mu^2\lambda_i^2 \qquad 0 < \mu < \frac{2}{\lambda_{max}}$$

$$\underline{\lambda}^t (I-F)^{-1}\underline{\lambda} = q + \frac{\mu^2 q^2}{1-\mu^2 q} = \frac{q}{1-\mu^2 q}$$

$$M = \frac{\mu^2 q}{1-\mu^2 q} \qquad q = \underline{\lambda}^t \underline{A}^{-1} \underline{\lambda}$$

$$= \sum \frac{\lambda_i^2}{1-p_i}$$

$$= \sum \frac{\mu^2\lambda_i^2}{2\mu\lambda_i(1-\mu\lambda_i)}$$

$$= \sum \frac{\mu\lambda_i}{1-\mu\lambda_i} \qquad = \sum_{i=0}^{N} \frac{\mu\lambda_i}{1-\mu\lambda_i} = R$$

$$M = \frac{\sum_{i=0} \frac{\mu\lambda_i}{1-\mu\lambda_i}}{1 - \;''}$$

$$= \frac{R}{1-R}$$

This thing, lambda transpose I minus F inverse lambda, this quantity at I minus F inverse is this. So lambda transpose on this, lambda on this, lambda transpose A inverse lambda which by definition is q the scalar number. So, q is coming and here also on the left hand side lambda transpose, on the right hand side lambda. Mu square is a scalar forget it, you bring it out. So, lambda transpose A inverse lambda which is a scalar and that is q, again lambda transpose A inverse lambda again q. So, mu square q square and now this is the thing calculations I was having in mind, I did not pre multiply and post multiply that is why it was not coming.

Now, if you multiply this there all scalar numbers, you see mu square q square cancels. So, what you get is q divided by this guy and what is M, M is this quantity multiplied by mu square. This quantity multiplied by the mu square. So, M is mu square into q divided by 1 minus mu square q, what was q? Now q was lambda transpose A inverse lambda. So, I rewrite again, q was lambda transpose A inverse lambda, what is A inverse is a diagonal matrix again. 1 by 1 minus rho 0, 1 by 1 minus rho 1, 1 by 1 minus rho 2 dot, dot, dot, dot that times lambda and lambda transpose

So, this will be summation lambda I square by 1 minus rho I, you agree. Diagonal matrix multiply lambda, so Ith diagonal nth into Ith element here, that again Ith entry of lambda transpose, this what you get. Now what was rho I, if you remember our previous this derivation rho I, I give it a name actually rho I was, please check your notes it was something like a quantity like this. 2 mu lambda I, please correct me if I am wrong because I am just relying on my memory. This was rho I, right. So, 1 minus rho I is you can easily see, so mu square q, what is mu square q, what is mu square q, q is this.

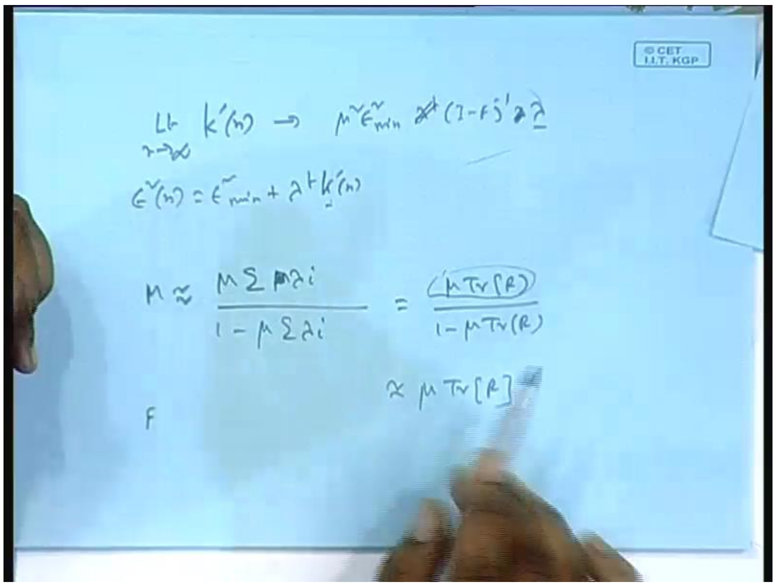
So, mu square lambda square by 1 minus rho I very simple arithmetic algebra, 1 minus rho I is 2 mu lambda I 1 minus and the base mu lambda I. So that mean, A means nothing but summation mu lambda I divided by 1 minus mu lambda I, that is your mu square q divided 1 minus, if I just put like this is it okay? This quantity here, mu square q again mu square q, so this entire quantity here 1 minus that this will be M. So, now this gives an exact dependence on mu and you see it is not a simple linear dependent, mu on top mu below so on and so forth, isn't it. This gives us exact dependence, so you can plot it giving lambda and all, you can plot it and see the dependence on mu and all that that is our exact thing.

Normally we prefer, we suggest to be as much adjustment M to be much less. Of course, I can tell you before I had that, for that you know you have to bring down mu. You have to take lesser value of mu, mu has a range for convergence within the range also you have to take lesser value, then only M will be less. And the prize that you pay there is that mu is less from Steepest descent unit it will take more time, it will such calling with some approximation. It will take more time means its converge time, it takes more time to converge. So converges rate decreases, but you will as you see later you have to take less value of mu, to make M less.

Anyway, suppose our purpose is to see that this excess mean square error it is really much less, it is not very high, it is a 10 percent or even less 5 percent. In such cases what happens to this expression, now look at the numerator this quantity mu lambda I by 1 minus mu lambda I. Can you see that this denominator is always positive, why? Because mu was 2 by… So mu is less than 1 by lambda max also, is it not? Mu sorry, if you take, suppose you take all right, yeah, mu is you have to take mu that way, less than 1 by M. So, this positive and this quantity, because I am telling you have to take mu very less. I am that, if that happens this quantity is positive, this quantity if you call say any quantity say, you give you the name some name say R, capital R, this M is nothing but R by 1 minus R, is it not?

So when will M be less, if R is not R is less and less so that, 1 minus R is approximately equal to 1 and R by that is R. This is simply ((Refer time: 15:24)), R by 1 minus R this quantity if you take R very less, if you take R very less what happens? R is have to very close to 0, means then only denominator is maximized, isn't it, R numerator minimize and then this becomes approximate equal to R, right. So, under that thing we mu this mu lambda I by 1 minus mu lambda I, if you take mu to be very less this approximate equal to just mu lambda I. If about this, this, this I am this, this logic I am extending here first, this quantity suppose mu lambda I equal R. So, R by 1 minus R, R I, R I by 1 minus R I, while R I is very less, because of a choice of mu, this is approximately equal to R I and then this summation.

(Refer Slide Time: 16:18)



$$\lim_{n\to\infty} k'(n) \rightarrow \mu \tilde{\epsilon}_{min} (1-F)^{'}\underline{\lambda}$$

$$\tilde{\epsilon}(n) = \tilde{\epsilon}_{min} + \underline{\lambda}^t \underline{k}'(n)$$

$$M \approx \frac{\mu \sum \lambda_i}{1-\mu\sum\lambda_i} = \frac{\mu Tr(R)}{1-\mu Tr(R)}$$

$$\approx \mu Tr[R]$$

F

So, this is nothing but approximately equal to M is these are all approximations, approximately equal to summation mu, so mu can go outside lambda I. And then summation lambda I is what, is same as trace of the autocorrelation matrix, because of similarity transformation I told. That sum of diagonal elements is same as sum of the

Eigen values. So this is trace R, this is trace R, this is equal to mu into trace R divided by 1 minute mu trace R. Again by your choice of mu, if this quantity is be less, if this quantity is very less then, this is approximately equal to only the numerator.

So you see when this misadjustment is very less in this, actually proportional to mu, for lower range the misadjustment is actually proportional mu. So, then thumb rule is you take as mu as low as possible. So, this is for this misadjustment analysis, I told you other day that, that matrix rather than vector k prime N, small k prime N as it goes to infinity it remains bounded, provided that matrix, what matrix F, F matrix has Eigen values within the range in my nature within the range 0 to 1.Then that will be contribute to a 0 matrix and you get a constant term, which gave rise to this misadjustment and all that, isn’t it.

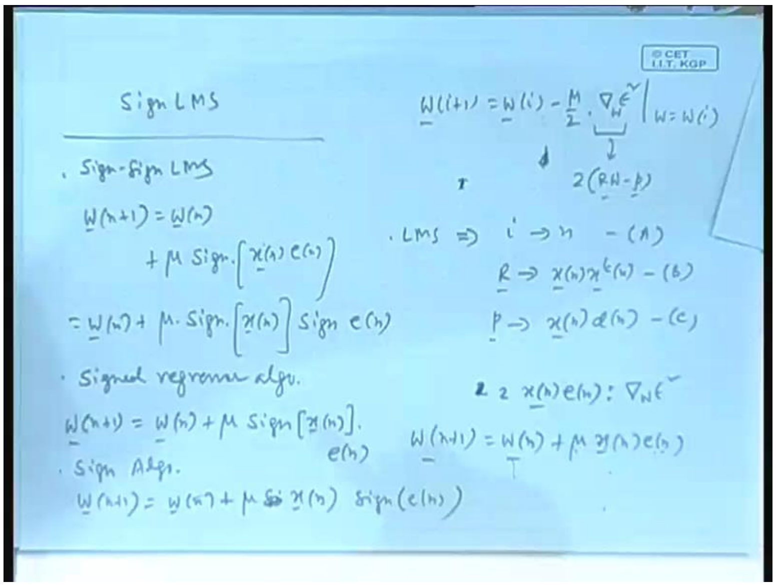
So Eigen values of F all bounded by 1, I mean from top in magnitude for that I told it is sufficient if we follow this condition. If mu is satisfies this, this will automatically satisfy, this is stronger condition actually for having that satisfy, but this is good enough. So that, that matrix F has got its own Eigen values within the range I mean in magnitude 0 to 1, actually minus 1 to 1. So, then there excess that matrix K prime I, I M remember K prime came from that weight error covariance matrix k N multiplied by T transpose P multiplied by T transpose, that k prime I what I am talking about this. This vector as n tends to infinity, this gives to goes to a constant term, that constant term was mu square epsilon square mean.

This is repetition only, I minus F inverse is it not? Lambda, this is the constant term, lambda to not lambda transpose, this we come later. Because, that error epsilon square n was epsilon square mean plus lambda transpose, there will be lambda here, there is a lambda here. Lambda transpose what, k prime n this you have seen, k prime n has n tends to infinity goes to this finite value, provided matrix A if has Eigen values that have magnitude between 0 to 1 not have 1. Then only it is stable you get only this finite value, which gives rise to this excess mean square component, you take the ratio call its misadjustment that just analysis analyze it.

This is something you have to leave with, but again you can minimize it by taking less and less mu provided, you also ready to bear the prize of you know slower convergence. So, this is a very elaborate analyze the LMS algorithm, both in mean and mean square.

One purpose was to kind of give a taste of for given you some exposure of how to do such statistical analysis. We now consider briefly some other variants of LMS algorithm. Some of this I will touch up 1 and 1 I will take up in little detail, and that will I again take some DSP knowledge and I do not know how much DSP knowledge you have. So, have to fill up that space also.

(Refer Slide Time: 20:49)



Sign LMS

· Sign-Sign LMS

$\underline{W}(n+1) = \underline{W}(n) + \mu \, \text{Sign.}\left[\underline{x}(n) e(n)\right]$

$= \underline{W}(n) + \mu . \text{Sign.}\left[\underline{x}(n)\right] \text{Sign } e(n)$

· Signed regressor algo.

$\underline{W}(n+1) = \underline{W}(n) + \mu \, \text{Sign}\left[\underline{x}(n)\right] . e(n)$

· Sign Algo.

$\underline{W}(n+1) = \underline{W}(n) + \mu \, \underline{x}(n) \, \text{Sign}(e(n))$

$\underline{W}(i+1) = \underline{W}(i) - \frac{\mu}{2} . \underbrace{\nabla_W \tilde{\epsilon}}_{\downarrow} \Big|_{W = W(i)}$

$2(\underline{R}\underline{W} - \underline{p})$

· LMS ⇒ $i \rightarrow n$ – (A)

$\underline{R} \rightarrow \underline{x}(n)\underline{x}^t(n)$ – (B)

$\underline{p} \rightarrow \underline{x}(n) d(n)$ – (C)

$2 \, \underline{x}(n) e(n) : \nabla_W \tilde{\epsilon}$

$\underline{W}(n+1) = \underline{W}(n) + \mu \, \underline{x}(n) e(n)$

© CET I.I.T. KGP

One category of LMS algorithm is called sign LMS, as I, how did go to LMS algorithm we first started with this Steepest descent, isn't it. We first start with Steepest descent, that time we are doing the iteration like this w i plus 1, i was the iteration index was from w i minus mu by 2, mu by 2 was that thing constant into del at w equal to w I, remember these are Steepest descent. Then, I said that look and this was giving rise to a standard term, this gradient was, this gradient thing came out to be twice or w minus P, you know. Yes, 2 R w minus P it came to a from this minus sign here 2, 2 cancels it becomes P minus R w rather w i. This what we did, then that time we said that look suppose I do not know R matrix or P vector and we have to carry out this iteration in real time, same iteration suppose you have to carry out to real time.

So, let I index be replaced by time index n at n equal to 0 0th iteration, n equal to 1 1st iteration like that you proceed and R and P to be replaced by appropriate data dependent quantities. I mean quantities which are obtain from data, some estimate based on the available data. Then I replace R by some weird estimate, wild estimate just x n into x n transpose, so derive for P and I got that LMS algorithm. That LMS algorithm was having what? So, LMS what did I do? LMS I was replaced by n this is number 1, R was replaced by B and P was replaced by C, we made the substitutions, we got the LMS algorithm.

What is the gradient we got then, if we really put R P and all this here, look at the gradient at w n i is n now, what are the gradient? Gradient turned out to be if you really put R as x and x transpose n and then w i or w n and x n into B n, what are the gradient, gradient turn out to be 2 and 2 cancels, I mean gradient turns out to be 2 into x n e n this is the gradient. Whether an estimate of the gradient, a wild estimate of the gradient, simple minus sign, so it becomes P minus R w P is x n into d n R is x n x transpose, w means at ith at nth iterations. So, w n x transpose w n is the filter output y n x n vector to you take common.

So, d n minus because this is x n into d n d n minus y n which is e n, we all know this, these are the gradient and this gradient estimate we put back in the update equation, we got that LMS algorithm. In the LMS algorithm we had this thing, this is LMS algorithm. Now, look at the computational cost for this LMS algorithm, there are 2 operations I told you, 1 is the filtering operation if you remember the why I wrote the algorithm given w n you find out first y n, how? W n vector transpose time x n there is the convolution, w 0 into w 0 n into x n w 1 n into x n minus 1 w 2 n into x n minus 2 dot, dot, dot all added.

That I have to always do because filtering is indiscoverable, filtering is what I have to do always, isn’t it. I am looking for a better filter, but filter is primary filter has to be there I only trying to get out as close to an optimal filter as possible. So, that computation is required for a length n filter you did n multiplications there, n minus 1 additions. Then d n minus y n 1 addition, addition subtraction they are equivalent in terms of hardware, 1 reason. But here you see for each tap weight is a vector, for each tap weight what you have to do you have to multiply mu into 1 data from this vector into e n.

Even if you, we take mu to be some power of 2 normally you know it is a style that take mu to be a power of 2. Whether, 2 to the power minus 2 or 2 to the power minus 3 or 2 to the power 3, depending on the range for mu power of 2. Power of 2 means you know in multiplication by power of 2 actually in hardware means, they are shifting operation.

Suppose, even if I do that at least there is this product x n into e n, x n minus 1 into e n, x n minus 2 into e n, x n minus 3 into e n you multipliers. One way I can bring down this cost, both in terms of speed and hardware is to opt for some modification here. That is here in the steepest descent I said, that look I will take the gradient itself both magnitude and sign and I will go the opposite direction of the gradient.

So if the, if it is increasing I will going the opposite direction, but again I am taking magnitude also means, if the gradient is steep I will go back by a bigger leap. If the gradient is not, so I will just scroll. So, that will make convergence faster of course, at least in the at least in the limited context of Steepest descent you can see. If it is steep, that means further away from the optimal point, but I will I will come back to optimal point will be from greater force. That is why I took the entire thing, I got the good elements algorithm, but suppose for the time being I said a look I do not mind scrolling I will only take the sign, I will only take the sign. Then what happens, that means instead of this gradient I first evaluate the gradient and take it sign.

So, here also I go same way I replaced by e n or by this I find out the gradient here as before these my gradient and these gradient, I just take the sign. So that means, actually this called sign, sign LMS has got 3 verities the pure sign LMS algorithm is called sign-sign, sign-sign LMS. Here plus mu into, mu is a positive number sign of mu is positive I am not taking sign of mu, mu into sign on the gradient. Sign of the gradient, sign of when I say x n bar vector into e n, what does it mean actually for each element of the vector multiply that element by e n take the product take its sign. Sign means, plus 1 if it is positive or greater than 0 minus 1 if it is negative.

So it will become a vector of plus 1 or minus 1, it will become a vector of plus 1 or minus 1, are you following me? Take a sign of a gradient, sign means either plus 1 or minus 1. So, plus 1 means it is plus mu or minus mu, so the product is gone, you can still ask me sir how the product is gone, because you are multiplying and then take a sign, but you know if I give you 2 numbers x and y sign of x into y is same as sign of x into sign of y. If x and y both are positive, x y positive sign of x is plus 1 sign of y is plus 1 and sign of x y plus 1, 1 plus 1 into plus 1 equal to plus 1 satisfied. If any of them is negative, sign of x could be plus 1.

But sign of y could be minus y 1, 1 into minus 1 minus 1, but x y again minus negative, sign of that is minus 1 and when x and y both are negative, sign of x is minus 1 sign of y is minus 1 product is plus 1. On the other hand x y is plus minus 1 and then, plus sign is plus it satisfied. So that means, I can ((Refer Time: 29:25)) apply sign on this and on this.

That means, this is nothing but mu into mu is very hardware friendly algorithm, mu plus minus multipliers and all that. Sign of e n sign of x n vector, each entry it sign multiplied by sign of e n, this is the absolute sign element algorithm called fine sign LMS algorithm.

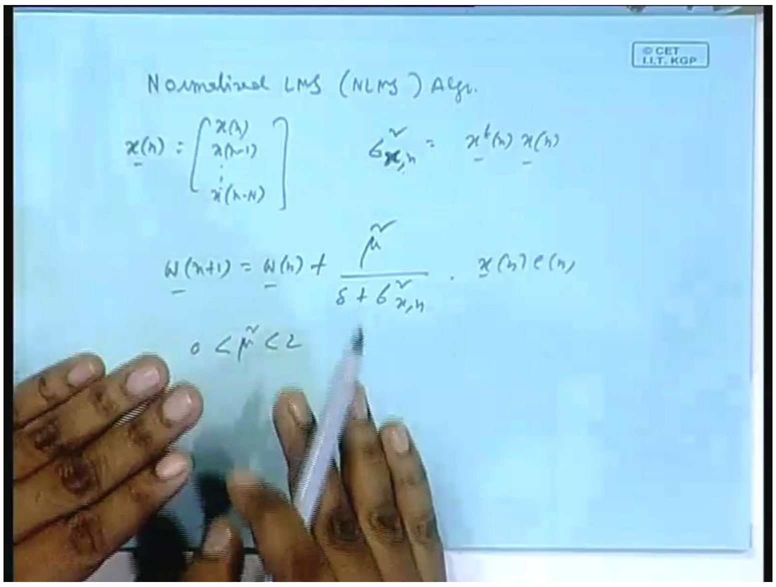
Then there are some modifications of it, there you say that look here I have thrown away the entire gradient information, its magnitude an all only sign, but you can takes its magnitude partially and sign of course. So one way signed regressor algorithm, this x n vector is called a sign regressor. So here we apply only sign over here, do not give this sign assuming mu to be a power of 2, still no multiplication is required and shifting only these will be required and then addition, addition or subtraction depending on the signs.

And alternatively this simply sign algorithm where, you have apply sign take up the sign here.

So, you see sign of the gradient is not changing, earlier gradient was x n into e n here, I am taking, I am saying that I will retain the sign of say x and y. That sign into e n e n itself will carry its sign, so original sign of the gradient remain same otherwise it is dangerous, are you following me? Either here or here, but I am giving partial information of the gradient, varying magnitude. I have, they have done convergence analysis of all these, but only things is that is very complicated and not so easy, because of this sign thing sign is non-linear a thing you know.

Sign of x plus is y is not sign of x plus sign of y, it is not non-linear thing and you are lost and lot of complications just we do not study that here. I only put the algorithms. Sorry, but you of course then between the 3 these 2 have better convergence characteristic then this is, because you are still giving some idea of the, some information about that partial information of magnitude of a gradient also, so which you are not giving here. But this is most hardware friendly, because no shifting of mu is also is required thus, even when mu is not power of 2, if mu is not power of 2 you need a multiply here in both cases, oh not here.

(Refer Slide Time: 33:22)



© CET I.I.T. KGP

Normalized LMS (NLMS) Algo.

$$\underline{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-N) \end{bmatrix} \qquad \tilde{\sigma}_{x,n} = \underline{x}^t(n)\,\underline{x}(n)$$

$$\underline{w}(n+1) = \underline{w}(n) + \frac{\tilde{\mu}}{\delta + \tilde{\sigma}_{x,n}} \cdot \underline{x}(n)\, e(n)$$

$$0 < \tilde{\mu} < 2$$

There is another algorithm called this is a fantastic algorithm, normalized I will just touch up on it LMS, because it is bad if I do not just mention these algorithms while teaching a course on adoptive filter. Normalized LMS, NLMS algorithm in the case of sign LMS algorithm I was making the algorithm simplified in terms of hardware or computation. And therefore, I was paying the prize of convergence speed. It will take more time converge as you can see you know, I am just only taking the sign not the entire thing. So, convergence speed was thrown away I mean, let me take more time, but it will become hardware friendly.

On the other hand if I want the convergence speed to go up then, in ordinary LMS algorithm then, I can make the algorithm more complicated, instead I simplifying I can introduce some extra complication. So, convergence speed will go up and such method is normalized LMS. I will just code these we have got the x n vector, isn't it. This we know, suppose I say sigma n square sigma x n, this is x, x n square it is nothing, but norms square of these vector. Norm square means, x square plus this square plus this square norm square which is the bigger of the energy, square of this sample is a power and which becomes energy.

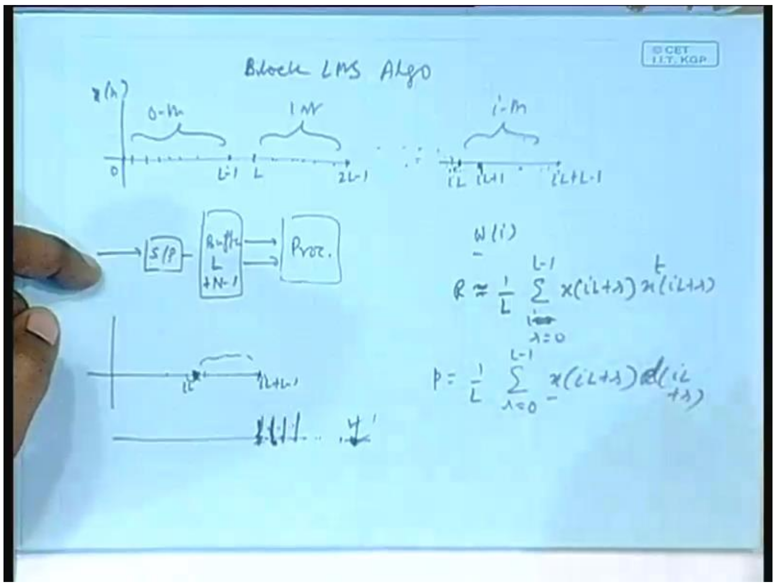
These then filtering part remember, filtering part is common in LMS sign LMS sign iteration an all that. Filtering, but if you give me a set of filter weight I will filter in the usual way. Get y n get en that part is untouched in all these LMS versions, that part untouched on this LMS version. So, here also I am only showing weight of that part getting y n and then d n minus y n equal to e n that is common, as in the LMS it is so in all these algorithms. So, I am only concentrating on the weight update part here, w n plus 1 is as before of this form, but you take a mu parameter mu tilde, divided by a small quantity say delta. I will tell you why I need the delta, actually delta was not needed, but delta I tell you why.

And this sigma square x n into the normal thing x n vector e n, earlier we had this into a mu. Now mu is a complicated thing, to calculate mu you need this quantity. So, it is a time varying mu, because sigma x n square varies with time. This delta n is given, because sometimes this norms square can becomes, so small then it can lead to a division mean it by 0 kind of thing. So, delta is added and this mu tilde for this algorithm to converge, mu tilde should be within this range I think which is, I am not sure this I think it is 2 to 0. Either 1 to 0 or 2 to 0, if a memory is right, it is 2 to 0 I will check and tell you later, for convergence I think its mu tilde 2 to 0, and you have to check I have taken, I think mu tilde this range then it converges.

So, you understand computationally you are doing more, you have to do this division firstly. At this thing, every time you do not have to calculate it can be obtained recursively isn’t it, because even you go to n plus 1 ith index, that vector will have some overlap with this 1 extra term will come and other term will go out. So, it can be made recursive kind of thing, but still that computation is required. So, if you do more computation, but this algorithm we have faster convergence rate this called LMS algorithm.

Now, I go to another form of LMS which I will deal with at length and may be it will take 2 or 2 days also, because it has got some DSP component, which I am not sure whether you know or not, some properties have DFT FFT and all those. So, again I have to come down to DSP in that course it called block LMS algorithm.

(Refer Slide Time: 38:11)


Block LMS Algo
x(n)
0-th
1-st
i-th
0
L-1
L
2L-1
iL
iL+1
iL+L-1
S/P
Buffer L +N-1
Proc.
W(i)
$R \approx \frac{1}{L}\sum_{\lambda=0}^{L-1} x(iL+\lambda)x^t(iL+\lambda)$
$P = \frac{1}{L}\sum_{\lambda=0}^{L-1} x(iL+\lambda)d(iL+\lambda)$
iL
iL+L-1

Here also our procedure is same as before that is we start with the steepest descent, we will start with the steepest descent. Then I said that look let me carry out this iteration, I in real terms I was replaced by n and R by this and P by I got this. Here before taking that step I am saying that, suppose this is your input you take a block of data say block of length L 0 to L minus 1 this block of data you called 0th. Then L 2 2 L minus 1 this is another block of data first dot, dot, dot said ith block it will start at I L, I L plus 1 dot, dot, dot I L plus L minus 1, I can be 0, I can 1 it can go even before 0, I am not drawing it, a block of data. I will say in this update means I will say that, I will carry out the iteration, but I is not replaced by timing index n, but let block index n.

While processing 0th block, I will call it by 0th iteration stage that means, what weight vector will not be changed from here to here to here where as a hold it constant. I will not do any update and all that, whatever you have giving me whatever calculated let it remain, that is my 0th iteration, weight for 0th iteration. Only when I go to a next block then it is my second iteration. So, that time I need an weight update, so I go from 0th iteration to fist iteration again used that do not change it at this intermediate indices, but use the filter, to use that those weight you go on filtering. Because you need to apply the filter at every point, at this point, at this point, at this point filtering goes on.

Only at the next index, I mean when this next block comes the second block, that time again you carry out weight update, there is that is the second iteration comes. So that means, within each block I am not changing any weight, weights are remaining fixed, I am doing like a ordinary if I have filter, filter it. Only at the block transitions from L minus 1th to Lth when I move, I am making a change this is what I do. In hardware I am not comparing it, but in hardware what is do actually it means you know, there is serial to parallel and a buffer length, length how much, to start with assume length L, but it will not it will not be length L it will require you will require little more than that.

So, input data is held in this buffer and then when the buffer is full pass on to the next stage processor. Processor works on it processor knows what is the weight vector for this particular block of data, say Ith block of data. So you will not, it will not change the weight vector it is simply use the data and go on filtering and give you y n at all the points. At only when this block is finished, it will carry out a weight update to get your next iteration. Then by that time, that computation is done these buffer is getting filled.
So, when the computation is done these buffer is filled with that next block of data, it will accept the next block and again go on filtering without changing the weight and only at the end of that block, just carry out a weight update and again accept the new block of data, that is how is operating.

In time, this way processing by the way suppose you are here, this block has got weight w i, I am not yet done weight update, but even while filtering you can see w i. There is a vector what w I index, I indicate what, filter weight for the Ith block that is constant, it will not change here or here or here no updating. But at this point of time, what is a filter output w i transpose into x vector, which vector x i L. So, it will consist of not only a data from this block, some tail from the previous block this is very important. Tail from the previous block is required, how many? n minus 1 capital N minus 1 because capital N is the lay order of the filter.

For the time being you can assume just for making your life simple, that length L that is the block length is larger assume, just assume to kind of you know, I mean draw the picture in your mind. That this length L, L is larger than the filter length N so that, here and just few more samples from the very adjustment block only, block only coming, but it is no restriction you can easily adjust for your ease of imagination, ease is in imagining I am saying that anyway. So, x i L and n minus 1 then next at this index w transpose I times the data vector, for index that will consist of this data, this data and n minus 2 data so on and so forth.

So, essentially what you need you need that Ith block data plus n minus 1 data last n minus 1 data of the previous block. So, this buffer size actually this and in real time operation I am not, I have not done into weight update all. But before that I am showing you in real time what is happening that suppose, this is the Ith block, this up to this time you weight this data, this data, this data all this data you store in them buffer. And n minus 1 data, already we have their inside you do not have to restore it, I mean recapture from here they are already part of it. So, their stored have these data stored, only at this point when this thing is done, you give it to the processor.

Processor will do what, at only in the indexes it will give the filter output only, no weight updating till you reach the end for and then, you do the weight updating. So, your first output filter output comes at this index or rather this index that means, there is a delay what was earlier coming in the LMS case, at this point of time that we get delayed and that will appear at the in the same time scale. At this point of time this is called latency, those who have done my VLSI course they can see this typical block processing. Because I am first storing them. And then, I am not processing this data as the time when it arrives, I am holding it, till the buffer is filled then passing on to processor.

So, the output corresponds to this index will not come at this index, but will come at here. So, there is a L cycle delay and then here, here, here only at the end it will do a further computation of weight update. So that means this, this clock this clock width will be broken by what, not just time taken to compute filter output as here, but also at the end filter output has to computed plus mu weight has to computed by an weight update equation. That time, so both the time you have to consider that together must be what the clock period must be greater than or equal to that. This is a simple hardware question are you following me?

Not because at this point no updating only filtering, but that does not mean because only the only filtering means much less computation, but only at the at the end when you go for the next block, before that weight have changed by a weight updating. Some appropriate weight updating equation, which I will do now. So, this is a last sample, so before between these and the next sample, you have to carry out filtering for this and weight updating. So, time taken for both have to be added that time should be what, clock period must be equal to, must be greater than or equal to that time .

These are real life, I mean what is happening in real hardware. Now, coming here to weight update, weight update I will follow the same ((Refer Time: 47:09)). So, I will replace by I only, but here the meaning of I is Ith block I iteration means, Ith the weight vector for the Ith block, that will be our Ith iterate. Then, this will move to the weights will change to i plus 1th block, that is of another weight that will be my i plus 1th, same iteration I will carry out. But what this period of time, I am not changing the weight, I am keeping it I will go on filtering. Only at this point I will again move, carry out by next iteration like this.

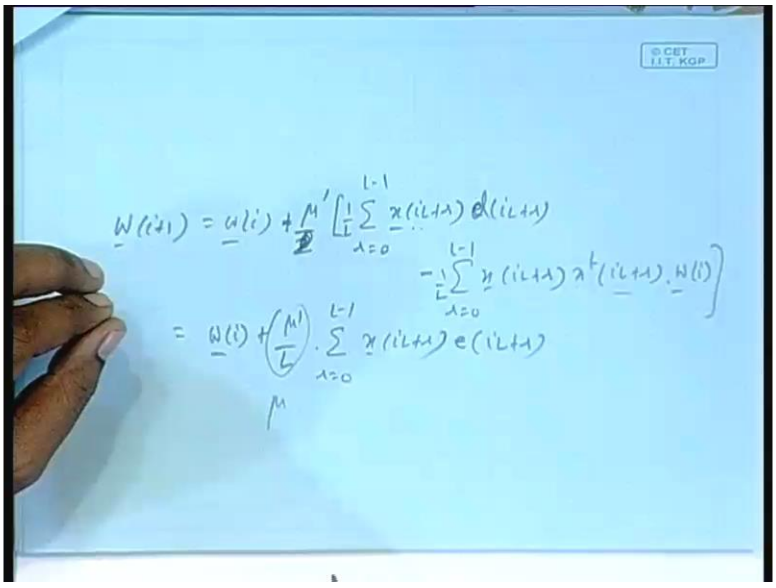
And that will be my i plus 1th iterate which will also equivalently the filtrate vector for i plus 1th data block so on and so forth, but R and P here I have the advantage, but that what is R here I R i replaced by this, because in the case of LMS I was doing point by point no block business. X n into x transpose n or x n plus 1 into x n plus 1 n transpose so on and so forth. But here, since the weight is not changing and I have all the filter output e n everything available here also, here also, here also, here also for the same weight, for the same weight, isn't it. For the same weight I have got so many error components available e of n, e of n here, e of n here, e of n here, e of n here. So, statistical availing can be done in a better way.

There are so many sample are there for the same en, where weight is not changing weight is same. Earlier I had only 1 en, because weight was changing at every index, but here for so many points weight is not changing, I have given the same weight I am finding. One sample of en another sample of en, another sample why not use of them to get a better estimate or P say. Similarly, I have got a data vector here, data vector here, data vector here, data vector here you can use all of them to get a better estimate of R.
So, our better estimate will be what, data vector at this I Lth point time is transpose plus data vector at this point time is transpose plus dot, dot, dot data vector at this point into its transpose, all added divided by L there will be better estimate.

Earlier, I just working was used the wild estimate just x n into x transpose n, but this time I am say no. I will have similar cases, because I have the luxury because weight is not changing within this block. So, till the weights change till the weight as long as the weight remains stationary, I will consider all the weight data vector, I do not have to change over to another weight. I will carry out this thing and en most importantly en, because x you can always even in LMS case, if you wanted you can do always you have taken many sample of x n vector. Problems comes in the en, because weight is changing at every index, weight is changing at a every index. And therefore, you are not getting the same en, I mean you have now various sample of en for the same weight vector.

Here I do not have that, I have got 1 weight vector common and I have got various cases, various outcome experimental outcomes like you know for en. So, I can use them for averaging. So that means, what I will do here R will be 1 by L, R will be replaced by 1 by L into x I L plus r, r transpose I am doing for the real case, but I am sure you can extend it to complex case, that is not a problem not r, r equal to 0 2 L minus 1 you follow this isn't it. R equal to 0 means data vector at Ith point of time, I equal to 0 means data vector at this point of time into x transpose. R equal to 1 means data vector at this point of time into x transpose, here all getting added and then divided by L it is a good average. Similarly, P will be what P is after all excepted value of x n into en, is it not? P will be what 1 by L data vector at various indices multiplied by error at those indices added averaged, oh sorry, I am very sorry, d.

(Refer Slide Time: 52:00)



$$\underline{W}(i+1) = \underline{w}(i) + \frac{\mu'}{\cancel{2}}\left[\frac{1}{L}\sum_{r=0}^{L-1} \underline{x}(iL+r)\, d(iL+r) - \frac{1}{L}\sum_{r=0}^{L-1} \underline{x}(iL+r)\, x^t(\underline{iL+r})\,.\,\underline{W}(i)\right]$$

$$= \underline{W}(i) + \left(\frac{\mu'}{L}\right) . \sum_{r=0}^{L-1} \underline{x}(iL+r)\, e(iL+r)$$

$$\mu$$

Or if you put them back in that weight update equation w i plus 1, that is for the, i plus 1th block or i plus 1 iterate will be same as w I then mu by 2. What is the gradient sorry, gradient is as it is for the gradient these are gradient. So, that 2 and mu i 2 that will cancel, let me could bring it mu prime, because it is not mu it is a mu prime, you are following, isn't it. This 2 and 2 will cancel, so I can forget out this 2. R n P will be replaced and there is a minus sign. So, it will become plus P minus R w, you will be replaced by your this term xi L plus r e i L plus r minus into w, w i or w i and then, resample take the summation out, take this for it is, very simple you know take the first component as common out, the summation out, first component xi L plus r xi L plus out common.

So, I left with, so this is d, left with this minus x transpose there is a 1 by L here, there is a 1 by L here. So, mu prime by L quickly I not going through a steps, this is common within bracket you have the I L plus r minus x transpose for the same index multiplied by w i, that is a filter output at this index filter remains I, w i not w i L plus r, ith block weight. But x transpose I L plus r means, data vector at i plus index that times w i filter out put d of d minus that. So, there is an error so this and mu prime by L, you can call it now mu, this is block LMS algorithm this computation can be done. This computation we are filtering part can be done very fast by using FFT, which we will take up later, but we have do some convergence analysis of that also, little bit convergence and mean, that we will take up later nicely.

Thank you very much.