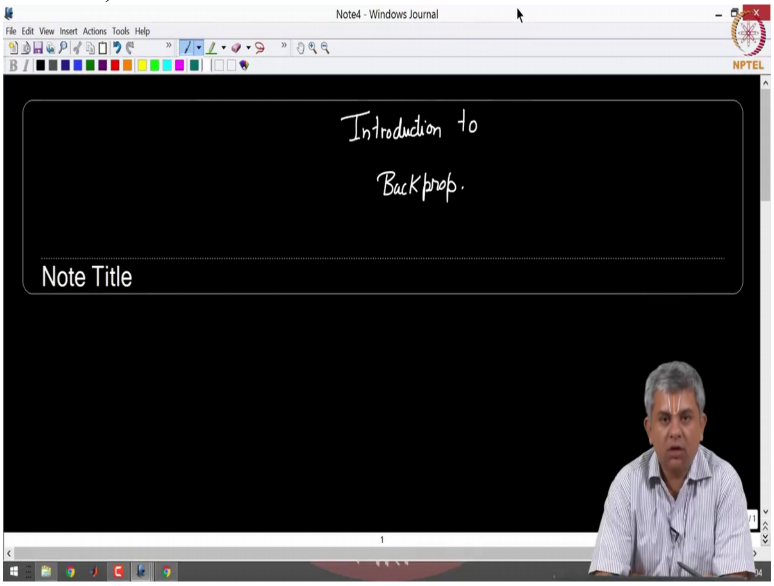
Machine Learning for Engineering and Science Applications
Professor Doctor Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology Madras
Introduction to back prop

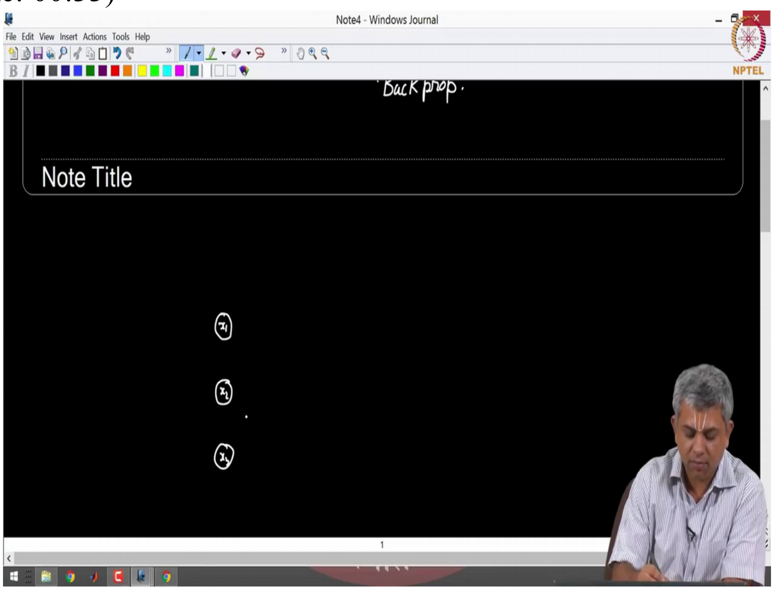
(Refer Slide Time: 00:13)



Welcome back. In the previous couple of videos you saw how a neural network does its forward pass. Recall that when you have some neurons, let us draw a simple neural network of that sort.

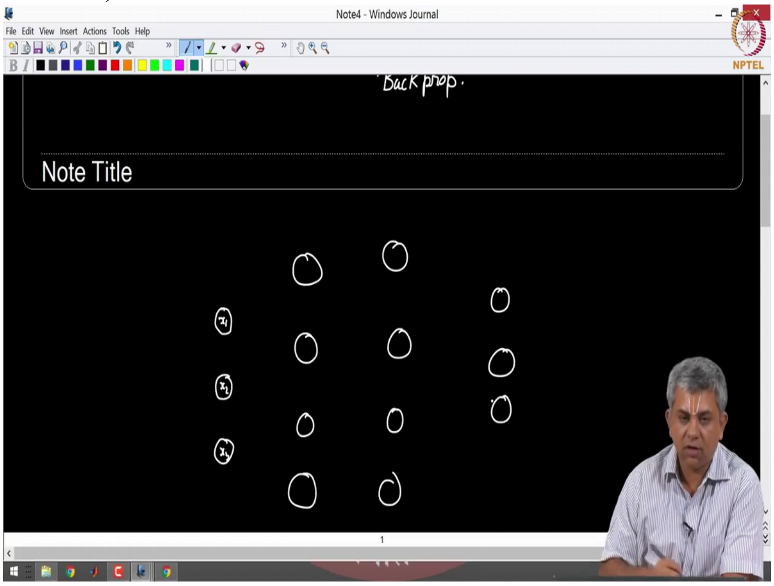
So let us say you have 3 features here, x 1, x 2, x 3

(Refer Slide Time: 00:33)



and some 4 hidden layers here, another 4 here. And let us say we have

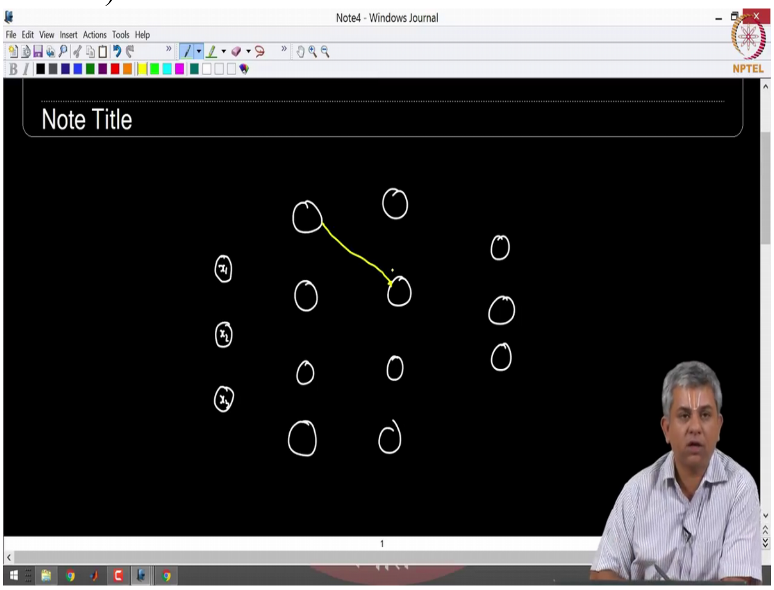
(Refer Slide Time: 00:42)



3 as output layer, Ok.

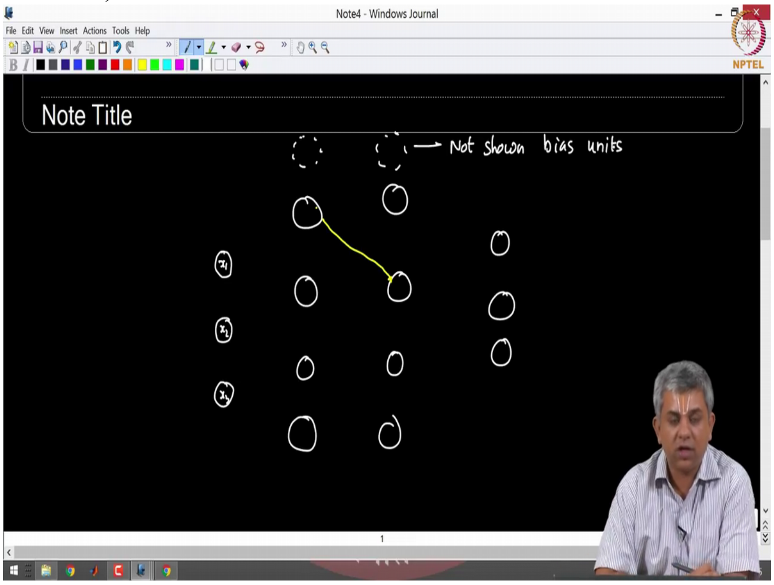
So let us say we have a neural network which looks somewhat of this sort. Remember that

(Refer Slide Time: 00:53)



each node is connected to each subsequent nodes. Also notice that we typically do not show the bias units, Ok.

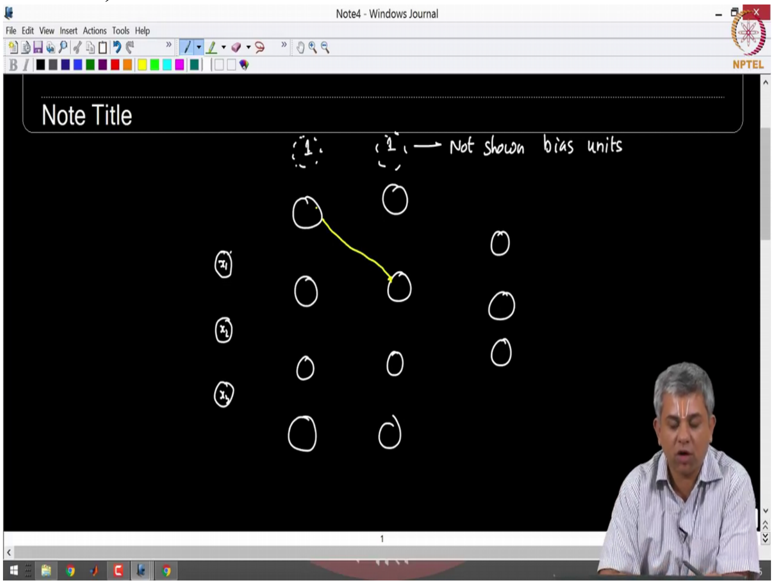
(Refer Slide Time: 01:09)



The reason is very simple.

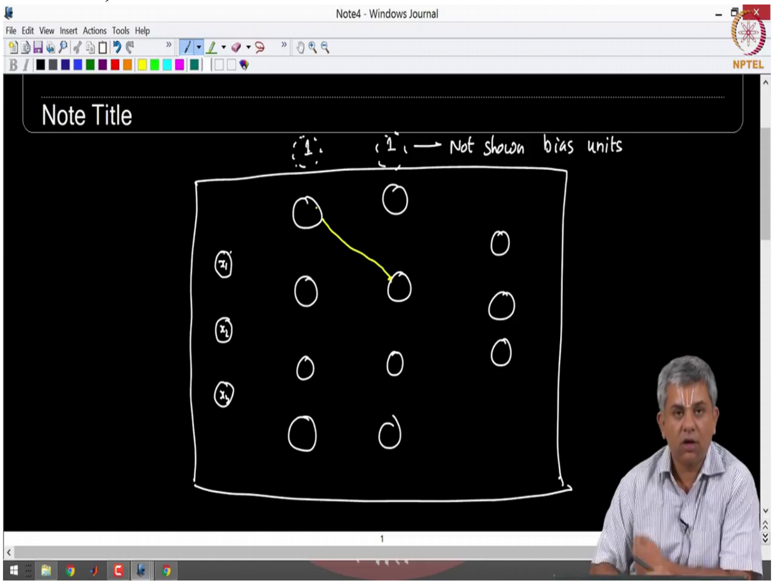
The reason we do not show a bias unit is there is nothing that goes from here to here. Because this does not affect this unit which is always 1,

(Refer Slide Time: 01:18)



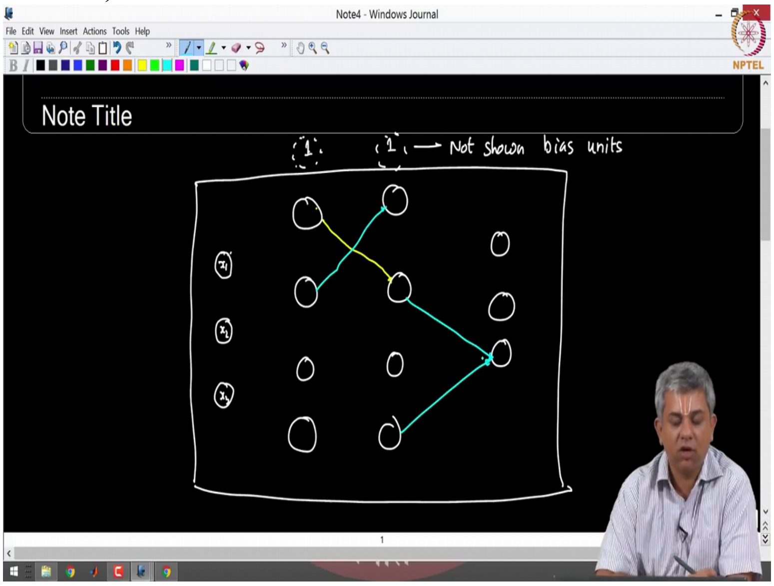
which is why we do not show it. So when you see a neural network diagram you typically see only this portion, Ok.

(Refer Slide Time: 01:27)



Now notice that other than this, every node is connected to every other node. Ok so for example this is connected to this, this is connected to this etc. I will not,

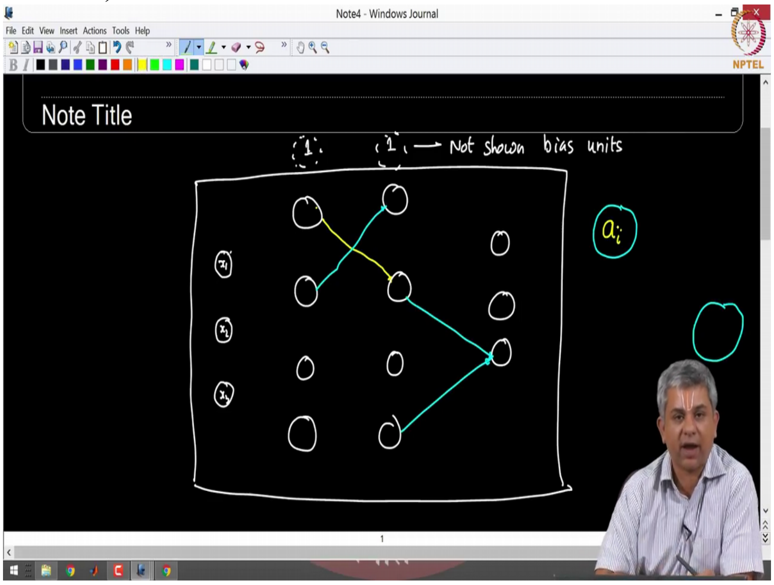
(Refer Slide Time: 01:38)



you know mess this up by drawing every single thing here.

So if I generally consider a node in one layer, let us call this a i, remember

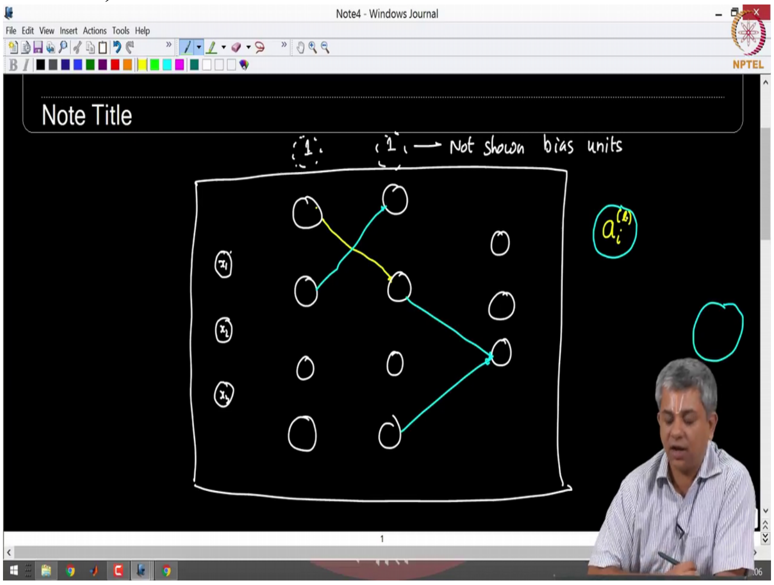
(Refer Slide Time: 01:51)



a stands for activation. The activation comes after the two operations have happened, after the summation and after the nonlinearity.

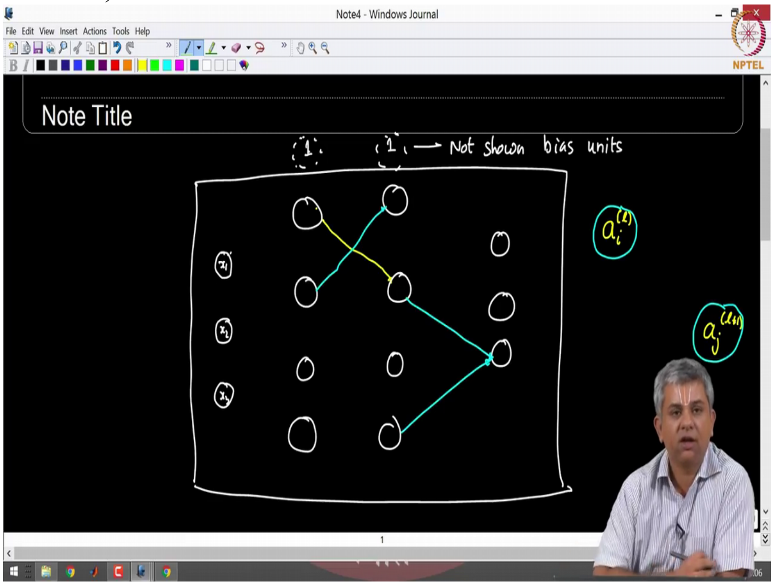
Ok so if I look at this a i which is sitting at

(Refer Slide Time: 02:03)



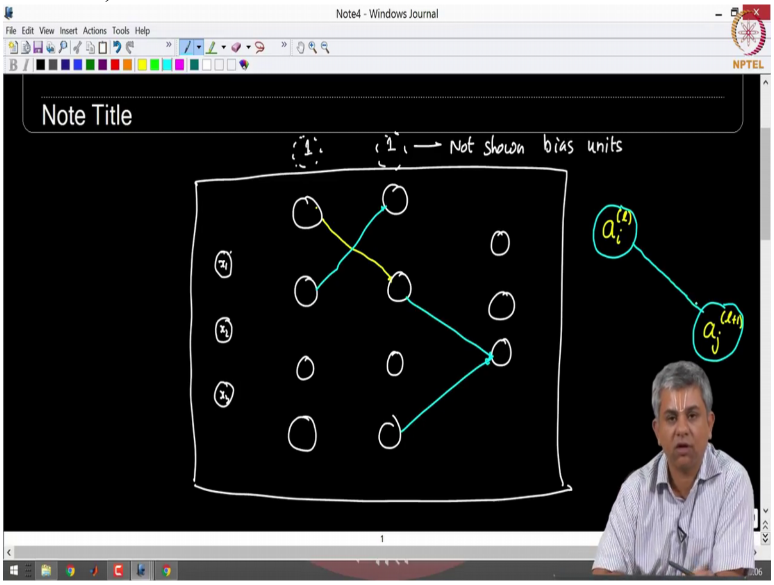
level l and a j which is sitting at

(Refer Slide Time: 02:08)



level l plus 1, the two are connected by a single line,

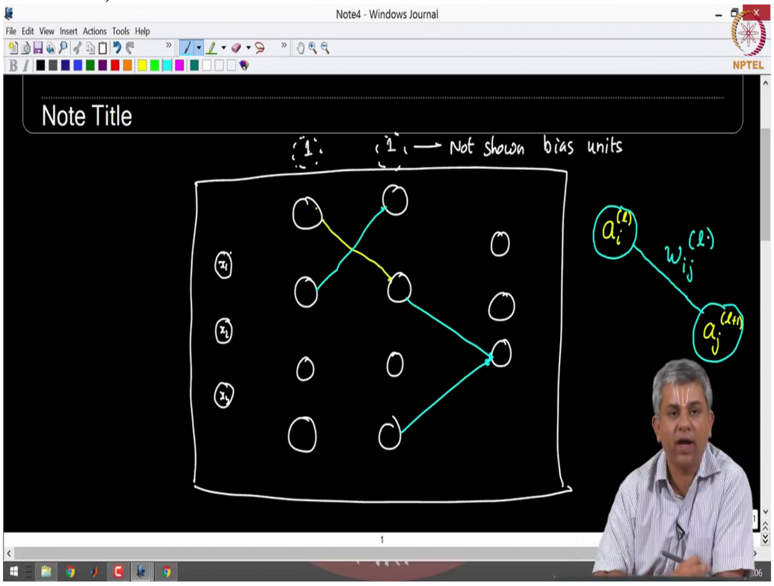
(Refer Slide Time: 02:15)



Ok. It is sort of like the Myelin sheath, anyway.

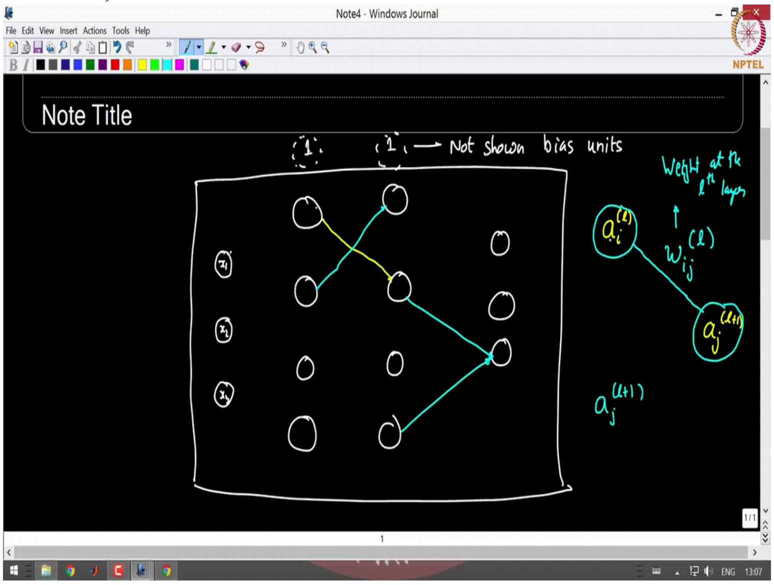
This we will denote by

(Refer Slide Time: 02:23)



w i j l. w i j by itself, that value is a scalar. It is a single value. It tells you that a j l gets some contribution from a i l and the portion of that contribution is multiplied by w i j of l. Ok so this is the weight at the lth layer connecting the a ith

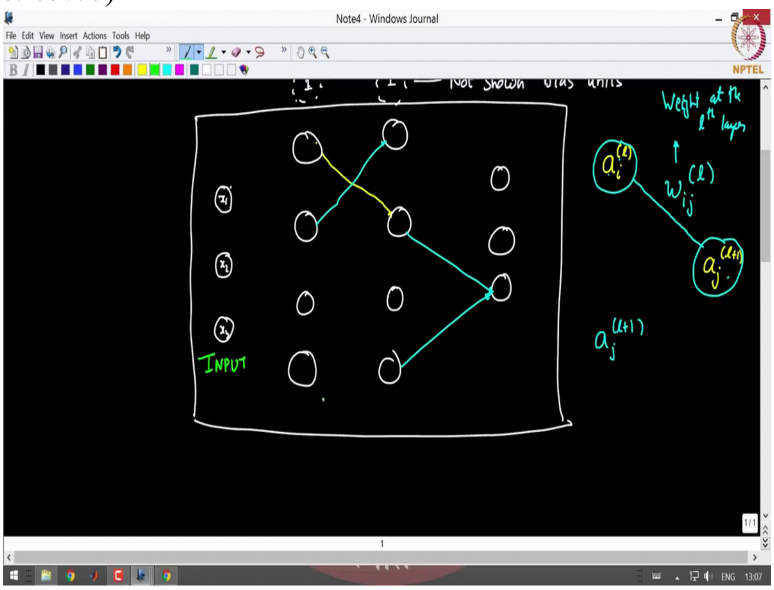
(Refer Slide Time: 02:54)



neuron of level l with the jth neuron of level l plus 1.

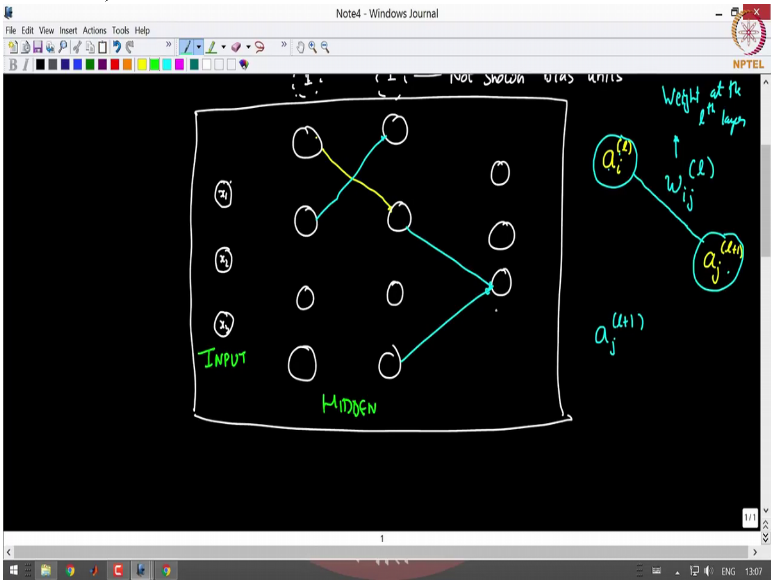
So this notation is actually pretty straightforward. Remember this was the input layer,

(Refer Slide Time: 03:07)



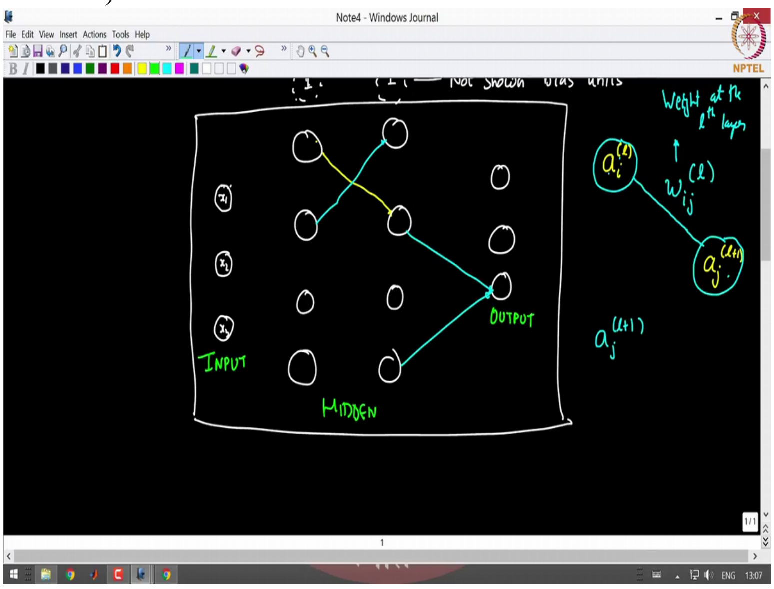
these are the hidden layers

(Refer Slide Time: 03:10)



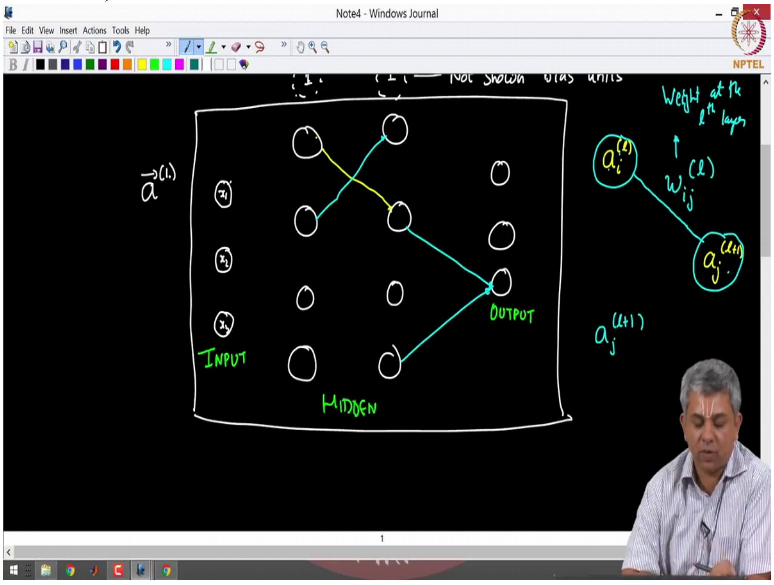
and this is the output layer.

(Refer Slide Time: 03:13)



Now depending on what notation scheme you choose, you can call this x 1, x 2, x 3. Or some people and I will also do so; we choose to call it a vector at level 1,

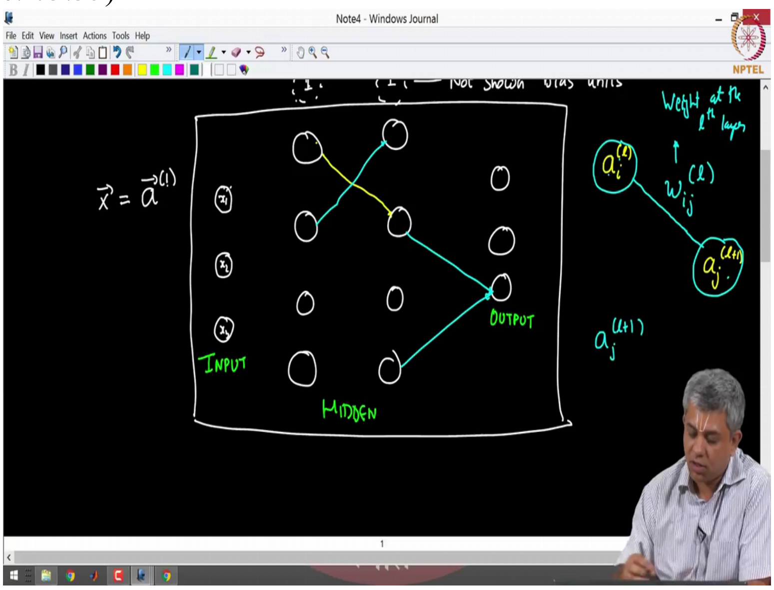
(Refer Slide Time: 03:29)



Ok.

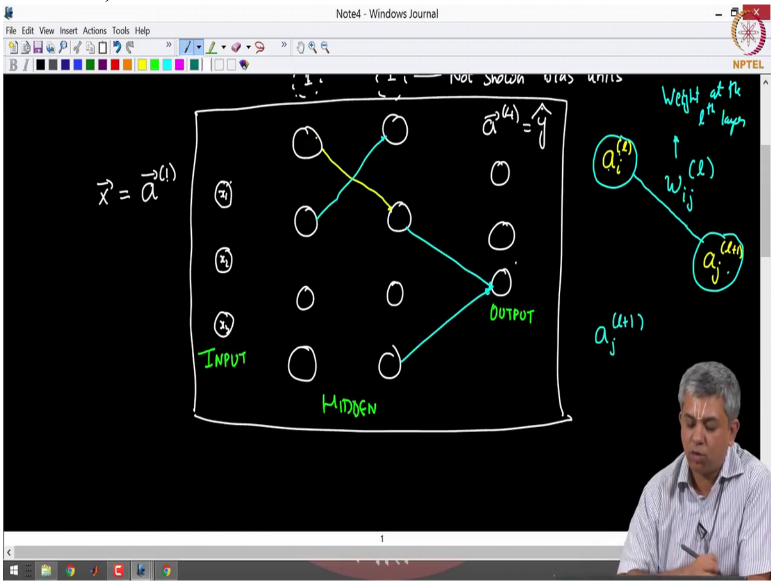
So level 1 simply is input vector.

(Refer Slide Time: 03:35)



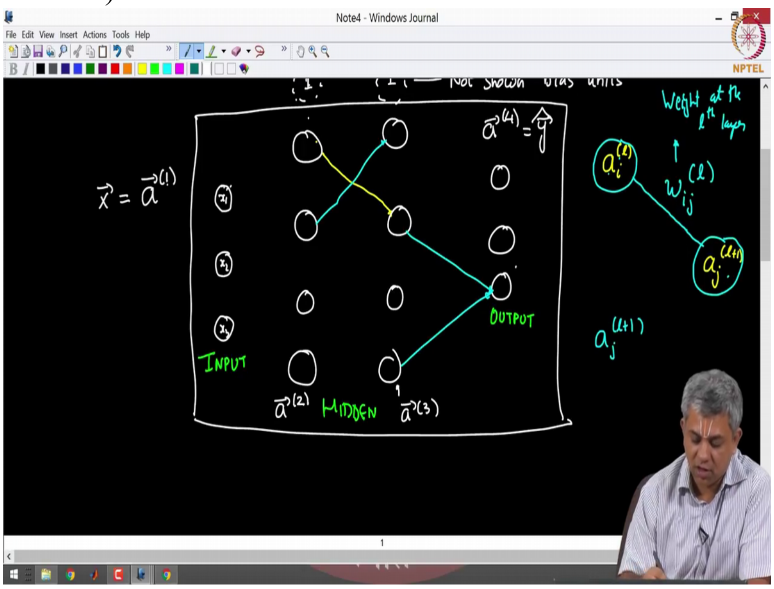
Similarly this I could call a vector at level 4 which is the

(Refer Slide Time: 03:41)



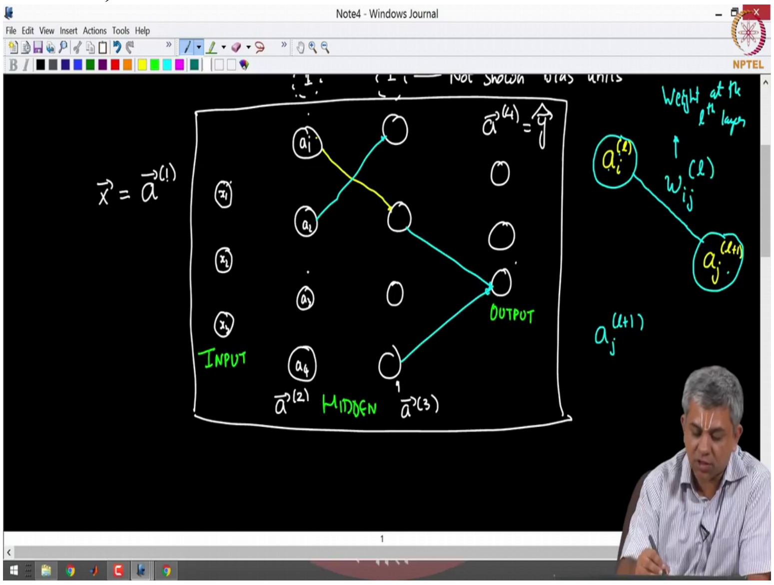
output layer, y hat layer, Ok. So this then would be a vector at level 2. And this would be a vector at level 3.

(Refer Slide Time: 03:53)



Remember a vector itself has 4 components, a 1, a 2, a 3, a 4 with the

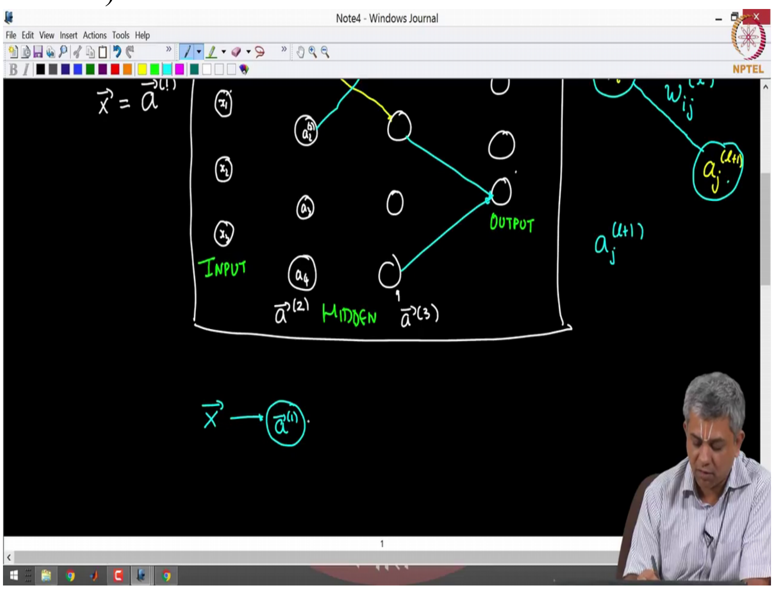
(Refer Slide Time: 04:00)



superscript 2.

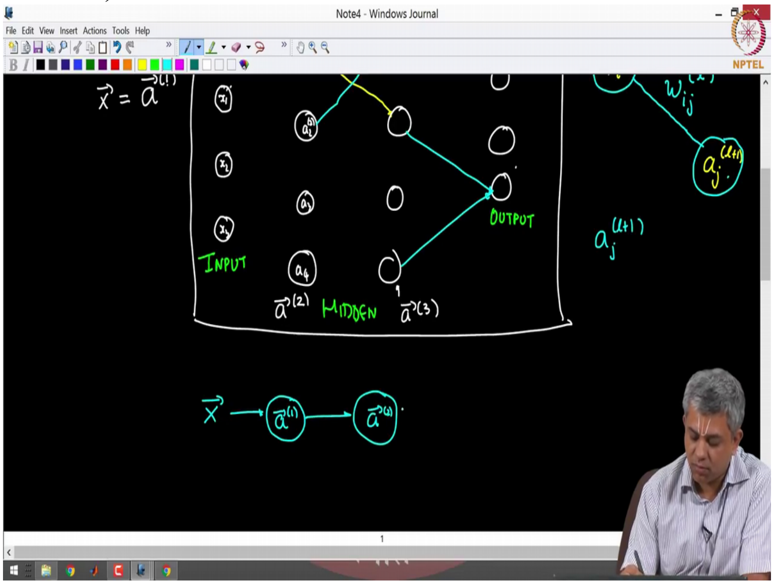
Similarly a 1, a 2, a 3, a 4 with the superscript 3. All this is so that I can kind of abstract this out and show this as x vector which somehow gives me a vector at level 1

(Refer Slide Time: 04:21)



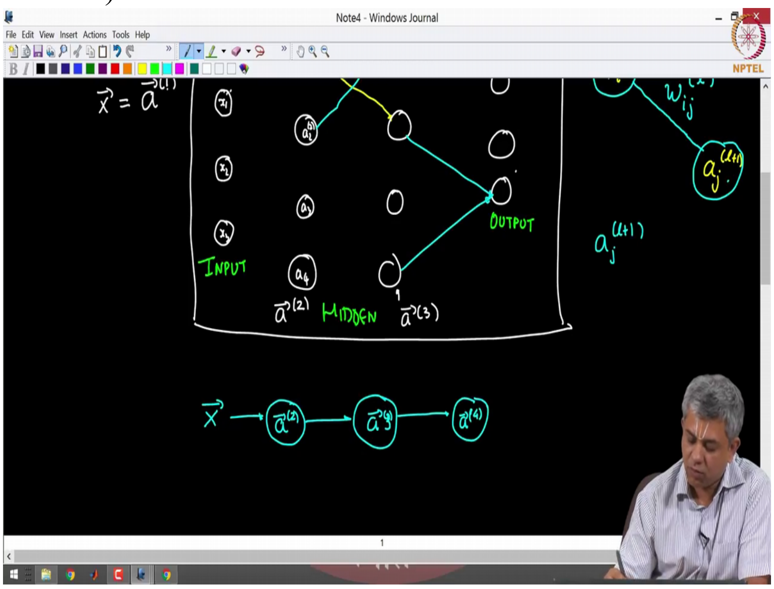
which somehow gives me a vector at level 2,

(Refer Slide Time: 04:25)



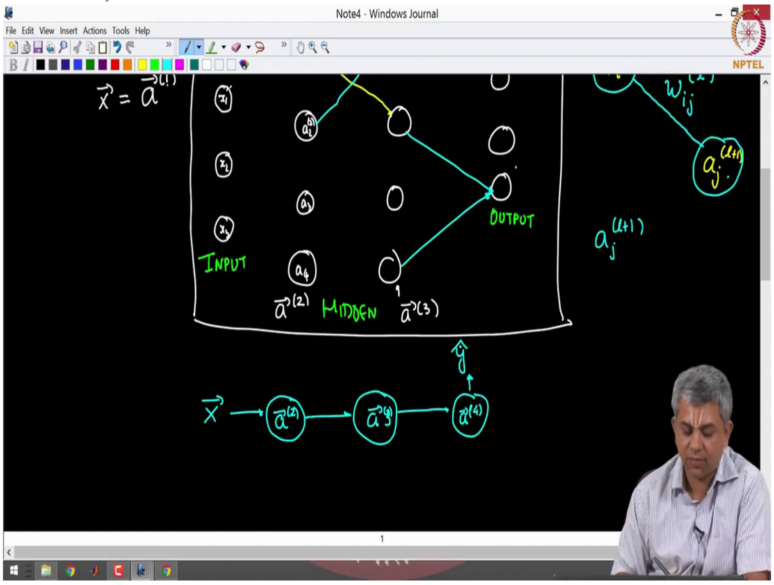
sorry, based on our notation I should call this a vector at level 2, a vector at level 3, gives me a vector at level 4

(Refer Slide Time: 04:47)



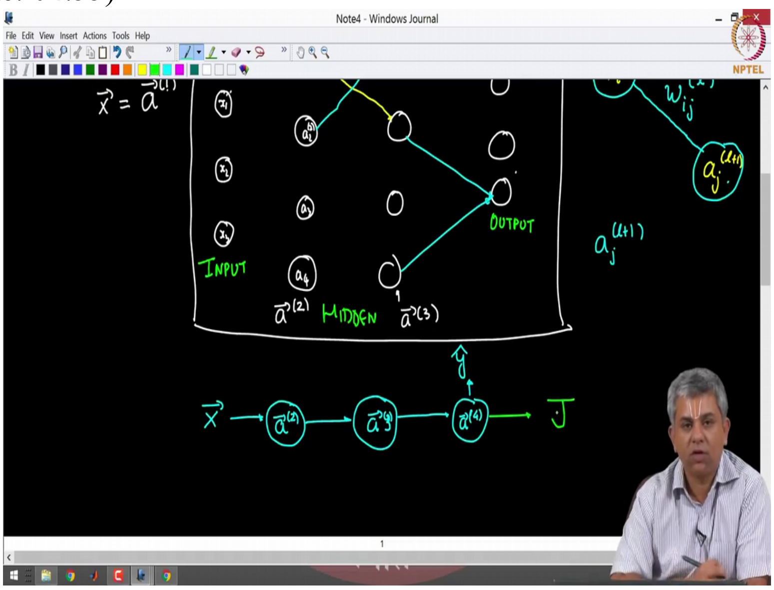
which is the same at y hat.

(Refer Slide Time: 04:50)



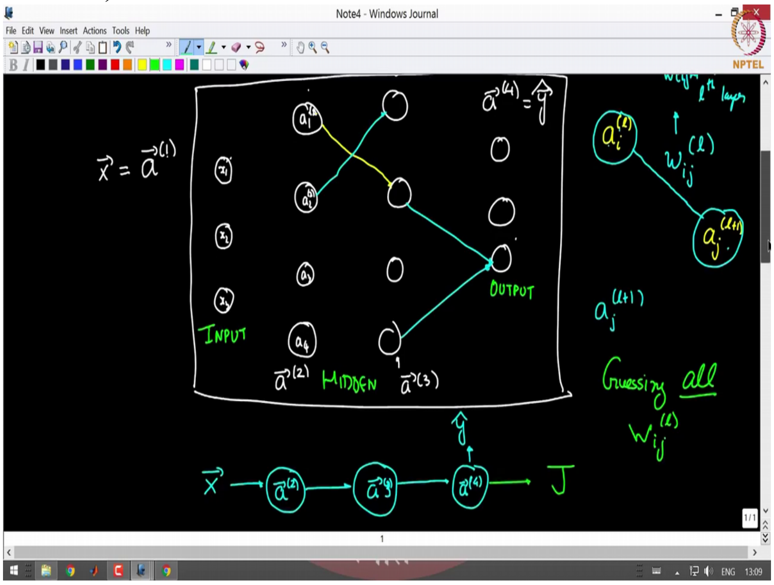
Now at the end of this

(Refer Slide Time: 04:55)



y get your cost function J. Now remember in all our procedures what we will be doing is we will be guessing for all of the weights. As you can see

(Refer Slide Time: 05:16)



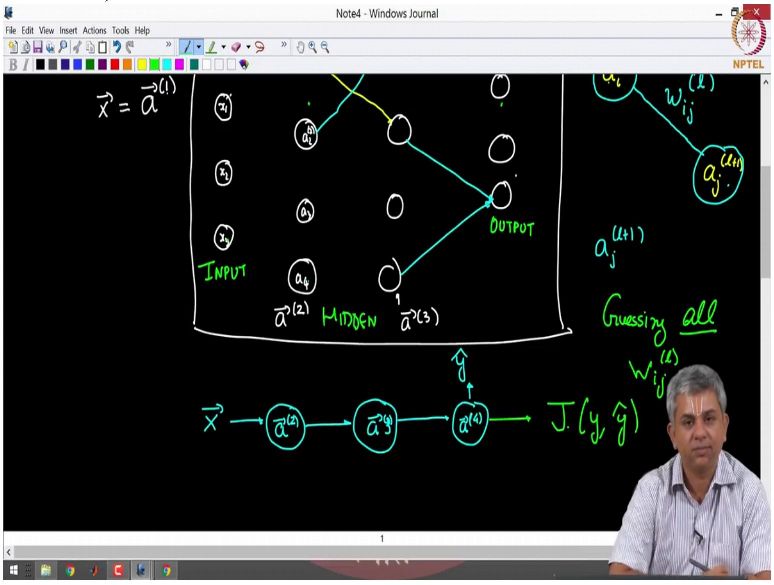
even in this simple diagram there are a lot of weights, Ok.

So you have 3 neurons here, 4 here. You have 12 but actually more than 12 because you have your bias unit also. But let us just talk about the weights other than the bias. So there are 12, 16, another 12 and then add the bias units also. You have that many unknown weights. So you have to initialize all of them by simply guessing.

Once you guess you get a cost function J. Now ideally you would want that cost function to be 0. You know that that is not going to be the case because your guess is typically not going to be so good.

So this J is the function of y, the ground truth and your y hat.

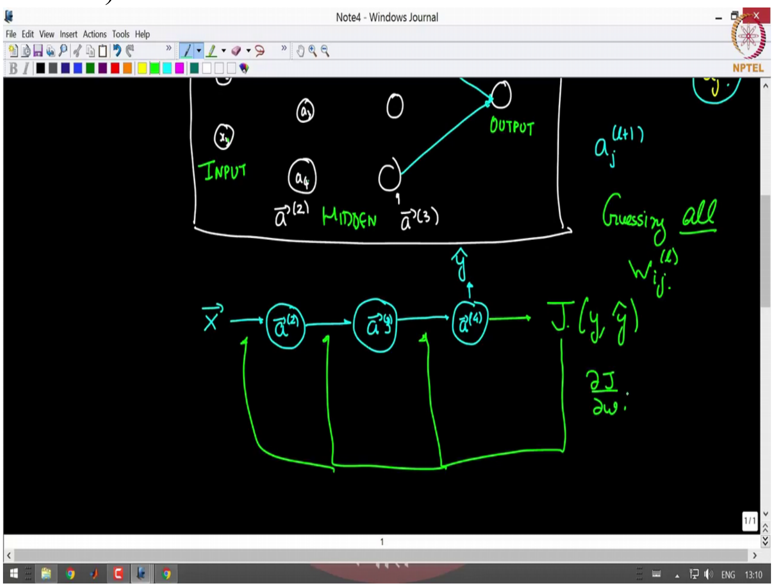
(Refer Slide Time: 05:55)



And given that you are going to get the J, you have now to figure out which of these weights was responsible for this higher J. What you want to do is essentially redistribute this J to all these weights, Ok.

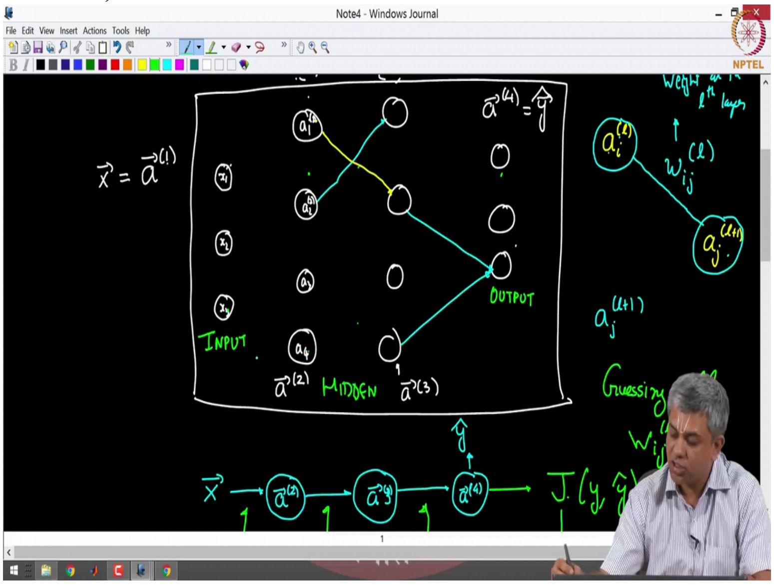
Remember

(Refer Slide Time: 06:19)



that some of the weights are just here, some of the weights go back here, some of the weights go back here, Ok. So this procedure

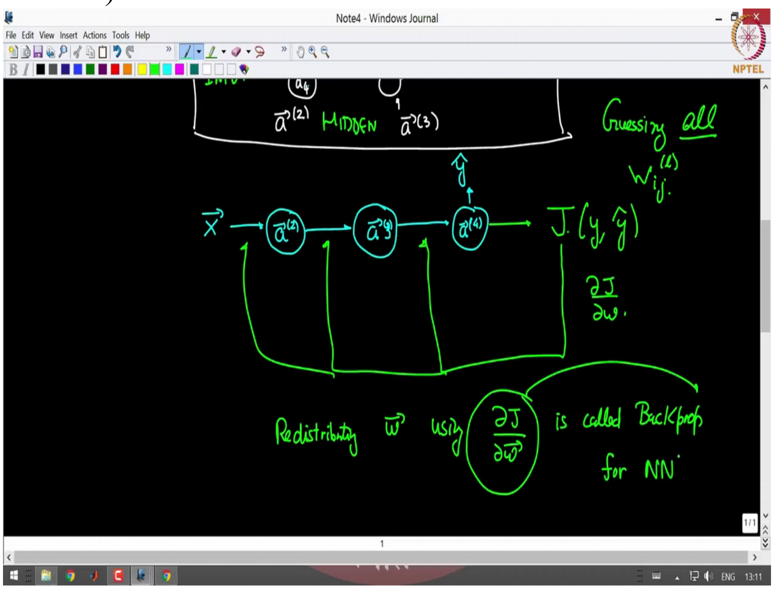
(Refer Slide Time: 06:26)



of redistributing w using del J del w is of course called gradient descent but just calculating del J and del w is called back prop or back propagation, more informal, more formally for neural networks, Ok.

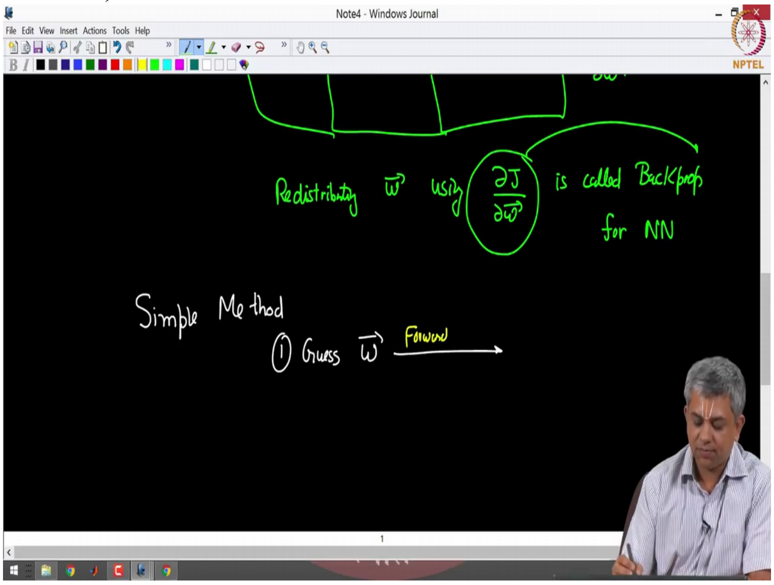
So whenever you hear

(Refer Slide Time: 07:02)



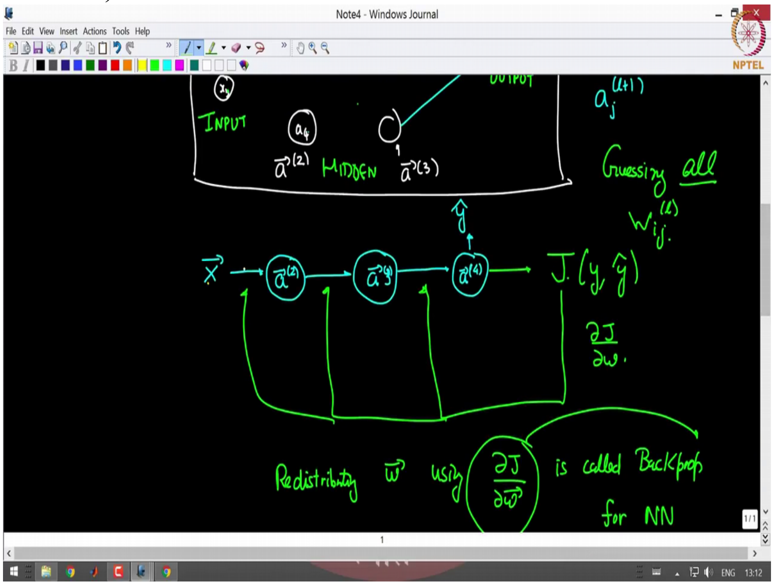
back prop, please remember this. All back prop is doing is calculating del J del w. So what is the big deal in calculating del J del w? Why not do it in a simple way? So there is a simple method. It is like this. You guess w, do a forward pass. What does the forward pass mean?

(Refer Slide Time: 07:32)



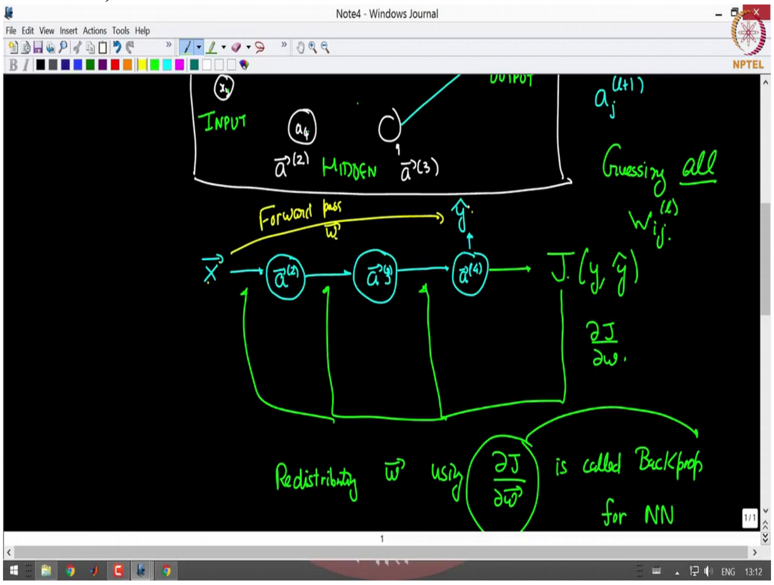
For a given x,

(Refer Slide Time: 07:34)



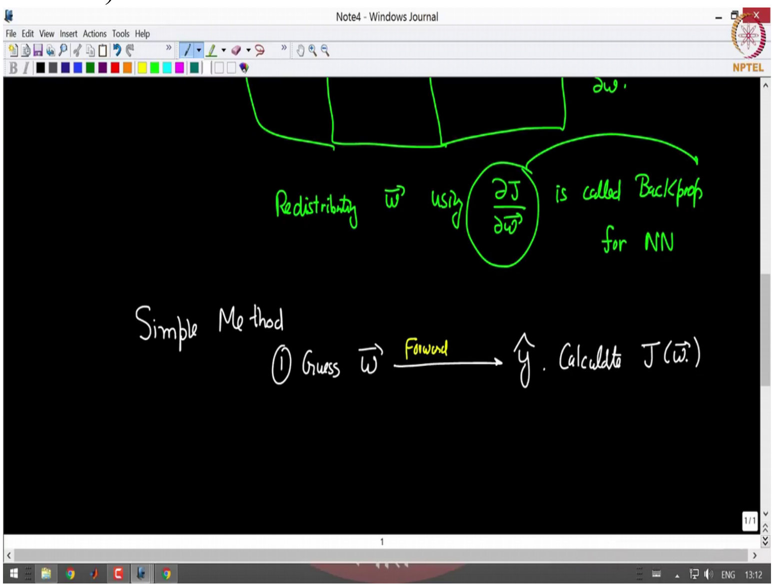
put those ws, calculate y hat. That is what the forward pass means. Remember this. That is why we call it a feedforward neural network, Ok. So forward pass is simply going from x, using some w

(Refer Slide Time: 07:48)



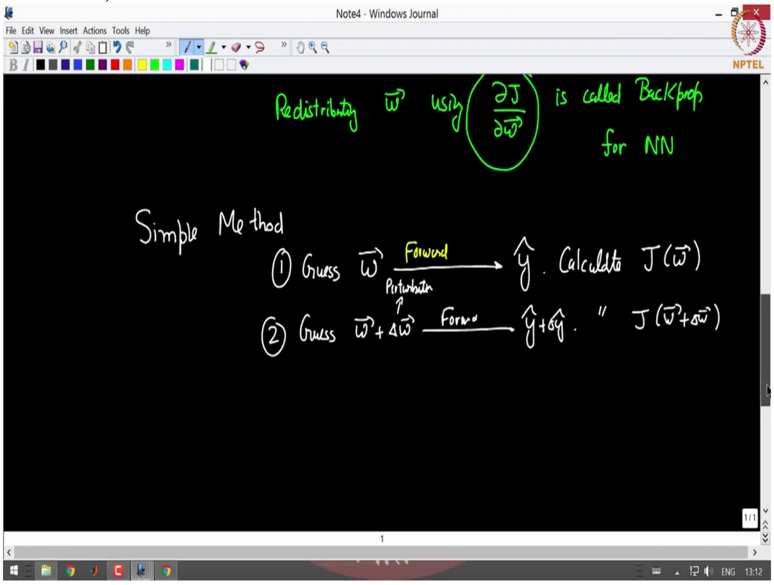
to y, y hat, Ok. Do a y hat, do a forward pass, get y hat. Calculate J of w.

(Refer Slide Time: 08:03)



Now make a slightly different guess. This is some perturbation. Again make a forward pass. You will get some slightly different y. Calculate J of w plus delta w,

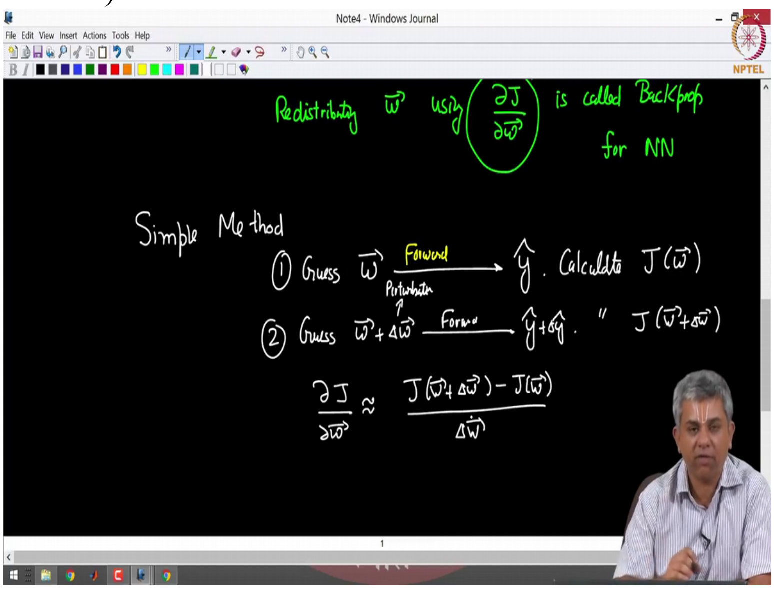
(Refer Slide Time: 08:30)



Ok.

Then del J del w is approximately equal to J of w plus delta w minus J of w divided by, even though you cannot divide by

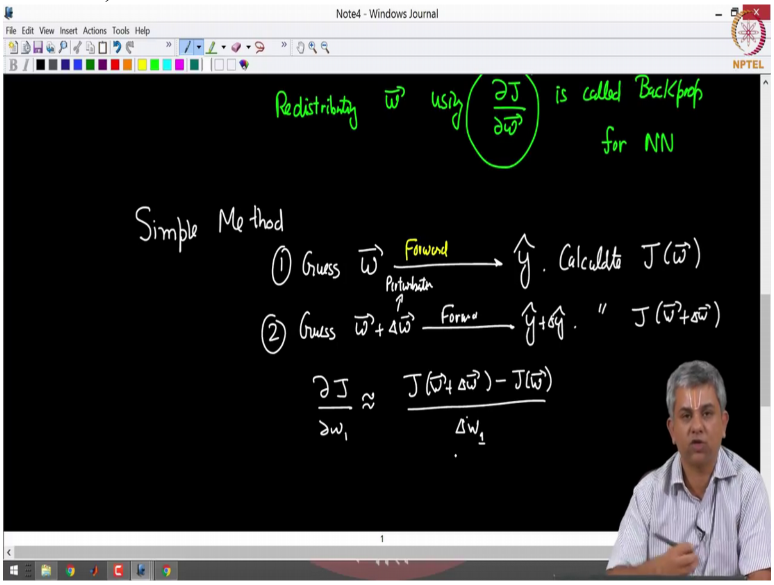
(Refer Slide Time: 08:46)



a vector, there are technical ways of doing it.

You do for, if you want del w, del J del w 1, you do delta 1,

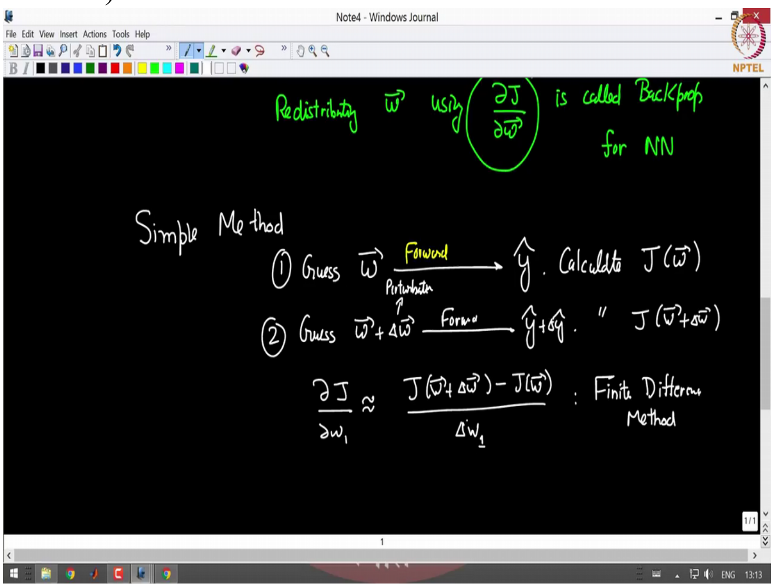
(Refer Slide Time: 08:59)



so on and so forth, as we say in the partial derivatives example when we were doing multivariable calculus. So this is called the finite difference method.

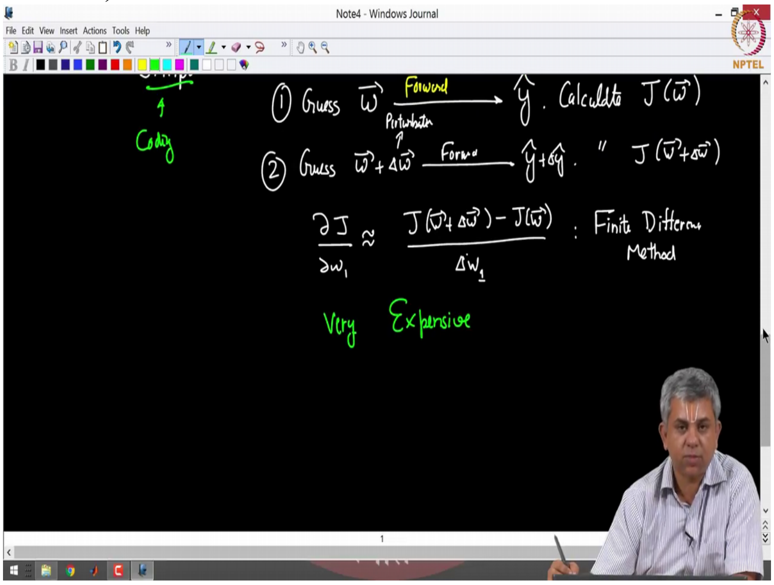
Now what is the problem here?

(Refer Slide Time: 09:17)



Why not use this always? The problem historically with neural networks was this is, even though it is simple, simple in terms of coding, it is very simple to code. But it is very expensive.

(Refer Slide Time: 09:36)

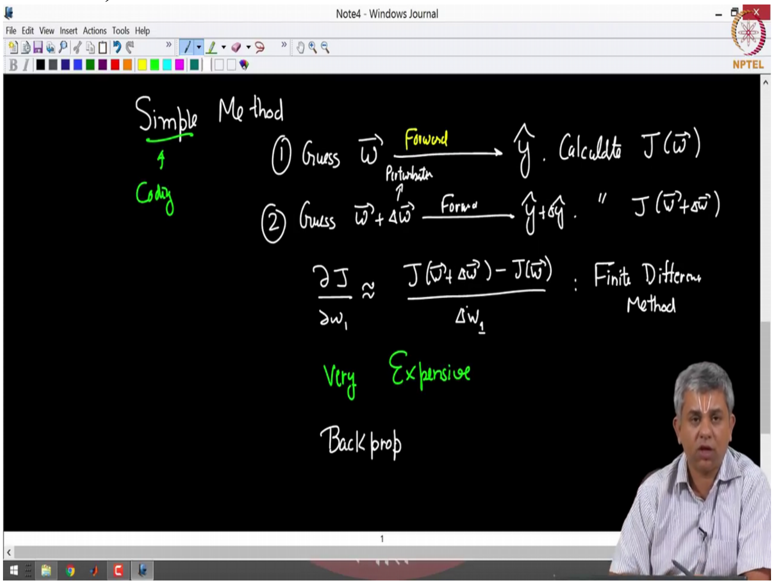


Why is this expensive? It is expensive because for each forward pass, for each gradient descent pass that you have to do, you have to calculate multiple of these del j del ws, Ok. For each parameter you will have to calculate del J del w and you could have millions of parameters.

So these are millions and millions of calculations and for each calculation you will have to do 2 calculations, J of w and J of w plus delta w. This is very, very expensive. So this turns out to be extremely expensive. Until the 60s, 70s also there was no easy way to do this and which is why lot of people did not do large networks.

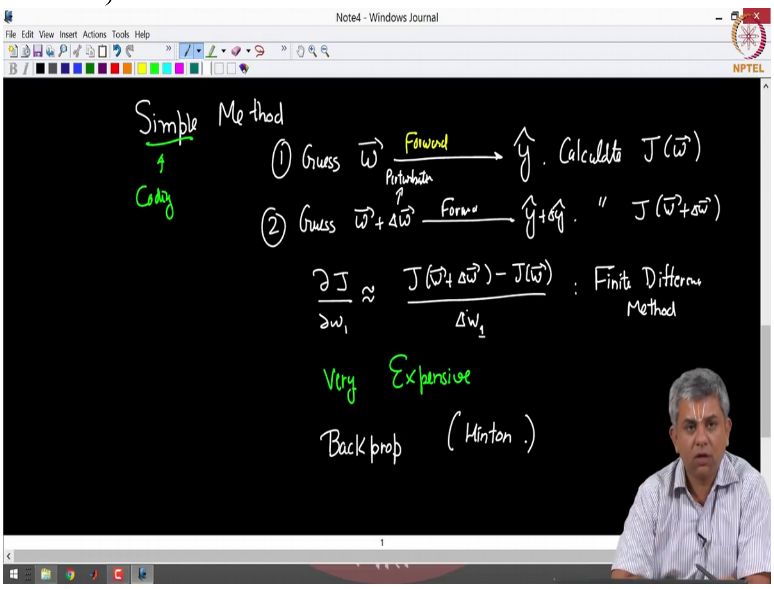
Till came the algorithm for back prop,

(Refer Slide Time: 10:24)



called the back propagation algorithm. So sort of the founding father of neural networks,

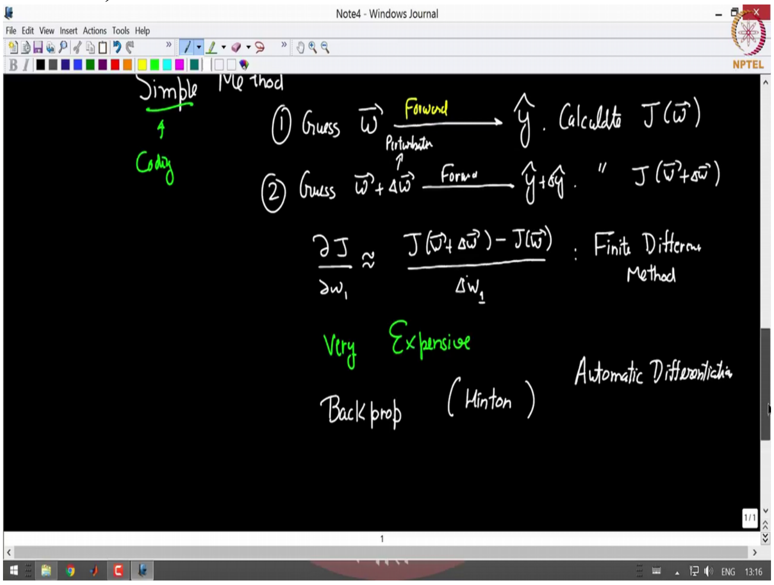
(Refer Slide Time: 10:30)



Hinton was one of the people who wrote a classic paper on back propagation. This is application of neural networks to what is called automatic differentiation.

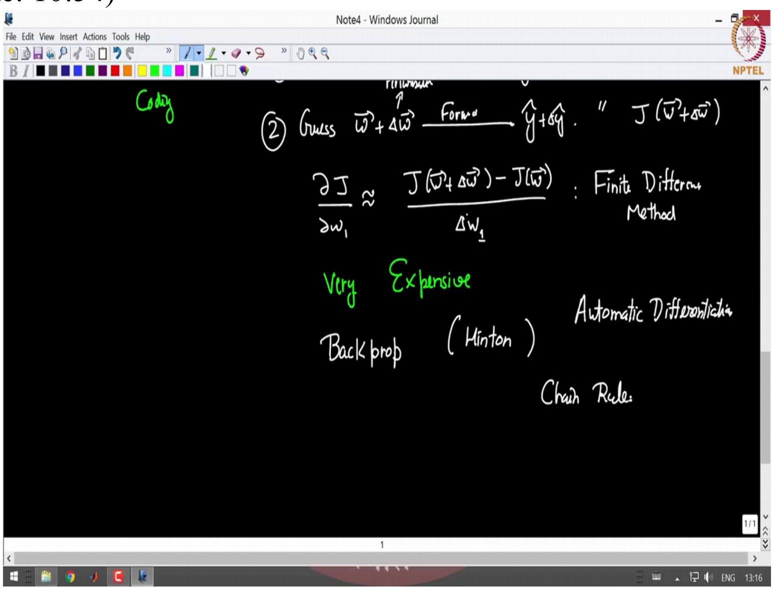
All these are fancy names. Basically

(Refer Slide Time: 10:50)



what we do is we use the Chain Rule. And it is

(Refer Slide Time: 10:54)



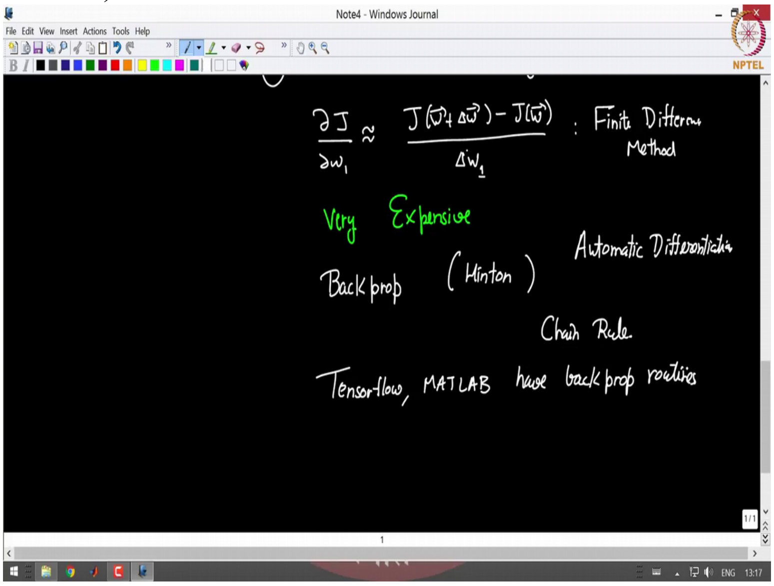
very similar to what I did in logistic regression, Ok. So we will be using the Chain Rule.

I am not going to do the full back propagation algorithm. This is not that kind of course. Programming it is also very, very difficult. Kind of ironically, programming finite differences is very easy but it is very expensive. Programming back propagation is very hard but it turns out to be very cheap.

So we do all the calculations in terms of doing the theoretical as well as computational work in order to do cheaper competitions. Tensorflow and every single package that you will find, including MATLAB etc have back prop routines.

So what I am showing

(Refer Slide Time: 11:45)

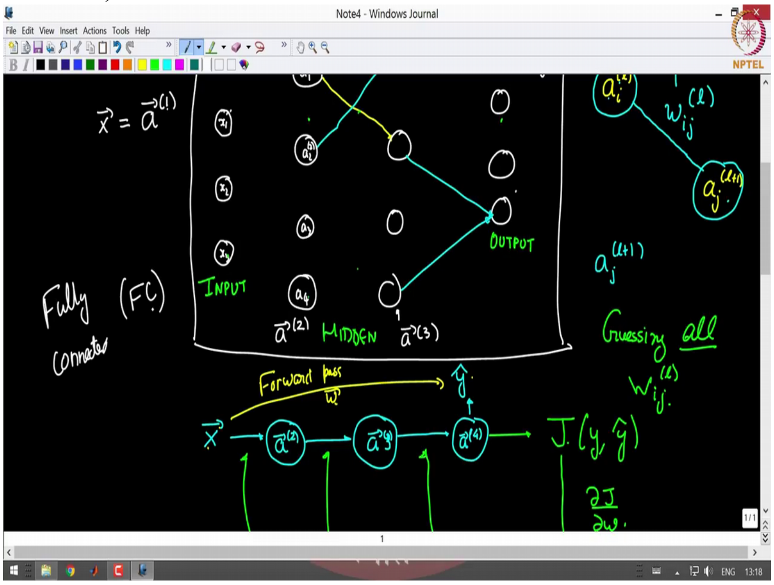


in this video we will just, so that you can get a flavor of what is happening. So that you can get some intuition of what is happening. And we will be using only this portion of this intuition when we come to C N Ns or R N Ns to explain what problems we encounter while training neural networks, Ok.

So please remember, this is just sort of, we will give you some of the final expressions. Of course if you have complicated network architectures, these might or might not work.

But for simply, fully connected neural network of this sort, fully simply means that each neuron is connected to every other neuron in the next layer called F C

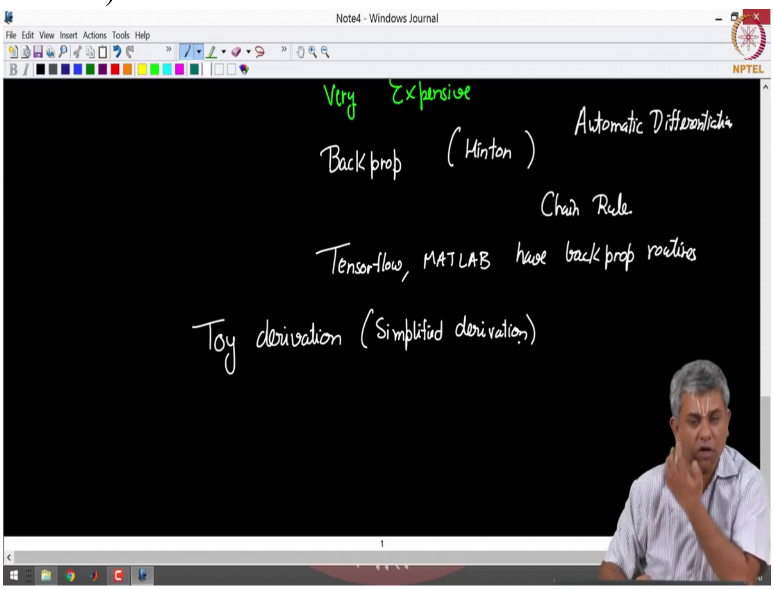
(Refer Slide Time: 12:29)



network, fully connected network, in such cases typically the expressions that I will give later on will be true.

More importantly my derivation you can treat as a toy derivation because I will do a very specific, very, very simple case. This is going to be a very simplified derivation of the rule. This is just to give you a flavor

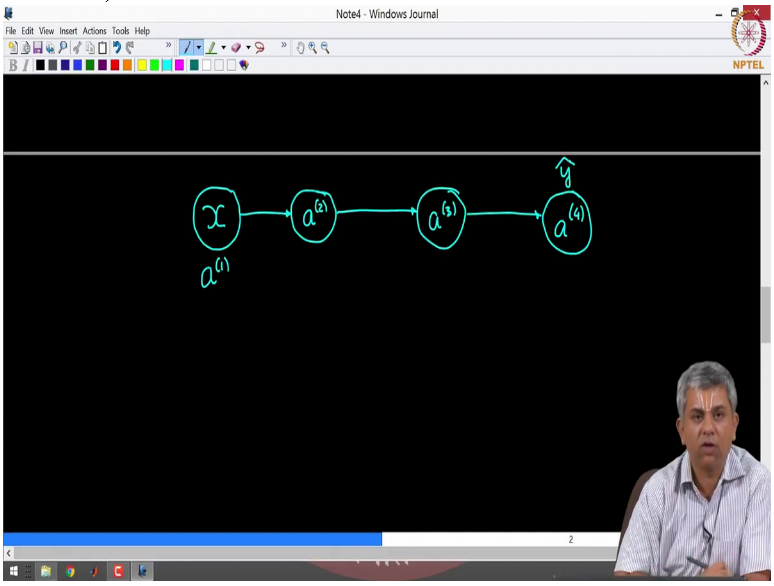
(Refer Slide Time: 12:57)



of how back prop works. And I will come to some technical details towards the end of this week.

So let us start with back prop and we will take the same case as before. I had x vector. We can treat this the same as a 1. This gave me a 2. I had 2 hidden layers. There is a 3. And then there is a 4. a 4 is the same as

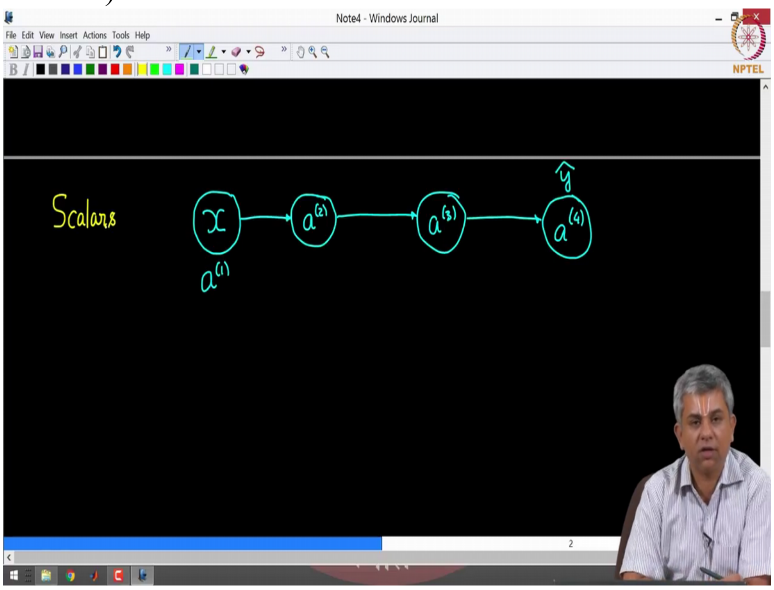
(Refer Slide Time: 13:32)



y hat. So please remember this picture.

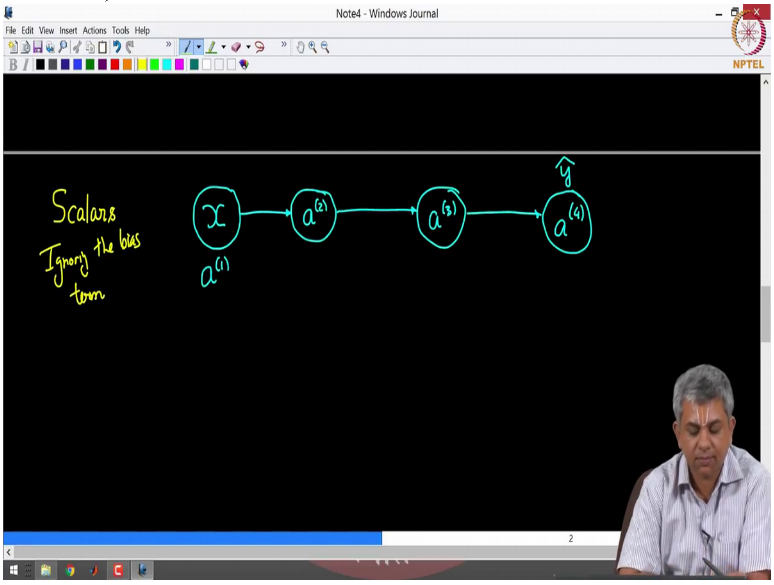
Now the change that I am going to make, so as to make our derivations simple is to treat all these as scalars. I am also

(Refer Slide Time: 13:46)



ignoring the bias term,

(Refer Slide Time: 13:56)

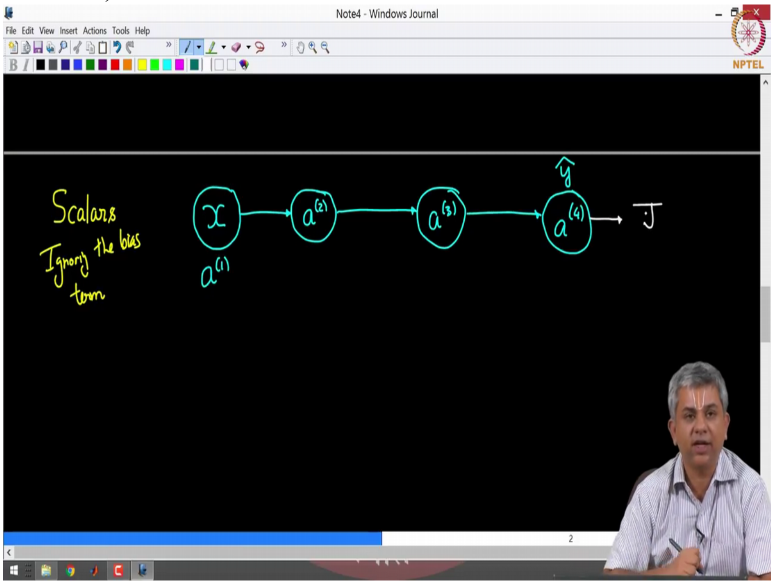


Ok.

So in the general case remember these were vectors. The weights were matrices. All that complication is being thrown away by me just in order to get you a picture.

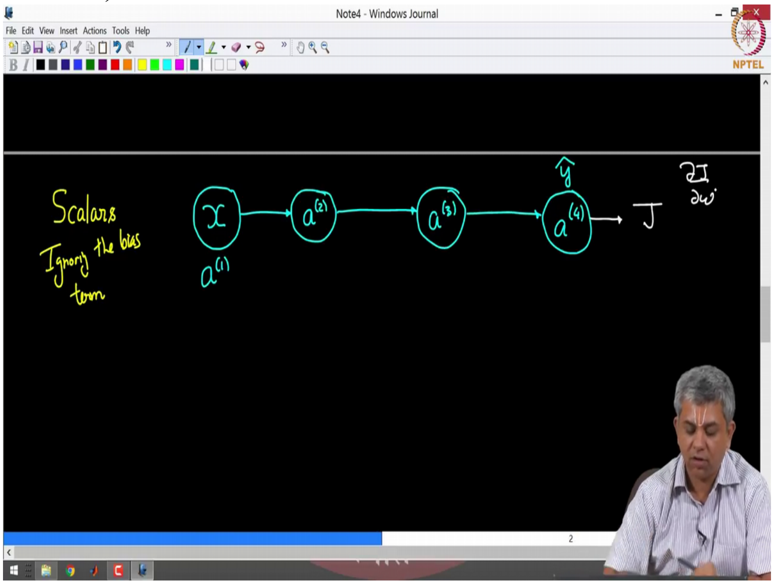
Now surprisingly enough, even with that the expressions we get are very, very close to the final complex expressions, Ok. So after this

(Refer Slide Time: 14:18)



we get J and we want to find out what is del J del w.

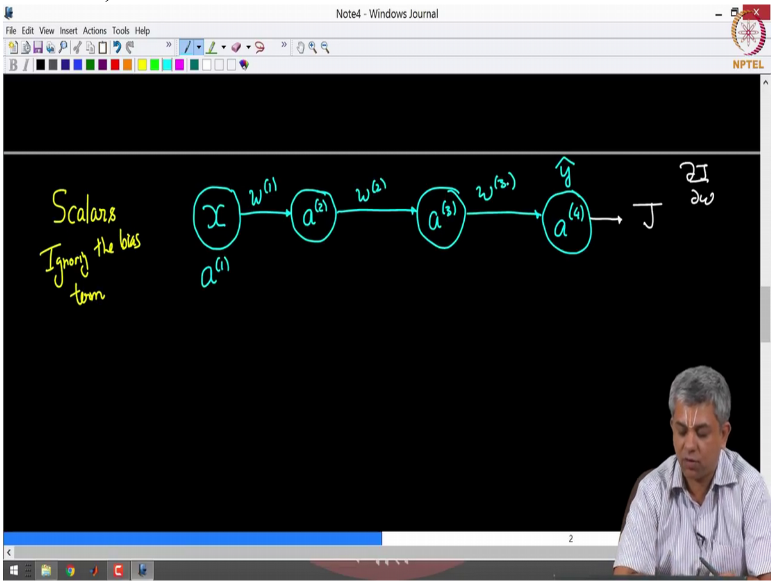
(Refer Slide Time: 14:23)



Now how many weights do we have here?

Notice from a 1 to a 2 you have one weight. Here you have another weight. And here you have another weight,

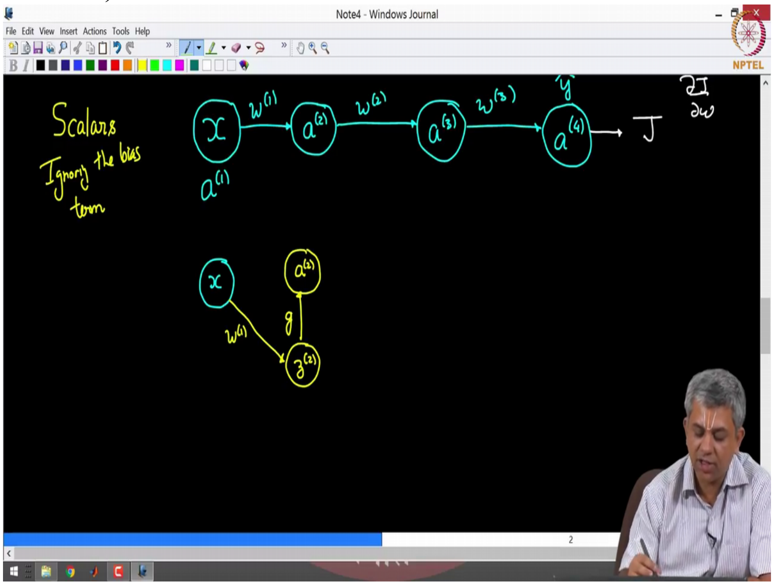
(Refer Slide Time: 14:34)



Ok. So we have 3 sets of weights and this case just 3 weights because these are scalars.

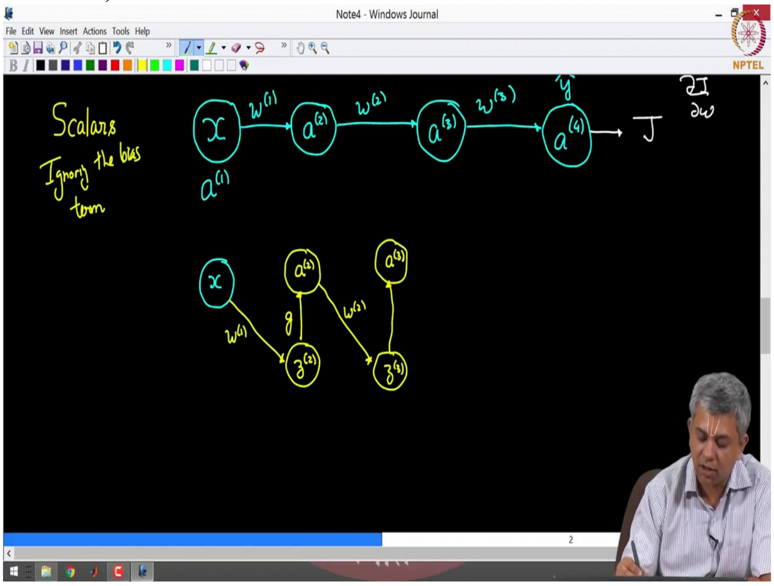
So I am going to do the same thing as I did in logistic regression. I will just draw this figure slightly differently just so that for ease of comprehension, Ok. So first we have the linear operator which gives us z, let us call it z 2. Go back here and we get a 2, Ok. So we have a nonlinearity g.

(Refer Slide Time: 15:08)



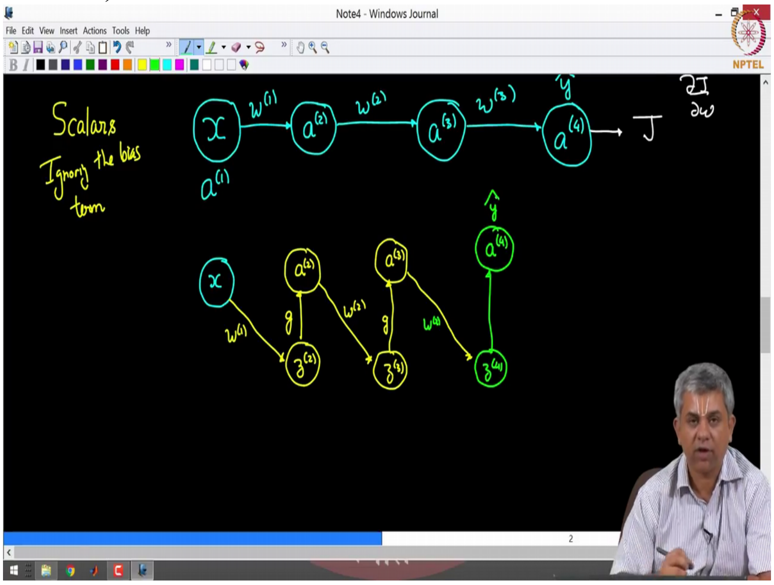
Similarly you take a weight w 2, get z 3, go back here, you get a 3.

(Refer Slide Time: 15:20)



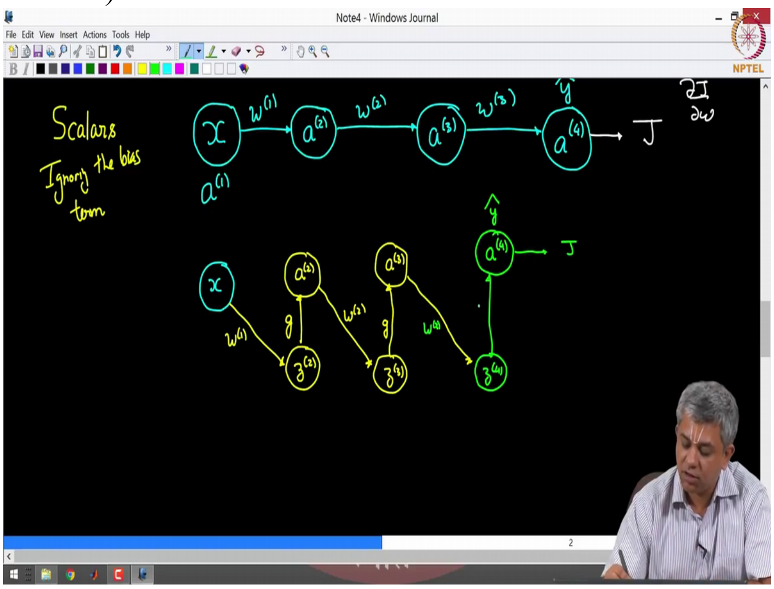
Similarly here, so let us put a g here also. I will change colors. You have the weight w 3, you get z 4. Go back here, get a 4. a 4 is the same as y hat.

(Refer Slide Time: 15:40)



And I get here the J here.

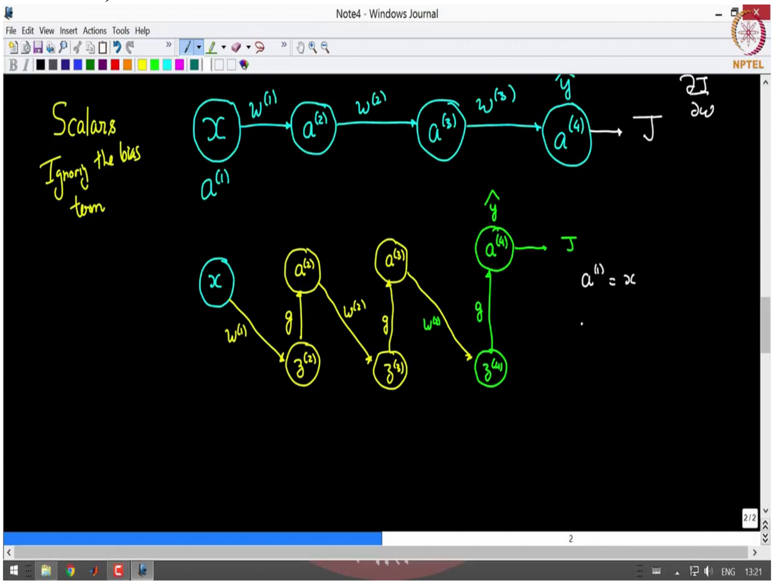
(Refer Slide Time: 15:44)



This is the nonlinearity g, Ok.

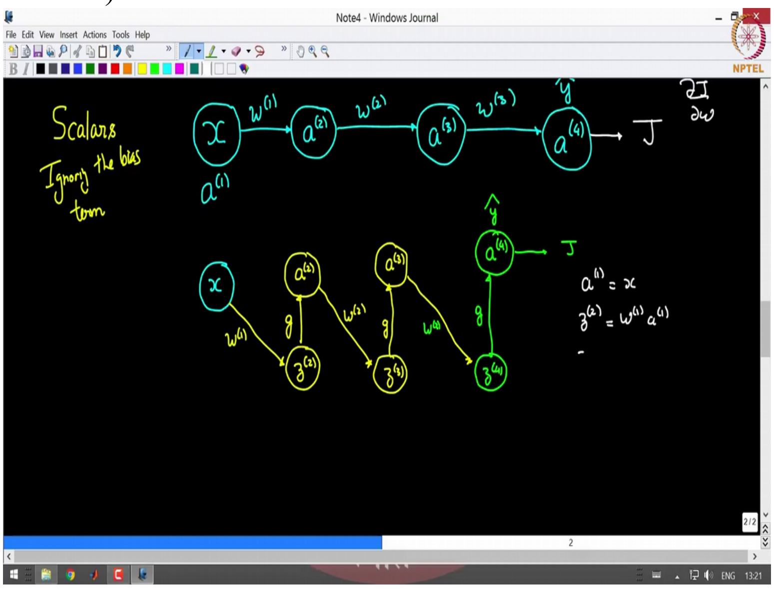
So let us write some expressions down just so that we can use them for clarity. a 1 is the same as x. Then

(Refer Slide Time: 15:59)



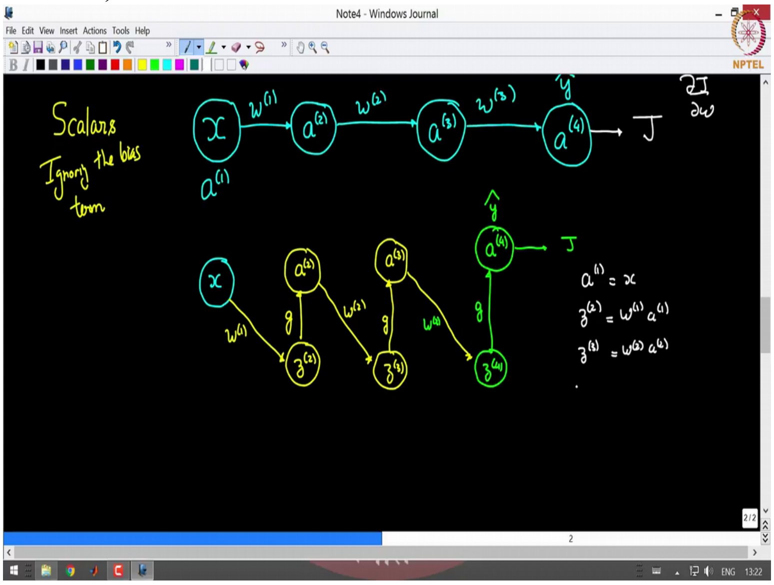
if you notice here z 2 is equal to w 1 a 1.

(Refer Slide Time: 16:07)



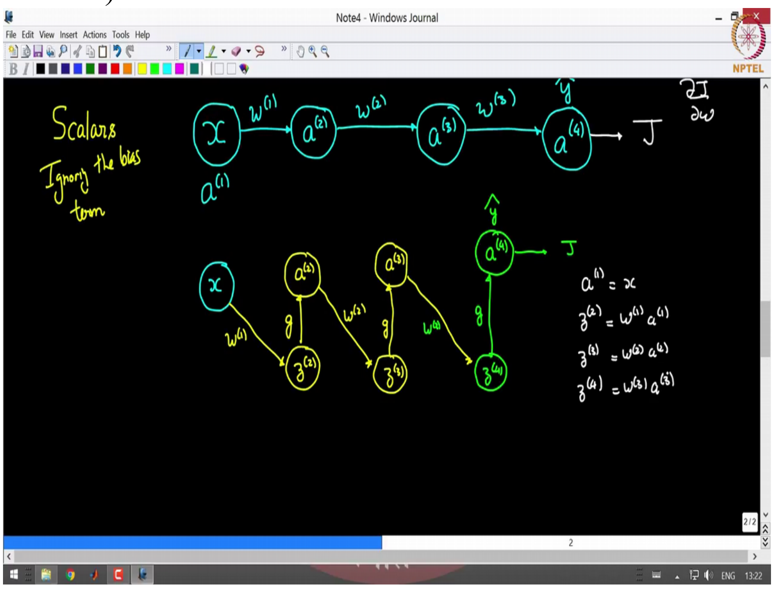
z 3 is equal to w 2 a 2.

(Refer Slide Time: 16:13)



z 4 is equal to w 3 times a 3. This is my simplification of the

(Refer Slide Time: 16:23)

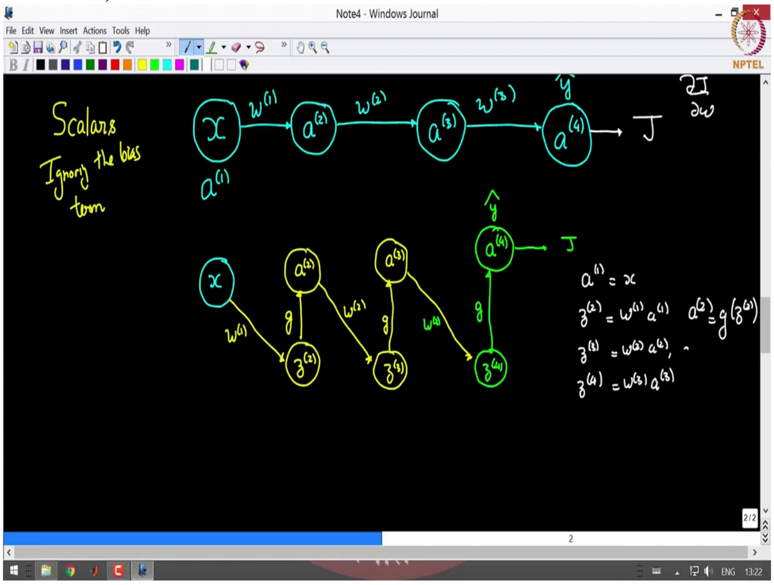


linear summation process that we have.

If we were dealing with the full vector case, all that would change here is this would become w transpose times a vector, w 2 transpose times a 2 vector etc. Now apart from this, we have the nonlinearities. We have a 2 is the nonlinearity applied on z 2.

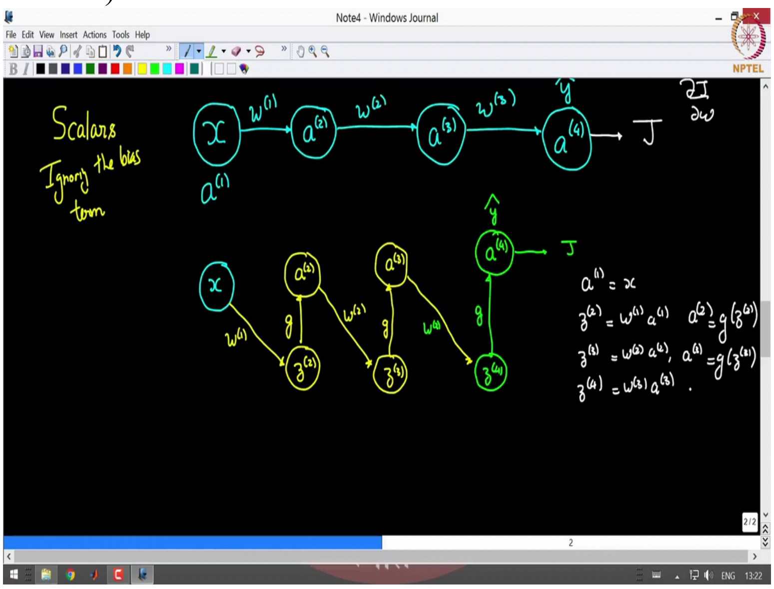
Similarly

(Refer Slide Time: 16:50)



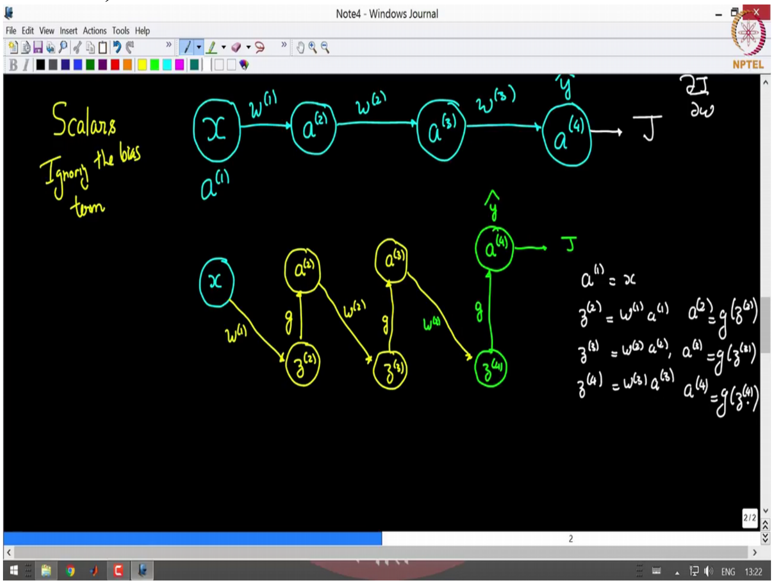
a 3 is g of z 3,

(Refer Slide Time: 16:55)



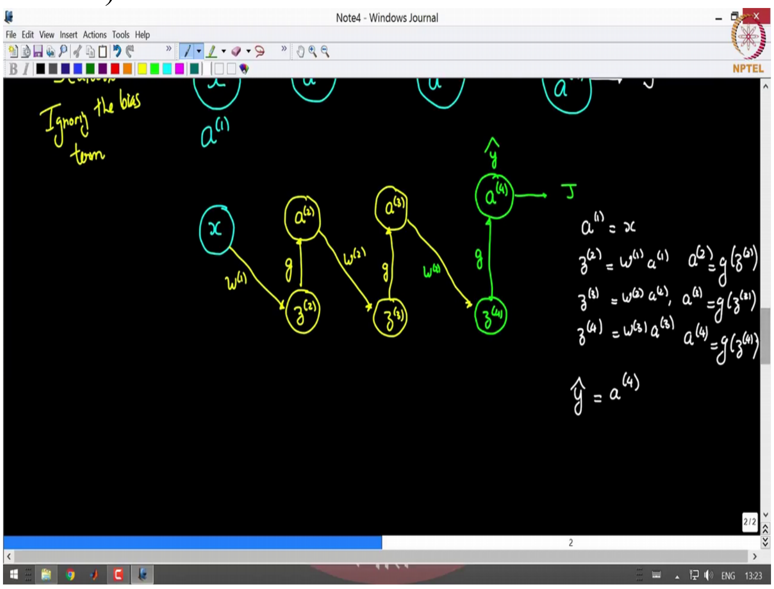
a 4 is g of z 4.

(Refer Slide Time: 17:01)



Finally y hat is simply a 4, Ok.

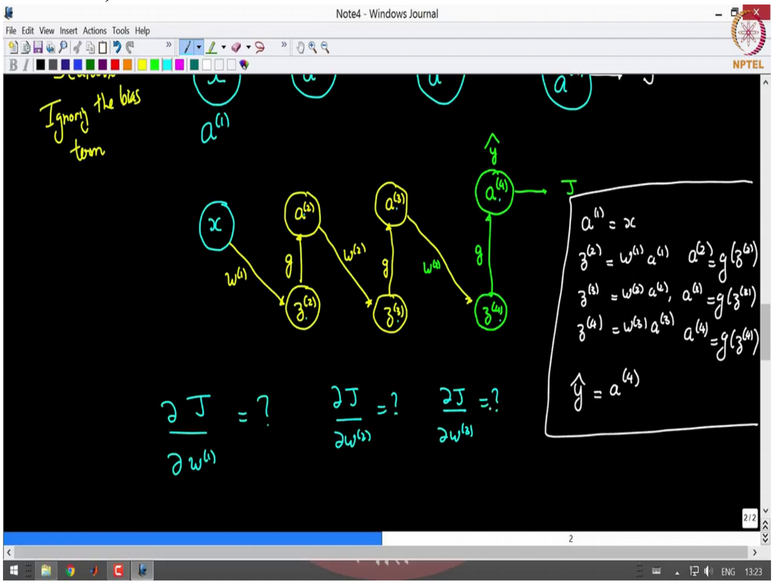
(Refer Slide Time: 17:10)



So these are our relationships. Finally what we want to find out is J. How much does J change due to a change in w 1, Ok? You will notice, what will happen? The moment w 1 is changed z 2 is changed, a 2 is changed, z 3 is changed, a 3 is changed, z 4 is changed, a 4 is, so it is a cascading set of problems.

So if you have del J del w 1, then what is this? If I have del J del w 2 what is this? Similarly del J del w 3 what is this?

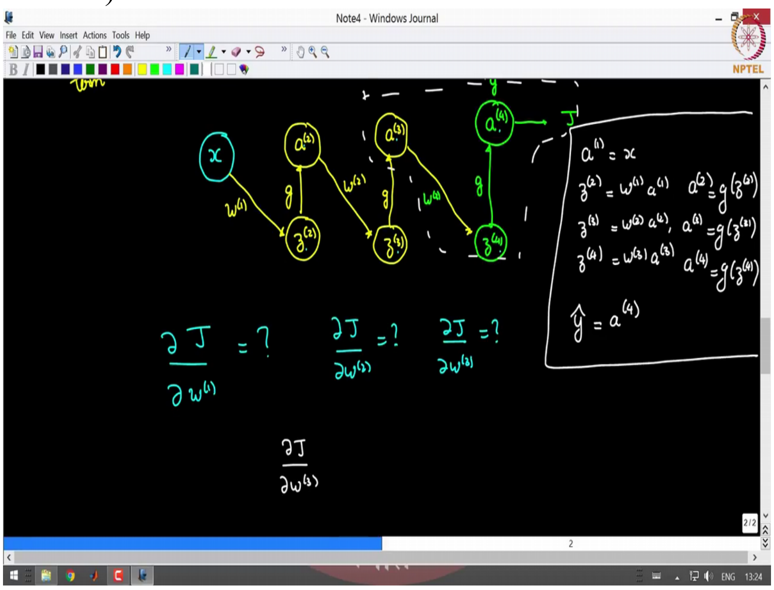
(Refer Slide Time: 17:51)



So these are the questions that we need to answer. Now notice it is actually easiest to find out this term. Why is that? Because this is closest for being responsible for J.

So let us find this term first, del J del w 3. If you had been very careful, you will actually notice that this is very similar, in fact practically identical to what we had

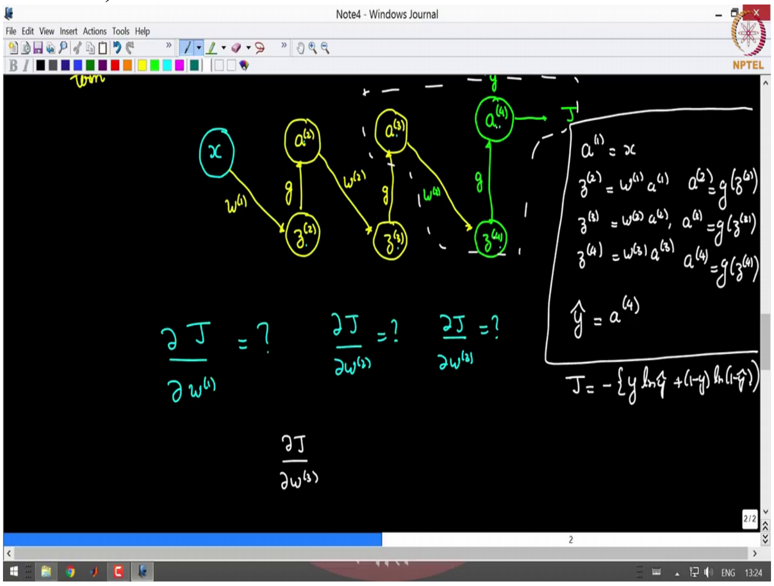
(Refer Slide Time: 18:19)



in logistic regression. You had a input, summation, nonlinearity you immediately got the output and that is what we got in logistic regression.

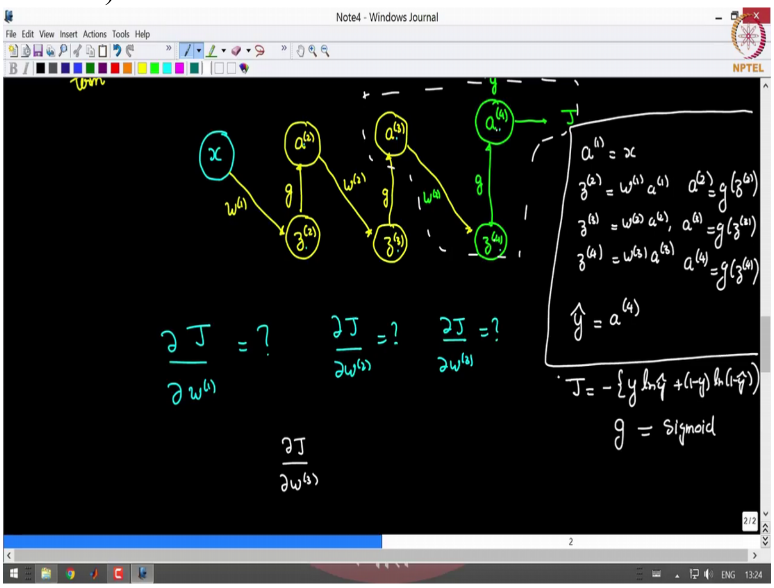
So for now, for the sake of this example I assume that J is the binary entropy cross function, y l n y hat plus, because we had already done some calculations for this.

(Refer Slide Time: 18:43)



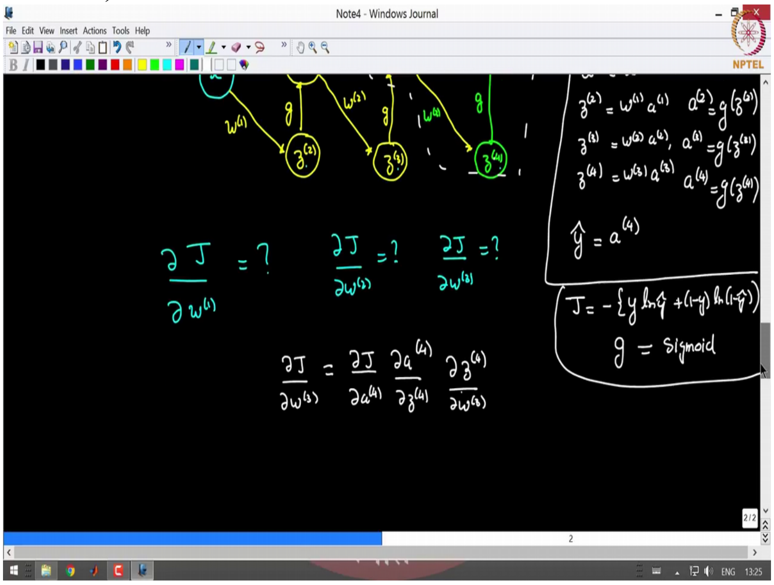
So let us assume that this is the binary entropy cost function. And that g is the sigmoid.

(Refer Slide Time: 18:52)



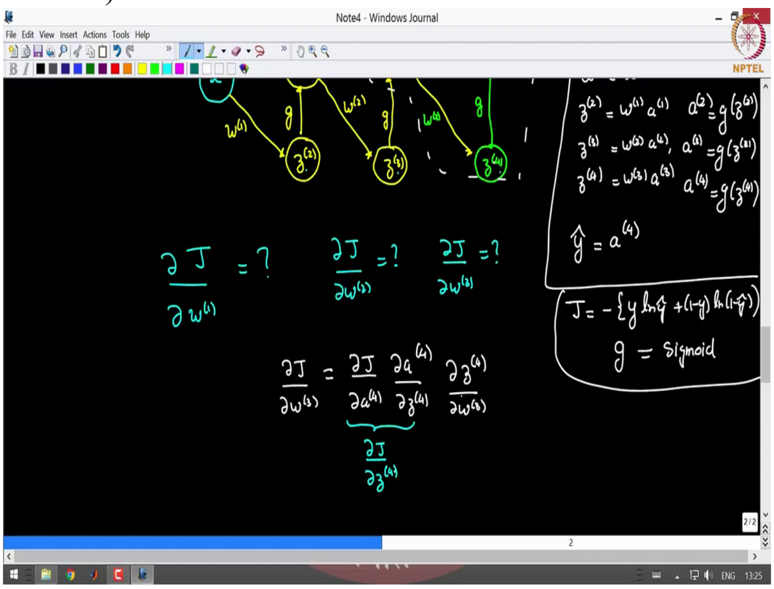
We will assume this but you can do this process for any g and any J, Ok as you will see, you will see shortly. So let us say I want del J del w 3. What is it equal to? del J del a 4, this step times del a 4 del z 4, that is this step multiplied by del z 4 del w 3, Ok.

(Refer Slide Time: 19:26)



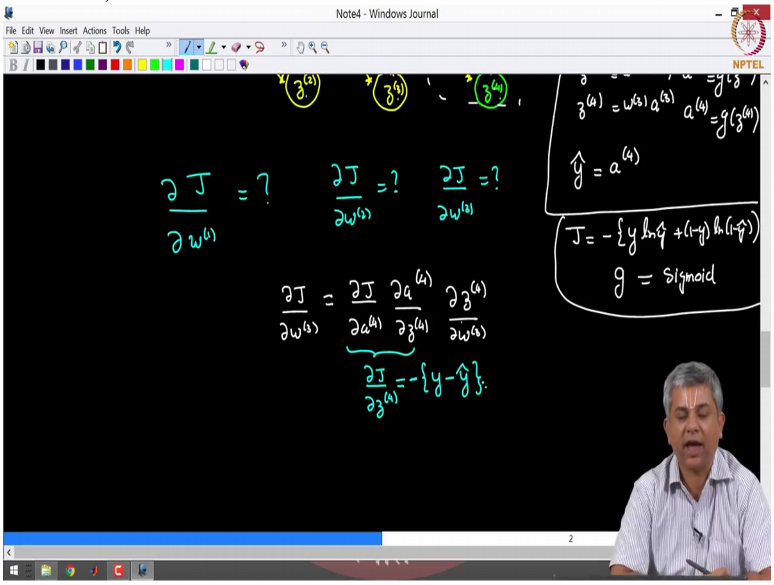
Now if this is the binary entropy cross function we had actually done this calculation. This is the same as del J del z 4

(Refer Slide Time: 19:38)



and we have done this in the previous video. This is already equal to minus y minus y hat.

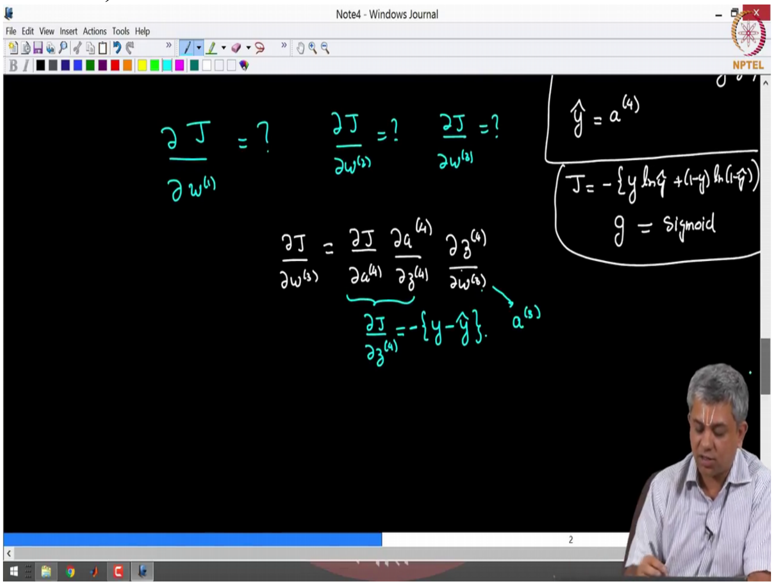
(Refer Slide Time: 19:50)



So we have already calculated this before. Please look up that video to convince yourself that this is exactly the same.

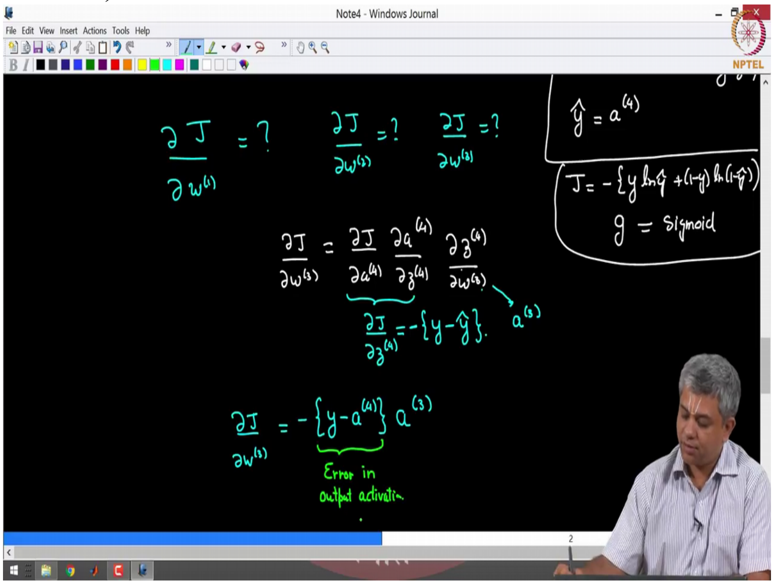
What is del z 4 del w 3? You can automatically see this. This is a 3. So let me write this down.

(Refer Slide Time: 20:07)



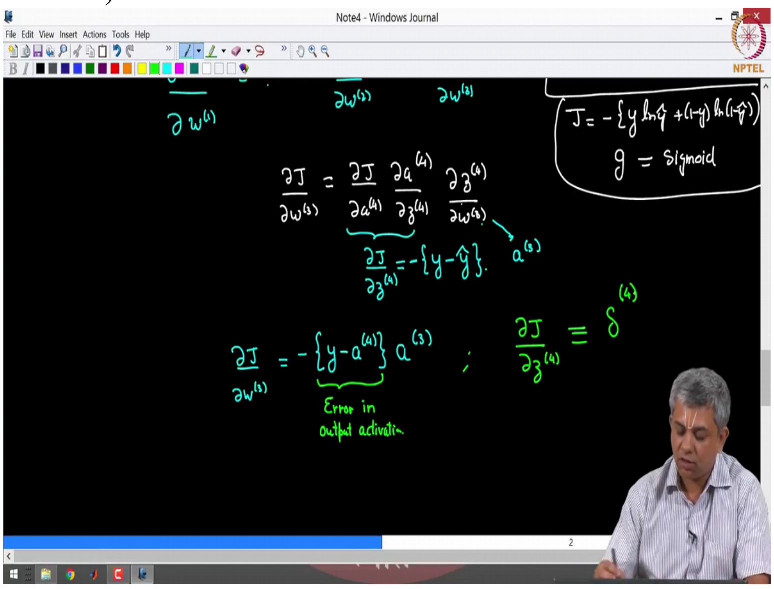
del J del w 3 is equal to minus y, minus instead of y hat, I will write it as a 4 times a 3. Now this term as we have seen before, is the error in output activation.

(Refer Slide Time: 20:39)



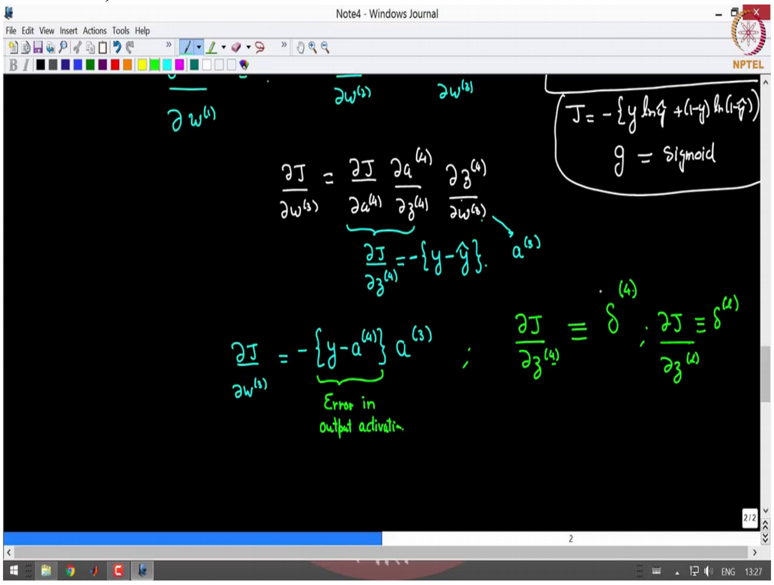
Now we use a particular notation, Ok, we use the notation that del J del z 4 is defined as a quantity called delta 4.

(Refer Slide Time: 20:56)



Notice this 4 and that 4 are not the same. Similarly we will say del J del w or del z l is defined as delta l. What does it denote?

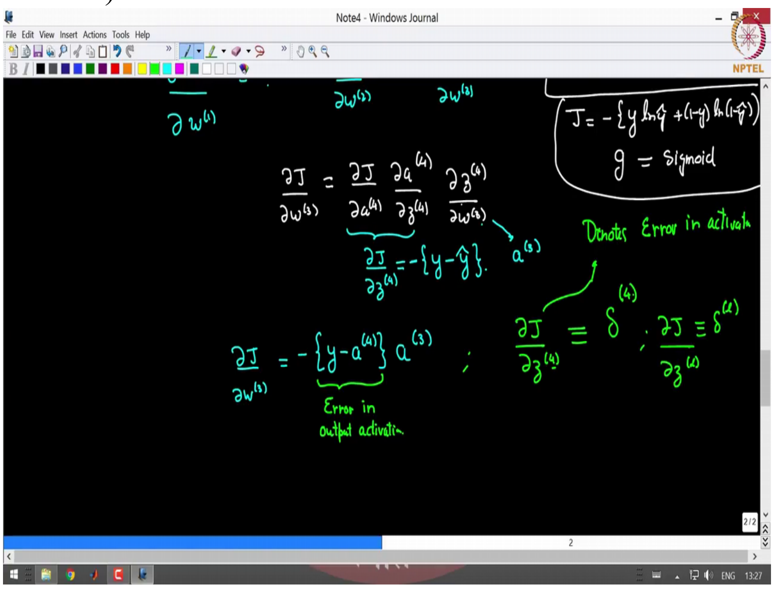
(Refer Slide Time: 21:08)



It kind of denotes, Ok this is not exact but it denotes this term, error in activation.

So I want to warn you before several questions

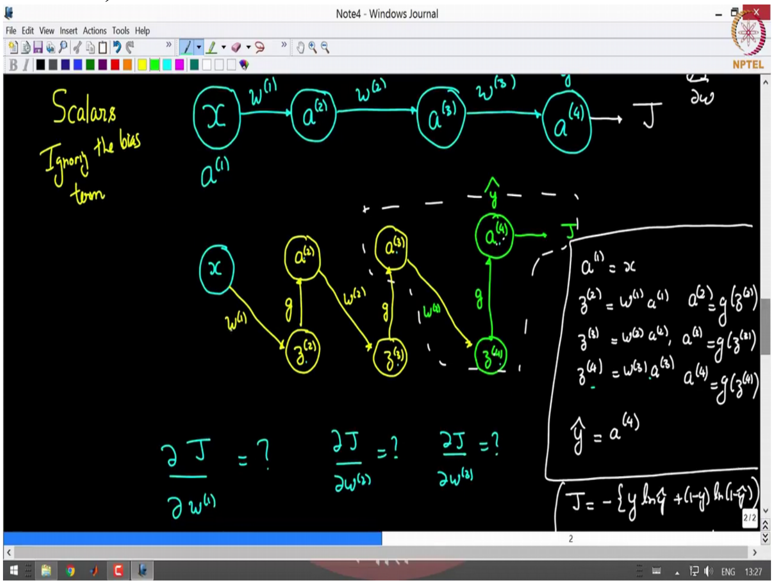
(Refer Slide Time: 21:24)



come that this is simply heuristic or just for, in order to you, in order for you to build an intuition about this thing. So what does this mean?

As you

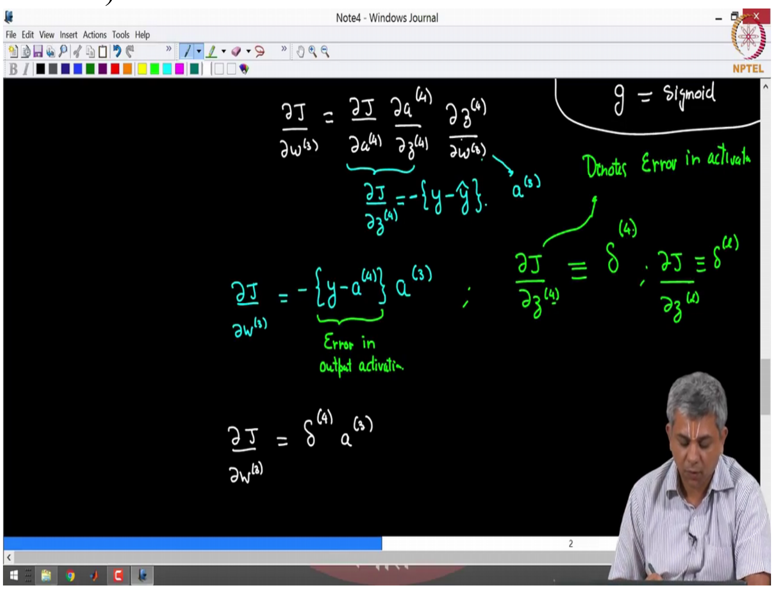
(Refer Slide Time: 21:33)



perturb this, instead of what is supposed to be the actual a, you know in the final case when it is very well fit, you are going to have something slightly different, Ok, just like this term, Ok instead of a, you would have a plus something, Ok or a minus something.

And that is what this term delta 4 denotes. So we will keep that and we will write, but I have not made any approximation here. All I have said is del J del w 3 is equal to delta 4 times a 3, Ok.

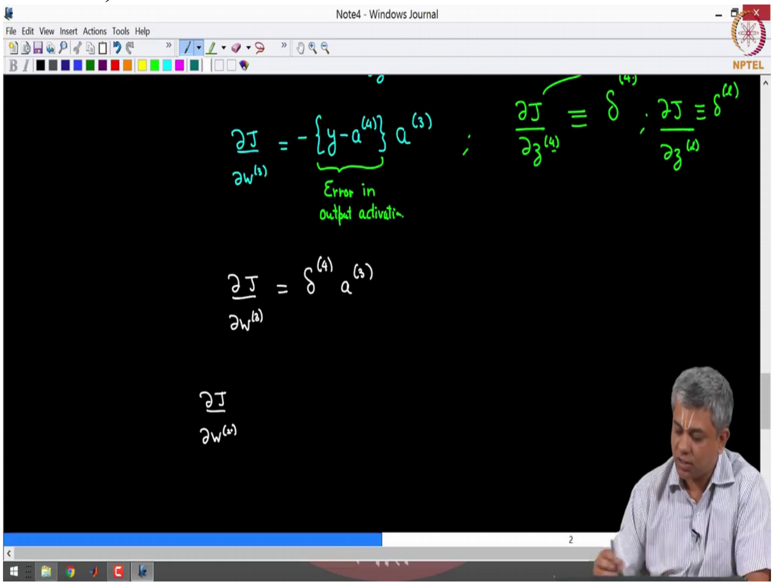
(Refer Slide Time: 22:10)



So please notice this, Ok

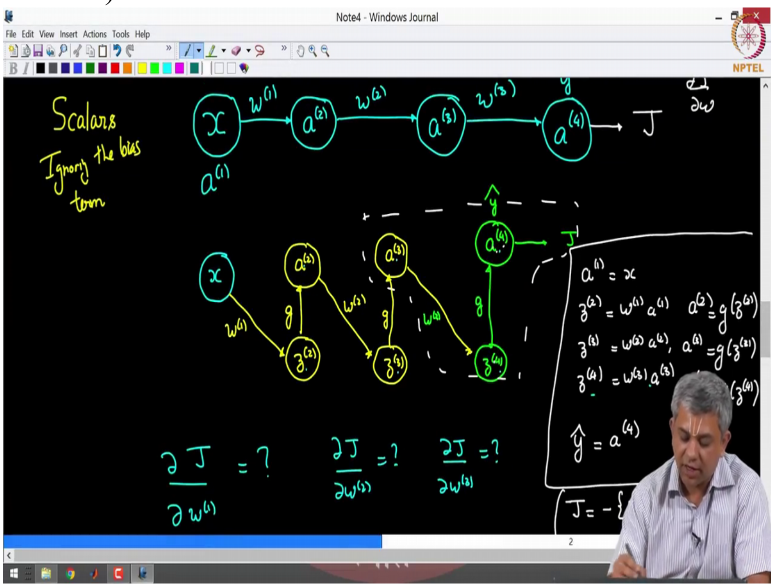
Now suppose I am going to do del J del w 2.

(Refer Slide Time: 22:21)



Let us go back to the figure del J del w 2.

(Refer Slide Time: 22:25)

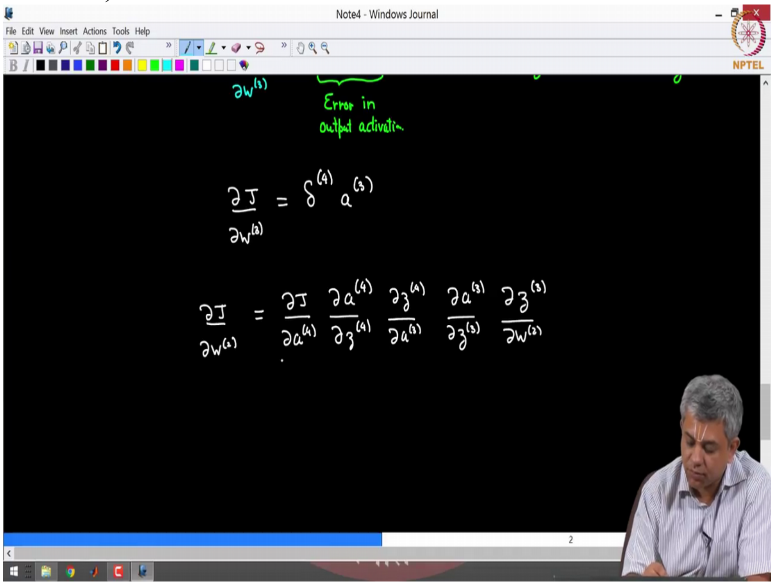


What will it be? You will do this, this, this, this, and finally this. So you are going to have 5 terms. So please track it with me. This is going to be del J del a 4 del z 4, del a 4, del z 4 del a 3. Please notice this, del z 4 del a 3.

Then del a 3, let us go back to the figure, del a 3 del z 3. And finally del z 3 del w 2 as you can see in the figure here, del z 3 del w 2.

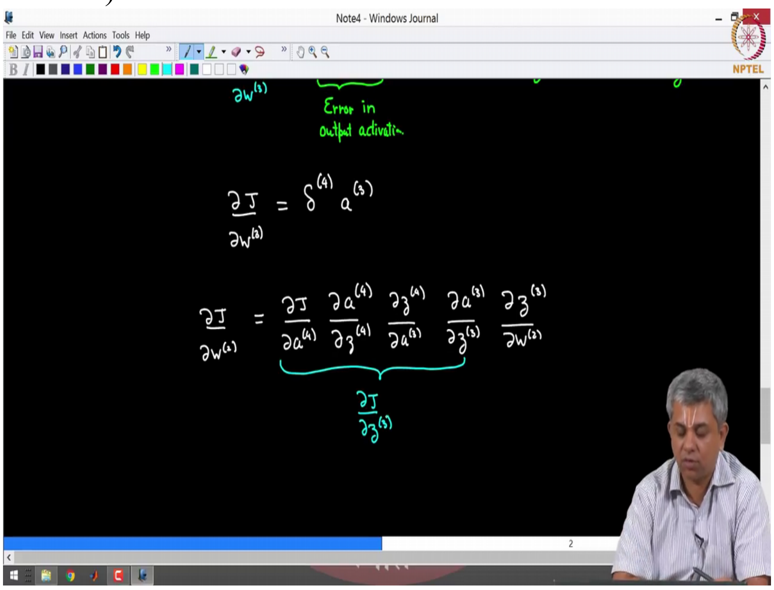
So this term, this term, this term, this term and finally this term, Ok. So this looks very tedious until we notice a certain pattern.

(Refer Slide Time: 23:38)



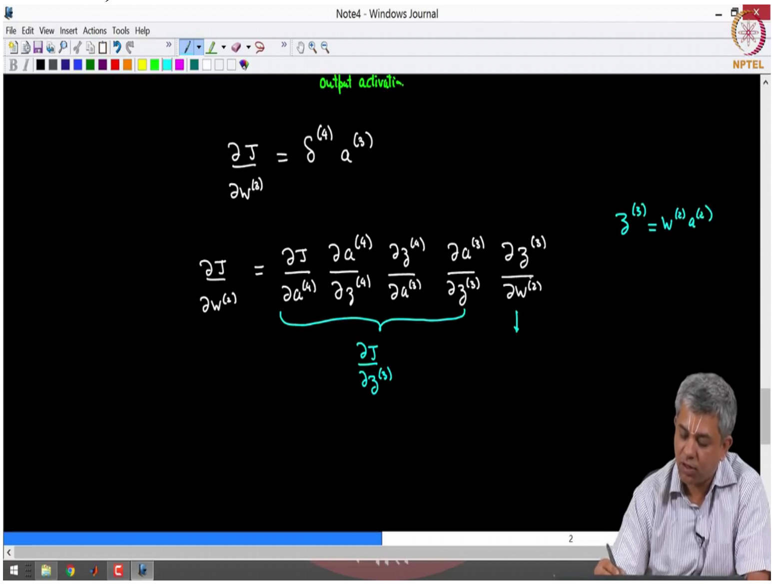
You will notice that all this chain is simply del g del J del z 3.

(Refer Slide Time: 23:49)



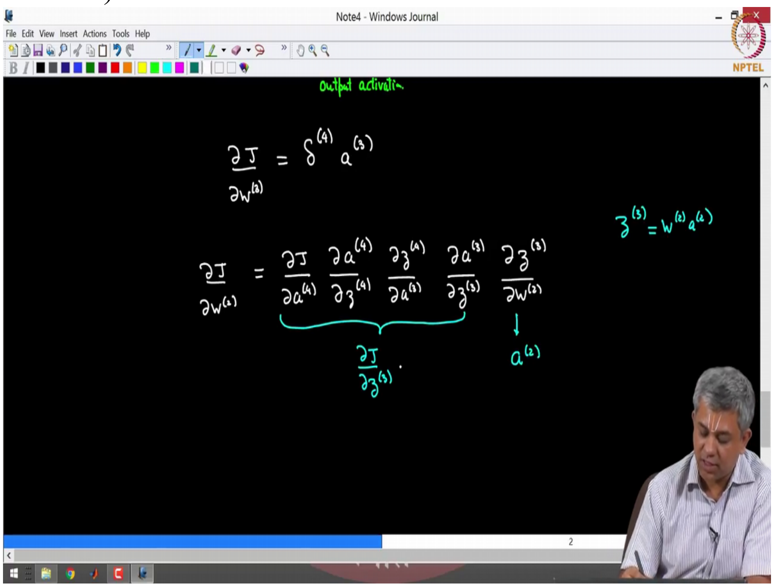
This term is simple because z 3 is equal to, let us go back to our relationships, z 3 is equal to w 2 a 2. Therefore del z 3 del w 2 is simply a 2, Ok

(Refer Slide Time: 24:05)



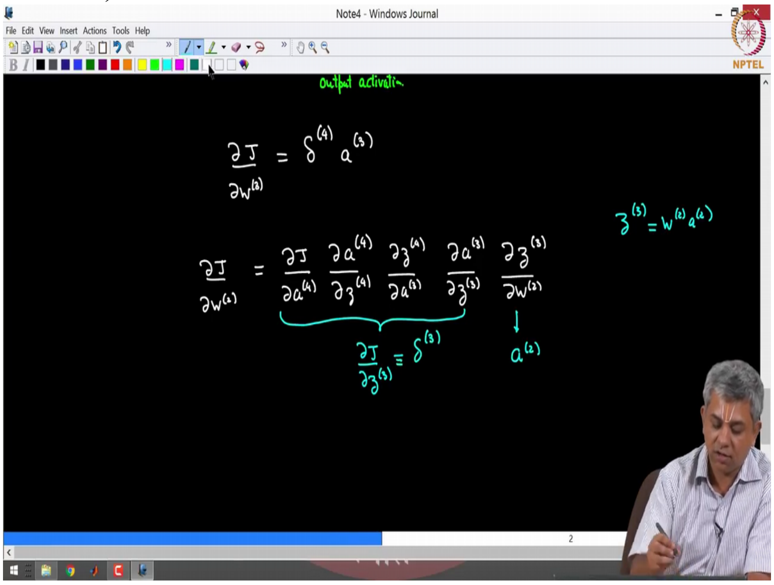
Therefore del z 3 del w 2 is simply a 2, Ok

(Refer Slide Time: 24:11)



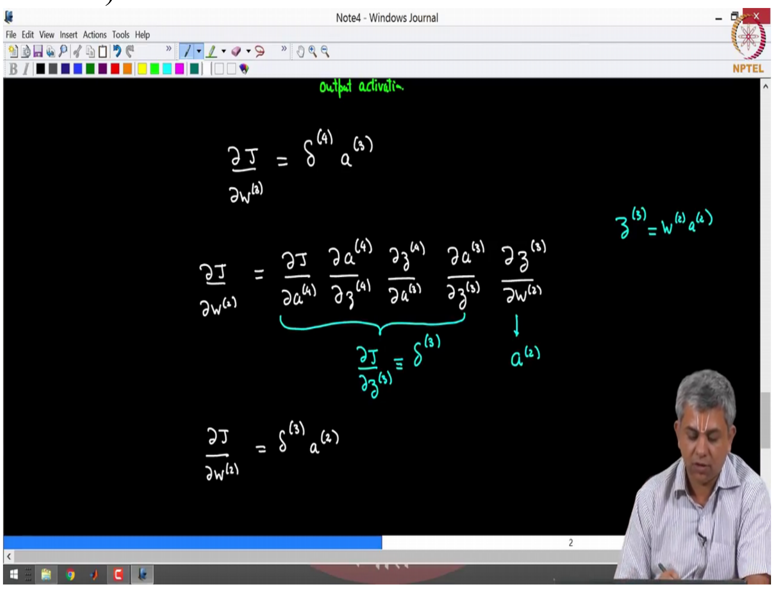
So if you notice this, del J del z 3 by our notation we had called this delta 3.

(Refer Slide Time: 24:18)



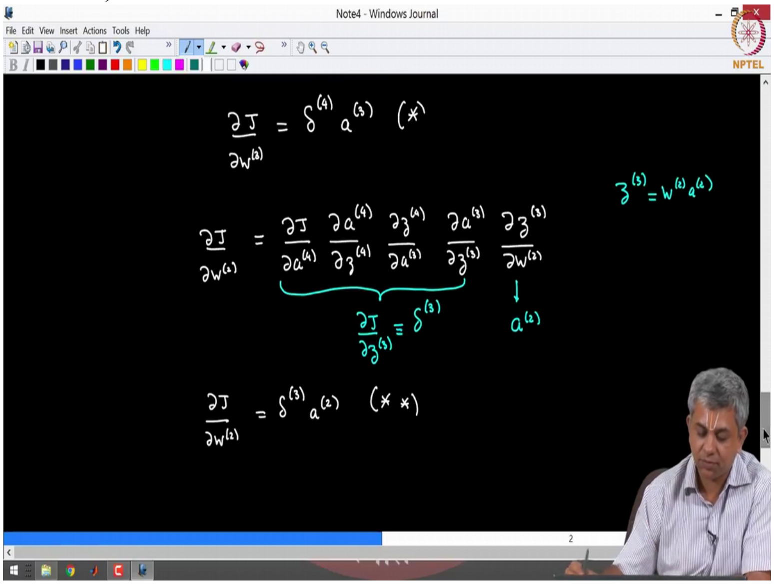
So you will get del J del w 2 equal to delta 3 a 2.

(Refer Slide Time: 24:28)



Notice these two formulae and you will automatically see a pattern.

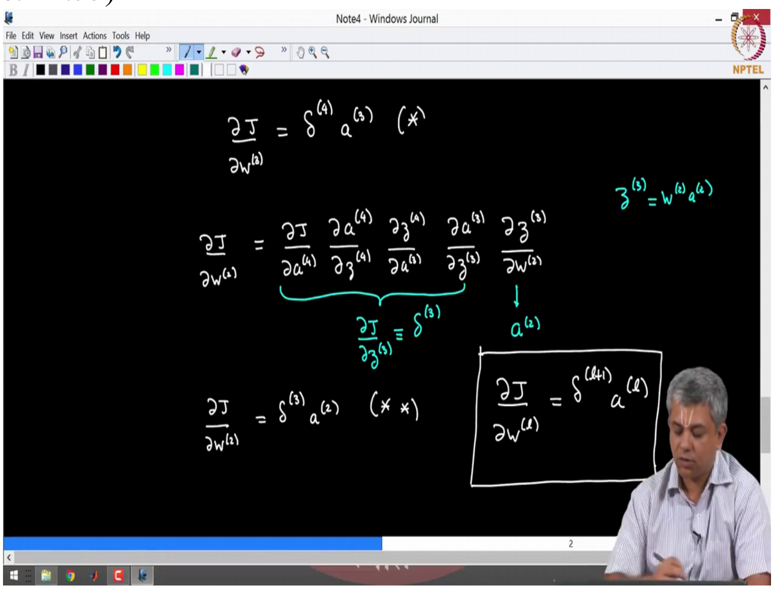
(Refer Slide Time: 24:35)



You notice that del J del w l is equal to delta l plus 1 a n.

This is our

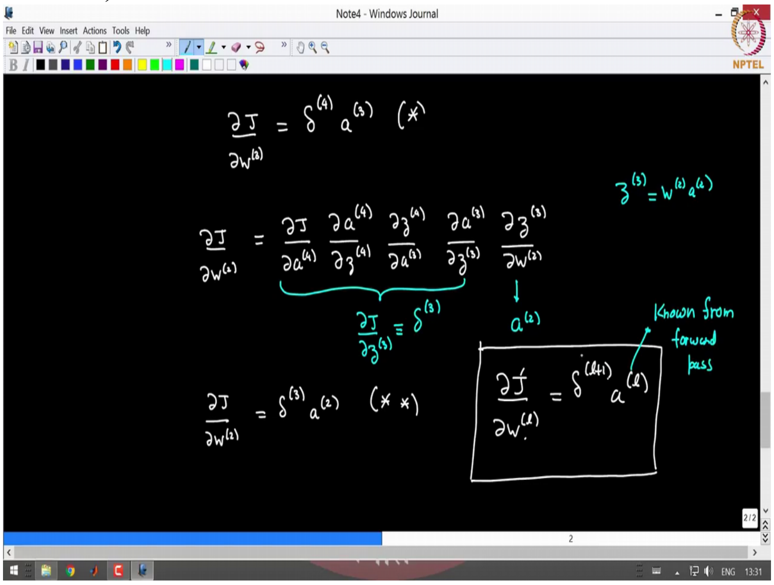
(Refer Slide Time: 24:53)



first relationship, that del J del w l is delta l plus 1 a l. Ok so in this term this is what we want to find out finally. Do we know everything? We know this. How do we know a l? Simply from the forward pass. So if I made a guess for w, I would already have a l before finding out y hat, Ok.

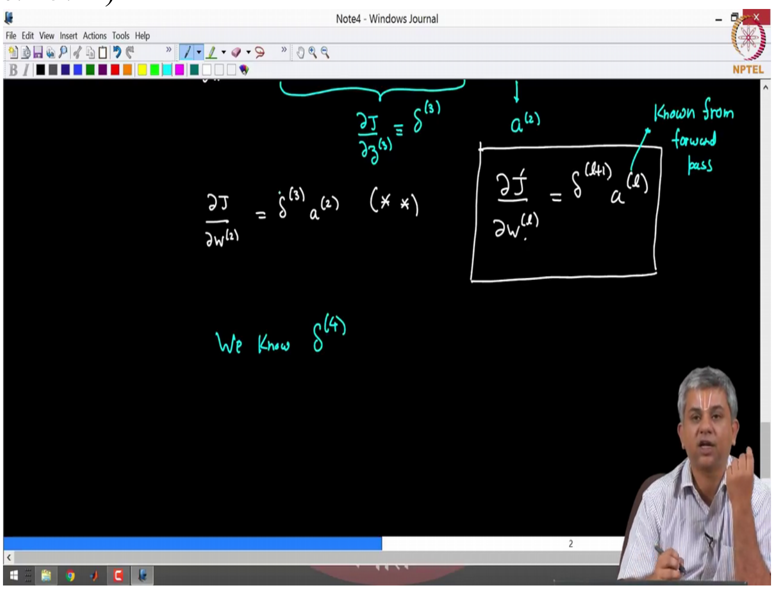
So a l is known from the forward pass. However delta l plus 1

(Refer Slide Time: 25:28)



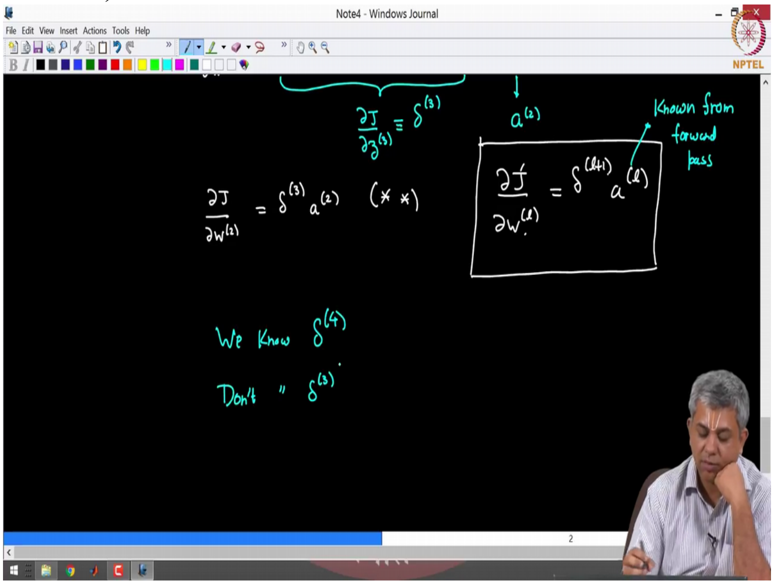
is not known. It is known only in one specific case. Which case? This one. Because it was the output error, Ok. So we know delta 4. But suppose if I asked you

(Refer Slide Time: 25:44)



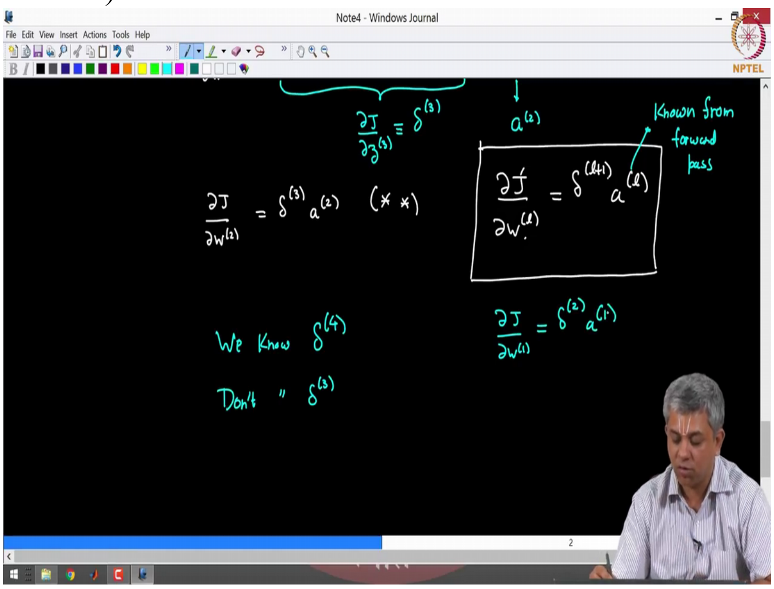
what is delta 3, we do not know, Ok.

(Refer Slide Time: 25:51)



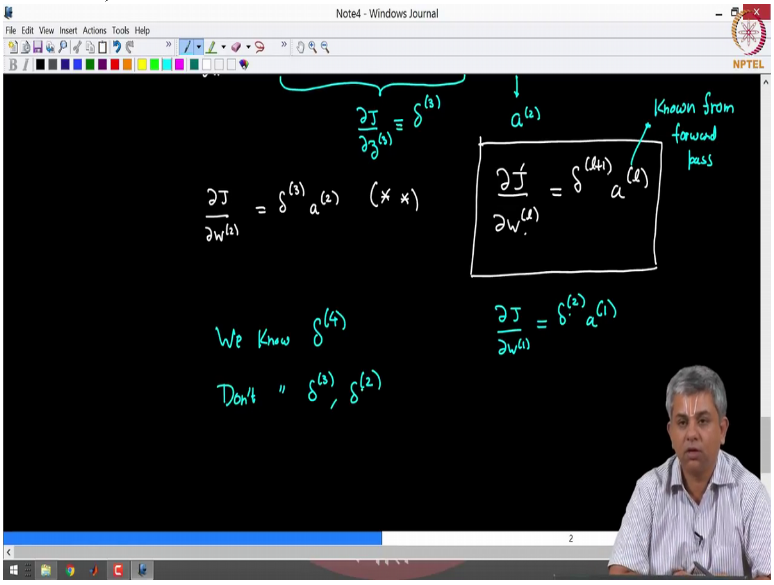
Similarly if I wrote del J del w 1 this would be delta 2 times a 1. a 1 is known

(Refer Slide Time: 26:00)



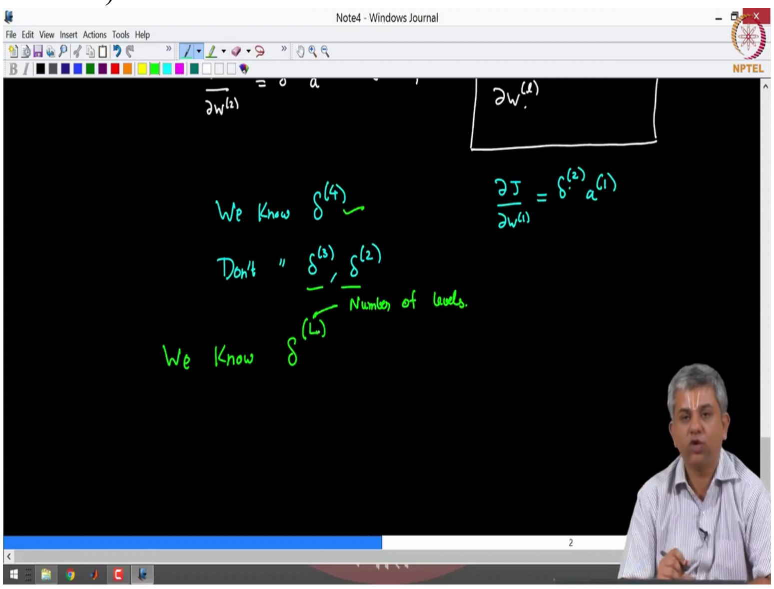
but delta 2 is not known.

(Refer Slide Time: 26:04)



So can we find out these two terms? We know the final one and you will always know the final error. So I will call it L where L is the number of levels.

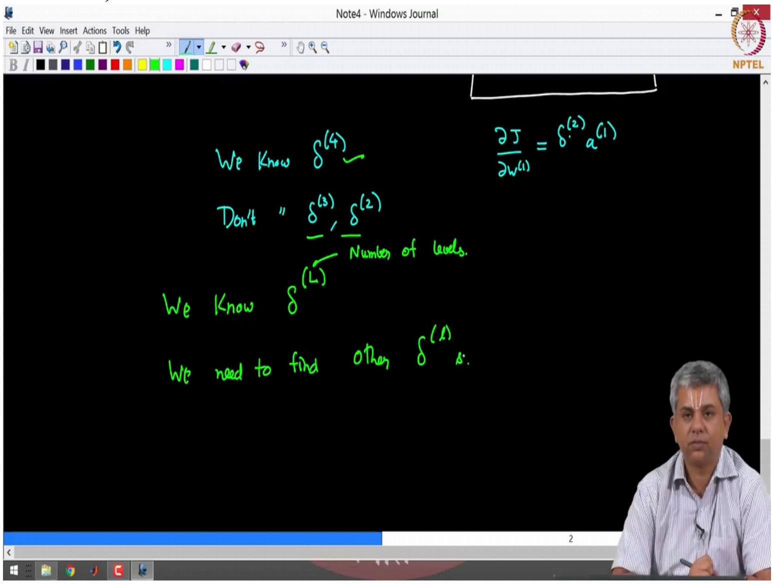
(Refer Slide Time: 26:25)



So you will always know the delta at the final layer one way or the other, you can always find out this, as a combination of analytical and computational procedures using the same idea that we did here.

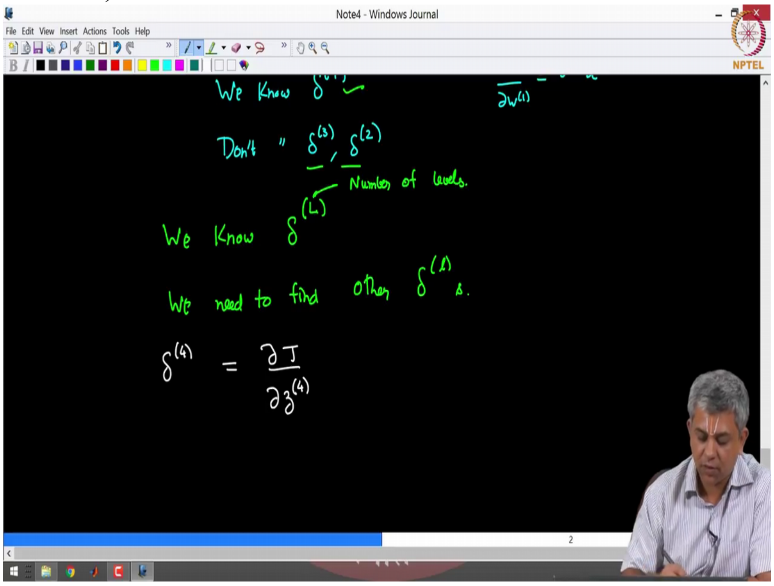
You just differentiated, use whatever nonlinearity you have there. So that can be found out. So we need to find other deltas,

(Refer Slide Time: 26:55)



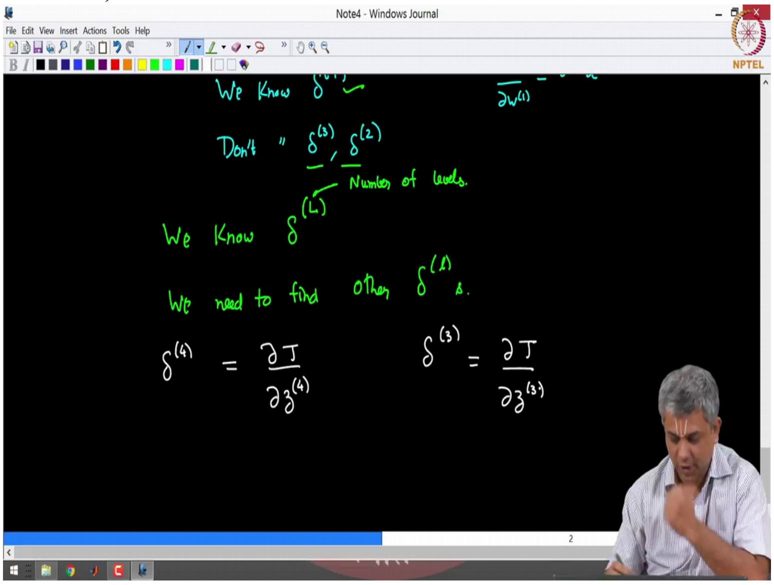
Ok. So let us now write the relationships for these two. So notice this. Remember delta 4 was defined as del J, let us go back up here, so del J, del J del z 4.

(Refer Slide Time: 27:24)



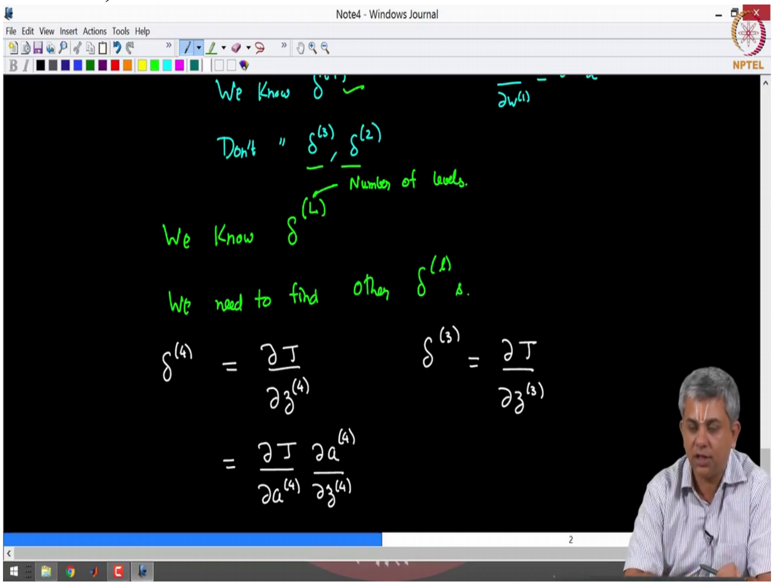
And delta 3 is del J del z 3, Ok.

(Refer Slide Time: 27:33)



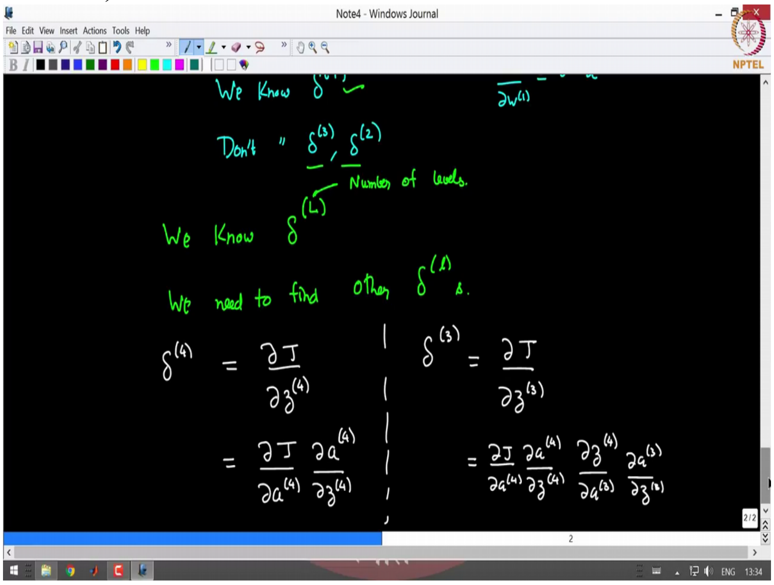
So let us write the expressions for this. del J del z 4 was del J del a 4 multiplied by del a 4 del z 4.

(Refer Slide Time: 27:48)



What about delta 3? This was del J del a 4 del a 4 del z 4. Just to refresh your memory let us go back to the figure. del J del a 4, del a 4 del z 4, now we want till z 3. So del z 4 del a 3, del a 3 del z 3, del a 3 del z 3, Ok.

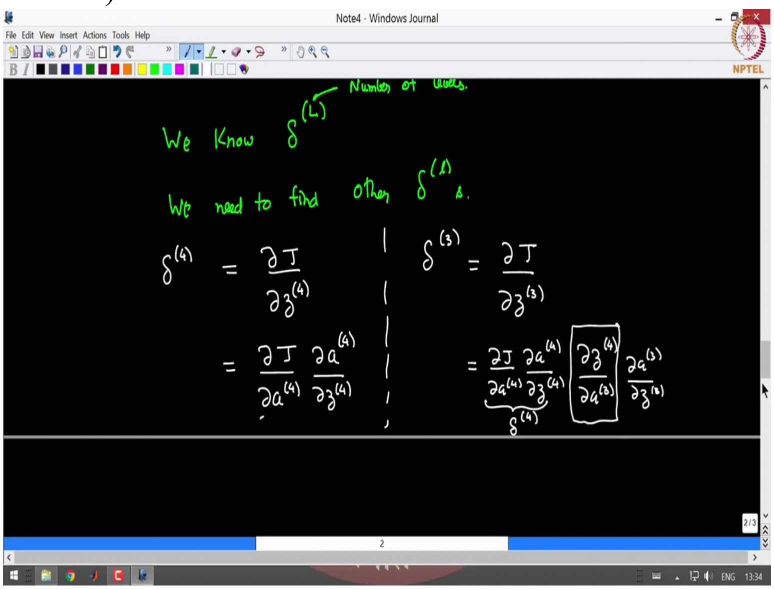
(Refer Slide Time: 28:28)



If you compare these two expressions, this and this, you will notice that this portion is already delta 4, Ok. So that portion is straightforward.

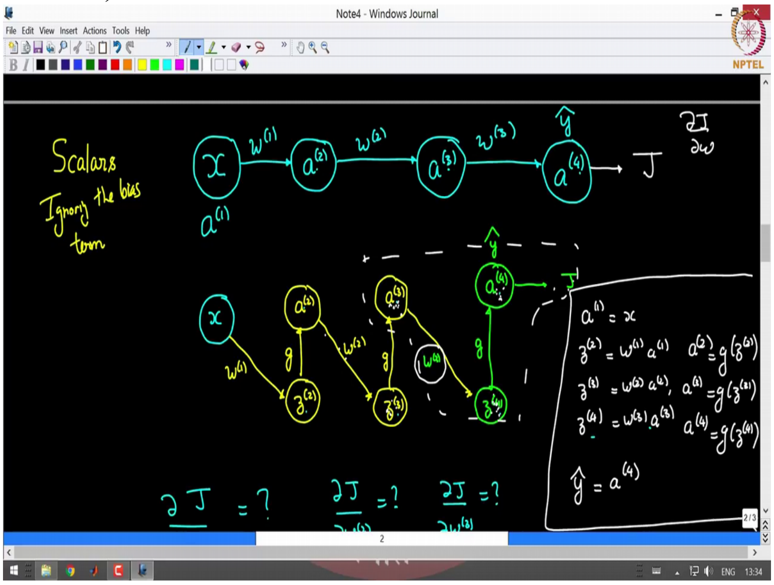
What about this? Let us look

(Refer Slide Time: 28:47)



at this term, del z 4 del a 3. Come back to the figure, del z 4 del a 3 is simply this term w 3. You can

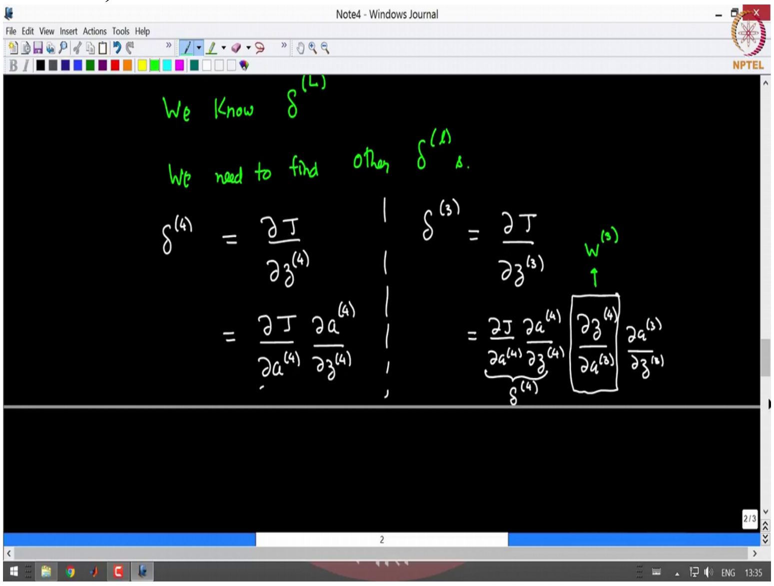
(Refer Slide Time: 28:59)



see that. del z 4 del a 3 is w 3. So let us write that here. This term is w 3.

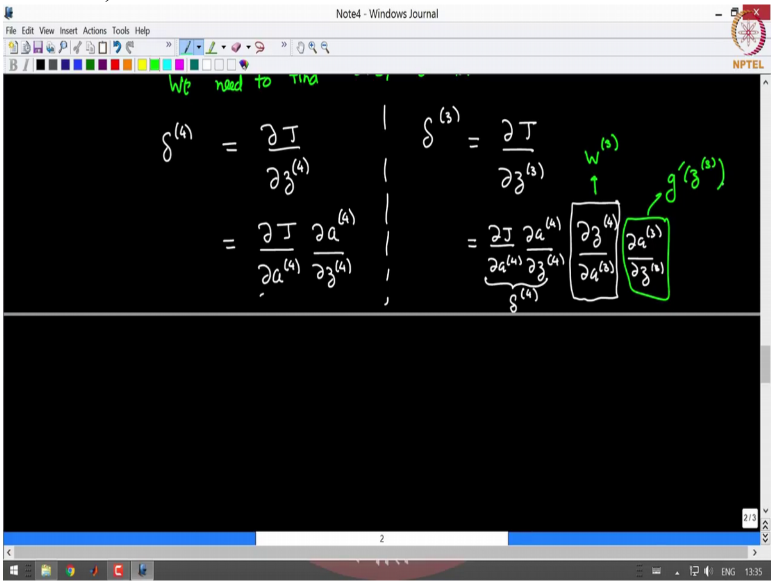
What about del a 3 del z 3?

(Refer Slide Time: 29:13)



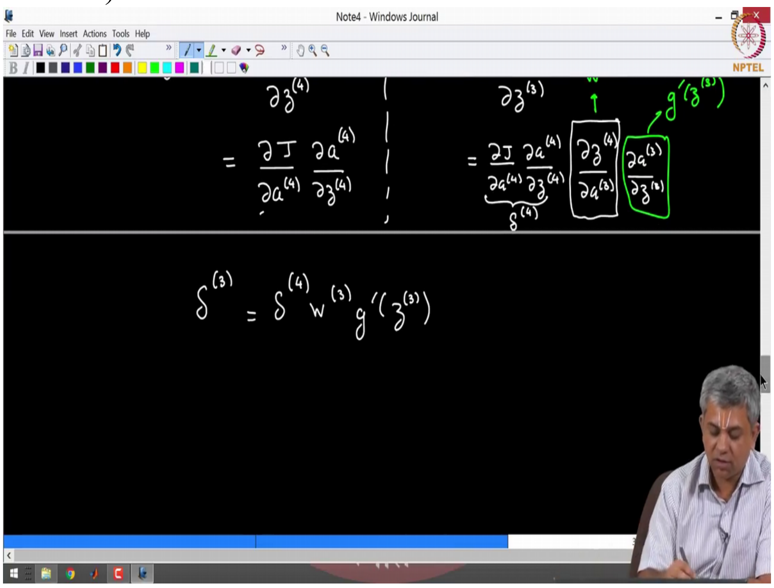
del a 3 del z 3 is related by g, so notice this del a 3 del z 3 is simply g prime of z l.

(Refer Slide Time: 29:33)



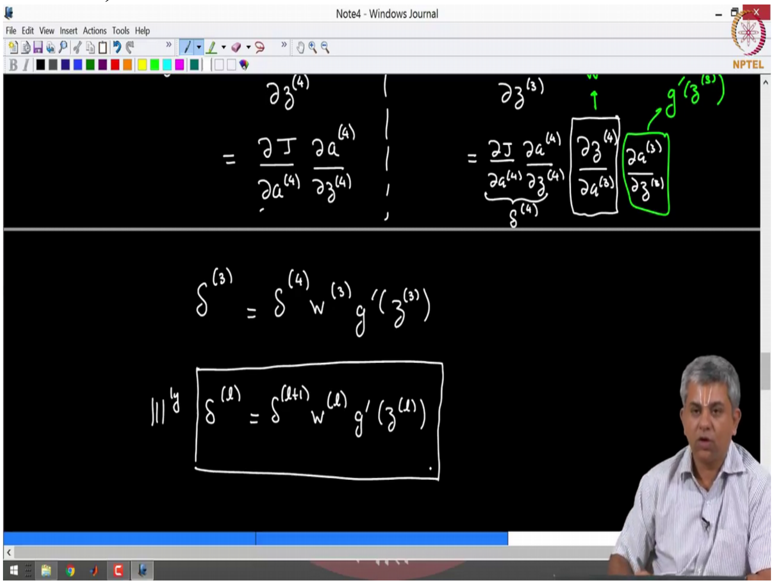
So let us write this, delta 3 is equal to delta 4 multiplied by w 3 multiplied by g prime z 3.

(Refer Slide Time: 29:49)



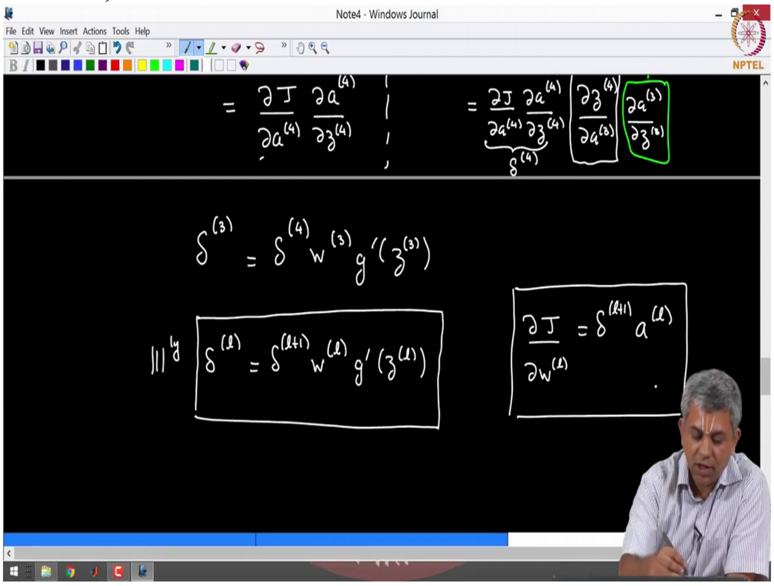
Turns out that similarly delta l is equal to delta l plus 1 w l...

(Refer Slide Time: 30:13)



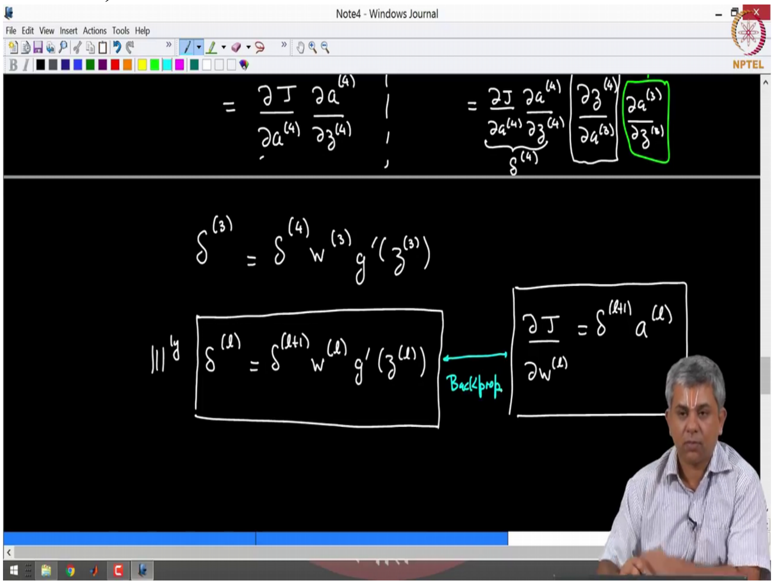
So if you combine this with the other expression we have, it was del J del w l, let us go back to the expression here, is equal to delta l plus 1 a l.

(Refer Slide Time: 30:34)



These two combined give us the back propagation algorithm.

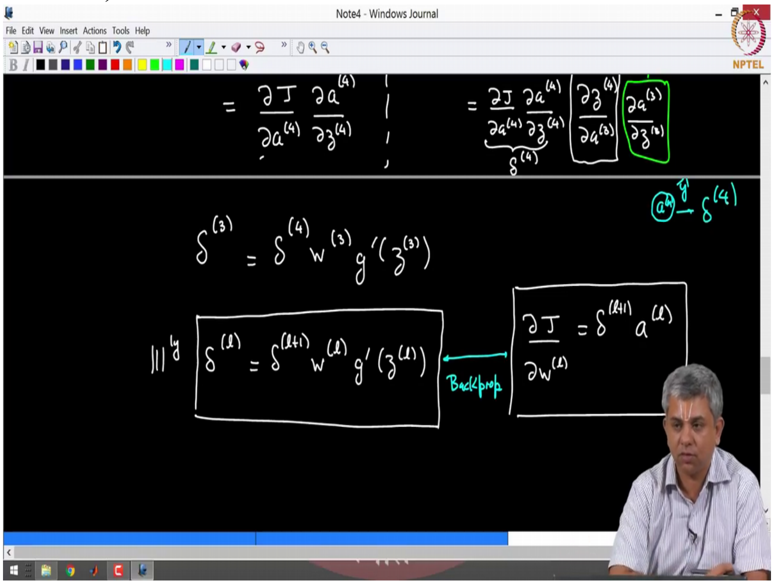
(Refer Slide Time: 30:41)



How is that? It is very simple.

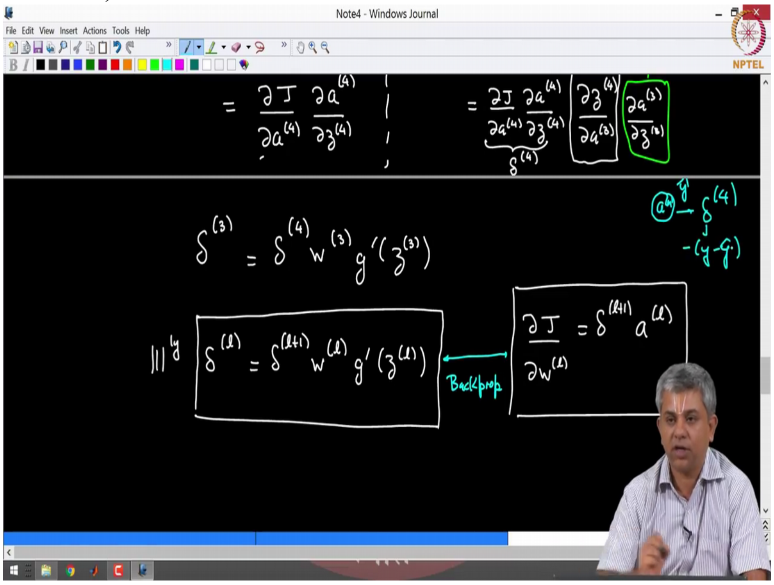
In this network you start with the last layer, Ok which was the a 4 or the y hat layer. Calculate delta 4 there.

(Refer Slide Time: 30:55)



In the example that we took, delta 4 was simply minus y minus y hat,

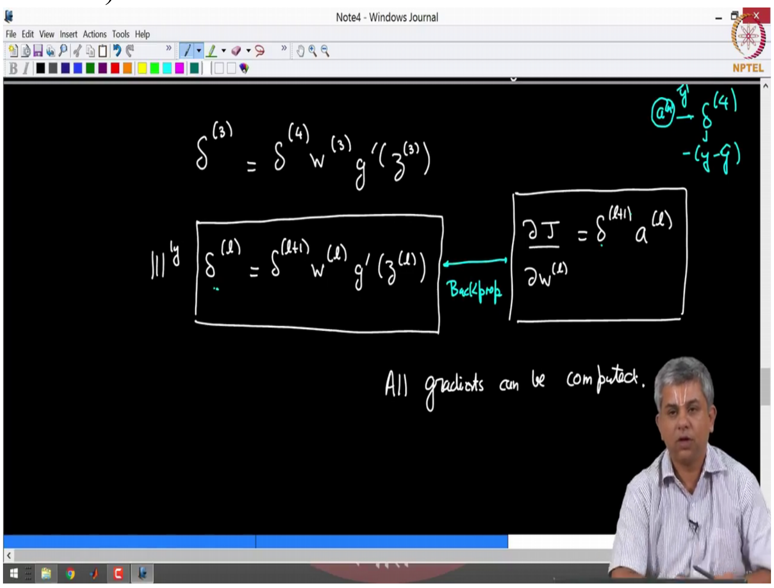
(Refer Slide Time: 31:02)



Ok with the particular nonlinearity and the cost function that we took, Ok.

Once you have delta 4, using this expression you have delta 3, you have delta 2. And then using this expression you simply have del J del w 1, del J del w 2 and del J del w 3. So every single thing, so all gradients can be computed.

(Refer Slide Time: 31:33)

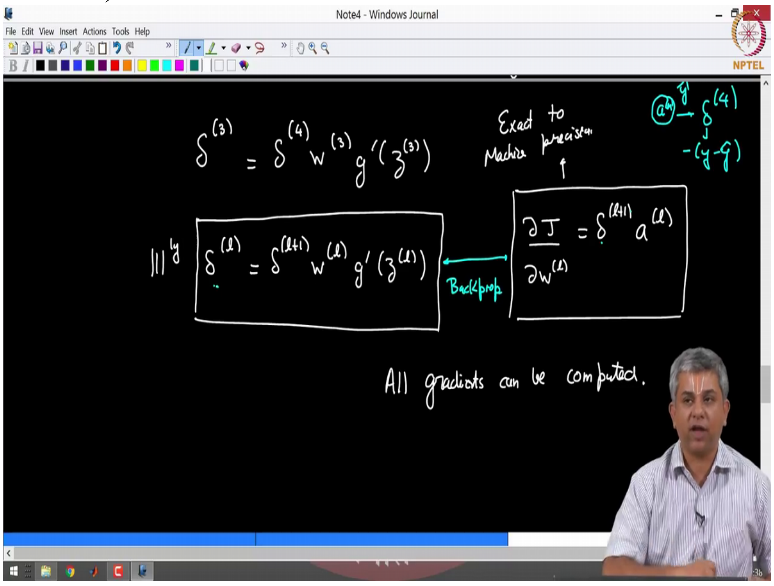


So notice that for one particular gradient computation, unlike finite difference, you do not have to go to each J or each w and calculate a simple, a different perturbation and calculate a different weight. That is much too expensive. In fact we do it only in order to cross check whether we have written the gradient, whether we have written the back propagation routine correctly or not.

Other than that, it is actually possible to do one single pass, calculate all the as and then simply do one, one single back pass, Ok or back prop step and calculate all the gradients.

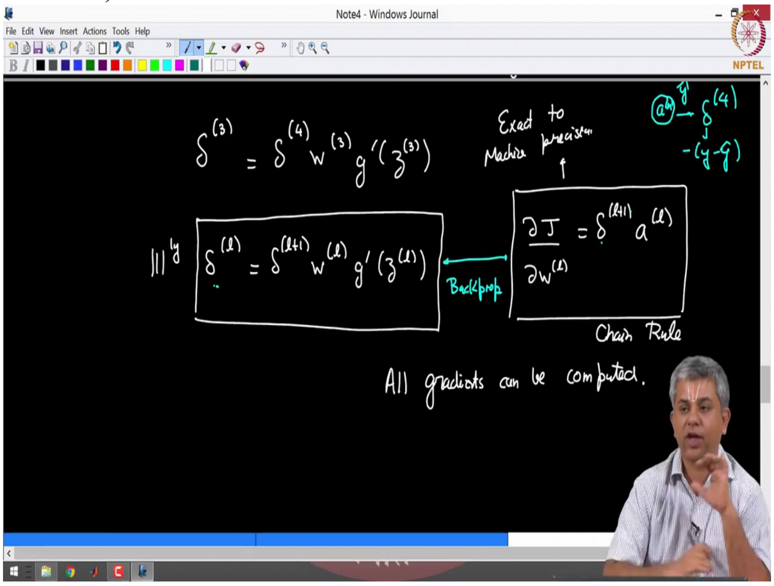
In fact these gradients are exact to machine precision. Why? Because we have not used any approximation anywhere.

(Refer Slide Time: 32:20)



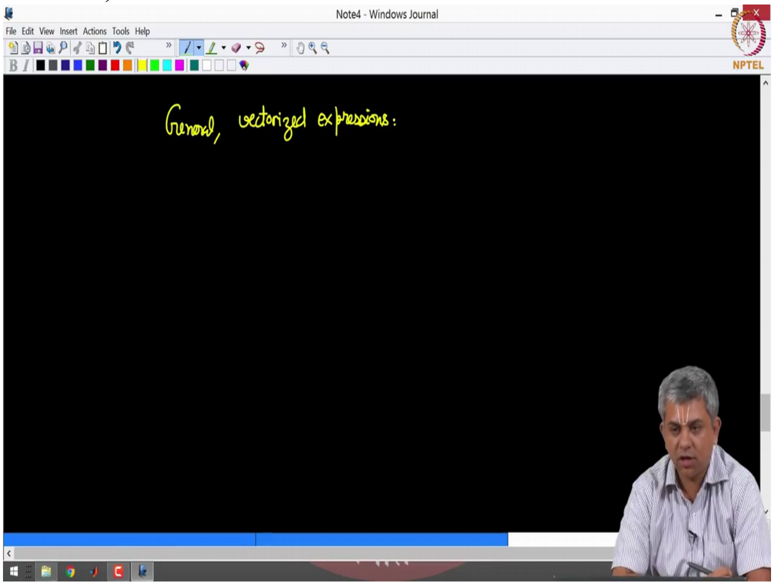
We have only used a simple Chain Rule, Ok so this is simply the algorithmization of Chain Rule.

(Refer Slide Time: 32:28)



We have not used any approximation here. The only approximation which will come is due to machine round off errors which we had seen earlier. This is the back prop algorithm in case of a scalar expression. For the general vectorized expressions,

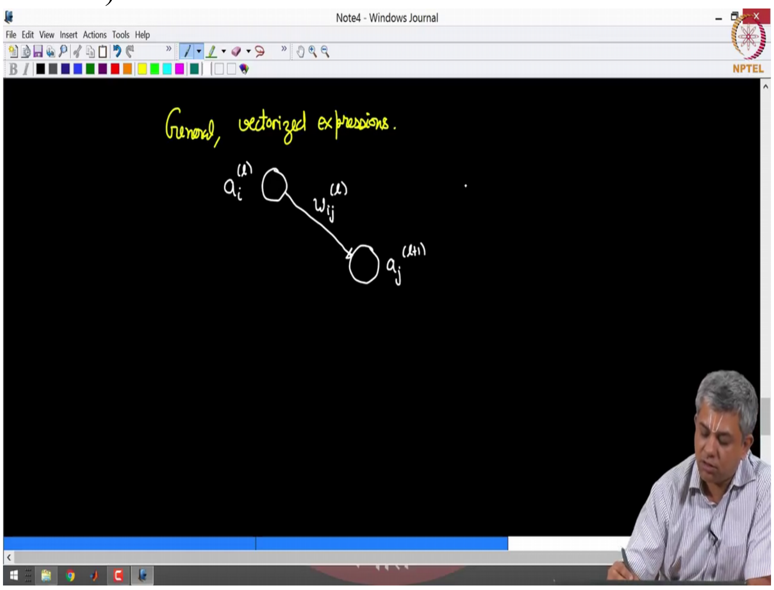
(Refer Slide Time: 32:56)



it turns out that the expressions are remarkably similar.

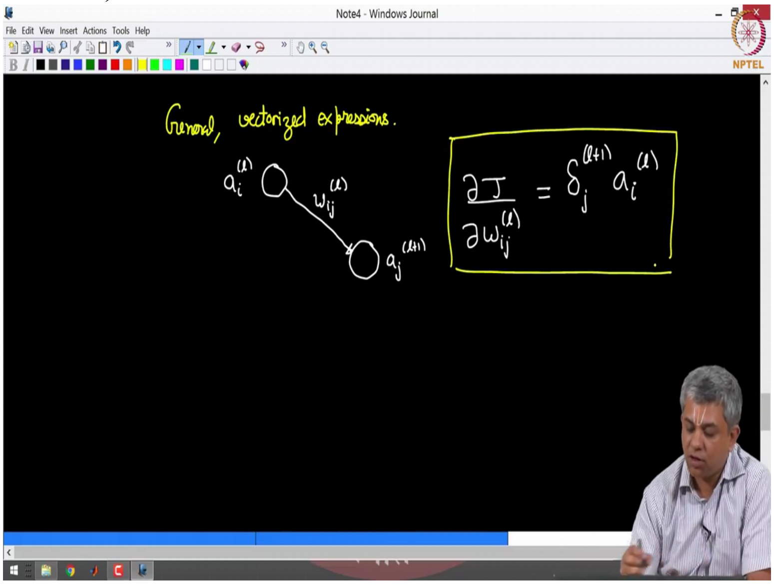
So if I take this case. You have the weight w i j l connecting a i l to a j l plus 1.

(Refer Slide Time: 33:19)



Then if you want del J del w i j l, this is equal to delta J l plus 1 multiplied by a i l. It is actually remarkably similar to the previous formula that we have. It is very simple.

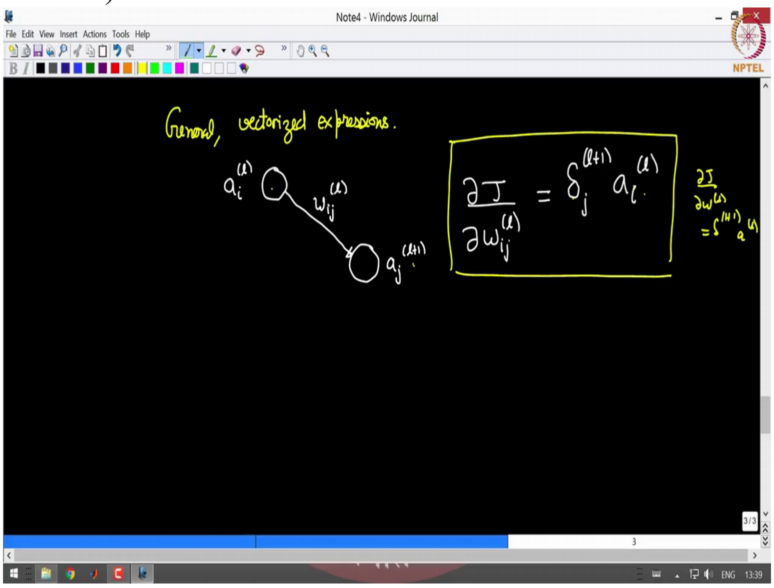
(Refer Slide Time: 33:46)



Error in activation of the next layer multiplied by the activation of the previous layer. This is all there is.

Please compare it to our other expression which was del J del w l is equal to delta l plus 1 multiplied by a l. This is very, very similar. Ok this is the full scale expression

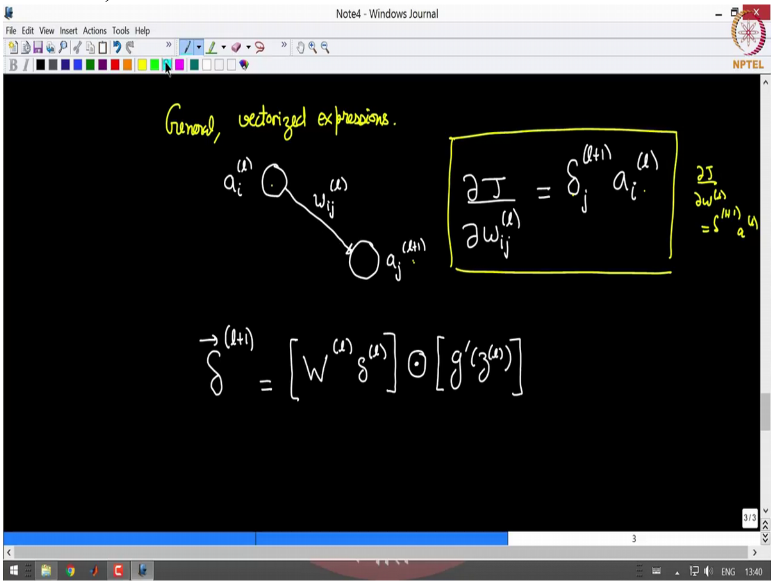
(Refer Slide Time: 34:06)



in case of a fully connected layer.

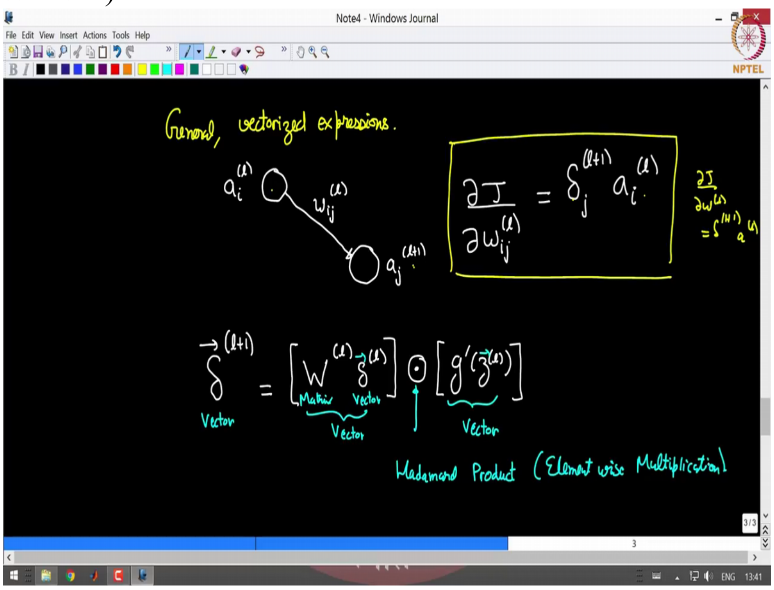
What about the other activation formula? Now you will have to deal with whole vectors. delta l plus 1 turns out, is equal to weight at l multiplied by W l...

(Refer Slide Time: 34:43)



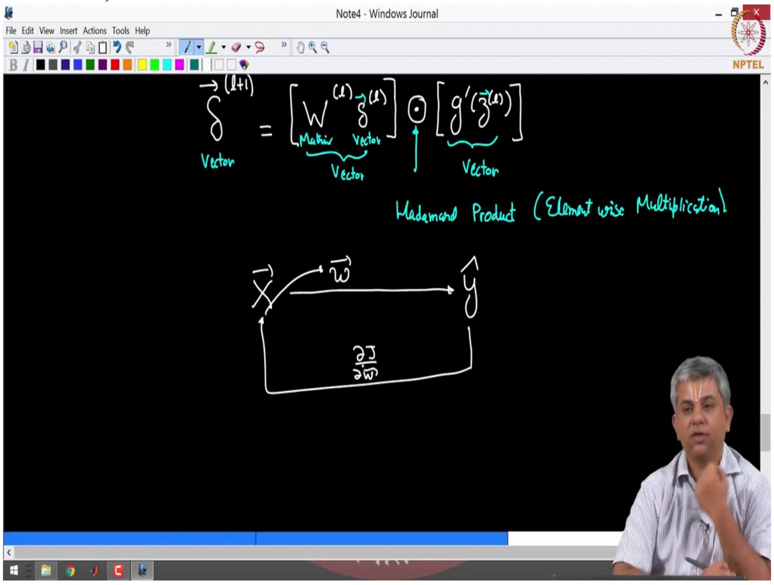
Notice that this is a matrix, this is a vector. When you multiply it the two, you will get a vector. This is also a vector. The sizes work out due to the weight matrix. This is another vector. z is also a vector now. And this, remember is our Hadamard product which we had seen in week 1, what is called element wise multiplication.

(Refer Slide Time: 35:24)



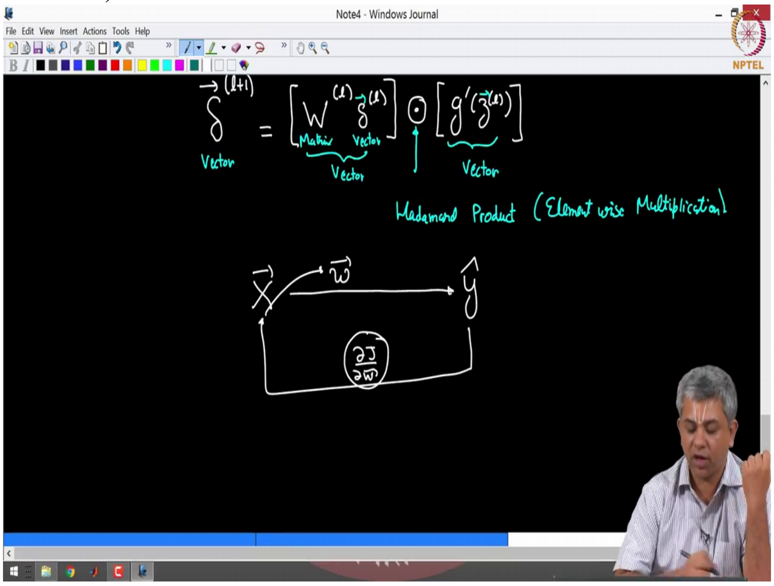
So just as a summary of the video what we saw is, if you treat a neural network more or less just like we have been doing either logistic regression or multi-longitudinal logistic regression or in fact even linear equation you had some x, somehow or the other using some guess weights w, you are getting y hat and you improve your w using del J del w.

(Refer Slide Time: 35:52)



The main computation in neural networks is calculating this del J del w for a given

(Refer Slide Time: 35:58)

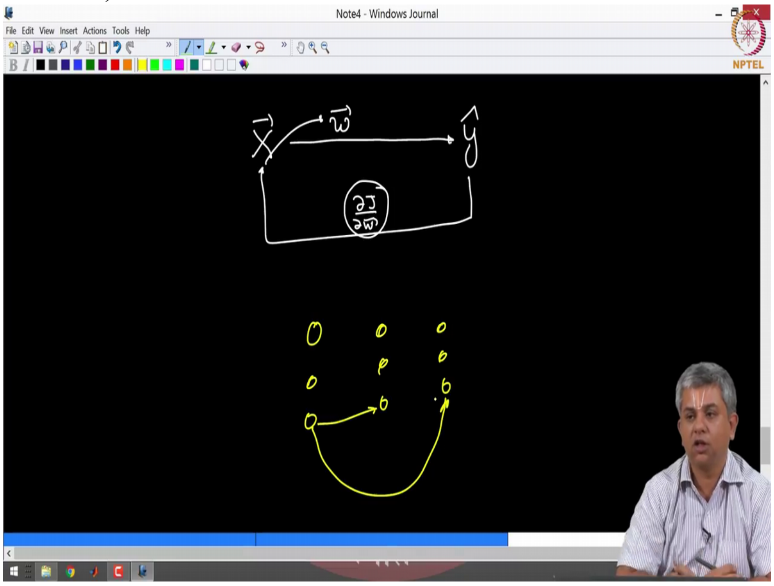


guess w. That we do using back propagation. The idea is already shown here. The reason it is called back propagation should be obvious. That is because we first calculate the error at the last layer and then start propagating.

Ok, if this was my total error how much was my each node responsible for this total error? So you take delta at the last layer, find out delta at the previous layer, previous layer, previous layer, previous layer so forth and then simply del J del w is delta at the next layer multiplied by activation of the previous layer.

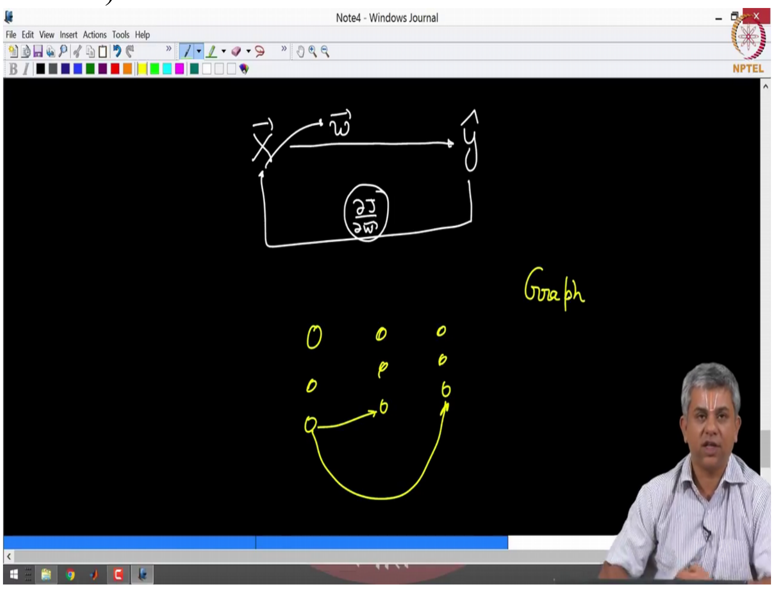
So this, in this video I just showed you a quick scalar derivation of back propagation. In general for complicated networks, you know, you could have networks with all sorts of skip connections which instead of going from here to here would do this.

(Refer Slide Time: 36:51)



So what Tensorflow and other software like that do is to create what is known as a graph, that is

(Refer Slide Time: 36:59)



they find out how is each node connected to every other node, this is how you would represent, in fact Tensorflow network diagrams and based on automatic differentiation it does back prop for you. You do not have to write it.

Currently nobody really has to write back propagation routine. It is actually already available in every single piece of software; this is just for you to build your intuition, on what tends to happen. As we will see in next weeks, it is sometime troublesome when you have very large networks because what is called gradient does not flow back, Ok.

So if you have a very, very long sort of back propagation algorithm. Errors multiply and due to machine epsilon problems, you tend to not have proper changes in gradient later on. So this explanation was just to give you the intuition about what could possibly happen. Thank you.