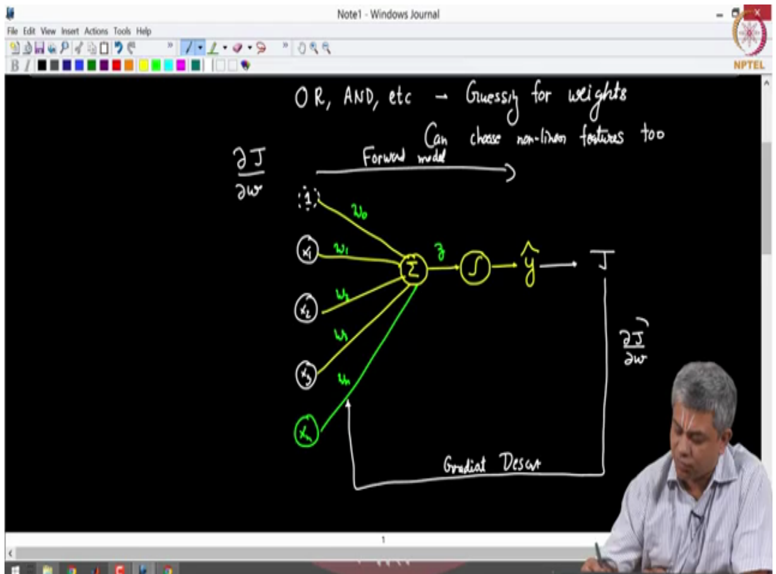
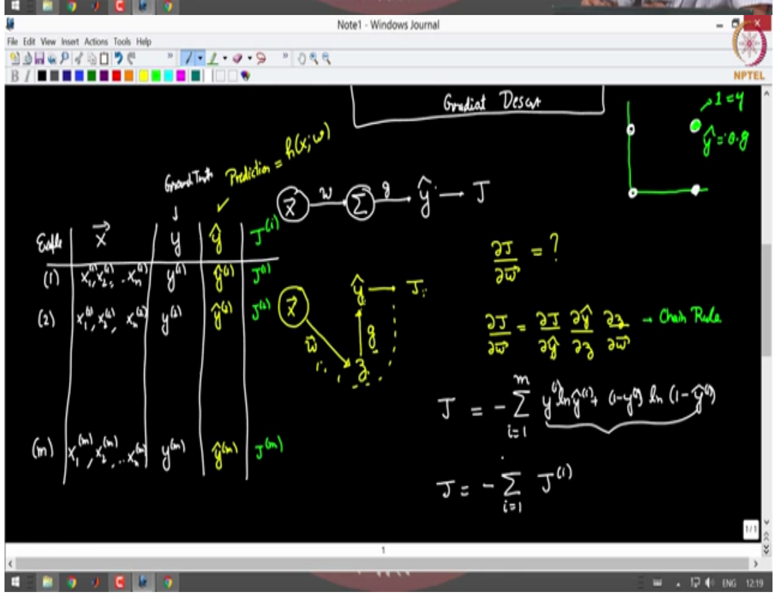
Machine Learning for Engineering and Science Applications
Professor Dr. Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology, Madras
Gradient of Logistic Regression

(Refer Slide Time: 0:25)





Welcome back. In the previous videos we saw our forward model of logistic regression. We also saw how we could use it to in order to simulate the OR gate, NOR gate etcetera. And during all these cases what we were doing is we were sort of guessing for weights. Without actually doing the gradient decent, sort of by heuristic sort of rule of thumb guessing we figured out some weights for which we could replicate the OR gate. Now obviously that happened well because we had only four data points and we just had to fit a classifying line between those four data points. In general, of course it is not really possible to just guess for a good line, which is why we look at how to calculate the gradient of the logistic regression which is will complete our loop.

So remember for logistic regression we are dealing with input modes as usual, x1, x2, x3 and we can have our usual bias unit sitting there. We have one output. Why only one output? Because this is binary classification, so the output is either going to be 1 or 0 or in our prediction case y hat which lies between 0 and 1. And all these put together let us split it into two. We have our summation and then we have our sigmoid. This gives us y hat. At the end of the summation, whatever we get is z and at the end of the sigmoid whatever we get is y hat.

Now the weights which we do not know are here, w not, w 1, w 2, w 3 and so on and so forth up till w n. Remember just like in linear regression you could choose x 1, x 2, to the x 1 square, x 3 to the x 1, x 2, et cetera. So you can choose non-linear features as well. Okay, this is just as a reminder. So what is missing in this picture? We do not know these ws. This is just the forward model and we follow our usual procedure. You have the forward model. You guess for the ws. You get a y hat, from the y hat you get a J. From the J you feedback using del J, del w.

And through gradient decent or some version of gradient decent to calculate all that and improve guess for the ws. So which is missing here is of course this del J, del w. So I am going to write the whole thing in a vectorial representation. You have x vector, it runs through a sigma using a w, then through a non-linearity, through a sigmoid you get y hat. And this y hat is what gives you J. So just for simplicity or clarity, I am going to represent this slightly differently.

Let us take x vector, I will show the sigma here. I run it through w, I get z. I run z through a non-linearity and I get y hat. From y hat I get J. And what I want is del J, del w vector. So question is, what is the expression for del J, del w vector? Now this is fairly straightforward. Let us trace this back. Del J, del w vector equal to del J, del y hat times del y hat del z, times del z del w vector. So this is sort of the dependency. J changes because w changes. Why? Because J changes due to changes in y, y hat. Y hat changes due to changes in z and z changes due to changes in w.

In other way, when I perturb w, it will perturb z which will perturb y which will perturb J. So that is what we are chasing down here. This is essentially the Chain Rule. When we come to neural networks, you will see that it is exactly the same idea which is applied for what is called a back propagation algorithm. So let us now calculate the expression for this. So let us look at each of these terms individually. First, what was J? J if you recall, I had written as summation of y ln y hat, plus+ 1 minus- y, ln 1 minus- y hat.

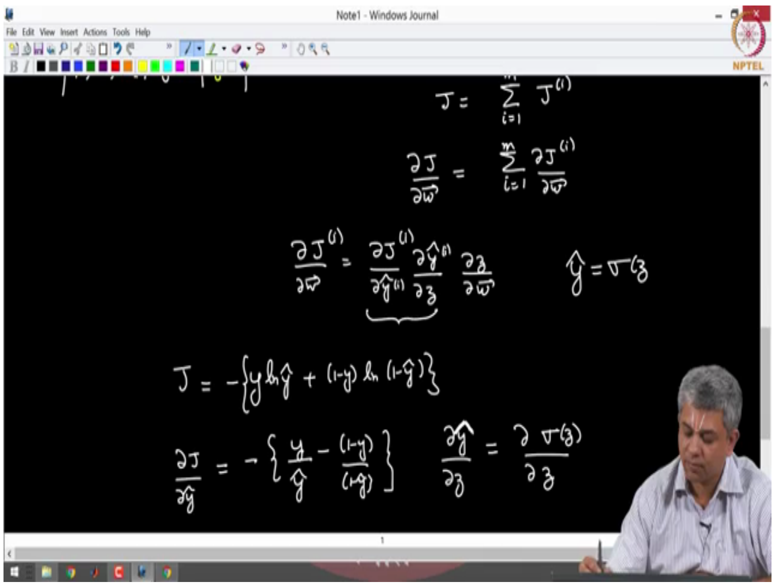
I am going to put an i here and i equal to 1 to m. And what is this i equal to 1 to m? It is very very similar to what we did, in fact it is the same to what we did when we were doing linear regression. Recall, suppose you have x vector, as in the example that we have shown here, and you have multiple examples or multiple data points. So for the first data point x vector was x 1, x 2, up till x n where you have n features. And I am going to put a superscript 1 to say that this is the first set of data points.

In the case of the OR example, we had four such data points. So you had an x 1 1, x 2 1 which was 00 in that case. So similarly have x 1 2, x 2 2, x n 2. And we could have m such examples very similar to the multiple linear regression example that we did. Now we have the ground truth and since this is binary classification, ground truth here is y 1. y 1 will be either 0 or 1, y 2 will be either 0 or 1 and you have y m. Then we have our y hat which is our prediction which is h of x given the parameters w.

So you are going to have y hat 1, y hat 2 up till y hat m. Finally I am going to introduce something new just for clarity. I can have J i, what is that? This is the loss just due to the ith example. So going back here, suppose you had four points and let us say this value was 1. This was y and y hat was let us say 0.8. The very fact that y hat and y differed will give you some amount of loss. So J i is just the loss in the ith example. So you will have J1, J2, up till Jm.

Even in linear regression you can see if you have a line and you have multiple points, the loss for each one of those points will be denoted as J i. So coming back here, all I am writing is J is the summation of all the binary entropy losses from each individual data point.

(Refer Slide Time: 9:31)

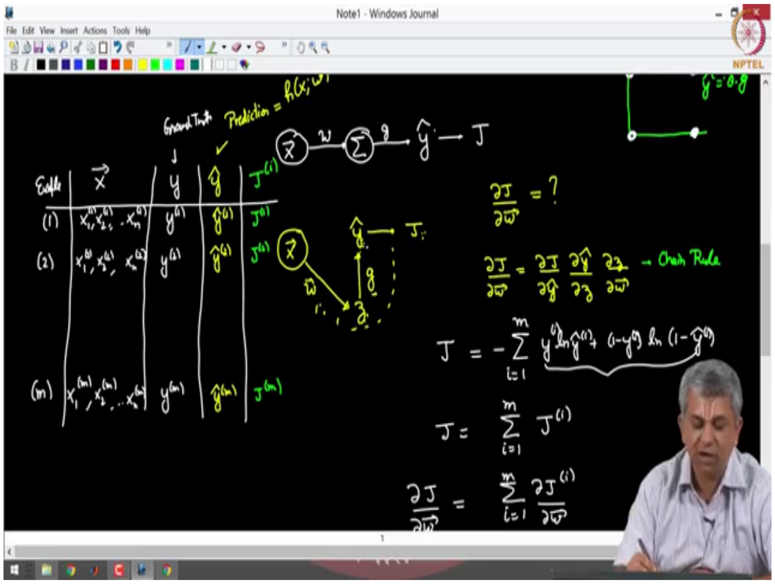


So it is simply the summation of all individual losses. So suppose I want del J, del w, this is also going to be, actually I will remove the minus, let us include the minus within the J i. This is del J I, del w vector. So we have this del J i del w vector is equal to del J i del y hat, del y hat del z, del z del w vector.

Technically speaking, this is y hat. Okay, for now I am going to drop the i in the future expressions and we will just sum it back just for clarity of notation. Let us look at this term first. So I am going to drop the i as I said. J is minus- y ln y hat, plus+ 1 minus- y ln 1 minus- y hat. Notice that the derivative is with respect to y hat because that is what depends on w. y is fixed, y is what we gave, y are the labels that we give. y hat is the parameter that, y hat is the hypothesis function that we get out of the prediction using our parameter w.

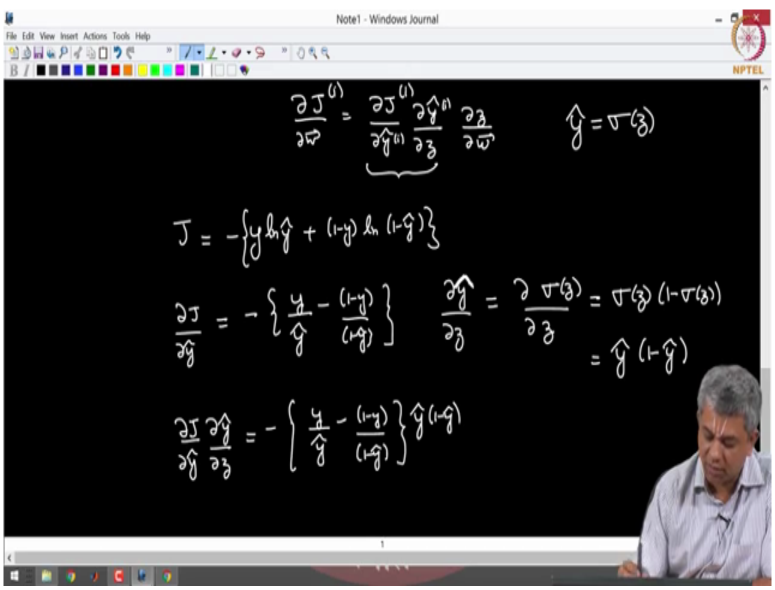
So what is del J, del y hat? Del J del y hat, this term, is minus- y by y hat, minus- 1 minus- y by 1 minus- y hat. So we have that expression. What about del y hat del z? Del y hat del z is, remember y hat is simply sigmoid of z, that is how we calculated it.

(Refer Slide Time: 11:46)



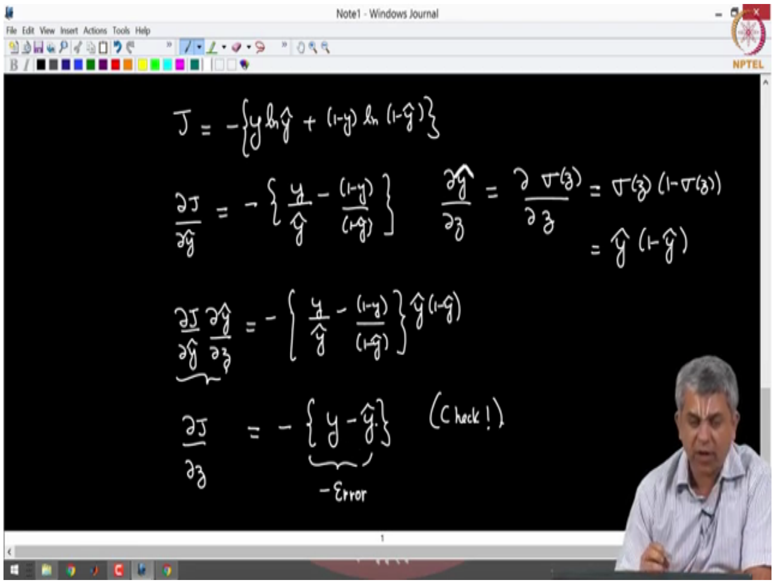
If you come here to this picture, y hat was calculated as g of z or sigmoid of z.

(Refer Slide Time: 11:55)



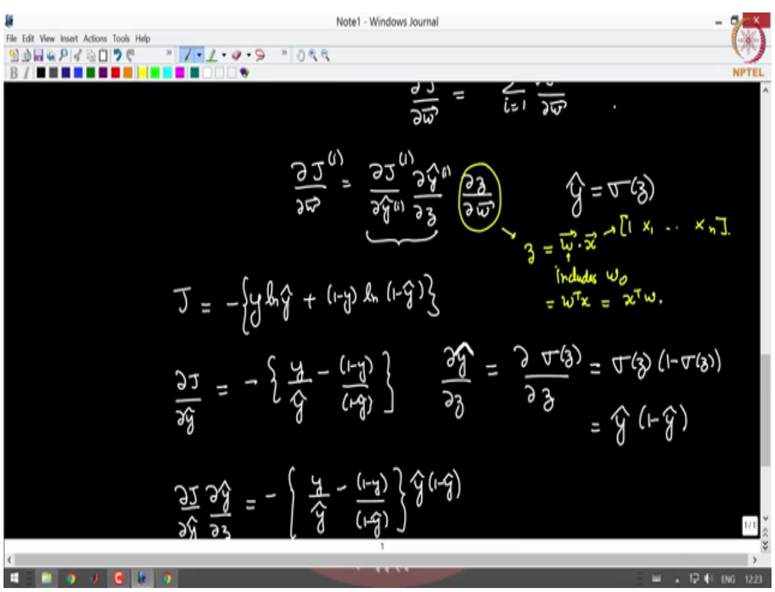
Now luckily in a previous video we already calculated this derivative. This was simply sigmoid of z times 1 minus- sigmoid of z, which is the same as y hat times 1 minus- y hat. So if you put it together, then you get del J, del y hat multiplied by del y hat del z, is equal to minus- y over Yes. hat, minus-, 1 minus- y over 1 minus- y hat, times y hat times 1 minus- y hat.

(Refer Slide Time: 12:37)



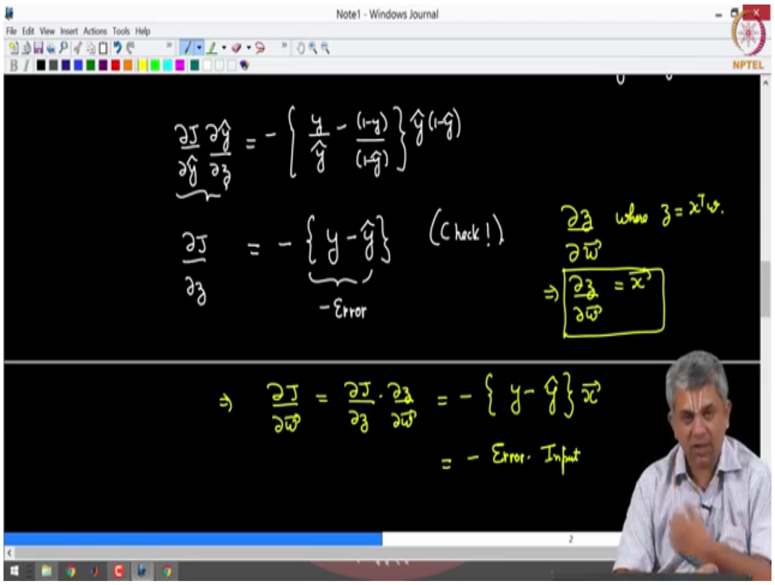
And if you calculate this, you get a very simple expression which is simply minus- y minus- y hat. Please check this for yourself. So we can call this expression even del J del z by the chain rule. Del J del z is equal to minus- y minus- y hat. This is simply the negative of the error. That is if you made a prediction y, y hat and the ground truth was y, del J del z is simply minus- y minus- y hat.

(Refer Slide Time: 13:16)



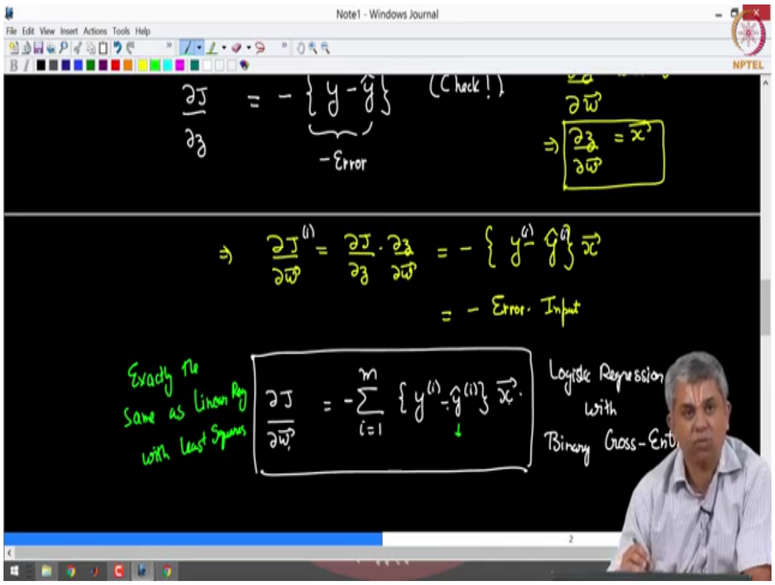
We have one other term left in this expression which is del z del w hat. Remember z was w vector dotted with x where w vector includes w not. This expression we saw the augmented w and the augmented x. x now includes 1, x 1, up till x n. So this can be written as w transpose x, also as x transpose w.

(Refer Slide Time: 14:08)



We saw in the linear algebra videos that in such a case when you have del J, del w where z is equal to x transpose w, this means del z del w is simply equal to x vector. You can do this by algebra also or you could do this using the notation that I used in week 1 of this course. So if you put these two together, you get del J del w, equal to del J del z, multiplied by del z del w. This is equal to minus- y minus- y hat, multiplying x vector, which is minus- the error multiplied by the input. Now there are several noteworthy things here. I will go over them one by one.

(Refer Slide Time: 15:13)

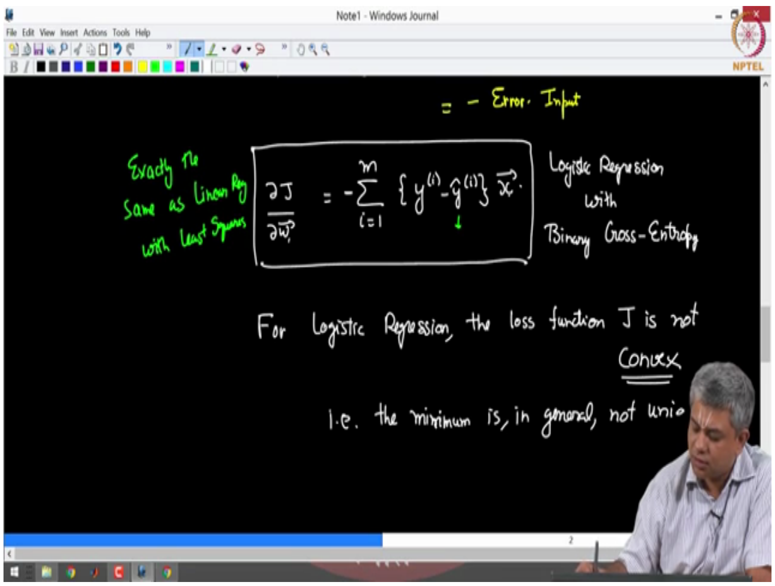


First I will the general expression, this of course is del J i del w. So if I look at del J del w, this is going to be summation from i equal to 1 to m of y i, minus- y hat i, multiplied by x vector. So for example, if you wanted del J del w not, you will get y i minus- y hat i, multiplied by x not which is simply 1, so on and so forth. For each component of w, you take the corresponding component of x. This is a vector, that is also a vector. So this expression is for logistic regression with binary cross entropy. Now some of you might recall that we had exactly the same expression, except for the factor of 1 by m which we can either include or not, exactly the same as linear regression with least squares.

So please refer back to your notes and check whether this is true or not. So this is actually remarkable. You get the same expression for the gradient for logistic regression with binary cross entropy as you get with linear regression with least squares. This is true even when you start including regularization. So when you include the regularization of, as Dr Ganpathi had shown earlier, if you include regularization of lambda norm of w square, that would also add normally within logistic regression also.

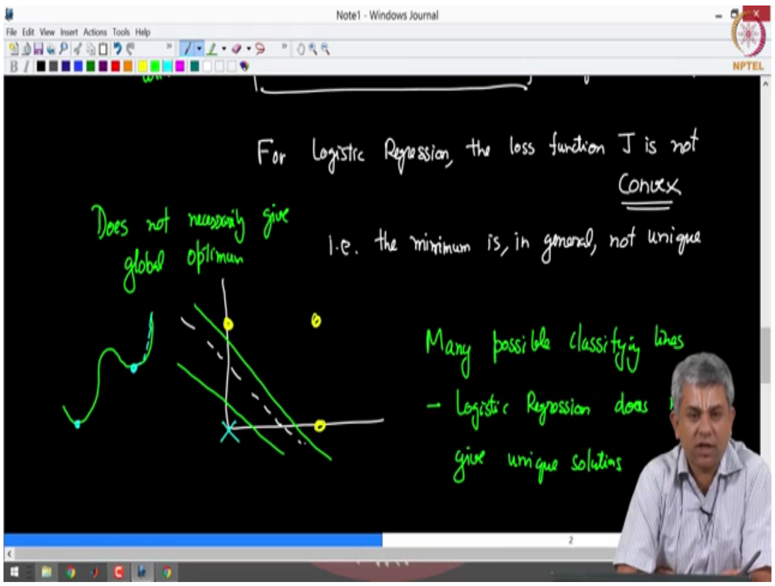
The same thing holds true even for neural networks. So but the expression is actually exactly the same. This does not mean that the J is the same. Remember that this y hat means different things for linear regression and for logistic regression. For linear regression y hat was simply w transpose x, for logistic regression y hat is now sigmoid of w transpose x. So please do remember that.

(Refer Slide Time: 17:48)



The second noteworthy thing here which is not obvious is that for logistic regression the loss function, J is not convex. So I will not go into what a convex set means but you can think when we had least squares generally the cost function will look simply like a paraboloid. So this is what is called a convex function where you have only 1 minima. The most important thing for us to know is that the minima, the minimum is in general not unique. We can understand this kind of intuitively too.

(Refer Slide Time: 18:48)

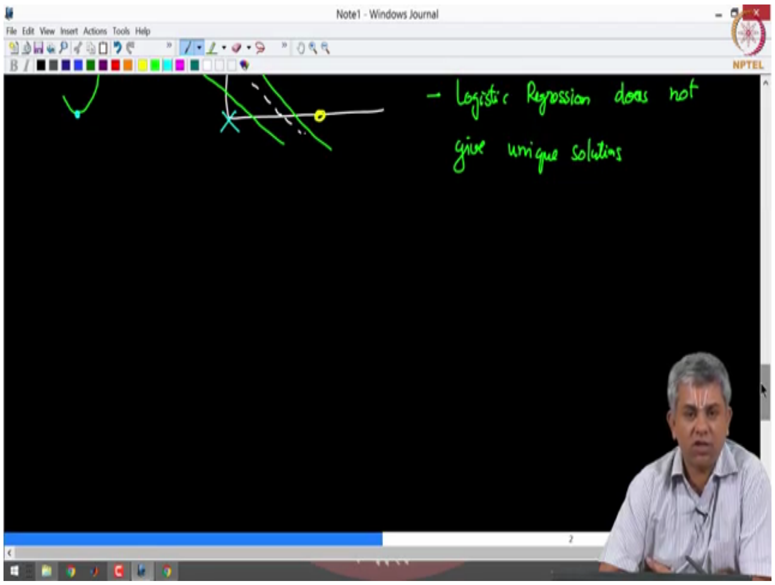


For example, we were trying to classify the set. Now notice when we were doing OR gate, we chose one line which was like this. But there is no reason to choose only this line. Even this line functions as a classifying line, this line functions as a classifying line. So there are many possible classifying lines. So logistic regression does not give unique solutions. So in our whole process depending on which w you start with, remember that our initial sets of weights we were guessing randomly. So the random guess depending on where it is, you might get one classifying line or the other.

So logistic regression actually depends on your initial conditions which classifying line you get. So that is an important constraint. This is also a problem with neural networks. It will never necessarily, does not necessarily give the global optimum. So in gradient decent let us say you have two local minima. At one place your gradient is 0, at another place also your gradient is 0. For example, if you have something of this sort, it is quite possible that your gradient decent comes here and gets stuck and does not move from there rather than come here which would actually be the local optimum.

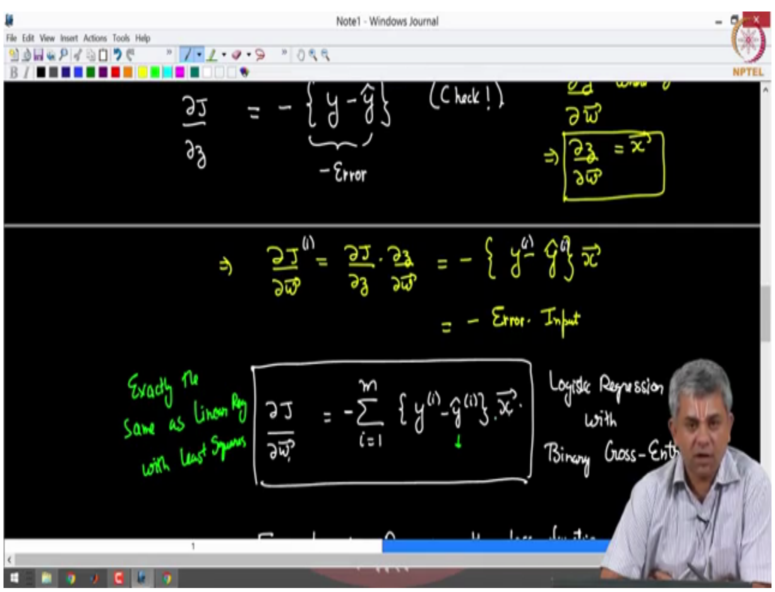
So this is true whether it is logistic regression or whether it is neural networks which we will see shortly in future videos. But for linear regression typically if you converge, you will converge to only the global optimum because there is only one optimum because it is a convex function. So this is important for you to remember that logistic regression does not necessarily deal with convex loss functions.

(Refer Slide Time: 21:15)



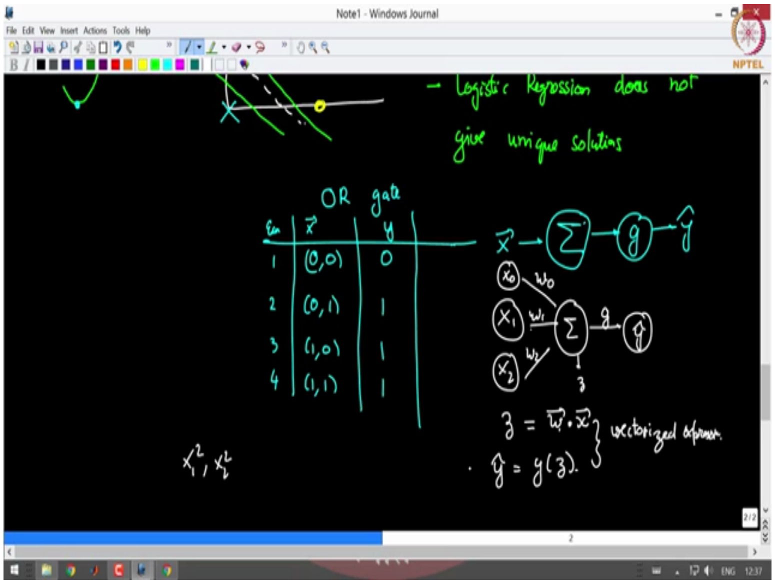
So in order to sort of solidify your understanding of the logistic regression process, let us now look at a code which does logistic regression. You will see that this is just a small modification of the linear regression code that we had already used earlier. We had written a generalized linear regression code.

(Refer Slide Time: 21:40)



So since the gradient expression for logistic regression is practically indistinguishable from that of linear regression we will use the same code in order to do logistic regression. I will take the simple example of the OR gate.

(Refer Slide Time: 21:50)

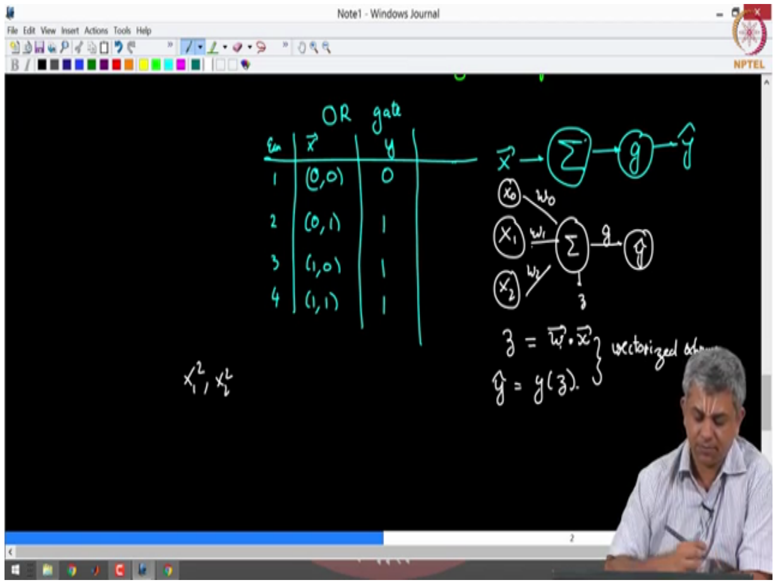


And in this case we will start with once again our example, 1, 2, 3, 4, we will treat it as four data points. And x vector which was 00, 01, 10, 11 and we have our ground truth which was 0, 1, 1, 1 and we put the same model as before, a summation followed by a sigmoid gives us y hat after taking in x. Now what I will also be doing is both in code which you will notice as well as in the expression here, we will write it as if we are writing vector expressions just to tell you how that works.

So if you have x 1, x 2, and x not here which is just 1, we sum these three up, w not, w 1, w 2. What comes out is y hat after you put sigmoid and g. You can write this as z which is the intermediate output of this, z is equal to w dot x. Remember this w also includes the w not which is the biased term and x also includes x not. And y hat is equal to g of z. This would be the vectorized expression. And this is what lets us write a general code with great amount of ease because suppose I decide to use other non-linear features.

For example, you saw the XOR gate example, in the XOR gate example suppose you want to include x 1 square, x 2 square et cetera, you could simply include extra features x 3, x 4 and that would logistic regression would still work as it is.

(Refer Slide Time: 24:08)



So we will look at a vectorized code for this in the next video which would be the code. Thank you.