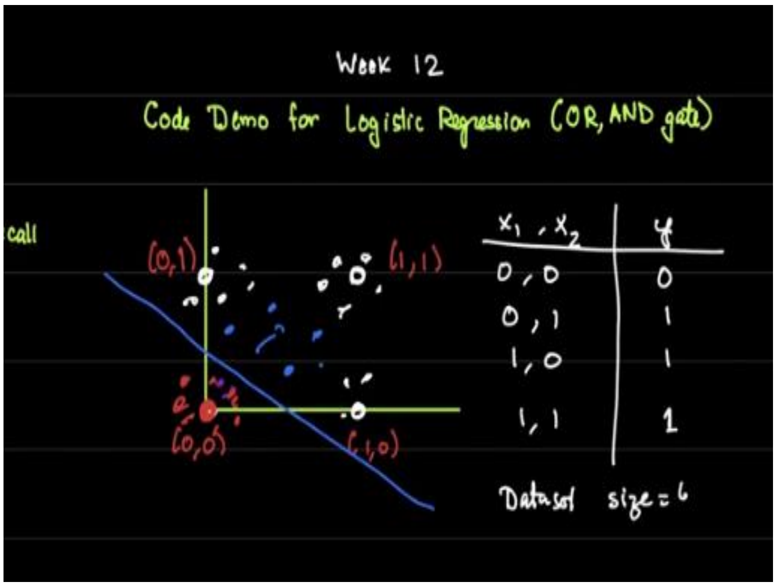
Inverse Methods in Heat Transfer

Prof. Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology-Madras

Lecture - 62
Code Examples of Logistic Regression -- OR and AND Gates

Welcome back. In this video, I want to show you a code demo for logistic regression, especially for OR and AND gates.

(Refer Slide Time: 00:29)



Week 12

Code Demo for Logistic Regression (OR, AND gate)

x_1, x_2	y
0, 0	0
0, 1	1
1, 0	1
1, 1	1

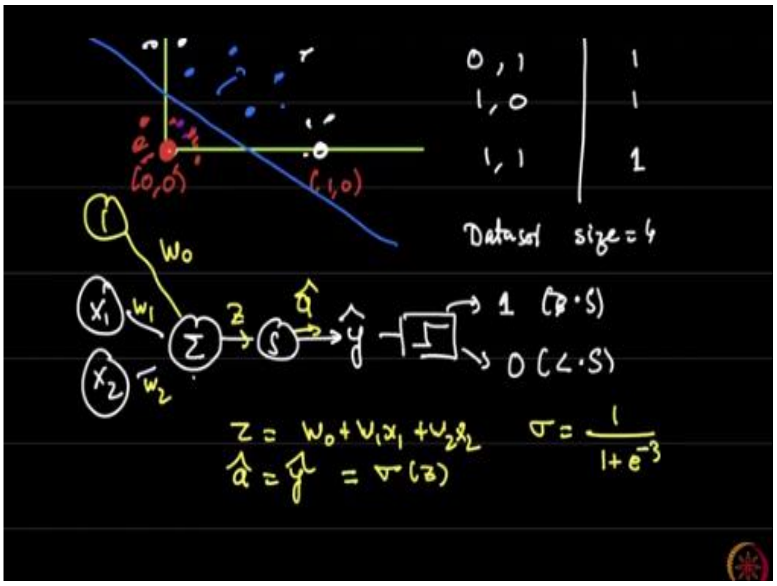
Dataset size = 6

Just recall since we have had some distance from this, what we mean by this, so remember logistic regression was a simple classification algorithm. And when we are given some data, let us say some data of the sort of an OR gate, which has four possibilities 0,0 0,1 1,0 and 1,1. Then we typically get 0 only at one place, which is at 0,0. And everywhere else, we basically get, this is 1,1.

Everywhere else we actually get this one should be 1,0, everywhere else we actually get 1, okay? So, this is what is made as the table that when x 1 is 0, 0, then the Ground Truth is 0. And everywhere else it is actually 1. As I had told you during our logistic regression discussions, basically the classification line goes somewhere here, so that every data here would be labeled.

If I give some arbitrary data here to be labeled as 1 and any arbitrary data on this side, that will be labeled as 0. This is supposed to be a proxy for lot of data points here and lot of data points here. And of course, we are just giving four data points, just as representative data. So, our data size is a limited data set size, this is equal to 4.

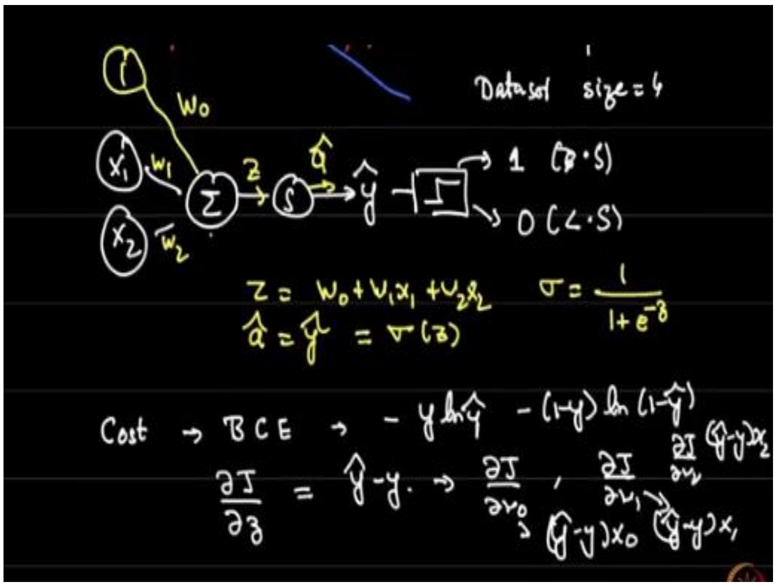
(Refer Slide Time: 02:19)



And the model with which we are trying to accomplish this is the graphical model or that I have shown here. Here you have y and of course or $\hat{y}$ and you put a filter here, if greater than 0.5, then you call this 1. If it is less than 0.5, we call it 0. So, I hope you recall all this. And the model here was of course, w_0, w_1, w_2. Then the summation $z = w_0 + w_1x_1 + w_2x_2$. And that is the output here.

And â is $\hat{y}$. It is the output here is what we call â. And that is of course sigmoid of z. Recollect that sigmoid is $\frac{1}{1+e^{-z}}$.

(Refer Slide Time: 03:23)

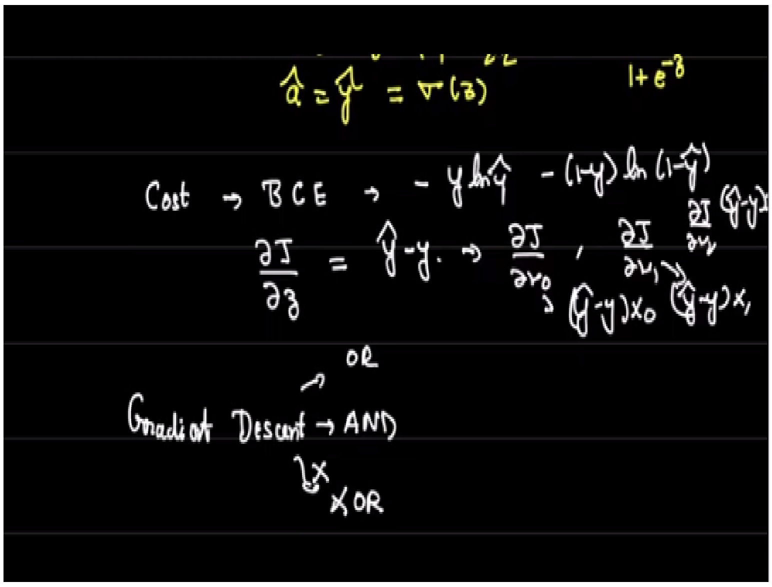


Dataset size = 4
1
w_0
x_1
w_1
x_2
w_2
Σ
z
S
$\hat{a}$
$\hat{y}$
1 (G.S)
0 (L.S)
$z = w_0 + w_1x_1 + w_2x_2$
$\hat{a} = \hat{y} = \sigma(z)$
$\sigma = \frac{1}{1+e^{-z}}$
Cost → BCE → $-y \ln \hat{y} - (1-y) \ln (1-\hat{y})$
$\frac{\partial J}{\partial z} = \hat{y} - y \rightarrow \frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}$
$(\hat{y}-y)x_0$
$(\hat{y}-y)x_1$
$(\hat{y}-y)x_2$

Finally, we had our cost function. This was the binary cross entropy cross function, and that was given as $-yln\hat{y} - (1-y)ln(1-\hat{y})$. Recollect that we had the calculation that $\frac{\partial J}{\partial z}$ if you calculate it always comes as $(\hat{y} - y)$. And we use this finally in backprop and calculate $\frac{\partial J}{\partial w_0}$ and $\frac{\partial J}{\partial w_1}$ and $\frac{\partial J}{\partial w_0}$ was $(\hat{y} - y)x_0$.

$\frac{\partial J}{\partial w_1}$ was $(\hat{y} - y)x_1$. And $\frac{\partial J}{\partial w_2}$ was $(\hat{y} - y)x_2$. So, all this is just to remind you of what we had done here. The idea of course, is in this case, to start with arbitrary w_0, w_1 and w_2, just some random weights I had given you in the video in the ninth week, some heuristic arguments on how to come up with w_0, w_1 and w_2.

(Refer Slide Time: 04:37)

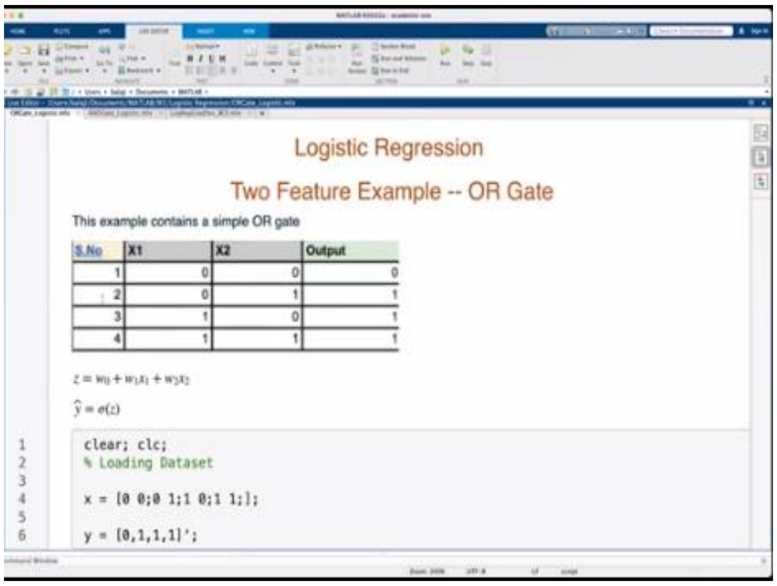


$\hat{a} = \hat{y} = \sigma(z)$
$1+e^{-z}$
Cost → BCE → $-y\ln\hat{y} - (1-y)\ln(1-\hat{y})$
$\frac{\partial J}{\partial z} = \hat{y}-y. \rightarrow \frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}$
$(\hat{y}-y)x_0$ $(\hat{y}-y)x_1$ $(\hat{y}-y)$
OR
Gradient Descent → AND
XOR

But here we wish to do a full gradient descent solution. And what I want to show you in the video that comes, the code demo that comes is that I can use the same gradient descent idea to find OR gate and to find AND gate. But it turns out it does not work for XOR gate. Why it does not work for XOR gate will be a little bit interesting. But as you remember, that is basically because XOR gate is not linearly classified.

But nonetheless, let us look at the examples that I am going to show you shortly on code. So let us move on to the code. This was just a short video to remind you of the theoretical answers that we had so far. And we will move on to the code using this. So, I will see you in the code.

(Refer Slide Time: 05:27)



Logistic Regression

Two Feature Example -- OR Gate

This example contains a simple OR gate

S.No	X1	X2	Output
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	1

$z = w_0 + w_1x_1 + w_2x_2$

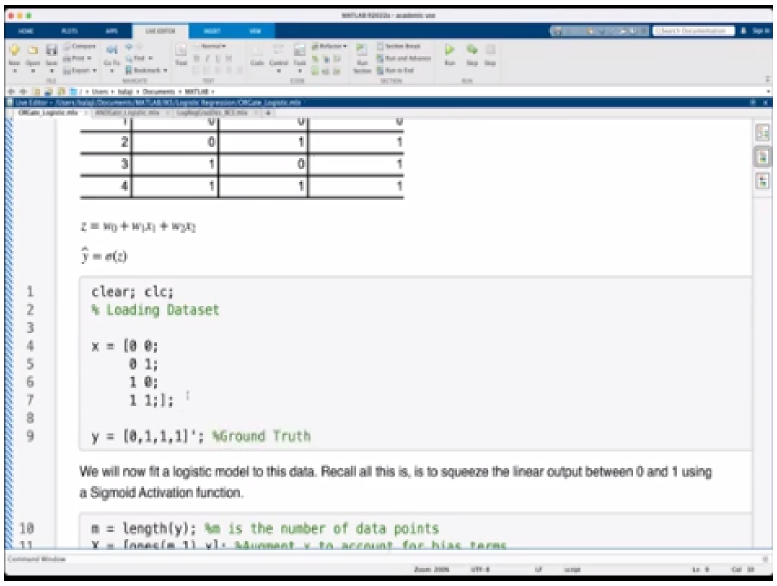
$\hat{y} = \sigma(z)$

```
clear; clc;
% Loading Dataset

x = [0 0;0 1;1 0;1 1;];

y = [0,1,1,1]';
```

Here is a code that you should find within your directories. It should have been shared by NPTEL. So, I have now written a general logistic regression code for a two-feature example. So. this is the data that I showed you earlier. Then we basically are treating it as if there were just four data points 0 0, 0 1, 1 0, 1 1.

And on the right-hand side, in the output column, of course you have y, which is the Ground Truth that is for 0 0 it is 0, etc. Now our forward model is decided by the simple model, $z = \mathrm{w}_0 + \mathrm{w}_1\mathrm{x}_1 + \mathrm{w}_2\mathrm{x}_2$. And $\hat{y}$ is sigmoid of z. So that is what we are trying to impose here.

(Refer Slide Time: 06:14)



2	0	1	1
3	1	0	1
4	1	1	1

$z = w_0 + w_1x_1 + w_2x_2$

$\hat{y} = \sigma(z)$

```
clear; clc;
% Loading Dataset

x = [0 0;
     0 1;
     1 0;
     1 1;];

y = [0,1,1,1]'; %Ground Truth
```

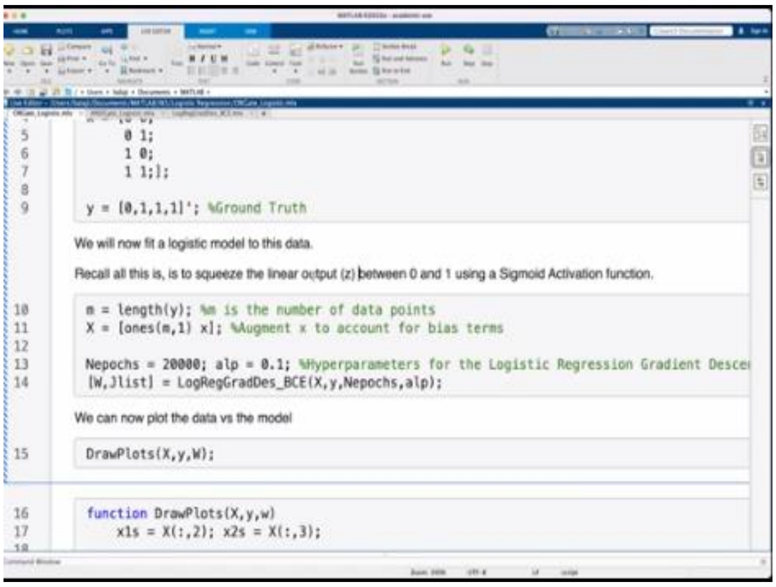
We will now fit a logistic model to this data. Recall all this is, is to squeeze the linear output between 0 and 1 using a Sigmoid Activation function.

```
m = length(y); %m is the number of data points
X = [ones(m,1) x]; %Augment x to account for bias terms
```

Now here is the data set. I have just written it in this way, this is just MATLAB style for writing these four data. We can write it this way too. It just makes it a little bit more compact, if I write it in the way that I wrote earlier. But as you can see, 0 0, 0 1, 1 0 and 1 1, and I have the appropriate y outputs here 0, 1, 1, 1, okay.

So, I will just go back to the old style of writing it, I have just written it as 0 for this 1 for this 1 for this and 1 for this. That is the Ground Truth. This of course is the general input.

(Refer Slide Time: 07:00)



```
         0 1;
         1 0;
         1 1;];

    y = [0,1,1,1]'; %Ground Truth
```

We will now fit a logistic model to this data.

Recall all this is, is to squeeze the linear output (z) between 0 and 1 using a Sigmoid Activation function.

```
    m = length(y); %m is the number of data points
    X = [ones(m,1) x]; %Augment x to account for bias terms

    Nepochs = 20000; alp = 0.1; %Hyperparameters for the Logistic Regression Gradient Desce
    [W,Jlist] = LogRegGradDes_BCE(X,y,Nepochs,alp);
```

We can now plot the data vs the model

```
    DrawPlots(X,y,W);
```

```
    function DrawPlots(X,y,w)
        x1s = X(:,2); x2s = X(:,3);
```

Now what we are going to do as written here is we are going to fit a logistic model to this data. What really, we are doing is we are squeezing the linear output, which is $z = w_0 + w_1x_1 + w_2x_2$ to lie between 0 and 1, which is our requirement okay, basically using a sigmoid activation function. So, I will just write it in two lines here. The linear output here is z of what we call linear activation, basically.

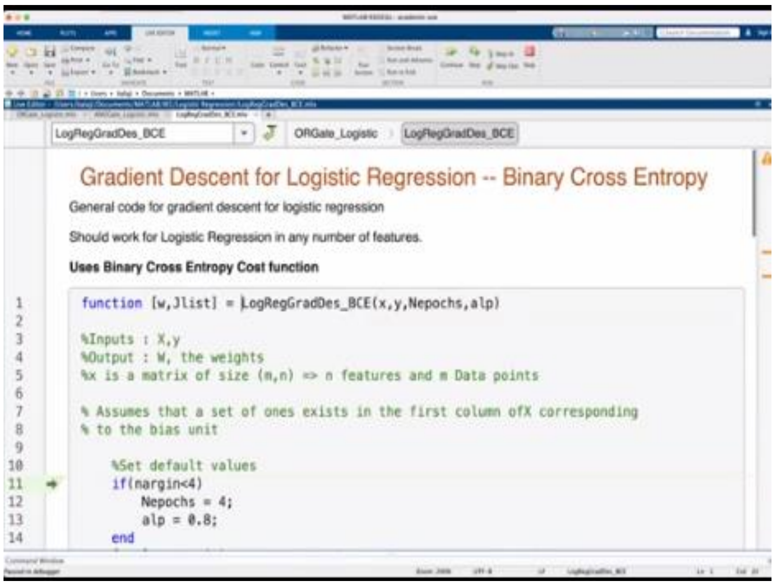
So, this is the way I have written it. I have written a general code. This basically m here means the number of data points. In this case, we have only four data points. Later on, when I show you the XOR gate, I will just hard code this, that is I will just assume m is equal to 4 when I use a full neural network. Now in this case, I have actually used an extra column of one, set of columns of 1.

You might recall this from our linear regression case. We used to use this all the time, just so that we can have a simple bias column here. So, I will just demonstrate what x looks like. So, you can see on your screens, x looks like 1 in the very first column and then you have 0 0, 0 1, 1 0 and 1 1, okay. So, the purpose is really simple. Purpose is to use the bias units appropriately.

Now this, I am going to use stochastic gradient descent. But we are going to use 20,000 epochs. Why 20,000 epochs? I have just chosen a random number. But the number of epochs could be lower, could be higher, it really does not matter too much, because this is a simple enough case. And I have chosen some random gradient learning rate value. As we saw earlier, if you use too high learning rate value, you get into trouble.

So, I have chosen alpha equal to 0.1, okay? Now here comes the interesting thing. What I have done now is basically updated the w okay, or I have actually started with some random guesses and actually have done a full-scale logistic regression gradient descent.

(Refer Slide Time: 09:23)



LogRegGradDes_BCE

ORGate_Logistic › LogRegGradDes_BCE

Gradient Descent for Logistic Regression -- Binary Cross Entropy

General code for gradient descent for logistic regression

Should work for Logistic Regression in any number of features.

Uses Binary Cross Entropy Cost function

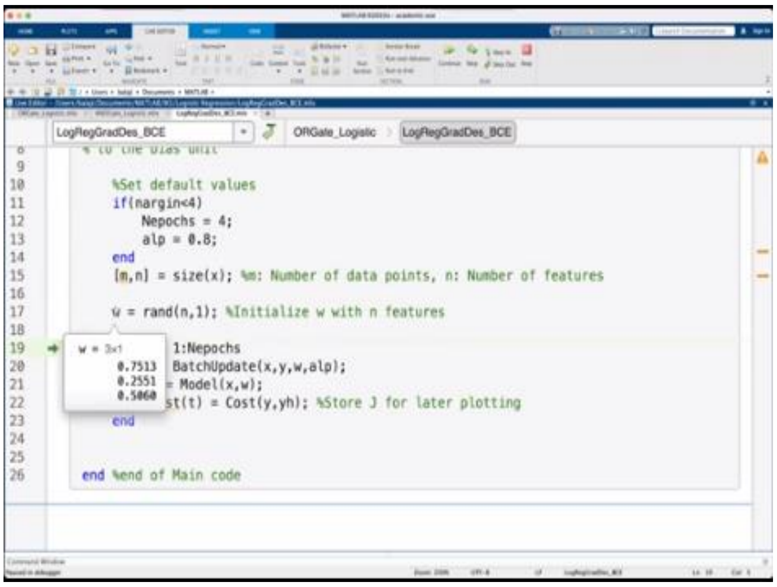
```
function [w,Jlist] = LogRegGradDes_BCE(x,y,Nepochs,alp)

%Inputs : X,y
%Output : W, the weights
%x is a matrix of size (m,n) => n features and m Data points

% Assumes that a set of ones exists in the first column ofX corresponding
% to the bias unit

    %Set default values
    if(nargin<4)
        Nepochs = 4;
        alp = 0.8;
    end
```

So, what I will do now is actually I will step into this code to the logistic regression gradient descent code. Just give me a second, it does not seem to be loading properly. So let us see if we can make it load properly here. So, I managed to load this code. So here you notice, this is basically a gradient descent code for logistic regression. I have assumed the binary cross entropy loss function.

Now the code is written a little bit generally. So, it should actually work for if you try to use it for any number of features. Here, I am going to show you just for two features x_1 and x_2. But you use some other problem with large number of features. So regardless of what that problem is, this code should work because it has been written there.

The assumption is very simple, that you have some input x, you have some output y with some labels and w are the weights, okay? So, weights are what we are going to discover.

(Refer Slide Time: 10:25)



LogRegGradDes_BCE | ORGate_Logistic > LogRegGradDes_BCE

```
    % to the bias unit

        %Set default values
        if(nargin<4)
            Nepochs = 4;
            alp = 0.8;
        end
        [m,n] = size(x); %m: Number of data points, n: Number of features

        w = rand(n,1); %Initialize w with n features

                  1:Nepochs
                  BatchUpdate(x,y,w,alp);
                  = Model(x,w);
                  st(t) = Cost(y,yh); %Store J for later plotting
        end


    end %end of Main code
```

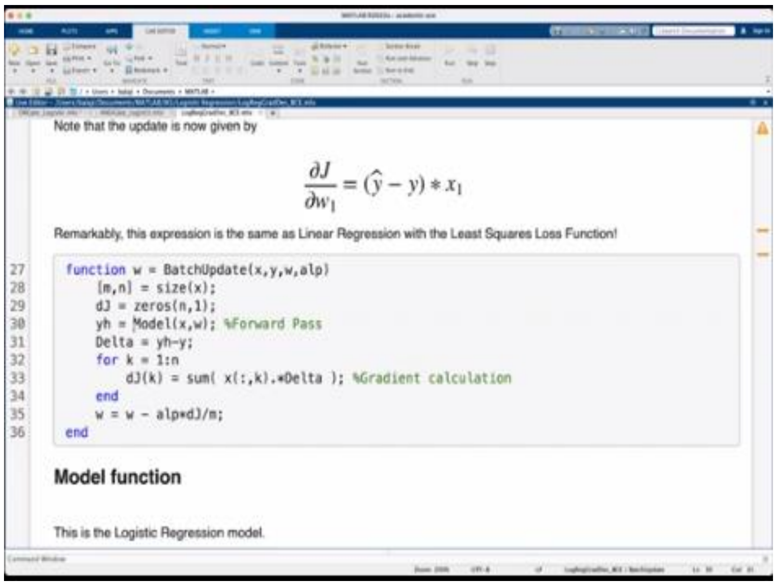
```
w = 3x1
    0.7513
    0.2551
    0.5060
```

Now I have set some default values here in case I have not given the number of epochs, it will set it to four, alpha would be set to default at 2.8, you need not look at all these. This one reads the data and just finds out the number of points and number of features. So, for example, we now move once you read here, let me step through this, you will see that it automatically knows that there are four data points, and that there are three features.

Why three features, there is the bias. And then there are the two extra features. So, when I do this, you can see that it just initializes these three weights. The first weight 0.7513 is the bias, it is initialized randomly. Then you have 0.2551 that is w_1. So, the first, w_0 is initialized to 0.7513, w_1 to 0.2551 and w_2 to 0.5060. We had already declared 20,000 epochs. So, epochs are 20,000.

And I am just going to cycle through that. And we are doing a batch update. Sorry, I said I would do a stochastic gradient update, but looks like I am actually going to show you the batch update code. So here once again, we have the opportunity to go to the batch update code. So, this is part of this main code only.

(Refer Slide Time: 11:54)


Note that the update is now given by

$$\frac{\partial J}{\partial w_1} = (\hat{y} - y) * x_1$$

Remarkably, this expression is the same as Linear Regression with the Least Squares Loss Function!

```
function w = BatchUpdate(x,y,w,alp)
    [m,n] = size(x);
    dJ = zeros(n,1);
    yh = Model(x,w); %Forward Pass
    Delta = yh-y;
    for k = 1:n
        dJ(k) = sum( x(:,k).*Delta ); %Gradient calculation
    end
    w = w - alp*dJ/m;
end
```

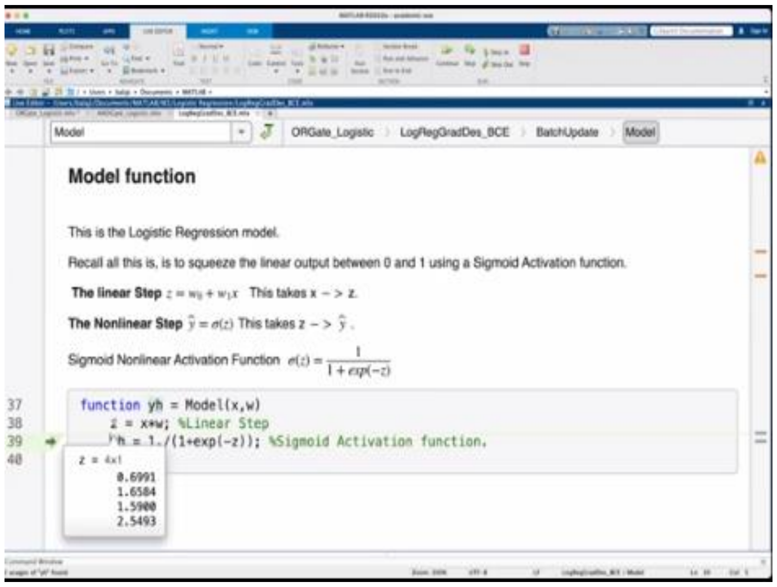
Model function

This is the Logistic Regression model.

I have written the batch update code. Once again, all this batch update does is takes the data, the data is visible on your screen. All the biases, the bias terms 1,1,1,1, then the original data 0 0, 0 1, 1 0, 1 1. The Ground Truth for that 0,1,1,1 the current guess for w. And you can also see the current learning rate alpha. Now what does this do? So, notice this portion, this is pretty universal.

You will be surprised that more or less, except for maybe this term, this is how even complicated neural networks would look, okay. So, all you would do is have a forward pass. So, this here, yh is basically the forward pass. So, the forward pass is this. What you do is you go to the model.

(Refer Slide Time: 12:52)


Model function

This is the Logistic Regression model.

Recall all this is, is to squeeze the linear output between 0 and 1 using a Sigmoid Activation function.

The linear Step $z = w_0 + w_1 x$ This takes x -> z.

The Nonlinear Step $\hat{y} = \sigma(z)$ This takes z -> $\hat{y}$.

Sigmoid Nonlinear Activation Function $\sigma(z) = \frac{1}{1+exp(-z)}$

```
function yh = Model(x,w)
    z = x*w; %Linear Step
    yh = 1./(1+exp(-z)); %Sigmoid Activation function.
```

```
z = 4x1
    0.6991
    1.6584
    1.5900
    2.5493
```

And you can see the model here, the logistic regression model. You have the linear step, and then you have the nonlinear step. So just a combination of these two achieves the forward model. If we see what happens here? We can quickly check what happens when we look at the forward model, you see the whole x has come here. You have our current guesses for w. So, you basically do x multiplied by w.

So, what that will give you is z okay, so you get you can see four different z's. What does this correspond to? Each one of them corresponds to one forward pass through one example. So, the first z corresponds to the 0 0 data point, the second z corresponds to the 0 1 data point, then 1 0, and finally 1 1. So, we are doing four different forward passes. If you write it in this way, you can do it at one shot.

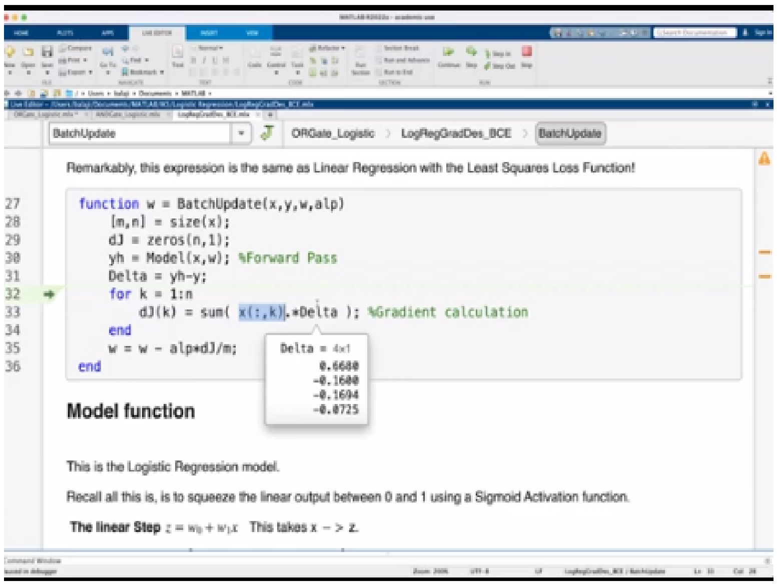
This is just a vector implementation or a one-shot implementation of this entire batch. Now once we do that, all we are finding out is $\hat{y}$, which is sigmoid activation function. So, I will do that too. So, when I do that, for each z, I get a $\hat{y}$. So, what really, we are doing is, instead of putting a for loop outside and saying for data set going from 1 to 4, do four forward passes. I have just done it as a vector.

When I show you, the neural network using backprop for XOR, I will actually show you a step-by-step implementation of this. But this is just a simple, elegant way to do it. And really, this is how practical neural network ports are written. But it is easier to see when we see the simple case of a logistic regression code. So here back to the original code. We now have $\hat{y}$ calculated for all four passes.

So, notice what has been done. We had x which was the original data, we had y which is also original data, we have a guess for w. Now this guess for w mixes with the guess for x and you actually get what the forward passes, you actually get a value for $\hat{y}$ for each value of x, okay. So please remember our original picture. Given a value of x, given a value of w you get a value of $\hat{y}$.

So, you can see that right on your screen. Now delta of course, is the error.

(Refer Slide Time: 15:27)



Remarkably, this expression is the same as Linear Regression with the Least Squares Loss Function!

```
function w = BatchUpdate(x,y,w,alp)
    [m,n] = size(x);
    dJ = zeros(n,1);
    yh = Model(x,w); %Forward Pass
    Delta = yh-y;
    for k = 1:n
        dJ(k) = sum( x(:,k).*Delta ); %Gradient calculation
    end
    w = w - alp*dJ/m;
end
```

```
Delta = 4x1
     0.6680
    -0.1600
    -0.1694
    -0.0725
```

Model function

This is the Logistic Regression model.

Recall all this is, is to squeeze the linear output between 0 and 1 using a Sigmoid Activation function.

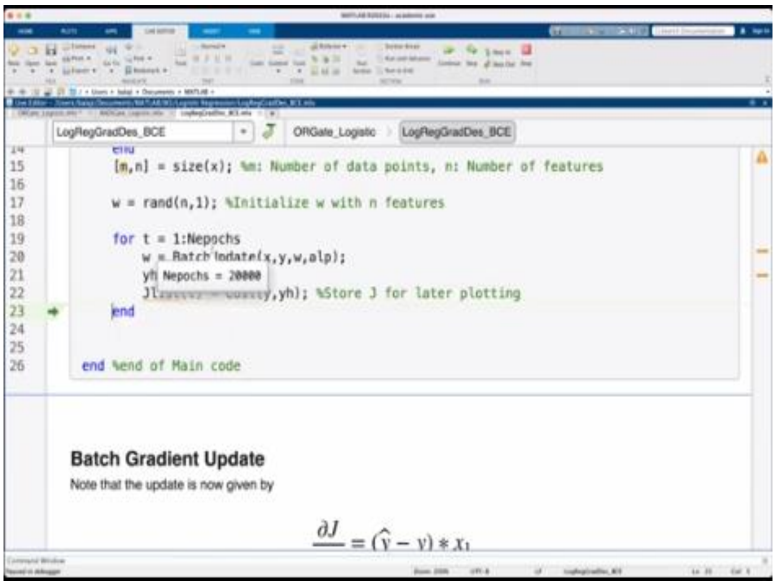
The linear Step $z = w_0 + w_1 x$ This takes x − > z.

So, the error in the output, you can see. You know, you have a whole bunch of errors. Now, the advantage is, we knew the gradient calculation was if you remember $(\hat{y} - y)\mathrm{x}_0$, I had shown you this in the initial theoretical calculation, and I had also shown you this in week nine. Then $(\hat{y} - y)\mathrm{x}_1$ is $\frac{\partial \mathrm{J}}{\partial \mathrm{w}_1}$. So, you can see the original guess for del J.

dJ here simply represents the three gradients $\frac{\partial \mathrm{J}}{\partial \mathrm{w}_0}$, $\frac{\partial \mathrm{J}}{\partial \mathrm{w}_1}$ and $\frac{\partial \mathrm{J}}{\partial \mathrm{w}_2}$. So, you have these three calculated here. And of course, we are summing over all four examples. The reason I have written it like this is just to ensure all the matrix multiplications are covered. So, you have four different errors for these four different forward passes, and then you multiply with the appropriate x's.

So, once you do that, so let me do that here and come till here. You now have dJ. This is the updated three different gradients that you have. So, 0.2661, -0.2419 and -0.2325. And once you do that, multiply by the appropriate alpha, you get the new w. So, the old w was something, you remember 0.6771 or something of that sort. Now you have an entirely new w. And once we go back, one epoch has passed, okay.

(Refer Slide Time: 17:07)



```
    end
    [m,n] = size(x); %m: Number of data points, n: Number of features

    w = rand(n,1); %Initialize w with n features

    for t = 1:Nepochs
        w = BatchUpdate(x,y,w,alp);
        yh Nepochs = 20000
        Jl[illegible],yh); %Store J for later plotting
    end


end %end of Main code
```

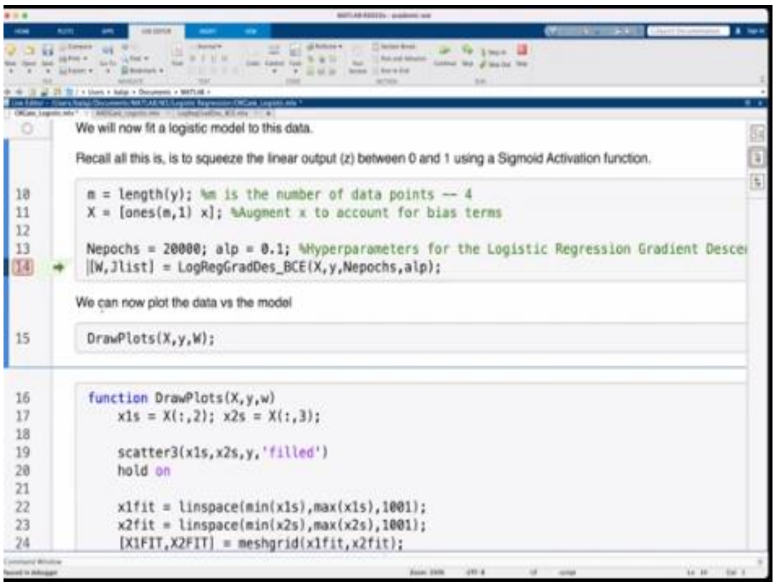
Batch Gradient Update

Note that the update is now given by

$$\frac{\partial J}{} = (\hat{y} - y) * x_1$$

So, I step here. I can now find out what the new y hat is at this new value of w. And I can now find out what the cost is. So, if I look at it, my new cost is 0.3833. So, this is the cost for guessing w to be whatever our current value, older value was. Now of course, we have to now repeat this 20,000 times, okay? So, if we keep on repeating this 20,000 times, we will basically get a full update of this logistic regression cost function.

(Refer Slide Time: 17:48)



We will now fit a logistic model to this data.

Recall all this is, is to squeeze the linear output (z) between 0 and 1 using a Sigmoid Activation function.

```
m = length(y); %m is the number of data points -- 4
X = [ones(m,1) x]; %Augment x to account for bias terms

Nepochs = 20000; alp = 0.1; %Hyperparameters for the Logistic Regression Gradient Descen
[W,Jlist] = LogRegGradDes_BCE(X,y,Nepochs,alp);
```

We can now plot the data vs the model

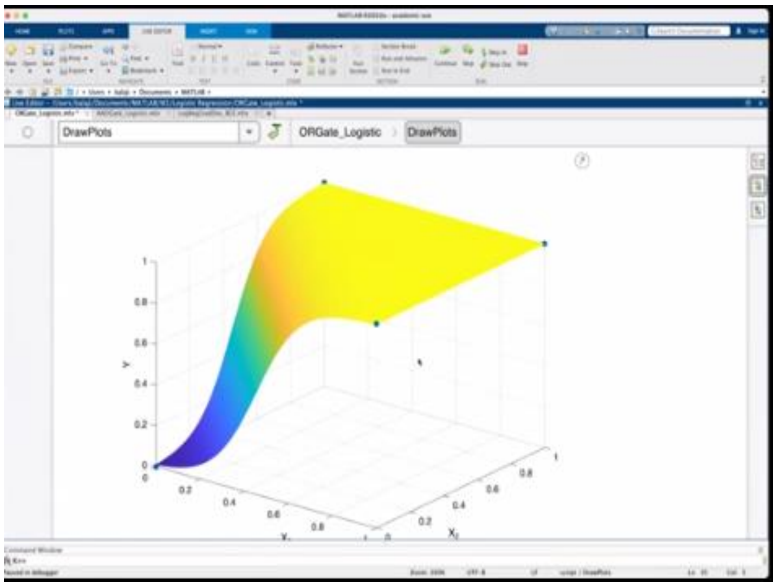
```
DrawPlots(X,y,W);
```

```
function DrawPlots(X,y,w)
    x1s = X(:,2); x2s = X(:,3);

    scatter3(x1s,x2s,y,'filled')
    hold on

    x1fit = linspace(min(x1s),max(x1s),1001);
    x2fit = linspace(min(x2s),max(x2s),1001);
    [X1FIT,X2FIT] = meshgrid(x1fit,x2fit);
```

So, we can come back here and try to go back to here. So, when I do this, I will come back here now. When I come here actually the number of 20,000 epochs have passed. You can look J list will be a long vector 20,000. So, you can see slowly your costs are actually reducing, okay? So let me see if I have plot of the J list. I do not seem to have that here.

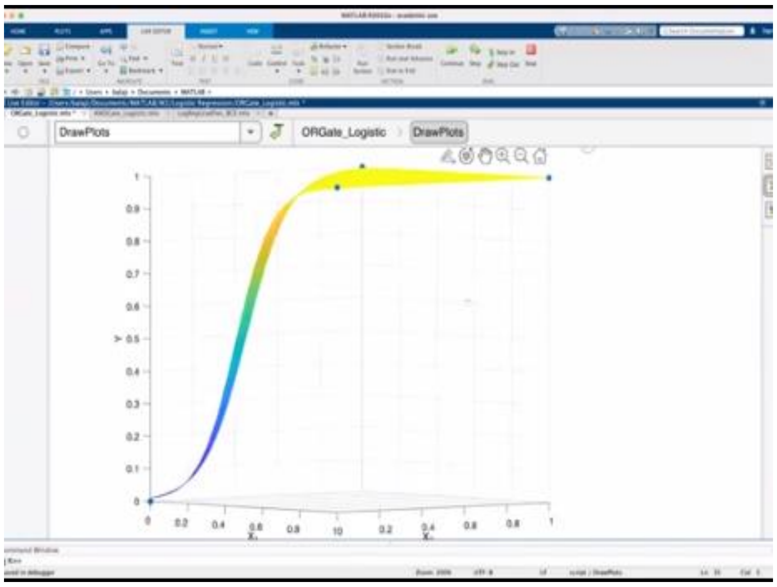
But I can plot that before anything else, a little bit here. Let us see if I can plot that. I will skip that. Let me actually come back here and show you something a little bit more interesting. So, if I come here, and we try to plot what this entire function looks like. Okay, so I will come here.

(Refer Slide Time: 18:53)



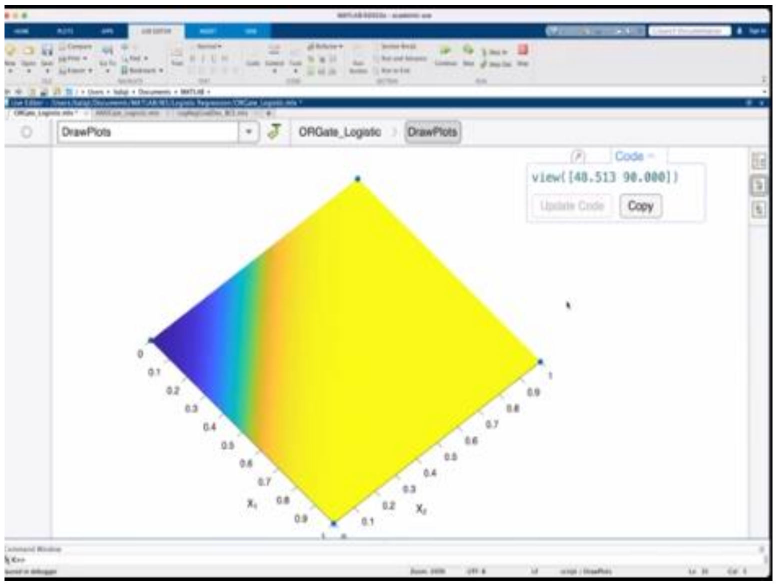
So, you can see the 3d plot of our y hat. So, notice these four points. The point at 0 0, the point at 0 1, the point at 1 0 and the point at 1 1, this is what we want to match, okay? The function that it is predicting is somewhere like this. So, it is actually predicting the function, please notice that our neural network or in this case, the logistic model, is basically simply a function. We can look at it from various angles.

(Refer Slide Time: 19:27)



If you look at it from here, it more or less looks simply like the sigmoid okay.

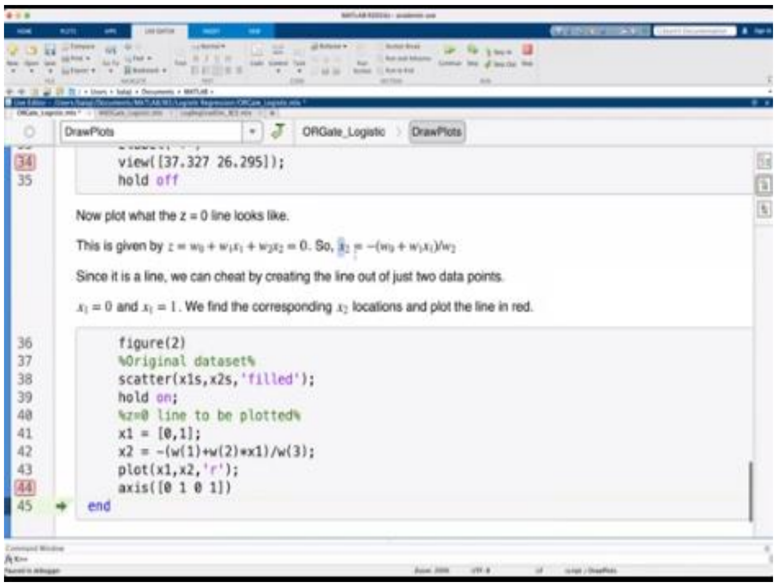
(Refer Slide Time: 19:34)



```
view([48.513 90.000])
```

So, but at from a different angle from top when you look at this, you will notice that there is actually a sharp line here, which classifies it as either belonging to one side either belonging to 0 or belonging to 1. In fact, if you wish, we can now plot what that line is. So let me just go here and continue.

(Refer Slide Time: 19:58)



```
    view([37.327 26.295]);
    hold off
```

Now plot what the z = 0 line looks like.

This is given by $z = w_0 + w_1x_1 + w_2x_2 = 0$. So, $x_2 = -(w_0 + w_1x_1)/w_2$

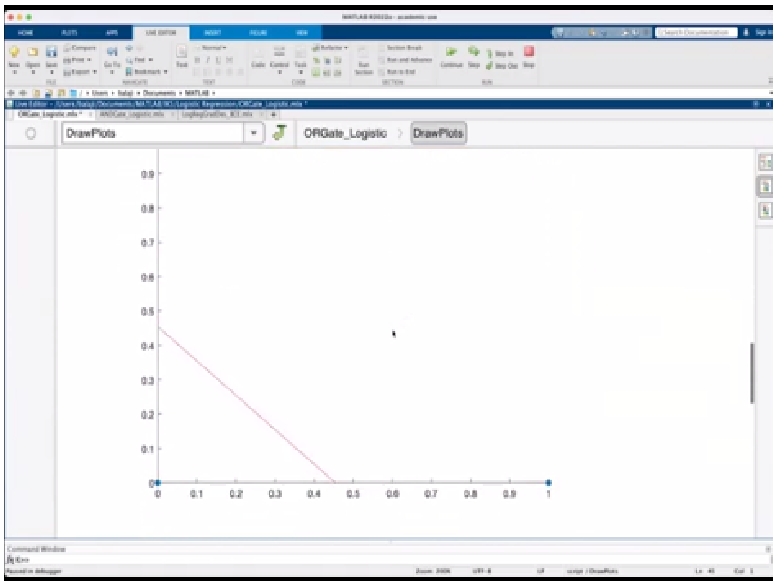
Since it is a line, we can cheat by creating the line out of just two data points.

$x_1 = 0$ and $x_1 = 1$. We find the corresponding x_2 locations and plot the line in red.

```
    figure(2)
    %Original dataset%
    scatter(x1s,x2s,'filled');
    hold on;
    %z=0 line to be plotted%
    x1 = [0,1];
    x2 = -(w(1)+w(2)*x1)/w(3);
    plot(x1,x2,'r');
    axis([0 1 0 1])
end
```

I will come here I can continue here and take a step. And let us see if that is plotted. So yes. So, this of course, the top here was the 3d plot.

(Refer Slide Time: 20:17)

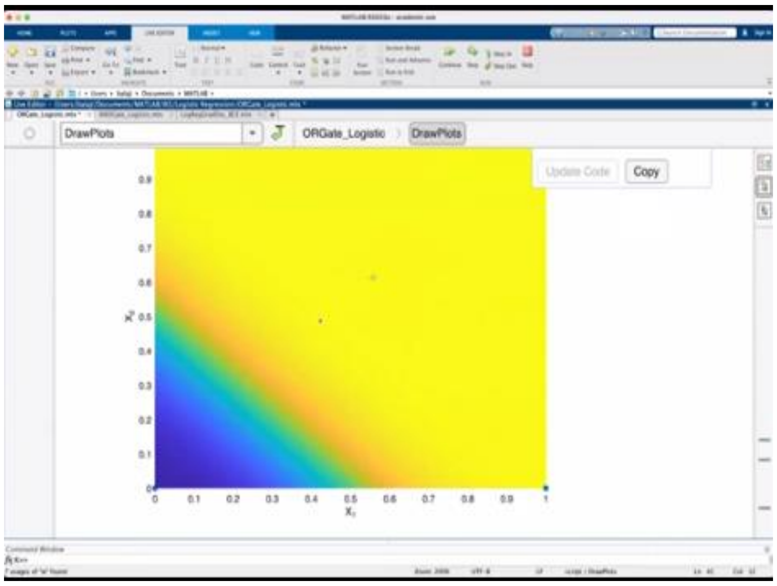


And this is the 2d plot. This red line that you see on your screens is the classifying line, the line for z equal to 0. The z equal to 0 line is basically plotted. You can see y equal to – w 0 plus w 1 x 1 by w 2. So, you can basically just grade this this line very easily. So that is what I did. I have, you can go through the code and figure out how I drew this line. But basically, it is a line the values of which are given here.

We can actually check the values of w. So, w you can see is approximately w_0 is -5, w_2 is 10 and w_3 is 10. So, you can think of this, if you take a ratio as -1, 2 and 2, which is pretty close to the values that we had given in the previous video. So, you can see this nice logistic regression. We started with some random guess for w when you go back to this code and check, we had actually given a random guess for w.

And from that random guess for w, we got these converged values and these nice plots for OR gate. So, the way this OR gate was achieved, I hope you can notice I had again shown you the sort of top view of this, I do not know if I can achieve that once more.

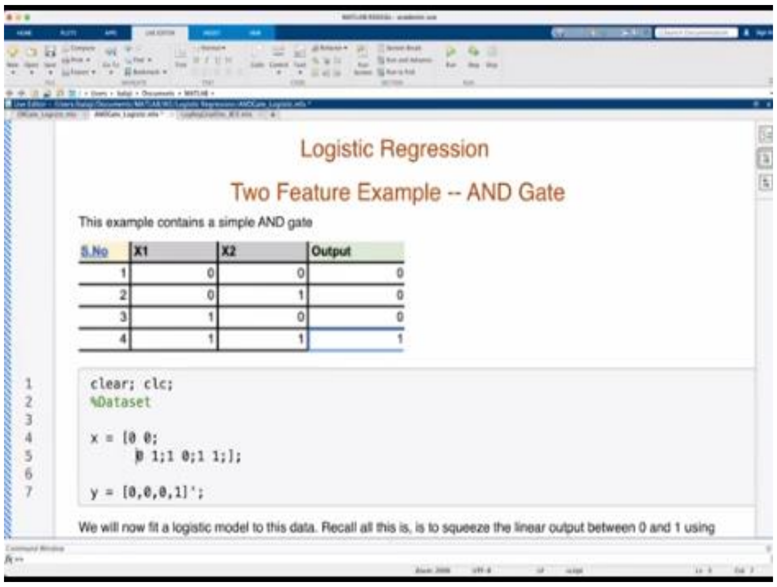
(Refer Slide Time: 21:36)



Yeah, here it is. So, this top view from here x_1 and x_2 shown here, you can see this nice, straight line, which changes from you can see from dark blue to dark yellow. As further and further you move away from this line, it gets more and more sure that it is 1. Further and further, you move towards 0, it gets more and more sure that it is 0.

So, this is an example of simply using logistic regression for the OR gate. We can repeat this exercise for AND gate as well.

(Refer Slide Time: 22:11)



Logistic Regression

Two Feature Example -- AND Gate

This example contains a simple AND gate

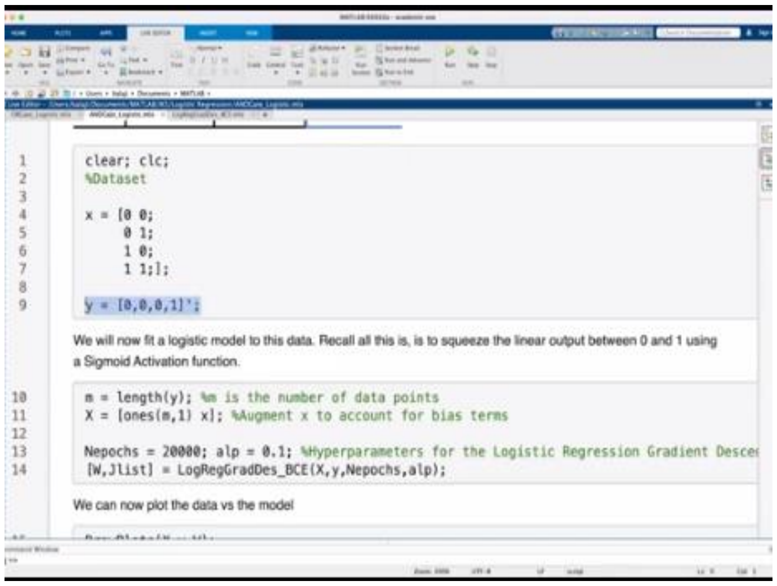
S.No	X1	X2	Output
1	0	0	0
2	0	1	0
3	1	0	0
4	1	1	1

```
clear; clc;
%Dataset

x = [0 0;
     0 1;1 0;1 1;];

y = [0,0,0,1]';
```

We will now fit a logistic model to this data. Recall all this is, is to squeeze the linear output between 0 and 1 using

And really nothing changes other than the data. So, we can see that here AND gate. Same except now, instead of the 0 1 1 1, which was the case for OR gate, it is now 0 0 0 1.

(Refer Slide Time: 22:33)



```
clear; clc;
%Dataset

x = [0 0;
     0 1;
     1 0;
     1 1;];

y = [0,0,0,1]';
```

We will now fit a logistic model to this data. Recall all this is, is to squeeze the linear output between 0 and 1 using a Sigmoid Activation function.

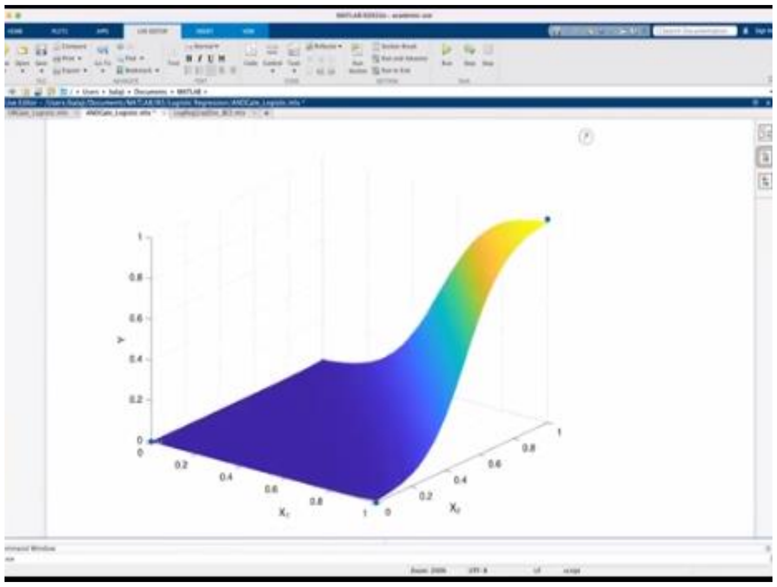
```
m = length(y); %m is the number of data points
X = [ones(m,1) x]; %Augment x to account for bias terms

Nepochs = 20000; alp = 0.1; %Hyperparameters for the Logistic Regression Gradient Desce
[W,Jlist] = LogRegGradDes_BCE(X,y,Nepochs,alp);
```

We can now plot the data vs the model

So, you will see that literally, almost nothing else has changed. The only thing that has changed is the data. So, if you compare this code here, and this code here, you will see I have changed nothing other than y. And the rest of it is probably verbatim exactly the same. I did nothing other than use the same logistic regression gradient descent. I am not going to change the code also.

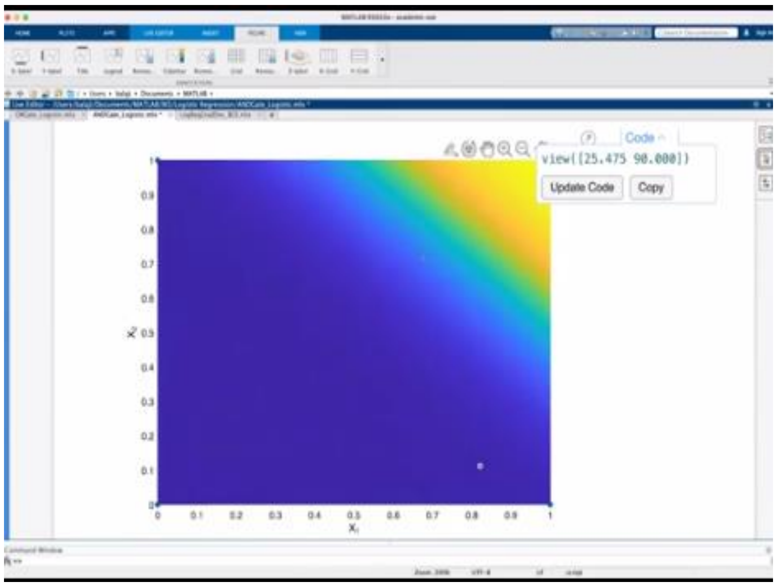
So, this is the power of the approach. The power of the approach is, any data you give, you basically are just playing with the data set, in order to create a model. Of course, the weight is going to change, the model is entirely different. So, if we run this, let us try running this and see. I hope I have not made any errors in the interim.

(Refer Slide Time: 23:26)



So now you can see. So, it has figured out an entirely different model, a different set of weights. So, 0 0 0 for these three values, and it is now shot up to 1. Again, we can probably try rotating this. Yes.

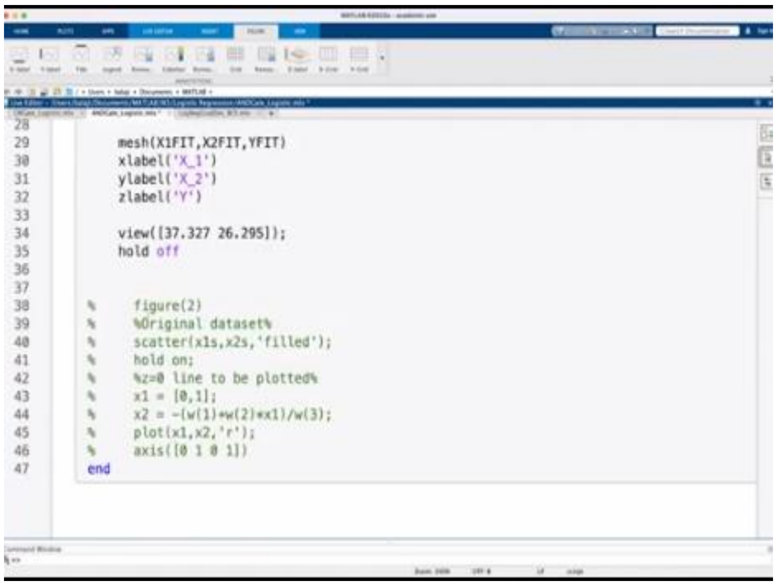
(Refer Slide Time: 23:45)



```
view([25.475 90.000])
```

So, if we try rotating this, you will see you have a model, which is sort of diametrically the opposite of what we had earlier. It is shorter about 0 here and shorter about 1 here as you go further and further away.

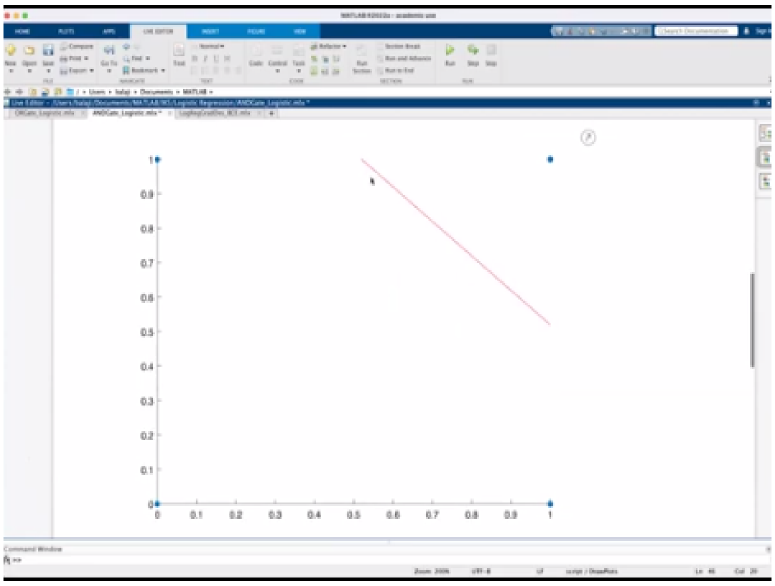
(Refer Slide Time: 24:05)



```

    mesh(X1FIT,X2FIT,YFIT)
    xlabel('X_1')
    ylabel('X_2')
    zlabel('Y')

    view([37.327 26.295]);
    hold off


%     figure(2)
%     %Original dataset%
%     scatter(x1s,x2s,'filled');
%     hold on;
%     %z=0 line to be plotted%
%     x1 = [0,1];
%     x2 = -(w(1)+w(2)*x1)/w(3);
%     plot(x1,x2,'r');
%     axis([0 1 0 1])
  end
```

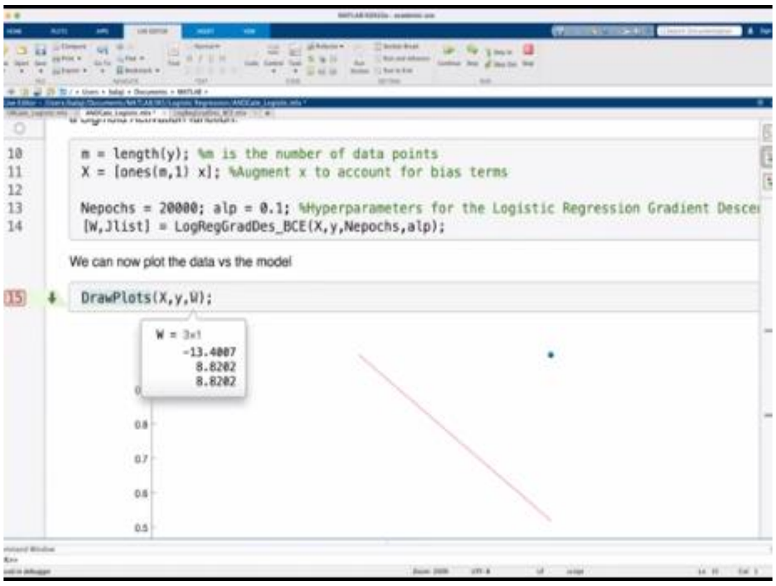
And I do not know if I have drawn the z equal to one line. So, I could probably do this. And we can run this code again and can see whether yeah, so you can see this.

(Refer Slide Time: 24:18)



So, the classifying line has now, is the red line, it has now moved. That is because w_0, w_1, w_2 or w_1, w_2, w_3 have changed. So, the values of w, let us keep those values here. Let me just check and run this once more.

(Refer Slide Time: 24:40)



```
m = length(y); %m is the number of data points
X = [ones(m,1) x]; %Augment x to account for bias terms

Nepochs = 20000; alp = 0.1; %Hyperparameters for the Logistic Regression Gradient Desce
[W,Jlist] = LogRegGradDes_BCE(X,y,Nepochs,alp);
```

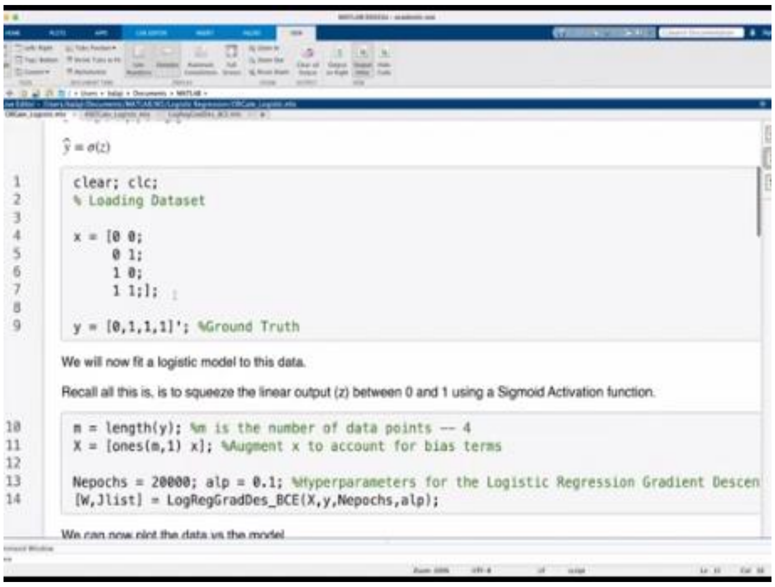
We can now plot the data vs the model

```
DrawPlots(X,y,W);
```

```
W = 3x1
   -13.4007
     8.8202
     8.8202
```

So, you can see it is now -13, 8.8 and 8.8. So, it has chosen something like -3 by 2 and 1 by 2, 1 by 2 or something on that sort. So roughly again similar to the cases that I had shown. No this is not 1 and third, this is one, one and a half or something, okay? So, 1, 3 by 2, 3 by 2. So, it is approximately somewhere. But the remarkable thing once again is that it does find a classifying line, and it finds it simply by iterating using some blind process such as gradient descent. Now we might get excited, let me now clear all this output.

(Refer Slide Time: 25:28)



$\hat{y} = \sigma(z)$

```
clear; clc;
% Loading Dataset

x = [0 0;
     0 1;
     1 0;
     1 1;];

y = [0,1,1,1]'; %Ground Truth
```

We will now fit a logistic model to this data.

Recall all this is, is to squeeze the linear output (z) between 0 and 1 using a Sigmoid Activation function.

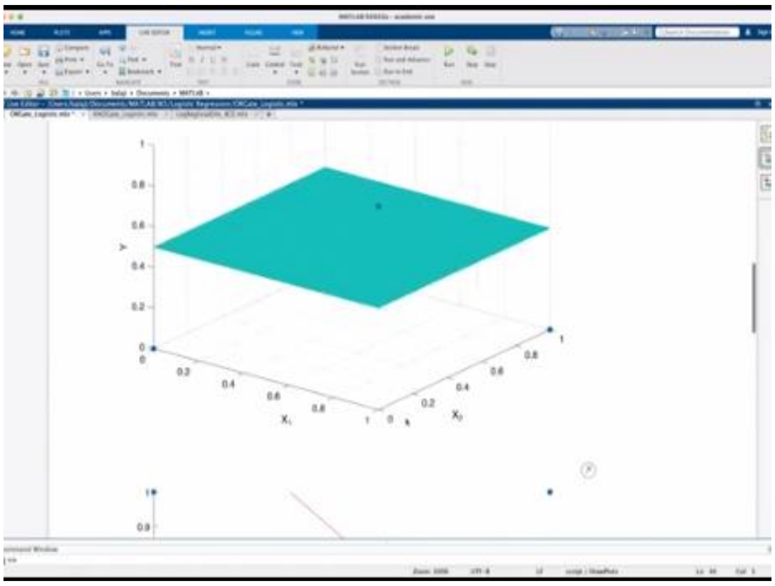
```
m = length(y); %m is the number of data points -- 4
X = [ones(m,1) x]; %Augment x to account for bias terms

Nepochs = 20000; alp = 0.1; %Hyperparameters for the Logistic Regression Gradient Descen
[W,Jlist] = LogRegGradDes_BCE(X,y,Nepochs,alp);
```

We can now plot the data vs the model

And suppose I change this to let us say, let us go back to the OR gate example, just in case because this is close, and I make this something like XOR. So, I put the XOR gate, which is or I think actually I should switch this to 1 0 0 1. It does not matter. 0 1, 1 0. Yeah 0 1, 1 0 would be XOR. So, suppose I run this. So, the question really is, does this converge or does this not converge? So, when we run this, let me just go back, run this case.

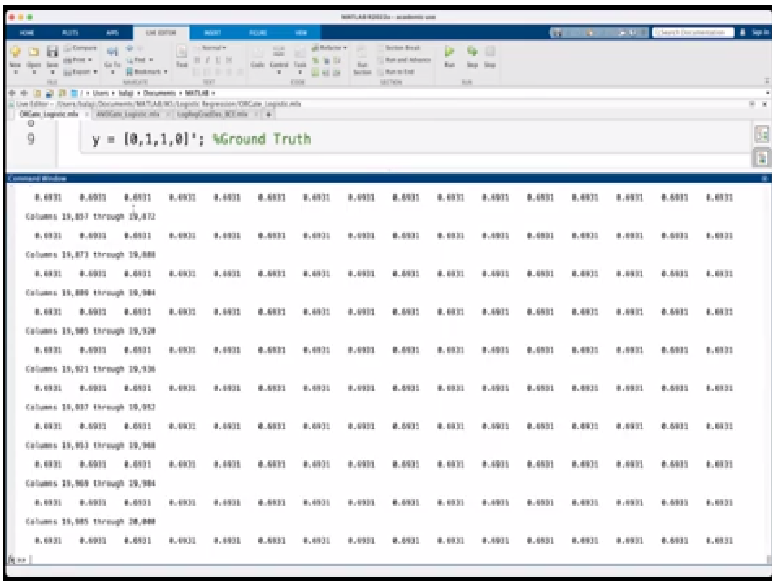
You will actually find I will continue and run the whole case. Let us run this tool.

(Refer Slide Time: 26:24)



That it actually does not classify well. You can see the model; the model is a really bad model. It has made some arbitrary classification line, which is not the correct line at all, okay?

(Refer Slide Time: 26:45)

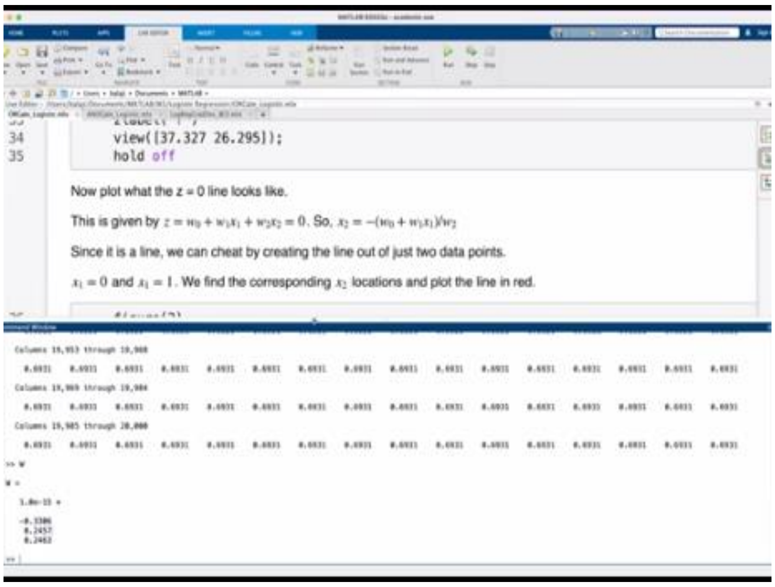


```
y = [0,1,1,0]'; %Ground Truth
```

```
    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,857 through 19,872

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,873 through 19,888

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,889 through 19,904

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,905 through 19,920

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,921 through 19,936

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,937 through 19,952

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,953 through 19,968

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,969 through 19,984

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,985 through 20,000

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931
```

In fact, if we look at the list that we have of J, I can write this. You can see it seems to have converged, okay? But it converges right from the beginning, okay? So, it is not really moving from the beginning, because it finds out some arbitrary w's, which satisfy whatever guests it has, okay?

(Refer Slide Time: 27:04)



```
view([37.327 26.295]);
hold off
```

Now plot what the z = 0 line looks like.

This is given by $z = w_0 + w_1x_1 + w_2x_2 = 0$. So, $x_2 = -(w_0 + w_1x_1)/w_2$

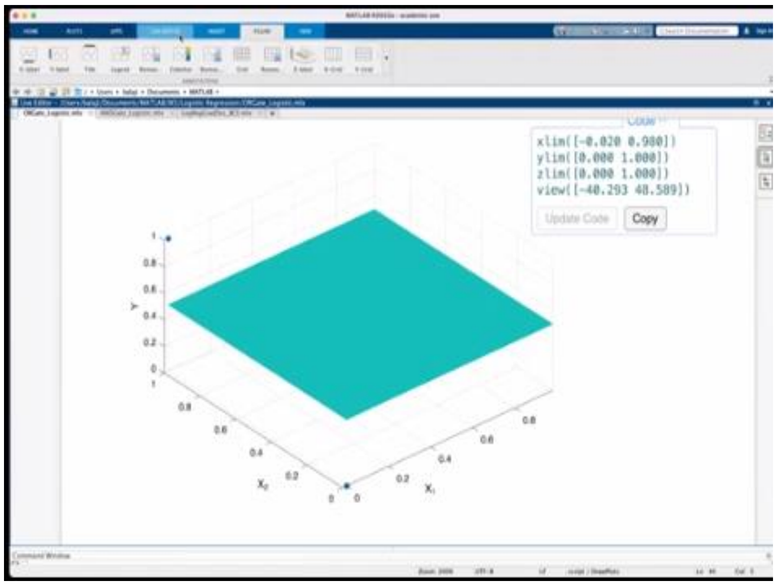
Since it is a line, we can cheat by creating the line out of just two data points.

$x_1 = 0$ and $x_1 = 1$. We find the corresponding x_2 locations and plot the line in red.

```
  Columns 19,953 through 19,968

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,969 through 19,984

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

  Columns 19,985 through 20,000

    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931    0.6931

>> w

w =

   1.0e-15 *

  -0.3306
   0.2457
   0.2462

>>
```

So, it is w is basically you see, 10 power -15, w is 000. So, the best it can do, after a first few initial guesses, it does converge, but it converges to bad values, because this is simply a bad model, okay. So, there is no good model to be found here. If you look at, let me look at a different angle, just to make this clear to you because it is a little bit difficult to see.

(Refer Slide Time: 27:00)



```
xlim([-0.020 0.980])
ylim([0.000 1.000])
zlim([0.000 1.000])
view([-40.293 48.589])
```

So, the problem here is that XOR cannot be classified using this model. So, the model converges, but it converges to the wrong value. So just because something converges does not mean that it is great. It simply means that it converges. But the original model, that model, which is there within our bands, is a bad model, okay? So that is the important lesson of running XOR through a logistic regression.

In the next video, I will actually show you how to run XOR. I will just restore this code to the original code so that when I share it with you guys, you have the right code. But in the next video, I will show you how to run XOR through a neural network. So, you actually get a proper correct prediction.

So, for example, when you come here and look at the prediction here, for the XOR that we had, let me see whether that exists here yh i's what I think I called it. I seem to have called it something else. Let us just check this back. This is important. So, I will restore this and run this once more. We will continue with this. The model here was stored in $\hat{y}$, but that model is not sitting here I suppose.

Okay sorry, it is not sitting here. But we can simply check that the Ground Truth when you set 000 is obviously not going to be great, okay? It is simply going to be sigmoid of zero every time. So, you are going to get 0.5, 0.5, 0.5, 0.5. So that is our prediction for all four cases here, which you can see here. It is just 0.5 throughout, okay?

So, you can see when you look at the left that the actual prediction, which is this green square is regardless of what value I gave for x_1 and x_2, it is 0.5 at every single case. So that is not a great model, but what we will show is, we will get the XOR prediction if we use a neural network with even one hidden layer. So, we will see that in the next video. Thank you.