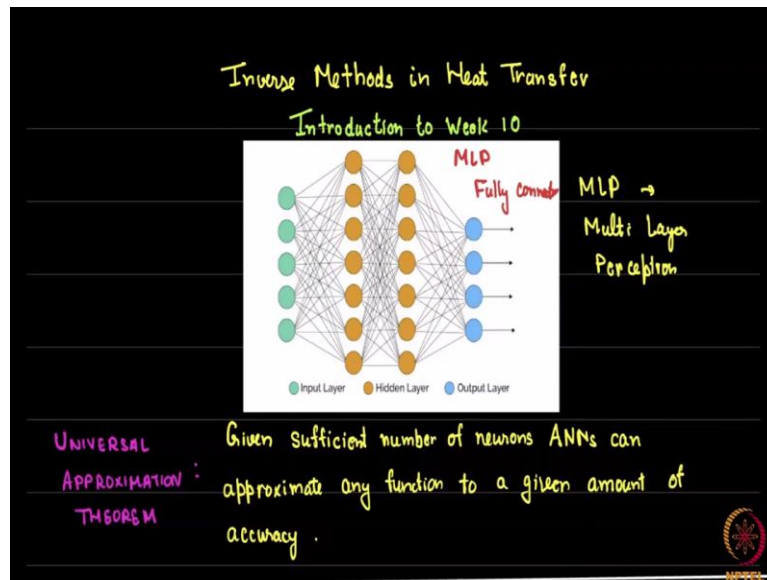


Inverse Methods in Heat Transfer
Prof: Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology – Madras

Lecture - 52
Introduction to Week 10: XOR and Deeper networks

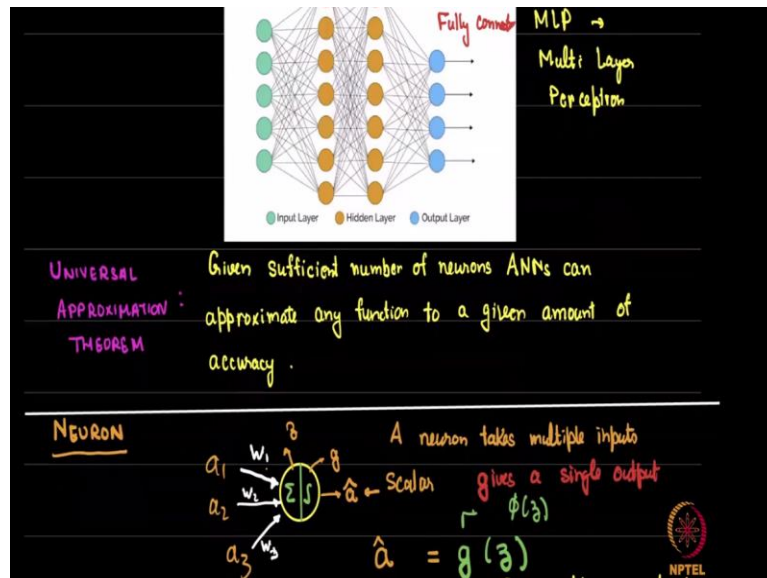
(Refer Slide Time: 00:18)



Welcome back. This is the first video of Week 10 in Inverse Methods in Heat Transfer. What I wish to do in this video is to give you a brief introduction to Week 10, but also a few additional topics are there, specifically we will be looking at the XOR gate, once again we finished with the XOR gate in the last week, but I will sort of naturally lead you to this sort of structure for neural networks that you can see here, which is one input a few hidden layers and an output layer.

This layer is called the MLP or the multi-layer perceptron. MLP stands for multi-layer perceptron. In short it basically is a whole bunch of neurons connected in lines and all these collections are full and such a connection is also called a fully connected layer. So, there are bunch of fully connected layers between the input and the hidden layer, there are every neuron in one of the layers is connected to every other layer neuron in the next layer and so on and so forth. So, this fully connected MLP is what we will be coming to sort of naturally within this video.

(Refer Slide Time: 01:49)

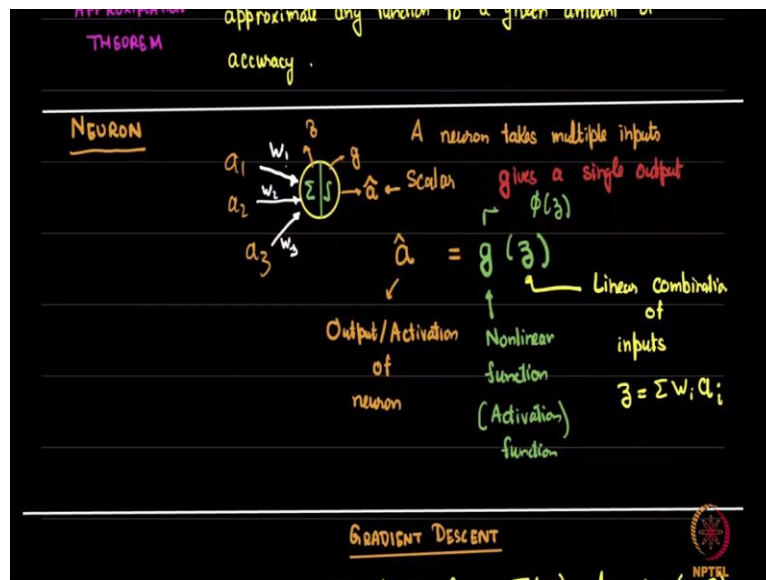


The main purpose of connecting things this way is we are able to use what is known as the universal approximation here, which I had briefly alluded to a couple of weeks ago. The basic idea behind the universal approximation theorem is that given sufficient number of neurons, if we are given sufficient number of neurons and artificial neural network of the MLP type can approximate any function to a given amount of accuracy.

So, if I give a certain amount of accuracy, you can actually find out a sufficient number of weights which will approximate this function. Now why does this come about I will briefly talk about that in this video, but the primary purpose of the week is to actually give you a full introduction to this structure called MLPs, tell you how forward propagation happens through this and also how to solve these for these weights through gradient descent.

Basically, in effect solving an inverse problem. The next week after this one we will look at how to use MLP for various problems of our interest.

(Refer Slide Time: 02:58)



Now all of this utilizes a neuron a basic structure called neuron, which I kind of introduced you to in the last week. The idea is very simple when you have any of these coming in you have a whole bunch of weights which come in here. So, you could have $w_1 w_2 w_3$ you sum those up and you run them through a non-linearity. This is the fundamental idea behind a neuron.

So, the notations we will use are a is the output of a neuron you can see here \hat{a} is the output of the neuron from the previous layer you could have a 's coming in for example a from here would be input here and a from here would be input to the next layer. So, anyway we will come to this notation once more later on this week, but one thing I would like you to remember is the same thing that we did in logistic regression.

You first have a linear combination and then you pass it through a non-linearity. So, this non-linear function is called an activation function and this function is simply a linear combination of the things that are coming in. Remember that for a neuron multiple inputs come in, but only one thing leaves out. So, we will discuss this one further as we go through this week.

(Refer Slide Time: 04:17)

neuron
function
(Activation)
 $\hat{z} = \sum w_i a_i$

function


GRADIENT DESCENT

If we are minimizing a fn of the form $J(w)$ where w (or θ) is the set of parameters $w_1, w_2 \dots, w_n$ we may do so using

$$w \leftarrow w - \alpha \nabla_w J \text{ or}$$

$$w_1 = w_1 - \alpha \frac{\partial J}{\partial w_1}; w_2 = w_2 - \alpha \frac{\partial J}{\partial w_2} \dots; w_n = w_n - \alpha \frac{\partial J}{\partial w_n}$$

Some variants: Batch, Mini-batch, SGD.



Recall also that we were being using gradient descent throughout and one of the main purposes of this week is to do these two things in a little bit more detail. The forward propagation which requires you to calculate the output of a neuron and the back calculation which requires you to calculate the gradient. Why we require the gradient of course is to update the weight within the neural network or any basic model that we have.

Remember we are doing data-based models. So, these database models have various weights and they are updated using gradient descent which requires you to find out ∇J with respect to the w .

(Refer Slide Time: 04:59)

If we are minimizing a fn of the form $J(w)$ where w (or θ) is the set of parameters $w_1, w_2 \dots, w_n$ we may do so using


$$w \leftarrow w - \alpha \nabla_w J \text{ or}$$

$$w_1 = w_1 - \alpha \frac{\partial J}{\partial w_1}; w_2 = w_2 - \alpha \frac{\partial J}{\partial w_2} \dots; w_n = w_n - \alpha \frac{\partial J}{\partial w_n}$$

Some variants: Batch, Mini-batch, SGD.

Summary of supervised Models

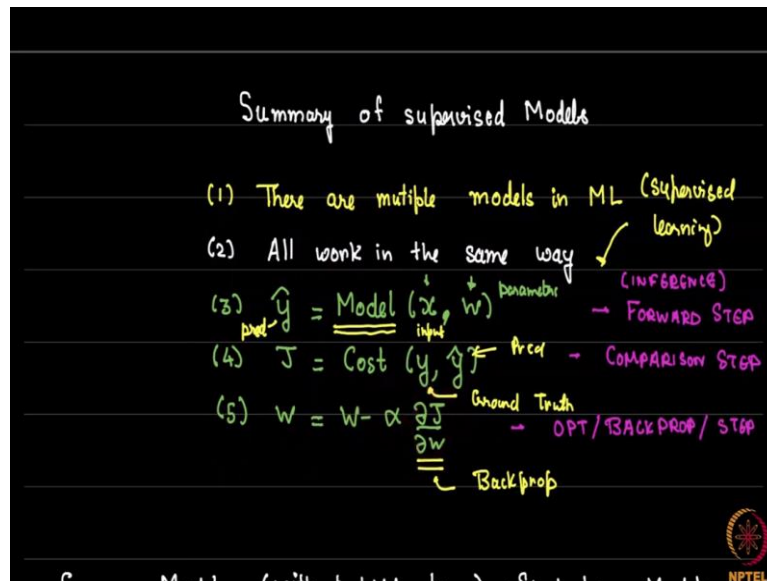
(1) There are multiple models in ML



Last week we also saw some variants of this gradient descent. The batch gradient descent which uses all the data together, mini batch which uses portions of data, in SGD or stochastic gradient

descent which uses each data individually. Now before we step into this week it is useful to see what we learned last week. The reason for this is I will now put several things that we looked at in last week in one single framework.

(Refer Slide Time: 05:25)



So, the important thing to remember is that there are various multiple models within machine learning, especially when you look at supervised learning. That is wherever we provide labels. All of these work in the same way. Now it is useful to remember why these work in the same way. Remember even some input data x and some guess for some unknown parameters w . So, w , are the parameters and x of course are the input.

You run it through some model, now some various choices of the model I am going to show you. one specific model is the neural network, but we had a linear model, we had a logistic model. We had sort of a multi class model etcetera, any of these things can be just thought of as some model then after this class you look at other machine learning algorithms, it is useful to think of all of them within the simple framework.

So, you run it through a model, once you run through the model you get a prediction. So, this prediction now of course you have your data, so you have the actual truth and you have your prediction. So, between the two there is a gap. So, these steps have specific names, this step is known as the forward step. So, you move forward and actually calculate what your prediction is or you can think of this as sometimes called the inference step.

The next step is called the comparison step. So, you made a prediction, there was some truth, compare the two, let us call the comparison step and the final step is the optimization step or what we call as the back prop step, some people call it the gradient descent step, but let us call this the back prop step. Essentially what you require where is I am kind of abusing notation because back props specifically apply here in calculating these gradients.

So, this is really speaking back prop, but I am going to sort of talk casually and some people do that and say that the weight is updated using back prop basically meaning calculating the gradient. So, we have these three steps in every single model that you can think of whether it is a simple model or a complex model. Now what we want to do is to look at all the models we have looked at so far in this course in the same light.

(Refer Slide Time: 08:29)

Backprop

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}^T \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = w_0 + w_1 x_1 + w_2 x_2$$

Some Models (without hidden layers) - Single Layer Models

| NAME | Math Model | DIAGRAM | DATA REPRESENTATION | COST |
|--------------------------------|---|---------|---|--|
| Linear Regression | $\hat{y} = W^T x$ Ex: $w_0 + w_1 x_1 + w_2 x_2$ Features | | Good idea to scale input data $\rightarrow [0, 1]$ NNs \rightarrow Batch norm | $J = \frac{1}{2} (y - \hat{y})^2$ Least Square Cost |
| Logistic Classification Binary | $\hat{y} = \sigma(W^T x)$ $= \sigma\left(\frac{z}{3}\right)$ | | y is $(0, 1]$ \rightarrow Scalar Binary $0 \leq \hat{y} \leq 1 \rightarrow$ Model Prob | $-y \ln \hat{y} - (1-y) \ln (1-\hat{y})$ BCE |

Some of the models which we have used so far without using any hidden layers let us call them single layer models. So, a single layer model will function like this there will be an input, there will be an output and there will be no intervening hidden calculation going on. So, the simplest model we looked at of this form is the linear regression. what was the model?

The model was here $\hat{y} = W^T X$. So, this x can actually be a matrix as I have shown later, for example, your x could have x_1 and x_2 . In that case if you have w is let us say w_0, w_1, w_2 and x is let us say x_0 which is just 1, x_1 and x_2 we have seen this a few times. So, this gives you,

$$w^T x = w_0 * 1 + w_1 x_1 + w_2 x_2$$

So, that is what is written here in the linear model I have written this as the mathematical expression.

So, what is written up here if you are not able to see it clearly is the mathematical or the analytical or the algebraic model. So, for example the linear regression model or the linear model is simply $w_0 + w_1x_1 + w_2x_2$. These x_1 and x_2 are called features or attributes and I will come back with this point later on in this video also. When we represent the same thing diagrammatically you have one x_1x_2 you sum the three together multiplied by $w_0 w_1 w_2$ and you get \hat{y} we saw this in last time.

Now when it comes to the data representation as I told you in previous videos it is a good idea typically to scale these input data. that is you have x_1 and x_2 suppose you have temperature going from 10 to 150 it is generally a good idea for gradient descent to work to rescale it. so that you write it in terms of theta some normalized coordinates so that it scales between 0 and 1.

I did not emphasize this too much but it is a good idea. In neural networks there is an equivalent which is called the batch norm, we are going to ignore that as this course is a fairly elementary sort of portion of our entire course. So, I am going to skip this, but just for your knowledge and neural networks this is known as the batch norm, the generalization of this is known as the batch norm.

The final thing is the cost function. The cost function is J is simply a least square cost function, this is what we use typically for linear regression. So, this is called the least square cost, what happened when we looked at the logistic function. So, the logistic function is generally used for classification and typically binary classification and there we had y -hat equal to some g of z where z is again w transpose x same as before.

But we use the sigmoid as a non-linearity. So, we represented it this way to sum these up and then run them through this sigmoid and then you get an output, you see that there is still nothing in between these two. Nonetheless in this case you have two things going on you have a linear function activation and then you have a non-linear activation through the sigmoid function. Now what was the data representation y was either 0 or 1.

This either basically it was a scalar and it was binary and the output \hat{y} was between 0 and 1 again, but it was not discrete it was not binary it is a real number and it represented the probability. Now cost function if you remember was the BCE or the binary cross entropy loss function,

$$J = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$$

we did this last week.

(Refer Slide Time: 12:36)

The image shows handwritten notes on a blackboard background. The notes are organized into columns and rows. The top row is labeled 'Classification Binary' and contains the formula $J = -y \ln \hat{y} - (1 - y) \ln(1 - \hat{y})$. The middle row is labeled '(Multi class) $K > 2$ ' and contains the formula $\hat{y} = \text{softmax}(z)$ and $z = w^T x$. To the right of this is a diagram of a neural network with multiple input nodes connected to multiple output nodes. The notes also mention 'y is ONE-HOT' and provide an example of a probability vector $[0.2, 0.7, 0.1]$. The bottom of the slide has the text 'Topics in Week 10' and the NPTEL logo.

The final example we did was for multi class classification. Notice once again z is w transpose x remains the same still the linear activation and what you have at the end of course is a soft max. So, I had given you reasons for why we use soft max in order to normalize properly in the last one. What does it look like instead of having only one output you could have multiple outputs let us say it is a three-class classification problem.

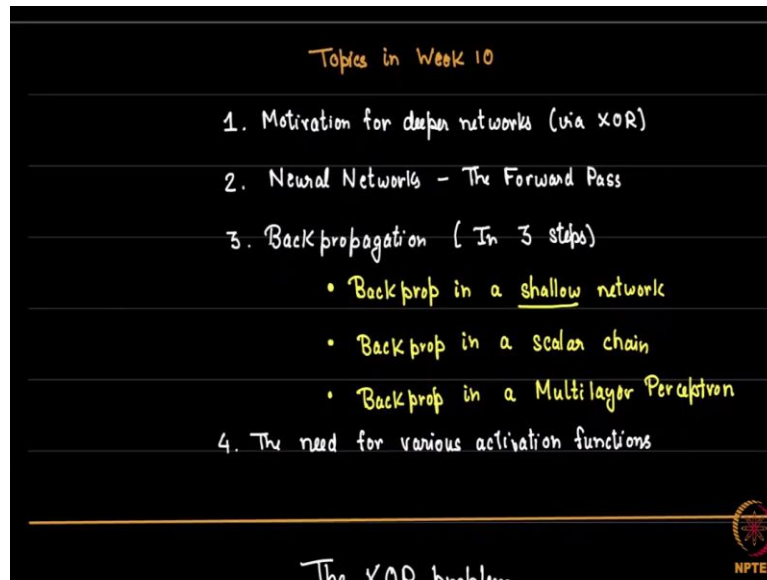
Once again, every neuron in the input is connected to every other neuron is an output and you get a \hat{y} which is a vector. The representation of y was a one hot vector and \hat{y} basically representing the probability that it belongs to class 1, class 2, class 3 etcetera. So, you would have these three sums up to one and we have the categorical cross entropy cross function here which was,

$$J = - \sum y_i \ln \hat{y}_i$$

I had also talked about the connection between the two.

Now what is the use of reviewing this? The reason is that we are going to try to integrate all these and as well as the lessons that we learned while finding out the gradients for this into this one single picture. So, the purpose of this week is to sort of the grand unification of all the methods that we have seen so far into one single thing which is a neural network.

(Refer Slide Time: 14:00)



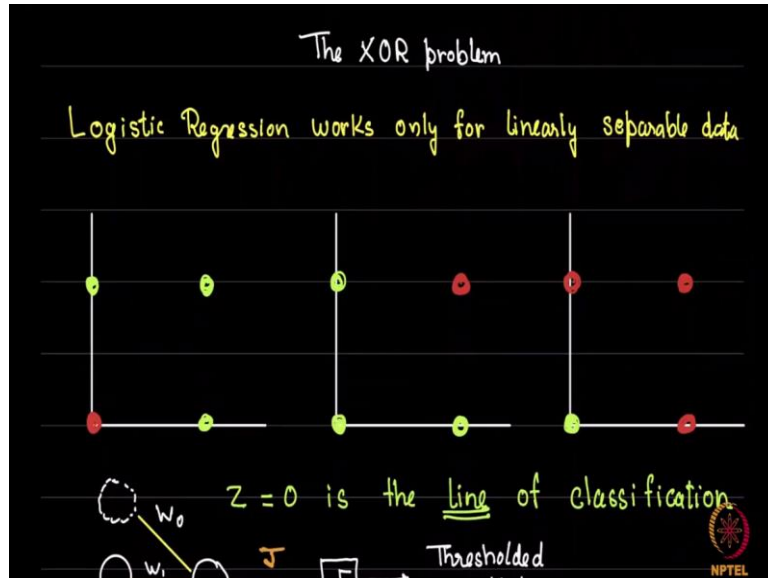
So, the topics that we are going to cover this video the current video that we have after the short introduction, I am going to cover the motivation for deeper networks via XOR. I will briefly refer to it in the last video also, but I am going to dig a little bit deeper in this video. Next, we will look at the forward pass that is how do you start from the input and go to the output for a neural network.

Then we will do the remember the comparison step is trivial, but the back propagation step is non-trivial. Typically, this takes a lot of time for students to understand. So, I am split it into three different types of networks. One is the kind of networks that we just did which are lots of neurons, but input and output are immediately connected. So, this is a shallow network, so that is what we will do first.

We will find out how to find gradients for all the w in such a situation. The next one is what I call a scalar chain, it is a deep network, but there is no width at all. So, there is only a single neuron in every layer and we just go through that deep network and we will try to do back prop there and after we are done with that, we will come to this final case which is a normal multi-layer perceptron of course I will do a sort of simpler example here also.

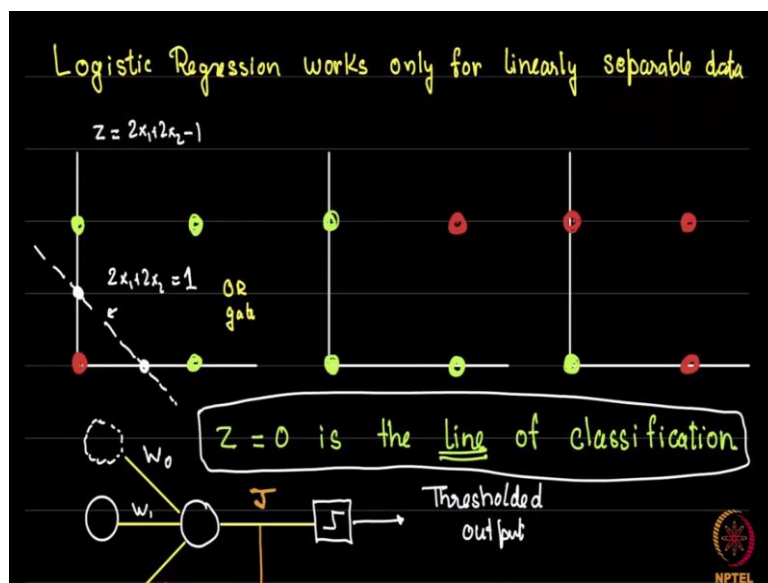
But by the end of this you should be clearly able to understand how it is that gradients are calculated in a general neural network. Finally, I have only talked about sigmoid activation function, we will talk about various other activation functions and in the context of back prop we will talk about why these various activation functions are required.

(Refer Slide Time: 15:43)



So, let us start with the first topic for this week, which is the XOR problem. So, remember that logistic regression works only for linearly separable data I had talked about this in the last week.

(Refer Slide Time: 16:01)



So, I had sort of made up some classification examples. So, these for example these three are elementary gates. This is for example is what is known as the OR gate, I had discussed this the

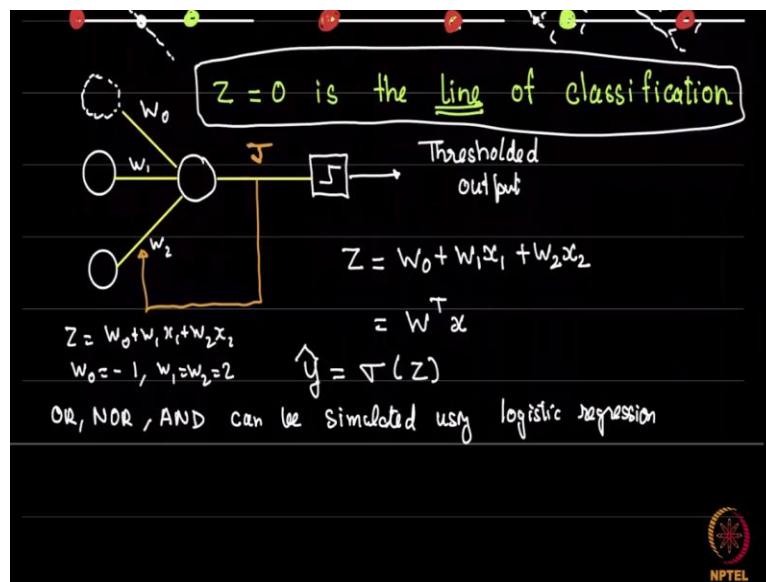
last time. The OR gate remember is a simple case where you are only trying to classify four points really there is no generalization here which is or you remember 0, 0 gives you 0.

0, 1 gives you 1 if either of these are active that is why it is called the OR gate you get 1. So, I have denoted here green as 1 and red as 0. So, this is a simple OR gate example. Now when I say OR gate all of us can intuitively draw a classification line. So, this classification line we can draw let me draw it in dotted lines so that it is a little bit thick. So, if you look at this line let us say this pass is right in the midpoint half zero and zero half.

So, this would then be the line $2x_1 + 2x_2 = 1$. You can simply check this calculation very simply if I set x_2 to 0 then x_1 is half if I set x_1 to 0 then x_2 is half. So, this is the line which passes right in the middle and this is the classification line. Now remember we had also seen in the last week that the line which is the classification line is the line their Z equal to 0.

So, this Z equal to 0 line is the classification line. So, here, for example, you can say $z = 2x_1 + 2x_2 - 1$.

(Refer Slide Time: 18:00)



Which in this structure here if you remember Z was also equal to $w_0 + w_1x_1 + w_2x_2$. So, by our interpretation w_0 is - 1, w_1 and w_2 are equal to 2 so that is because the coefficient of x_1 is 2, coefficient of x_2 is 2 and the constant term when it comes on the left-hand side is - 1. So, there is a one-to-one correspondence between these three. The algebraic expression for z the picture here and the classification line.

Now I had talked a little bit about this last week also, but I will say again that when you have a straight line it has the equation $w_0 + w_1x_1 + w_2x_2 = 0$ and z physically represents the linear activation here z physically represents this classification line. Now, if I similarly draw the classification line here. So, now I want to separate these three green points from the red point, this of course if you can see it is the AND gate.

I think I have drawn it in the wrong color, let me just change the color here. So, these three are of is 0, 1, 1, 0, 0, 0 are of and this one thing is on, so this is the AND gate. So, the AND gate is going to look like this. Now just by inspection you can sort of write the equation for the AND gate you can see that this will be around 3 by 2 and this will be 3 by 2. So, you could have something like $3x_1 + 3x_2$ I had also done this algebraically in the last week.

So, minus let us say 2 equals to 0 this would be the equation for the AND gate. So, that when x_1 equal to 0, x_2 equal to 3 by 2 so on and so forth. So, this will be the AND gate so that means that you can replicate the AND gate by another network whose weights are $-2, 3$ and 3 . So, just like the OR gate can be replicated by a network with w_0 equal to -1 and w_1 and w_2 equal to 2 AND gate can be replicated with w_0 equal to -2 .

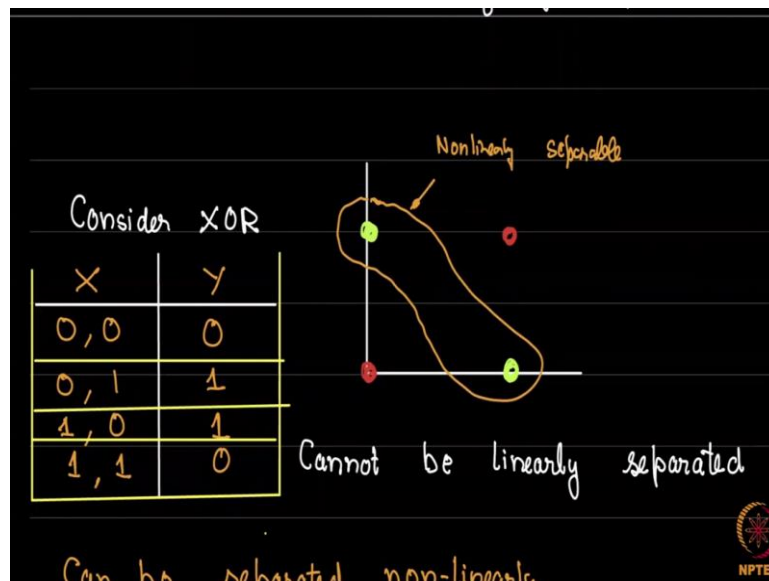
And w_1 equal to let me write that down instead of just saying it, but this could be w_1 , this could be w_0 . Finally, this one is an example of a flip gate this is the OR gate sorry this is the NOR gate you will see it is exactly the opposite or it is not of OR gate so NOR gate. Now it seems to have the same classification line as the OR gate except we are going to write the other way around and flip directions.

So, here z is basically $-2x_1 - 2x_2 + 1$. So, the weights are exactly flipped. So, as I had told you this one will be one on this side, whenever this distance is positive and it will be 0 here this one is exactly the opposite it will be 1 here and 0 there and as I also indicated last week really speaking the data points, we are looking at in realistic scenario are more like this. So, you have lots of data points here and some other data points here.

So, you will have a lot of red points here, red points here, red points here and green points here. Obviously are it being somewhat as if you are just selecting four data points. So, why all this exercise? The reason is each OR, NOR and AND can be simulated using logistic regression.

As I told you this allowing a simulation of elementary gates allows us to simulate more complex logic, this was the basic historic idea why people looked at this.

(Refer Slide Time: 22:47)




So, if we look at the XOR gate. So, the XOR gate is like this if you have both the same whether it is 0, 0 or 1 and 1 you are going to get 0. So, this and this are going to be 0. If the two are different 0 and 1 gives you 1, 1 and 0 gives you 1. So, for example both are different you are getting this. Now the problem is this cannot be linearly separated as I discussed in detail last week. You can however separate it using some curve of this sort.

This is a non-linearly separable. So, both are true it is not linearly separable as well as it is separable, but non linearly. So, you have to make a nonlinear surface somewhere around it to make it separable. So, since that is the case, I will tell you a couple of expressions that will achieve this.

(Refer Slide Time: 23:52)

| X | Y |
|------|---|
| 0, 0 | 0 |
| 0, 1 | 1 |
| 1, 0 | 1 |
| 1, 1 | 0 |



Cannot be linearly separated

Can be separated non-linearly.

For instance, $\hat{Y} = x_1^2 + x_2^2 - 2x_1x_2$ ✓

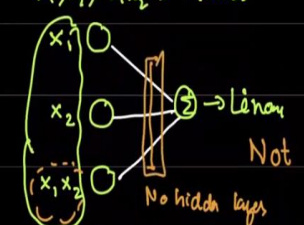
OR $\hat{Y} = x_1 + x_2 - 2x_1x_2$ ✓

So, for example, if it were linearly separable then you will be able to get some expression which will look like $w_0 + w_1x_1 + w_2x_2$ which would be a line which is separated which we know is not possible, but now if I looked at another expression $x_1^2 + x_2^2 - 2x_1x_2$. So, if you look at an expression of this sort this of course is $(x_1 - x_2)^2$ you will see $x_1 - x_2$ the whole square is 0, $(x_1 - x_2)^2$ is 0.

And $(x_1 - x_2)^2$ is 1 here. So, this actually works, but the price you pay here is you have now introduced new quantities. We only knew x_1 and x_2 now you are introduced x_1^2 x_2^2 and also across some $x_1 x_2$ in fact not this $x_1^2 + x_2^2 - 2x_1 x_2$ even $x_1 + x_2$, $x_1 x_2$ works because when x_1 is just 0 or 1 then x_1^2 and x_1 and x_1^2 and x_2 and x_2^2 are the same. So, this one work as well.

(Refer Slide Time: 25:02)

$x_1, x_2 \rightarrow$ Attributes
 $x_1, x_1^2, x_1 x_2 \rightarrow$ Features



Network. How?

to know

But how do we get x_1, x_2 ?

general in higher dimensions

Not possible in

CURSE OF DIMENSIONALITY

Solution \rightarrow Representation Learning

Feature Learning

\rightarrow Deep (er) networks

Now, suppose I want to represent this as a network how would I do it? The way to do it is as follows. You take an input we have done this before you take an input x_1 , you take an input x_2 you also take a third input $x_1 x_2$. Now it should seem you will think well why should I need this I have already given x_1 and x_2 . The reason is these are only linear combinations. So, this is never going to arise unless you add it explicitly.

So, typically what we say is x_1 and x_2 are attributes that is the original properties we were given as inputs, but x_1 , x_2 and $x_1 x_2$ are features, features meaning you put these together. So, for example, you want to say somebody face is beautiful you might say both the eyes they have good features that is because it is not just the original single eye left eye, right eye you are looking at you are looking at the combination of the two or combination of all the features put together.

So, these combinations that occur, these are the features. So, you can simply do linear regression for this data set, but the catch is just like the quadratic and the cubic regressions that we did earlier this linear regression works only if you give $x_1 x_2$ explicitly, but the key question is this, how are you going to know magically that you need $x_1 x_2$. So, rather than saying how do we get $x_1 x_2$ which is just a simple calculation.

How do we get to know that it was $x_1 x_2$. Now here the data I took was simple I just took 0, 1, 1, 0 what if you know it was a numerical data and I needed x_1 square x_2 or x_1 cube x_2 or a whole lot of other possibilities. So, what happens is, in general, it is not possible to discover these or keep on giving a list of all possible combinations systematically in higher dimensions; higher dimensions mean here we are just dealing with two variables.

But if I deal with multiple variables which I typically do in machine learning. Now high number of variables, high amount of data, this is the general catch behind machine learning. This is generally not possible and that is what is known as the curse of dimensionality that is you cannot arbitrarily provide these features intelligently or systematically in higher dimensions and this is where we have our deep learning algorithms.

So, the idea is what is known as representation learning or in our case let us it feature learning. How do I find these features out automatically. So, for that what we need are deeper networks,

that is we currently had no depth at all, no hidden layer and the catch is if you want to achieve things without hidden layers you have to pay the price by adding a large number of arbitrary features here. Instead of that if somehow you put a middle layer.

You add a middle layer here and you do some unknown computations or you do some computations with unknown meanings then you result in deeper networks.

(Refer Slide Time: 29:00)

XOR via a deep network


$$\text{XOR} = \text{NOR}(\text{NOR}(x_1, x_2), \text{AND}(x_1, x_2))$$

$$\text{XOR} = \text{NOR}(A_1, A_2) \text{ where}$$

$$A_1 = \text{NOR}(x_1, x_2)$$

$$A_2 = \text{AND}(x_1, x_2)$$

| (x_1, x_2) | A_1 | A_2 | Y |
|--------------|-------|-------|-----|
| 0, 0 | 1 | 0 | 0 |




So, I want to show you how that happens in XOR via a deeper network. So, as I told you the XOR network can be written in terms of simpler logic gates also. For example, the XOR truth table that I showed you which was 0, 0 and 1, 1 give us 0 and 0, 1 and 1, 0 give you 1 it can be written as this combination. you had x_1 and x_2 apply a NOR then for the same x_1 and x_2 apply an AND then put a NOR of the output of this two and that will give you XOR.

So, another way of writing it is let us say NOR of x_1 x_2 is A_1 AND of x_1 x_2 is A_2 then you do not of A_1, A_2 and that will give you XOR.

(Refer Slide Time: 29:57)

$A_2 = \text{AND}(x_1, x_2)$

| (x_1, x_2) | $\text{NOR}(x_1, x_2)$ $\text{AND}(x_1, x_2)$ $\text{NOR}(A_1, A_2)$ | | |
|--------------|--|-------|-----|
| | A_1 | A_2 | Y |
| 0, 0 | 1 | 0 | 0 |
| 1, 0 | 0 | 0 | 1 |
| 0, 1 | 0 | 0 | 1 |
| 1, 1 | 0 | 1 | 0 |

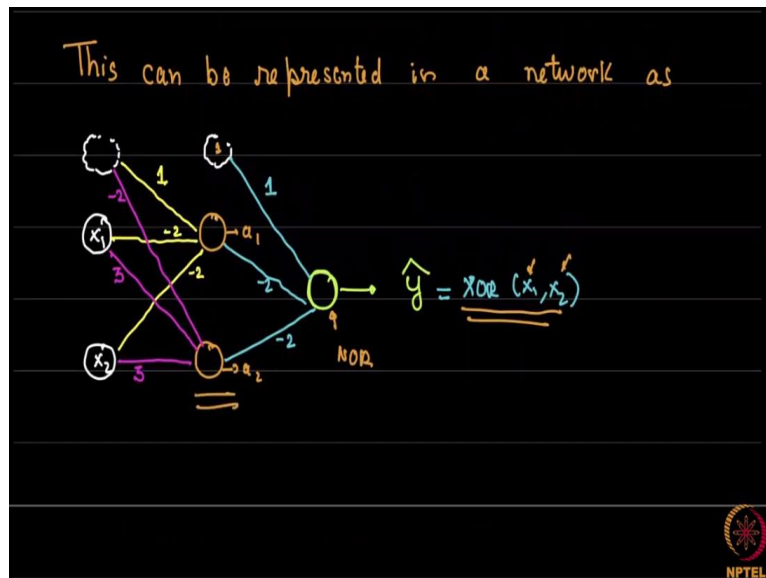


So, just to show you that this is true remember I had written NOR of x_1 x_2 NOR of sorry AND of x_1 x_2 and NOR of A_1 A_2 so this is how these quantities were calculated. So, just as an example if you have x_1 x_2 if I do NOR or gives me 0. So, NOR is just not of r so that gives me 1. Similarly, if you do AND I get 0 now I take 1 and 0 and I do a NOR I get y which is 0.

So, you can repeat this exercise and test it out and you indeed get the XOR outputs. So, for example, 0, 0 gives you 0; 1, 0 gives you 1; 0, 1 gives you 1 and 1, 1 gives you 0. Notice, how it achieved it some of these outputs were sort of spread together you got the same output here or you got somewhat flipped outputs here, but the catch here is these two are intermediate calculations.

These intermediate calculations made it possible to calculate y without ever notice we never invoked x_1 square or x_2 square or x_1 x_2 . You only invoke the linear features x_1 and x_2 , but you did some intermediate calculation and that is how y suddenly magically came into b.

(Refer Slide Time: 31:37)



Now, we can represent this as a network, how so? So, for example, remember that we had all these weights from before. So, for example this was x_1 , this is x_2 and this one is supposed to be NOR of x_1 x_2 .. Suppose I say that I first do a NOR just like I did before. So, all we do is use the weights that we already knew, the NOR weights were given here right up here. So, the NOR weights were w_0 is 1; w_1 is - 2 and w_2 is -2.

So, you can just write those weights here w_0 is 1, w_1 is - 2 and w_2 is - 2. Now it turns out that well let me do this in a short while. So, similarly the second one is AND the AND gate if you remember was 3, 3, - 2 so this one would be - 2 this one is 3 and this one is 3. So, what comes out as output here is what we call a_1 this one is a_2 and this one is as usual the biasing. Now this gate is again the NOR gate. So, it has the same weights as before.

So, the same weights as before happen to be 1, - 2, - 2. So, you combine this network together with the hidden outputs being a_1 and a_2 . In this case we can interpret it and the final output then is going to be \hat{y} is XOR of x_1 x_2 . Now I want to point out that all these you know to assume this of course we assume that there was sigmoid here, sigmoid here, sigmoid here as it turns out this will not be exactly XOR because there is a sigmoid which will give you an input between 0 and 1 at the end you have to do some thresholding this I showed here.

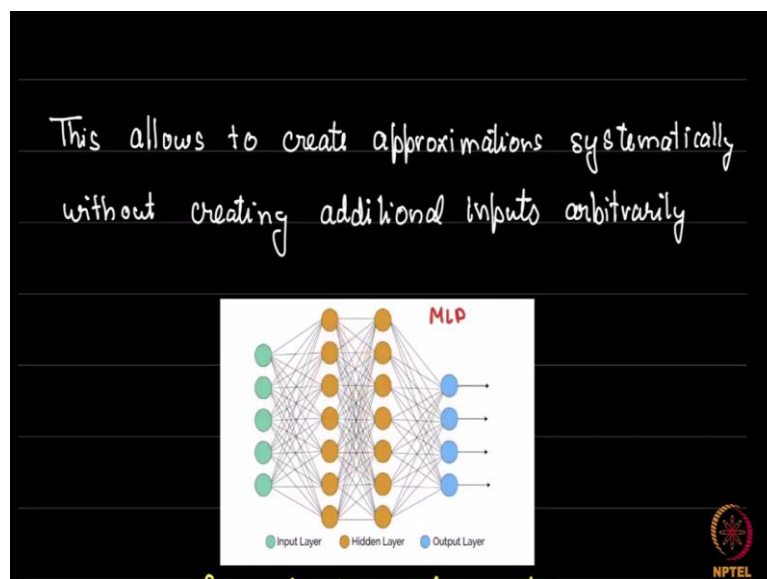
Typically, when we say that the output is an OR output, we want 0 or 1, obviously the output is not going to be that. It is going to be greater than 0.5 or less than 0. 5. Let us say the output comes as 0.6 then if you want the output you have to put a step function here. So you have to

put some step function here and this is the thresholder output which says that greater than 0.5 goes to 1 and less than 0.5 goes to 0.

However, when we take the gradients, they are taken from the non-threshold output, the direct output of \hat{y} because after thresholding you cannot really differentiate. You can only differentiate the sigmoid, so you calculate the sigmoid, you will get a J using our binary cross entropy and then you do a back prop or you do a gradient from there and connect w_0 w_1 and w_2 .

So, similarly when you actually run this in practice you have to be careful about whether you have, you know, you will not actually achieve this, this is sort of a toy example, but the primary purpose is to motivate that a deep network actually arises naturally if you only want to use these linear terms provided you are willing to expand these intermediate calculations.

(Refer Slide Time: 35:45)

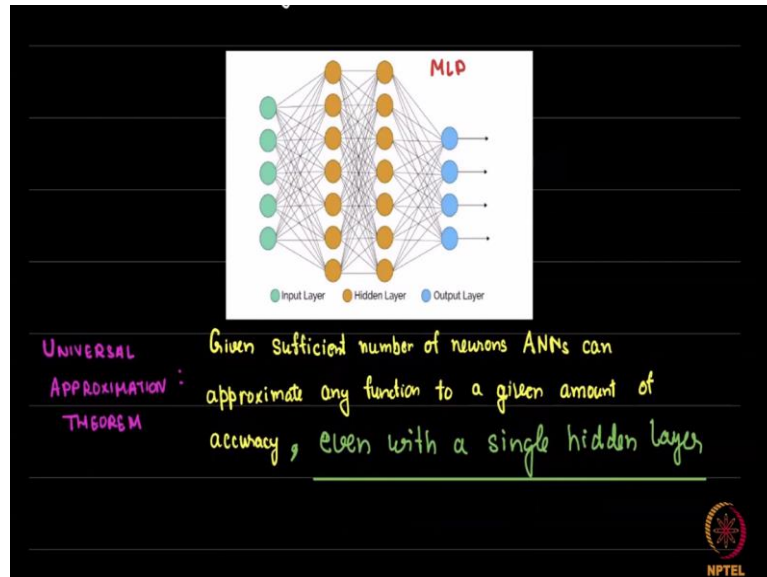


So, what happens here through this entire process is this allows us to create approximations systematically. Remember our entire purpose of using a network is to create an approximate model connecting the input to the output. So, this is our entire purpose. Now if I have 4 inputs, I want to create x_1 square, x_2 square, x_3 square, x_4 square, x_5 square, x_1 , x_2 , x_3 , x_4 etcetera that just becomes humongous expensive as well.

As we really do not know which are terms to take, should I take, you should I take power 4 how many terms should I take that is why it is easy like when we feel that the approximation is not good enough, we simply add more layers or we add more neurons. I will come to

somewhat of the subtle distinction between these two in the next couple of videos, but nonetheless we will basically achieve our purposes rather than touching the input which we keep as it is, we start adding stuff in the middle and as it turns out that is good enough.

(Refer Slide Time: 36:49)



In fact, we have the universal approximation theorem which says that if you just give me one layer, just give me one intermediate calculation layer as mentioned here. Even with one single hidden layer, you can approximate any function to any given amount of accuracy, provided you give me sufficient number of neurons. So, that is it for this video. In the next video we will start seeing how to actually move here through this network from beginning to end. Thank you I will see you in the next video.