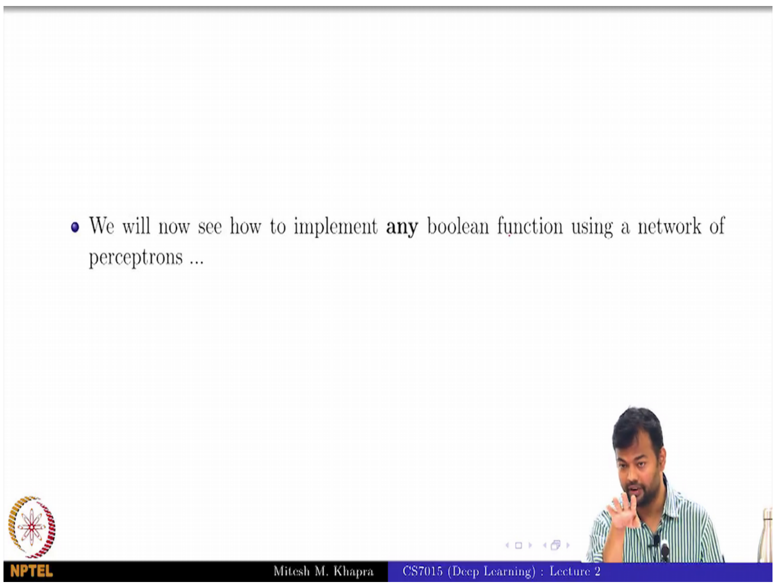
Deep Learning

Prof. Mitesh M. Khapra
Department of Computer Science and Engineering Indian Institute of Technology, Madras

Module – 2.8
Lecture – 02
Representation Power of a Network of Perceptrons

We will go to the next module, where we talk about a Network of Perceptrons and then we talk about the representation power of a network of perceptrons. So, this module should have been titled as Network of Perceptrons, ok. So now, in particular what we are going to see is how any Boolean function whether linearly separable or not can be represented using a network of perceptrons.

(Refer Slide Time: 00:25)

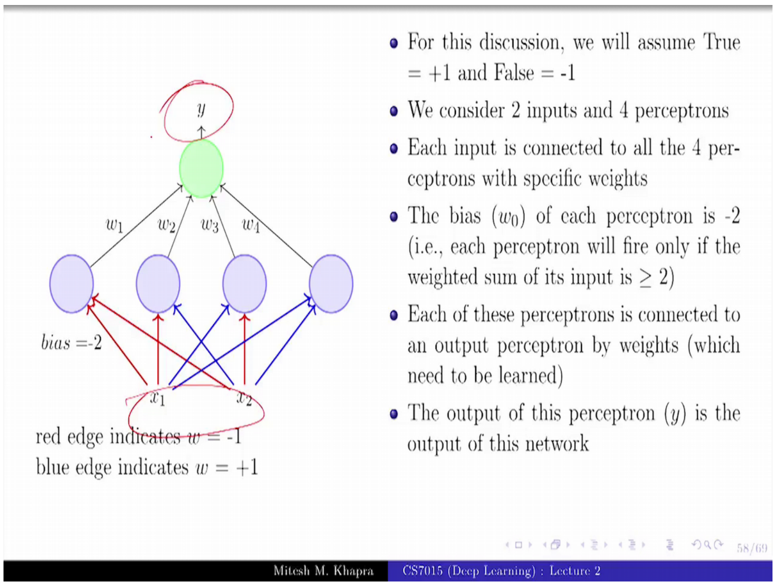


- We will now see how to implement **any** boolean function using a network of perceptrons ...

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

Now, what do I mean by represented during a network of perceptrons? What it means is that I will give you a network of perceptrons, you take any Boolean function feed any value of x 1 to x n and the network will give you the same y as it is expected from the truth table, ok, that is what representation means just to put it out clearly, ok, fine.

(Refer Slide Time: 00:57)



- For this discussion, we will assume True = +1 and False = -1
- We consider 2 inputs and 4 perceptrons
- Each input is connected to all the 4 perceptrons with specific weights
- The bias (w_0) of each perceptron is -2 (i.e., each perceptron will fire only if the weighted sum of its input is ≥ 2)
- Each of these perceptrons is connected to an output perceptron by weights (which need to be learned)
- The output of this perceptron (y) is the output of this network

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

58/69

And now I am going to again do a setup, I am not giving you the solution, I am just making some set up and then we will discuss the solution, right.

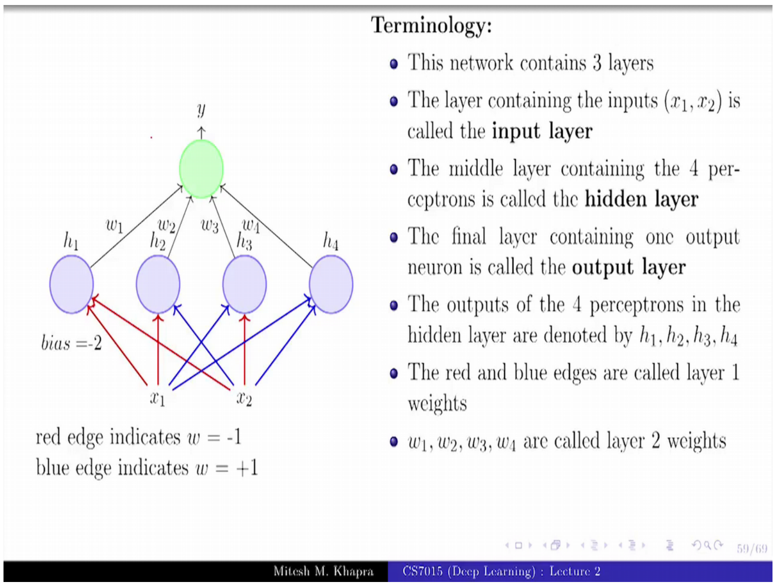
So, for this discussion we will assume that true equal to plus 1 and false equal to minus 1. So, instead of 0 and 1, we will assume minus 1 and plus 1, ok. And these are your inputs x 1 and x 2 we are taking the 2 input case. And I will have 4 perceptrons first, I will have 4 perceptrons; and I will also have very specific weights connected to from the inputs to these perceptrons. So, red means minus 1 and blue means plus 1, right. So, the first 2 inputs are connected with a weight of minus 1. The next 2 inputs with minus 1 plus 1, plus 1 minus 1 and the last would be?

Student: (Refer Time: 01:43).

Plus 1 plus 1, ok. Now, once I have this, I will set the bias of all these perceptrons to minus 2, ok. So, that will; that means, they will fire only if they are weighted sum of the inputs is greater than 2, ok, fine? Ok now after this I will have one more perceptron. So, I had 2 inputs, I converted that to 4 values, these 4 values are now going to feed into one more perceptron ok.

And these weights I will not fix them, these are the weights that I am going to learn ok, these and the final output of this perceptron which is the green perceptron is the output of the network, right. So now, coming back to what I said that it can represent any function, what I mean is that you take any function, feed in any combination of x 1, x 2, this network will give you y and I am telling you that it will match the truth table of that function, ok, right.

(Refer Slide Time: 02:43)



Terminology:

- This network contains 3 layers
- The layer containing the inputs (x_1, x_2) is called the **input layer**
- The middle layer containing the 4 perceptrons is called the **hidden layer**
- The final layer containing one output neuron is called the **output layer**
- The outputs of the 4 perceptrons in the hidden layer are denoted by h_1, h_2, h_3, h_4
- The red and blue edges are called layer 1 weights
- w_1, w_2, w_3, w_4 are called layer 2 weights

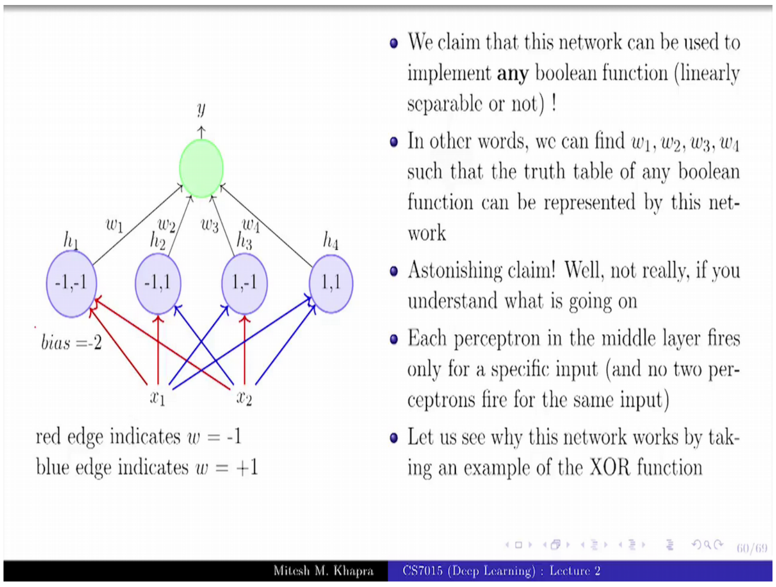
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2 59/69

Now, let us define some terminology this will also stay with us for the rest of the course.

So, this network contains 3 layers. The layer containing the inputs it is called the input layer; very creative, the middle layer containing the 4 perceptrons is called the hidden layer. And the output layer which gives you the output is called the output layer output perceptron; which gives you the output is called the output layer right. So, you have a input layer or hidden layer and an output layer.

And the outputs of the 4 perceptrons I am going to call them as h 1, h 2, h 3 and h 4, ok. And the red and blue edges are called the weights for the layer 1, which we have not learned we have actually set them by hand. And the weights for w 1, w 2, w 3, w 4 are called the weights for the second layer, other the layer 2 weights and these are the weights that we want to learn, ok.

(Refer Slide Time: 03:34)


y
w_1
w_2
w_3
w_4
h_1
h_2
h_3
h_4
-1,-1
-1,1
1,-1
1,1
bias =-2
x_1
x_2
red edge indicates w = -1
blue edge indicates w = +1

- We claim that this network can be used to implement **any** boolean function (linearly separable or not) !
- In other words, we can find w_1, w_2, w_3, w_4 such that the truth table of any boolean function can be represented by this network
- Astonishing claim! Well, not really, if you understand what is going on
- Each perceptron in the middle layer fires only for a specific input (and no two perceptrons fire for the same input)
- Let us see why this network works by taking an example of the XOR function

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

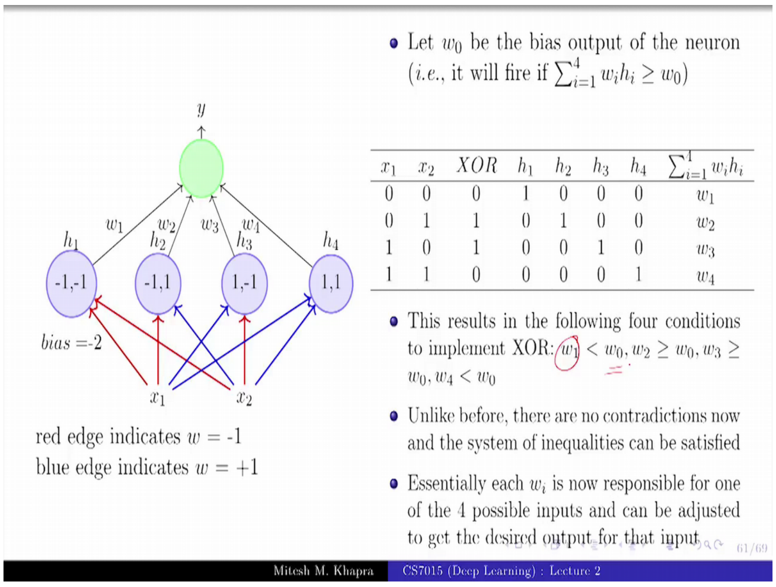
60/69

Now I make this claim that this network, it can take any Boolean function, it can implement any Boolean function, right? So, this same network can implement any Boolean function; that means, if I take this network, and if I try to learn the values of w 1, w 2, w 3, w 4, for any Boolean function whether it was originally linearly separable or not; I will be able to implement it, right.

So, isn't this an astonishing claim? Any Boolean function, do you think this is an astonishing claim? Well not really if you really understand what is happening here right. So, let us see what exactly is happening here. So, when will perceptron 1 fire? When the input is false false 0; will it fire for any other input? When will perceptron 2 fire? Any other input? Same for the next perceptron? Same for the next perceptron?

So, you start getting an intuition of what is happening here? You do, let us see. So now, for this particular network now that I have given you some intuition of what is happening. Basically every node or every neuron in the hidden layer is catering to one of the inputs, and it will fire only for that input, it will not fire for anyone else right.

(Refer Slide Time: 04:59)



- Let w_0 be the bias output of the neuron (*i.e.*, it will fire if $\sum_{i=1}^{4} w_i h_i \geq w_0$)

x_1	x_2	XOR	h_1	h_2	h_3	h_4	$\sum_{i=1}^{4} w_i h_i$
0	0	0	1	0	0	0	w_1
0	1	1	0	1	0	0	w_2
1	0	1	0	0	1	0	w_3
1	1	0	0	0	0	1	w_4

- This results in the following four conditions to implement XOR: $w_1 < w_0, w_2 \geq w_0, w_3 \geq w_0, w_4 < w_0$
- Unlike before, there are no contradictions now and the system of inequalities can be satisfied
- Essentially each w_i is now responsible for one of the 4 possible inputs and can be adjusted to get the desired output for that input

61/69

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

So now let us try to implement the XOR function, and see what are the ok. So now, let us try to implement the XOR function, and see what are the set of inequalities that result from this. Earlier when we try to look at the set of inequalities, we ended up with a contradiction, let us see if that happens now. So, this is x 1, x 2 this is your XOR function. So, that is just like any truth table then I am noting down the intermediate values. And then my final input to the green perceptron is going to be summation of these, ok, fine and it will fire if this summation is greater than equal to 0, or else it will not fire, ok.

Now, for the first case when the input is 0 comma 0 what is h 1 going to be? 1 and everything else is going to be 0, that is exactly what we saw in the previous slide. So, what is the summation going to be? Just w 1 right, so, it is w 1 h 1 plus w 2 h 2 so on, but h 2 to h 4 are 0. So, only thing that remains is w 1; for the second case, w 2; for the third case, w 3 for the 4th case w 4.

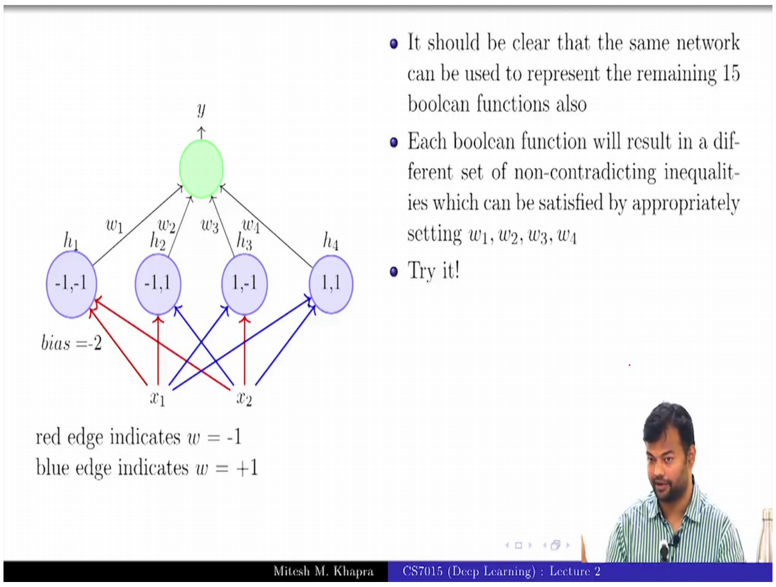
So, is it clear now what is happening? Let us go a bit more into detail right. So now, for the XOR condition, what are the conditions that we need? W 1 should be less than w 0, right? Because this should not fire, w 2 should be greater than equal to 0, w 3 should be greater than equal to sorry w naught not w is not 0, and w 4 should not fire. So, w 4 should be less than w. Are there any contradictions here? By design no right. So, we have made sure that for the final layer only one of these guys feeds to it.

So, it does not matter what the remaining outputs are, right? They do not interfere with each other, unlike earlier where we had conditions like w 1 should be something, w 2 should be something, and then w 1 plus w 2 should be something. There are no such contradictions here. Because we have made sure that every neuron in the middle layer actually caters to one specific input, and now the weights in the final layer can be adjusted so that we get the desired output for that input, right.

So, I can set whatever value of w and I need to set; so, that I can fire the new on it. In fact, I could just fix w 0 as 0, and then I can adjust the weights of w 1, w 2, w 3. w 4, right. And I can implement the XOR function. So, are you convinced that this can be used to implement any Boolean function? How many if you are not convinced? So, the negative question never works, how many if you are convinced? Sure, ok.

Now, what if we had 3 inputs?

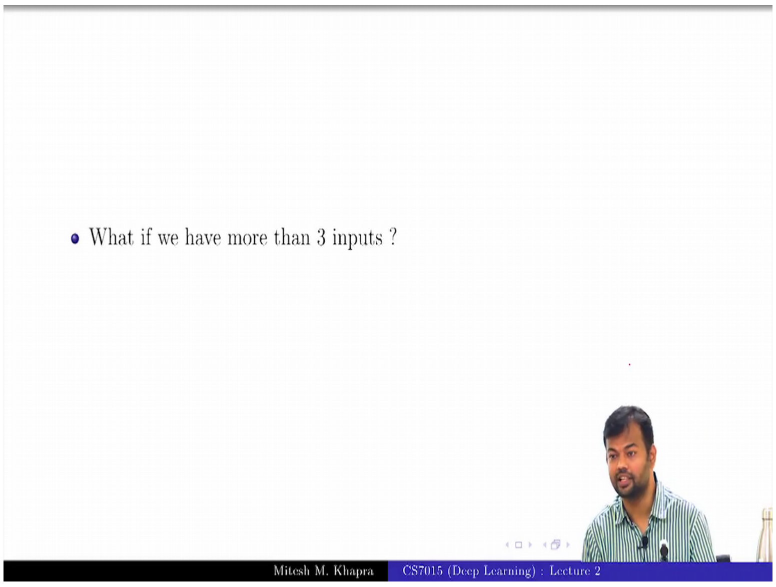
(Refer Slide Time: 07:40)



- It should be clear that the same network can be used to represent the remaining 15 boolean functions also
- Each boolean function will result in a different set of non-contradicting inequalities which can be satisfied by appropriately setting w_1, w_2, w_3, w_4
- Try it!

Before that it should be clear that the same network can be used for any of the remaining 15 functions, right? And for each of these functions we will end up with a different value of w 1, w 2, w 3, 4, but you will be able to satisfy the truth table, right and you can go home and try it, which I am sure you will do.

(Refer Slide Time: 07:56)


- What if we have more than 3 inputs ?

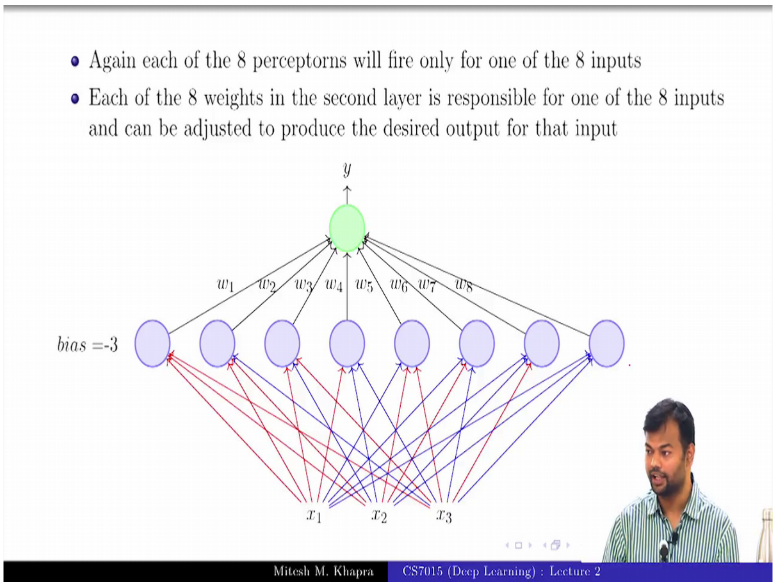
Mitesh M. Khapra
CS7015 (Deep Learning) : Lecture 2

Ah So, what if we have a function of 3 inputs? 256, what is 256? No.

Student: (Refer Time: 08:04).

8 right? Fine, sure, ok.

(Refer Slide Time: 08:11)


- Again each of the 8 perceptrons will fire only for one of the 8 inputs
- Each of the 8 weights in the second layer is responsible for one of the 8 inputs and can be adjusted to produce the desired output for that input

So, this is what it will look like, and anything specific about the weights of the initial layer, can you tell me what the weights would be? Just tell me red red red red blue blue whatever colours you like this thing. First perceptron what would the weight colours be red, red, red, then?

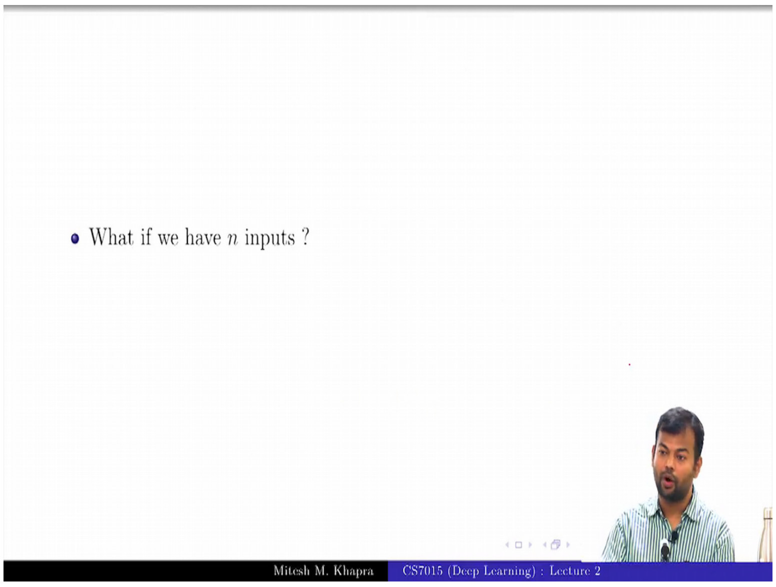
Student: (Refer Time: 08:29).

Then.

Student: (Refer Time: 08:30).

Enough so, this is how it will look right, right and now this same thing will work with the same logic. For any Boolean function of 3 inputs, you will get these 8 inequalities, and they will not interfere with each other, and you can set the values of w 1 to w 8 so that you can satisfy it ok, fine.

(Refer Slide Time: 08:47)



- What if we have n inputs ?

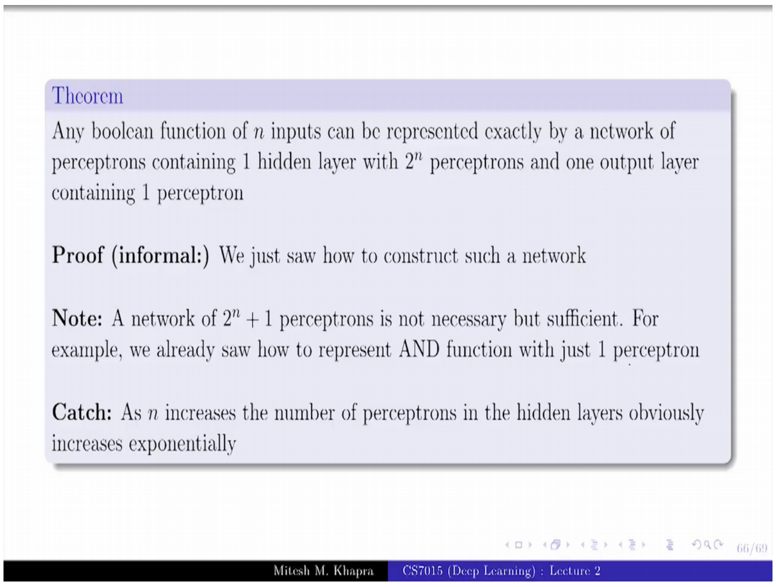
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

So, what if we have n inputs?

Student: 2 power n.

2 power n perceptrons in the middle layer, right, ok?

(Refer Slide Time: 08:55)



Theorem

Any boolean function of n inputs can be represented exactly by a network of perceptrons containing 1 hidden layer with 2^n perceptrons and one output layer containing 1 perceptron

Proof (informal:) We just saw how to construct such a network

Note: A network of $2^n + 1$ perceptrons is not necessary but sufficient. For example, we already saw how to represent AND function with just 1 perceptron

Catch: As n increases the number of perceptrons in the hidden layers obviously increases exponentially

66/69

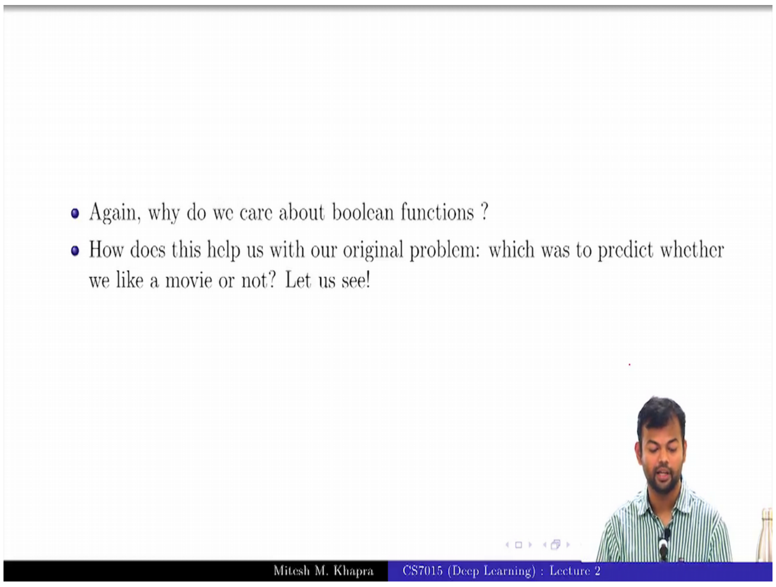
Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

So now here is the theorem, any Boolean function of n inputs can be represented exactly by a network of perceptrons containing one hidden layer with 2 raised to n perceptrons, and one output layer containing 1 perceptron, right? We just saw an informal proof of this, we just constructed I just gave you the answer it this is how you will get it, right? Now note that a network of 2 raised to n plus 1 perceptron, is it sufficient or necessary? Or both? Sufficient yes that is what it says, is it necessary?

No, we already saw the and function which we can just represent using a single perceptron, right. So, it is not necessary, but it is sufficient ok. So, this is a very powerful theorem if you think of it right. So now, this whole objection right remember this history, and when we have the c I winter when people showed that perceptron cannot handle the XOR function that is for a single perceptron.

If you have a network of perceptrons you can actually have any Boolean function, but what is the catch? As the value of n increases the number of neurons increases exponentially right, but still in theory you could have a solution, ok.

(Refer Slide Time: 10:01)

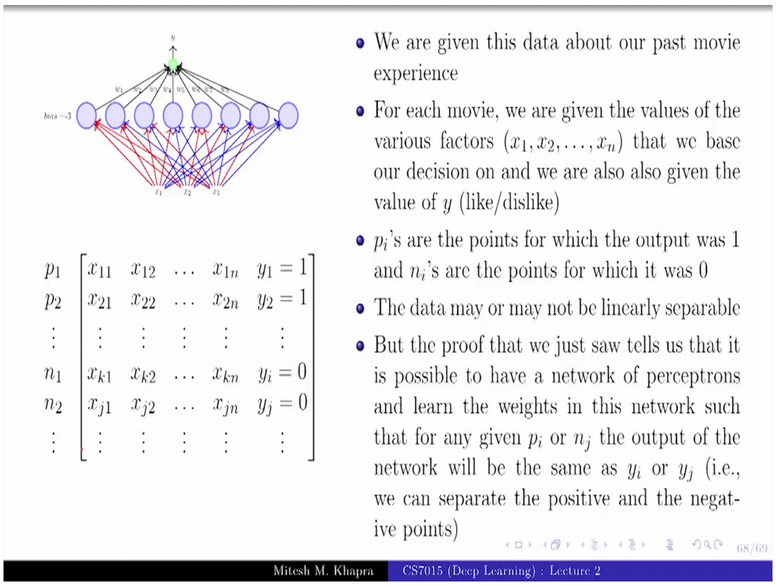


- Again, why do we care about boolean functions ?
- How does this help us with our original problem: which was to predict whether we like a movie or not? Let us see!

Now again why do we care about Boolean functions, I keep coming back to this. Why do we care about Boolean functions, because you took this and so, the question that I the question that I want to answer is, how does this relate back to our original problem? Right we know any Boolean function can be implemented, how do we go back to our original problem; which is whether we like a movie or not, right.

And you could see that there is a whole family of problems there, right, whether we like a movie or not, whether we like a product or not, whether I want to go home today or not, yes no any kind of a yes no problem, it is a whole family of problems there, right?

(Refer Slide Time: 10:34)



$$\begin{array}{c|cccccc} p_1 & x_{11} & x_{12} & \dots & x_{1n} & y_1 = 1 \\ p_2 & x_{21} & x_{22} & \dots & x_{2n} & y_2 = 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_1 & x_{k1} & x_{k2} & \dots & x_{kn} & y_i = 0 \\ n_2 & x_{j1} & x_{j2} & \dots & x_{jn} & y_j = 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

- We are given this data about our past movie experience
- For each movie, we are given the values of the various factors $(x_1, x_2, \ldots, x_n)$ that we base our decision on and we are also also given the value of y (like/dislike)
- p_i's are the points for which the output was 1 and n_i's are the points for which it was 0
- The data may or may not be linearly separable
- But the proof that we just saw tells us that it is possible to have a network of perceptrons and learn the weights in this network such that for any given p_i or n_j the output of the network will be the same as y_i or y_j (i.e., we can separate the positive and the negative points)

68/69

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

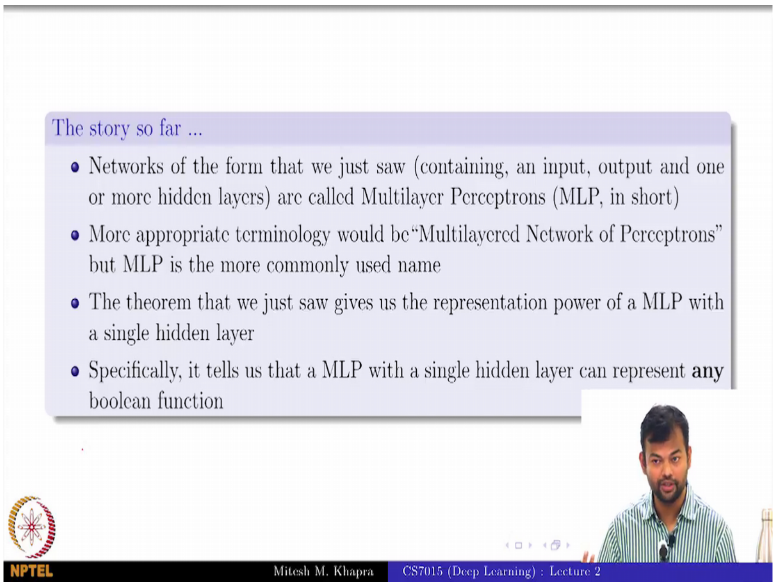
So, let us see so, we are given this data from our past experience, right. So, we are told that this is what the movie looks like. These are the actor’s, director’s, joiners everything; we also know whether we like these or not.

So, we have a set of positive points and we have a set of negative points, right. And now we want to have a computational model which can satisfy this data; that means, once the model is trained, once whatever algorithm I algorithm I use has converged, it should be able to give me the correct output for a given input. That is what we are interested in, and that is a real classification problem that we are interested in. Now for each movie we are given these factors as well as the decision.

And I said p i’s and n i’s are positive and negative points. The data may or may not be linearly separable. It is not necessary that the data is linearly separable, those were the goody cases it, but in general that may not happen. But do we worry about it now? No what the previous theorem told us is that irrespective of whether your data is linearly separable or not, I can give you a network which will be able to solve this problem modulo; that it might be very expensive in the number of neurons in the middle, right? But if you keep that aside, I have a solution for this.

And that is why we care about Boolean functions, because many problems we could actually cause to it in a simplistic way, if we ignore the real inputs. And if you even think of the real inputs, right? Suppose it could take all values between 0 to 1, you can always make it binary, you could say that is the value between 0 and 0.1 is the value between 0.1 and 0.2. And you could make it as small the scale as small as possible right. So, that is why we care about this.

(Refer Slide Time: 12:09)



The story so far ...

- Networks of the form that we just saw (containing, an input, output and one or more hidden layers) are called Multilayer Perceptrons (MLP, in short)
- More appropriate terminology would be "Multilayered Network of Perceptrons" but MLP is the more commonly used name
- The theorem that we just saw gives us the representation power of a MLP with a single hidden layer
- Specifically, it tells us that a MLP with a single hidden layer can represent **any** boolean function

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 2

So, the story so far has been that the network of networks of the form that we just saw, it which contain one input layer, output layer and one or more hidden layers. These are known as multilayer perceptrons, but a more appropriate terminology would be multi-layered network of perceptrons, right?

Because the perceptron is not multi-layered; you have a network of perceptrons and that network has many layers right. But generally there is abuse of notation; we always call it MLP which is multi-layered perceptrons. And the theorem that we just saw gives us the representation power of an MLP, and basically tells us that it can represent any Boolean function that we want to deal with, right? So, that is where we will end this class, and in the next class we will talk about sigmoid neurons, ok.

Thank you.