

Deep Learning for Computer Vision
Professor. Vineeth N. Balasubramanian
Department of Computer Science and Engineering
Indian Institute of Technology – Hyderabad
Lecture – 22
Neural Networks: A Review-Part 2

(Refer Slide Time: 00:15)

The XOR Conundrum

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0$$

- The fourth condition contradicts conditions 2 and 3
- No solution possible satisfying this set of inequalities

Indeed you can see that it is impossible to draw a line which separates the red points from the blue points

Vineeth N.B. (IIT-H) 54.1 Neural Networks Review

The XOR Conundrum

x_1	x_2	XOR	
0	0	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
1	1	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 > -w_0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 > -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 < 0 \implies w_1 + w_2 < -w_0$$

- The fourth condition contradicts conditions 2 and 3
- No solution possible satisfying this set of inequalities

Vineeth N.B. (IIT-H) 54.1 Neural Networks Review

We said that although Rosenblatt proposed the perceptron. Minsky and Papert later showed that the perceptron is limited to only a certain kind of data configurations and does not work for data configurations beyond those times and the examples they used was the XOR gate. Let us try to understand that we know with the graph truth of the XOR gate the truth table for

the XOR gate is given by something like this whereas one of the inputs is 1 you get a 1 otherwise a 0.

So from a perceptron perspective what we would want is in the first scenario we would want a $w_i x_i$ to be less than 0 because we want the output to be 0. Similarly, these two cases we would want it to be greater than or equal to 0 and in the last case we wanted to be less than 0 again. So, let us analyze this let us take the first equation which says w_0 plus w_1 into 0 which comes here plus w_2 into 0 which comes from here.

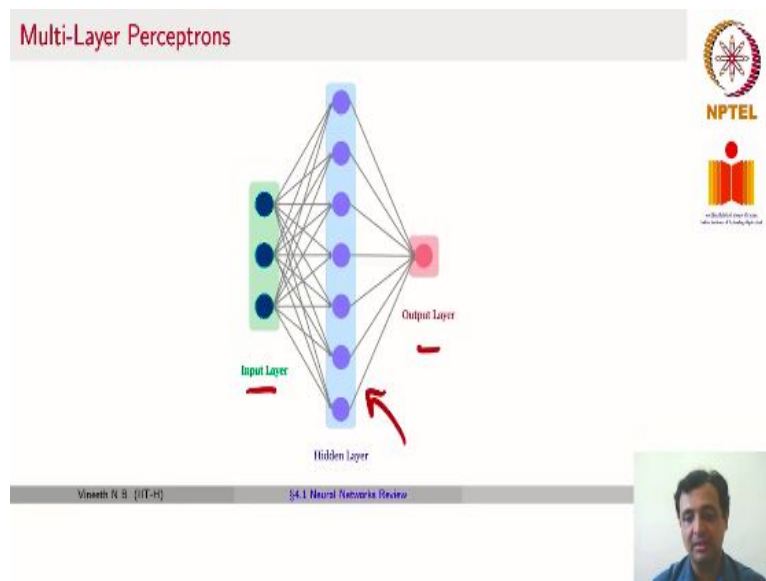
We ideally wanted to be less than 0 which is what we want the perceptron to do which means w_0 has to be less than 0. Similarly, from the second table a line of the truth table you have $w_0 + w_1 \cdot 1 + w_2 \cdot 0$ which are the inputs in the second line. We want that to be greater than or equal to 0 which means w_1 is greater than minus w_0 . So since w_0 is less than 0 w_1 would be a positive quantity greater than the absolute value of w_0 .

Similarly, the third line of the truth table would similarly get you w_2 to be greater than negative w_0 and the final line would show that $w_0 + w_1 + w_2$ should be less than 0 or $w_1 + w_2$ must be less than $-w_0$. It is quite clear here that because w_1 and w_2 will be positive $w_1 + w_2$ cannot be less than $-w_0$ because we know that individually each of them are greater than $-w_0$ which itself is a negative number.

w_0 by itself is a negative number, so $-w_0$ will be a positive number. So we can see a contradiction in these criteria here and that should clearly show you what the perceptron cannot solve the XOR problem. Also visually speaking you have the XOR problem to be represented as you want these two elements which are 0, 1 and 1, 0 to have labeled to be 1 and you have these two elements 0, 0 and 1, 1 to have labeled it to be 0.

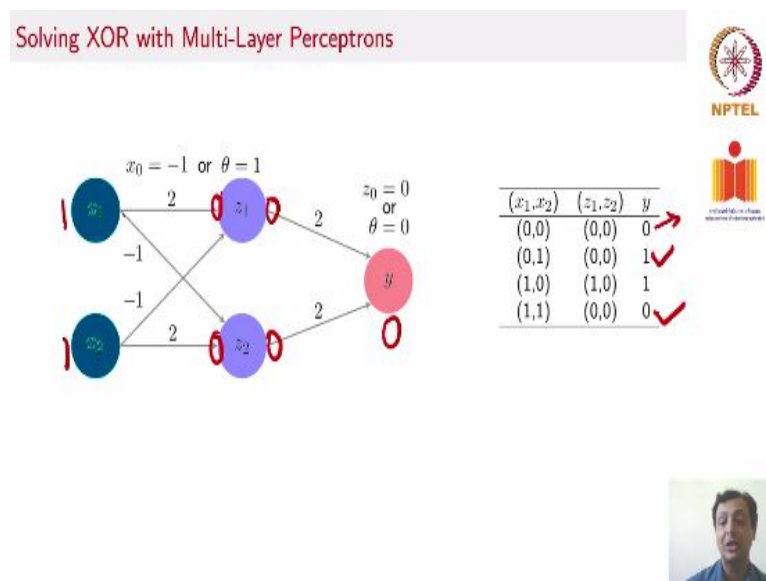
And as we said a perceptron simply embodies a line and you should draw a line here you are going to get this element wrong and if you draw a line here you are not going to get this element wrong and with XOR gate cannot be solved by the linear model. So, as you can see it is impossible to draw a line which separates the red points from the blue points here.

(Refer Slide Time: 03:52)



And that leaves us to the concept of multi layered perceptron or the multi layer perceptron as the figure shows. It is not restricted unlike perceptron to only input layer and an output layer and also has the convenience of including a hidden layer of neurons. The number of neurons in this hidden layer is a designed decision.

(Refer Slide Time: 04:21)



So here is an example of how multi layer perceptron can be used to solve the XOR problem. This is just one way configuration that can solve the XOR problem. There could be other big configurations that may solve or may not solve the XOR problem. All that we are trying to

say here is that there is at least one way configuration with a multi layer perceptron that can solve the XOR problem.

Let us look at this example so you have the weights denoted on each of these connecting edges here and you also have the y is given to be minus 1. So with these values given to us let us take a couple of cases from the truth table and welcome that this indeed gives a solution. So if you have 0, 0 as a input you are going to get 0 into 2 plus 0 into minus 1 plus minus 1 z_1 we get a input minus 1.

So it is a perceptron and its input is negative input given output to be 0 you would get exactly the same output for z_2 you will get minus 1 and its output to be 0 which means you are going to have what y gets as 0 which we can assume that only values greater than 0 is a threshold to 0 and only values greater than 0 gives you 1 is input and you would get and because the output at y is 0 you would now get the output to be 0.

Let us take the second case now and see how this works out for this particular scenario. So, let us consider the input now to be 0 and 1 so if that be the case you know z_1 is going to get a 0 and a minus 1 and a minus 1 which would turn out to be minus 2 which means the output would be 0 and unless z_2 you would get z_2 would get a 0 and z_2 and a minus 1 which should be 1 which means the output of z_2 will be 1.

Which means you are going to get an output of 2 here plus 2 into 1 plus 2 into 0 which corresponds to a 1. You could hold a similar thing would hold for the third row third tuple 1, 0 let us rather work out the last one now just to be sure about this again. So let us consider 1, 1 when we have 1, 1 you are going to have 2, minus 1 minus 1 which is 0 which means 0.

Similarly, you have a 2, minus 1, minus 1 coming from bias as 0, 0, 0 and a 0. We can put this out if this was fast you will notice that this is a valid solution for the XOR problem.

(Refer Slide Time: 07:15)

Multi-Layer Perceptrons

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): How? Each of the 2^n hidden layer perceptrons can model (or can be fired by) one combination of n inputs.

Note: A network of $2^n + 1$ perceptrons is not necessary but sufficient

But why does this matter? As n increases, the number of perceptrons in the hidden layers increases exponentially. We'd ideally want this to be as few as possible.



You can in fact show that any Boolean function of n inputs can be represented exactly by a network of perceptrons containing one hidden layer with 2^n perceptrons and one output layer containing one perceptron. So, if you have a hidden layer rather a multi layer perceptron and you have 2^n perceptrons in that hidden layer you can represent any Boolean function of n inputs. How do you prove this?

We are not going to formally prove it, but informally it is fairly simple because every case that you have can be represented as neuron in your hidden layer. Remember, if you have n inputs assuming we are talking about Boolean functions there are 2^n combinations and you can ensure that the right perceptron clicks for each of those combination.

So you can come with a great configuration that ensures that each of those 2^n perceptrons in the middle layer corresponds to one combination of n inputs which automatically will give you your solution fairly straight forward to see it informally a test, but one thing to keep in mind here is while we say that any Boolean function can be implemented by hidden layer with 2^n perceptrons, this is sufficient, but not necessary which means you could solve a problem with less than 2^n neurons too.

For example we just now solve XOR problem which we solved with a hidden layer with just two hidden neurons according to what we said in the next slide we should have needed $2^2 = 4$ neurons, but that is not the case and that is the reason why we say that a network of 2

power $n + 1 + 1$ for the bias is not necessary, but it is sufficient. You can probably find solutions with even less.

But you can definitely find a solution with 2^n neurons or perceptrons in a way. Why does this necessary insufficiency matter? The reason is as n increases the number perceptrons in the hidden layers increases exponentially 2^n . So, your multi layer perceptron can become too computationally intensive to train or even just to take a formal pass through. So you do not want that always to be computationally intensive.

You ideally want to find the multilayer perceptron solution that has the least number of neurons in your hidden layer.

(Refer Slide Time: 10:00)

The slide features a title bar at the top with the text "Going Beyond Binary Inputs and Outputs" in red. On the right side, there are two logos: the NPTEL logo (National Programme on Technology Enhanced Learning) and the IIT Madras logo (Indian Institute of Technology Madras). In the center, there is a grey question box containing two bullet points: "What about arbitrary functions of the form $y = f(x)$ where $x \in \mathbb{R}^n$ (instead of $\{0, 1\}^n$) and $y \in \mathbb{R}$ (instead of $\{0, 1\}$)?" and "Can we use the same perceptron model to represent such functions?". The second bullet point is underlined in red. At the bottom right of the slide, there is a small video feed showing a man's face.

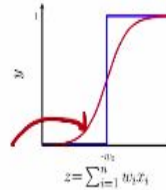
So just ask the question what do we do if you want to go beyond binary inputs and outputs? We only spoke about Boolean inputs, we did say that perceptron could handle the inputs beyond binary, but are there any relationship with the understanding with the previous result that we showed was only that a multi layer perceptron with 2^n hidden neurons can solve taking Boolean function.

What if that function was not Boolean can be used the same perceptron in order to represent such functions.

(Refer Slide Time: 10:40)

Need for Activation Functions

- A perceptron only fires if weighted sum of its inputs is greater than threshold $-w_0$
- Thresholding logic could be harsh at times
- E.g., when $-w_0 = 0.5$, though output values 0.49 and 0.51 are very close to each other, the perceptron would assign different labels to them.
- This behavior is not a characteristic of the problem, the weights or the threshold; it is a characteristic of the perceptron function itself which behaves like a step function
- There will always be a sudden change in decision (from 0 to 1) when $\sum_{i=1}^n w_i x_i$ crosses the threshold ($-w_0$)
- For most real-world applications, we'd expect a smoother decision function which gradually changes from 0 to 1



The answer is we need something called activation functions. Why activation functions we will see that in a moment. So far we noticed that a perceptron only fires when the weighted sum of its inputs is greater than threshold minus w_0 or which was θ . So this thresholding logic can become very harsh at times. For example if your minus w_0 was 0.5 even 0.49 and 0.51 which are very close to each will end up giving very different results because one of them is below the threshold.


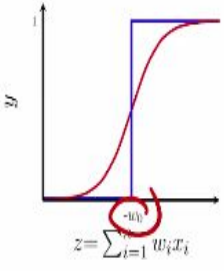
And one of them is about the threshold which means your thresholding function is a step function where you have a sudden change in your output even with a very small change in your input. Typically, this behavior does not occur in the real world, but even in this case the behavior is not a characteristic of the problem. It is about the characteristics of using a step function as a thresholding function.

And we all know that in the real world you generally expect a smoother decision function such as the one shown on red here as you go one value to another value, you do not want to jump up, you do not want the output as the perceptron to be one suddenly when the value goes from 0.49 to 0.5 or 0.499 to 0.5. How do we handle this?

(Refer Slide Time: 12:24)

Sigmoid Neurons

- We could use any logistic function to obtain a smoother output function than a step function
- One form is the sigmoid function:
$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=0}^n w_i x_i)}}$$
- No longer a sharp transition at the threshold $-w_0$
- Also, output is no longer binary but a real value between 0 and 1 which can be interpreted as a probability
- Unlike the step function, this one is smooth, continuous at $-w_0$ and most importantly **differentiable**



The way we handle this is to introduce what are known as activation functions which aim to replace the threshold function that you have with more smoother functions and one early example which was used for several decades is known as the sigmoid activation function and the perceptron or a neuron that uses a sigmoid activation function is known as a sigmoid neuron.

You ideally produce any logistic function which has a shape such as this to obtain a smoother output function than a step function. So one that we are particularly going to talk about here is the sigmoid logistic function which is given input wx which can be expanded this way. The sigmoid function computes 1 by 1 plus e power minus that input that is your sigmoid function which in a graph form has this particular shape.

Clearly here you no more have a sharp transition at a threshold, but smooth transition that goes as your input keeps changing. Also your output now is no longer just binary it is not just 0 or 1 , but your output now can be any value lying between 0 and 1 which could potentially be interpreted as a probability of the output. So which means if you used a sigmoid activation function on the neuron in your output layer.

It would give you a value between 0 and 1 which can associate a probability with whether a point belongs to the positive class or a negative class. So, if your output was 0.5 you perhaps 0.5 let us take another example let us say 0.6 you would say that assuming your input as

patient records you would say this patient has 60 percent risk of say suffering from cancer or heart attack or whatever the problem you are modelling in this particular scenario.

More importantly unlike the step function this function is smooth, it is continuous at minus w_0 which is your threshold it is continuous there as long as continuities and is also differentiable. Why is this important? We will see very soon it being differentiable it is extremely important for how we are going to train these kind of networks.



(Refer Slide Time: 15:13)


Other Popular Activation Functions

Considering $z = \sum_{i=0}^n w_i x_i$

Z

- Sigmoid: $y = \frac{1}{1+e^{-z}}$
- Tanh: $y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Rectified Linear Unit (ReLU): $y = \max(0, z)$
- Leaky ReLU: $y = \max(\alpha z, z), \alpha \in (0, 1)$
- Exponential Linear Unit (ELU):
 $y = \max(\alpha(e^z - 1), z), \text{ where } \alpha > 0$



There are other popular activation functions we will brief you little bit here, but we will cover in detail in a later lecture this week, but a sigmoid activation function we just said is given by assuming your input as $z = \frac{1}{1+e^{-z}}$ and h as $h = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. There is also an activation function called the rectified linear unit which if you see here is the blue line which is the blue line on this particular graph which is given by if your input is z the output is $\max(0, z)$.

So if your input is negative it would give you 0. If your input is positive it give you value itself that is the very popular activation function. There is also a variant of the ReLU activation function called the leaky ReLU which does not make this 0, but keeps this a very small value and we will see each of these being designed for each of these a little later this week.

A more general variant of the ReLU activation function is known as the exponential linear unit or ReLU which is simply a smooth form of w or leaky ReLU activation function which is given by $\alpha \max(z, 0) + e^z$ where α is the number greater than 0 and you can see the ReLU activation function in the total color on this particular graph.

We will see these activation functions a bit later in detail, but all that we are trying to tell here is the activation function is important for us to understand how well multi layer perceptrons can model non binary data, non boolean data.

(Refer Slide Time: 17:12)

Representation Power of MLPs

Theorem: A multilayer network of sigmoid neurons with a single hidden layer can be used to approximate any continuous function to any desired precision. (Also known as **Universal Approximation Theorem**)


Proof: Cybenko, 1989² and Hornik et al, 1989³

Note: Also, refer to [Chapter 4 of Michael Nielsen's online book](#) for visual explanation of Universal Approximation Theorem

²Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals and Systems, Vol 2, pp 303-314, 1989

³Hornik et al, Multilayer feedforward networks are universal approximators, Neural Networks Vol 2:5, pp 359-366, 1989

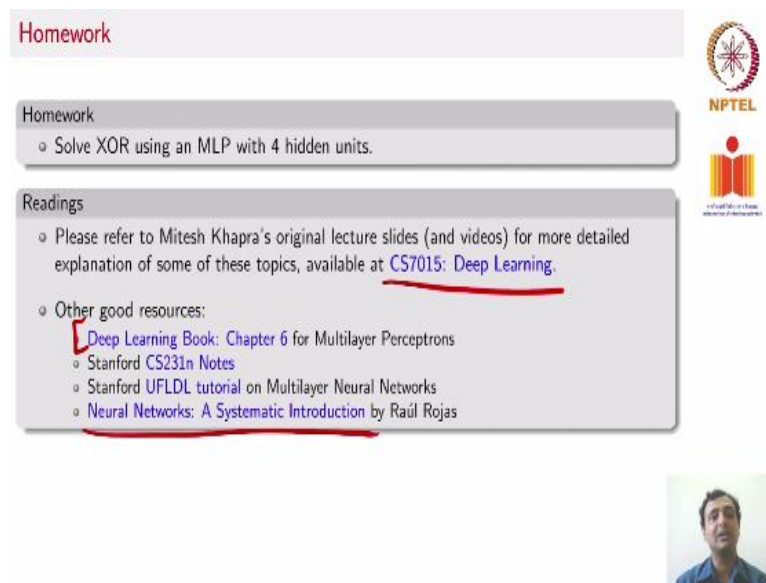
NPTEL



And that is where the representation power or study of the representation power of multi layer perceptions MLPs stands for multi layer perceptrons comes in. A very well studied very well sighted theorem is known as the universal approximation theorem which states that a multi layer network of sigmoid neurons with a single hidden layer can be used approximate any continuous function to any desired precision.

This is a fairly strong statement we are saying that if you give any continuous function we can use a simple multi layer perceptron sigmoid neurons and one hidden layer to approximate that continuous function. We are not going to formally prove it here if you are interested these papers sighted here are good pointers to the proof. There is also a very nice visual explanation of the universal approximation theorem in chapter 4 of Michael Nielson online book on Neural Networks.

(Refer Slide Time: 18:27)



The screenshot shows a slide with two main sections: 'Homework' and 'Readings'. The 'Homework' section contains one bullet point: 'Solve XOR using an MLP with 4 hidden units.' The 'Readings' section contains two bullet points: 'Please refer to Mitesh Khapra's original lecture slides (and videos) for more detailed explanation of some of these topics, available at [CS7015: Deep Learning](#).' and 'Other good resources:'. Under 'Other good resources', there are four sub-bullets: 'Deep Learning Book: Chapter 6 for Multilayer Perceptrons', 'Stanford CS231n Notes', 'Stanford UFLDL tutorial on Multilayer Neural Networks', and 'Neural Networks: A Systematic Introduction by Raul Rojas'. To the right of the slide content are two logos: the NPTEL logo and a logo for an institution with the text 'Institute of Information Technology' and 'University of Hyderabad'.

So your homework for this lecture is try to solve XOR using a multi layer perceptron with four hidden unit come out with your own weights and solve XOR using a multi layer perceptron with 4 hidden units. This should also help you understand the theorem that we spoke about how any Boolean function can be represented by a multi layer perception with 2^n hidden unit this should also help you get an intuition of that.

For further reading please also feel free to refer Mitesh Khapra original lecture slides which are on the website linked here. There are other good resources Deep Learning book which is publically available on website called Deep Learning Book.org is a general resource that we may point to various parts of this course. Chapter 6 is a good introduction to multi layer perceptron.

There is also the Stanford CS231n course which is also a good course whose notes are available here. There is also the Stanford UFLDL tutorial and a very nice introduction to neural networks by Raul Rojas.

(Refer Slide Time: 19:42)

References

- George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems 2* (1989), pp. 303–314.
- Tianping Chen, Hong Chen, and Ruey-wen Liu. "Approximation Capability in by Multilayer Feedforward Networks and Related Problems". In: *Neural Networks, IEEE Transactions on* 6 (Feb. 1995), pp. 25–30.
- Mikel L. Forcada. *Neural Networks: Automata and Formal Models of Computation*. <https://www.dist.ua.es/~mlf/nna/nec/pbook.pdf>. 2002.
- Michael Collins. *Convergence Proof for the Perceptron Algorithm*. <http://www.cs.columbia.edu/~mcollins/courses/6990-2012/notes/perc.converge.pdf>. 2012.



There are some references and we will stop here for now.