Neural Networks for Signal Processing-I

Prof. Shayan Srinivasa Garani

Department of Electronic System Engineering

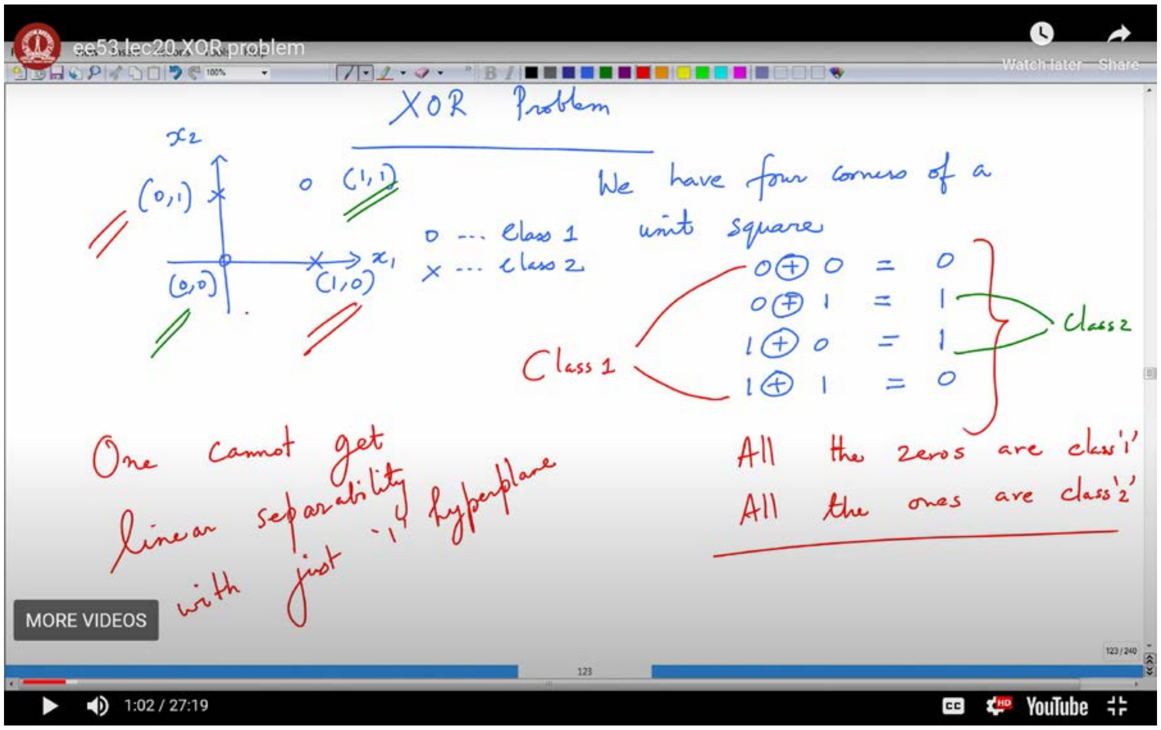
Indian Institute of Science, Bengaluru

Lecture – 20

XOR Problem

Let's explore the XOR problem as an example of how non-linear activation functions can address the issue of pattern separability. In essence, we have a two-class problem, and the challenge is to separate the data points into these two classes using non-linear activation functions.

(Refer Slide Time: 01:02)



XOR Problem

We have four corners of a unit square

o ... class 1

x ... class 2

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

Class 1

Class 2

One cannot get linear separability with just '1' hyperplane

All the zeros are class '1'

All the ones are class '2'

As discussed in the previous lecture, the XOR problem is defined by the following truth table:

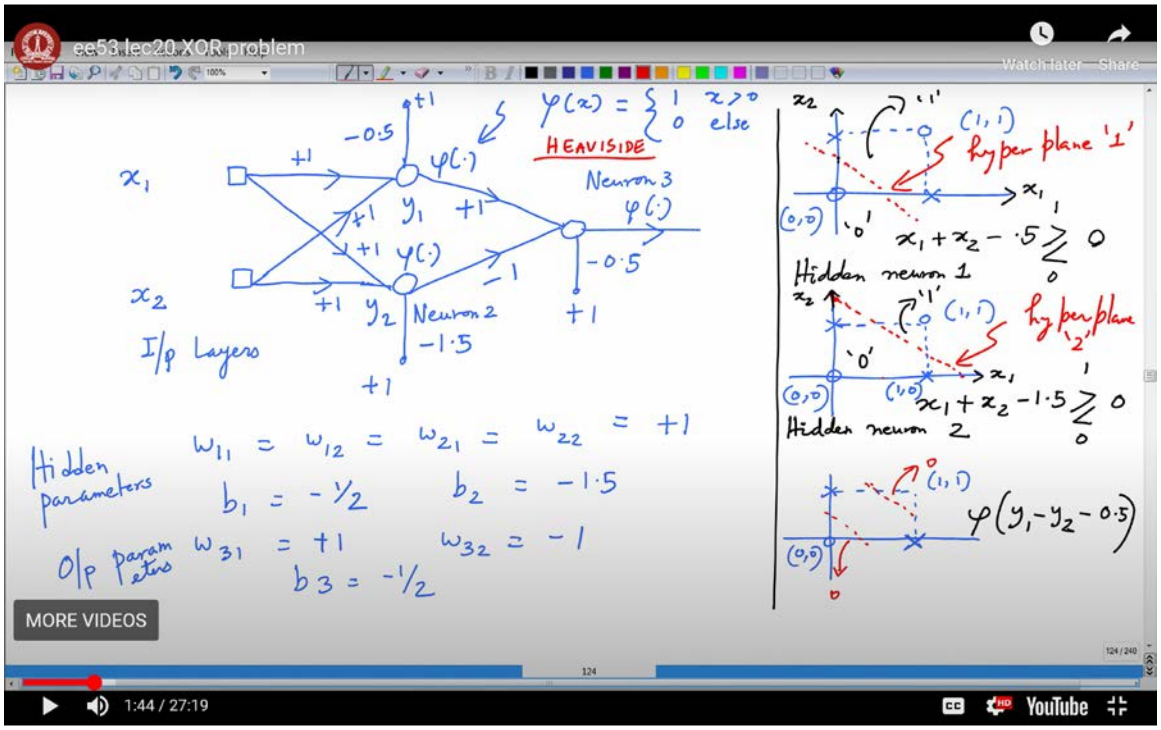
- (0, 0) → 0

- (0, 1) → 1
- (1, 0) → 1
- (1, 1) → 0

Clearly, this set of points is not linearly separable. To tackle this, we use a multilayer perceptron with a hidden layer to introduce non-linearity into the model.

Here's the setup: We start with a hidden layer consisting of two neurons. We initialize synaptic weights connecting the inputs to these hidden neurons, and these weights are fully connected. Additionally, we have synaptic weights connecting the hidden neurons to the output neuron.

(Refer Slide Time: 01:44)



ee53 lec20 XOR problem
Watch later
Share
x_1
x_2
I/p Layers
$+1$
-0.5
$\varphi(\cdot)$
y_1
y_2
Neuron 2
-1.5
Neuron 3
-0.5
$\varphi(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$
HEAVISIDE
Hidden parameters $w_{11} = w_{12} = w_{21} = w_{22} = +1$
$b_1 = -1/2 \quad b_2 = -1.5$
O/p parameters $w_{31} = +1 \quad w_{32} = -1$
$b_3 = -1/2$
$(1,1)$ hyper plane '1'
$x_1 + x_2 - .5 \gtrless 0$
Hidden neuron 1
hyper plane '2'
$x_1 + x_2 - 1.5 \gtrless 0$
Hidden neuron 2
$\varphi(y_1 - y_2 - 0.5)$
MORE VIDEOS
1:44 / 27:19

In this model:

- Neuron 3 is the output neuron.
- Neurons 1 and 2 are the hidden neurons.
- x_1 and x_2 are the input variables.

For simplicity, we initialize all the synaptic weights connecting the inputs to the hidden neurons to be +1. We also introduce two biases: bias B_1 for neuron 1 is set to -0.5, and bias B_2 for neuron 2 is set to -1.5. Let's see how this works.

Consider neuron 1 in the hidden layer. We focus on the computation for this neuron. The activation function used is a Heaviside step function, defined as:

$$\phi(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

For each data point, the input to neuron 1 is computed by taking x_1 and x_2, each multiplied by the synaptic weights (which are +1), summing these products, and then subtracting the bias -0.5. The expression is:

$$x_1 + x_2 - 0.5$$

Applying the Heaviside function, if the result is positive, the output is 1; if it is zero or negative, the output is 0.

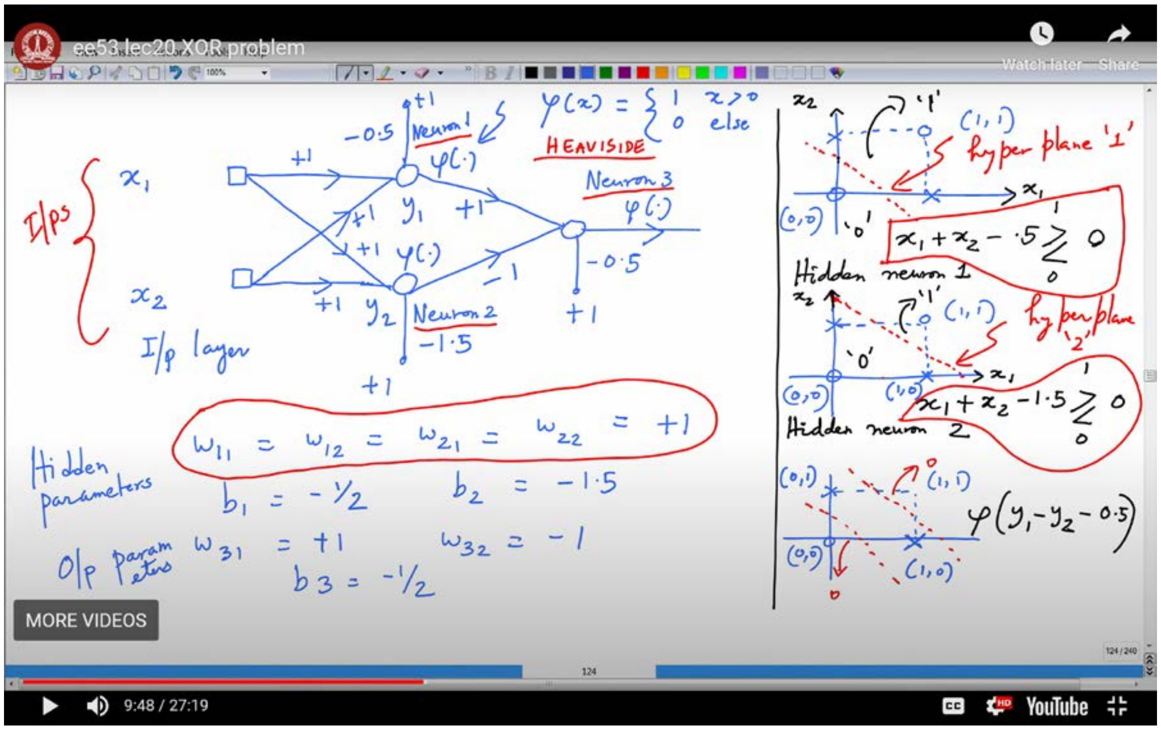
This process is repeated for all data points. For each point, the result of the Heaviside function determines the activation of neuron 1. This non-linear transformation helps to achieve the required pattern separability in the XOR problem.

Let's delve into the interpretation of the equation in question. Pay close attention to the boxed equation here: $x_1 + x_2 - 0.5$. According to this equation, if the result is greater than 0, the output is 1; if it is less than or equal to 0, the output is 0.

Similarly, for hidden neuron 2, the computation is as follows: x_1 is scaled by +1, x_2 is scaled by +1, and there is a bias of -1.5. Therefore, the expression is $x_1 + x_2 - 1.5$. Applying the Heaviside function to this expression, we get a result of 1 if the value is positive, and 0 otherwise.

These computations can be visualized through hyperplanes. Sketching the hyperplanes $x_1 + x_2 - 0.5$ and $x_1 + x_2 - 1.5$, we can observe that the first hyperplane separates the data points into classes where it results in a 0 for some data points and 1 for others. For the second hyperplane, the situation is similar, where the output is 1 for some data points and 0 for others.

(Refer Slide Time: 09:48)



To combine the outputs of hidden neurons 1 and 2, we use their functional signals y_1 and y_2, respectively, and connect them to the output neuron via synaptic weights. There is also a bias on the output neuron. The combined function is:

$$y_1 - y_2 - 0.5$$

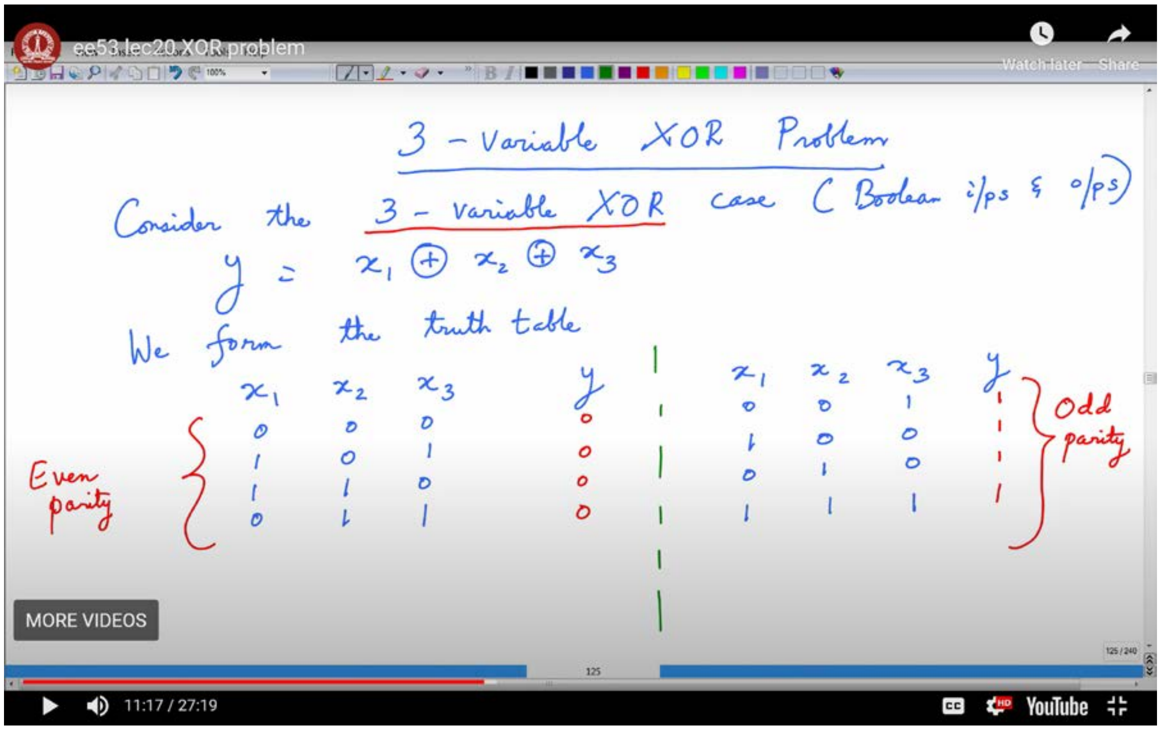
Applying the activation function to this local receptive field, the result is 1 for data points (1, 0) and (0, 1), and 0 for the other two data points. This careful combination of the hyperplanes enables us to achieve the desired pattern separability that was not possible with a single perceptron.

With this understanding of the two-variable XOR problem, we can extend our approach to the three-variable XOR problem. The three-variable XOR problem is a straightforward extension of the two-variable case. Here, the output y is defined by the XOR operation among three Boolean variables x1, x2, and x3. We will form a truth table to represent this extended problem.

Let's discuss how to handle the three-variable XOR problem by leveraging neural networks to achieve non-linear separability. In this problem, we need to differentiate between even and odd parity among three Boolean variables x1, x2, and x3. Specifically, for the tuples (0, 0, 0), (1, 0, 1), and (1, 1, 0), the output is 0, indicating even parity. Conversely, for the tuples (0, 0, 1), (1, 0, 0), (0, 1, 0), and (1, 1, 1), the output is 1, indicating odd parity. This creates a classification problem where even parity corresponds to one class and odd parity corresponds to another.

To visualize this, consider plotting these points in a three-dimensional space. Here, the points representing odd parity will be marked with an "X," while those representing even parity will be marked with bold circles. Clearly, this demonstrates that the problem is non-linearly separable, there is no single hyperplane that can separate the two classes effectively.

(Refer Slide Time: 11:17)



3 - Variable XOR Problem

Consider the 3 - variable XOR case (Boolean i/ps & o/ps)

$y = x_1 \oplus x_2 \oplus x_3$

We form the truth table

Even parity

x_1	x_2	x_3	y
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	0

Odd parity

x_1	x_2	x_3	y
0	0	1	1
1	0	0	1
0	1	0	1
1	1	1	1

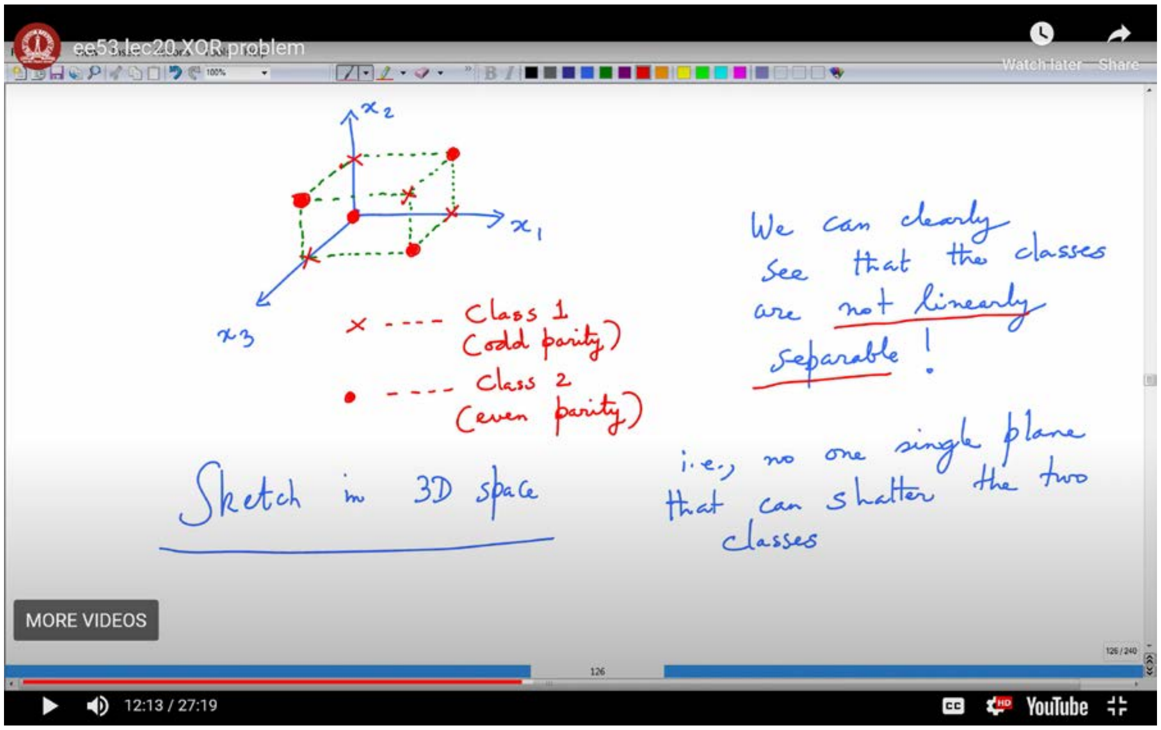
Given this, our approach will be to use a neural network with a hidden layer of neurons to address this challenge. The intuition from the two-variable case suggests that by combining the outputs of hidden neurons, we can achieve the desired pattern separability with just one output neuron.

Let’s construct a neuron designed to respond when at least one of the inputs is 1. In other words, if any of the variables x_1, x_2, or x_3 is 1, the neuron should activate. We start with a signal flow graph where the inputs x_1, x_2, and x_3 are fully connected to the hidden neuron H_1 with synaptic weights initialized to 1. Therefore, the weight vector is [1, 1, 1].

The output of this neuron is given by the expression $\varphi(w^T x + \text{bias})$, where w is [1, 1, 1], x is any Boolean 3-tuple, and the bias is set to -0.5. We will use the Heaviside function for the non-linear activation.

Let's see how this works with a specific example. For the input (0, 0, 0), the weighted sum $w^T x$ is 0, and adding the bias -0.5 yields -0.5. The Heaviside function will therefore output 0, indicating that the neuron does not activate for this input.

(Refer Slide Time: 12:13)



Sketch in 3D space

We can clearly see that the classes are not linearly separable!

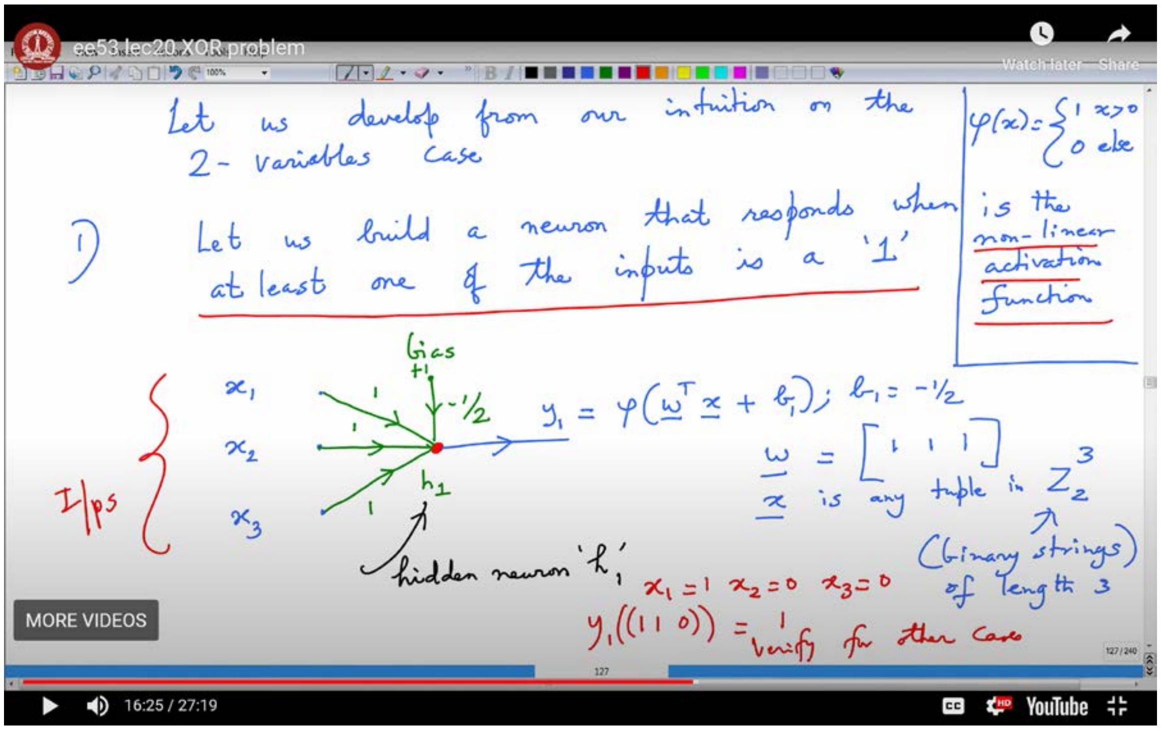
i.e., no one single plane that can shatter the two classes

Thus, by implementing such a neural network and adjusting the weights and biases, we can achieve the pattern separability required for the XOR problem, even in the three-variable case.

To elaborate, if at least one of the inputs is 1, the output will be 1. For example, if x_1 is 1 while x_2 and x_3 are 0, the calculation would be as follows: 1 $\times$ 1 (weight) plus 0 $\times$ 1 plus 0 $\times$ 1, which equals 1. Subtracting the bias of -0.5, we get 1 - (-0.5) = 1.5. Since this value is positive, the activation function outputs 1. Hence, for the input (1, 0, 0), the output is 1. This demonstrates that the neuron correctly activates for any case where at least one input is 1.

Next, our goal is to build a neuron that activates only when at least two of the inputs are 1. This means we need to configure the neuron to respond when exactly two or all three of the inputs are 1.

(Refer Slide Time: 16:25)



Let us develop from our intuition on the 2-variables case

$\varphi(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases}$ is the non-linear activation function

1) Let us build a neuron that responds when at least one of the inputs is a '1'

$y_1 = \varphi(\underline{w}^T \underline{x} + b_1)$; $b_1 = -1/2$

$\underline{w} = [1 \ 1 \ 1]$

$\underline{x}$ is any tuple in Z_2^3 (binary strings of length 3)

$x_1 = 1$ $x_2 = 0$ $x_3 = 0$

$y_1((1 \ 1 \ 0)) = 1$ verify for other cases

Let's adjust the bias to achieve this. While the weights remain the same at 1, the bias adjustment is crucial for the desired response. In the previous case, where we wanted the neuron to activate with at least one input being 1, the bias was set to -0.5. To adjust for at least two inputs being 1, we'll set the bias to -1.5.

Consider the following signal flow graph with inputs x_1, x_2, and x_3, and synaptic weights of 1. We will use a bias of -1.5 for the neuron.

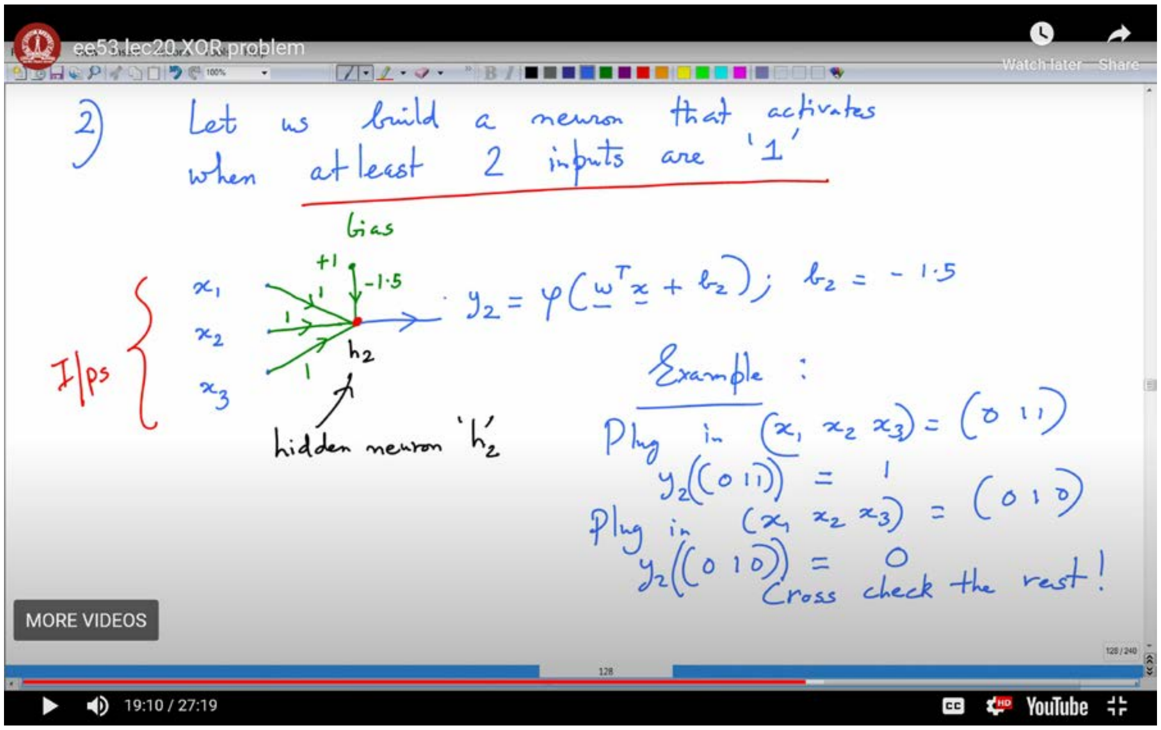
For the input tuple (0, 1, 1), we calculate:

$$\text{Weighted sum} = (0 \times 1) + (1 \times 1) + (1 \times 1) = 2$$

Subtracting the bias of -1.5:

$$2 - (-1.5) = 2 + 1.5 = 3.5$$

(Refer Slide Time: 19:10)



2) Let us build a neuron that activates when at least 2 inputs are '1'

$y_2 = \varphi(\underline{w}^T \underline{x} + b_2)$; $b_2 = -1.5$

Example :

Plug in $(x_1\ x_2\ x_3) = (0\ 1\ 1)$

$y_2((0\ 1\ 1)) = 1$

Plug in $(x_1\ x_2\ x_3) = (0\ 1\ 0)$

$y_2((0\ 1\ 0)) = 0$

Cross check the rest!

Since this result is positive, the activation function will output 1. Thus, for the input (0, 1, 1), the output is correctly 1. You can apply similar calculations for the case where all inputs are 1, and verify that the neuron activates as expected.

This adjustment ensures that our neuron responds appropriately based on the required number of active inputs, demonstrating how careful bias adjustment helps achieve the desired functionality in neural network models.

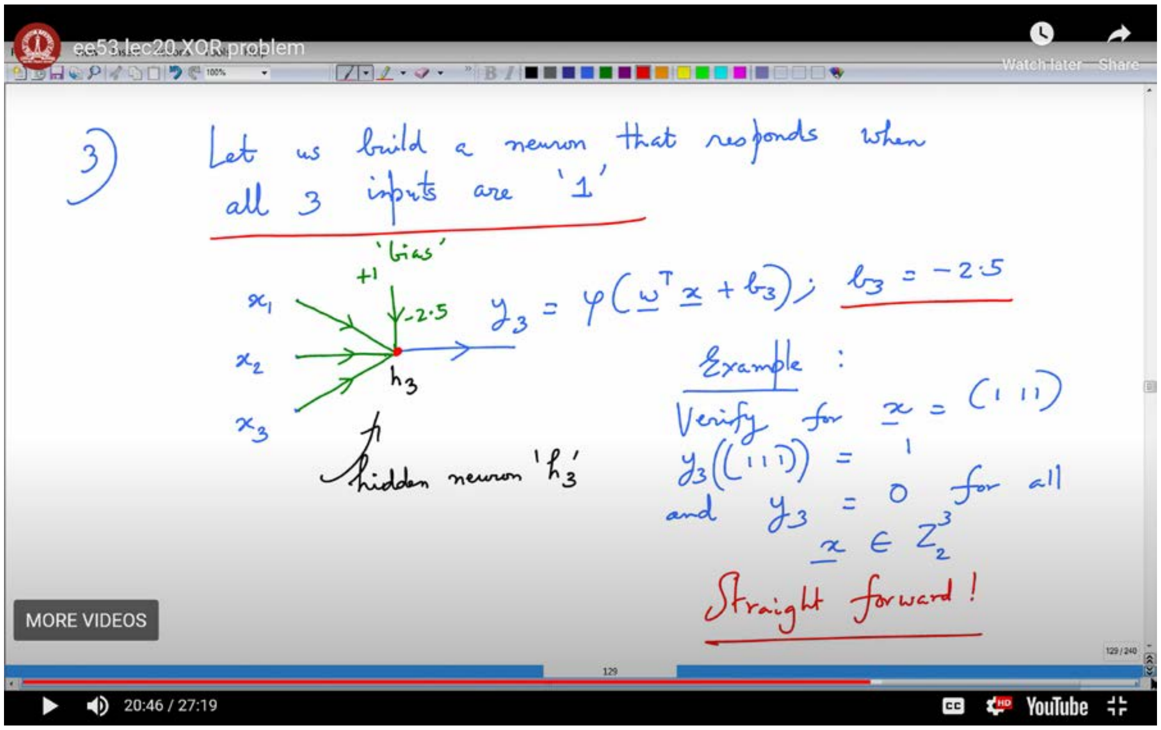
Let's cross-check our work by considering the case where x_2 is set to 1 and the other variables are 0, i.e., the tuple (0, 1, 0). The calculation is straightforward: here, the only active variable is x_2, which is scaled by a weight of 1. Thus, the computation is:

$$1 \times 1 + 0 \times 1 + 0 \times 1 - 1.5 = 1 - 1.5 = -0.5$$

Since this result is negative, the output is 0, which is consistent with our expectations.

With this understanding, if we now want to build a neuron that activates only when all three inputs are 1, we should anticipate that the bias needs to be adjusted. Specifically, the bias should be set to -2.5. This adjustment ensures that the neuron activates only when all three inputs are 1.

(Refer Slide Time: 20:46)



3) Let us build a neuron that responds when all 3 inputs are '1'

$y_3 = \varphi(\underline{w}^T \underline{x} + b_3)$; $b_3 = -2.5$

Example :

Verify for $\underline{x} = (1\ 1\ 1)$

$y_3((1\ 1\ 1)) = 1$

and $y_3 = 0$ for all $\underline{x} \in Z_2^3$

Straight forward!

To verify, consider the tuple (1, 1, 1). The weighted sum is:

$$1 \times 1 + 1 \times 1 + 1 \times 1 = 3$$

Subtracting the bias of -2.5:

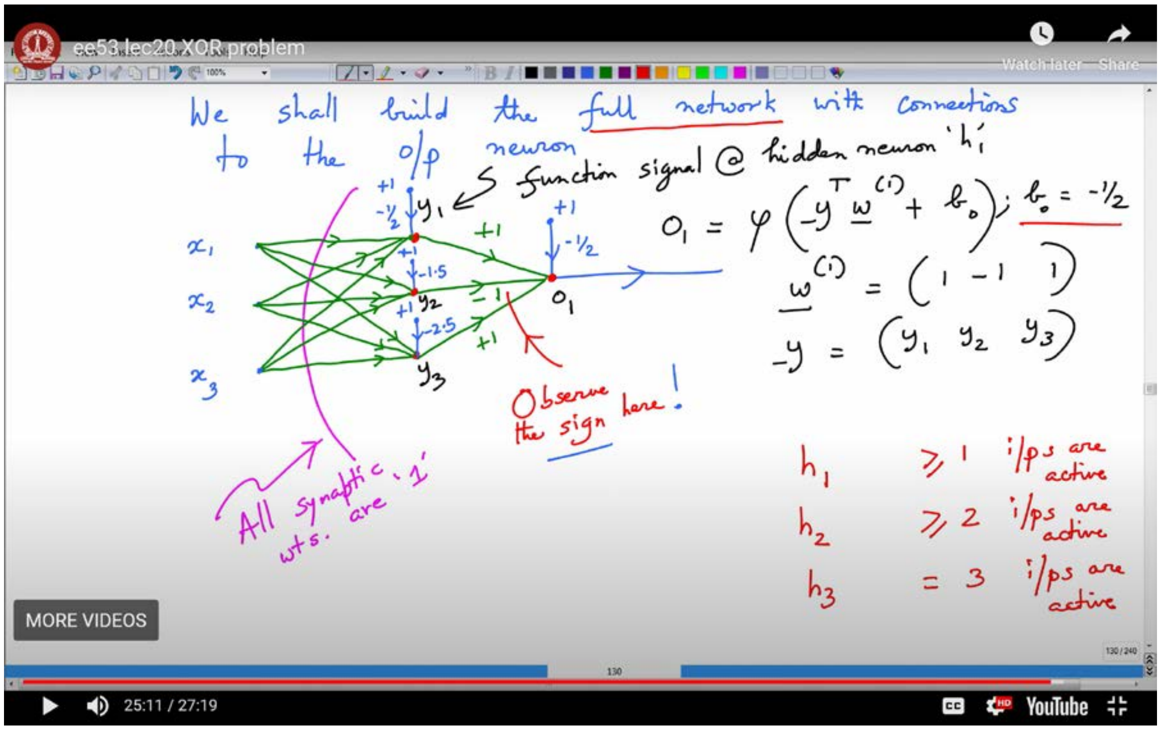
$$3 - 2.5 = 0.5$$

Since 0.5 is positive, the activation function will output 1. This confirms that the neuron correctly activates for the tuple (1, 1, 1). For all other input combinations, the output should be 0.

Now, let's build the full network with connections to the output neuron. We need to connect the functional signals from the hidden neurons to the output neuron. The network involves three synaptic weights that link the hidden neurons H_1, H_2, and H_3 to the output neuron.

The synaptic weights are carefully chosen based on the previous calculations. Hidden neuron H_1 is activated when at least one input is 1, H_2 is activated when at least two inputs are 1, and H_3 is activated when all three inputs are 1.

(Refer Slide Time: 25:11)



We shall build the full network with connections to the o/p neuron

function signal @ hidden neuron 'h_i'

$$o_1 = \varphi(\underline{y}^T \underline{w}^{(1)} + b_o); \quad b_o = -1/2$$

$$\underline{w}^{(1)} = (1 \ -1 \ 1)$$

$$\underline{y} = (y_1 \ y_2 \ y_3)$$

Observe the sign here!

All synaptic wts. are '1'

h_1	≥ 1	i/ps are active
h_2	≥ 2	i/ps are active
h_3	$= 3$	i/ps are active

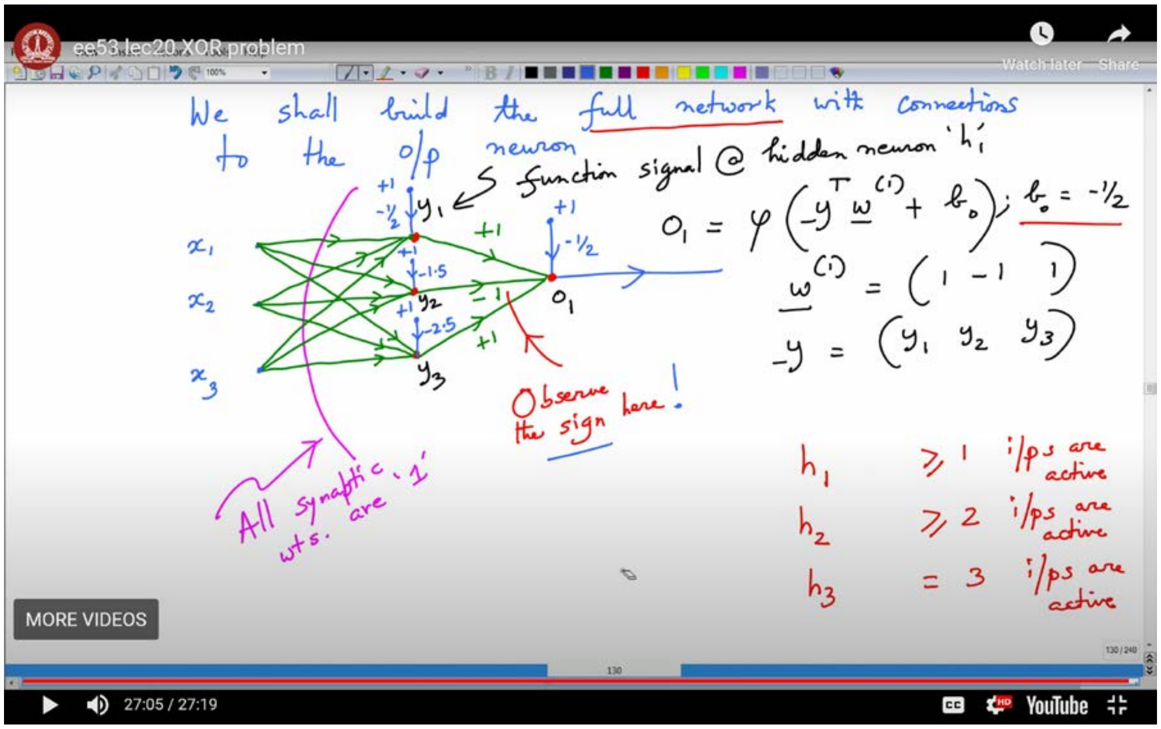
For the output neuron to correctly classify odd and even parity, we must adjust the synaptic weights. Specifically:

- A synaptic weight of +1 connects H_1 to the output neuron. This ensures that if H_1 is active (i.e., at least one input is 1) and the other two neurons are not, the output neuron should respond with 1, recognizing odd parity.
- A synaptic weight of -1 connects H_2 to the output neuron. This helps to ensure that if H_2 is active (i.e., at least two inputs are 1), the output neuron is not falsely activated for even parity.
- A synaptic weight of +1 connects H_3 to the output neuron. This ensures that when all three inputs are 1, the output neuron will activate correctly.

Thus, the output signal of the neuron is:

$$\phi(y^T w + \text{bias})$$

where the bias is set to -0.5, and the functional signals y_1, y_2, and y_3 correspond to the outputs of the hidden neurons. The weights connecting the hidden neurons to the output neuron are initialized to +1, -1, and +1. This configuration ensures that the output neuron produces 1 for odd parity and 0 for even parity, as verified by the truth table.



We shall build the full network with connections to the o/p neuron

S function signal @ hidden neuron 'h_i'

$$O_1 = \varphi\left(\underline{y}^T \underline{w}^{(1)} + b_0\right); \quad b_0 = -\tfrac{1}{2}$$

$$\underline{w}^{(1)} = (1 \;\; -1 \;\; 1)$$

$$\underline{y} = (y_1 \;\; y_2 \;\; y_3)$$

h_1	≥ 1	i/ps are active
h_2	≥ 2	i/ps are active
h_3	$= 3$	i/ps are active

The key to solving the XOR problem with more than two variables lies in balancing the contributions from neurons that respond to different numbers of active inputs. With this approach, we can extend our network to handle a general n-variable XOR problem. This exercise is left to the student or reader as an opportunity to explore further.

You might be curious about why this particular architecture was chosen and why the three-variable XOR problem wasn't tackled by extending a two-variable XOR network. It's possible to construct a three-variable XOR problem from a two-variable XOR gate by adding an additional input, and you could design a neural network to handle this in a cascaded manner. However, using a cascaded approach would typically require more neurons than the example presented here.

The architecture we discussed, comprising just three hidden neurons and one output neuron, is quite optimal for solving the three-variable XOR problem. This efficiency is one of the reasons we opted for this specific setup.

We'll conclude this topic here and move on to the next module in our upcoming lecture. Thank you.