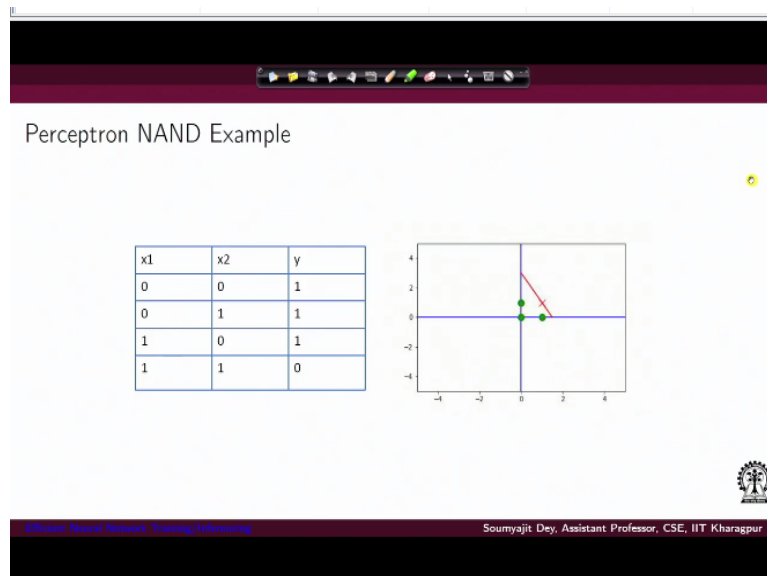


**GPU Architectures and Programming**  
**Prof. Soumyajit Dey**  
**Department of Computer Science and Engineering**  
**Indian Institute of Science Education and Research-Kharagpur**

**Lecture-55**  
**Efficient Neural Network Training/Inferencing (Contd.)**

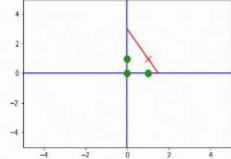
Hi, welcome to the lecture series on GPU architectures and programming.


**(Refer Slide Time: 00:32)**



Perceptron NAND Example

x1	x2	y
0	0	1
0	1	1
1	0	1
1	1	0

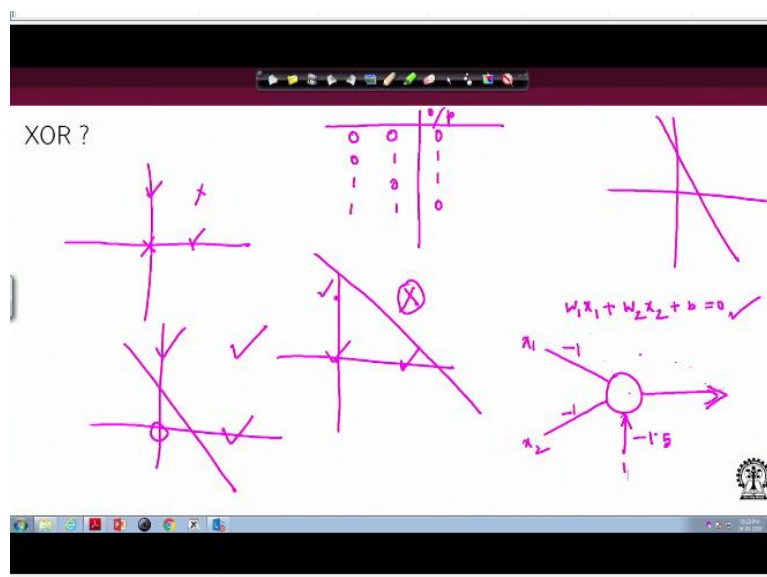




Efficient Neural Network Training/Inferencing      Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

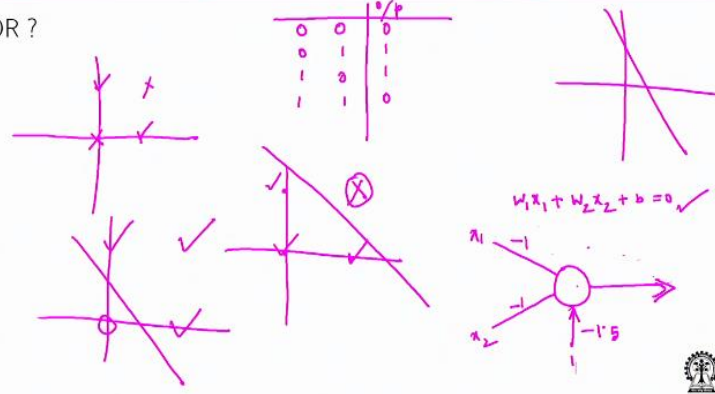
So, in the last lecture we have discussed about this perception example for the NAND function where we found that will, it is separable.


**(Refer Slide Time: 00:37)**



XOR ?

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0





And then the issue was that we are trying to identify what happens if we consider XOR as the function. And so let us start from that example. So if I consider XOR so considering the truth table of XOR, so we have the positive output here, here. And this is the output corresponding to the other class, as we can see that we cannot have a linear classifier for the XOR function because it is fundamentally the function is not linearly separable, right.

So now this brings us the problem that how really are we going to tackle this issue. So getting into the root cause of the problem. Let us first identify that how really does this I mean binary classifiers where the functional form is linear, how really does it work. So in general, if I consider any linear classifier is basically the equation of a line right. So I can simply write, it is an equation of this kind of a form.

So essentially this is I mean this is the way we can capture any such linear classifier which is separating the function and separating the values. But as we see for XOR, this is not possible. However, what about the other situations as we have seen that for NAND is definitely possible. So if we just recollect what happens for NAND. So we will have positive outputs at all these points essentially 0 0 0 1 1 0.

And we will have the NAND function will evaluate to 0 only for the point 1 1 where AND evaluates to 1, right. So that definitely can be classified by a linear classifier. So this is linearly separable. Well, what about OR we can just do a quick check and find out what happens with OR so far, OR I have positive outputs here, here and here. That is 0 1 1 0 1 1. And here I have the negative output.

So this is also linearly separable right. Now, in a similar way, I can figure out what happens with AND also and we can see that for each of these cases we can actually have a linear classifier, right. So the next thing that we will try to do is we will try to figure out whether these kind of linear classifiers can help us to figure out how to handle the XOR function, which fundamentally is not linearly separable, right.

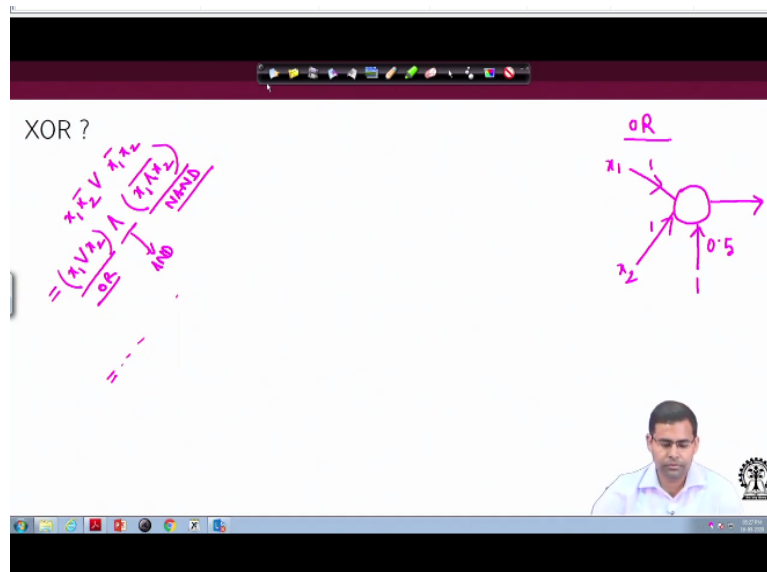
So for that, let us see first whatever it has been our linear classifiers if I can express them in the form of perceptrons, since I mean and that should be technically possible, because they are linearly separable. So, this was about NAND function right. So, since I have a line like

this, I can check the corresponding since this I will get a linear classifier, I can see that well this will give me a line like this, which treatable values of  $w_1$   $w_2$  and  $b$ .

And what will happen is we will have weight  $w_1$  and  $w_2$  as minus 1, here, here and here. I can get the output where I give input of 1 and I have a multiplier factor of - 1.5. So, what we are trying to do here is creating a small perceptron corresponding to this NAND function right. So, we are trying to see that well, since this NAND function has linear separability I can express the in terms of this kind of an equation.

And if I can express it in terms of the equation, I will have the value of  $w_1$  and  $w_2$  and  $b$  as like this - 1 - 1 - 1.5 and then I can create a small perceptron like this where  $x_1$  gets multiplied by the weight - 1,  $x_2$  gets multiplied by the weight - 1 and then I add the bias term which is this constant 1 multiplied by the weight of - 1.5. And then, if I am considering the output, I get the corresponding line which will have this kind of feature right.

**(Refer Slide Time: 05:56)**



Now, similarly, I can just stroke out well for all I can create a similar perceptron. So let me just write it for OR, I will have a similar behaviour with  $x_1$ ,  $x_2$ , getting multiplied by these weights 1 and 1. And here we will add the bias, constraint 1 multiplied by the scaling factor 0.5. And we can easily check that this is going to act as our OR, right. Now when we're talking about XOR well, let us figure out first whether I can actually express it using this OR and NANDs.

Actually, it should be possible. So for XOR as we know I can actually write it as  $x_1$  or  $x_2$  and  $x_1$  and  $x_2$  bar. So, this is my OR, this is my NAND and here I have AND. So all these are kind of linearly separable right. So, if we just break this down, I can have a representation like of course, I mean, you can just check that, in this I can have this thing expressed like this. So, what it does is it gives me an idea that it may be possible for something which is not linearly separable to consider that, well.

I have some intermediate neurons with whom I can actually create this values, right. But what I mean, well, in this case for XOR I am able to create an I mean, I am able to check that I can create a ended version of XOR using ending of OR and NAND's where each of these OR and NAND's are linearly separable. But how does this work out in general, can I say that for any such complex function, which is not linearly separable, I can have linear combinations combining and creating such a value.

Well, that is definitely not going to be true, right. Because following our linear superposition principle, that is really not going to be possible right. So, in general, I need a bit of non linearity in case I am trying to model sufficiently complex functions which are highly nonlinear. It has to be a combination of individual linear transformations and they need to be combined with non linear functionals right.

So, we can say that well in I am creating an version of I mean of XOR where I am considering 2 linearly separable values OR and NAND and combining them with an AND, but in general how does that work.

**(Refer Slide Time: 09:30)**

Building Block of a NN

Figure: Perceptron

- ▶ The yellow nodes are inputs while the grey node is a neuron.
- ▶ The edges (synapses) have weights
- ▶ The incoming value to the neuron is  $f = \sum w_i x_i$
- ▶ The outgoing value is a nonlinear function of  $f$  i.e.  $o = \sigma(f)$

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, the way we will look at it is as we have been discussing that we need a simple some kind of nonlinear function to be combining with linear components. And this is exactly what is done in the real perceptron which forms the building block of a normal neuron network. So, the way we do it is like this, you consider these kind of inputs  $x_1$ 's and  $x_2$ 's, which are marked here by this yellow nodes.

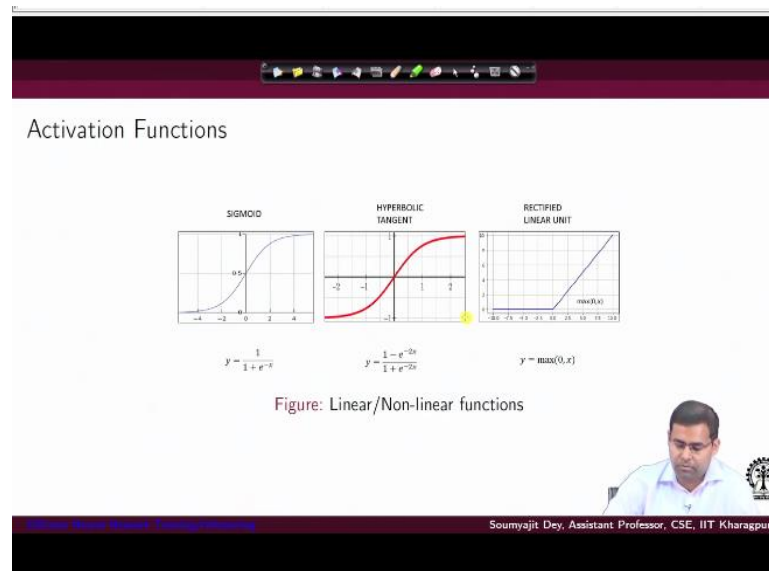
And these inputs you are multiplying by weights  $w_1$  and  $w_2$ . So, that gives you linear transformations, right. You can add the weights, I mean the bias here the  $b$  values, I mean, so, you put a constant and multiply by scaling factor like we have been doing for modeling NAND's and OR's, right, but then again, what you have is those kind of linear expressions,  $w_1 x_1 + w_2 x_2$ , and similarly, many more, and you can have some constants added through a proper bias, but that is not enough.

We do not have any kind of nonlinear component here. So, fundamentally, what we will do is well, will add our nonlinear transformation here, which is known as the activation function. So what we do is we will have edges or synapses like this, right, so these are my edges or synapses modeling, the neurons synapses they are kind of modeling the linear transformations, and then add these nodes, we have neurons enabled with their corresponding activation functions.

For example, here we have a picture of one activation function, right. And what we can observe here is, these are nonlinear function, all it is doing is taking the input and mapping the output corresponding to this kind of a functional map that is provided here. So, in this

case, you can see that this is not a line, right, so it is a nonlinear function. So, what we get here is let us call it a some sigma and the final output is sigma composed with f and f is the linear transformation.

**(Refer Slide Time: 11:51)**



Now of course, this the function we have here is not the only possible nonlinear activation function there can be many others. So, when we say that this neuron fires that means this function will have the input to the output, right. So, this example that we have here is for sigmoid for which essentially the closed form expression is like this  $y$  equal to  $1$  by  $1$  plus  $e$  to the power  $-x$ .

So, as you can see it has a  $0$  crossing at  $0.5$  for this input  $0$  you get the output as  $0.5$  beyond that the function saturates to  $1$  and  $-1$  as you go forward, right. So,  $1$  and  $0$  as you go forward, right. So, of course, if you are putting the  $x$  value is very high, essentially this exponential term will decay and you get a  $1$  you put the  $x$  value as very low, as very high but negative, then again this will become very positive  $e$  to the power  $-x$ .

So, then again what you get is that the  $y$  will come down to  $0$ , right so, the highest possible value is approximate  $1$  asymptotically, the lowest possible value is as intrinsically reaching  $0$ . And at  $0$  at  $x$  equal to  $0$ , you have  $0.5$ . So there is a sigmoid function. And in general, you can have other kinds of nonlinear functions. For example, you can have dysfunction, which is the tan hyperbolic function.

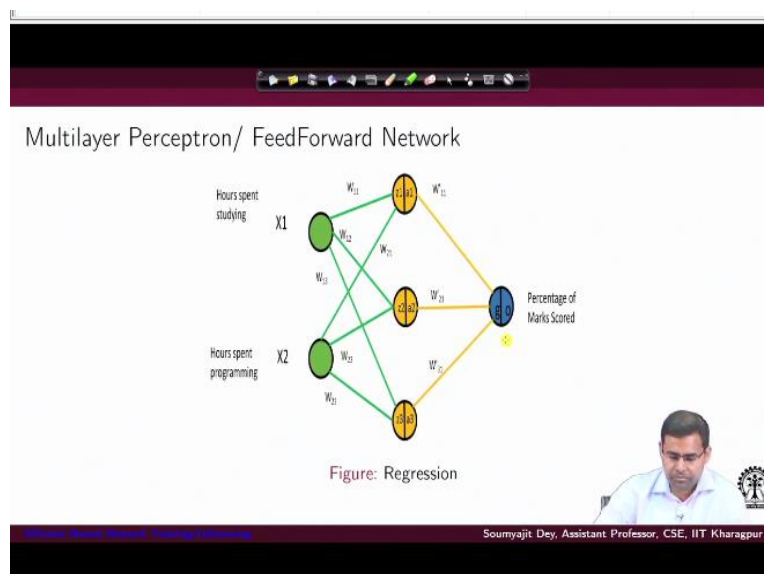
As you can see, it is a different functional with respect to sigmoid, here we have the closed form functional representation. And also as you can see that the value you are essentially having  $z$  I mean as central 0 and asymptotically with positive and negative values you are approaching 1 and - 1 respectively. The other very commonly used activation function is value or rectified linear unit.

So essentially is just a max between 0 and the input value. So if your input value is positive, that is what you get as output. So that is approximate that is kind of modelled by this linear this, slope right, and if your input value is negative when you get a 0, so, that is why you have for all negative values you have essentially on the x axis. So, these are the commonly used activation functions in neurons.

Again, what is the objective here, we are trying to model highly complex nonlinear relationships, the way we are doing it is we will create a neuron network built using these kind of perceptron blocks, where the neuron is essentially modeling a nonlinear activation function the inputs are modeling linear combinations with bias and we are assuming that with this kind of combinations, we should be able to come suitably approximate any complex nonlinear function.

And also as you can see, these functions satisfy some nice properties like differentiability, so that it helps you in to computing the weight something we will see how this kind of specific functional forms really help you.

**(Refer Slide Time: 15:00)**



So, with this background, let us come to the genetic structure of a multi layer perceptron or a feed forward network right. So, what is the motivation behind this kind of multilayer perceptron as we have seen that their OR functions which are definitely not linearly separable, we will like to combine them with suitable linear maps along with non linearity and eventually we like to get outputs right.

So, in that way for modeling complex behaviours, we will have inputs going inside to neurons which are there in the hidden layer and then they map the outputs to the output layer. And inside in each layer you have neurons which are connecting with the inputs or the previous stage neurons. And whatever set of layers you have in the intermediate between the input and output layer.

They are the hidden layers and in deep neural networks, I mean all that difference is you have significant depth in the hidden layers, how that helps is you are able to combine a lot of linear and nonlinear activation functions in different possible ways. So that, that gives you the flexibility to capture many possible complex functional relationships, right. So, these are an example taken from source called Welch labs, which is in our YouTube.

And it provides a very nice example, that suppose, you are trying to create a correlation between hours spent studying an hour spent programming along with what is the probability that you are going to pass or fail right you are going to model this kind of relationship. Well, this is an example there could have been many other examples. So, we are trying to propose a simple neural network architecture here.

So, from the inputs, you have this weights. So, since from a 2 input system, you are mapping to a 3 neuron hidden layer and as you can see, you this is like a fully connected system here that means every hidden layer neuron has its connection to all the inputs in the input layer right. So, you just denote these connections by the weights  $w_{11}$  from  $x_1$  to  $f_1$ . Similarly,  $w_{21}$  from  $x_2$  to  $f_1$  right like this.

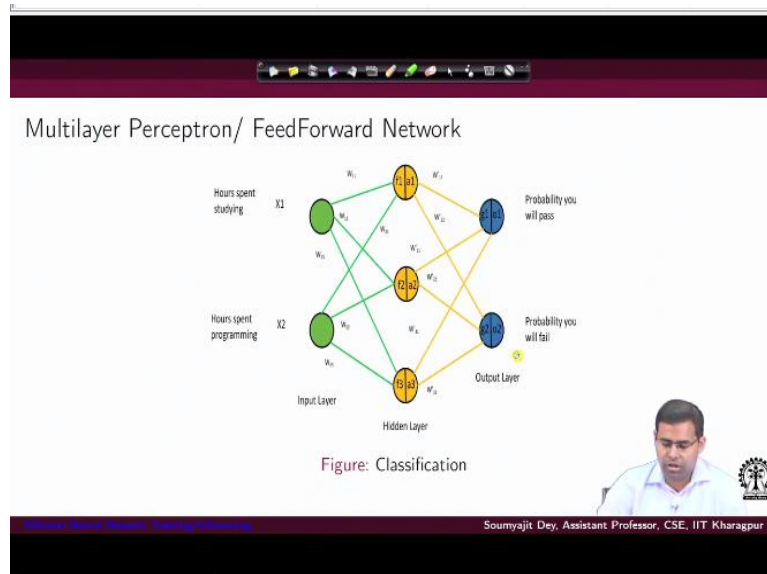
Then  $w_{22}$  from  $x_2$  to  $f_2$   $w_{23}$  from  $x_2$  to  $f_3$  like this right. So, here, what you have is after you get the values here from the synapses where you get  $x_1$  multiplied by  $w_{11}$ , you have the activation functions  $a_1$   $a_2$   $a_3$  like this, through which the outputs will pass once that



neurons fire and then you are again multiplying them by these weights, the output layer weights  $w_{11}$   $w_{12}$   $w_{21}$   $w_{22}$  prime, so and so forth.

To get into the output layer, where again you have this nodes denoted by  $g_1$   $g_2$  like that and finally, you have the output activation layer with  $I$  mean nodes denoted by  $o_1$   $o_2$  like that, right.

**(Refer Slide Time: 18:16)**



So, this is a sample architecture for a classification system. So, essentially you are doing a 2 class classification is for whether you are going to pass or whether you are going to fail like that. And then, what about regression. Well, like we discussed earlier regression is basically about computing the continuous function which is the approximation idea we have been talking about.

So, that would mean at the output you have only one node in the output layer and that gives you a value in a continuous range right well. Whereas, in classification, you get values in all of this and whatever is the maximum you will like to come to consider that as the  $I$  mean the classification well, the definitions can vary as well your architectural right. But overall we can say that these are generic structure for classification.

We have multiple outputs, and you are computing of real value here and here you have only one output for that because you got you want a functional output, right.

**(Refer Slide Time: 19:15)**

Note

- ▶ The objective of this course is to get you acquainted with the computation involved while training and testing a neural network.
- ▶ We shall not discuss core ML principles which should be followed while designing neural networks for various problem domains.

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

Now, what is the objective in our case, again, we will try to focus on that we are not going into that deep theory of neural networks and all that, rather than that, we are trying to focus on what is the exact computation involved while training and testing a neural network, right. So of course, we are not going to discuss core ML principles that need to be followed and all that we are trying to consider that suppose I am given this network.

My objective is to train the network figure out by training we mean figuring out these values of these weights, given a test data set, so that we get a set of values which best approximates the functional relationship underlying the test data set, and, and how that computation can be accelerated by the GPU architectures and all that.

**(Refer Slide Time: 20:03)**

Neural Networks

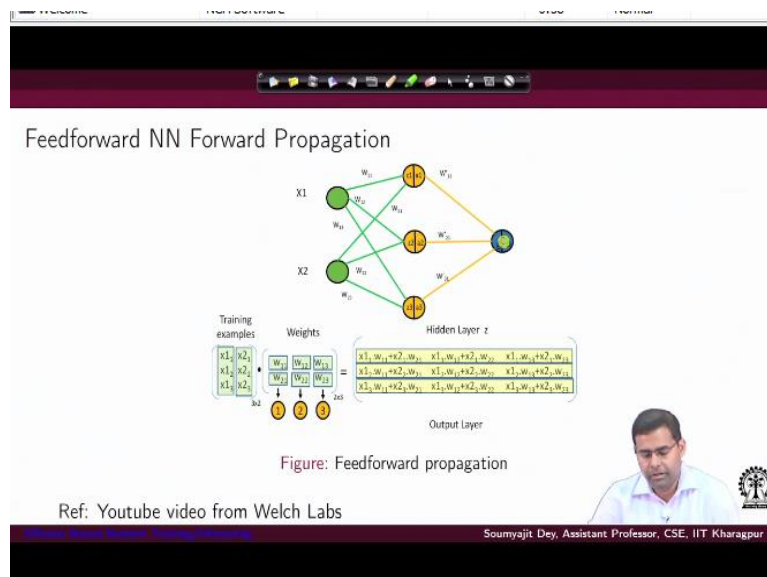
- ▶ Each neuron of a layer accumulates a weighted sum of the inputs from the previous layer.
- ▶ Each neuron applies an activation function to its input and propagates the output to a neuron of the next layer..
- ▶ This results in a series of linear and non-linear transformations from the input layer to the output layer.
- ▶ The predicted output value for every input example is a function of the input feature values and the weights and activation in the network

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, just doing a short summary here. So, for neural networks, what we have is each network has I mean, you have a set of neurons, and each of these neurons have a layer they accumulated the weighed sum of inputs from the previous layer and then a neuron fires that means, it applies its activation function and it will simply propagate the output of the activation function to the neurons of the next layer, right.

So, essentially that we in that way, we have a series of linear followed by nonlinear transformations going on the inputs and propagating finally, to the output layer. Why is this necessary, because, as we discussed earlier, they are in real life you have many complex functions which are not linearly separable. So, the way we like to approximate them is through a combination of linear active linear maps, followed by nonlinear transformations and again linear maps like that, right.

**(Refer Slide Time: 10:56)**



So let us now come to the underlying computational parts here. So let us consider this simple system of feed forward neural network forward propagation. Okay, so let us see how things are going on here. So we have this training example where the input is like this. So you have a set of inputs, like for  $x_1$ , you have these inputs, like  $x_{11}$ ,  $x_{12}$ ,  $x_{13}$ . So that is the first input pair.

Again, the second input, I mean, the second one you have is  $x_{21}$  and  $x_{22}$ , and then the third one is  $x_{23}$ , and  $x_{24}$ , right. Now, so for this training examples, they are going to get multiplied by the weights. Now, as you can see, this is the weight collection. Essentially, it is

a 3 I mean, you have a 3 cross 2 matrix, because you have weights,  $w_{11}$ , I mean, because each of them are mapping to this like this right.

So, as you can see, you have got 6 weight components, but how really are they arranged. So, for  $w_1$  you have 3 components because  $w_1$  maps to the first neuron in the hidden layer second and  $w_{12}$  and  $w_1$  also has a component mapping to the second neuron in the hidden layer and similarly,  $w_1$  also has a component mapping to the third neuron in the hidden layer right.

So, these are the essentially the components of  $w_1$  right. Similarly, you have 3 components for  $w_2$  right and then when you are considering the input, so, for this  $x_1$  and  $x_2$ , you have these training examples right now, when you are creating these multiplier you want the final value to be propagating to the hidden layer right. So, like since  $x_1$  has these components right, you have you must mean the way this is arranged that means, for each of the values you are going to for each of the training examples.

You are going to have values for these kind of couples, right and they are going to get multiplied with the corresponding components for the  $w$ 's right. So, this gets multiplied with this column and similarly, right for the next row, you have it multiplied with the second column like that, and in that way, you are going to get the corresponding entries. So, the first row and the first column multiply to give you the 1 1 at entry here, right.

So, that is what you get so  $x_{11}$  multiplied by  $w_{11}$  plus  $x_{21}$  multiplied by  $w_{21}$ , and then you have the second  $x_{12}$ , multiplied by  $w_{11}$ . I mean, if I consider the next multiplication like the one I highlighted here, so this is going to give me so the second row and second column, that is essentially going to give me this entry right. So  $x_{12}$ , multiplied by  $w_{12}$  plus  $x_{22}$  multiplied by  $w_{22}$ , right.

And in that way, I can easily figure out all these values that are slowly getting computed here. So that so using the training examples and the weights, I can have these values propagating to the net towards the output layer. Of course, these values now, we will have to pass through the activation functions before they get mapped to the before they get further multiplied by the weights and propagate to the output layer right.

**(Refer Slide Time: 25:06)**

Feedforward NN Forward Propagation

Figure: Feedforward propagation

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

So, let us see how that happens. So, here you have the inputs for the hidden layer right. So, these are the inputs for the hidden layer and here they get they need to be I mean computed right I mean, so, you have this is the output here for the hidden layer and then this hidden layer output gets mapped to the activation function some sigma and when this activation function applies out of that, you are going to get a set of activations which we have noted here like this a 1 1 2 like that, right.

This is what we're trying to show is what happens after the activation functions get applied here. So essentially, as you can see that for a 1, you will have this next this like that, so and so forth.

**(Refer Slide Time: 25:52)**

Feedforward NN Forward Propagation

Figure: Feedforward propagation

Ref: Youtube video from Welch Labs

Soumyajit Dey, Assistant Professor, CSE, IIT Kharagpur

And then at the output layer, you have these inputs coming out of the activation functions of the hidden layer. So, again let me repeat what we just did. So, we have these training examples getting multiplied by the weights right. So, as you can see, since we are considering this 2 input system  $x_1$  and  $x_2$  and we have weights arranged like this, so, this is how one instance of the training examples will get propagated, right.

Since the weights functionally form here a 3 cross 2 matrix you multiply it like this to get components flowing through the interaction right. So, since you are going to have this I mean since you are going to really have this kind of a setup, let you have these many values propagating through the layers. So, at each point here, you are getting you are receiving values like this.

So, let us say for  $x_1$  it gets multiplied by  $w_{11}$  and then you have  $x_2$  getting multiplied by  $w_{21}$ , and they are getting summed up here. And then after activation is going to move on here, right. Similarly, if I do get the other points like  $x_1$ , getting multiplied by  $w_{12}$ , and, I consider the other combination of  $x_2$ , getting multiplied by  $w_{22}$ , and it is getting received here.

So, in that way I can say that each of these neurons, they are receiving the different possible combinations, right. So this guy is going to receive a combination of from  $x_1$  and  $x_2$  multiplied by  $w_{11}$  and  $w_{21}$ , right. This is going to receive a combination from  $x_1$  and  $x_2$  multiplied by  $w_{12}$  and  $w_{22}$ , right. And similarly, let us just take another example. This one is going to receive from  $x_1$  and  $x_2$  and it is getting multiplied by  $w_{13}$ .

And the other guy is getting multiplied by  $w_{23}$ , right. So similarly, let us just consider one such example again, so four  $x_1$ . And, of course that is going to happen for all the different components of the values, right. That is also going to happen for all the different components of the values. And the components are like as we have for  $x_1$ , we have 3 components. For  $x_2$  we have 3 components, and so on, so forth right.

So for each of these components, I will have this kind of combinations, right. So for the first set of components, I have the combination  $w_{11}$ ,  $w_{21}$ , then  $w_{12}$ ,  $w_{22}$ ,  $w_{13}$ ,  $w_{23}$ . And this is, this is like what gets propagated in each of them, right, so for the first activation

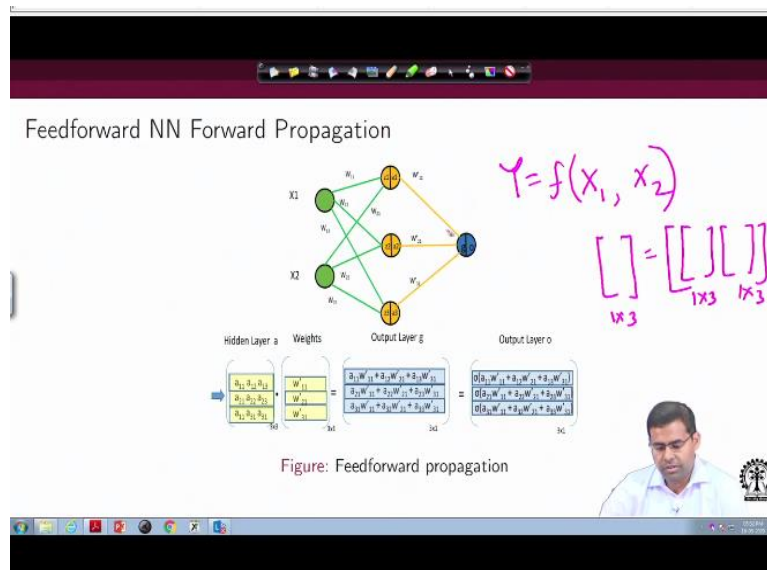
function, this is the part that is getting propagated for the second activation function here. This is the part that is getting propagated.

And for the third activation function here, this is the part that is really getting propagated. I hope that is clear from this. And that is really happening because we are also considering that the inputs they are in this kind of a vector form right here it means something different, I mean for example, if the inputs are scalar that would be a different kind of computation. So, you have to really take that into account right for  $x_1$  and  $x_2$  both of them we are considering that will the inputs are arranged like this kind of I mean size 3 vectors right.

$x_1$  is the size 3 vector  $x_2$  is a size 3 vector and then for them I have this kind of values that are getting propagated. And after that when these values are propagated, and then I have these entire thing flowing through the network and then is reaching these activation functions and then the activation functions modify I mean they do suitable thresholding and they will create outputs.

For example, for the first one I will get this a  $11$  a  $21$  a  $31$  it should be like that, a  $12$  a  $22$  I mean, so, this one let me just correct it here. So, this should be a  $31$  a  $32$  a  $33$  like that right.

**(Refer Slide Time: 30:21)**



So, once this activation values reaches then what do we really have. So, these are the values that are flowing out of the network right and now, they will get multiplied. So, this is what is coming and they now get multiplied by the output weights here right. So, what are the output

weights here, so, as you can see now, since I mean here we are concentrating the regression example, so, everything is going to marginal right.

So, from the hidden layer to the output layer we have single connections and they are represented by  $w_{11}$  prime  $w_{21}$  prime  $w_{31}$  prime, 1 represents a single neuron and output and 1 2 3 the first index denotes that different elements in the hidden layer, right. So here we have the hidden layer outputs, they are going to get multiplied by these weights of the synapses that the connection and the hidden layer to the output neuron here.

And then what we have is these vectors of a 11, a 12, a 13, I mean they are I mean and a 21 a 22, a 23 and so, like these, this entire 3 cross 3 thing is getting multiplied with these corresponding weights here. So essentially here, what you have is a combination like this. So what is very propagating. So, you have a 11 multiplied by  $w_{11}$  prime plus a 12 multiplied by  $w_{21}$  prime, a 13, multiplied by  $w_{31}$  prime these values.

So considering the first components, right, considering the first components that are coming out, so, that is what is coming out here right. So, this is the first component, this is the set of first components that are coming out right for each of them and so, here what we have is all the components now, from each of them whatever is the first component that would get to propagate through this network.

And then you have at the output these kind of combinations. For example, what is the second element in the output layer, it should be a 21 multiplied by  $w_{11}$  prime plus a 22 multiplied by  $w_{21}$  primes so on so forth. So, if we just again mark it out, so, this and this will give you the first element and similarly for the others. So, with this we have all outputs flowing to the output neuron where the corresponding activity function is now getting I mean, that is indicated here by the sigma and you apply the sigma here.

So, then you get the final output from the output layer, which will be again a vector of size 3. So, here again just to summarize, we are considering that we have inputs which are kind of 3 size vectors. So essentially we are trying to learn a relationship like this. So  $x$  1 x 2. So f of this we are trying to learn it like this where this output is a vector of size 1 cross 3 and the inputs are all 1 cross 3s, right.



So that is the kind of relationship we are trying to learn here. And accordingly, the system is tune, as you can see for the input weights, they are all having 3 components here. And for the output words, they are all having since I am finally mapping to a single neuron, they are all having single weights. So, I mean, we are not getting into further details of I mean, what is the motivation and all that.

We are trying to cover it in simplistic terms more in terms of what is the underlying matrix multiplication relationships that are going to happen right. So with this, we will end this lecture. And in the next lecture, we will see how this really lends to the neural network training part and how I mean what are the underlying computations that need to be done for the training part. Thank you.