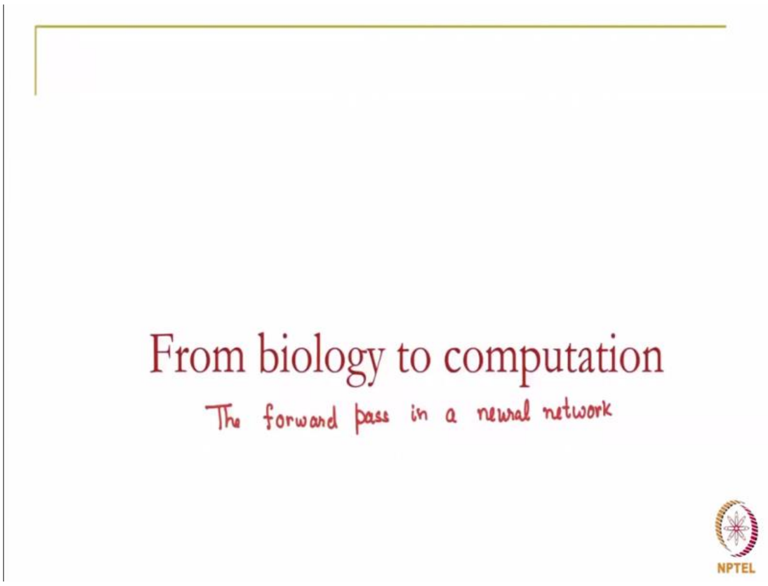
Inverse Methods in Heat Transfer
Prof: Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology – Madras

Lecture - 53
Forward Pass through a Simple Neural Network

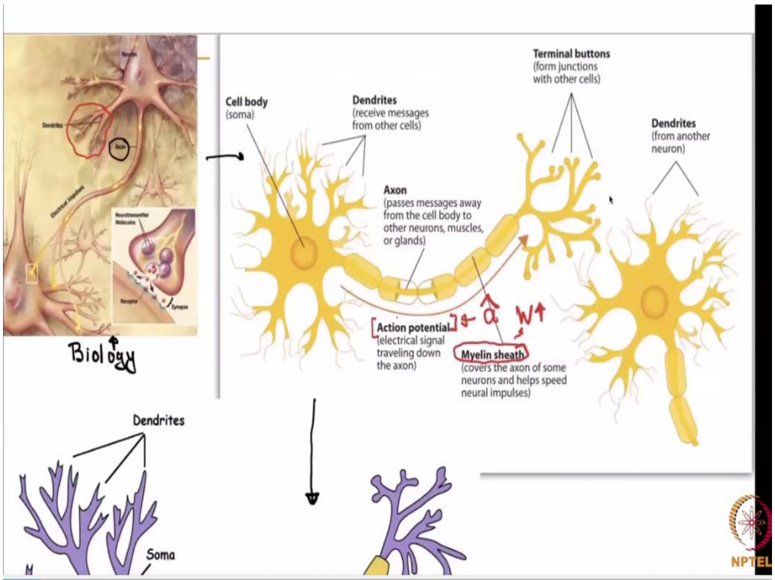
(Refer Slide Time: 00:19)



Welcome back. This is Week 10 of Inverse Methods in Heat Transfer. We are in the second video. In the last video I had talked briefly about why we try to use deep networks giving you a simple example of the XOR gate. In this video, we basically aim to go through a forward pass, I had talked about the forward model of the neural network we want to do that. I will also give you brief introduction this is going to be very brief and possibly slightly inaccurate on what the idea is.

I mean why does the neural network have any relation whatsoever to biology or to our brain. I do want to point out that in my personal opinion this analogy between a brain and a neural network is quite stretched. Nonetheless, this was the historical reason how neural networks came to be and by now I take a very at least in this course, I am going to take a view that neural network is simply a function approximation like a Fourier series or something of that sort.

(Refer Slide Time: 01:23)



But nonetheless since there are biological reasons for this, let us take a look at that. So, when you look at the biological picture here it is on the left-hand side. So, something like this basically exists within our brains quite a complex picture, you have a whole bunch of cells and there are electrical impulses whenever you have a thought when you see something there is an electric impulse that moves in a specific direction.

So, this thing this line in the middle is somewhat of the concrete example of what we draw as weights and the electrical impulse is basically our forward propagation on what we basically simulate as if it is a computation and when you look at it you have the cell body and these dendrites are some kind of network connections, we will see them as network connections which receive messages from other cells.

Axons pass messages away from the cell. Now within this there is something called the action potential which is the electric signal which travels down an axon and when it comes to the next cell it either activates it or it does not. Now one thing which we do know is something called neurons that fire together, wire together this kind of information you would have probably heard.

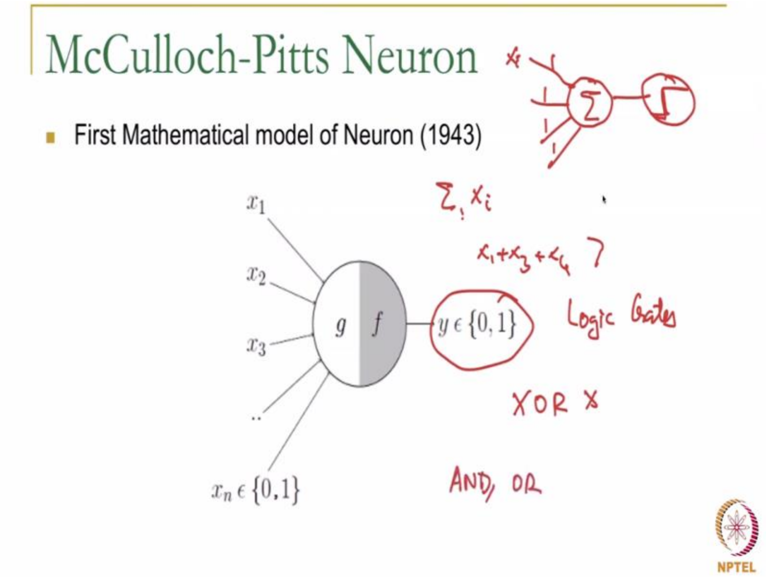
These messages are mediated by these connections that go in the middle and you have something called the myelin sheath which you can see here. This myelin sheath basically tends to reinforce or and make communication easy between two cells that fire together very frequently. So, the idea is supposed to be similar to increasing the weight between one neuron and the other neuron in case it turns out that the connection is really strong.

So, these are some basic ideas that people took the idea of weights in the middle of two neurons. The weights can actually grow stronger or weaker just like myelin sheath enables and maybe other biological mechanisms enabled and the fact that even if something hits this need not fire fully. As I will show you later and as I have shown you earlier also just because things sum up through all these connections.

So, you can see these as incoming weights to this neuron and just because they sum up together to a certain value does not still mean that it has to fire. There might be a step function after a certain amount of activation has taken place then the neuron might fire otherwise it might refuse to fire just like a human being has to be shaken up a little bit before they wake up.

Now all this is just a brief and like I said a slightly inaccurate description of biology because I am not a biologist obviously. So, here is looking at this kind of picture something similar to this is the first attempt that was made for neural networks. I have not covered the history of neural networks here it is the course is not meant for that, but as you can see it is at least a history.

(Refer Slide Time: 04:33)

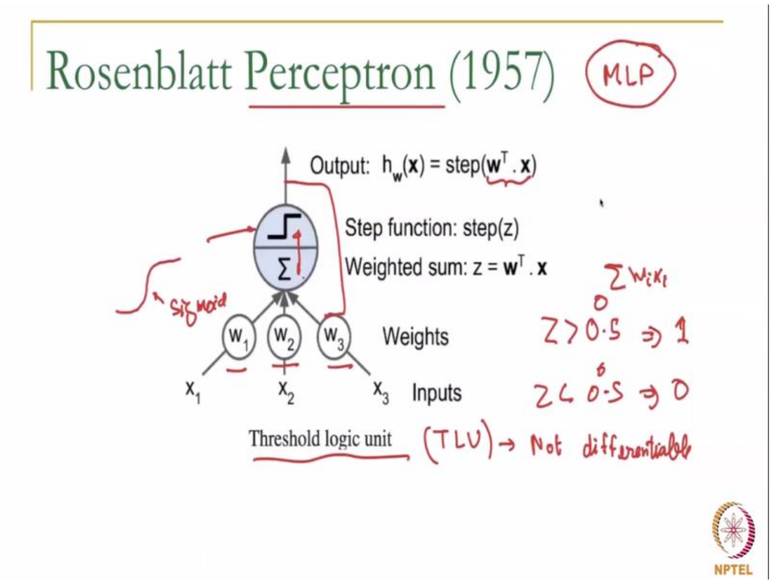


So, the first mathematical model of a neuron was like this you will see it looks remarkably similar to what we are doing except all we do here is this was called the McCulloch-Pits Neuron. You will basically have $\sum x_i$. Notice there will be no w_i here you simply sum up $x_1 + x_2 + x_3 + \ldots$ etcetera and you just check whether it is greater than a specific threshold.

So, now our current notation we will simply do a sigma all these weights are 1, these are xi you sum these up and then you run through a step function that is basically the McCulloch-Pits Neuron. It was the first mathematical model as you can see the output is either 0 or 1. The idea was to simulate simple logic gates and as I told you put together an entire circuit based on these simple logic gates.

Once again you can simulate AND or OR etcetera, but you cannot simulate XOR with this kind of behavior you really cannot, but you can get AND, you can get OR and stuff like that with this kind of neuron.

(Refer Slide Time: 05:50)



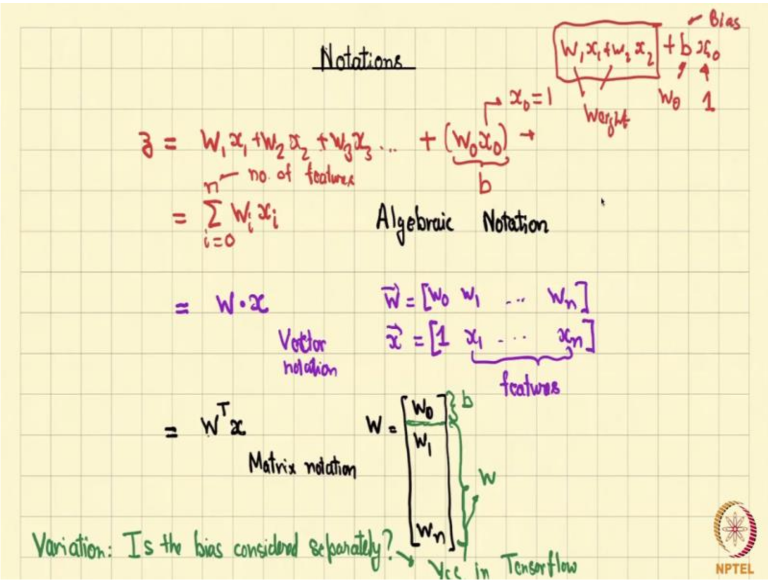
Now after this the next major breakthrough was the Rosenblatt Perceptron and what we are going to discuss later on today in this video something called the multi-layer perceptron based on this word perceptron which sits here which is basically for perception. So, now you notice you have w transpose x very much like what we had before. Now you have these ways which are sitting here.

So, now you have $\sum w_i x_i$. So, it took 14 years to go from $\sum x_i$ to $\sum w_i x_i$ which is the usual thing which we use in linear regression. Now apart from that you can see in the symbol this is basically a threshold. It is called a TLU, a threshold logical unit. All it does is you know you check once again whether z is greater than some value let us say Z is greater than 0.5 implies 1, Z less than 0.5 implies 0 something of that sort usually this limit will be actually 0.

We are not going to do either Rosenblatt Perceptron or the McCulloch-Pits Neuron. The reason I am showing this is to show that these ideas have been around for a long time and what we are doing is one very simple small change. So, instead of just going from sigma to this we actually people added a differentiable unit. So, this one obviously is not differentiable because it is a simple step function.

So, we basically put a sigmoid unit which is an approximation of this step unit. So, we use a sigmoid unit which is an approximation of the step unit and that makes a huge difference. So, we take output from there and then give feedback as I showed you in the last video. So, this was the Rosenblatt Perceptron it is only putting together a series of Rosenblatt Perceptron that we get the multi-layer perceptron which we will look at shortly.

(Refer Slide Time: 08:00)

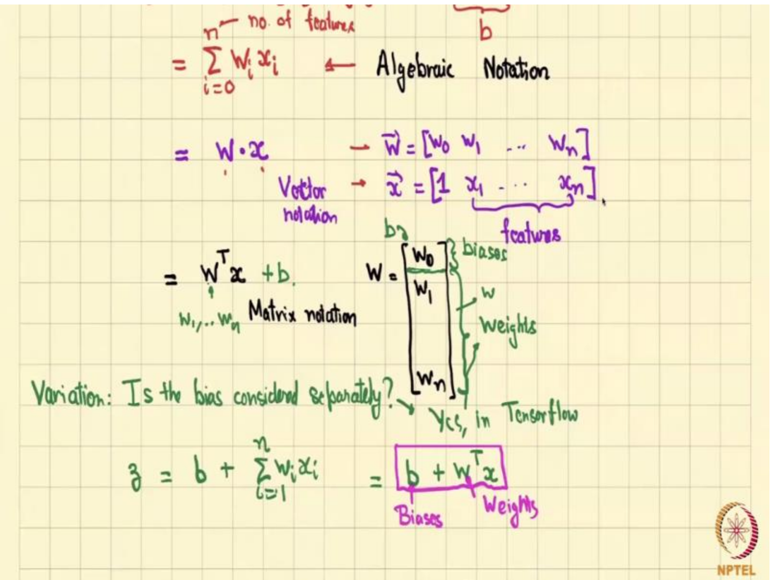


But before that here are some notations that we will be using as we go forward. Some of the notation can be confusing based on what exists in the literature. I am going to use typically w for all the unknown parameters, many people use theta and the literature. If we use theta, it gets easily confused with temperature so we are not going to use theta I am going to use w. So, typically notice we have $z = w_1x_1 + w_2x_2 + w_0x_0$.

Another way of writing it is $w_1x_1 + w_2x_2$ let us say there are only two units + b. This whole thing is called b where b is w_0 and then we do not multiply by an x_0 which was always 1. So, this is called the weight I will say this again in this video and this is called a bias. So, this notation I am going to call the algebraic notation as I said earlier, some people might write w dot x though this is almost never there.

But you can use it that way because w is a vector $w_0, w_1, \ldots, w_n$, x is a vector where one of these is a constant and others are variables $1, x_1, \ldots, x_n$ if you take a dot product you again get this $w_1x_1 + \ldots + w_0x_0$ also in the last video couple of videos I have used this W^TX, w is a column matrix and x is also a column matrix if you take w transpose multiplied by x you will again get $w.x$. This portion is called b, the bias by several people and w_1 through w_n are called weights.

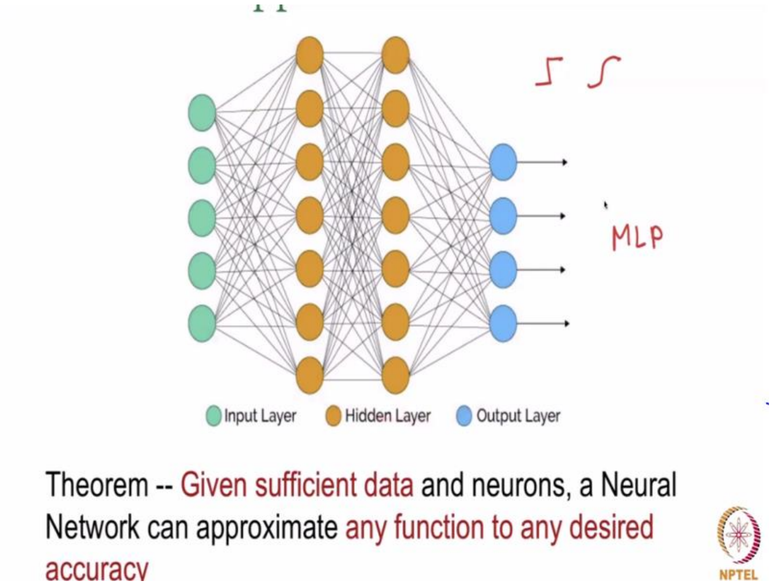
(Refer Slide Time: 09:50)



So, once again I will write this here there is a company in fact called weights and biases which deals with tracking various machine learning experiments that you have done. So, this is w, this is b. So, depending on how a code is written some people write $w^Tx + b$ and some people simply write w transpose x in case they write plus b then this w will consist of only w_1 through w_n and will not have w_0.

So, in packages such as tensor flow it is sometimes easier to consider the bias separately, rather than in a unified fashion. I am primarily going to use the notation where w_0 is already included and b is thrown out. It is easier for me to write it that way. So, I am going to use that notation at least for the rest of this course.

(Refer Slide Time: 10:52)



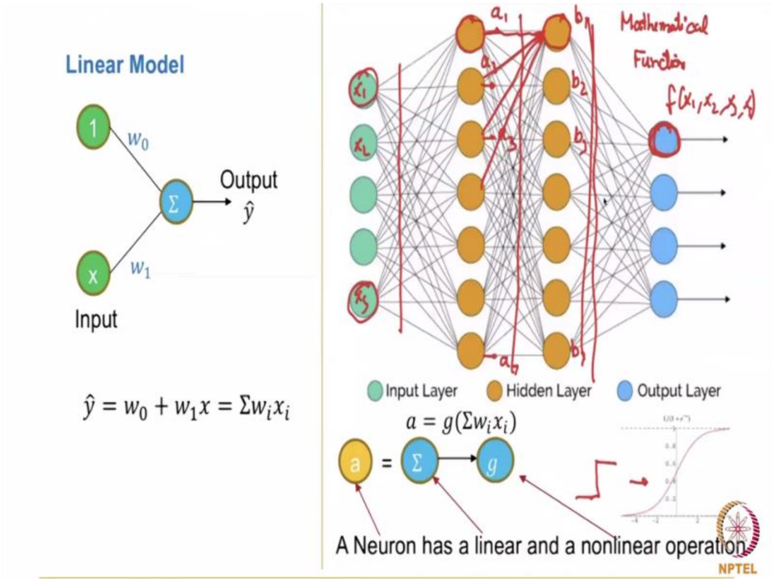
The universal approximation theorem as I showed you last time, simply says that if you have sufficient data neurons, we can approximate any function to any desired accuracy.

The point here is each one of these things is basically a perceptron except you will usually not have the step function, you will usually have something smoother, but that is also not necessary you can put a step function.

And this will still be called an MLP, a multi-layer perceptron, instead of that you can use a sigmoid etcetera, etcetera. The key thing here of course is the universal approximation theorem that is what lets us use neural networks easily. So, neural networks were inspired as I said here by biology and the structure of this kind of connection that is each neuron is connected to many other neurons.

But in the brain, it does not look like layer by layer, no it is not as systematic as this there might be some connections which are missing, some might be connected, but etcetera this is simply a MLP is a nice abstract structure that we can use systematically in order to solve neural network problems.

(Refer Slide Time: 12:01)



I had shown this a week or so ago, but this is just to again reinforcing your mind that every picture corresponds to some algebra and every algebra corresponds to some picture. If I want to represent $w_0 + w_1 x$, I can do so with a figure or with algebra. So, the figure is here $w_0 + w_1 x$, the algebra is here. Similarly, this figure corresponds to some algebra. So, when you see this, you should not think picture you should actually think that this is a mathematical function.

So, this entire picture represents a mathematical function. what function does it represent? now that is a little bit hard to write you can think of this as a series or as a function of a function. So, the way we do it is if you take this neuron this neuron gets input from all these 5 neurons then you do $w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5$ sum those together and then run it through some function g.

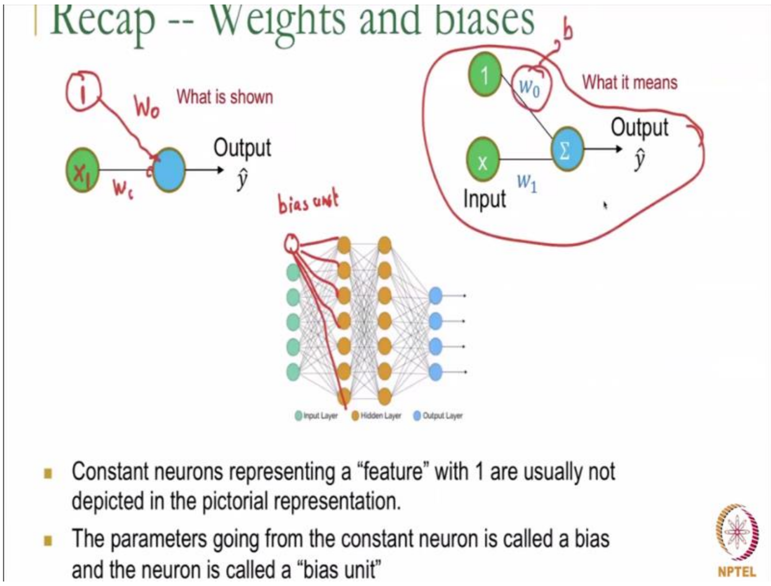
What that function is depends on how the neural network is specified I will talk about in the last video of this week, I will talk about various functions you can use or have been used successfully with the neural networks, but the earliest one was the sigmoid. The sigmoid as is written here is the same thing that we used for logistic regression $\frac{1}{1+\exp(-z)}$ or $\frac{1}{1+\exp(-x)}$, it goes to 0 to 1.

This is basically a smoother version of our step function. So, instead of using a step function which is not differentiable you would like to use a sigmoid. So, from here to here is a sigmoid and that is it. So, you calculate this neuron it gives an output we call it a_1 this one gives an output we call it a_2 a_3 just like we did with the XOR gate and let us say a this is 7 neurons so a_7 and when you look at this one gets an input from a_1, from a_2, from a_3 from a_4.

So, you can write an actual mathematical expression which will involve lots and lots of unknown parameters. When you look at this is finally still going to be a function of all these five input neurons except the expression is going to look a little bit messy other than that there is no big deal. So, it is just a long, long expression this one let us call this b_1, b_2, b_3 etcetera up till b_7.

You can write this in terms of b_1 through b_7 you can write b_1 through b_7 in terms of a_1 through a_7 and a_1 through b_7 can be written in terms of x_1 through x_5. So, in the end this is always a function of x_1 through x_5. So, this is some function of x_1, x_2, x_3, x_4, x_5. The point of course is the entire thing is just a diagrammatic representation of a mathematical function.

(Refer Slide Time: 15:14)



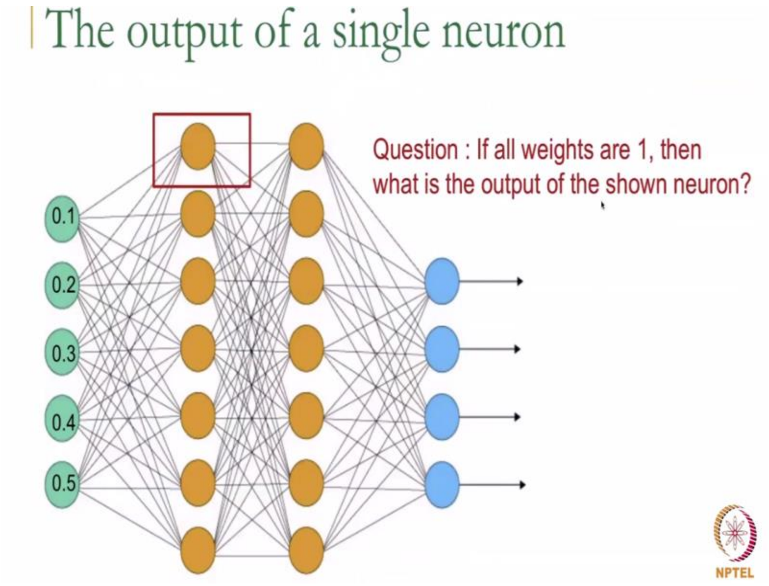
So, one important thing in terms of pictorial notation, so this is basically a pictorial notation, weights and biases as I talked about. Now typically what is shown in a network is something of this sort. The biases are almost never explicitly shown something in this sort is what is shown, but what it really means is there is an extra unit here which feeds into all these. So, this is basically the bias unit.

So even though only this is what is shown what it means is there was a_1 here there is x_1 there was also a w_0 here and a w_1 and the actual picture is what is shown here. So, this unit is basically called a bias unit sometimes denoted as I said by b sometimes by w_1 that is because why we do this is when you draw this it means you might have to connect this unit and this unit, but that is not true.

The bias unit has no inputs coming from anywhere else because it is always a constant. The bias unit is always a constant so this one is always a constant it is not as if something causes this one to be what it was, whereas if you look at anything else in the middle like this neuron. This neuron bought some input here, but if I draw a bias unit at the top that bias unit at the top will not have this.

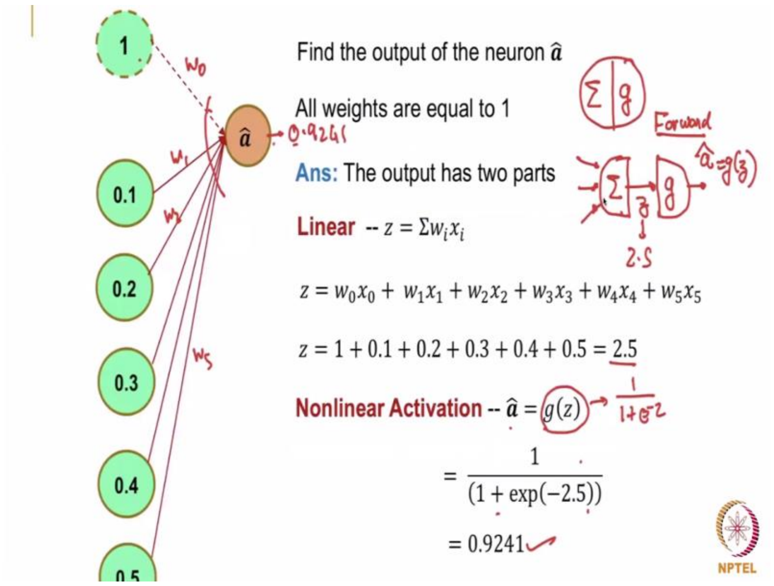
So, this will not be there that is because it is 1 why should it get an input from anywhere else. So, this is just some pictorial notation.

(Refer Slide Time: 17:07)



So, let us take a simple example we have a single neuron. Let us assume that all the weights are 1. So, all these weights that are shown here this one, this one every single weight here is one I want to know what is the output of this single neuron. So, this is like I said from biology to computation. How would we actually compute the output of this neuron.

(Refer Slide Time: 17:33)



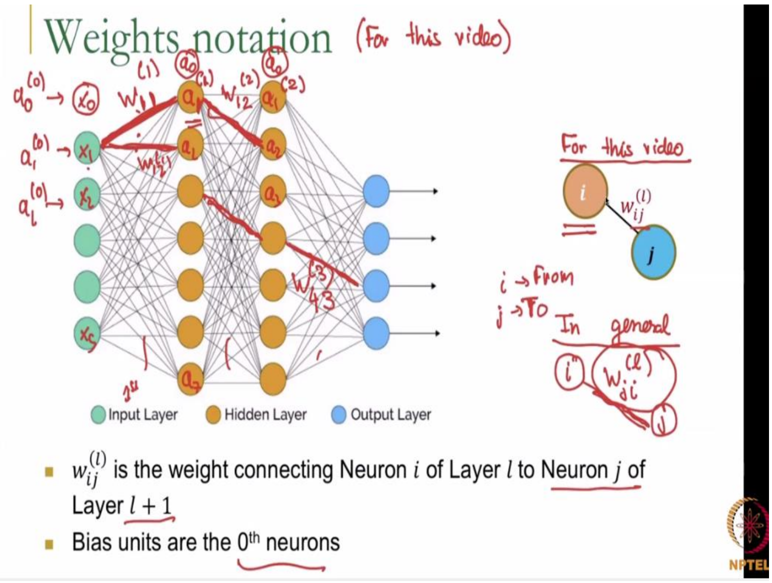
The solution is simple we want the output of this neuron a-hat all the weights are equal to 1. So, the output has always remembered two parts. The first part is the linear part which we also call the linear activation. So, the linear activation simply is $w_0 + w_1x_1 + w_2x_2 + \ldots + w_5x_5$. So, we can write that down here $w_0 + w_1x_1 + w_2x_2 + \ldots + w_5x_5$, w_0 is 1 that is because all weights and biases are equal to 1.

So, let us assume w_0 is 1 it multiplies 1 of course so that is 1 then it is one times 0.1 + 1 times is 0.2. So, this whole thing comes to 2.5. The next part always remember is we have two parts this $\hat{a}$ actually comes as two portions there is one summation which we did here and there is a g, which is a non-linear function. You can also think of it this way you had multiple inputs coming sum them and what came out was z.

Now z goes into this nonlinear part g and what comes out is $\hat{a} = g(z)$. So, this is what happens in a forward pass through a neuron these two steps. So, the reason I am spending time here is this becomes really important once we do the back pass or when we do the back propagation. So, linear fairly simple this portion z was 2.5, the nonlinear activation is $\hat{a} = g(2.5)$, $g(z) = \frac{1}{1+e^{-z}}$.

So, $\frac{1}{1+e^{-2.5}}$ and if you calculate it this comes to 0.9241. So, we can basically see that what comes out here is 0.9241. I request you to go through this example a couple of times just to understand what is happening. Remember that the input of a network is a vector as in you have multiple x is coming in, but what comes out is one single number. So, the input is a vector, but the output is a scalar to every neuron.

(Refer Slide Time: 19:59)



Now, for this video alone I am going to use a weights notation, that is because the first-time people see this kind of calculation, there is a natural way in which they think. If you write left to right this is the way you will think I am going to show you that weight notation, but as it turns out for writing code and for writing matrix expressions it is actually useful to use something else, but let me just write for this video.

I am going to use this notation, but in general I will show you a notation which is slightly different. For this video what I am going to assume is this. So, let us say I have this weight I cannot call them w_1, w_2, w_3, w_4 and keep on writing them that becomes too long. You need some nice notation to denote which layer we are in and which neuron is connected to which neuron.

So, if I look at this, I am going to call the bias unit which is never shown as I showed you or as I told you as the 0th unit this is x_0 this will be a_0, this will again be a_0 and I will mention what this a_0 is shortly, but this will be x_1 up until x_5, x_1 through x_5. This one here we are going to call a_1 because it is the first neuron this one, I am going to call a_2 and this will go till a_7.

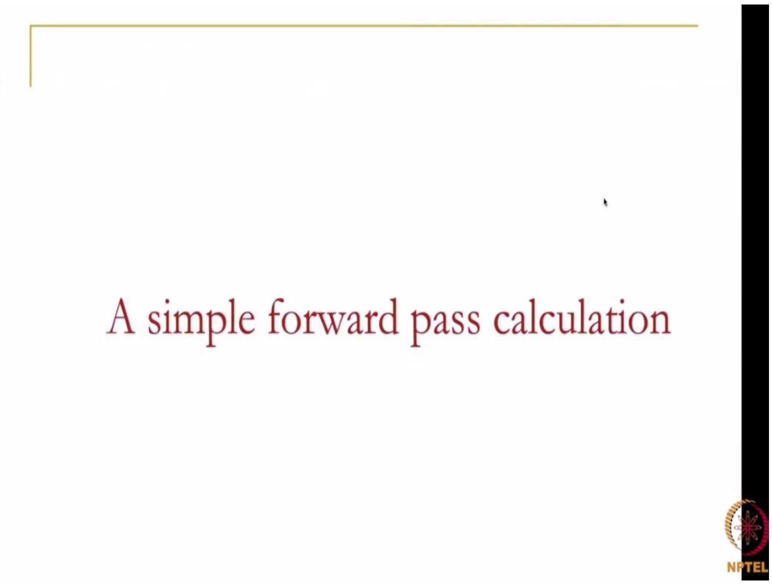
And typically, there is $\hat{a}$ which is put even if it is not put it is okay. So, let us just call it a_1, but I want to call this also an if this is a_1 and this is a_2 we need something else. So, we will put a superscript I will call this $a_1{}^{(1)}$, I will call this $a_1{}^{(2)}$ or some people call this $a_1{}^{(2)}$ to $a_1{}^{(3)}$ and this one has $a_1^{(1)}$ you can choose whichever way you want for now let me call this $a_1^{(0)}$ and this is $a_2^{(0)}$ by that trick this will be $a_0^{(0)}$.

Now what do I call this weight here. So, this weight here is in the first layer, this is the first layer. This is connecting the left neuron which is 1 with a right neuron which is 1 so we will call it w_{11} and it is in the first layer so I will call it 1. Similarly, this one is connecting the first neuron to the second neuron here. So, this will be called w_{12}, but I should not get it confused with this one which is also a w_{12}, but this is in the first layer.

This one on the other hand will be in the second layer. So, this is $w_{11}^{(1)}$ this one is $w_{12}^{(2)}$. So, just as an experiment if we have this, we simply count this is 1, 2, 3, 4. So, this is w_4 and this is the third neuron in the output layer w_{43} and this is the first layer, second layer, third layer so this is $w_{43}^{(3)}$. So, $w_{ij}^{(l)}$ is the weight connecting the neuron i of layer l to neuron j of layer l + 1.

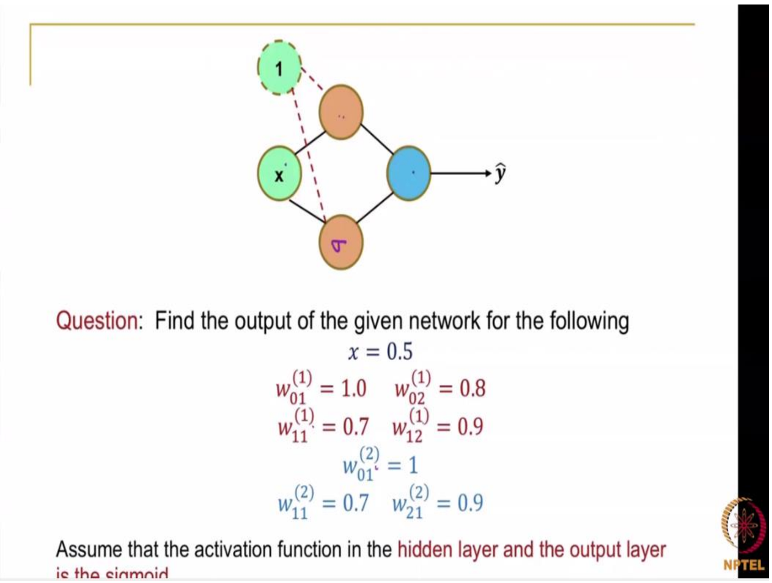
So, this is bias units are the 0th neurons. However, in general this ij is unfortunately or it is called $w_{ij}^{(l)}$ that is the order is flip, but for this video we will stick with this the i represents the from neuron and j represents the two neurons. Now why we use this will become clearer as we go on to back propagation because it makes matrix computations a little bit easier. So, slowly that is the reason it is used that way, but some books might flip the notation. I am just telling you what we are doing in this video.

(Refer Slide Time: 24:08)



So, let us do a simple forward pass calculation here instead of doing just one single neuron I am going to do a network kind of situation.

(Refer Slide Time: 24:21)

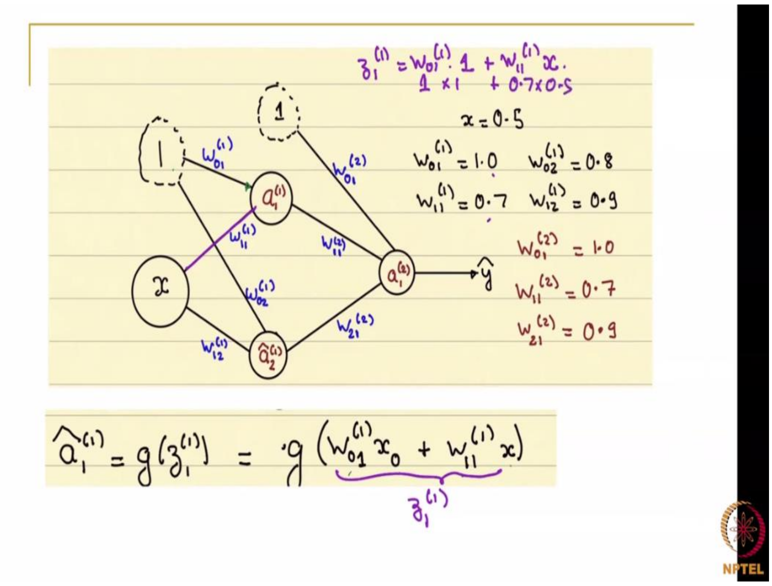


So, the situation is like this, we have a simple network again the bias units are not shown and we are given that x equal to 0.5 and we want to find out the output of the given neural network. We assume that all non-linearities are sigmoid. So, we assume that all non-linearities are sigmoid usually in many cases in fact in the final layer people do not use sigma x, people use linear activations that is there is nothing further that happens.

But let us say we are using sigmoid in all neurons. Now of course we need all these weights, we do not know these weights we also need the bias. So, let me first give you two of these so it said ${w_{01}}^{(1)}$ is 1. So, remember goes from the 0th unit which is the bias unit to the first unit and it is in the first layer this is 1 and the second one this one is given as ${w_{02}}^{(1)}$ is 0.8. So, we have these two pieces of information.

Here is the full data which is given for this network, you are given w_{01} you are given w_{02}. This is these two weights w_{11} which is this weight w_{12} which is this weight and then in the final layer you have 3 weights because we need a bias unit here also. So, 1, 2, 3 + 4 weights here so you have a total of 7 layers here. Let us assume that the activation function everywhere is the sigma as I said assume sigmoid non linearity.

(Refer Slide Time: 26:13)

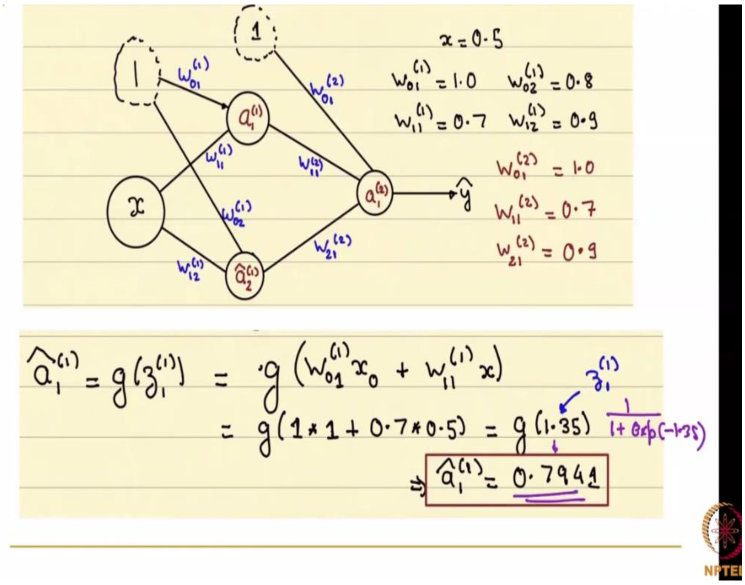


So, here is the network drawn a little bit more clearly with all the information given here on the right-hand side. I would recommend that those of you who are watching it, pause the video and try this out by yourself just to ensure that you understand the notation well, it will obviously be straightforward once I describe it. The calculation is not hard you just have to make sure that you are connecting everything like things to like things etcetera.

Just make sure all the things are properly they are computed. So, we do this systematically we first calculate $\widehat{a_1}$. So, $\widehat{a_1}$ remember is made up of two portions. You first calculate the linear portion which is the linear combination of these two and then you calculate the non-linear portion. So, we do the same thing we first calculate z_1 is so what is z? ${z_1}^{(1)}$ is ${w_{01}}^{(1)}$ multiplied by the bias + this ${w_{11}}^{(1)}$ multiplied by x as I have written here this is z_1.

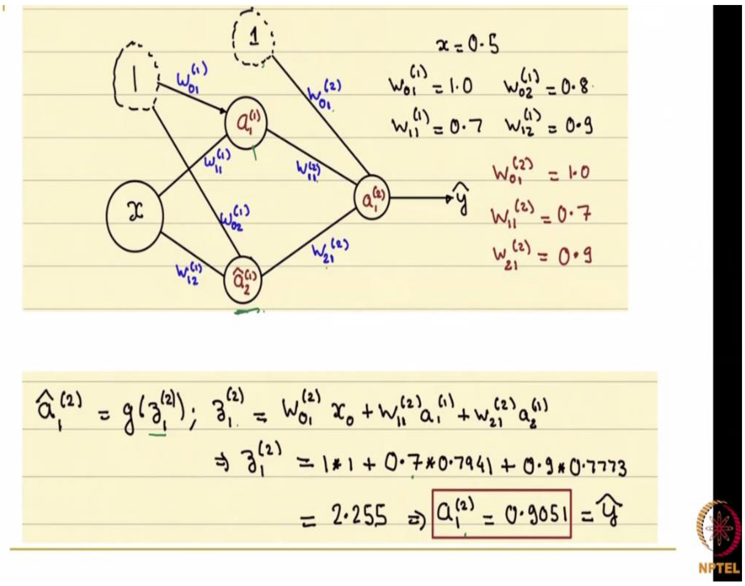
You calculate this comes to ${w_{01}}^{(1)}$ is 1 written here, multiplied by 1, ${w_{11}}^{(1)}$ is 0.7 into 0. 5. So, if you do this calculation, you can write down here.

(Refer Slide Time: 27:50)



So, we basically get $z_1^{(1)}$ is 1.35 from here you then calculate the non-linear portion first the linear motion then the nonlinear portion. So, once you calculate this you get $\widehat{a_1}$ or $a_1^{(1)}$ is basically 0.7941 it comes by doing $\frac{1}{1+e^{-1.35}}$ because it is the sigmoid which we decided was the nonlinearity in this unit.

(Refer Slide Time: 28:28)



Now similarly we now need to calculate $a_2^{(1)}$ which is this thing here sorry $a_2^{(1)}$ which I have written the final answer as 0.7773 we can see the calculation for this 0.7773, it is a straightforward calculation just like before, no major mysteries here. This has a linear path which is z_2 and it is g of z_2 which comes out as a_2 and this is a simple calculation it is fed by x and by 1.

So, w_{02} times x_0 which is simply $1 + w_{12} * x_1$ which is this is now you can see the weights w_{02} is 0.8 and w_{12} is 0.9, so you write 0.8 times 1 + 0.9 times 0.5 which is 0.8 + 0.45 1.25 ${z_2}^{(1)}$ is 1.25 calculate the sigmoid of that it comes to 0.7773. So, this is also the straightforward calculation. Now our final job is to calculate this; this is what lets us calculate the $\hat{y}$ which we are finally interested in.

So, we want $\hat{y}$ and $\hat{y}$ is basically exactly the same as ${a_1}^{(2)}$ that is what we throw out as $\hat{y}$. So, this one has input from 3 places ${w_{01}}^{(2)}$, w_{11} and w_{21} and it multiplies just like the normal perceptron by the respective weights. So that is what we have here $w_{01} * x_0 + w_{11} * a_1 + w_{12} * a_2$. So, fairly simple do the calculation you can see that I have written exactly the same thing as what existed in the weights here 0.7 times.

Now remember this value was 0.7941 which we already computed and this value was 0.7773. So, you add all those you get the linear activation as 2.255, do a non-linearity on top of that you get ${a_1}^{(2)}$ is 0.9051. So, as you can see the network was fairly simple. The forward passes once you are given this value move here calculate this value. Once you are given that calculate the final layer.

This is regardless of whether there is 1 layer, 2 layer how many our layers the computation takes place in exactly the same way. So, what we saw in this simple example was that you can do a simple forward pass through the network. We will see later in this week that you could have done this via a matrix also or matrix operations rather than doing separate scalar operations here and that makes things more effective.

Overall, in this video you simply saw why a neural network is defined the way it is and how to do a simple forward pass for a neural network and we will now exploit this in the videos to come this week where we will also do back propagation. Thank you.