Foundations of Cyber Physical Systems

Prof. Soumyajit Dey

Department of Computer Science and Engineering

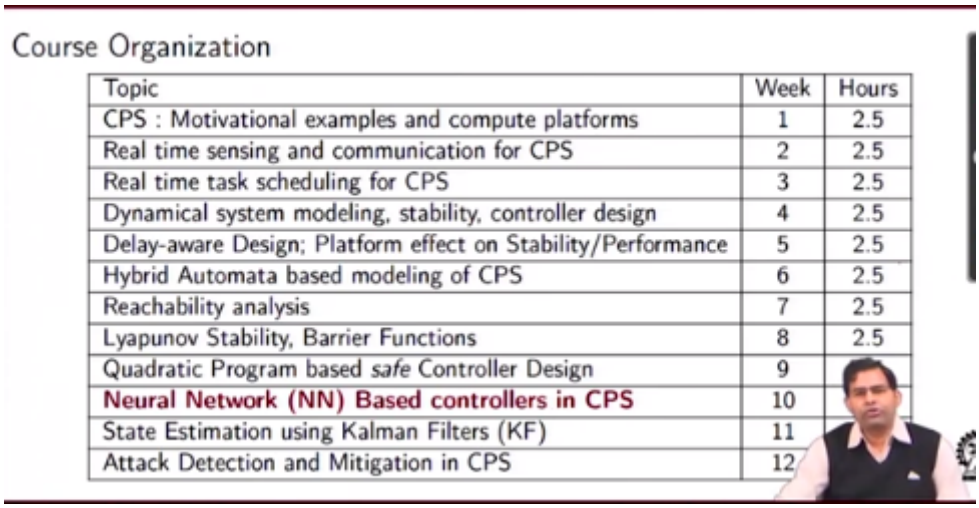
Indian Institute of Technology-Kharagpur

Lecture - 47

Neural Network (NN) Based Controllers in CPS

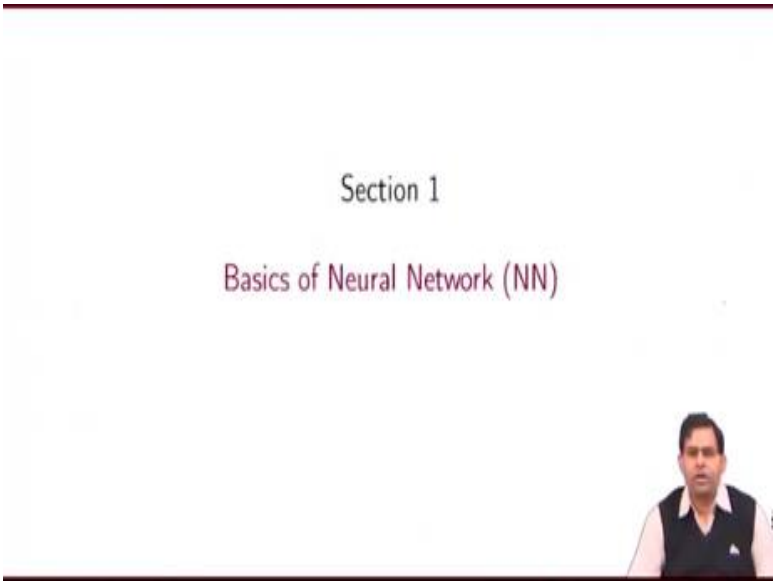
Hello and welcome back to this lecture series on Computational Foundations of Cyber Physical Systems.

(Refer Slide Time: 00:35)



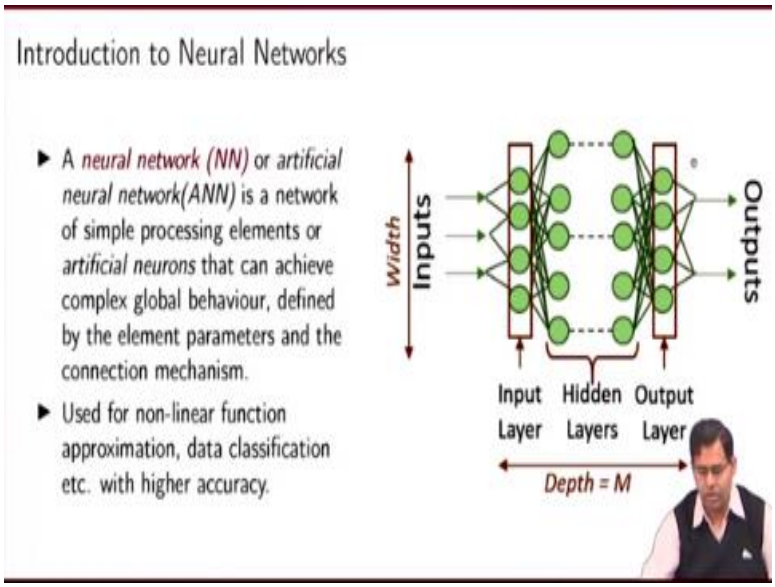
So we will start our week 10 coverage. So here we will be focusing on neural network-based controller design for cyber physical systems. As we can understand that neural networks and data driven system and methods are becoming very common in our day to day life as well as large system design computational cyber physical even hardware systems also, okay.

(Refer Slide Time: 00:58)



So for getting on with this, since we want completeness in the course and we do not assume that you are coming here with a background in AI, ML, neural networks and stuff, we will just start with the basics of NN and then we will see that well how NN driven control really works in practice.

(Refer Slide Time: 01:13)



So if we try to introduce neural networks, so this is basically a network of simple processing elements or artificial neurons, which try to achieve a complex behavior. So the way neural networks typically work is first they require some training. That means you have some data set and using that you define the parameters of a neural network.

And when a neural network is trained, what it represents is a function, a nonlinear, possibly high dimensional complex function, which gets represented by this. And inside a neural network, what you have is at an atomic level, you have neurons. Neurons are basically small, functional forms, okay. And they are arranged in a specific way. They are layers, so it is like it is like a pipeline.

So if you look at a new neural network it is nothing like but a pipeline, where each stage of the pipeline is like a layer and inside a layer you have a set of neurons. So some data is coming, it is a forward I mean, it is it is just like a data stream going inside the pipeline and coming out of the pipeline with some transformations. So you have data coming in here. In this stage of the pipeline, there are certain neurons.

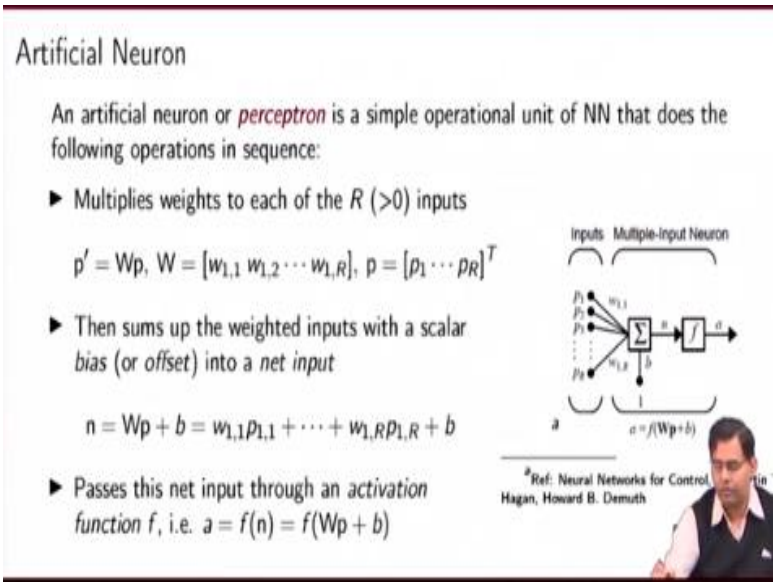
They will produce some output based on these data. These output will propagate it to the next stage, which will again contain some layers, right. And in that way, they will go to the finite layer final layer, which is called the output layer from which I will get some outputs, okay. Depending on the number of layers, we will have a depth. Each of the layers are representing elementary functions.

Of course, those elementary functions need parameter values. These parameter values get freezed during a training process. And once the training is complete, all the parameter values are freezed. Essentially you have a full network and that full network may be representing like I say, a complex function F which is differentiable and this is high dimensional.

That means x may be a, multi I mean, I mean comprising many variables x_1, x_2, x_3, x_n, x_{100} like that, okay. So the target of the neural network is that well, there is some unknown behavior somewhere, but that unknown behavior's output is observable with respect to an input. We are trying to capture based on some inputs and outputs that I know about that unknown system, what is the mathematical function that best captures

the dynamics of this unknown system. So that is what typically an NN artificial neural network will do.

(Refer Slide Time: 03:49)



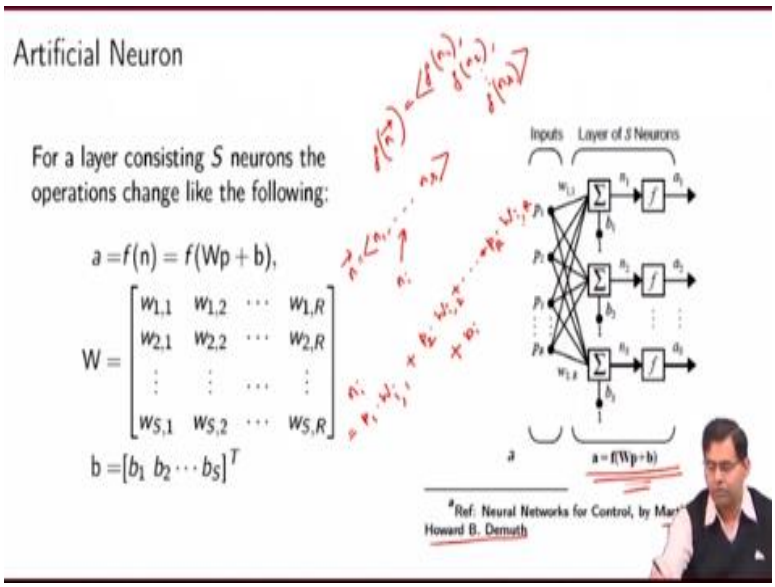
So uh at a basic level, let us first talk about each neuron and a single neuron or a perceptron is a simple operational unit, which does the following. So what it has is there are some weights. It takes a R array input. So let us say this is the input vector okay p. So p is a vector of individual scalar components p_1 to p_R, okay. Now this neural network will have a weight matrix W comprising this kind of, so let us say this is neural network level 1.

So $w_{1,1}$, $w_{1,2}$ like that, let there be I mean since they are R inputs to process there must be R weights so that there is a valid dot product okay. So then, at this point here, we will be, what we will be doing is we will be carrying out a weighted sum like a dot product here. So with that we will get a value let us say that is n okay? So this is how using this summation we will represent this node.

So essentially given a vector input this weight matrix is converting that vector input to a scalar output okay? And what will happen is this scalar output will be offset by a given bias value b. That means n will be equal to this W_p plus this bias value b okay. And in that way I have reduced the input vector to a scalar by first weighting the components then adding them up and then giving the bias.

And then this n I will pass through a final stage function. This function is called an activation function okay, so that as a function of n and I will get some final activation output here. We will we will see what an activation function is, how those things are defined.

(Refer Slide Time: 05:42)



So that is the, whatever we said till now is just about one neuron. Now I can arrange a set of neurons here, okay. So once I arrange each of those, so in this way you can see here I have this one one neuron represented by this weight matrix W and the bias b, right. Now what I can do is I can have a collection of such neurons.

So this collection of neurons can be represented by weight matrix where the such that overall I have 2 cross 2 weight matrix and it is such that each row of this weight matrix represent one neurons operations, okay. So so that is fine. You have this kind of representation here and with each of these weight matrices, so for weight matrix one if you see earlier this was about one new one weight matrix right for one neuron.

Now I have this S number of weight matrices with S neurons right. So for each neuron you have one row here and with everything together you have this two dimensional weight matrix. That means the set of R inputs that input vector of size R it is feeding

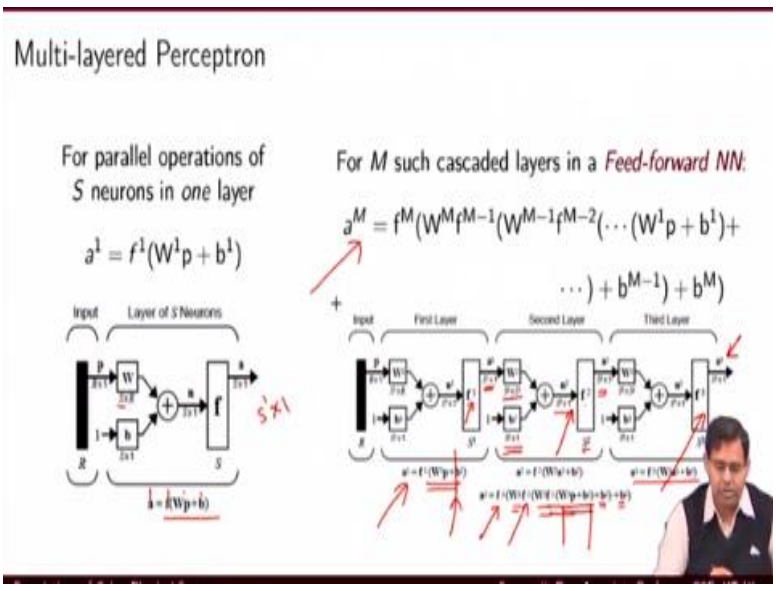
all these individual neurons and they also have their individual biases here b_1, b_2 up to b_S, okay.

So that finally, what happens is here you get a vector n 1, a vector n which is nothing but a collection of these individual values n_1, n_2 up to n_S okay. And each of these values if you pick up some n_i here how do you get it is n i equal to what? You have p_1 times $W_{i,1}$, p_2 times $W_{i,2}$ like this up to p_R times $W_{i,R}$, right. And with that, you will add this bias for that i-th neuron b_i.

So this will give you some n_i. And with all those values of n_i you have this final vector n here. And you will apply the function, final activation function f component wise on this vector. So essentially it will be f (n_1). So that is what you get as an output. So overall the your activation value is this and this is given in a vector form like this and it is represented graphically like this with a set of neurons, set of activation functions in the activation layer and the inputs all written in a vector form nicely here.

So this coverage has been from a tutorial in the American Control Conference in 1999 which was given by Martin Hagen and Howard B. Demuth. They also have a very nice book on neural networks which you can just consult for this purpose.

(Refer Slide Time: 08:59)



Now so that was just about one stage of the pipeline, right. Now I can, if you see I can just I can just cascade many such stages now, right. So so for parallel operations for this kind of n new S neurons in one layer what what I can write is I can represent this weight matrix here, I mean this entire thing here right with S number of weights, I mean all these things etc., by this single node right.

So this weight matrix W which is of size R cross S or sorry S cross R can be represented by this single node here. And as we discussed, there will be S number of biases and there will be R number of inputs, they are all going in here. And S number of biases are feeding here so that you get an S sized vector n on which an activation f will work so that you can get an S sized activation which is a vector a like I computed earlier.

All I am saying is this entire thing we are drawing here can be represented in a vector form like this. And like I said that we can now add multiple layers here. So you can see what are you getting as input getting as output given an input. You gave a you gave an R sized input of vector p, which was processed by this S cross R sized weight matrix, and you get an S sized output, right?

So given this S sized output a 1 of size S_1 cross 1, this 1 is now coming, because let us call it at the stage one of the pipeline or first layer. So this is your input, this is stage one of the pipeline or layer one. So layer one's output, let us call it a 1, okay. And we are representing this layer one operation by this equation f (W_p + b) and since this is layer one, we are just sub superscripting with 1 here.

So a 1 equal to f 1 W_1 b_1 here, so that is what we have. So we get as output S cross 1 sized matrix a. So this is again layer one. So we are calling it S_1 cross one sized matrix a 1 okay. So this S_1 cross one sized matrix called a 1 is being fed to the second layer. In the second layer again what you are going to have is a W of size let us say some S_2 cross S_1 and for b you should have some S_2 cross 1 number of bias values.

So here you will get a vector n_2 in the second layer output whose size is S_2 cross 1, just like it was S cross 1 here. This S is same as this. Here we chose S_2 with this existing S_1 number S_1 should match with a 1 here right. And we got an S_2 sized vector. It is processed by an S_2 sized activation layer, so that we get an S_2 sized activation here, output in the second layer.

And that will be processed again by the third layer so that you get an S_3 sized activation here called a 3. So in effect, if you write this entire thing, if you write all these, if you cascade all these pipeline stages, layer wise, what you are getting is the first layer activation W_p^1 + b^1 will be, is being processed by the first layer activation, which is f^1 here, right. You get the output, right.

So this is your output here. Then it gets multiplied by W^2 and it gets added with the bias b^2, right? So that is that is your output here. So if you see this, this is your output here which is being processed by f^1 here. So you come up to here, gets multiplied by W^2 here. Then gets added with the bias of this stage which is b^2 here. Then it should be processed by this activation layer f^2 here, right.

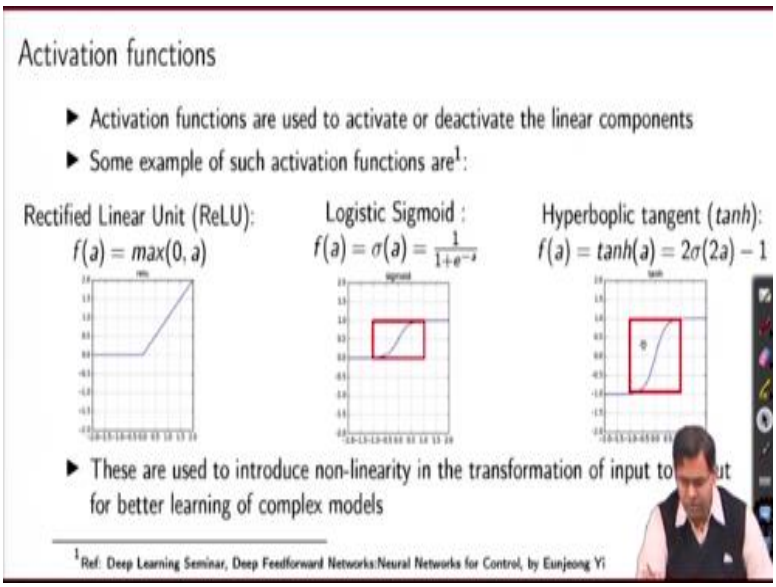
And then it gets multiplied by the weight matrix W^3 here. Again gets added with the bias of the third stage here. And this entire thing will be now passed to the last activation layer the third layer which is f^3 here. So that is the a 3, right. So if you have, so here we are just showing a picture up to three layers, but if you have M such cascaded layers, so in general, if you just change these three to M this is what you will get as the equation right.

So now in your equation, you will have a M. For that you will have an f^M. Here you will have W^M working on f^{M-1}. M – 1 is application of the activation layer working on this and it will continue like this, right. So overall for a feed forward neural network, if

you are looking at the final polynomial functional form, it should be something as complex as this, right. The question is how do I design the parameter values.

And also why am I going to do that, we will explain that. But now let us also talk about what is activation? We have been just talking about this activation function.

(Refer Slide Time: 14:14)



Typically, these are functions which are used to what we call as activate or deactivate nonlinear components. So basically they are elementary classifiers, okay. So let us see an example. For example, these are the popular activation functions which are used, okay. For example, you have this rectified linear linear unit activation.

What it does is given the input data, it will produce it as an output like this, like essentially it is a function that suppose the input is a it will either produce a or 0 in case the value is negative. That means, if the input to the activation layer is negative, your rectified linear unit the value output will be 0. And if it is positive, it will output that positive value. So the graph looks like this that as long as the input, see this is your input, right.

So this is your input, I mean on this side the input is negative, on this side the input is, this side negative, this side positive. So as long as it is negative, it is not passing on the

negative value. It is just outputting 0. The moment it becomes positive, whatever is the value it is passing it. So it is a gradient 45 degree line, okay. And also you have activation which is the logistic sigmoid function.

So what it does is this is your 0. At 0 it will produce an output of 0.5 and then it will increase smoothly and eventually it will saturate it out, output of 1. And whenever it is negative, it is decreasing smoothly, from 0.5 it is decreasing and then it is settling at 0. So the logistic sigmoid function will always give you a value which is between 0 and 1 and in between it will increase smoothly.

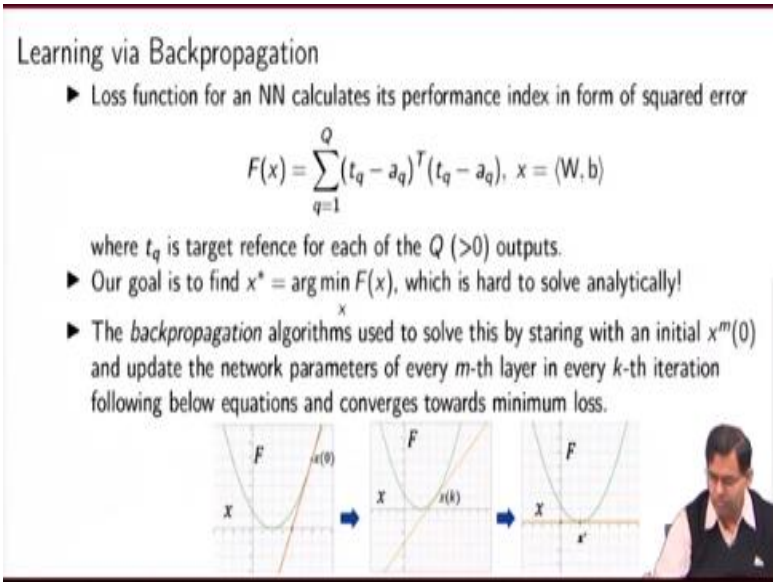
And also you have this tanh the hyperbolic tangent function which is also another candidate activation function. It looks almost like this, where the bounds are, instead of 0 and 1 the bounds are like this, that it is -1 to + 1. So let us understand why these activation functions are there. These are primary there to introduce non-linearity in the transformation of input to output. Now why do I need the non-linearity?

That is a very important thing, right. The question is my inputs and the weights and the biases they are all linear components, right. But I want my function to be smooth, differentiable. And I also want my overall function to be able to capture arbitrarily complex behavior. Now complex is typically nonlinear behavior, right. I mean, it should be smooth and differentiable.

So I mean, the functions should be nonlinear, so that they are smooth and differentiable, right. And I also want my overall neural network to capture complex nonlinear behavior, right. So so for that, we will like to introduce non-linearity in the entire transformation process. If you see the weight matrix multiplication, the bias solution basically, these are linear transformations that are happening, shifting that is happening.

So in order to introduce non-linearity we need to have these kind of these kind of activation functions which are inherently nonlinear. If you see all of these have nonlinear trajectories here. Okay.

(Refer Slide Time: 17:50)



Now the question is well, how do I learn these values of W and b, because they are not known right. So first of all, what is known to me? So when I am going to train a neural network, like I said that we are assuming there is a closed system to which we have given some inputs and the system has produced some outputs, okay. So given those inputs, the inputs are these values p's right, this vector p's.

We obtain some outputs, the outputs are these a's, right. So there are Q number of training inputs, capital Q, 1 to capital Q, for each of the training inputs, the the actual value, the reference that given some, given that training input, the actual output, or the target reference is t_q. Whereas given some input p_q here, the inputs are p is here, right.

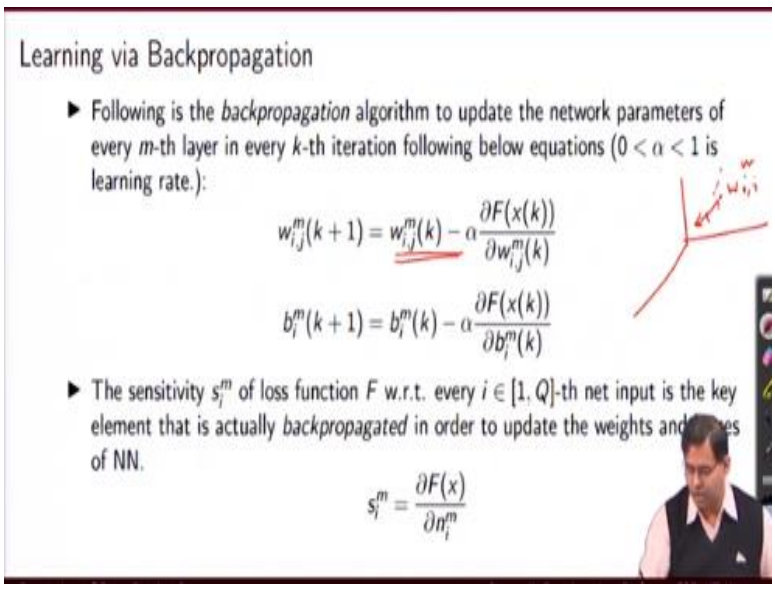
So given some q-th input p_q, the training output expected is t q, whereas what the neural network actually gave me, let us say it is a_q, right. So this is the error and all we are doing is we are creating a quadratic objective function with respect to this error and my objective is simple right. I want to just minimize this quadratic objective function.

So what we want is we want to suitably parameterize this function f that means figure out the values of W and b. So you if this F is a function of x, where x is W and b, okay. And what we want to do is we want to we want to choose suitable W and b, so that this loss that is the errors, the errors in their quadratic expression, this loss is minimized, okay. And the loss is defined as F as a function of W and b, okay.

It is hard to solve analytically. And that is what for we for this we have the famous backpropagation algorithm. So what this backpropagation algorithm does is it will, it will start with an initial value uh x m 0, okay. It will start with an initial value x m 0, and update the network parameters. In every k-th iteration, it will update the network parameters of every every layer following the equations, and it will converge towards some minimum loss, okay.

So so so that is the point. I mean, we are just trying to figure out how the loss can be minimized and what are the values of the weight matrix W and b, for which the loss essentially gets minimized. So let us see how that can be done.

(Refer Slide Time: 20:31)



So what the backpropagation algorithm does is it updates these network parameters of every m-th layer in every k-th iteration. So let us say I am in some intermediate iteration of that algorithm. And right now the ij-th weight matrix for the m-th layer, so if you can

remember, let me get the notation done first. So this is my m-th layer some layer m. Here I have W_m, and inside W_m, I have an ij-th component, right.

So we call it as w i,j with a superscript m. And this is my value of w m i,j in the k-th iteration, and I want to update it to some $w_{i,j}^m$ in the k + 1-th iteration, okay. So what we are saying is, the update will be given by this equation where we are calculating that well, what is the rate of change of F with respect to the rate of change of w at this point, okay.

And you multiply that with something called learning rate, okay. So that means, and you subtract it from here. Let us understand what it means. It means suppose, see what is my objective? My objective is that I want to minimize F, okay. Now we are having some choice of value of $w_{i,j}^m$. We see what is the derivative of F with respect to $w_{i,j}^m$ at this point.

And if we see that the value is positive, that means well, if I increase w, F is going to increase. That means, if I decrease w, around that specific point which is $w_{i,j}^m$ F will decrease, and my objective is to decrease it, right. So I will decrease F in, basically I will make the algorithm move the value of w towards the direction in which F is decreasing. That is what I should try to do.

So from the current estimate of w, I will, I will subtract this rate multiplied by this learning rate. So let us, let us just think something like this. I mean, suppose you are you are moving around, okay. This is the time axis and here you have you have a value of speed, okay. Suppose right now your speed is here, and you want to calculate the speed after some interval t_1.

So what you do is you, you take the initial value here, and you multiply this interval with the rate of change of x, which is here, that is the what is the acceleration here. So you take the value of speed here, then you add acceleration times this value t_1, okay. Now this value of acceleration is what the rate of change of speed at this point, okay. The smaller you take t_1, the more, the more accurate result you get right, by choosing a smaller time step.

Because the rate of, the acceleration may be something here, the acceleration may be something larger or smaller in between, right. So what I really evaluate here is the rate of change of speed at this point. And if I take t to be small, then somehow I can say that well the rate of change of x is kind of constant inside the interval. So we multiply it by the rate of change you computed and that gives you what is the speed at this point or this point, right.

So same thing we are doing, but not in the time axis, but in the in the in the direction of w, I mean, I mean in the direction of this change of w, okay. So what we do is at that value at my current estimate of w, I see what is the gradient of f, if it is decreasing or increasing, say it is positive. So that means since I, my objective is to decrease f, what I will do is I will move my optimization to the opposite direction.

So by moving in the opposite direction, essentially it means that you multiply this gradient with a learning rate and you subtract it from your current estimate of w. That means, if I decrease w around my current estimate, F is supposed to decrease. Since my objective of F is to decrease, I will choose to subtract from w this value. Now what happens if I increase the learning rate?

So if I increase the learning rate that means my approximation may not be good, right? Because that means I am assuming this rate to be same, around the larger, so suppose I am in an in-dimensional space, and right now I am estimating everything here for my

value of w i,j m, right. So if I come this direction and if I take a larger step. For a larger step, larger alpha means a larger step.

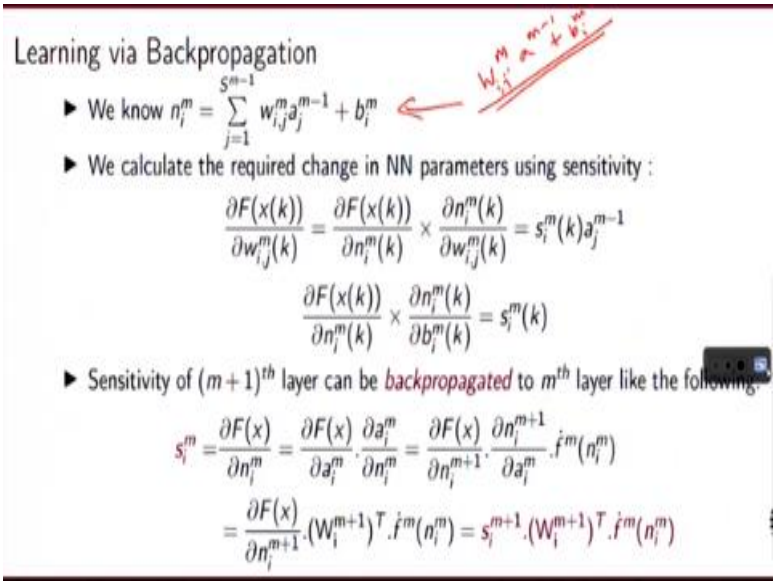
If I take a larger step in between the gradient may have switched signs, right. From start decreasing it may, from increasing it may have gone to decreasing, right. I do not want to miss that switching. That is why this learning rate should be small here. I mean, there are several ways to choose learning rates, more on that is later. So fine. So using this equation, we will update the value of w from the k-th estimate to the k+1-th estimate.

Exactly by similar logic we can also choose the update update on the value of the bias, okay? Now the question is how do I put these things into practice, okay. Now the sensitivity of the loss function we will try to define that because that is also essential. You see, we have already defined the loss function and with respect to every input, we want to define what is sensitivity and because it is a key element here that is actually backpropagated.

So what we do is we try to compute what is the rate of change of this loss function with respect to the value n that is the value computed using the weight and the bias transformation before the activation, okay. So so what we do is we define this quantity delta F delta n i as the sensitivity of the i-th stage okay here. So if you see n i means some some i-th stage and in that i-th stage what is the sensitivity of the loss function.

That means how how does the value of F change with respect to that n-th stage, that i-th stage output is what we define as the sensitivity and we will see that why it is important.

(Refer Slide Time: 27:04)



So observe one thing we know that how to compute this n i right. n i that means the i-th stage n well, I mean we and what and what is its value for the m-th estimate. It is given like this right. It is basically the summation of $w_{i,j}$ a_j plus b_i okay. What is a_j? a_j is the m minus the previous stage value, right. So j equal to 1 to s^{m-1}. So there is my thing right?

And what we are doing is, so this is this is my n here right and what we are doing is, sorry let me just repeat this thing. So m represents the m-th stage here okay. Yeah m represents the m-th stage here, like you see n_2, then n_3 like that. And in the m-th stage, it has a component, we talk about the i-th component here as n_i^m. So let the definition be there.

So when I say n m i, it represents m-th stage i-th component value before activation. You may you may like to pause your video here and go back to that picture earlier and, and corroborate with that. So that is how we define that the rate of change of F with respect to this thing is what I call as the sensitivity of the function in the m-th stage with respect to the i-th component of that outputs of the stage output.

So well we will see why it matters, because, we are doing something here right. So once we are doing here we want to figure out what is del F, del $w_{i,j}$, what is del F(b_i) right.

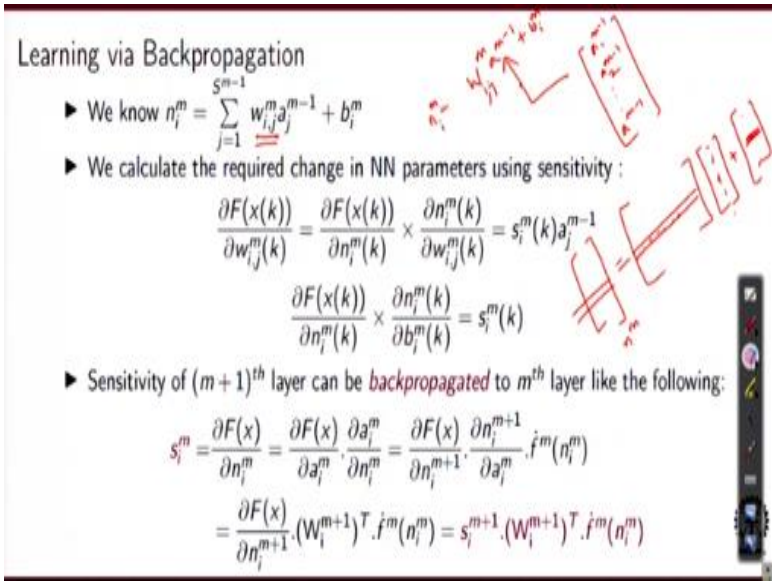
We will see that how sensitivity helps in that respect. So first let us go back to what is n i m. n i m is simply what is the m minus 1-th stage activation output, you take that and you do a vector multiplication with this i-th stage weight that is $w_{i,j}$ where j value is changing from 1 to s^{m-1}.

So just for reference here in any stage you have some s^m right. That is s m is the number of elements here, right. So if you see s^1, s^2, s^3 like that and we are talking about some m-th stage. So at the m-th stage you have in that vector these many things, these many values from 1 to m – 1. And in w you have $w_{i,j}$ in the m-th stage, right. So and in the m-th stage, you have the input coming from the m – 1-th stage.

So you will have w a m - 1. So if I just write it in the vector form, it is w m, the m-th stage weight multiplied by a m - 1 right, plus the m-th stage bias. Let us see. And here all you have is it written in using the vector form. And since I am focused on the output's i-th component, so further what I do what I am doing is I am only looking at the i-th component right here, okay.

So for looking at the i-th component, what we are doing is we are keeping i fixed and we are letting the j vary like this. So you can you can verify the calculation like that. So let me let me just repeat what I just said. So if you see what is the i-th stage vector. The i-th stage vector is something like this, right. So the i-th stage output is some n i, the m-th stage, sorry the m-th stage output is some n m, right.

(Refer Slide Time: 31:29)



So if I look at this n m, it is given by some W^m, which is the m-th stage 2d matrix, and then the previous stage output m - 1 plus this m-th stage bias, right. So that is what we have. Now here I am looking at the i-th component, right. So since this W^m is a 2d matrix, and here I have this, and plus this, and I am looking at some n here, this is equal to this n m, right? And I am looking at some i-th row here.

So if I am looking at the i-th row, that means for that i-th row I should take this and I should, I should just some I should take all the elements in the i-th row and multiply them with this and then do the addition with this right, and take the sorry and take take the do the addition with the i-th component. So this is the i-th component for the bias.

And then from the i-th row, you take all the components and do a dot product with the full thing here, right. This a - 1, right. So that is what you do with this change with with varying these for the i-th row you have all these w i m right, $w_{i,j}^m$ right. So you are just varying the value of j. That means you are moving over this row and you are taking all these values from here, right.

Because you are taking this. So this essentially is a m – 1 1, a^{m-1} sorry. So let us see how a looks like. So that is nothing but a_1^{m-1}, a_2^{m-1} like that a m – 1 whatever is the dimension here, right? So for that you have this j changing and you have you are moving

over this row. So you have this j changing and you are multiplying them and then you are adding this b_i^n, right.

So that is what is happening here. So this explains how the sensitive how n i um, this explains kind of how the value of n that means the i-th component of the m-th stage output is actually calculated and the i-th component's value you can get through this equation for the m-th stage output. And then let us see that well how that helps me to define this rate of change of F with respect to w and rate of change of F with respect to b.

So this is my rate of change of F with respect to w in the m-th stage. And what we can do is we can apply the chain rule of differentiation and we can write this as well del F del n_i^m. That means what is the value of F with respect to n in the m-th stage multiplied by what is the derivative of n in the m-th stage with respect to $w_{i,j}^m$, right. We can do that.

So if we if we write it using this chain rule, then this first first thing, the first component becomes my sensitivity because that is how we have defined sensitivity, okay. And then the second component is, as you can see, well this is simple, we have already obtained this expression of n_i^m right. n_i^m has this thing $w_{i,j}^m\ a_j^{m-1}\ b_i^m$. So you want to take a derivative of n_i^m with respect to $w_{i,j}^m$.

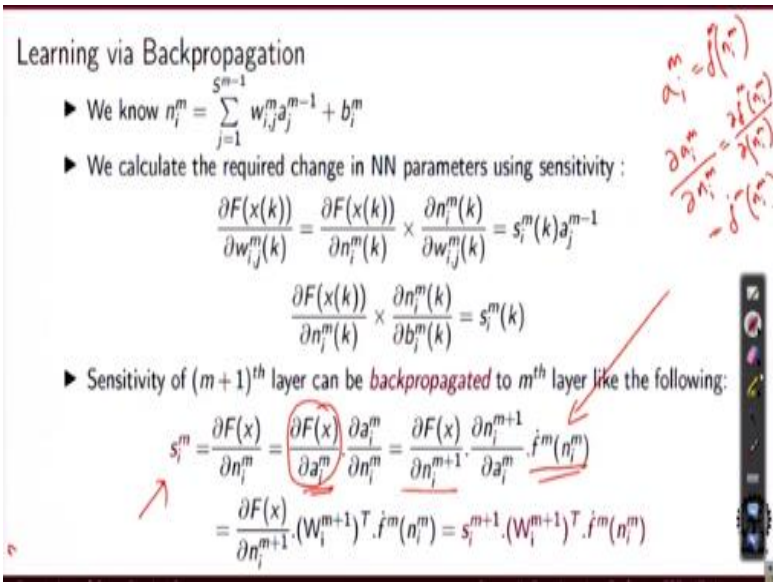
Observe that all these things are happening on the k-th stage estimate which is fine. So if I take this derivative with respect to $w_{i,j}$ in the m-th stage for the k-th step of the iteration, then I will definitely get a_j^{m-1} no doubt about that, right. So here I get a_j^{m-1} and the first term is my definition of sensitivity basically the rate of change of F with respect to the output of that stage before the activation, which is n okay. So fine.

And similarly for the other one which is the term we have not written here is basically this term del F del b. For that term, we can again apply the chain rule del F x k del $n_i^m(k)$ times del $n_i^m(k)$ del $b_i^m(k)$. Now again, if you see $n_i^m(k)$ and you have b i m, so derivative of del $n_i^m(k)$, $n_i^m(k)$ and del $b_i^m(k)$ that is 1 right. So this becomes one and all remains is this which is again your sensitivity, okay.

So so fine. You have $s_i^m(k)$ as here and here you have $s_i^m(k)$ times a_j^{m-1}. So now we have the sensitivity. The good thing about the sensitivity is it can be back propagated to the m-th layer. The question is well, I mean, if you if you look at the structure of this equation that we defined earlier, we are giving a weight update rule, right. But where am I supposed to do the weight update?

I am going to do it in the m-th layer and definitely, that information needs to go back to the previous layer. And similarly, there will be a weight update happening in the previous layer. That is how it should work right. So if I say that well I have done this update in the m plus 1-th stage, we will backpropagate it to the m-th layer like this following.

(Refer Slide Time: 37:52)



So suppose we are doing any sensitivity computation here, we define s_i^m. So we write that as del F del n_i^m. So that can be del F del a i m that is the activation times del a_i^m

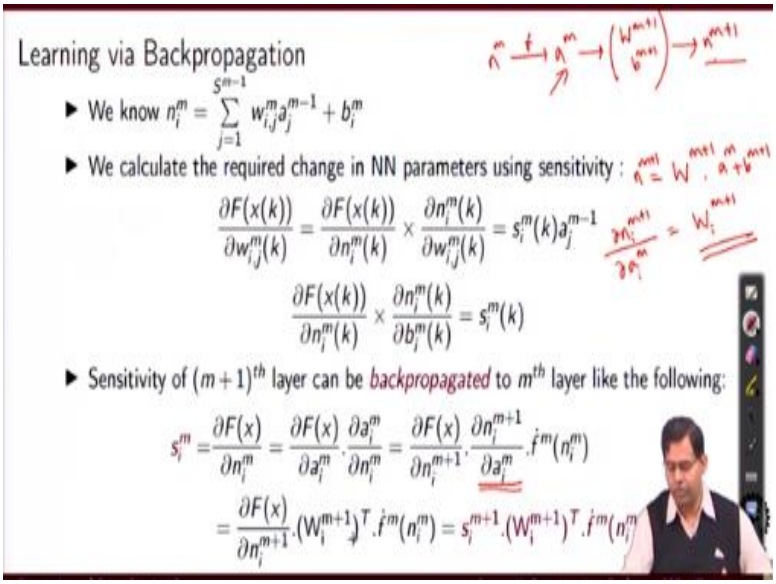
del n_i^m right. So we can write this. Now this also means that well if I take this part, on this if you apply the chain rule you can write del F del n i at m + 1 times del n i at m + 1 with respect to del a_i^m okay.

And you have this this thing which is derivative of the activation output with respect to the derivative of the activation input, right. This is nothing but the activation functions derivative, right. Because if you see a_i^m is nothing but the activation m-th layer with argument n_i^m right. So if I take derivative of a_i^m with respect to derivative of n_i^m, that is derivative of basically del del x f x form we have here.

So it is just you can write it in shorthand as, that is what we have here. Basically a derivative of the activation function with respect to its argument itself, okay. So let us understand what we really did. Sensitivity was defined with respect to the the loss, the loss' derivative with respect to that layer's output before activation. We changed it with that layer's output after activation times the derivative of the activation, okay.

And then what we said is well, that layer's output after activation can be written like this, that that layer's output with respect to the next layer's n value. And the next layer's n value is changed with respect to the current layer's activation, okay. There is a reason why we are doing this, which is simple. So you see, now it can transform it to this right, which is the first thing remains same, derivative of F with respect to n.

(Refer Slide Time: 40:57)



So if I draw a picture, so this is, so let us draw a nice picture here. So you have n_i^m, going through this activation f and it is generating a_i^m okay. And then it is going through this w^{m+1}, b^{m+1}, the m + 1-th layer and it is generating n^{m+1}, right. So that is that is what is happening here, right. So essentially, what what we want is the derivative of F derivative of F with respect to with respect to this, right.

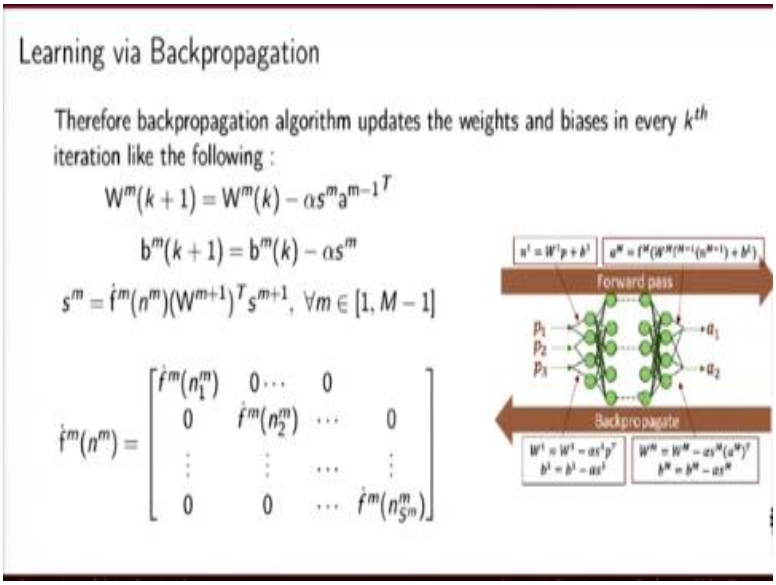
Because that is how we have transformed the equation. So we keep it as this, derivative of F with respect to n^{m+1} its i-th component times what did we really have? The derivative of the output here okay, this one with respect to this, the derivative of this with respect to this. So I mean, this is obvious, because what is the relation? n^{m+1} is very simply W^{m+1} times a m plus b^{m+1}. That is how it is right?

So if I take del n^{m+1} with respect to derivative of a^m, we must get W^{m+1} and if that is happening for all the i-th components, then that is it, W_i^{m+1} in its transpose form of course, because the matrix is represented that way. So fine, that is what we get, right? So this is this is the reason why we are writing sensitivity in this form, and we are linking it with the next layer, okay.

So because we are now able to write the sensitivity in terms of the next layer sensitivity, the next layer weight matrix, and this current layer's activation function's derivative.

So you can see we are, we are changing something here in the next layer. So if I change if you update the sensitivities, if we if we update the w's here, with that we can now update the sensitivity here in the in the previous stage, right. So that is the trick. That is about backpropagation.

(Refer Slide Time: 43:51)



So this algorithm is updating the weights and biases in every case iteration in this simple rule, right. So what we are happening we are doing is suppose we have in the in the in the in the k-th update, we have some estimate of W^m okay. And we have some estimate of s, we have some we have and we have we have the values of a etc., etc., okay. Now we apply this learning rate equation here.

So if you just go back, so this was it right, and we had this and then we were trying to figure out what is this del F del w and then we figured out that well, del F del w is nothing but s i a, some a this, s i I mean s times a, s m times a m - 1. And then we came up with this idea that well it can be represented by the next stage values of s w and this stage activation's derivative, right.

So once these things are known, what can I write overall? Overall I can see that well w m in the k + 1-th stage in the k + 1-th stage in the vector form can be linked up with W m in the k-th stage right minus this thing, which is the learning rate is there, and then

we needed the derivative of F with respect to W and that became my derivative of F with respect to W became s i m and s i m times a – 1.

And then we have figured out that well how to write that in terms of s m + 1. So you have s m then a m - 1. So let us write those things here. s m m – 1 and the the learning rate right there. And b if you see earlier b, this was b right. b was this b in the k + 1-th stage equal to b in the k-th stage minus alpha times s, right? So the minus alpha times s. So these are all in the stages here.

So this is this is this is right coming right from these two equations, okay. You have got del F del w as s a, s a m – 1 and you just put it there W^m minus alpha times s m a m - 1 transpose. So that is how it comes. And similarly, for the equation in the b. So this way you get the update equations for W and b. And the thing is, we have already figured out that well, how to link the sensitivities across stages using these equations.

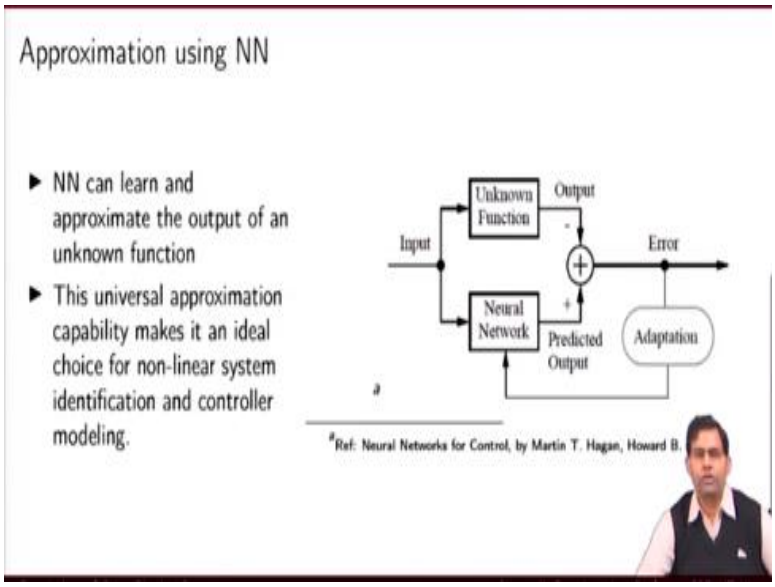
How the sensitivity in the m-th stage of the layer can be backpropagated to the m + 1-th layer using this equation. So I mean sorry the the I mean the value of sensitivity and weights in the m + 1-th layer how they can be used to update and backpropagate to the previous layer, right. So what we can do is well, we can figure out all the things in the previous layer and using using those values of s^{m+1} then w^{m+1} etc., you can now update s in the previous layer.

Once you have updated this in the previous layer, see I already know what is my a^{m-1} etc., values right. So once I have this knowledge of s m you can put this value of s m here and you can evaluate for the previous layers also, okay. So in this way, I have this update equations across these time steps, case time step values of W and b I can use along with the sensitivity values to update the k + 1-th time stay values of W and b, okay.

And we backpropagate the sensitivity values in this equation and overall for, what about the activations derivative, we need the activations derivative, well that is easy to compute, because the activations have got their well-formed formula right. So overall here I have in backpropagation what we call as a forward and backward pass combination.

So in the forward pass we have this this this computations of n is done right. So what we do is well, we compute the I mean given these inputs p_1's and the outputs a's right, you compute these layer values, and you compute these values of a's here at the output. And then what you do is you use the backpropagation equations, and you will just figure out what will be the weight updates that are happening and what will be the sensitivity value updates that are happening okay, in the backward pass.

(Refer Slide Time: 49:03)



And this will continue and and in that way you will get the uh you will get the values of the W's and the b's figured out in such a way that you have the function I mean with the minimized amount of loss and of course, once the value of the loss is minimized, you have it nicely approximating the unknown polynomial, I mean the unknown relation in terms of a complex polynomial, because this is what you are trying to evaluate.

And you start with a seed, you keep on doing these forward and backward passes and that is it you have the values done. So, with this, we will end the lecture here. Thank you for your attention.