

**Foundations of Cyber Physical Systems**  
**Prof. Soumyajit Dey**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology-Kharagpur**

**Lecture - 48**  
**Neural Network (NN) Based Controllers in CPS (Contd.)**


Welcome back to this lecture series on Foundations of Cyber Physical Systems. So in the last lecture, we had been discussing this topic that is neural network based controller design for cyber physical systems. And with regard to that, we started introducing ourselves to this issue of how to design a trained neural network, and we were discussing that well, why these update rules should be there.

**(Refer Slide Time: 00:54)**

---

Learning via Backpropagation

- ▶ Following is the *backpropagation* algorithm to update the network parameters of every  $m$ -th layer in every  $k$ -th iteration following below equations ( $0 < \alpha < 1$  is learning rate.):
$$w_{i,j}^m(k+1) = w_{i,j}^m(k) - \alpha \frac{\partial F(x(k))}{\partial w_{i,j}^m(k)}$$
$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{\partial F(x(k))}{\partial b_i^m(k)}$$
- ▶ The sensitivity  $s_i^m$  of loss function  $F$  w.r.t. every  $i \in [1, Q]$ -th net input is the key element that is actually *backpropagated* in order to update the weights and biases of NN.
$$s_i^m = \frac{\partial F(x)}{\partial n_i^m}$$



And if you recall, these are the update rules that we have been talking about that suppose, you have a multi-layer network and in some  $m$ -th layer you want to update the weights the  $w_{i,j}$  weights and the bias  $b_i$  from some value which is this in the to to this. So this is the value that has been updated in the  $k$ -th step and this is the value that will be updated in the  $k + 1$ -th step.

And the idea was that well there would be this gradient that is going to be computed and this gradient would be multiplied by the learning rate, and then accordingly the

weight and the bias is going to be updated. And in this context, we also defined what is sensitivity of the loss function, which was like derivative of the loss with respect to this value that is being computed n, right.

So basically, we call it sensitivity, because it tells that well how much the loss is going to change with respect to the value that is being computed by the, by the neural network. So fine, let us, let us move further on this.

**(Refer Slide Time: 02:06)**

Learning via Backpropagation

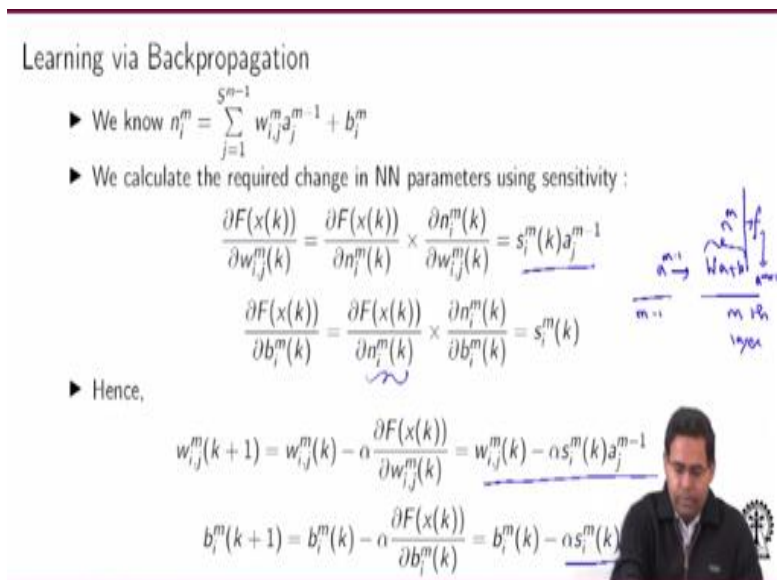
- ▶ We know  $n_i^m = \sum_{j=1}^{s^{m-1}} w_{ij}^m a_j^{m-1} + b_i^m$
- ▶ We calculate the required change in NN parameters using sensitivity :

$$\frac{\partial F(x(k))}{\partial w_{ij}^m(k)} = \frac{\partial F(x(k))}{\partial n_i^m(k)} \times \frac{\partial n_i^m(k)}{\partial w_{ij}^m(k)} = s_i^m(k) a_j^{m-1}$$

$$\frac{\partial F(x(k))}{\partial b_i^m(k)} = \frac{\partial F(x(k))}{\partial n_i^m(k)} \times \frac{\partial n_i^m(k)}{\partial b_i^m(k)} = s_i^m(k)$$

▶ Hence,

$$w_{ij}^m(k+1) = w_{ij}^m(k) - \alpha \frac{\partial F(x(k))}{\partial w_{ij}^m(k)} = w_{ij}^m(k) - \alpha s_i^m(k) a_j^{m-1}$$

$$b_i^m(k+1) = b_i^m(k) - \alpha \frac{\partial F(x(k))}{\partial b_i^m(k)} = b_i^m(k) - \alpha s_i^m(k)$$


So what we saw is that well these derivatives can be expressed in a nicer form using this chain rule of differentiation that you can write these derivatives in terms of the sensitivity. That means, the first derivative here would be sensitivity times derivative of n with respect to derivative of weight. And given this formula relating n and the weights, it becomes sensitivity multiplied by these inputs itself of the m – 1th stage.

So if you just draw a brief picture okay, so let us say you have this output coming from the m – 1th layer and this is the mth layer. So the output that comes is  $a^{m-1}$ . I am not writing the j-th index or i-th index, those thing, I am writing this thing as a whole. So this goes and this case, multiply it with this weight here, right. So this W gets multiplied with a, and then the bias gets added. And so that kind of is your n in the m-th layer, right.

And this would now, so this  $n$  would now go to the activation  $f$  and this is going to generate this  $a^{m+1}$ , the  $m+1$ -th stage, right. So in a way, if you take the derivative of this  $n$  with respect to the weights, you get the  $a^{m-1}$  values right, which is the  $a-1$  and that is what you have. And if you take the derivative here of  $F$  with respect to  $b$ , and then you express it in terms of  $n$ , so then this first term becomes the sensitivity again, right and this term is just 1 from this formula. So overall, you can write this update rules from the  $k$ -th to the  $k+1$ -th step using this, right.

You have the update at  $k$ , related with minus alpha  $s_i^m$ . And in the  $k$ -th step with the activation from the previous stage, right. And similarly, you can relate the value of bias in the  $k+1$ -th step with the bias in the  $k$  plus  $k$ -th step along with the learning rate multiplied by the sensitivity.

**(Refer Slide Time: 04:35)**

Learning via Backpropagation

In vector form,

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha s^m(k) \mathbf{a}^{m-1T}$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha s^m(k)$$


Sensitivity of  $(m+1)^{th}$  layer can be *backpropagated* to  $m^{th}$  layer like the following in any  $k$ -th step: [We can use the  $k$ -th step  $W$  update if required, (it will be)]

$$s_i^m = \frac{\partial F(x)}{\partial n_i^m} = \frac{\partial F(x)}{\partial a_i^m} \cdot \frac{\partial a_i^m}{\partial n_i^m} = \frac{\partial F(x)}{\partial n_i^{m+1}} \cdot \frac{\partial n_i^{m+1}}{\partial a_i^m} \cdot \dot{f}^m(n_i^m)$$

$$= \frac{\partial F(x)}{\partial n_i^{m+1}} \cdot (\mathbf{W}_i^{m+1})^T \cdot \dot{f}^m(n_i^m) = s_i^{m+1} \cdot (\mathbf{W}_i^{m+1})^T \cdot \dot{f}^m(n_i^m)$$

$n_i^m = f^m(n_i^m)$   
 $\Rightarrow \frac{\partial n_i^m}{\partial a_i^m} = \dot{f}^m(n_i^m)$

So, [ignoring the  $k$ -notations here]

$$s^m = \dot{f}^m(n^m) (\mathbf{W}^{m+1})^T s^{m+1}, \forall m \in [1, M-1]$$


So overall, if you are writing this in the vector form, then what you have is this expression would come in like this in the vector form, right. So the overall weight matrix of the  $m$ -th stage can be will have the  $k+1$ -th update as per this. And the  $B$  matrix in the  $k+1$ -th stage will have update as per this.

And the other requirement as you can see, so you have a refinement of both  $W$  and  $b$  happening for all the layers based on these values, I mean from the  $k$ -th to the  $k+1$ -th

state, right. So this thing can be done for the entire network, right. So we can keep on computing for the entire network that what is what is the mechanism using which the weight will be recomputed and the bias will be recomputed.

You can do this for the entire network, right. But the next question that comes is well, I am doing this, but how is going how is this going to, I mean, will there be any change again? So suppose I have already applied this thing. Now how in again in the next step is the W or b going to change? The point is, well with these rules that we have written, we have updates happening on the weights and, and the b value.

But that is not the only thing. The other thing is, the sensitivity value also needs an update. So what happens is we should be able to link up sensitivity value across layers. And let us do that here. So let us let us see that how the sensitivity value can be defined. So this was our definition of sensitivity, right. So this was our definition of sensitivity and that means derivative of the loss with respect to the value that is computed.

So express it in terms of a and again we use the chain rule of differentiation. Now as you can see that then, what you have is if you if you look at this next term, which is a and n. So if you see n is acted on by the activation and uses a right. So effectively  $f^m$  is activation in the m-th layer. This we have this relation. So that would that would really mean if I am doing this  $\delta a_i^m$ , delta that is actually a derivative of  $f^m, n_i^m$ .

So that is the derivative. So that is how this second term becomes this derivative. And what about this first term, derivative of F with respect to derivative of  $a_i^m$ ? So let us let us try to write it in a different way. So you can write it like derivative of F with respect to derivative of n in the next stage. So why is that important? Let us try to understand. Again, let us understand how the different layers would be linked up.

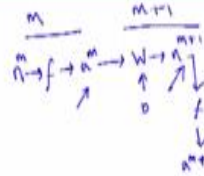
**(Refer Slide Time: 08:03)**

## Learning via Backpropagation

In vector form,

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha s^m(k) \mathbf{a}^{m-1T}$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha s^m(k)$$



Sensitivity of  $(m+1)^{\text{th}}$  layer can be *backpropagated* to  $m^{\text{th}}$  layer like the following in any  $k$ -th step: [We can use the  $k$ -th step  $W$  update if required, (it will be)]

$$s_i^m = \frac{\partial F(x)}{\partial n_i^m} = \frac{\partial F(x)}{\partial a_i^m} \cdot \frac{\partial a_i^m}{\partial n_i^m} = \frac{\partial F(x)}{\partial n_i^{m+1}} \cdot \frac{\partial n_i^{m+1}}{\partial a_i^m} \cdot f^m(n_i^m)$$

$$= \frac{\partial F(x)}{\partial n_i^{m+1}} \cdot (\mathbf{W}_i^{m+1})^T \cdot f^m(n_i^m) = s_i^{m+1} \cdot (\mathbf{W}_i^{m+1})^T \cdot f^m(n_i^m)$$

$$n^{m+1} = \mathbf{W} \mathbf{a}^m + \mathbf{b}$$

$$\frac{\partial n^{m+1}}{\partial \mathbf{a}^m} = \mathbf{W}$$

So, [ignoring the  $k$ -notations here]

$$s^m = \mathbf{f}^m(n^m) (\mathbf{W}^{m+1})^T s^{m+1}, \forall m \in [1, M-1]$$



So you have the  $m-1$ -th layer and you have the  $m$ -th layer. So in the  $m-1$ -th layer, you have the output as or sorry, let us try to link up  $m$ -th and  $m+1$ -th layer that will be more relative with this expression that we have. So we have the  $m$ -th layer, and we have the  $m+1$ -th layer. So that is kind of a sequence, right. So  $n^m$  acts acted upon by activation  $f$ , gives you a  $n$ .

Then you have the weight of the  $m+1$ -th layer to give you del I mean  $n$  of  $n^{m+1}$ . And then it is acted upon by the activation layer again and you get the final value  $a^{m+1}$ , right? So what we are saying is well, we have the loss with respect to  $a$  in the  $m$ -th state which is this, but rather than that, let us express it in terms of this. We will see why.

So instead of writing this you can write the del  $F$  del  $n_i^{m+1}$  and then take a derivative of delta  $n_i^{m+1}$  with respect to derivative of  $a^m$ , right. So effectively, we want the derivative of  $F$  with respect to  $a$ , and then we express it in terms of the, because we want to link up the different layers, right.

So we want to we take the derivative of  $F$  with respect to  $m+1$ -th layer, and then take the derivative of  $n$  in  $m+1$ -th layer with respect to the derivative of  $a$  in the  $m$ -th layer, okay. So if you do that, what what what does it come come out to be? So if you see you

have this relation between  $n$ 's and  $W$  right. So you have we have something like this, right. So if you take this derivative you get this  $W$ .

But this  $W$  is of the  $m + 1$ -th layer, right. So the second derivative, this term gives you this. And you still have the first term. But by the way, what is the first term? So if you see, this is this is just the definition of sensitivity for the  $m + 1$ -th layer, right. So then you have sensitivity of the  $m + 1$ -th layer, multiplied by weight in the  $m + 1$ -th layer, its transpose, multiplied by the derivative of  $f$ , right, the activation.

So overall, we can link up the  $m$ -th layer and  $m + 1$ -th layer sensitivities like this, that  $s^m$  is nothing but the derivative of  $F$ , multiplied by the transpose of  $W^{m+1}$  multiplied by  $s^{m+1}$ . So so that is how that is how it is. That is how this sensitivities are going to be linked up across layers. But then what is the implication of these?

**(Refer Slide Time: 12:02)**

Learning via Backpropagation

Therefore backpropagation algorithm updates the weights and biases in every  $k^{th}$  iteration like the following :

$$W^m(k+1) = W^m(k) - \alpha s^m a^{m-1T}$$

$$b^m(k+1) = b^m(k) - \alpha s^m$$

$$s^m = \dot{f}^m(n^m) (W^{m+1})^T s^{m+1}, \forall m \in [1, M-1]$$

$$\dot{f}^m(n^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \dots & 0 \\ 0 & \dot{f}^m(n_2^m) & \dots & 0 \\ \vdots & \vdots & \dots & \vdots \\ 0 & 0 & \dots & \dot{f}^m(n_{S_m}^m) \end{bmatrix}$$

The slide contains a diagram of a neural network layer with nodes  $n_1, n_2, \dots, n_{S_m}$  and weights  $W^m$ . It shows the forward pass  $a^m = f^m(W^m(n^{m-1}) + b^m)$  and the backward pass  $s^m = \dot{f}^m(n^m) (W^{m+1})^T s^{m+1}$ . Handwritten notes in blue ink show the relationship between layers  $M-2, M-1, M$  and weights  $W^{M-1}, W^M$ . A small video inset shows a person speaking.

So in effect, what we have is that we have update rules for the weights as well as an update rule for the for the sensitivity. So you you need to have a method by which you initialize the weight's sensitivity value and the and the bias. And there are well-known methods in neural network literature using which they are going to be initialized. You can study this material, which I have cited here.

And also the authors of the same material, they have organized a book on neural networks, you can study from there that there are well-known techniques of how to initialize these values. So that means if you initialize  $W_0$ ,  $s_0$  and  $b_0$  from there you can keep on computing these  $W$  values. They are  $k$ -th estimate and then  $k + 1$ -th estimate like that, right?

So right now, let us say you have some estimate of  $s$  here, the sensitivity and you are using that estimate, to compute using the current estimate of  $W$  what is the, what is the next estimate of  $W$  and from the current estimate of  $b$ , what is the next estimate of  $b$ . In that way, you move through what we call as a forward pass through the entire network, and you estimate these values of  $W$ 's and  $b$ 's okay.

So once you have done that for the entire network, you will have newly computed values of  $W$ 's and  $b$ 's for the entire network. Now you see, so you have you have updated the  $W$ 's right. So once you have updated the  $W$ 's, that means you have a you have a change here. Once you have a change here, you use this change to recompute your sensitivities. That is the point.

So that means now after this forward part, whatever earlier was your  $W$  and  $b$  they have all changed. So you start from the last layer okay, for which you have the sensitivity  $s_m$  known. So use  $s_m$  and this updated value of  $W_m$  to compute  $s^{m-1}$ . Use  $s^{m-1}$  and the updated value of  $W^{m-1}$  to compute  $s^{m-2}$  like that.

And this is what we call as a backward pass, and you go back and recompute all the sensitivity values, okay. So that is like an update on the sensitivities. Once the sensitivity values are updated, you use these updated sensitivity values to recompute again in the forward pass, the next forward pass further updates to happen on the  $W$ 's and the biases.

Use these updates on the biases and start with the last computed sensitivity value at the end and the last layer and again use a backward pass to go back. So that is the back and forth computation which is part of the backpropagation algorithm. And that is how the neural network will finally converge through to a to a to a position from where the loss function is not getting minimized anymore.

So let us understand what is really is happening. Every time when you are recomputing the sensitivities you are essentially refining your gradient. You are you are essentially refining your estimate of the loss function with respect to some position in the state space and you are retuning your values of weights and you are trying to figure out what are the values of weights for which the loss function is going to be more minimized, more minimized like that.

So that is basically a gradient descent kind of technique through which you are making the loss function approach a minimum value. So there are well-known optimization theory literature on which you can find more details on this. Here we are just telling you that how this is kind of getting done more in a mechanical way. Fine.

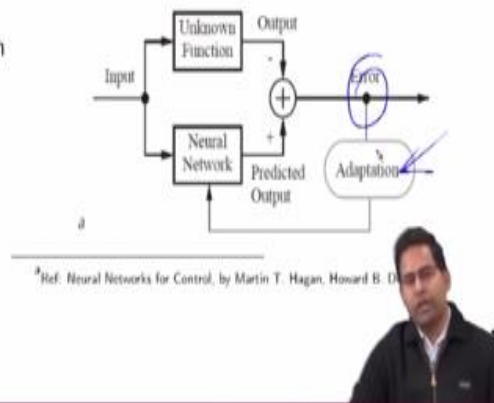
So we have an idea that well how the neural network training actually happens and let us now figure out that well, what how we can really use these kind of trained neural networks in our context of cyber physical system design.

**(Refer Slide Time: 16:06)**



## Approximation using NN

- ▶ NN can learn and approximate the output of an unknown function
- ▶ This universal approximation capability makes it an ideal choice for non-linear system identification and controller modeling.



So when we are doing this kind of CPS design, neural networks can be useful for their nature that they are very good at approximating some unknown functions. So if you look into this literature you will see that they actually provide you lot of interesting examples that how with different choices of weights and bias values, you can make the neural network approximate a wild class of algebraic functions, okay.

So so for any kind of unknown behavior with sufficient data that is gathered, sufficient training data that is gathered, it should be possible to create a neural network based representation of of of that unknown function. So this is what can be used to create and approximate model of a nonlinear system and that is how it is used. Like it can be used to model unknown plant dynamics.

Suppose there is a plant which for which you want to design a controller, but the plant's model is unknown. So what we one can do is one can give inputs to the plant and check what is the output coming from the unknown model and one can also use these kind of data set to train an approximate neural network and then you can you can keep on doing this like you have the input and output and you also have the predicted output from the neural network.

So compare these two outputs to get the error function and retune the neural network to using some training algorithm, which we denote here by this adaptation block, where the training algorithm's objective is to keep on minimizing the error.

So you have inputs and based on that you have errors and you ask the training algorithm that well for this neural network, you optimize the weights more and more, so that with every iteration, whatever the unknown function is outputting and whatever you are outputting the neural network is outputting the error must get minimized, okay.

So if so here whenever I talking, we talk about adaptation, we mean such a training method for which the objective is to minimize this error, okay. So this is kind of a very standard technique, which is used for nonlinear system identification. You can see that in MATLAB, there is a system identification toolbox using which these kind of methods can be automatically done.

So you can model a plant dynamics using this. And you can learn the plant's dynamics, and not only that, you can use these kind of technique to figure out that well, for a plant I mean, so that is about modeling a plant's dynamics. But not only that, you can also use neural network-based approximations to create a controller model. We will talk about that more, maybe in the next lecture. Thank you for your attention.