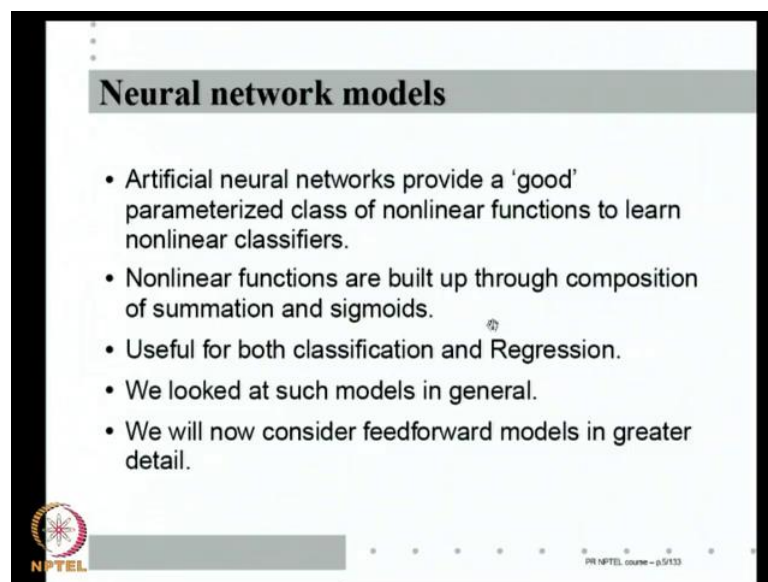


Pattern Recognition
Prof. P. S. Sastry
Department of Electronics and Communication Engineering
Indian Institute of Science, Bangalore

Lecture - 27
Multilayer Feedforward Neural networks
with Sigmoidal activation functions

Hello and welcome to this next lecture on Pattern Recognition. We have started looking at methods for learning non linear classifiers and regression models, earlier we have looked at linear classifiers and how to learn linear regression mainly using least squares methods. So, last class we started looking at neural networks as a model for learning non linear classifiers.

(Refer Slide Time: 00:43)



The slide is titled "Neural network models" and contains a bulleted list of five points. The text is as follows:

- Artificial neural networks provide a 'good' parameterized class of nonlinear functions to learn nonlinear classifiers.
- Nonlinear functions are built up through composition of summation and sigmoids.
- Useful for both classification and Regression.
- We looked at such models in general.
- We will now consider feedforward models in greater detail.

The slide also features the NPTEL logo in the bottom left corner and the text "PR NPTEL course - p51133" in the bottom right corner.

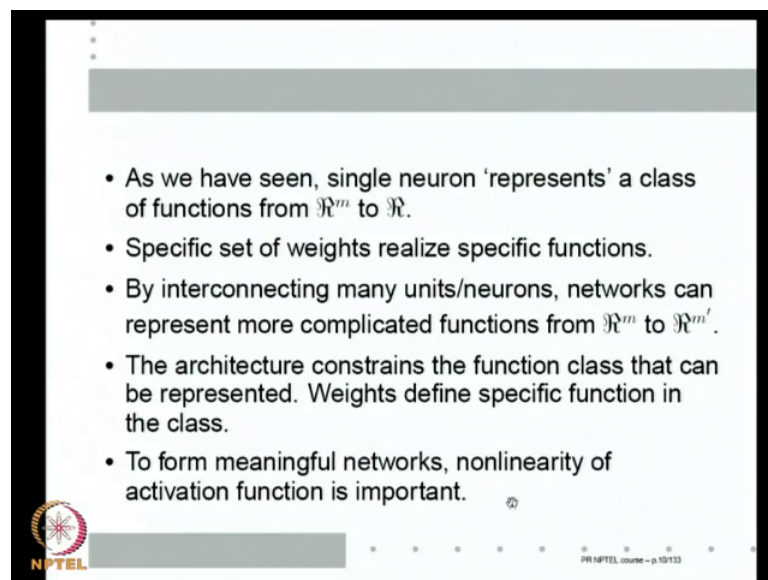
We have, we looked at neural networks in general historically what is the motivation for such models are and so on. But as I told you already for this class, our main motivation our interest in artificial neural network models, is that they provide a good parameterized class of non linear functions, because they provide a good parameterized class of non linear functions, that is useful for us to learn learn non-linear classifiers non linear regression models.

We have seen in last class non linear functions in the here are built through composition function to composition of summation and sigmoids, the models are useful both for classification and regression problems because essentially these networks can represent

any kind of non linear functions. So, including binary valued functions or multiply valued functions. So, of course, most of the time they represent continuous functions, we appropriately threshold things to get classifiers we will see the details, but essentially these models are useful for both classification problems as well as regression problems.

We already looked at the models in general and I have mentioned to you that we will be looking at only feed forward the, so called multilayer feed forward network models because those are the only thing that we will be using in this course. So, this class we will discuss a little more on multilayer feed forward and how one learns such models given some training data.

(Refer Slide Time: 02:15)



As you see in last class a single neuron, essentially represents a class of functions from \mathbb{R}^m to \mathbb{R} because it essentially it is output is some activation function applied on summation of weighted inputs. So, f of $\sum w_i x_i$, so given the number of this inputs and the kind of activation function, certain class of functions can be represented, so such an expression right.

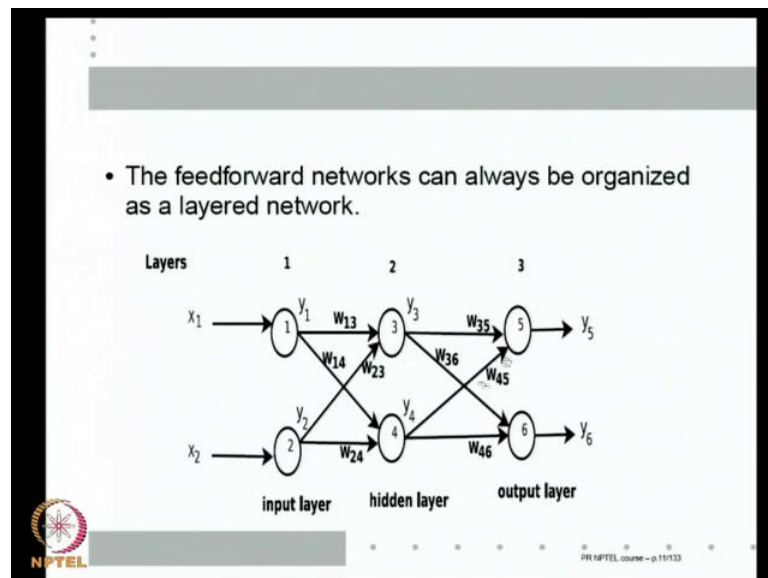
And specific set of weights, if I put specific w it realizes specific function, as you already seen things like perceptron and so on. Or essentially examples of such single neuron classifiers, by interconnecting many neurons, one can make the networks represent many more complicated non linear functions, right from arbitrary m dimensional real Euclidian

space to some other m prime dimensional real Euclidian space, where m and m prime could be arbitrary right.

So, when we use a multilayer network, essentially the architecture of the network, that is the number of layers we have, number of nodes we have in each layer, how we interconnected nodes all this, kind of constrains the class of functions that can be represented. So, a given network architecture is capable of representing certain family of functions and within that family when I put specific weights I will represent a particular function.

So, we normally choose an architecture based on what kind of functions we want to represent and then learn the weights. So, that the network can be particularised to specific function, by using the training data. So, we have also seen that, to form meaningful networks, nonlinearity of the activation function is very important. So, before we go into general notation for general feed forward network models let just take another look at a simple network.

(Refer Slide Time: 04:12)



So, as we have already seen all feed forward networks can organised into layered structures, that is I can organise the nodes into layers, so that any node say in layer 3 gets inputs only from nodes in layer 2. Similarly, nodes in layer 2 get inputs only from layer 1 as it is a any feed forward structure as long as there are no feedbacks, then always be put into this by having dummy duplicated nodes if need be.

So, here is a simple 3 layered network, the first layer gets external inputs. So, that is called the input layer, the outputs of the nodes in the last layer or what are given out as the final outputs of the network right. So, the layer 1 is always called the input layer, the last layer in this case layer 3 is the output layer, right everything in between is called hidden layer because these these nodes are not seen by the external environment at all.

These nodes are seen in the sense, the inputs come from the external world or external environment and these nodes are seen in the sense their outputs affect the external world or they are at the external outputs whereas, for these nodes, both their inputs and outputs are internal to their networks. So, they are hidden from the external environment so they are called the hidden nodes, in a as a notation here, we are representing the output of node i by y_i . So, output of node 1 is y_1 output of node 2 is y_2 output of node 3 is y_3 and so on.

A node that connects a a a a link that connects node i to node j will have some weight as we said this this is an optic weight, our notation is w_{ij} . So, link that connects node 1 to node 4 is w_{14} . So, what does any node do, as we have already seen last class it takes a weighted sum of inputs, that is you multiply whatever is the input on this line by that lines y weights and so on sum it up and then pass it through an activation function right.

And essentially by a notation we take up the input layer as special. So, when the external inputs come because all inputs into all nodes, we want to organise like this, we think of the input layer nodes as simply fanning out nodes. So, for the input layer the output is same as the input, right it is just a input port. So, y_1 will be x_1 , so that is what is going as the input into nodes 3 and 4 in this figure. So, let us say just to get our notation right, let us say we want to write the output of node 5, what will be the output of node 5 it is a weighted sum, so y_3 into this phase w_{35} and y_4 into w_{45} .

(Refer Slide Time: 07:08)

• The function represented by the above network can be written as

$$y_5 = f_5(w_{35} y_3 + w_{45} y_4)$$

The slide also features the NPTEL logo in the bottom left corner and a small inset image of a man in the bottom right corner.

So, if I want to write the output of node 5, I will write y_5 as f_5 that is the activation function of node 5, into it is net input which is $w_{35} y_3$ plus $w_{45} y_4$ right, $w_{35} y_3$ plus $w_{45} y_4$.

(Refer Slide Time: 07:22)

• The feedforward networks can always be organized as a layered network.

The diagram shows a feedforward network with three layers:

- Layer 1 (input layer):** Two input nodes labeled 1 and 2, receiving inputs x_1 and x_2 . Their outputs are y_1 and y_2 .
- Layer 2 (hidden layer):** Two nodes labeled 3 and 4. Node 3 receives inputs from node 1 (w_{13}) and node 2 (w_{23}). Node 4 receives inputs from node 1 (w_{14}) and node 2 (w_{24}). Their outputs are y_3 and y_4 .
- Layer 3 (output layer):** Two nodes labeled 5 and 6. Node 5 receives inputs from node 3 (w_{35}) and node 4 (w_{45}). Node 6 receives inputs from node 3 (w_{36}) and node 4 (w_{46}). Their outputs are y_5 and y_6 .

The slide also features the NPTEL logo in the bottom left corner and the text 'PR NPTEL course - p.11/133' in the bottom right corner.

Now, of course how do I get y_3 and y_4 by the same way, y_3 is its activation function let us say f_3 of its net input, that is y_1 into w_{13} plus y_2 into w_{23} right, you take the weight of sum of inputs into this node, pass it through its activation function you get y_3 similarly for y_4 .

(Refer Slide Time: 07:51)

• The function represented by the above network can be written as

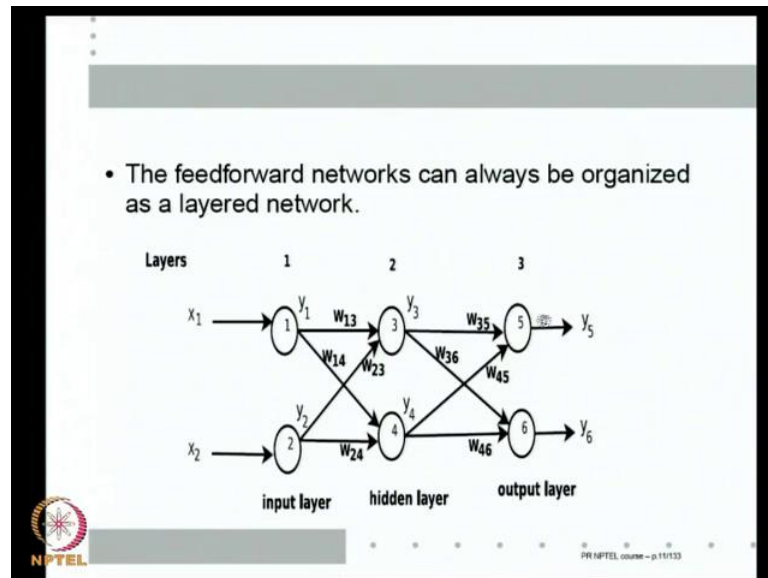
$$\begin{aligned}y_5 &= f_5(w_{35}y_3 + w_{45}y_4) \\ &= f_5(w_{35}f_3(w_{13}y_1 + w_{23}y_2) + w_{45}f_4(w_{14}y_1 + w_{24}y_2)) \\ &= f_5(w_{35}f_3(w_{13}x_1 + w_{23}x_2) + w_{45}f_4(w_{14}x_1 + w_{24}x_2))\end{aligned}$$

NPTEL PRE NPTEL course - p 14133

So, I can get, I can write y_3 as f_3 of $w_{13}y_1$ plus $w_{23}y_2$. Similarly, y_4 is $w_{14}y_1$ plus $w_{24}y_2$. Now we have already said y_1 and y_2 are nothing but the inputs. So, I can substitute. So, the y_5 is a function of the inputs x_1 x_2 where function is given by this. So, it is essentially I have summation a sigmoid now, these are composed. So, I have a summation and sigmoid and now, on top of that I have another linear summation and another sigmoid this is how, by composing summation sigmoids we are building more and more complicated non linear functions right.

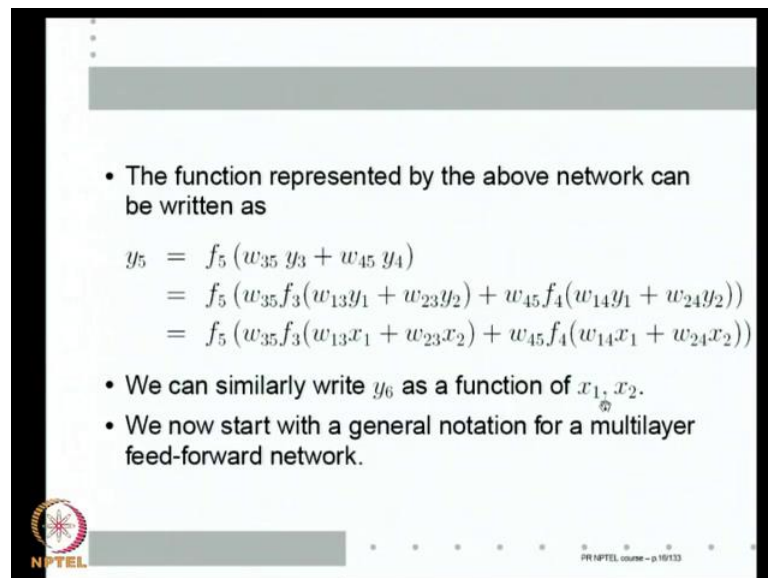
Also this tells you why the activation function should be linear because if it should be non linear. Suppose, f_3 is linear, then this is some linear function of x_1 and x_2 you are multiplying with a constant it will become another linear function of x_1 and x_2 this will also become a linear function of x_1 and x_2 you add them up you will get a linear function and you apply another linear function you will get another linear function. So, ultimately whether you have multiple layers or a single node, it will represent the same linear function. So, the activation functions have to be necessarily non linear, we need a little more we will see later on just any non linear function you want to.

(Refer Slide Time: 09:13)



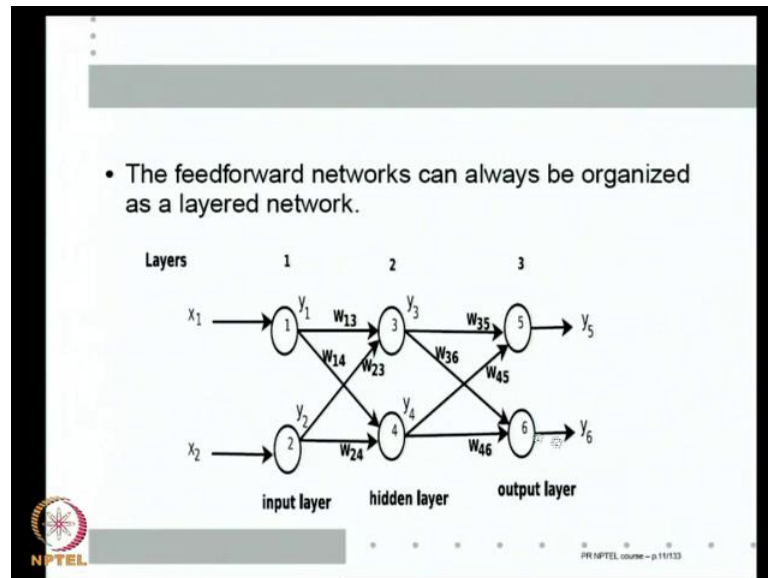
But, this is how we can represent the output of this network; right I hope the notation is clear right.

(Refer Slide Time: 09:21)



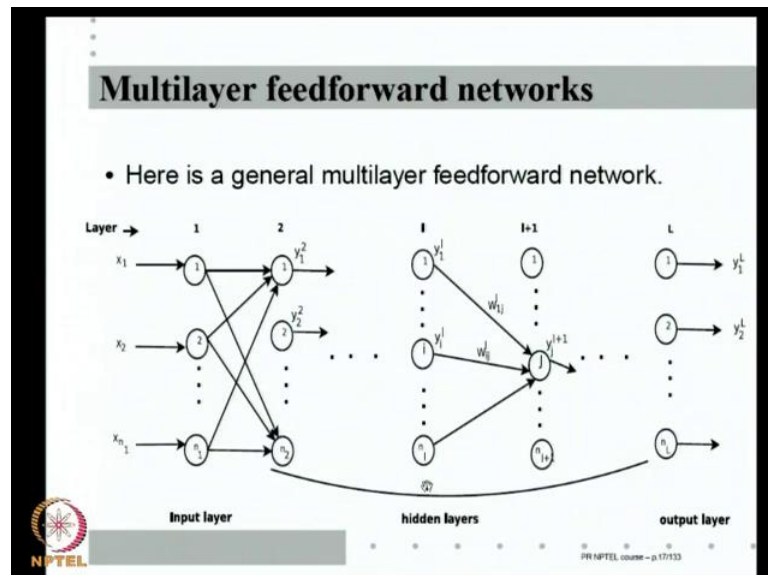
So, similarly we can represent y_6 also as a function of x_1 and x_2 right.

(Refer Slide Time: 09:30)



But, the problem with this way of writing is if I have an arbitrary network with arbitrary number of layers, I cannot write it as 1 2 3 4 because I do not know how many numbers there are. So, I need some more consistent notation to represent any general network. So, first let us start with some general notation to represent a general network.

(Refer Slide Time: 09:46)



So, this is going to be our notation. So, we have capital L number of layers we will always use capital L for the number of layers. So, layer 1, 2, so on as said layer 1 is input layer, layer l is output layer everything in between are hidden layers, there can be one or

more hidden layers it is a multilayer network. So, there can be any number of layers, there is the first layer is the input layer, last layer is the output layer everything in between are hidden layers, input layer gets external inputs x_1, x_2, \dots . So, within a layer first layer has n_1 nodes, second layer has n_2 nodes and so on. So, l 'th layer has n_l nodes that is our notation.

(Refer Slide Time: 10:29)

Notation

- L – number of layers
- n_ℓ – number of nodes in layer ℓ , $\ell = 1, \dots, L$.

NPTEL logo and footer: PPI NPTEL course – p 171133

So, we have L layers, L is the number of layers, n_1 is the number of nodes in each layer, n_l is the number of nodes in layer l and layers go from 1 to l .

(Refer Slide Time: 10:40)

Multilayer feedforward networks

- Here is a general multilayer feedforward network.

Layer → 1 2 ... l $l+1$... L

x_1 → (1) → y_1^2 → ... → (1) → y_1^l

x_2 → (2) → y_2^2 → ... → (2) → y_2^l

\vdots → \vdots → \vdots → \vdots → \vdots

x_{n_1} → (n_1) → $y_{n_1}^2$ → ... → (n_l) → $y_{n_l}^l$

Input layer hidden layers output layer

NPTEL logo and footer: PPI NPTEL course – p 171133

We take the inputs layer nodes or external inputs, for anybody else the inputs come from the previous layer. So, for node let us say 2 in layer 2 the we will represent output as y_2^2 y superscript 2 superscript 2 denote the layer, subscript denote the number of node in that layer. So, a node j layer 1 will be y_{1j} , y superscript 1 subscript j and so on right. And all weights connect only between layers. So, in for in general if I take a layer l and layer $l + 1$ there will be a weight, connecting a node j in layer l say node i in layer l to node j in layer $l + 1$ right all all weights connect only a layer to it is next layer.

So, we will represent that such weight as w_{ij}^l , w_{ij}^l connects node i layer l to node j layer $l + 1$, I only need to do this notation only need to give me, the node number here, node number here and this layer number because I know what the layer number where the j is because it is a it is a layered network weights only go from 1 layer to the immediate next layer.

(Refer Slide Time: 12:04)

Notation

- L – number of layers
- n_ℓ – number of nodes in layer ℓ , $\ell = 1, \dots, L$.
- y_i^ℓ – output of i^{th} node in layer ℓ ,
 $i = 1, \dots, n_\ell$, $\ell = 1, \dots, L$.
- w_{ij}^ℓ – weight of connection from node- i , layer- ℓ to node- j , layer- $(\ell + 1)$.
- η_i^ℓ – net input of node- i in layer- ℓ
- Our network represents a function from \mathbb{R}^{n_1} to \mathbb{R}^{n_L} .

NPTEL
PRIPATEL course - p.23113

So, let us continue with the notation n_l is a number of nodes y_{il} is output of the i 'th node in layer l , w_{ij}^l is the weight connecting node i in layer l to node j in layer $l + 1$ right all weights connect only from the l to $l + 1$ for some layer number l . So, w_{ij}^l is the weight of connection that connects node i in layer l to node j in layer $l + 1$. So, l 's go from 1 2 up to capital L minus 1, right w_{ij}^{L-1} connects some node in layer $l - 1$ to node in layer l . So, there will not be any weight, whose superscript is capital L because I do not have any $l + 1$ plus first layer, as we represent η_{il} as the net

input to node i in layer l , as we seen net input is nothing but the weighted sum of inputs coming to it.

(Refer Slide Time: 13:14)

• The input layer gets external inputs.

• The outputs of the output layer are the outputs of the network.

• If we want to learn a function from \mathbb{R}^m to $\mathbb{R}^{m'}$ then we would have $n_1 = m$ and $n_L = m'$.

NPTEL
PR NPTEL course - p.26/133

So, what does this network represent, it represents a function from inputs to outputs how many inputs and outputs we have.

(Refer Slide Time: 13:22)

Multilayer feedforward networks

• Here is a general multilayer feedforward network.

Layer → 1 2 ... l l+1 ... L

x_1 → (1) → y_1^2 ... (1) y_1^l (1) y_1^{l+1} ... (1) y_1^L

x_2 → (2) → y_2^2 ... (l) y_l^l (l) y_l^{l+1} ... (2) y_2^L

\vdots \vdots ... \vdots \vdots ... \vdots

x_{n_1} → (n_1) → $y_{n_1}^2$... (n_1) $y_{n_1}^l$ (n_1) $y_{n_1}^{l+1}$... (n_1) $y_{n_1}^L$

Input layer hidden layers output layer

NPTEL
PR NPTEL course - p.17/133

The number of inputs is equal to number of nodes in layer 1 by our notation is n_1 , number of outputs will be number of nodes in layer l by our notation is $n_{\text{subscript } l}$; so $n_{\text{subscript capital } L}$.

(Refer Slide Time: 13:36)

Notation

- L – number of layers
- n_ℓ – number of nodes in layer ℓ , $\ell = 1, \dots, L$.
- y_i^ℓ – output of i^{th} node in layer ℓ ,
 $i = 1, \dots, n_\ell, \ell = 1, \dots, L$.
- w_{ij}^ℓ – weight of connection from node- i , layer- ℓ to node- j , layer- $(\ell + 1)$.
- η_i^ℓ – net input of node- i in layer- ℓ
- Our network represents a function from \mathbb{R}^{n_1} to \mathbb{R}^{n_L} .

NPTEL PRE NPTEL course - p.23/133

So, this network represents a function from n one dimensional real Euclidian space to n 1 dimensional real Euclidian space right this is our general notation.

(Refer Slide Time: 13:49)

- The input layer gets external inputs.
- The outputs of the output layer are the outputs of the network.
- If we want to learn a function from \mathbb{R}^m to $\mathbb{R}^{m'}$ then we would have $n_1 = m$ and $n_L = m'$.
- The nodes in input layer are just to fanout the inputs.
- All other nodes calculate net input as weighted sum of inputs and pass it through an activation function to compute outputs.

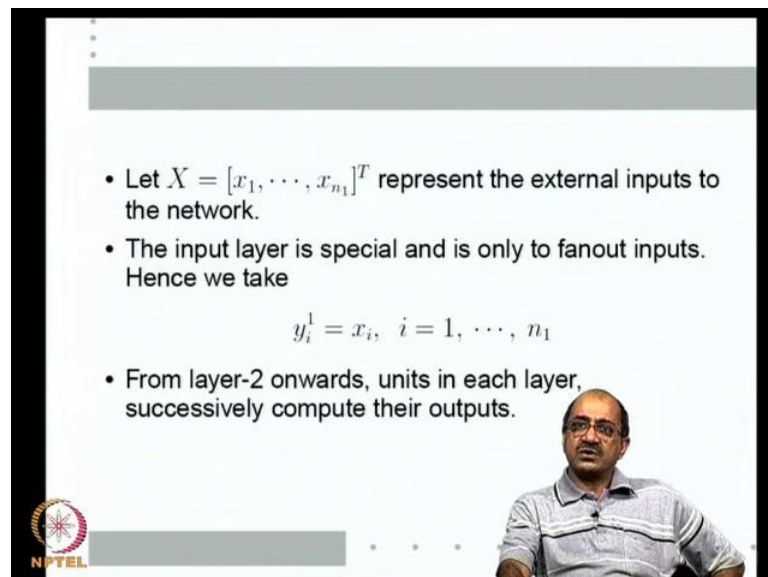
NPTEL PRE NPTEL course - p.23/133

Now, the input layer as we have seen gets external inputs right that is all the input layers job is to find out the external inputs, the outputs of the output layer are the outputs are in network. So, in what does this mean, if you want to learn a function from \mathbb{R}^m to $\mathbb{R}^{m'}$ right, then all it means is that n_1 will be m and n_L will be m' . So, if I am if

we want to represent a learn a function from m dimensional space to m prime dimensional space; that means, I need m input nodes and m prime output nodes.

So, number of nodes in the input layer will be m and number of nodes in the output layer will be m prime. The nodes in input layer is just to find out the inputs as we have already said right, they do not be any any weighted sum and non linear function computation. All other nodes essentially calculate their net input, as weighted sum of inputs that are getting and pass it through an activation function to compute their outputs, this is the generic structure in which the network functions.

(Refer Slide Time: 15:06)



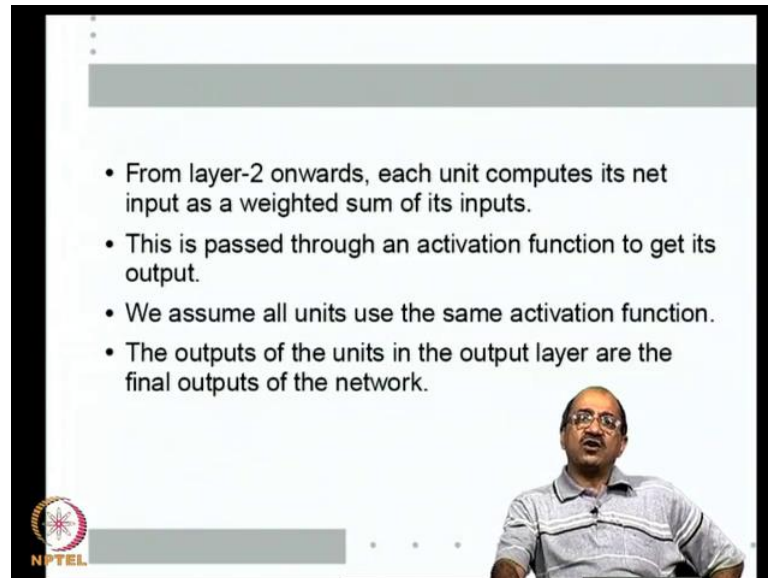
- Let $X = [x_1, \dots, x_{n_1}]^T$ represent the external inputs to the network.
- The input layer is special and is only to fanout inputs. Hence we take
$$y_i^1 = x_i, \quad i = 1, \dots, n_1$$
- From layer-2 onwards, units in each layer, successively compute their outputs.

So, let us say we represent by capital X the input all the inputs to the network. So, because I have n_1 input nodes capital X will have n_1 components will be a vector with n_1 components. So, let us say $x_1 \times n_1$ represent the components of capital X which represents the external inputs to the network. So, what does the input layer do the input layer as we seen is special is only to find out the output.

So, in layer 1 the output of node i which is y_i^1 is simply the i 'th input right because layer 1 is there only to find out the inputs, we can immediately write y_i^1 is equal to x_i for i is equal to 1 to n_1 . Now, what happens from layer 2 onwards, from layer 2 onwards units in each layer successively compute their outputs, see once all the outputs of layer 1 are are computed. Now, inputs to every node in layer 2 are ready. So, now, I can compute net inputs to nodes in layer 2 and hence there outputs.

Once all the outputs of layer 2 nodes are computed now, inputs for layer 3 are ready and, so on. So, from layer 2 onwards, units in each layer can successively compute their outputs how do they do this.

(Refer Slide Time: 16:24)



The image shows a video frame with a slide on the left and a speaker on the right. The slide contains the following text:

- From layer-2 onwards, each unit computes its net input as a weighted sum of its inputs.
- This is passed through an activation function to get its output.
- We assume all units use the same activation function.
- The outputs of the units in the output layer are the final outputs of the network.

The speaker is a man with glasses, wearing a light-colored shirt, sitting in front of a white background. In the bottom left corner of the slide, there is a logo for NPTEL (National Programme on Technology Enhanced Learning) featuring a stylized sun or starburst design.

Each unit computes its net input, as a weighted sum of its inputs right. So, a unit in layer 3 will compute its net input, as weighted sum of all the outputs of units in layer 2 to which it is connected and the weights of the weighted sum or the connection weights. So, once I calculate net input, it is passed through an activation function to get the output right. So, for simplicity of notation let us assume that all units with the same activation function then.

So, I will start from layer 1, layer 1 simply its output is simply equal to the input they are getting now, each unit in layer 2 compute its net input and hence its output, then layer 3 computes then layer 4 computes all the way up to output layer and once the output layer computes the outputs of all these nodes, those are the outputs of the network which I can give to the external world.

(Refer Slide Time: 17:26)

• The outputs of a typical unit is computed as

$$\eta_j^l = \sum_{i=1}^{n_{l-1}} w_{ij}^{l-1} y_i^{l-1}$$

$$y_j^l = f(\eta_j^l)$$

So, output of any typical unit is computed as follows right. So, we take some arbitrary unit, let us say unit j in layer l right, how does unit j in layer l computes it is output, as we said each unit computes it is output through 2 steps, first you have to compute the net input and then the output, what is the net input of unit j in layer l it is the weighted inputs you are getting, it is gets all its inputs from layer l minus 1.

So, essentially is y_i^{l-1} right, some node is the l minus 1 layer multiply. So, take it is output multiply by the weight, the weight connecting any node i in l minus 1 to this specific node j is w_{ij}^{l-1} . So, net input into node j in layer l is sum of $w_{ij}^{l-1} y_i^{l-1}$ where sum is over i and i varies from, i varies over the nodes in layer l minus 1. So, i varies from 1 to n_{l-1} . So, net input is i is equal to 1 to n_{l-1} $w_{ij}^{l-1} y_i^{l-1}$ right, that is it is that is the net input of a node in layer l.

Once I compute the net input, I compute the output by passing the net input through an equation function. So, this is how any any node typically computes it is solve a if if I want the output of this node. So, it I will look at this output. So, what it is connected to. So, those nodes output into the weight of that connection plus the next nodes output into weight of the connection and, so on find the net input and then the output.

Now, of course, in this that input we have not shown any bias input to the which is just taking only weighted sum of it is inputs, as we have already said last class each node can

optionally have a bias right, it is like as we want the linear model to be $w^T x$ only or $w^T x + b$ normally we need a b .

(Refer Slide Time: 19:40)

- If we want to include bias input, then

$$\eta_j^\ell = \sum_{i=1}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1} + b_j^\ell$$
- We can think of bias as an extra input and write

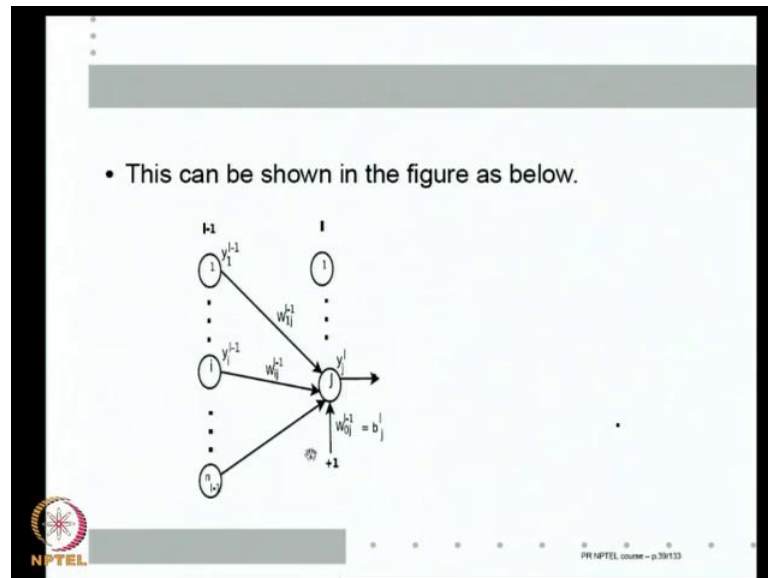
$$\eta_j^\ell = \sum_{i=0}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1}$$

where, by notation, $w_{0j}^{\ell-1} = b_j^\ell$ and $y_0^\ell = +1$ for all ℓ .

So, we can include a bias by adding some b_j for node j at the bias in calculating that input, very often we do not have to think of this as an extra special edition, we can think of bias as an extra input, by writing this as sum over i is equal to 0 to $n_l - 1$ instead of 1 to $n_l - 1$ $w_{ij}^{l-1} y_i^{l-1}$. So, compared to this sum, this sum because it goes from i is equal to 0 there is only 1 extra term namely w_{0j}^{l-1} into y_0^{l-1} , that term should represent b_j .

So, for that we take a notation, that w_{0j}^{l-1} after all there is no node 0 in any layer. So, there is nothing that connects from node 0 to node j . So, it is just a notation w_{0j}^{l-1} represent the bias weight for node j right, w_0^{l-1} represent the bias weight for node j in layer l right because of the way we put the notation that is how it works out and we always take the 0th node in each layer, so to say, to be something that is permanently struck at one. So, we simply think of y_0 for any any l is simply plus 1.

(Refer Slide Time: 20:59)



It is like, the same thing this node gets all its inputs from the previous layer plus 1 extra input, from something whose output is plus 1. So, this is like permanently connected to a plus 1 supply, right and this weight, weight on this connection is your bias. So, whenever we graphically want to show things, we will show the bias input like this right.

(Refer Slide Time: 21:24)

• If we want to include bias input, then

$$\eta_j^\ell = \sum_{i=1}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1} + b_j^\ell$$

• We can think of bias as an extra input and write

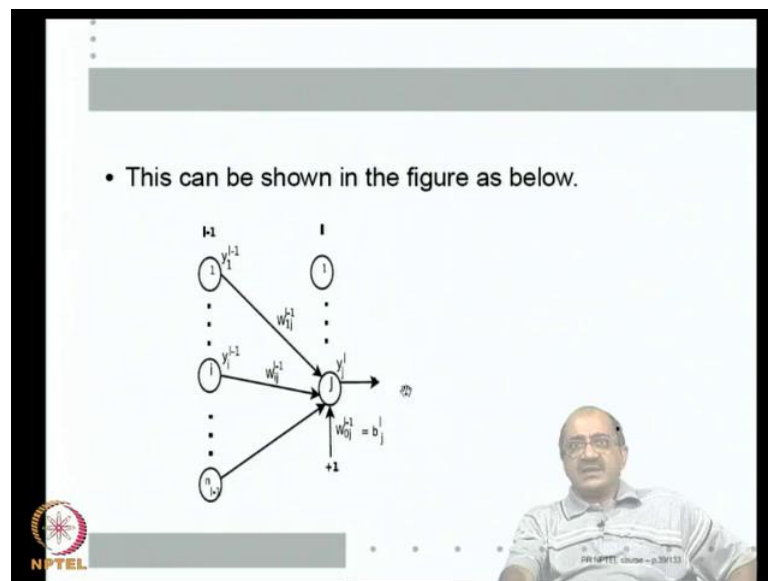
$$\eta_j^\ell = \sum_{i=0}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1}$$

where, by notation, $w_{0j}^{\ell-1} = b_j^\ell$ and $y_0^\ell = +1$ for all ℓ .

Algebraically bias input can always be incorporated like this. So, just like when we considered augmented feature vector as in considering linear classifiers, we know that bias input can always be included like this. So, with this in mind, we assume that

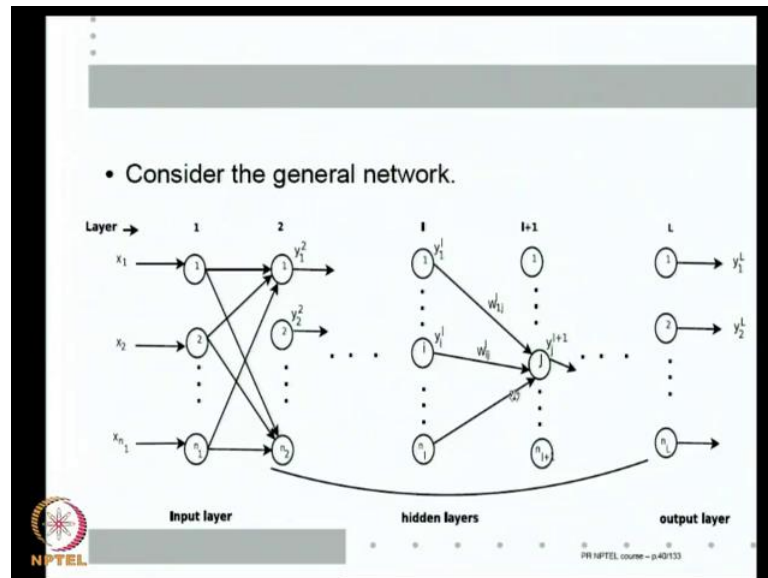
bias is included even though we sometimes put the summation from i is equal to 0 sometimes put it from i is equal to 1, but even if you put i is equal to 1 the idea is that whenever we need we can add bias right bias to every unit all right. Of course, units in the input layer would not have any bias input because they are they are just connected to external inputs and their job is only to find out inputs. So, even it is the first layer will not a bias, but all the other layers can have bias inputs, but they can be simply thought of as extra inputs through this notation.

(Refer Slide Time: 22:18)



And notation ally we will always represent it like this, right an extra bias input.

(Refer Slide Time: 22:25)



So, let us come back to our general network. So, I hope now things are clear. So, this is input layer, this is the output layer, these are the hidden layers, this is for any typical node I compute it is net input by taking the outputs of nodes is connected to multiply by that weight, sum it up I get the net input pass it through an activation function I get it is output, the way network compute input is from the external world we give inputs x_1 to x_{n_1} .

Now, first all the layer 1 units compute the output, which are simply equal to these external inputs. Now, all the layer 2 units compute their output right, they compute their net input and output by using the outputs of layer 1, once all the layer 2 ones got their output then layer 3 starts computing it is output and, so on all the way up to layer L and the outputs of the layer L are the outputs that are give out to the external world.

(Refer Slide Time: 23:25)

• Given inputs, x_1, \dots, x_{n_1} , the outputs are computed as follows.

• For the input layer: $y_i^1 = x_i, i = 1, \dots, n_1$.

• For $\ell = 2, \dots, L$, we now compute

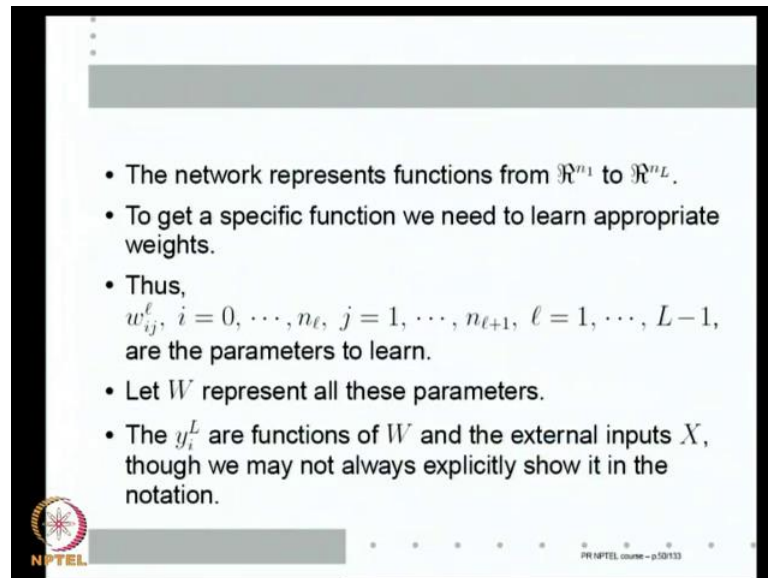
$$\eta_j^\ell = \sum_{i=1}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1}$$
$$y_j^\ell = f(\eta_j^\ell)$$

• The $y_1^L, \dots, y_{n_L}^L$ form the final outputs of the network.

NPTEL PRE NPTEL course - p45133

So, let us sum it up. So, given x_1 to x_{n_1} outputs are computed as follows, for the input layer y_i^1 is simply x_i and for layers 2 to L onwards I compute the net input η_j^ℓ for any node j in layer ℓ as $\eta_j^\ell = \sum_{i=1}^{n_{\ell-1}} w_{ij}^{\ell-1} y_i^{\ell-1}$ summed over i , i goes over all the nodes in layer $\ell - 1$ as I said I just put it 1 here, if I want bias input this simply becomes summation from 0, it is output of node j in layer ℓ is simply the net input passed through an activation function and because we are assuming the same activation function for all nodes I can put f there, this is how the output is computed. So, finally the outputs of the L 'th layer nodes capital L is the output layer. So, output of the capital L 'th layer nodes, that is the outputs of the output layer nodes, from the final output of the network.

(Refer Slide Time: 24:35)



- The network represents functions from \mathbb{R}^{n_1} to \mathbb{R}^{n_L} .
- To get a specific function we need to learn appropriate weights.
- Thus,
 $w_{ij}^{\ell}, i = 0, \dots, n_{\ell}, j = 1, \dots, n_{\ell+1}, \ell = 1, \dots, L-1,$
are the parameters to learn.
- Let W represent all these parameters.
- The y_i^L are functions of W and the external inputs X , though we may not always explicitly show it in the notation.

So, the network represents a function from \mathbb{R}^{n_1} to \mathbb{R}^{n_L} to get a specific function we need to learn appropriate weights. So, what are that we have to learn ultimately we have to learn w_{ij}^{ℓ} , if I assume that their biases, i goes from 0 to it is a w_{ij}^{ℓ} connects a node in layer $\ell - 1$ to a node in layer ℓ . So, i goes over nodes in layer $\ell - 1$ j goes in goes over nodes in layer $\ell, \ell + 1$. So, i goes from 1 to n_{ℓ} j goes from 1 to $n_{\ell+1}$ if I want to use a bias then I can also go to 0.

So, w_{ij}^{ℓ} where i goes from 0 to n_{ℓ} if there is a bias otherwise 1 to n_{ℓ} , j going from 1 to $n_{\ell+1}$ because j goes over the nodes in a layer $\ell + 1$ and the ℓ itself goes from 1 to n_{ℓ} this is the number of or these are the set of parameters we have, this the values if I put values for all these parameters, that determines a specific function, this is what I have to learn let put all of that together into some capital W , we can think of capital W as a big vector whose components are this of course, we will write the components with, so many subscripts. So, that we understand what the components are.

So, essentially the outputs of the output layer, which are the external, which are the outputs of a network are functions of all these parameters, functions of this capital W and also functions of the external input x right of course, we do not write it explicitly most of the time, in the beginning and a few times then we express in needed it I will write y_i^L as a function of w and x , but whether or not we write it we should always remember, there is always a function of the weights of the network and the external inputs.

(Refer Slide Time: 26:30)

• Consider a 2-layer network with single output node.

• This is like a Perceptron or an AdaLinE.

$$y = f \left(\sum_{j=1}^m w_{j1}^1 x_j \right)$$

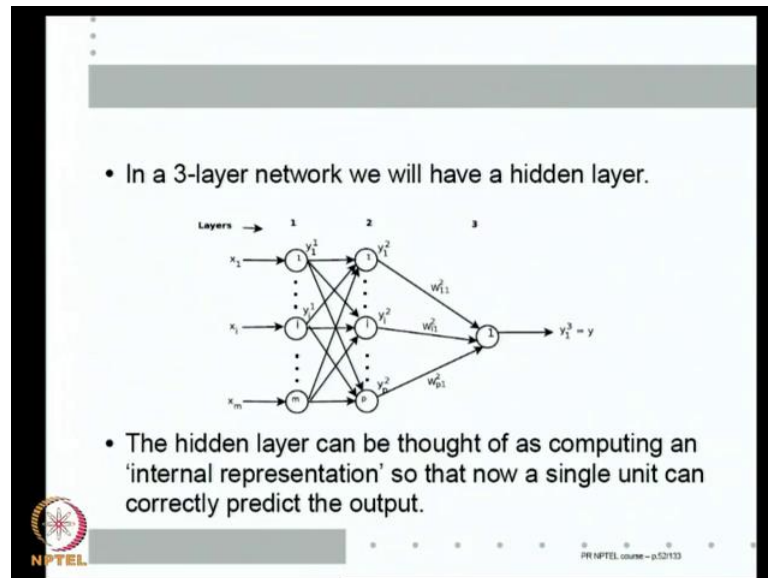
NPTEL PRE NPTEL course - p.5/133

So, let us just quickly look at this notation to understand something that we already know, suppose you take a 2 layered network. So, 2 layers then there is only input layer and output layer, unless we have only single output problem, right then what will be it is output. So, there is only one output. So, I call it y is f the activation function of this node of the net input, summation j going over this acts as a m nodes here, 1 to m w_{j1}^1 right from layer 1 to layer 2 connecting node j in layer 1 to node 1. So, these 2 are 1. So, whether or I put the 1 it does not matter it is like f of summation j $w_{j1}^1 x_j$.

So, what is it, it is like a perceptron or adaline, if I take f to be a step function, right a threshold function, then it is a perceptron, if I take f to be linear it is adaline, if I take f to be sigmoid it is like, what we use in logistic regression right. So, a 2 layered network by our notation is like a single neuron model and we already know single neuron model, many of what we thought many of things that we considered in linear classifiers are all single single neuron models.

So, this is how a. So, a single neuron model for us is essentially a 2 layered network because in a 2 layered network, there is no scope of any hidden layers. So, there are only inputs and outputs. So, output is simply f of linear sum of inputs, that is what all single neuron models are. So, they in our notation turn out to be 2 layer. So, in layer feed forward neural networks, 2 layer networks are like perceptron right.

(Refer Slide Time: 28:15)



So, I need at least a 3 layered network to have hidden layer, in a 3 layered network there is a hidden layer. Now, if I consider only layers 2 to 3 it is like a 2 layered network. So, this part of the network is like a perceptron or adaline. So, what does that mean, essentially if what will be x m 's for example, I am doing a pattern recognition problem these will be the feature values.

If I have given these outputs of these nodes as feature values, then this perceptron can represent, suppose this this particular network represent a classifier I want; that means, if these were the actual feature values, then a a single neuron model essentially a 2 layered network, could have represented it. So, what a 3 layered network is doing, it is taking the actual inputs and then transforming them, right on the hidden layer to some appropriate values. So, that now, a simple linear classifier can finish the job. So, we can think of the hidden layer, as computing what can be called an internal representation of the feature vector.

So, that now a single unit can complete the a classification or predict the right output right. So, we can think of the way these things are doing non linear classification, is we said a linear classifier is not good enough. So, what this network is doing is linear classification is not good enough only because of the particular way the feature vector is represented. So, by transforming this appropriately into some other representation, right this may be m dimensional representation this might be a p dimensional representation

right. So, by appropriately transforming into another representation. Now, a linear classifier would do because these 2 layers now, together are simply a 2 layered network. So, we will come back to this notion, but it is good to remember, that what the hidden layer is doing is computing an internal representation.

(Refer Slide Time: 30:25)

• Let us first consider using neural network models to learn a function.

• Suppose we have training data $\{(X^i, d^i), i = 1, \dots, N\}$, where $X^i = [x_1^i, \dots, x_m^i]^T \in \mathbb{R}^m$ and $d^i = [d_1^i, \dots, d_{m'}^i]^T \in \mathbb{R}^{m'}$.

• We want to learn a neural network to represent this function.

NPTEL

So, let us consider how do I train this networks to represent a function of interested. So, in all like a like all our pattern recognition we are first looking at the regression problem, we are not looking at the classification problem let us say, let us first looking at learning in a regression problem. So, given training data, all the while we are taking X_i y_i as the training data, X_i as the input vectors, y_i as the target values, but in this notation we are used y 's for representing the outputs of specific nodes.

So, the training data we will think of $x_n \times x_i$ and d_i where d is because we can think of it is a desired output, essentially this training example means when I give you input X_i the network should ideally say d_i because d_i is the desired output for that. So, let us say we think of X_i d_i i going from 1 to N as the training samples, we put the i in the superscript because now we have been, specifically need to look at the components of these vectors. So, then we can use the subscripts for the components.

So, that is what we are going to do. So, X_i is a vector with m components. So, it is an \mathbb{R}^m . So, this superscript denotes the index of the training training vectors. So, X_i is the i 'th training example, each training example input is a vector with m components and

let us say the desired output is also a vector of m' prime formula, meaning I am giving you pairs of vectors the the input part in m dimensions, output part in m' prime dimensions.

So, I want you to learn a function using these examples, which is from m to m' , if m to m' is 1 is the usual regression function, learning problem if m' prime is greater than 1 we simply mean that we are learning a vector valued function. So, given X_i and d_i are the training data X_i in \mathbb{R}^m and d_i in $\mathbb{R}^{m'}$, we want to learn a neural network that can represent this function. So, just like a least squares and, so on now we have to train the neural network meaning I have to learn the weights for this. So, how do I do that, I want to learn a function from \mathbb{R}^m to $\mathbb{R}^{m'}$. So, I need m input nodes and m' prime output nodes, right that much I already know.

(Refer Slide Time: 32:48)

- We can use a L layer network with $n_1 = m$ and $n_L = m'$.
- L and n_2, \dots, n_{L-1} , are parameters which we fix (for now) arbitrarily.
- We assume that nodes in all layers including output layer use the sigmoid activation function.
- Hence we take $d^i \in [0, 1]^{m'}, \forall i$.
- We can always linearly scale the output as needed. (Or we can also use a linear activation function for output nodes).

So, we use some L layered network with n_1 is equal to m and n_L is equal to m' . Then what do I have to fix I to fix what is the capital L how many layers I want then I have to fix the number of nodes in all the other layers, the first and last layer I know how many number of nodes because I have to learn a function from \mathbb{R}^m to $\mathbb{R}^{m'}$, but n_2, n_3 all the way up to n_{L-1} are the number of nodes in the hidden layers and I can fix it to be anything.

Let us say I somehow fix it, right now we will think of it arbitrary we will we will look at some guidelines later on, but really there is no correct ways of determining this, this is a


part of the parameters of your model, which you have to just choose. So, we will choose some number of layers and some hidden nodes for each layer of course, we assume that all nodes, in all layers including the output layer use the same sigmoid activation function right as I said that is our notation now.

Which essentially means, the outputs of the output nodes, can be only between 0 and 1 if I am using sigmoid, if I am using tan hyperbolic it can be only between minus 1 to plus 1. Now, of if I am giving an arbitrary function how can I represent it if my outputs are only can vary only between 0 and 1. So, we either assume that d_i are between 0 and 1 we can easily scale the outputs of the training samples or we can linearly scale the output node, right we we can multiply the sigmoid with any constant we want.

So, that instead of going between 0 to 1 it can go from 0 to k for any k or 0 to some m for any m of course, if you want the outputs to have both signs, we can use tan hyperbolic or we can simply use a simple linear activation function for the output nodes, we can do any of these to take care of the the output of the network being having the same dynamic range as d_i 's.

So, we will just without worrying about it we will assume that we will do the something like that but right now we will assume that we have an activation function f which in the in a specific case may be linear may be a a sigmoid with some extra constant and so on for the output layer alone, just because the output layers of outputs how to be able to represent all the numbers in going over the range that d_i takes. So, so now, with this small comment we would not bother about this anymore.

(Refer Slide Time: 35:18)



- Let $y^L = [y_1^L, \dots, y_{n_L}^L]^T$ denote the vector of outputs.
- We should actually write $y^L(W, X)$, $y_i^L(W, X)$ and so on.
- Now, the risk minimization framework is as follows.
- We fix a \mathcal{H} by fixing architecture of the network. (That is fixing, L and n_ℓ , $\ell = 1, \dots, L$).
- This hypothesis space is parameterized by W .
- We can now choose a loss function and learn 'best' W by minimizing empirical risk.
- We choose squared error loss function.

NPTEL course - p.70133

Let us say y^L is the vector that denotes the output of the network. So, its components are y_1^L up to $y_{n_L}^L$ because there are n_L nodes in the output layer. So, that is the vector, of course, we as I said we should actually write it as $y^L(W, X)$ or $y_i^L(W, X)$ and, so on. So, only when we need it we will write it, we will remember that the outputs are all functions of the weights and the input.

So, now, we can think of it in the risk minimization framework as follows, in the risk minimization framework we have a hypothesis space \mathcal{H} , then we choose a convenient loss function and we were searching \mathcal{H} to find a hypothesis, to find a function that minimises empirical risk with respect to the chosen loss function. So, we fix \mathcal{H} by fixing the architecture of the network. So, we fix the L we fix n_ℓ and. So, on that essentially fixes a class of function that can be represented now, by changing all the weights. So, having fixed \mathcal{H} , we know now this hypothesis space is parameterised by W , W contains all the weights in the network. So, what do I have to do now, which choose a loss function and learn the best W by minimizing empirical risk with respect to that loss function and for neural networks we will choose the squared error loss function, just like in the least square sense, here also we choose the squared error loss function. So, where does this take us.

(Refer Slide Time: 36:43)

• The empirical risk is given by

$$\hat{R}_N(W) = \frac{1}{N} \sum_{i=1}^N \|y^L(W, X^i) - d^i\|^2$$
$$= \frac{1}{N} \sum_{i=1}^N \left(\sum_{j=1}^{m'} (y_j^L(W, X^i) - d_j^i)^2 \right)$$

• We want to find a W that is a minimizer of $\hat{R}_N(W)$.

NPTEL PR NPTEL course - p.73133

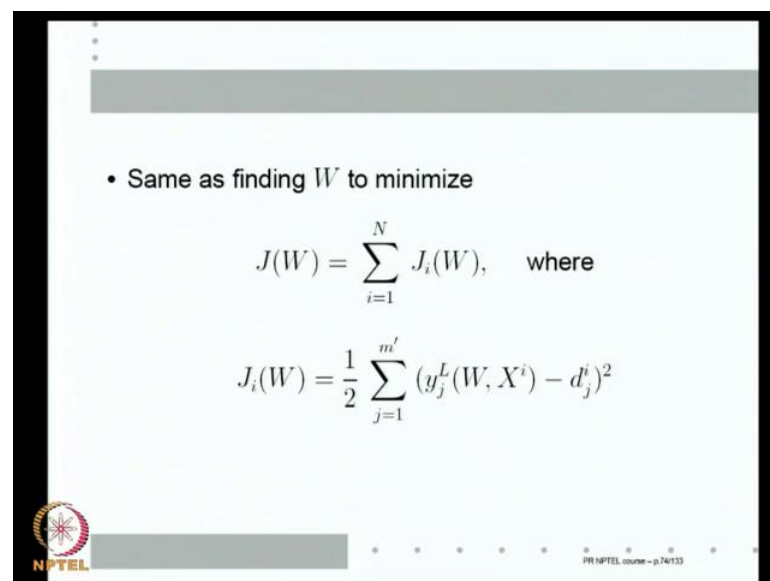
So, you know what is the empirical risk now, given any W and if I have n examples n training examples, then my empirical risk is $\hat{R}_N(W)$ as that is the notation we following earlier is $\frac{1}{N}$ summation over i , i is equal to 1 they are n examples W by 1 W comma X^i is the output of the network, right is the vector output of the network on input X^i the desired output is d^i .

So, $\|y^L(W, X^i) - d^i\|^2$ is the error i square it because these are vectors I have to take the norm and square it and that summed over all samples and $\frac{1}{N}$ gives me the empirical risk with respect to the squared error loss function, right what norm should I choose I can choose Euclidian norm. So, this norm square now, expands to sum over j over the components of the output they are m' outputs right. So, $\sum_{j=1}^{m'} (y_j^L(W, X^i) - d_j^i)^2$ right this d_j^i is the j 'th component of the desired output y_j^i is the j 'th output of the network y_j^L of course, is the function of the weights in the network and also X^i .

So, the what is it the inner parenthesis this big parenthesis, is the error between the output of the network sphere error, between the output of the network and the desired output when I use the training example the i 'th training example right. Now, sum it over all the training examples and divide by $\frac{1}{N}$ that gives me the empirical risk with respect to squared error loss. So, this is the function of W this is what I want to minimize.

So, ultimately given X i d i i I want to find a W to minimise this function, right minimizing this function we know that this 1 by N is of no consequence, that constant makes no difference to the minimization and instead of 1 by N I may want to use 1 by 2 because when I want to minimize ultimately I may have to take a derivative if I take a derivative this 2 will come out. So, not to carry that extra unnecessary extra factors I can cancel that 2 in when I differentiate by choosing a one by 2. So, instead of minimizing this I can as well minimize 1 by 2 of this the rest of the expression right. So, that is what we did even in linear least squares.

(Refer Slide Time: 39:25)



• Same as finding W to minimize

$$J(W) = \sum_{i=1}^N J_i(W), \quad \text{where}$$
$$J_i(W) = \frac{1}{2} \sum_{j=1}^{m'} (g_j^L(W, X^i) - d_j^i)^2$$

NPTEL logo and footer text: PR NPTEL course - p.74/133

So, because it is no longer actually equal to empirical risk it is only proportional to empirical risk, we use some other symbol.

(Refer Slide Time: 39:32)

• The empirical risk is given by

$$\hat{R}_N(W) = \frac{1}{N} \sum_{i=1}^N \|y^L(W, X^i) - d^i\|^2$$
$$= \frac{1}{N} \sum_{i=1}^N \left(\sum_{j=1}^{m'} (y_j^L(W, X^i) - d_j^i)^2 \right)$$

• We want to find a W that is a minimizer of $\hat{R}_N(W)$.

NPTEL PR NPTEL course - p.74/133

So, minimizing finding a W to minimise this, is same as finding a W to minimise this J . So, now, I am writing J W see this has 2 summations, sum over the example of the errors, then errors themselves are the Euclidian distance between the output of the network and the desired output. So, we will brought you the whole thing up the errors per example into one symbol.

(Refer Slide Time: 40:00)

• Same as finding W to minimize

$$J(W) = \sum_{i=1}^N J_i(W), \quad \text{where}$$
$$J_i(W) = \frac{1}{2} \sum_{j=1}^{m'} (y_j^L(W, X^i) - d_j^i)^2$$

NPTEL PR NPTEL course - p.74/133

So, I can write, what I want to find as you find a W to minimise J W which is summed over example J_i W , where J_i W I write as half of J is equal to 1 to m prime this ϕ

right, the only difference between this and this expression that I have taken this 1 by N and put a half there, only thing is this half I pushed into the first summation and called the term inside the first summation as J_i . So, essentially I am minimising this I may I am finding that W to minimise this, where each J_i is given by this.

So, what is J_i , J_i is the square of the error, actually half of the square of the error the input that half, but any half of the square of the error, between the output of the network and the desired output for the training example. So, I have just when I want distance I have taken Euclidian distance, that is why this sum comes. So, this square of the error between the output of the network and the desired output and you sum it over all examples and that is what you want to minimise.

(Refer Slide Time: 41:03)

- One method of finding a minimizer of J is to use gradient-descent.
- This gives us the following learning algorithm

$$W(t+1) = W(t) - \lambda \nabla J(W(t))$$

$$= W(t) - \lambda \sum_{i=1}^N \nabla J_i(W(t))$$

where t is the iteration count and λ is the step-size parameter.

How do I minimise we already know we have used it in the in the linear case, linear case also we had a similar thing, only thing is this is much simpler expression and what did we do use for example, in for the LMS algorithm we used a gradient descent right we can still do the gradient descent, say unlike in the least squares we we cannot, we cannot use linear algebra here because these are all some arbitrary non linear functions we cannot analytically solve them, but we can certainly use gradient descent.

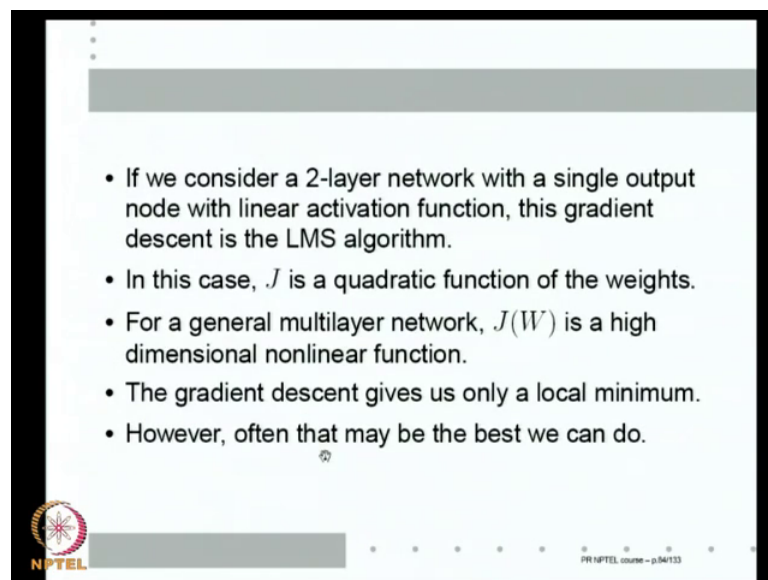
So, one way of minimising J is to use gradient descent. So, if I want to use gradient descent I know what my algorithm is it, is an iterative algorithm. So, my parameters at iteration t plus 1 are given by parameters at iteration t minus lambda times of gradient of

J at W^t right, given the current values W^t I find the gradient of J at the current value of my parameters and moving move a small distance, in a negative gradient direction right because I want to minimise.

So, this is my algorithm, components of W or W_{ij} components of gradient or $\frac{\partial J}{\partial W_{ij}}$. So, in the component equation this will be W_{ij}^{t+1} is W_{ij}^t minus λ times $\frac{\partial J}{\partial W_{ij}}$. So, this is my gradient algorithm. Now, we know that J is sum of J_i . So, gradient of J is sum of gradients J_i . So, I can write gradient of J as summation i is equal to 1 to n gradient J_i within bracket W^t simply means that of course, I have to use, because iterative algorithm I have to evaluate the gradients as the current value of parameters right.

T is the iteration count and gradient descent has a step size parameter λ I am using a constant step size I would not be doing any line such and as you know, if I use a constant step size gradient descent I have to keep this step size very small for you to converge, but this is my algorithm norm that is all.

(Refer Slide Time: 43:25)

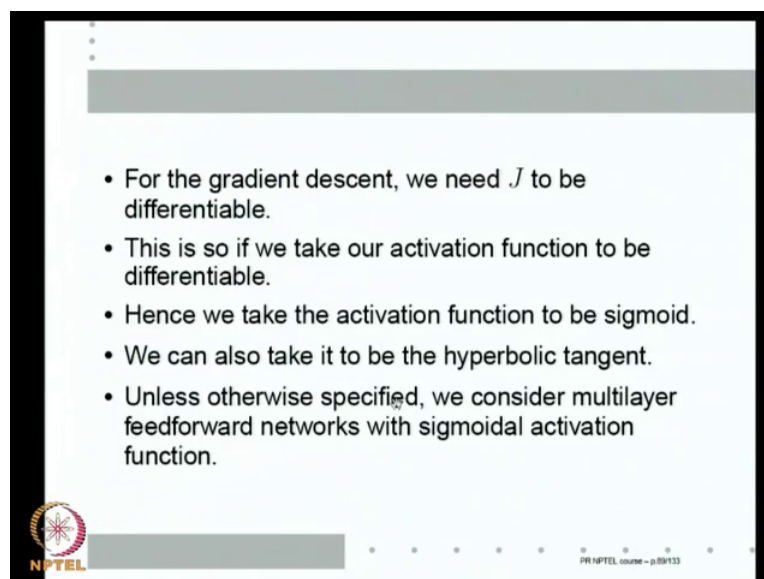


If I have to actually consider 2 layered network with a single output node with a linear activation function, this is exactly LMS algorithm, this is what we did in LMS algorithm right where the J is simply $W^T x - y$ whole square right. So, if I had used a 2 layer network, with a single output node, then this is exactly the gradient descent that is the LMS algorithm, in that case there is quadratic function of the weights and that is

why we can always learn good weights, but now J is a W is firstly, high dimensional there are, so many layers between every pair of layers there are, so many weights.

So, the number of weights would could be very large, we can make a rough estimate suppose I have got a 50 dimensional say 100 dimensional feature vector. So, there will be 100 nodes in the input layer, let us say I have put 100 nodes in hidden layer and 1 node in the output layer, how many weights are there, each of the 100 nodes in the input layer, how do we connect it to each of the 100 nodes in the output hidden layer. So, that is 10000 weights.

So, number of weights will grow pretty fast. So, in general J is a high dimensional non linear function, high dimensional meaning it is input is a high dimensional vector of course, J is a real valued function. So, minimising J is actually gradient descent in a very high dimensional space and J is also non linear function. So, gradient descent can only give us a local minimum right of course, our J is roughly same as our empirical risk, but we want to find the global minimize of the empirical risk to do well, well this is a non linear problem just because we want global minimum we may not be able to get global minimum. So, very often the best we can do is to get a local minimum. So, we will settle for it we are using gradient descent, we know we get a local minimum, but you know that is possibly very often the best we can do, so thus what will do to along this networks.



- For the gradient descent, we need J to be differentiable.
- This is so if we take our activation function to be differentiable.
- Hence we take the activation function to be sigmoid.
- We can also take it to be the hyperbolic tangent.
- Unless otherwise specified, we consider multilayer feedforward networks with sigmoidal activation function.

NPTEL

PR NPTEL course - p.09/13

Also for the gradient descent we need J to be differentiable right, essentially what is there inside J is J a while inside J a is π minus d i whole square. So, essentially if y is differentiable that is good enough, differentiable with respect to weights y is the output of the nodes, it will be differentiable with respect to weights, if the activation function is differentiable, right y is f of summation. So, summations are anyway differentiable.

So, just the question is whether or not the activation function is differentiable for example, if I use threshold activation function like in perceptron, it would not be differentiable, but if I use sigmoid it is differentiable. So, we will simply take the activation function to (()) a sigmoid of course, we would have taken tan hyperbolic and many other differential functions are there, but we we we simply for now, assume that it is always a sigmoid of course, it really does not matter we will always use f as the symbol for activation function. So, but if we want any specificity we simply think that we are using sigmoid activation function, any case for using gradient descent we want our activation function to be differentiable.

(Refer Slide Time: 46:28)

- To completely specify our algorithm for learning the weights, we need an expression for the gradient of J_i .
- In terms of the individual weights, the gradient descent is

$$w_{ij}^\ell(t+1) = w_{ij}^\ell(t) - \lambda \sum_{s=1}^N \frac{\partial J_s}{\partial w_{ij}^\ell}(W(t))$$

- As in the case of LMS, we can have batch or incremental version of the algorithm.

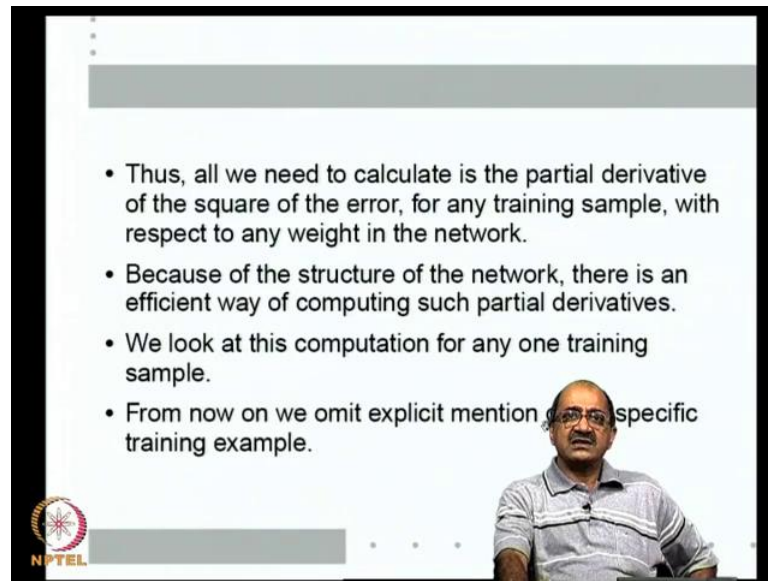
So, now if the algorithm completely specified, almost completely specified. If we can tell how to derive the expressions for the functions of gradient of J of course, it is a complicated network. So, unless we have some nice expressions at least, we may not even be able to numerically complete gradient or may not be able to complete gradient efficiently. So, what is that we want actually in terms of the individual weights as we

said $w_{ij}^{l+1} = w_{ij}^l + \text{gradient} \cdot \Delta J$ where ΔJ is summation J_s . So, it is summation over s , as a equal to 1 number of examples J_s by w_{ij} by 1.

So, essentially for any one input any arbitrary input I put at the input of the network I calculate the output and hence calculate the error between output of the network and the desired output, that error is J_s I need to know how to find the derivative partial derivative of error for anyone training example, with respect to any weight in the network that is all I need right of course, given this expression, we can immediately see that like in the LMS case, we can either have a batch or an incremental version of the algorithm right.

So, if I give examples one by one at the input calculate the outputs find the errors, find the derivative for each of them at the same W and after finishing all the presenting of all the n samples once, then I update the weights, that is the batch mode right. Otherwise I can present one input immediately calculate the output for that error I can calculate the gradient and then immediately update the weights and then go to the next thing, then that is an incremental version as we have seen in case of perceptron and in case of LMS we seen the batch versus incremental version all of them come only because your iterative algorithm has a summation over examples of the gradient right. So, it is a question of whether at the same value of W you calculate I present all the training examples one by one and only then update W 's or update it immediately after each training example. So, the same issues of batch versus incremental come here also. So, let us not go into them.

(Refer Slide Time: 48:52)



• Thus, all we need to calculate is the partial derivative of the square of the error, for any training sample, with respect to any weight in the network.

• Because of the structure of the network, there is an efficient way of computing such partial derivatives.

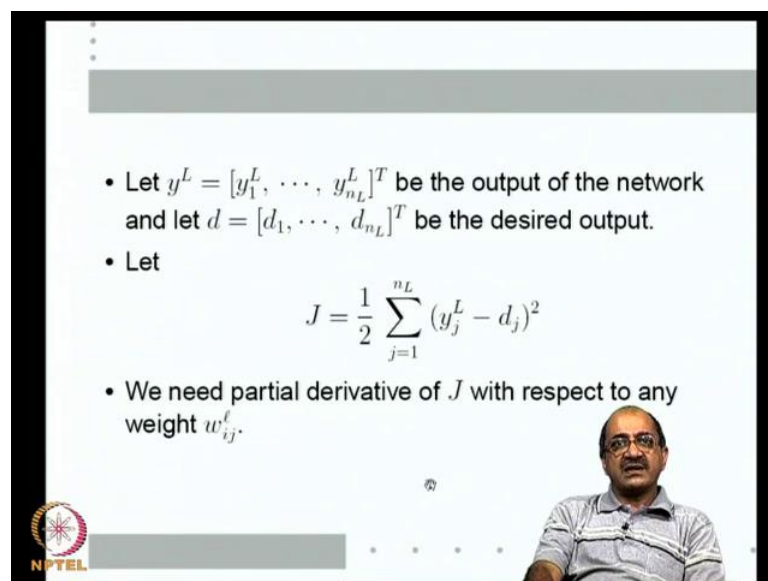
• We look at this computation for any one training sample.

• From now on we omit explicit mention of a specific training example.

NPTEL

So, what is that we have to do now we have to calculate the partial derivative of the square of the error, for any one training sample, with respect to any weight in the networks because if I can calculate for one I can calculate for each of the others right. So, that is the computational one. As it turns out because the structure of the network there is a very efficient way of doing such computation, that is the reason why we are going into, so much details here. So, we will do it for any one training example. So, let us not make any explicit mention of the specific training example, now.

(Refer Slide Time: 49:26)



• Let $y^L = [y_1^L, \dots, y_{n_L}^L]^T$ be the output of the network and let $d = [d_1, \dots, d_{n_L}]^T$ be the desired output.

• Let

$$J = \frac{1}{2} \sum_{j=1}^{n_L} (y_j^L - d_j)^2$$

• We need partial derivative of J with respect to any weight w_{ij}^L .

NPTEL

So, let us say the output now is y_1^L to y_n^L and we said desired output is d_1 to d_n . We are not putting any superscript on d because I can ultimately put what is superscript I want, but basic computation involved is present one input at the input at the network, calculate the output find the error with respect to the desired output and then find all the derivatives, right if I can do it for one training example I can do it for all of them.

So, take any one of them. So, now, we will define a J without any subscripts actually this is like a J where I am only finding error with respect to one example, but this is this is what we need to find given this $J = y_j^L - d_j$ whole square, right we need to find the derivatives of J with respect to each of the weights, we need partial derivatives of J with respect to each one of the w_{ij}^L 's.

(Refer Slide Time: 50:18)

- Any weight w_{ij}^l can affect J only by affecting the final output of the network.
- In a layered network, the weight w_{ij}^l can affect the final output only through its effect on η_j^{l+1} .

Now, let us ask any weight w_{ij}^l how can it affect J , it can J is only dependent on the final output of the network and any weight inside can only affect the final output. So, how does it affect the final output, it can affect the final output only through what it contributes to the net input in the next layer, this is the w_{ij}^l this weight, this weight only contributes to the net output of only the j 'th node in the $l+1$ layer. The only way this weight can effect what happens at the output is by its effect on the net input to this one particular node and nothing else right.

So, w_{ij}^l can affect J only through its effect on n_j^{l+1} because w_{ij}^l connects from layer l to layer $l+1$. So, w_{ij}^l can affect the output only by its effect on n_j^{l+1} .

(Refer Slide Time: 51:18)

• Hence, using the chain rule of differentiation, we have

$$\frac{\partial J}{\partial w_{ij}^l} = \frac{\partial J}{\partial \eta_j^{l+1}} \frac{\partial \eta_j^{l+1}}{\partial w_{ij}^l}$$

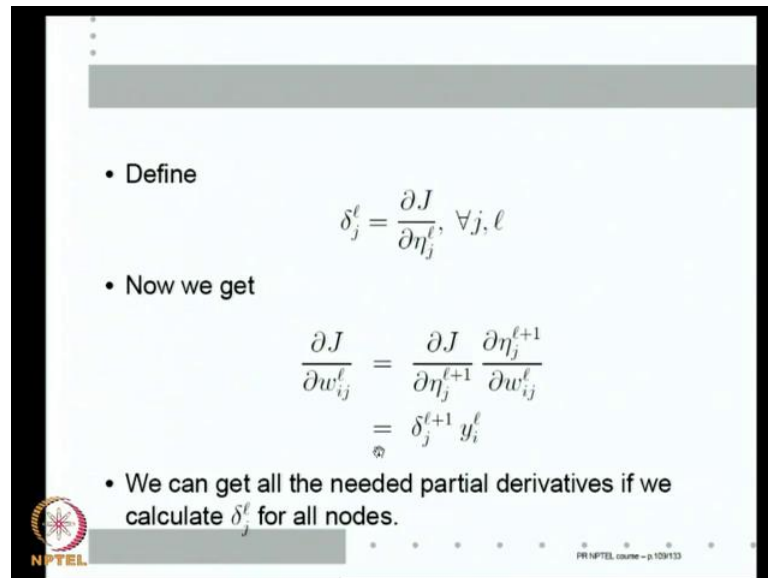
• Recall that

$$\eta_j^{l+1} = \sum_{s=1}^{n_l} w_{sj}^l y_s^l \Rightarrow \frac{\partial \eta_j^{l+1}}{\partial w_{ij}^l} = y_i^l$$

Which means using chain rule of differentiation I can write $\frac{\partial J}{\partial w_{ij}^l}$ by $\frac{\partial J}{\partial n_j^{l+1}}$ into $\frac{\partial n_j^{l+1}}{\partial w_{ij}^l}$ because I know that the only way w_{ij}^l affects capital J is through this net input to node j layer $l+1$. Now, this second part is easy to evaluate, why we know n_j^{l+1} net input to node j layer $l+1$ comes from layer l 's outputs, so $w_{sj}^l y_s^l$ as going from 1 to n_l .

Now, if I want derivative of this with respect to w_{ij}^l all the weights in this of j at the second index. So, only one time in the summation, namely when s becomes equal to i will contribute to the derivative and then the derivative is simply y_i^l . So, given this expression for n_j^{l+1} I know that $\frac{\partial n_j^{l+1}}{\partial w_{ij}^l}$ is y_i^l . So, this second time is y_i^l I do not know what the first term is let us first put a symbol for the first term.

(Refer Slide Time: 52:40)



• Define

$$\delta_j^\ell = \frac{\partial J}{\partial \eta_j^\ell}, \forall j, \ell$$

• Now we get

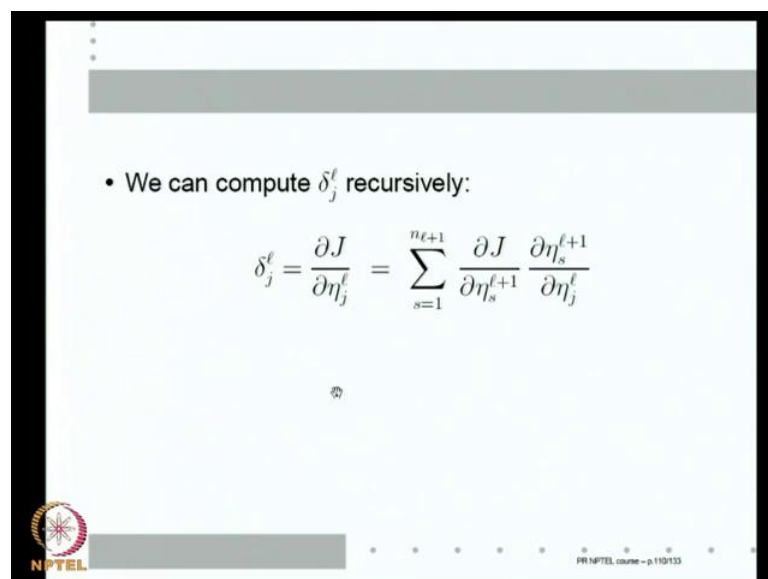
$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^\ell} &= \frac{\partial J}{\partial \eta_j^{\ell+1}} \frac{\partial \eta_j^{\ell+1}}{\partial w_{ij}^\ell} \\ &= \delta_j^{\ell+1} y_i^\ell \end{aligned}$$

• We can get all the needed partial derivatives if we calculate δ_j^ℓ for all nodes.

NPTEL PR NPTEL course - p.1101133

So, let us say delta l j is dou J by dou n j l of course, what we wanted is in that expression is dou J by dou n j l plus 1. So, that become delta n j l plus 1, but we are defining delta j l as dou J by dou n j l right. So, earlier we had dou J by dou w i j l is dou J by dou n j l plus 1 dou n j l plus 1 by dou w i j l. So, I can write this as delta j l plus 1 into y i l. So, y i l is very well known. So, known everything boils down to calculating delta j l plus 1 for various l's. So, if I can calculate l's I can calculate all the del these deltas I can calculate all the partial derivatives.

(Refer Slide Time: 53:27)



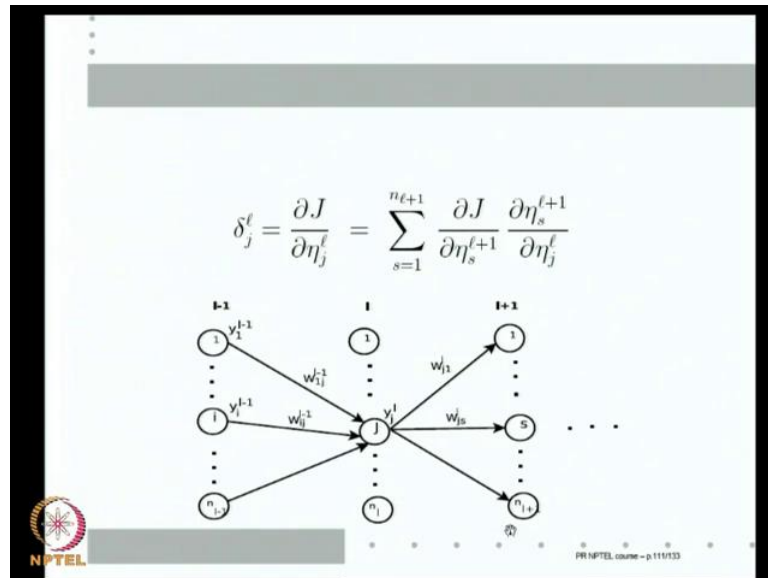
• We can compute δ_j^ℓ recursively:

$$\delta_j^\ell = \frac{\partial J}{\partial \eta_j^\ell} = \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial \eta_j^\ell}$$

NPTEL PR NPTEL course - p.1101133

And this is what we had, now this we can calculate recursively delta j l by definition is dou J by dou n j l and I can always write it using chain rule as dou J by dou n s l plus 1 dou n s l plus 1 by dou n j l.

(Refer Slide Time: 53:45)




Why I want how the final output gets affected by n_j . So, n_j is what is coming into this node, what is coming into this node can affect the final output, only in terms of what goes out of this node and what goes out of this node can only contribute to the net inputs of all the nodes in layer $l+1$. So, the only way this n_j affects capital J is through all the n_s 's. So, by chain rule of differentiation partial differentiation is s equal to 1 to n_{l+1} dou J by dou n_s dou n_s by dou n_j because what is coming into this node affects the final output, through it is affect on the net input to each of the nodes in the layer $l+1$.

(Refer Slide Time: 54:37)

• We can compute δ_j^ℓ recursively:

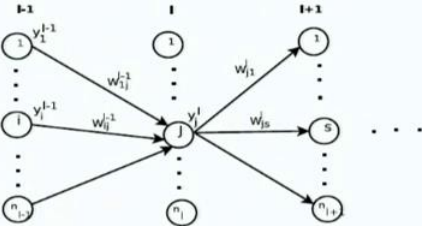

$$\delta_j^\ell = \frac{\partial J}{\partial \eta_j^\ell} = \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial \eta_j^\ell}$$

$$= \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial y_j^\ell} \frac{\partial y_j^\ell}{\partial \eta_j^\ell}$$


PR: NPTEL course - p.112133

So, I have this now I know how to do this $n_s + 1$ plus 1 I can write down $n_s + 1$ plus 1 by down $n_j + 1$.

(Refer Slide Time: 54:50)


$$\delta_j^\ell = \frac{\partial J}{\partial \eta_j^\ell} = \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial \eta_j^\ell}$$



PR: NPTEL course - p.112133

The only way the net input into this node, affects net input of any of these nodes is through the output right.

(Refer Slide Time: 55:01)

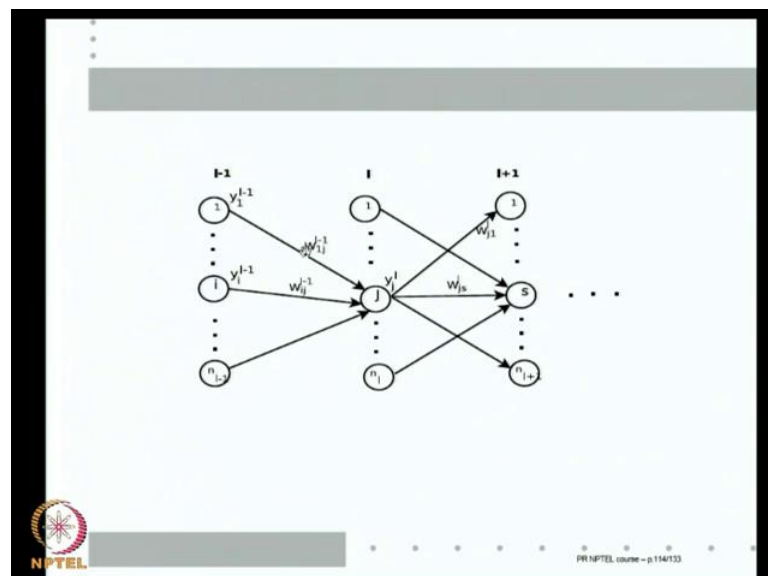
• We can compute δ_j^ℓ recursively:

$$\begin{aligned} \delta_j^\ell &= \frac{\partial J}{\partial \eta_j^\ell} = \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial \eta_j^\ell} \\ &= \sum_{s=1}^{n_{\ell+1}} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial y_j^\ell} \frac{\partial y_j^\ell}{\partial \eta_j^\ell} \\ &= \sum_{s=1}^{n_{\ell+1}} \delta_s^{\ell+1} w_{js}^\ell f'(\eta_j^\ell) \end{aligned}$$


PR NPTEL course - p114133

So, I can write n_{s+1} by n_j in terms of n_{s+1} by y_j right and y_j by n_j this I know what it is this simply y_j to n_j connection is in terms of the activation function. So, this is nothing but the derivative of the activation function. n_{s+1} by y_j connection you already know n_{s+1} is a sum over all the y_j 's n_{s+1} sum over all the y_j 's. So, I know how to calculate this partial derivative and this by definition is n_{s+1} .

(Refer Slide Time: 55:47)



So, this is $\delta_j^{\ell+1}$ plus 1 this as we this is easy to see because if I am asking what is $\delta_j^{\ell+1}$ plus 1 which is this y into this weight, this y into this weight right. So, if I want partial derivative with respect to the y 's, I get the corresponding weight.

(Refer Slide Time: 56:01)

• Thus, we can compute δ_j^ℓ recursively using:

$$\delta_j^\ell = \frac{\partial J}{\partial \eta_j^\ell} = \sum_{s=1}^{\ell+1} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial \eta_j^\ell}$$

$$= \sum_{s=1}^{\ell+1} \frac{\partial J}{\partial \eta_s^{\ell+1}} \frac{\partial \eta_s^{\ell+1}}{\partial y_j^\ell} \frac{\partial y_j^\ell}{\partial \eta_j^\ell}$$

$$= \sum_{s=1}^{\ell+1} \delta_s^{\ell+1} w_{js}^\ell f'(\eta_j^\ell)$$

So, that partial derivative is nothing but $w_{js}^{\ell+1}$ and this is f' prime. So, I can calculate $\delta_j^{\ell+1}$ if I can calculate $\delta_s^{\ell+1}$ for every s . So, this is the recursion right.

(Refer Slide Time: 56:19)

• Recall that the partial derivatives are given by

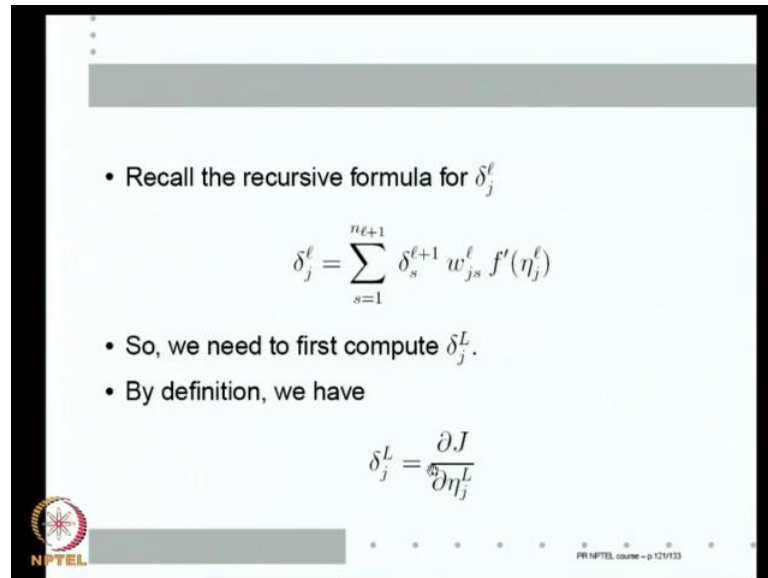
$$\frac{\partial J}{\partial w_{ij}^\ell} = \delta_j^{\ell+1} y_i^\ell$$

• For the weights, range of ℓ is $\ell = 1, \dots, (L - 1)$.

• Hence we need δ_j^ℓ for $\ell = 2, \dots, L$ and all nodes j .

So, all partial derivatives are given by this and for weights I need this for l is equal 1 to L minus 1. So, I need to calculate deltas for 2 to L if I can calculate delta j l for l is equal to 2 to L I am done right.

(Refer Slide Time: 56:38)



• Recall the recursive formula for δ_j^ℓ

$$\delta_j^\ell = \sum_{s=1}^{n_{\ell+1}} \delta_s^{\ell+1} w_{js}^\ell f'(\eta_j^\ell)$$

• So, we need to first compute δ_j^L .

• By definition, we have

$$\delta_j^L = \frac{\partial J}{\partial \eta_j^L}$$

NPTEL

PR: NPTEL course - p 12/133

I have this recursion formula. So, I have to first compute for capital L delta j l that is small l is equal to l minus 1 is what I have to compute first right. So, that is the only one for which I do not have a recursion formula. So, that is where the recursion has to end. So, let us try to calculate it delta j capital L that is dou capital J by dou eta j l this is easy, because J is given by this.

(Refer Slide Time: 57:01)

• We have

$$J = \frac{1}{2} \sum_{j=1}^{n_L} (y_j^L - d_j)^2$$

• Hence we have

$$\begin{aligned} \delta_j^L &= \frac{\partial J}{\partial \eta_j^L} = \frac{\partial J}{\partial y_j^L} \frac{\partial y_j^L}{\partial \eta_j^L} \\ &= (y_j^L - d_j) f'(\eta_j^L) \end{aligned}$$

NPTEL PR NPTEL course - p 124/133

So, differentiate this with respect to the weights. So, $\frac{\partial J}{\partial y_j^L}$ will be that half will cancel. So, $\frac{\partial J}{\partial y_j^L}$ by $\frac{\partial y_j^L}{\partial \eta_j^L}$ this is nothing but $y_j^L - d_j$ into f' prime right. So, this I can calculate for each node in the output layer.

(Refer Slide Time: 57:23)

• Using the above we can compute $\delta_j^L, j = 1, \dots, n_L$.

• Then we can compute $\delta_j^\ell, j = 1, \dots, n_\ell$ for $\ell = (L - 1), \dots, 2$, recursively, using

$$\delta_j^\ell = \left(\sum_{s=1}^{n_{\ell+1}} \delta_s^{\ell+1} w_{js}^\ell \right) f'(\eta_j^\ell)$$

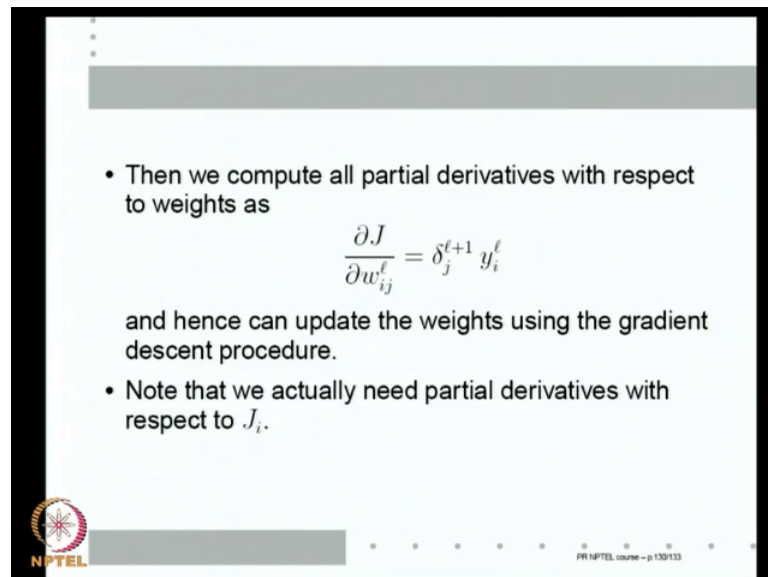
• We call δ_j^ℓ the 'error' at node- j layer- ℓ .

NPTEL PR NPTEL course - p 127/133

Once again calculate it for each node in the output layer, then we can calculate delta 1's for all the remaining nodes, recursively right, because we have just derived the recursion formula right then. So, what does that mean, we compute all the partial derivatives right because once I know the deltas for all the output layer nodes, then I can I know delta j 1.

So, if I put little l is equal to $l - 1$ here, then inside here I need δ s capital L for various s that I already know. So, I can calculate δ_j capital $L - 1$ for every j once, I know that I can calculate δ_j capital $L - 2$. So, that is why I am saying l is equal to $L - 1$ $l - 2$. So, on I can do all this calculation right.

(Refer Slide Time: 58:14)



- Then we compute all partial derivatives with respect to weights as
$$\frac{\partial J}{\partial w_{ij}^l} = \delta_j^{\ell+1} y_i^\ell$$
and hence can update the weights using the gradient descent procedure.
- Note that we actually need partial derivatives with respect to J_j .

NPTEL

PR NPTEL course - p 130/133

Then I can calculate all the partial derivatives and hence I can do the weight updates of course, I have which we shown it for one input, we have to do it for all inputs. So, this is essentially the learning algorithm for the network. So, we will start from here and look at why this is efficient, why this recursive comparative δ_j 's give us efficiency in the next class.

Thank you.