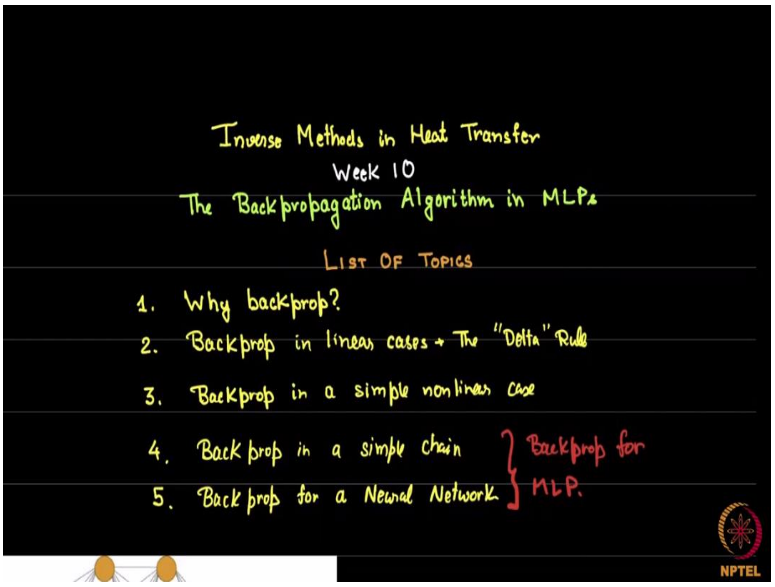
Inverse Methods in Heat Transfer
Prof: Balaji Srinivasan
Department of Mechanical Engineering
Indian Institute of Technology – Madras

Lecture - 54
Backprop in a Scalar Chain

(Refer Slide Time: 00:18)



Inverse Methods in Heat Transfer
Week 10
The Backpropagation Algorithm in MLPs

List of Topics

1. Why backprop?
2. Backprop in linear cases + The "Delta" Rule
3. Backprop in a simple nonlinear case
4. Back prop in a simple chain
5. Back prop for a Neural Network

} Backprop for MLP.

Welcome back to this Week 10 of Inverse Methods in Heat Transfer. In the previous video we did forward propagation within neural networks and specifically multi-layer perceptron. In this video we will look at the back propagation algorithm which is a very important algorithm historically in order to make neural networks possible on modern architectures as well as general practical problems.

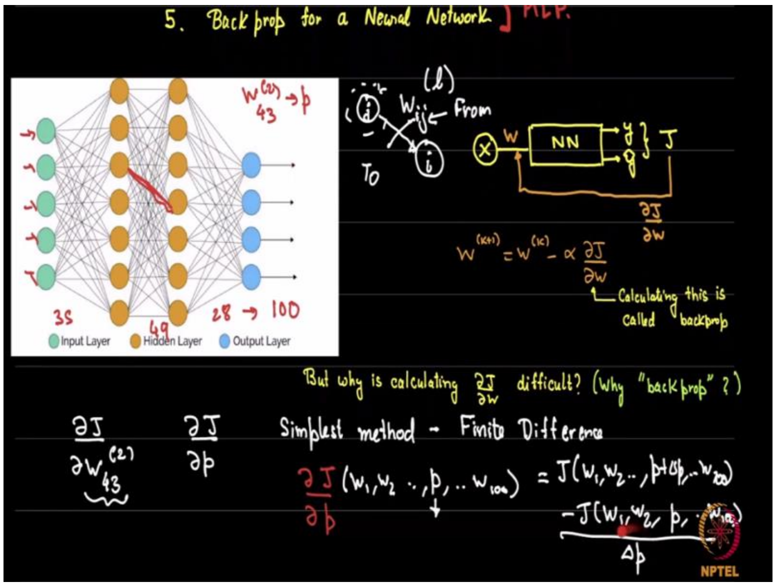
So here is the list of topics that I wish to cover in this video you will see there is a large list of topics. So, just in case I run a little bit more you know over time compared to what I wish to do, split this into two videos, but as of now the hope as I am starting to record it is that I have try to cover all these 5 topics within this video. So, the first question is why do we need a special algorithm for neural networks for back propagation.

What is it for I had already told you briefly what it was for in the previous video, but I will get into some more details in this video. Then I will split the back propagation algorithm into a few simpler cases just so that we can understand this final most important case of back propagation for a neural network I have found this effective in making sure that people get at least most of the important ideas behind what actually happens upon a back prop algorithm.

So, we will first do back prop in linear cases effectively we have already done this when we did gradient descent for linear regression, but I will explicitly point out something called the delta rule, when we do this, then we quickly extend this to a simple nonlinear case then we will look at a case where you have multiple layers, but each layer has only one neuron. These two cases can be thought of as just an input and output, but with multiple neurons.

So, these two are complementary cases and finally we will put this together in a neural network. These final two topics are effectively back propagation for multi-layer perceptron.

(Refer Slide Time: 02:24)



5. Back prop for a Neural Network] MLP.

$W^{(2)}_{43} \to p$

35 Input Layer, 49 Hidden Layer, 28 → 100 Output Layer

$W^{(l)}_{ij}$ ← From, To

$W^{(k+1)} = W^{(k)} - \alpha \frac{\partial J}{\partial W}$

Calculating this is called backprop

But why is calculating $\frac{\partial J}{\partial w}$ difficult? (Why "back prop"?)

$\frac{\partial J}{\partial W^{(2)}_{43}}$ $\frac{\partial J}{\partial p}$

Simplest method - Finite Difference

$$\frac{\partial J}{\partial p}(w_1, w_2 \ldots, p, \ldots w_{100}) = \frac{J(w_1, w_2 \ldots, p+\Delta p, \ldots w_{100}) - J(w_1, w_2, p, \ldots w_{100})}{\Delta p}$$

So, here is the structure of a neural network you must have seen it multiple times by now over the last few weeks. I have an input layer, a couple of hidden layers and an output layer and in the last video we saw what happens when we give some input here. So, suppose I give all these accesses we do some computations in these middle layers and we finally find out the outputs.

Now the question is when we do that exactly how do we train and find out these weights w. So, the algorithm as you have seen a few times is give some input, you give a guess for w this neural network you treat this as if it is a black box goes forward makes a prediction $\hat{y}$ ground truth y, gap is J which is just a function of y and $\hat{y}$ and then you take feedback from J and this feedback process is this; this is the gradient descent step.

The k + 1 titrations are the kth iteration minus alpha times this gradient. So, what we require in the feedback step explicitly is $\frac{\partial J}{\partial w}$. So, if I look at this figure here it has a lot of w so this is $5 * 7$ let us say if I ignore biases there are 35 weights here 49 here and about 28 here so around 100 weights exist within this neural network simple neural network. So let us take a simple weight.

So, for example, if I take this one in my old notation this was w_{34} and the second layer actually I am going to switch notation and I am going to use w_{43} second layer. So, the notation we are going to use now is if this is i or let me call this j and the next one is i then this is wij and whatever layer it is in. So, this is from and the to. So, we actually put where it comes from second and where it comes to first.

So, notation is a little bit flipped anyway. So, suppose we want $\frac{\partial J}{\partial {w_{43}}^2}$. So, let us call this something let us call this ${w_{43}}^2$ has p. So, suppose we want $\frac{\partial J}{\partial p}$, what is the difficulty in solving it. Now, why is it difficult? So, why exactly do we need this back prop algorithm. So, the catch here is this the older method or the simplest method of calculating it is finite difference.

You might not be familiar with this name finite difference, but it is a simple ideal. So, the idea is this there are about 100 weights here. So, let us say there are 100 obviously there are not 100, there are more than 100 weights here, but let us say there are 100 weights here so I think there are 102 or 112 something of that sort, but let us say there are 100 weights you fix 99 of them.

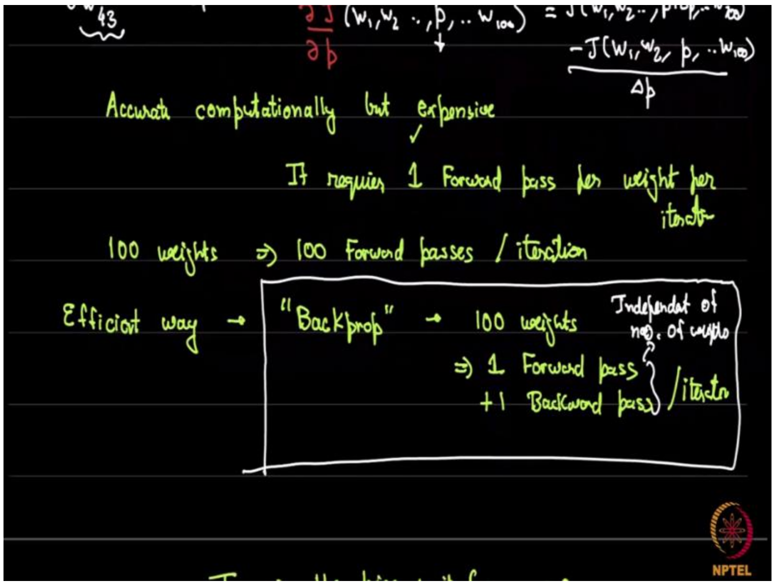
So, suppose you want $\frac{\partial J}{\partial p}$ and there were many other ways and J was a function of,

$$\frac{\partial J}{\partial p}\,(w_1, w_2, ..p, ..w_{100}) \;=\; \frac{J(w_1, w_2, ..., p + \Delta p, ..w_{100}) - J(w_1, w_2, ..p, ..w_{100})}{\Delta p}$$

So, the way you do it is you perturb just this variable so you do $J(w_1, w_2, ..., p + \Delta p, ..w_{100}) - J(w_1, w_2, ..p, ..w_{100})$ divided by Δp. Another way to say it is just perturbing this p by a little bit maybe if p is 1 make it 1.00001 and see what the output is.

Then this new output will be different from the old output by a little bit divide by Δp and as Δp goes to 0 you can estimate what $\frac{\partial J}{\partial p}$ is. So, there is a problem though.

(Refer Slide Time: 06:50)



$$-J(w_1, w_2, p, \ldots w_{100})$$

$$\Delta p$$

Accurate computationally but expensive ✓

It requires 1 Forward pass per weight per iteration

100 weights ⇒ 100 Forward passes / iteration

Efficient way → "Backprop" → 100 weights ⇒ 1 Forward pass + 1 Backward pass / iteration

Independent of no. of weights

This can be accurate computationally in fact the algorithm we are going to discuss is often check by checking it against finite difference, but it is expensive. Now why is it expensive? It is expensive because it requires one forward pass per weight per iteration. So, remember every iteration here requires you to calculate all $\frac{\partial J}{\partial w}$ that is what is meant by per iteration.

Now every weight will require you to perturb it once. So, for example, when I wanted p, I had to perturb this p up by a little bit. suppose I want $\frac{\partial J}{\partial w_1}$ I will have to perturb just w_1 and keep the rest a constant. So, basically you require if there are 100 weights means 100 forward passes per iteration. So, typical neural networks obviously have a whole lot more they can have thousands of millions and nowadays billions and some of the recent models even have a trillion.

So, just to do one iteration of gradient descent you will have to do trillion passes through the network and each trillion requires you to calculate the output of every neuron. obviously, this is extremely, extremely expensive and this was the historic reason why for a lot of time neural networks were never very big up until people found out a clever way of calculating these gradients.

And in fact, we calculate in some sense exact gradients on a computer by using chain Rule. So, what people found was there is an efficient way of calculating without doing finite difference this efficient way is what is known as back prop and back prop has this magical property that even for 100 weights you will still require one forward pass plus one backward pass per iteration.

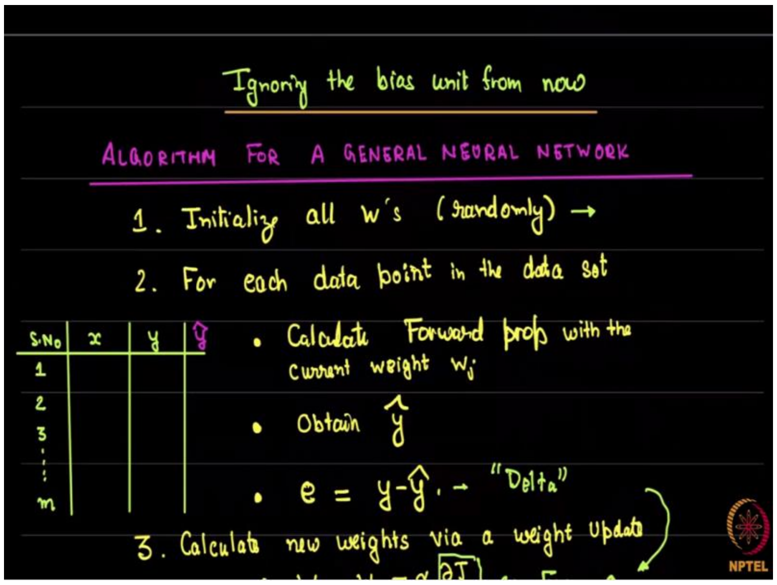
So, if you were doing something like the network, I have shown you, using back prop would be 50 times more efficient it is not just a small amount, it is 50 times that is like 5,000 percent more efficient if you were taking one day with back prop to do a computation, they would have taken 50 days a couple of months nearly and if you take a week here you would take a year by using finite difference.

So, that is the computational difference between finite differences just for 100 weights, typical networks as I will show you at least one in the next week will have a few thousand weights even for very simple cases, in that case you are going to get an efficiency gain of 500 times just by using back prop and if I come to million and billion and trillion we cannot even discuss you know there are things that would have taken an age of the universe for a single forward pass.

So, here is the catch. The catch for back prop is if you are using a gradient based method the back prop just requires one forward pass and one backward pass regardless of number of weights. So, it is independent of number of weights how many forward and backward passes you require and each backward pass is effectively only roughly as expensive as a forward pass so that is the reason for back prop.

Let us come to how this is achieved by the end of the sequence that I showed you should be able to see at least why back prop requires only one calculation.

(Refer Slide Time: 11:18)



Ignoring the bias unit from now

ALGORITHM FOR A GENERAL NEURAL NETWORK

1. Initialize all w's (randomly) →

2. For each data point in the data set

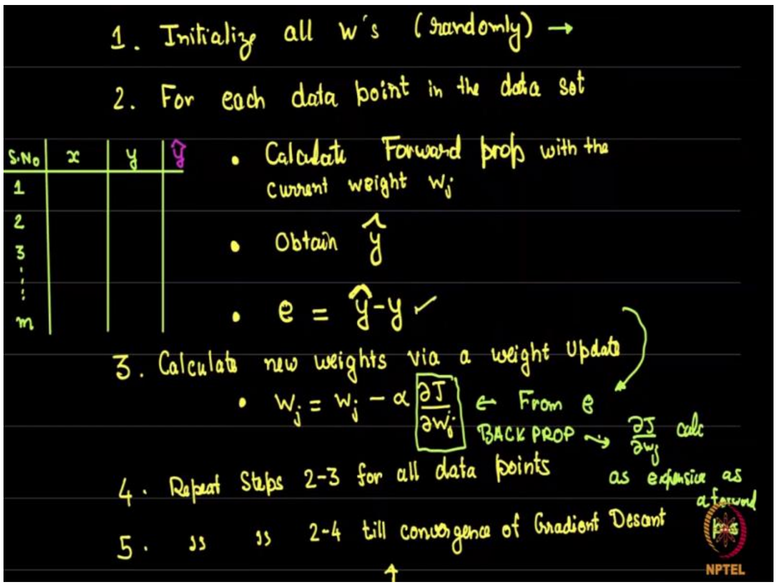
S.No	x	y	$\hat{y}$
1			
2			
3			
⋮			
m			

- Calculate Forward prop with the current weight w_j
- Obtain $\hat{y}$
- $e = y - \hat{y}$. → "Delta"

3. Calculate new weights via a weight Update

So, for what I am going to do now, I am going to ignore the bias unit from now on, all the expressions that I am writing now are written without the bias unit you should be able to do it for the case with bias unit by yourself, but that is not really expected of you at least as far as the exam is concerned, but for your verification and knowledge you can try and do that.

(Refer Slide Time: 11:45)



1. Initialize all w's (randomly) →

2. For each data point in the data set

S.No	x	y	$\hat{y}$
1			
2			
3			
⋮			
m			

- Calculate Forward prop with the current weight w_j
- Obtain $\hat{y}$
- $e = \hat{y} - y$ ✓

3. Calculate new weights via a weight Update
   - $w_j = w_j - \alpha \boxed{\frac{\partial J}{\partial w_j}}$ ← From e

     BACK PROP ⇝ $\frac{\partial J}{\partial w_j}$ calc as expensive as a forward pass

4. Repeat Steps 2-3 for all data points

5. " " 2-4 till convergence of Gradient Descent

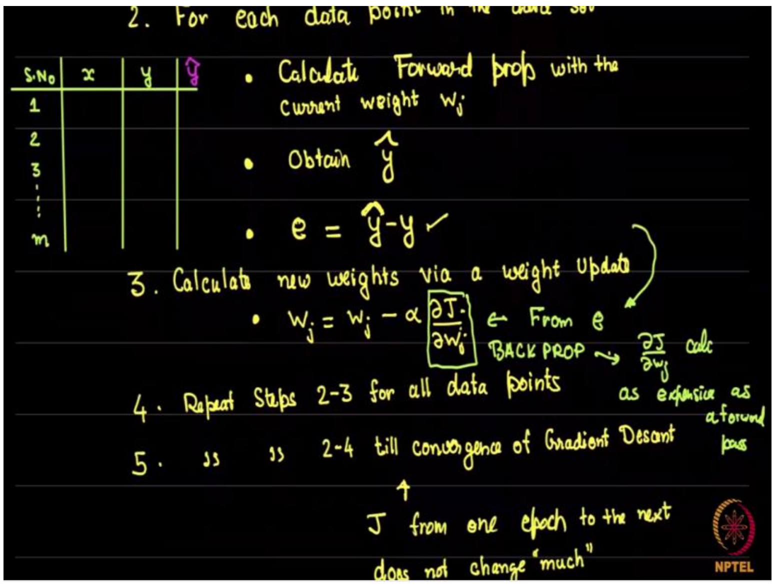
So, let us look at the general neural network algorithm. The general neural network algorithm for a network like this is very straightforward. You first initialize all these weights randomly and there are some specific initialization patterns, but we initialize them all randomly. Now for each data point in the data set. So, now remember you could have you know in our inverse cases typically we have 6, I do a case next week which will have 1000 data points for a fin.

So, let us say you have thousand points here and for all those x's and y's are collected already. So, these are the ground rules the inputs and the outputs and you do forward prop like we did in the last video. So, you do one forward pass with this you get a $\hat{y}$. Once you get a $\hat{y}$ you define an error I have called this delta, but let me just skip this for now. I will use slightly different notation.

So let us say I calculate the error which is $(y - \hat{y})$ in fact we will stick to a slightly different notation from the one that I have written here. We will call $(\hat{y} - y)$ if you see some differences between what I am writing here and the later videos or notes that I assume that e is either $(\hat{y} - y)$ or $(y - \hat{y})$ just ensure that I have used sign consistently. I will give you final expressions which will be consistent at the end of this video anyway.

So, now once you have calculated this error this is the gap between your ground truth and prediction.

(Refer Slide Time: 13:26)



2. For each data point

S.No	x	y	$\hat{y}$
1			
2			
3			
⋮			
m			

- Calculate Forward prop with the current weight w_j
- Obtain $\hat{y}$
- $e = \hat{y} - y$ ✓

3. Calculate new weights via a weight Update
   - $w_j = w_j - \alpha \boxed{\frac{\partial J}{\partial w_j}} \leftarrow$ From e

   BACK PROP $\rightsquigarrow$ $\frac{\partial J}{\partial w_j}$ calc as expensive as a forward pass

4. Repeat Steps 2-3 for all data points
5. " " 2-4 till convergence of Gradient Descent

$\uparrow$

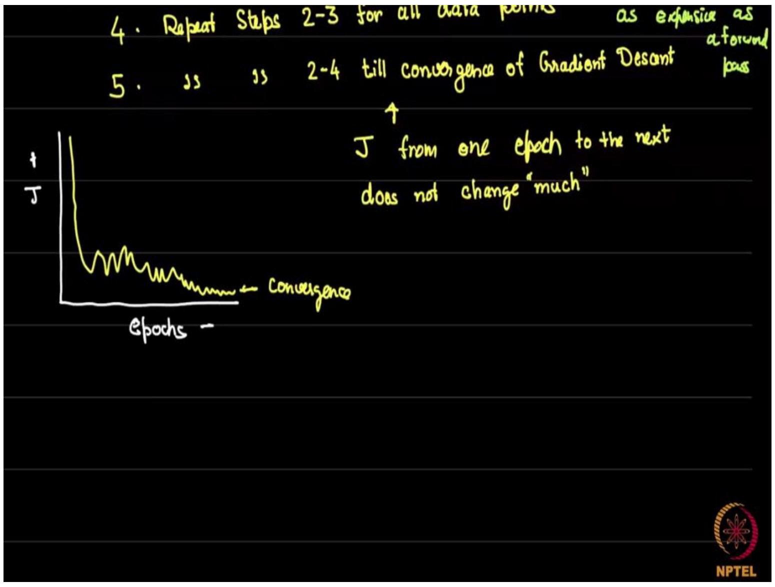
J from one epoch to the next does not change "much"

NPTEL

And then using this error you are somehow supposed to magically calculate this value $\frac{\partial J}{\partial w_j}$. So, this step of calculating this is what is called back prop as I have written here our expectation is if we do it right this entire process of calculating $\frac{\partial J}{\partial w_j}$ is only as expensive as a forward pass. So, now you repeat for this new w_j and do a forward prop and keep on repeating this each time for all data points.

Take an average or you have already taken this update so this is a stochastic gradient descent I will write this here; the way I have written it this is an SGD algorithm which we saw last week and we keep on repeating all these steps for new values of w until convergence of gradient descent.

(Refer Slide Time: 14:24)



4. Repeat Steps 2-3 for all data points — as expensive as a forward pass

5. " " 2-4 till convergence of Gradient Descent

↑

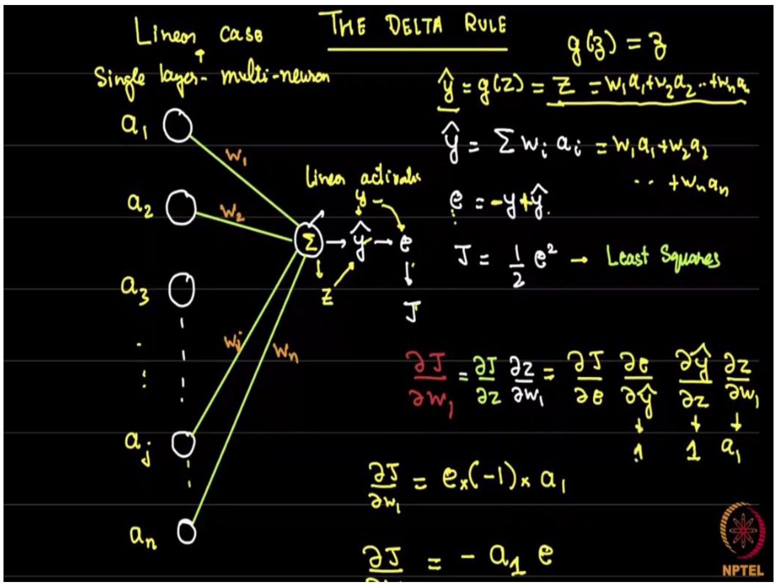
J from one epoch to the next does not change "much"

J
Epochs
Convergence

What do we mean by convergence? You typically plot how J varies with number of epochs. Remember one epoch is when you have seen the entire data set. In case you have 100 data points here in SGD you would have made 100 updates. So, after 100 updates you have seen all data points. Now suppose you plot J versus number of epochs which I will show you next week you will see something of this sort.

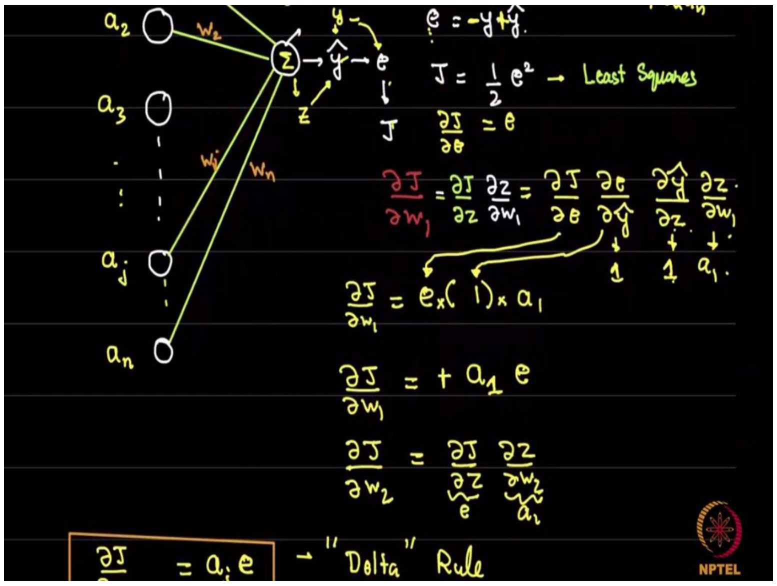
We will start high then so convergence happens somewhere here where J does not change by much. So, you will have to derive or define some predefined convergence limits. So, the key step here once again is just to remind you this is just calculating $\frac{\partial J}{\partial w_j}$ and let us now go ahead and do that in a few steps.

(Refer Slide Time: 15:20)


Linear case
THE DELTA RULE
Single layer - multi-neuron
a_1
a_2
a_3
a_j
a_n
w_1
w_2
w_j
w_n
Linear activation
Σ
y
$\hat{y}$
e
z
J
$g(z) = z$
$\hat{y} = g(z) = z = w_1a_1 + w_2a_2 \cdots + w_na_n$
$\hat{y} = \sum w_i a_i = w_1a_1 + w_2a_2 \cdots + w_na_n$
$e = -y + \hat{y}$
$J = \frac{1}{2}e^2 \rightarrow$ Least Squares
$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w_1} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_1}$
$1 \quad 1 \quad a_1$
$\frac{\partial J}{\partial w_1} = e \times (-1) \times a_1$
$\frac{\partial J}{\partial w_1} = -a_1 e$

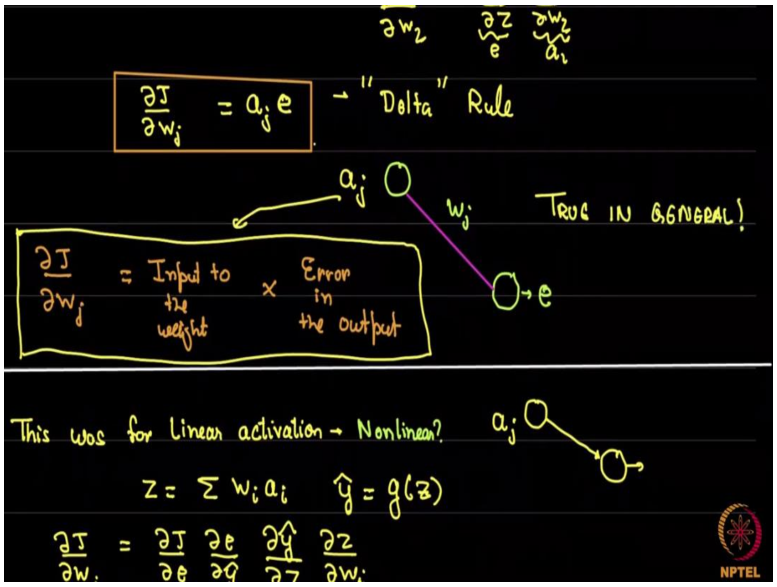
And what about $\frac{\partial J}{\partial e}$. So, you can see this here $\frac{\partial J}{\partial e}$ is differential of $\frac{1}{2}e^2$ with respect to e which is just e. So, $\frac{\partial J}{\partial e}$ is e, $\frac{\partial e}{\partial \hat{y}}$ is 1, $\frac{\partial \hat{y}}{\partial z}$ is 1 and $\frac{\partial z}{\partial w_1}$ is 1. So, all put together you get $\frac{\partial J}{\partial w_1}$ is $a_1 e$, I have written this here, but let us erase that. So that I was doing it for another reason, but let me not confuse. So, overall, what we notice is this $\frac{\partial J}{\partial w_1}$ is $a_1 e$.

(Refer Slide Time: 21:16)


a_2
a_3
a_j
a_n
w_2
w_j
w_n
Σ
y
$\hat{y}$
e
z
J
$e = -y + \hat{y}$
$J = \frac{1}{2}e^2 \rightarrow$ Least Squares
$\frac{\partial J}{\partial e} = e$
$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z}\frac{\partial z}{\partial w_1} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_1}$
$1 \quad 1 \quad a_1$
$\frac{\partial J}{\partial w_1} = e \times (1) \times a_1$
$\frac{\partial J}{\partial w_1} = + a_1 e$
$\frac{\partial J}{\partial w_2} = \underbrace{\frac{\partial J}{\partial z}}_{e}\underbrace{\frac{\partial z}{\partial w_2}}_{a_1}$
$\boxed{\frac{\partial J}{} = a_j e}$ → "Delta" Rule

Now I repeat this process and do let us say $\frac{\partial J}{\partial w_2}$ and you see that nothing changes except for it being $\frac{\partial J}{\partial z}\frac{\partial z}{\partial w_2}$. This whole number here still stays as e and this becomes a_2. So, you can see $\frac{\partial z}{\partial w_2}$ is simply a_2 in general $\frac{\partial J}{\partial w_j}$ will follow the same rule and it will simply be a_j times e.

(Refer Slide Time: 22:38)



$\partial w_2 \quad \underbrace{\partial z}_{e} \quad \underbrace{\partial w_2}_{a_1}$

$$\boxed{\frac{\partial J}{\partial w_j} = a_j e}$$ — "Delta" Rule

a_j w_j TRUE IN GENERAL!

$$\boxed{\frac{\partial J}{\partial w_j} = \text{Input to the weight} \times \text{Error in the output}}$$

$\to e$

This was for Linear activation → Nonlinear? a_j

$z = \Sigma w_i a_i \qquad \hat{y} = g(z)$

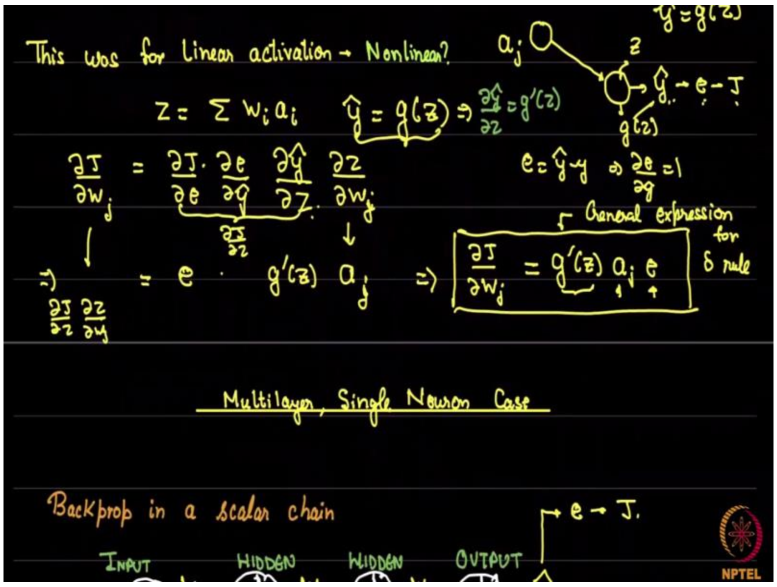
$$\frac{\partial J}{\partial w_j} = \frac{\partial J}{\partial e}\frac{\partial e}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial z}\frac{\partial z}{\partial w_j}$$

So, notice this $\frac{\partial J}{\partial w_j}$ is a_j times e. This is what is called the delta rule and we use this to great effect already in linear regression. So, in fact we use this even while deriving the linear regression formula. So, $\frac{\partial J}{\partial w_j}$ is a_j times e. Now let us look at the meanings of these terms. So, if you look at just these neurons the a_j neuron here and the output neuron. So, the input neuron and the output neuron are connected by a single weight w_j and that w_j.

And the output of this neuron is e or the error in the output is e. So, we can say that $\frac{\partial J}{\partial w_j}$ at least in the linear case seems to follow the rule that if I want $\frac{\partial J}{\partial w_j}$ I need to multiply the input to the weight which was a_j by the error in the output which was e. So, once again if I want this weight, I would multiply a_2 by the error in the output very simple rule it turns out that this is true in general even for complex networks.

If time permits, I will prove it if not you just take it on faith that is even in this network if I want $\frac{\partial J}{\partial w_j}$ of this, all I need to do is this input multiplied by error in this output. Now what is error in this output mean that I will clarify later, but the rule in true in general. So, that finishes the delta rule for the linear case. Now what happens in the nonlinear case.

(Refer Slide Time: 24:46)



This was for Linear activation → Nonlinear?

$z = \sum w_i a_i \quad \underbrace{\hat{y} = g(z)}_{} \Rightarrow \frac{\partial \hat{y}}{\partial z} = g'(z)$

$\hat{y} = g(z)$

$e = \hat{y} - y \Rightarrow \frac{\partial e}{\partial \hat{y}} = 1$

$\frac{\partial J}{\partial w_j} = \underbrace{\frac{\partial J}{\partial e} \cdot \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z}}_{\frac{\partial J}{\partial z}} \cdot \frac{\partial z}{\partial w_j}$

$\Rightarrow \frac{\partial J}{\partial z} \frac{\partial z}{\partial w_j} = e \cdot g'(z)\, a_j \Rightarrow \boxed{\frac{\partial J}{\partial w_j} = g'(z)\, a_j\, e}$

General expression for δ rule

Multilayer, Single Neuron Case

Backprop in a scalar chain

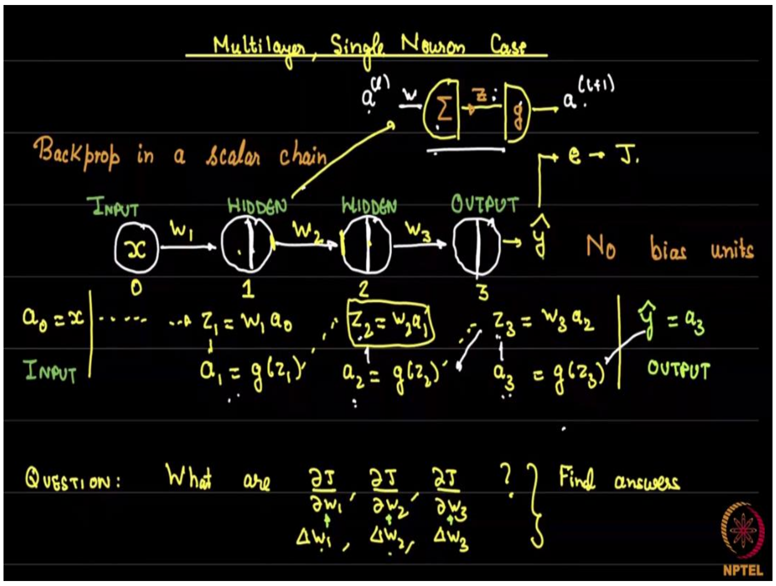
So, the nonlinear case we have a slight difference, so the difference is this; this is $\hat{y}$, $\hat{y}$ leads to e, e leads J till that everything is true. This is still z and giving g of z is what is $\hat{y}$. So, when do $\frac{\partial J}{\partial w_j}$ I need to do let us write it this way $\frac{\partial J}{\partial w_j}$ is $\frac{\partial J}{\partial z} \frac{\partial z}{\partial w_j}$ which is what I have written out here. This whole portion is $\frac{\partial J}{\partial z}$. So, you can see $\frac{\partial J}{\partial e} \frac{\partial e}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z}$.

So, the z remember is the linear output of this neuron $\frac{\partial J}{\partial e}$ is e, $\frac{\partial e}{\partial \hat{y}}$ is remember e is $(\hat{y} - y)$. So, simply $\frac{\partial e}{\partial \hat{y}}$ still stays as 1, but what is $\frac{\partial \hat{y}}{\partial z}$ last time $\hat{y}$ was z itself, but here $\hat{y}$ is g of z some non-linear function typically a sigmoid at least in the initial examples that we have taken as I have shown here, but in other cases it might not be.

So, $\frac{\partial \hat{y}}{\partial z}$ is g prime z it simply means I have taken a derivative of g with respect to z. So, here is the formula now $\frac{\partial J}{\partial w_j}$ is the old formula which was a_j multiplied by e multiplied by an additional term $g'(z)$. So, this is an important term here that makes an appearance. So, notice when $g(z)$ is z this simply becomes 1. So, this is b general expression for the delta rule.

Now why is it called the delta rule I will explain that shortly when I go to the multi-layer cases it will become a little bit clearer when we go to the final case, but all we have done here is taken these two cases. So, now you will notice input of the neuron exit error multiplied by some nonlinear function or the derivative of the nonlinear function that took us forward. This looks like a more complex formula than it actually is as you will see shortly. Now that we have seen a single layer.

(Refer Slide Time: 27:49)



Now let us take a more complex multi-layer case. I want to point out that this is something that I have made up this is not a practical example or an example mostly that you will find in any textbook. This is just here in order for you to understand what is actually happening within a neural network without some additional complications. So, I am calling this a scalar chain I am calling this a scalar chain because it is a neural network it is a chain.

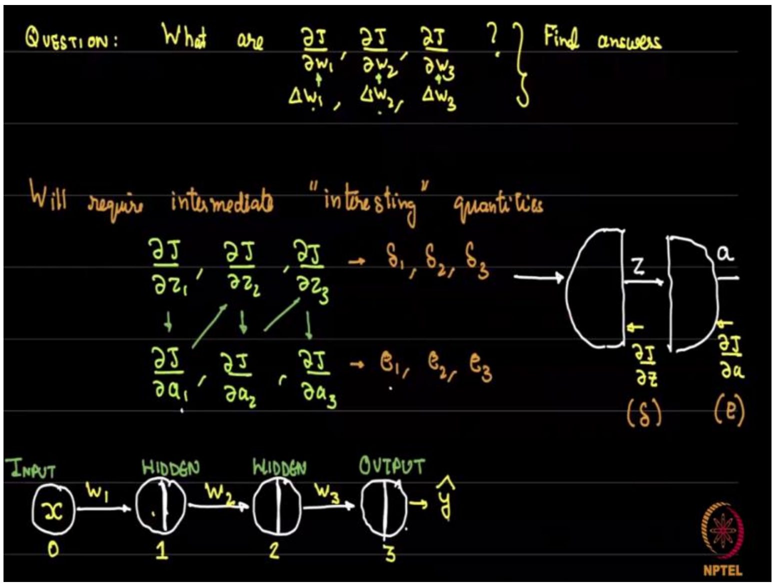
But everything is a scalar okay there is no vector here the input is a scalar, this one is a scalar there are no multiple neurons every single thing is a scalar here. So, this is what I call a scalar chain. In that case the forward prop becomes fairly simple to actually visualize. So, the forward prop goes like this you give x that I am calling the input a_0 is x. Now x gets multiplied by w_1 so z_1 is $w_1 a_0$.

There is no summation it is just a linear sum or it is just a linear transformation z_1 becomes $w_1 a_0$. So, we come here to the linear part remember this is the linear part, this is the nonlinear part easier to visualize it we think of the neuron as being broken up into two parts. In fact, as I will show later, I would recommend that you zoom in and think of it this way two parts the linear part which I am going to denote by sigma gives out z.

And the non-linear part which we can call g gives out a, which I typically call a-hat or simply a, let me remove the hat let us just call it a or let us call it $a_{(l+1)}$ for level $(l + 1)$ if level l went here and a weight went here. The weight got multiplied give z, g of z gives a. So, that is what is happening, but useful to think of this hidden neuron or the neuron being broken up into just like some a fruit or something or a coconut it is just broken into two parts.

So, z is $w_1 a_0$, a_1 here is g of this z. So, a_1 is g of z_1 immediately a_1 gets multiplied by w_2, so w_2 times a_1 is z_2 here and g of z_2 is a_2 that is what comes out here. Now a_2 gets multiplied by w_3 gives us z_3 and z_3 take g of z_3 and that gives us a_3 and finally we say well my final output $\hat{y}$ is nothing, but a_3 once again y hat leads to e, e leads to j we are assuming no bias units as I had mentioned earlier, we are just simply going to assume no bias units throughout.

(Refer Slide Time: 31:13)



QUESTION: What are $\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial w_3}$?
$\Delta w_1, \Delta w_2, \Delta w_3$
Find answers
Will require intermediate "interesting" quantities
$\frac{\partial J}{\partial z_1}, \frac{\partial J}{\partial z_2}, \frac{\partial J}{\partial z_3} \rightarrow \delta_1, \delta_2, \delta_3$
$\frac{\partial J}{\partial a_1}, \frac{\partial J}{\partial a_2}, \frac{\partial J}{\partial a_3} \rightarrow e_1, e_2, e_3$
z
a
$\frac{\partial J}{\partial z}$ (δ)
$\frac{\partial J}{\partial a}$ (e)
INPUT
HIDDEN
HIDDEN
OUTPUT
x
w_1
w_2
w_3
$\hat{y}$
0
1
2
3
NPTEL

The questions we want to ask is a simple question which is what are the gradients of j with respect to the intermediate ways that is what is $\frac{\partial J}{\partial w_3}, \frac{\partial J}{\partial w_2}, \frac{\partial J}{\partial w_1}$. Another way to ask it if I perturb w_1 by a little bit how much will that affect J. Notice perturbing w_1 affects z_1, a_1, z_2, a_2 it does not affect w_2, w_2 is just a variable it is an independent parameter z_2, a_2, z_3, a_3, $\hat{y}$, e and then J.

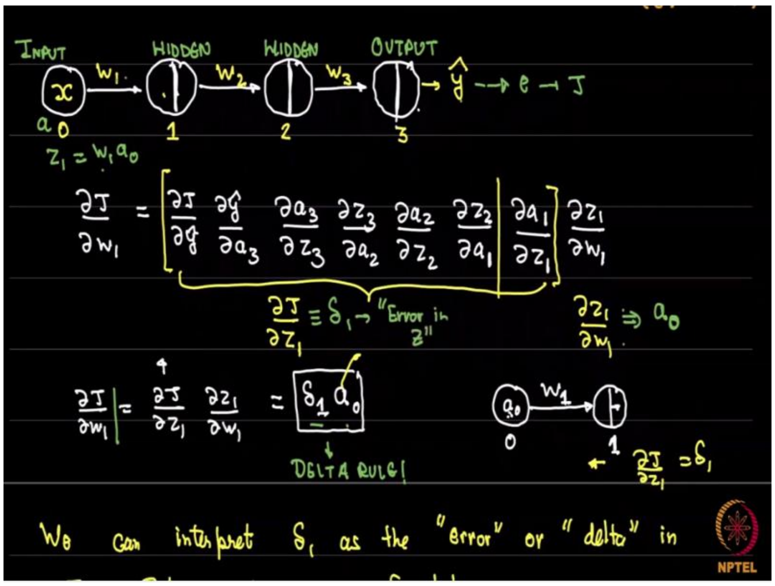
So, we want to trace that entire process the way we are going to do it is reverse the chain of causality and that is what we are going to do. These if you calculate $\frac{\partial J}{\partial w_1}$ you can calculate delta w_1 based on gradient descent etcetera. These are the final answers that are desired. Now as it turns out to calculate these final answers, we require some intermediate answers.

These intermediate answers are just like last time instead of calculating $\frac{\partial J}{\partial w}$ if I want $\frac{\partial J}{\partial w_1}$ well I will need $\frac{\partial J}{\partial z_1}$ because w_1 did affect z_1 so I need $\frac{\partial J}{\partial z_1}$. Now, if I need $\frac{\partial J}{\partial z_1}$; z_1 affects a_1. So, I need $\frac{\partial J}{\partial a_1}$. Now, if I need $\frac{\partial J}{\partial a_1}$ that means I require $\frac{\partial J}{\partial z_2}$ because a_1 affect z_2 and so on and so forth z_2 requires $\frac{\partial J}{\partial a_2}$ this requires z_3 and this requires a_3.

So, since we have so many quantities, we are going to give it names whenever there is a $\frac{\partial J}{\partial z}$ we will call it delta. So, just like z_1, z_2, z_3 we have $\delta_1, \delta_2, \delta_3$ with obvious meanings. δ_1 is $\frac{\partial J}{\partial z_1}$; δ_2 is $\frac{\partial J}{\partial z_2}$ and δ_3 is $\frac{\partial J}{\partial z_3}$. Similarly, wherever we have a $\frac{\partial J}{\partial a}$ we will call it e. So, $\frac{\partial J}{\partial a_1}$ is e_1; $\frac{\partial J}{\partial a_2}$ is e_2; and $\frac{\partial J}{\partial a_3}$ is e_3.

So, you can think of our process as being when we go forward, we calculate z and a, when we go backward, we calculate $\frac{\partial J}{\partial a}$ and $\frac{\partial J}{\partial z}$.

(Refer Slide Time: 33:51)



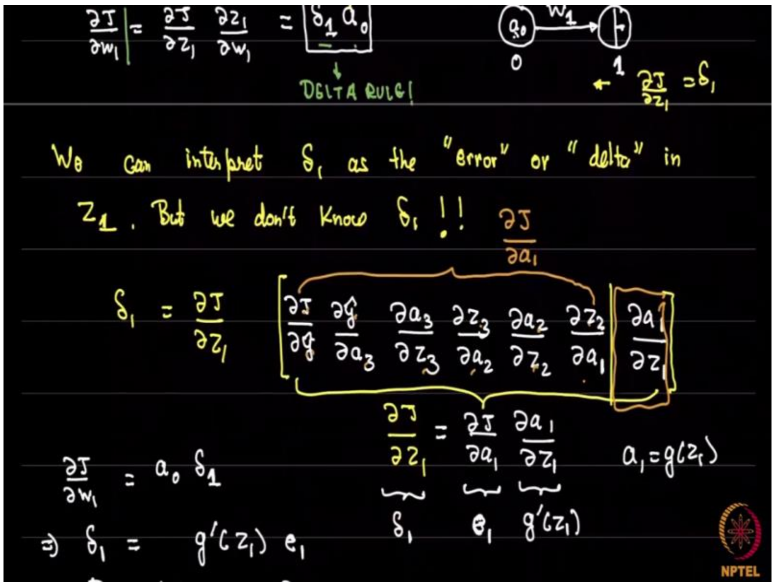
INPUT
HIDDEN
HIDDEN
OUTPUT
x
w_1
w_2
w_3
$\hat{y} \dashrightarrow e \rightarrow J$
a_0
1
2
3
$z_1 = w_1 a_0$
$\frac{\partial J}{\partial w_1} = \underbrace{\left[\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1} \frac{\partial a_1}{\partial z_1}\right]}\frac{\partial z_1}{\partial w_1}$
$\frac{\partial J}{\partial z_1} \equiv \delta_1 \rightarrow$ "Error in z"
$\frac{\partial z_1}{\partial w_1} \Rightarrow a_0$
$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_1} \frac{\partial z_1}{\partial w_1} = \boxed{\delta_1 a_0}$
DELTA RULE!
a_0
w_1
0
1
$\frac{\partial J}{\partial z_1} = \delta_1$
We can interpret δ_1 as the "error" or "delta" in
NPTEL

So, I will repeat the figure because we require a reference to the figure here. Let us say I am calculating $\frac{\partial J}{\partial w_1}$. So, once again remember this $\hat{y}$ leads to e and e leads to J. So, in a scalar chain it is very obvious to see what we are doing. when I want $\frac{\partial J}{\partial w_1}$ I am going to follow this entire process of first calculating $\frac{\partial J}{\partial z_1} \frac{\partial z_1}{\partial w_1}$ this is after all the same trick that we applied in the delta rule.

So, we are going to apply the same rule for the same idea and well $\frac{\partial J}{\partial z_1}$ is nothing, but delta 1 $\frac{\partial J}{\partial z_1}$ our notation was $\frac{\partial J}{\partial z_1}$ is δ_1 of course that does not mean we have calculated it we have just given it a name, but what is $\frac{\partial z_1}{\partial w_1}$. Notice $\frac{\partial z_1}{\partial w_1}$ is z_1 how did it come from $w_1 * a_0$ which is x and let us call this a_0. So, if a_0 is multiplied by w_1 it gives z_1 this means $\frac{\partial z_1}{\partial w_1}$ is nothing, but a_0.

So, we get $\frac{\partial J}{\partial w_1}$ is $\delta_1 * a_0$ which looks just like the delta root. Notice, input we want $\frac{\partial J}{\partial w}$ its input to this weight which is a 0 multiplied by the error in the output, but the error in the output here is simply delta error in z. You can think of it as the error in z. So, this portion is the immediate output of this neuron is z and the error in that or basically is what we call $\frac{\partial J}{\partial z}$ well we call this the error it is not really the error, but it is something like the error.

(Refer Slide Time: 36:05)



$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial z_1}\frac{\partial z_1}{\partial w_1} = \boxed{\delta_1 a_0}$$

DELTA RULE!

$$\begin{tikzcd} a_0 \arrow[r, "w_1"] & 1 \end{tikzcd}$$

0 1

$\leftarrow \frac{\partial J}{\partial z_1} = \delta_1$

We can interpret δ_1 as the "error" or "delta" in z_1. But we don't know δ_1 !!

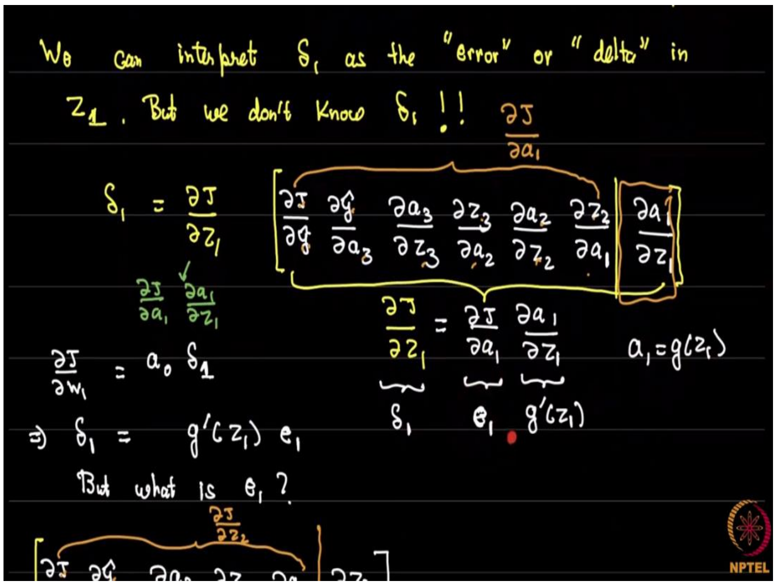
$$\delta_1 = \frac{\partial J}{\partial z_1} \left[\overbrace{\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1}}^{\frac{\partial J}{\partial a_1}} \boxed{\frac{\partial a_1}{\partial z_1}} \right]$$

$$\frac{\partial J}{\partial w_1} = a_0 \delta_1 \qquad \underbrace{\frac{\partial J}{\partial z_1}}_{\delta_1} = \underbrace{\frac{\partial J}{\partial a_1}}_{e_1} \underbrace{\frac{\partial a_1}{\partial z_1}}_{g'(z_1)} \qquad a_1 = g(z_1)$$

$$\Rightarrow \delta_1 = g'(z_1)\, e_1$$

So, this is exactly the delta rule except all it is saying is multiply a_0 multiplied by the delta here that will give you $\frac{\partial J}{\partial w_1}$. So that is the delta rule so as I have written here, we can interpret delta 1 as the error or delta in z_1, but unfortunately, we know a_0; a_0 is how we started the calculation. We started the calculation a_0, but we do not know δ_1.

(Refer Slide Time: 36:36)



We can interpret δ_1 as the "error" or "delta" in z_1. But we don't know δ_1 !!

$$\delta_1 = \frac{\partial J}{\partial z_1} = \left[\overbrace{\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3} \frac{\partial a_3}{\partial z_3} \frac{\partial z_3}{\partial a_2} \frac{\partial a_2}{\partial z_2} \frac{\partial z_2}{\partial a_1}}^{\frac{\partial J}{\partial a_1}} \boxed{\frac{\partial a_1}{\partial z_1}} \right]$$

$$\frac{\partial J}{\partial a_1} \frac{\partial a_1}{\partial z_1} \checkmark$$

$$\frac{\partial J}{\partial z_1} = \frac{\partial J}{\partial a_1} \frac{\partial a_1}{\partial z_1} \qquad a_1 = g(z_1)$$

$$\underbrace{\frac{\partial J}{\partial z_1}}_{\delta_1} \quad \underbrace{\frac{\partial J}{\partial a_1}}_{e_1} \quad \underbrace{\frac{\partial a_1}{\partial z_1}}_{g'(z_1)}$$

$$\frac{\partial J}{\partial w_1} = a_0 \delta_1$$

$$\Rightarrow \delta_1 = g'(z_1)\, e_1$$

But what is e_1 ?

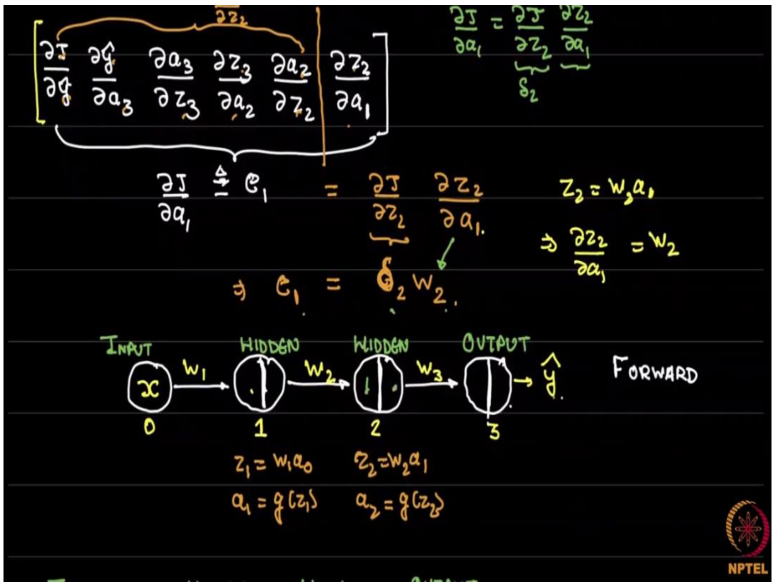
$$\frac{\partial J}{\partial z_2}$$

Now how do we calculate δ_1? δ_1 is $\frac{\partial J}{\partial z_1}$, but $\frac{\partial J}{\partial z_1}$ can now be thought of as one more step when I want $\frac{\partial J}{\partial z_1}$ I will calculate $\frac{\partial J}{\partial a_1}$ and multiply by whatever connects the 2. So, here it is $\frac{\partial J}{\partial z_1}$ is this whole calculation $\frac{\partial J}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_3}$ etcetera ignores all that just think of this as ∂a_1 all I am saying is $\frac{\partial J}{\partial z_1}$ is $\frac{\partial J}{\partial a_1} \frac{\partial a_1}{\partial z_1}$ that is a fairly non-controversial statement.

So, that is what is written here $\frac{\partial J}{\partial z_1}$ is $\frac{\partial J}{\partial a_1} \frac{\partial a_1}{\partial z_1}$, but what is $\frac{\partial a_1}{\partial z_1}$ how was a_1 calculated from z_1 it was simply a_1 is $g(z_1)$ so $\frac{\partial a_1}{\partial z_1}$ is simply g prime of this. So, now you notice δ_1 is $\frac{\partial J}{\partial a_1}$ that has a name this is just the name it is not a calculation it is a name it is $e_1 \; * g'(z_1)$ so now notice $\frac{\partial J}{\partial w_1}$ was $a_0 \delta_1$.

But δ_1 was $e_1 \; * g'(z_1)$. So, once again we have the question, I accept this for calculating this you needed δ_1 for calculating δ_1 you needed e_1, but what is e_1 we only have reposted to the figure again if I want e_1 here I need $\frac{\partial J}{\partial z_2}$ because that is the next calculation to this one.

(Refer Slide Time: 38:20)



$$\Big[\overbrace{\frac{\partial J}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2}\frac{\partial a_2}{\partial z_2}}^{\partial z_2}\Big|\frac{\partial z_2}{\partial a_1}\Big]$$

$$\frac{\partial J}{\partial a_1} = \underbrace{\frac{\partial J}{\partial z_2}}_{\delta_2}\frac{\partial z_2}{\partial a_1}$$

$$\frac{\partial J}{\partial a_1} \triangleq e_1 = \frac{\partial J}{\partial z_2}\frac{\partial z_2}{\partial a_1} \qquad z_2 = w_2 a_1$$

$$\Rightarrow \frac{\partial z_2}{\partial a_1} = w_2$$

$$\Rightarrow e_1 = \delta_2 w_2.$$

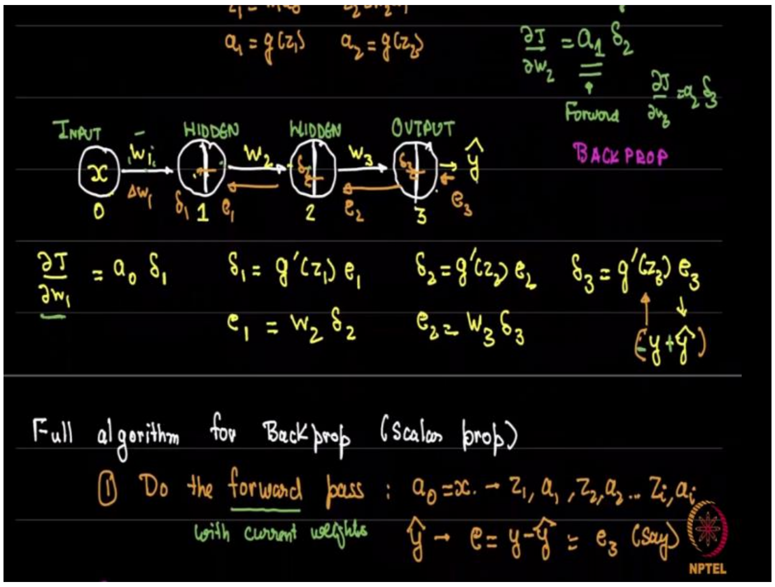
$z_1 = w_1 a_0$ $\quad z_2 = w_2 a_1$

$a_1 = g(z_1)$ $\quad a_2 = g(z_2)$

So, we will go back here we will say okay I want $\frac{\partial J}{\partial e_1}$ or sorry $\frac{\partial J}{\partial a_1}$ and $\frac{\partial J}{\partial a_1}$ is nothing, but $\frac{\partial J}{\partial a_1}$ is $\frac{\partial J}{\partial z_2}$ because that is the next calculation multiplied by $\frac{\partial z_2}{\partial a_1}$. Now this simply had a name this was called δ_2, but what about z_2 and a_1 what is their connection. So, let us go back here this is z_2 here a_1 here and how was it created z_2 was created by $w_2 a_1$, therefore $\frac{\partial z_2}{\partial a_1}$ is nothing, but w_2.

So, we write that here $\frac{\partial z_2}{\partial a_1}$ is w_2. So e_1 now becomes δ_2 multiplied by w_2 very good. So, now we have e_1 here which requires δ_2 what does δ_2 require e_2, what does e_2 require δ_3. What does δ_3 require e_3 and what does e_3 require simply $\frac{\partial J}{\partial \hat{y}}$. So, this is the entire process of back prop. So, it will look strange, but let me now show it to you step by step.

(Refer Slide Time: 39:54)



$a_1 = g(z_1)$ $\quad a_2 = g(z_2)$

$$\frac{\partial J}{\partial w_2} = a_1 \delta_2 \qquad \frac{\partial J}{\partial w_3} = a_2 \delta_3$$

$$\frac{\partial J}{\partial w_1} = a_0 \delta_1 \qquad \delta_1 = g'(z_1)\,e_1 \qquad \delta_2 = g'(z_2)\,e_2 \qquad \delta_3 = g'(z_3)\,e_3$$

$$e_1 = w_2 \delta_2 \qquad e_2 = w_3 \delta_3 \qquad (y, \hat{y})$$

Full algorithm for Back prop (Scalar prop)

① Do the forward pass : $a_0 = x \to z_1, a_1, z_2, a_2 \ldots z_i, a_i$

with current weights $\quad \hat{y} \to e = y - \hat{y} = e_3$ (say)

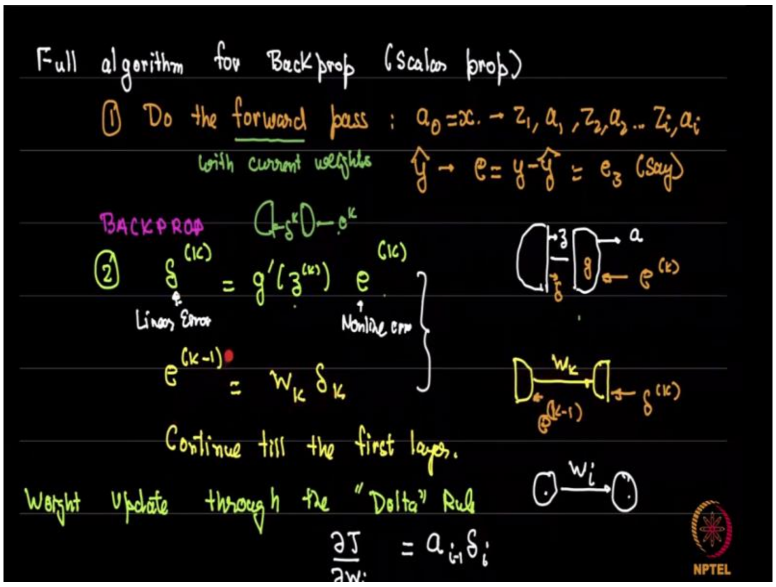
So, the way we do it is as follows. We want $\frac{\partial J}{\partial w_1}$. $\frac{\partial J}{\partial w_1}$ is simply this input multiplied by this output. So $\delta_1 * a_0$ that I prove, δ_1 is g prime multiplied by e_1. So, we calculated that if we want e_1 the connecting link between e_1 and δ_2 is w_2. So, e_1 is delta w_2 times δ_2 if I want δ_2 I want e_2 δ_2 is g prime times e_2. Now you can see here some kind of recursion relationship here if I want e_2; e_2 is simply w_3 times δ_3.

And if I want δ_3, δ_3 is simply g prime times e_3. Now what is e_3? The final error in the output which is simply $(y - \hat{y})$ or $(\hat{y} - y)$ as we define them. So, here is the full algorithm for the back prop in scalar case. So, you might say okay wait a second you only calculated $\frac{\partial J}{\partial w_1}$ what about $\frac{\partial J}{\partial w_2}$, $\frac{\partial J}{\partial w_2}$ follows the same logic $\frac{\partial J}{\partial w_2}$ is simply going to be the input which is a1 multiplied by the output which is δ_2.

Now a1 was calculated during forward prop and δ_2 was calculated during back prop so there it is. Now notice for each one of these weights just like in the forward calculation for each the entire forward prop was done only once. The entire this is the back prop this is done only once, we do not do different calculations for w_1, w_2, w_3 we simply calculate from here δ_3, δ_2, δ_1 finish.

And then after that you can calculate $\frac{\partial J}{\partial w_2}$ as $a_1\delta_2$ and similarly $\frac{\partial J}{\partial w_3}$ as $a_2\delta_3$.

(Refer Slide Time: 42:05)



Full algorithm for Back prop (Scalar prop)

① Do the forward pass : $a_0 = x \rightarrow z_1, a_1, z_2, a_2 \ldots z_i, a_i$

with current weights $\hat{y} \rightarrow e = y - \hat{y} = e_3$ (say)

BACKPROP

② $\delta^{(k)} = g'(z^{(k)})\, e^{(k)}$

Linear Error Nonlinear error

$e^{(k-1)} = w_k \delta_k$

Continue till the first layer.

Weight update through the "Delta" Rule

$\frac{\partial J}{\partial w_i} = a_{i-1}\delta_i$

δ^k
e^k
z
δ
a
$e^{(k)}$
w_k
$e^{(k-1)}$
$\delta^{(k)}$
w_i
NPTEL

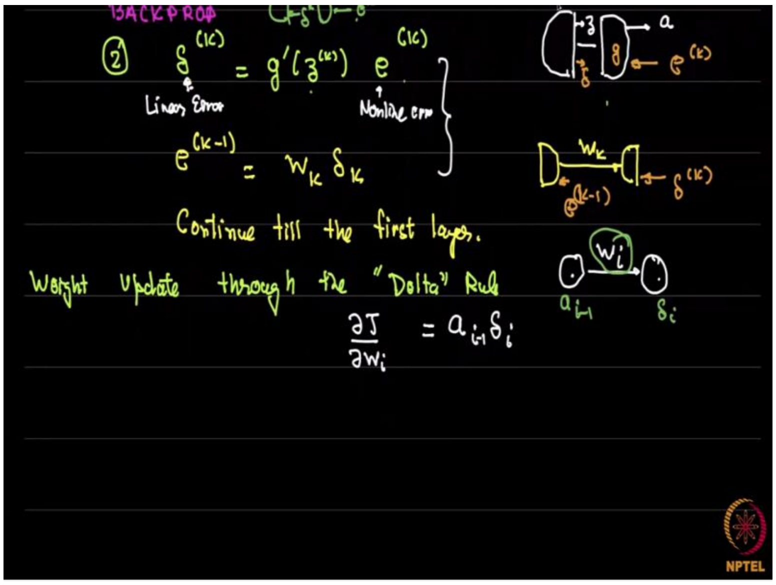
So, let me write the full algorithm for the scalar propagation. We first do one forward pass with the current weights. So, this is important we do the forward pass with the current weights. So, I have written that calculation a_0 is x then you multiply that get $z_1 a_1, z_2 a_2, z_3 a_3$ so on and so forth to keep on going till the end till you find out error. So, let us say it is e_3 at this point.

Now you start the back prop. How do you start the back propagation from e_3 calculate δ_3 from δ_3 calculate e_2 δ_2, e_1 δ_1 there it is. Once you have all the deltas and the e's then immediately in fact once you have all the deltas the e's are just intermediate calculations. Once you have all the deltas you can calculate all the $\frac{\partial J}{\partial w}$. So, the formula that comes is in the same location remember if you have e_k coming here you have δ_k coming here.

So, the relationship between the two of them is simply through g prime. So, delta is g prime multiplied by e_k as I have drawn in this figure here. Now, if you want to go to the next step so the output here is delta k and you want e_{k-1} we know e_{k-1} is $w_k \delta_k$ because the a here and the z here were related through w. So, similarly e_{k-1} is $w_k \delta_k$ then you keep on repeating this two-step dance just like in the forward prop.

You do linear g linear, linear g in the reverse thing you do g or g prime linear, g prime linear, g prime linear it is an exact analog of what happened in the forward prop.

(Refer Slide Time: 44:09)



BACKPROP

② $\delta^{(k)} = g'(z^{(k)})\, e^{(k)}$

Linear Error — Nonlinear err

$e^{(k-1)} = w_k \delta_k$

Continue till the first layer.

Weight Update through the "Delta" Rule

$$\frac{\partial J}{\partial w_i} = a_{i-1}\delta_i$$

z
a
δ
g
$e^{(k)}$
w_k
$e^{(k-1)}$
$\delta^{(k)}$
w_i
a_{i-1}
δ_i
NPTEL

Finally, we do the weight update between the two weights using the delta rule. So, you have a_{i-1} and you have $\delta_i w_i$ connects the two and you have $\frac{\partial J}{\partial w_i}$ is a_{i-1} multiplied by δ_i. So, this is back prop through a scalar chain and I hope you got at least some idea of how back prop is being done. Now the next topic is to go to the actual multi-layer perceptron case, but since this video has been long as I said at the beginning, I will move that to the next video. So, I will see in the next video. Thank you.