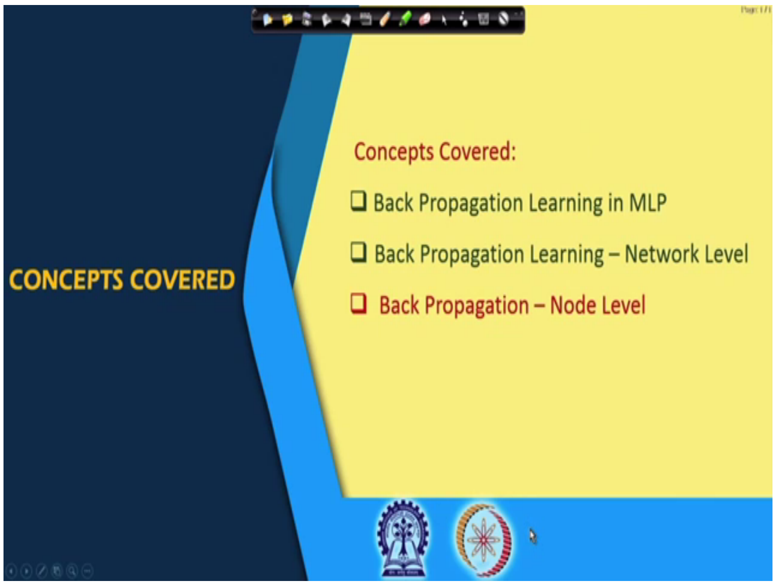
Deep Learning

Prof. Prabir Kumar Biswas
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 27
Backpropagation Learning – Example III

Hello welcome to the online certification course on Deep Learning. So, we are discussing about the Backpropagation Learning algorithms with some examples.

(Refer Slide Time: 00:40)



CONCEPTS COVERED

Concepts Covered:

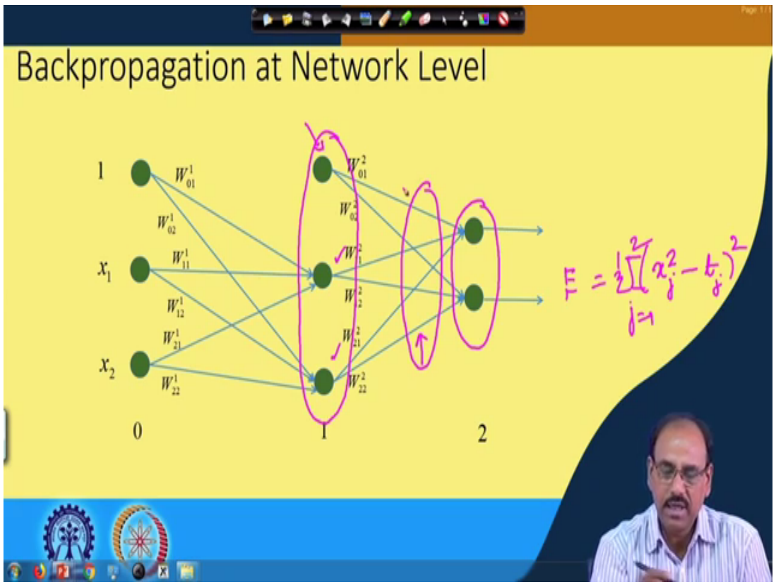
- ❑ Back Propagation Learning in MLP
- ❑ Back Propagation Learning – Network Level
- ❑ Back Propagation – Node Level

In the last class, we have taken an example at the network level that is given a feed forward neural network how to compute the error at the output node and then how do you propagate that error in the backward direction from output towards input layer and as you propagate the error from output layer to the input layer how the weights in every layer are updated.

The whole purpose of this back propagation algorithm and weight updation is that we want that output error that is the difference between the actual output that you get from the network and your target output that should be minimized. So, in the previous class we have discussed this with respect to a network in the network level.

Today's lecture we will see that the same back propagation or the gradient propagation is applicable within a node itself because every element in a neural network is nothing, but a neuron or a node and within the node as the complexity of the node increases, the node may also consist of a number of layers. It is not necessarily that every node will compute in a single layer fashion, so I can have multiple layers within a node and today we will see that how this gradient can flow within a node from output of the node to the input of the node.

(Refer Slide Time: 02:18)



Backpropagation at Network Level

$$E = \frac{1}{2}\sum_{j=1}^{2}(x_j^2 - t_j)^2$$

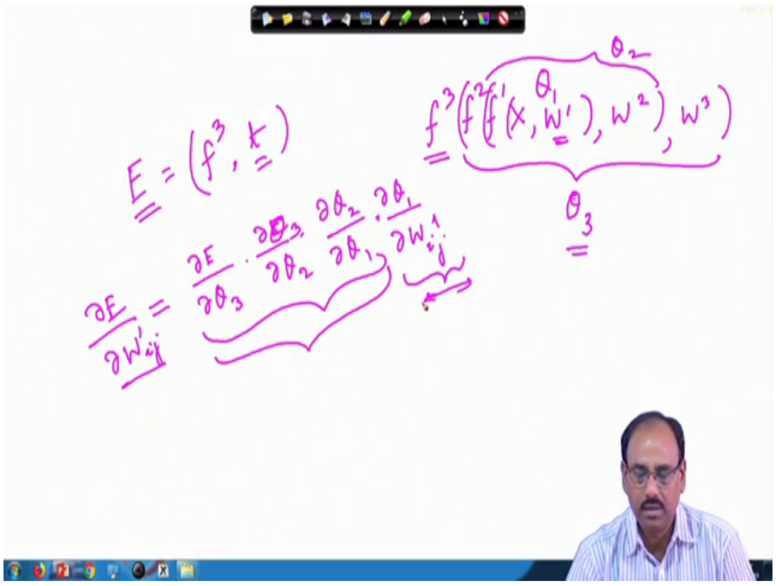
So, before we go into that let us just try to recapitulate what we have done in the previous class. So, in the previous class we had taken a simple 2 layer neural network. So, as is shown over here, so this neural network has an output layer having only 2 nodes and we also had an input layer which is also having 2 nodes that is node 1 and node 2 in addition we had another node which gives you the bias term.

And we had an input layer, so this was the hidden layer and we had an input layer again consisting of 2 nodes because we had 2 dimensional feature vectors x 1 x 2, but the number of nodes in the hidden in the input layer was also three because we wanted to augment these input vectors by an additional term by an additional component which is equal to 1 which takes care of the bias in every node and there we have seen that the error that you computed the output.

So, we had an error energy which was given by half of say your actual output x j and because this was a second there. So, we put it as x j 2 minus t j take the square of this and some of this for j is equal to 1 to 2.

So, that is the sum of squared error that you computed output and you take the gradient of this with respect to the weights between the hidden layer and the output layer and accordingly by gradient descent algorithm, you go on updating the weights of this hidden layer to the output layer then what we had seen is that whatever gradient that you get over here. You remember one thing that we discussed about, because this network is a multi layer network.

(Refer Slide Time: 04:33)



$$E = (f^3, t)$$

$$f^3(\underbrace{f^2(\overbrace{f^1(\overset{\theta_1}{X, W^1})}^{\theta_2}, W^2)}_{\theta_3}, W^3)$$

$$\frac{\partial E}{\partial W^1_{ij}} = \frac{\partial E}{\partial \theta_3} \cdot \frac{\partial \theta_3}{\partial \theta_2} \cdot \frac{\partial \theta_2}{\partial \theta_1} \cdot \frac{\partial \theta_1}{\partial W^1_{ij}}$$

So, if my input is X I have a function say f 1 which works on X along with and weight matrix or set of weight values which has a W 1 and this output this particular function is inputted to the next layer, which computes say f 2 along with the corresponding weights W 2. This again in turned inputted to the next layer which computes say f 3 along with the corresponding weights in W 3 and this continues this goes on and this is the final output that we get assuming that its a 3 layer network.

So, the output comes from the third layer which is the f 3. So, when I compute the error, the error E is computed with respect to f 3 and our target output that is the target that we want to have. So, here you find that this E or the error is actually available to f 3 or f 3 which is a function of let us call this argument as theta 3.

So, actually error is visible to theta 3, but error is not visible to which was output of the input layer which is the theta 1. Similarly this one I call as theta 2. So, error is actually visible to theta 3, it is not visible to theta 1. So, if I want to compute the gradient of error with respect to W 1 I cannot compute it directly. So, what I have to do is I have to compute the error with respect to theta 3 gradient of error with respect to theta 3, then multiply that way through the gradient of theta 3 with respect to theta 2 multiply that with the gradient of theta 2 with respect to theta 1 and multiply that with the gradient of theta 1 with respect to W.

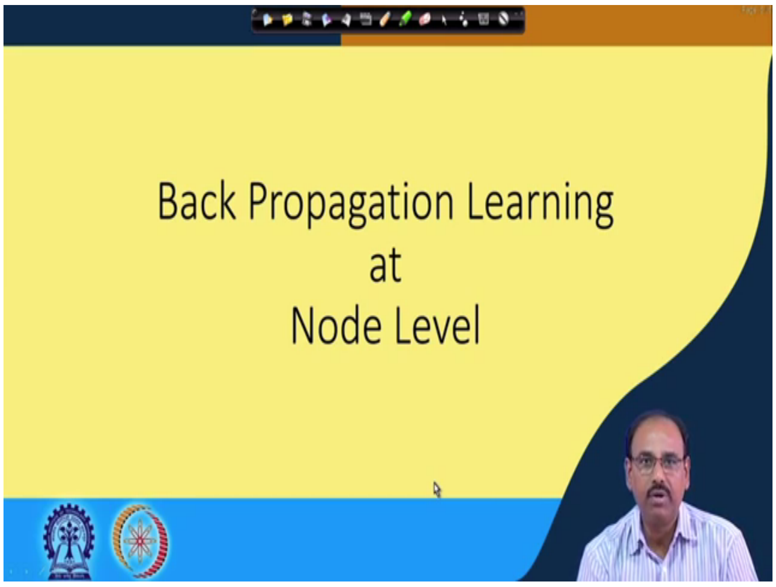
So, what I have to compute is if I want to compute del E say del W 1 let us put del W 1 i j because the W 1 over here is a matrix. So, this will be del E del theta 3 into del E del theta 2 into sorry del theta 3 del theta 2 into del theta 2 del theta 1 into del theta 1 del W i j 1. So, here you find that this product that is del theta 3 del E del theta 3 into del theta 3 del theta 2 into del theta 2 del theta 1 these are the gradients which are being back propagated from the output layer to this input layer and del theta 1 del W i j 1 that is what is the local gradient.

So, to find out the gradient of the error with respect to W i j 1 I have two components; one component is the propagated term which is this and the other component is the local gradient. So, what we are doing is we are multiplying the propagated gradient with the local gradient to get the actual gradient ok. So, this is the same concept which has been applied over here that whatever was gradient at the output the same gradient when it comes over here it is multiplied with the local gradient to find out how the error is propagated to the input side. And you remember one more thing that every node in the hidden layer is feeding input to every node in the next layer.

And that input is fed through a corresponding weight value. So, when we are propagating error or the gradient in the backward direction the gradient which is propagated from here through this and the gradient which is propagated through this that will be weighted by the corresponding weight values and added over here and this summation is or the aggregate of the gradient which is collected at this hidden layer node this is the total gradient which is available here through back propagation.

And this total gradient now has to be multiplied with the local gradient to find out what is the gradient off the output on this input weight and accordingly I have to adjust these weight components and that is what we have discussed in our previous class. So, now let us see that how the same concept works in the node level also.

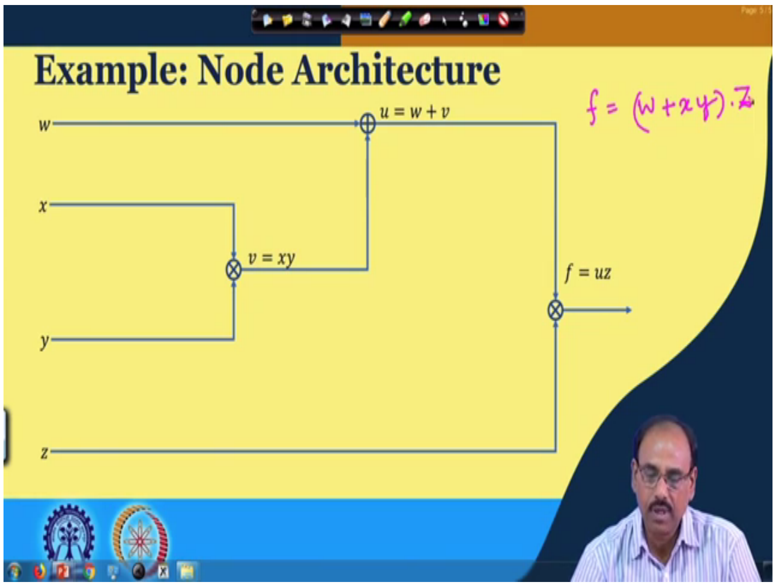
(Refer Slide Time: 09:54)



Back Propagation Learning at Node Level

So, what we are going to consider now is back propagation learning at node level. So, let us see how it works.

(Refer Slide Time: 10:03)



Example: Node Architecture

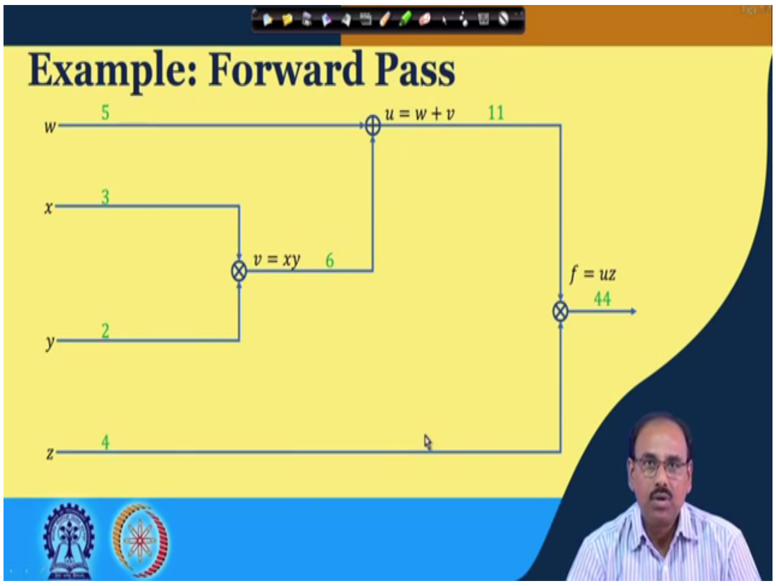
So, to explain this I will consider a simple example. Let us assume that node implements a function say f is equal to W plus x y into z. So, this is a function that this node computes. So, if I break or if I expand the particular node here in a hierarchical manner on or in a staged manner how this node actually computes this (Refer Time: 10:46). So, that every function is broken into smaller primitive set of functions.

So, what I do is, you are find that there are few hierarchical levels, so in the first level your compute x into y through a multiplier block. So, this is a multiplier block which computes x into y then I have an addition block which computes so this intermediate variable x into y that is assigned to an intermediate variables of v then I have a an adder block which computes w plus xy at w plus v because v is the intermediate variable which is equal to x into y. So, this w plus x y is assigned to another intermediate variable u and then finally, I compute f which is nothing, but u into z or which is nothing, but u into w plus x into y.

So, this is my total node structure and as we have discussed before that every back propagation learning algorithm is preceded by a forward pass. The forward pass in which the functional value is computed or coming to a network layer given an input vector and whatever with a set of weights at different layers available at that point of time using that you go for a feed forward pass and in the feed forward pass you compute what is the output vector that you are get and this if this output vector matches with the target then I do not have any error. So, that means, I do not have to go for back propagation, I do not have to I do not have anything to be adjust it, because whatever output I get that is my desired output or the target output.

But in case of an error when the output does not match with the target then I have an error and the gradient of the error has to be back propagated for weight adjustment. So, here also I have to have a forward pass; that means, whatever be the value of x y z and w using those values I had to compute what will be the value of f. And then if necessary; that means, if the f that you get is not the desired value then I have an error and the gradient of that error has to be propagated in the backward direction for updation wherever it is required. So, now let us see how this forward pass actually works.

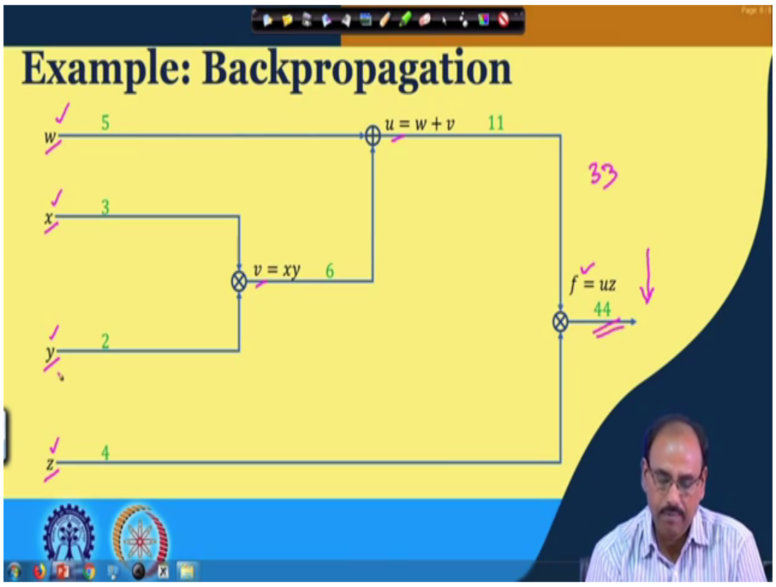
(Refer Slide Time: 13:41)



Example: Forward Pass

So, to see how the forward pass works let us assign values to w x y and z. So, you assume w has a value 5, x has a value 3, y has a value 2 and z has a value w z has a value 4. So, accordingly we will have v which is equal to x into y that that gets a value 6 then u which is w plus v or in other words it is w plus x into y if you compute this it will be 11 and then finally, f which is u into z or it is x w plus x y into z that will assume a value 44 four. So, in the forward pass I get the output to be to have a value 44 right. Now let us see how the backward pass we will work on this.

(Refer Slide Time: 14:45)



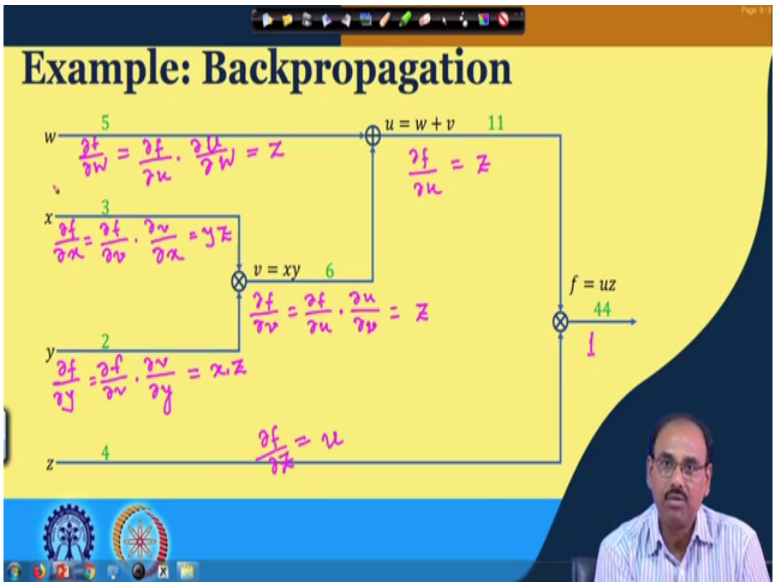
Example: Backpropagation

So, to work on backward pass see what I want to find out is, what is the sensitivity of output f? Sensitivity of f on w I want to compute what is what is the sensitivity of f on x, what is the sensitivity of f on y, what is the sensitivity of f on z? Or in other words if I need any correction on f suppose my desired value of f was something like 33, whereas, I am getting a value of f which is 44: that means, value of f has to be reduced.

So, in order to reduce the value of f whether I should increase value of w or I should also reduce value of w, whether I should increase value of x or I should reduce value of x, whether I should increase value of y or I should reduce value of y. Whether I should increase value of z or I should reduce value of z that is what is the sensitivity of f on w or on x or on y or on z and this is what I need I do not need how f varies with u or how f varies with v that is not my concern.

But my concern is how f varies with wxy and z, but I cannot directly compute del f del w or del f del x, the reason being f is not directly visible to w f is not directly visible to x, f is not visible directly visible to y, but yes I can compute del f del z because f is directly visible to z. So, now, let us see that how this gradient in this case in the backward direction can be computed again using the chain rule. So, for doing this I assume like what the gradient that you get at the output is say 1 then assuming this how it is propagated in the backward direction.

(Refer Slide Time: 17:17)


Example: Backpropagation
w
5
$u = w + v$
11
$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial w} = z$
$\frac{\partial f}{\partial u} = z$
x
3
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial x} = yz$
$v = xy$
6
$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial v} = z$
$f = uz$
44
1
y
2
$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial y} = xz$
z
4
$\frac{\partial f}{\partial z} = u$

So, here what I compute is del f del z and as I said I can directly compute it. So, del f del z is nothing, but in this particular case it is u and here what I have to compute is del f del u and del f del u is nothing, but z. Here I what I need to compute is what is the value of del f del w and as we said because f is not directly visible to w. So, I cannot really compute del f w del f del w directly. So, for doing this I have to make use of the chain rule. So, I have to make use of what is del f del u in this case that has to be multiplied by what is del u del w.

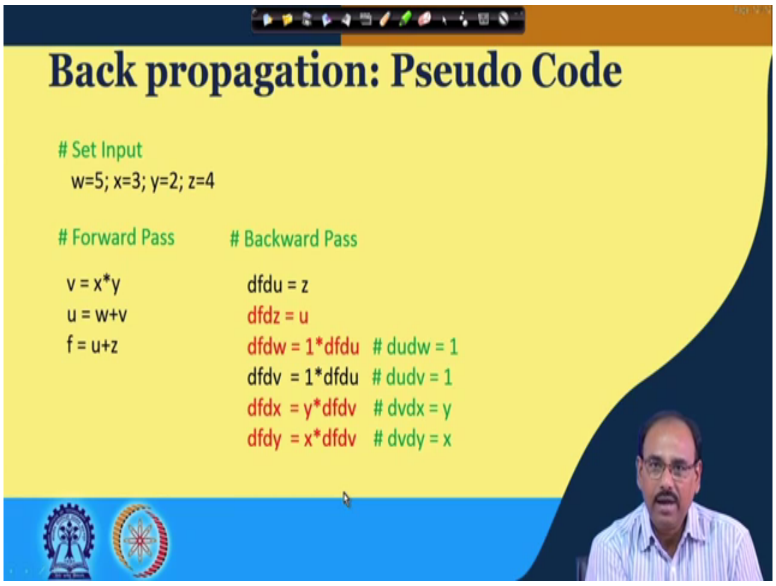
So, this is what I was saying that del f del u is the back propagated gradient up to this point and del u del w is the local gradient. So, you multiply the back prorated gradient with the local gradient to get what is del f del w. Now similarly over here I have to compute del f del v, but I cannot compute it directly because f is not visible to v. So, what I will compute is I will compute del f del u and then multiply with del u del v. So, that is what gives me the sensitivity of v, sensitivity of f with respect to v or the gradient of f with respect to v.

Again over here when I compute del f del x first I have to compute what is del f del v and then I have to compute what is del v del x. Similarly over here I have to compute to in order to find out del f del y I have to compute what is del f del v and then multiply that with del v del y right. So, here you find that your del f del u is equal to z and what is del u del w you come over here del u del w is equal to 1 right.

So, this will be simply the value of del f del u which is nothing, but z come over here what is del f del v? Del f del v is del f del u into del u del v and what is del f del u? Again del f del u is equal to z and what is del u del v right? Del u del v here you find that del u del v is again equal to 1. So, this value will simply be equal to z again. Coming over here del f del x equal to del f del v into del v del x and del v del x is nothing, but y.

So, del f del x will be y into z. So, this will be equal to y into z. Similarly here as del f del v is equal to z and del v del y del v del y is equal to x. So, this will be x into z. So, you find that finally, following this back propagation and the chain rule I find I can find out what is del f del x, I can find out what is del f del y, I can find out what is del f del z, I can also find out what is del f del w.

(Refer Slide Time: 21:27)



Back propagation: Pseudo Code

```
# Set Input
  w=5; x=3; y=2; z=4

# Forward Pass          # Backward Pass

  v = x*y                 dfdu = z
  u = w+v                 dfdz = u
  f = u+z                 dfdw = 1*dfdu  # dudw = 1
                          dfdv  = 1*dfdu  # dudv = 1
                          dfdx  = y*dfdv  # dvdx = y
                          dfdy  = x*dfdv  # dvdy = x
```

So, let us see that how we can implement this in modern programming languages. They I will just put this because this is the kind of structure or the constructs which are given by modern languages meant for deep learning or machine learning implementations. So, first what you do is, you set the input values. So, you find input values where said earlier was w is equal to 5 equal to x equal to 3, y is equal to 2 and z equal to 4, then what we have done is, we have computed the forward pass.

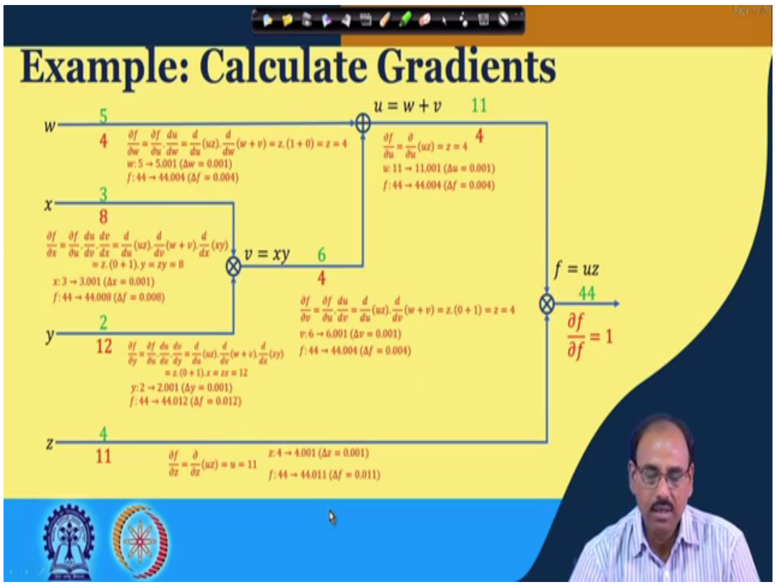
In the forward pass v is equal to x into y then u is equal to w plus v and then f is equal to u plus z. So, this is what we are computing in the forward pass and then if a backward pass. So, in the backward pass as we have seen earlier that you see what we have done in the backward pass. Firstly, we have to compute del f del u, I have to compute del f del z and then following the chain rule.

Finally I have to get what is del f del w what is del f del x del f del y and so on. So, in the same manner here you find out what is del f del u which is nothing, but z and del f del z which becomes equal to u and then del f del w is nothing, but del f del u into del u del w that is what I we had seen earlier and we have also seen that del u del w is equal to 1. So, del f del w simply becomes del f del u. In the same manner the del f del v also becomes del f del u. Del f del x becomes y times del f del v because del f del x is nothing, but del f del v into del v del x where del v del x is equal to y right. So, you simply come over here you can look over here that del v del x is equal to y. So, that gives me del f del x or df dx is equal to y times df dv and in the same manner df dy becomes x times df dv.

So, after this backward pass now we are ready with the sensitivity computation of sensitivity of the output with respect to the inputs to the note and the kind of graph that we have computed this is what is known as computational graph. So, all these which are shown in red over here they tell you that what is the sensitivity of f with respect to z, what is the sensitivity of f with respect to w, the sensitivity of f with respect to x and sensitivity of output with respect to y.

So, how does it help, why we are doing all this? One is we know that given a complicated function how it can be broken into simpler functions to give you a complete node function and then how we can write the back propagation algorithm as well as the feed forward algorithms using the modern and deep learning languages the programming deep learning tools that we have right.

(Refer Slide Time: 25:06)



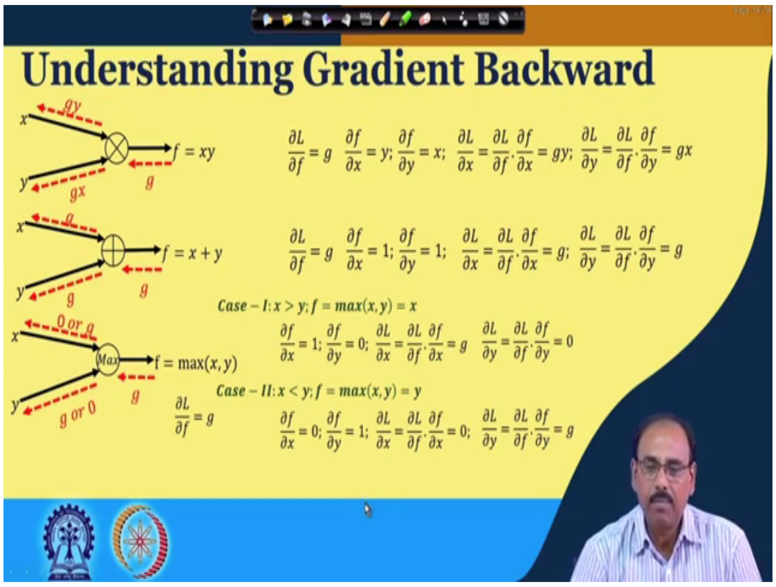
Example: Calculate Gradients

w
5
4
$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial u}.\frac{du}{dw} = \frac{d}{du}(uz).\frac{d}{dw}(w+v) = z.(1+0) = z = 4$
$w: 5 \rightarrow 5.001\ (\Delta w = 0.001)$
$f: 44 \rightarrow 44.004\ (\Delta f = 0.004)$
$u = w + v$
11
4
$\frac{\partial f}{\partial u} = \frac{\partial}{\partial u}(uz) = z = 4$
$u: 11 \rightarrow 11.001\ (\Delta u = 0.001)$
$f: 44 \rightarrow 44.004\ (\Delta f = 0.004)$
x
3
8
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}.\frac{du}{dv}.\frac{dv}{dx} = \frac{d}{du}(uz).\frac{d}{dv}(w+v).\frac{d}{dx}(xy)$
$= z.(0+1).y = zy = 8$
$x: 3 \rightarrow 3.001\ (\Delta x = 0.001)$
$f: 44 \rightarrow 44.008\ (\Delta f = 0.008)$
$v = xy$
6
4
$\frac{\partial f}{\partial v} = \frac{\partial f}{\partial u}.\frac{du}{dv} = \frac{d}{du}(uz).\frac{d}{dv}(w+v) = z.(0+1) = z = 4$
$v: 6 \rightarrow 6.001\ (\Delta v = 0.001)$
$f: 44 \rightarrow 44.004\ (\Delta f = 0.004)$
$f = uz$
44
$\frac{\partial f}{\partial f} = 1$
y
2
12
$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial u}.\frac{du}{dv}.\frac{dv}{dy} = \frac{d}{du}(uz).\frac{d}{dv}(w+v).\frac{d}{dx}(xy)$
$= z.(0+1).x = zx = 12$
$y: 2 \rightarrow 2.001\ (\Delta y = 0.001)$
$f: 44 \rightarrow 44.012\ (\Delta f = 0.012)$
z
4
11
$\frac{\partial f}{\partial z} = \frac{\partial}{\partial z}(uz) = u = 11$
$z: 4 \rightarrow 4.001\ (\Delta z = 0.001)$
$f: 44 \rightarrow 44.011\ (\Delta f = 0.011)$

So, this is simply same computation with the examples that we have shown earlier. So, here you find that your del f del w actually tells you that how f varies with w. So, if I increment w by its a small amount say for example, 0.001. So, we had fed input w to be 5, now we make w to be 5.005 and as a result you find that f has increased in initially value of f was 44. Now if has increased to 44.004 that says that by whichever amount we have incremented w f has been incremented by 4 times of that; that means, if I want to reduce w by varying reduce f by varying w then w has to be recriminated and the same thing becomes true with all x y and z.

And this graph is what is known as a computational graph and now you find that what is the use of the computational graph or how the computational graph helps you in writing such back propagation algorithms right.

(Refer Slide Time: 26:29)



Understanding Gradient Backward

$$\frac{\partial L}{\partial f} = g \quad \frac{\partial f}{\partial x} = y; \frac{\partial f}{\partial y} = x; \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial x} = gy; \quad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial y} = gx$$

$$\frac{\partial L}{\partial f} = g \quad \frac{\partial f}{\partial x} = 1; \frac{\partial f}{\partial y} = 1; \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial x} = g; \quad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial y} = g$$

$Case - I: x > y; f = max(x,y) = x$

$$\frac{\partial f}{\partial x} = 1; \frac{\partial f}{\partial y} = 0; \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial x} = g \quad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial y} = 0$$

$Case - II: x < y; f = max(x,y) = y$

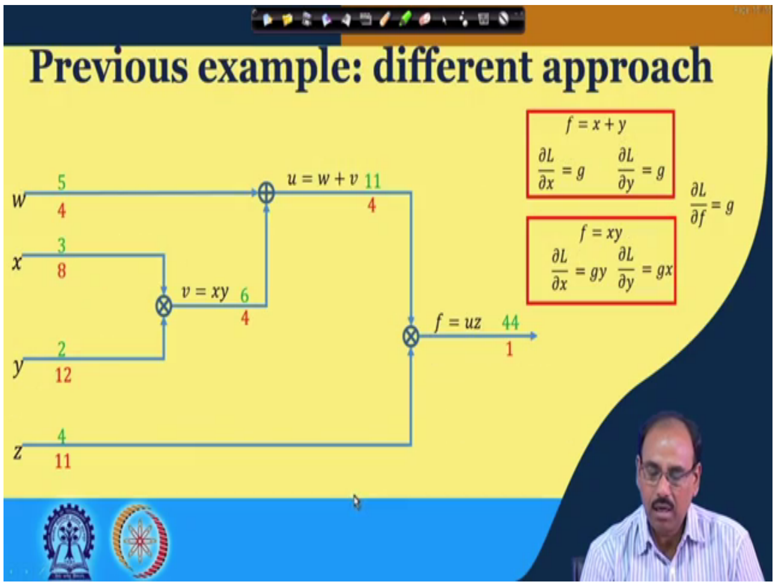
$$\frac{\partial L}{\partial f} = g$$

$$\frac{\partial f}{\partial x} = 0; \frac{\partial f}{\partial y} = 1; \quad \frac{\partial L}{\partial x} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial x} = 0; \quad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial y} = g$$

We have few more examples in the same manner, but I tell you that simple some simples that kids elementary circuits using which you form or you design the complicated nodes, one is add, one is multiply or another one is max. So, in the back propagation the multiply operation as we have seen before that if the output is say product x into y then in the back propagation if the gradient up to this output is g what you pass to the side of x is actually g into y and what you pass to the side of y is actually g into x.

Similarly, if it is summation node then whatever is the gradient at the output side it is simply copied to both the inputs. Similarly, if it is max then whatever is the gradient at the output the gradient is simply passed to the node of the variable which has maximum value. So, if x was maximum of the 2 x and y gradient g was passed towards x if y was maximum then gradient g would have to be passed to y and using this we can have we can represent complicated node functions using simpler partial differentiation or gradient operations and that helps us to write the functions in a much simpler way and that helps ultimately helps us to code the complicated functions.

(Refer Slide Time: 28:16)



Previous example: different approach

$f = x + y$

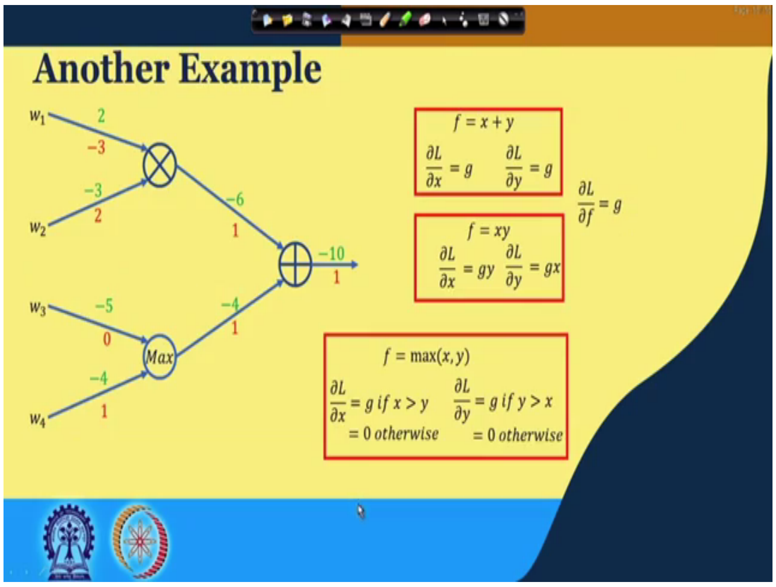
$\frac{\partial L}{\partial x} = g \quad \frac{\partial L}{\partial y} = g$

$f = xy$

$\frac{\partial L}{\partial x} = gy \quad \frac{\partial L}{\partial y} = gx$

$\frac{\partial L}{\partial f} = g$

(Refer Slide Time: 28:18)



Another Example

$f = x + y$

$\frac{\partial L}{\partial x} = g \quad \frac{\partial L}{\partial y} = g$

$f = xy$

$\frac{\partial L}{\partial x} = gy \quad \frac{\partial L}{\partial y} = gx$

$\frac{\partial L}{\partial f} = g$

$f = \max(x, y)$

$\frac{\partial L}{\partial x} = g \text{ if } x > y$, $= 0$ otherwise $\quad \frac{\partial L}{\partial y} = g \text{ if } y > x$, $= 0$ otherwise

I will, so this is the same thing I will just skip that, this is again an example using that multiply add and max operation and we had you can easily verify that how the forward pass information is propagated from input to output to get the final output and then in the backward pass we can compute the gradients following the same gradient descent procedure.

And I want to emphasize one more point over here that in every case the gradient computation at a node is highly local in nature as long as your gradient till the previous node, previous layer is computed and that is fed to the current layer. In the current layer you can make use of the gradient which has been propagated to current layer and all the computations in the current layer are highly parallel in nature. The computation of gradient of one node in the current layer is independent of the gradient computation in the other node.

So, as a result this inter function can be implemented in parallel machine. If I have a parallel machine then all the gradient computations of the back propagation learning can be implemented using parallel operations. So, I will stop here today and as we said now onwards I will not consider the gradient learning anymore or the gradient propagation or back propagation learning anymore. For any application subsequently I will simply use the term that the machine is learnt or trained using back propagation algorithm ok.

Thank you.