# Deep Learning

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

## Lecture – 40
## Vanishing and Exploding Gradient

Hello, welcome to the NPTEL online certification course on Deep Learning. So, we are discussing about some popular CNN architectures, in our previous few classes we have discussed about two different CNN models – one of them was AlexNet and the other one was VGG Net.

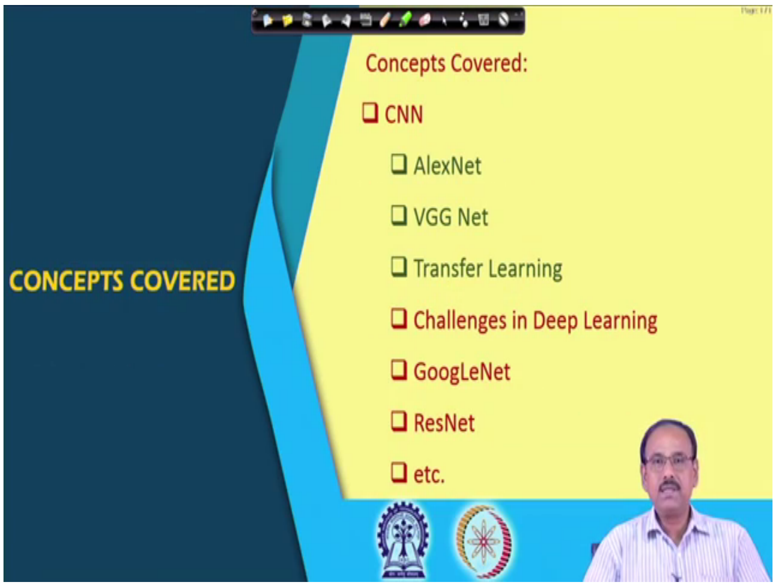
And, then also we have discussed about a concept which is widely used in the deep neural network domain that is the concept of a transfer learning. The concept is that if you have a network which is learnt or which is trained for a particular application domain then the knowledge gained or the knowledge that you have gained during the training of the network in some domain then this knowledge can be reused in some other domain.

And, that is possible because the features which are learned by the network in the earlier layers of the CNN network these features are mostly generic in nature. Say, for example, if I consider my domain of application is image recognition, then in some cases we have images of say animals and in some other application we have images of cars. In both the cases the features which are learnt in the first convolution layer are the edges; the edges may be vertical, the edges may be horizontal, the edges may be diagonal and so on.

And, these are the edges which are present in both the domains whether my image comes from the animal database or the image comes from the car database. So, while training my network onto the animal database the first layer whatever it has learnt that is the features which are different types of edges vertical, horizontal, diagonal and so on because they are also applicable in case of other database that is in the database of cars. So, the knowledge which are gained in this first layer can be reused in the second application as well. So, that need not be retrained.

So, if you look at that way in that case you can easily find that the knowledge which are gained in the early layers of the CNN can be reused in any other application. But, the knowledge which are gained or in the later layers of the neural network those knowledge is mostly domain specific. So, that has to be fine tuned for your other application whereas, the knowledge gained in the previous application can be reused. And, that is the concept of transfer learning and we have also talked about transfer learning in our previous lecture.

(Refer Slide Time: 03:29)



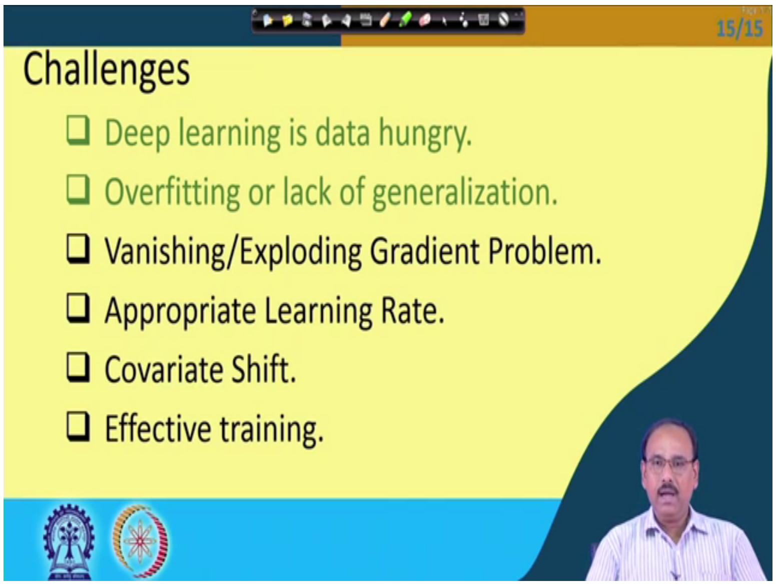
CONCEPTS COVERED

Concepts Covered:

- CNN
  - AlexNet
  - VGG Net
  - Transfer Learning
  - Challenges in Deep Learning
  - GoogLeNet
  - ResNet
  - etc.

So, in today's lecture let us first with the discussion on some of the challenges that are faced in deep learning or while training the deep neural network. And, then we will discuss about the other networks and while discussing them we will also see that how such challenge or some of these challenges are addressed in the networks like GoogleNet, ResNet and so on.

(Refer Slide Time: 04:09)


15/15

## Challenges

- ❑ Deep learning is data hungry.
- ❑ Overfitting or lack of generalization.
- ❑ Vanishing/Exploding Gradient Problem.
- ❑ Appropriate Learning Rate.
- ❑ Covariate Shift.
- ❑ Effective training.

So, first let us see that what are the challenges in deep learning or deep learning challenges. So, one of the challenge is deep learning is data hungry; that means, if I train a deep neural network I need thousands or even millions of data, millions of annotated data for proper training of deep neural network. So, as we have seen earlier that this has been done by data augmentation. This was the thing that we have seen earlier. So, how do you augment data? By taking different crops of or different parts of your input data set and multiplying the input data by taking crops from different regions.

There also you can do different types of transformations of the images which are there in the training data set. The transformations can be reflection, the transformations can be intensity variation, the transformations can be color distortion and so on. So, by applying all these different types of operations on your input data set or training data set you can multiply the size of your training data set and that is how even if I have limited data set for training I can enlarge the data set which will be actually used for training purpose.

The other problem or other challenge that you have in deep neural network is that as the number of parameters in the deep neural network is quite high so, it is quite likely that the network will simply try to memorize the input pattern which were presented at the time of training. And, as it tries to memorize it does not it may not really learn the representation of the data right. So, as a result your error during the training or the training error may be quite low, but when you actually apply your trained model to real applications in that case the error that is generated may be quite high and that is what is known as generalization error.

And, a problem that you face is known as over overfitting problem and this overfitting problem is due to lack of generalization. So, we have also seen when we discussed earlier say for example, when I discussed our auto encoder. We have seen that auto encoder has a layer known as bottleneck layer and the purpose of this bottleneck layer is when you are training the auto encoder, then the information flows through the on auto encoder through a restriction through a constriction as a result the inter information does not flow from the input to output and that prevents the auto encoder from memorizing your input pattern or input data.

So, auto encoder in such cases are forced to learn the features the important features of the input data and using that important features it should be able to reconstruct the input data. Similarly, we have. So, this is one of the way in which the overfitting can be avoided or the network is forced to learn the representation the network simply does not simply try to memorize the input data which is presented at the time of training.

The other kind of approaches that we have seen is by imparting some sort of constants a activation constant that also we have seen when we have discussed about training of the auto encoders. In the same manner there are various other ways in which this overfitting problem can be avoided. During training we can feed in the noisy data. So, from the noisy data the network actually turns to tries to learn the many fold right. So, that is another way of limiting the overfitting problem.

Similarly, in one of the previous lectures we have also discussed about a concept called a dropout. So, in case of dropout what you do is you choose some nodes in your network at random during the training time and those nodes are simply omitted from the network. So, as a result the network becomes more robust and as few of the nodes are omitted at random with certain probability, the other nodes in the network they have to take the responsibility of learning in absence of those nodes which are omitted or which had dropped.

So, as a result your network becomes more robust. It learns the generalized features and we can prevent the network from memorizing the input data. The other problem which is quite common in deep neural network and in fact, this increases with the depth of the network which is the vanishing or exploding gradient problem. So, as you have seen that during training what we use is the gradient descent approach and the information flows in the backward direction from output to input all the error information and while this information flows in the backward direction in every layer the parameters or weights of the network they are tuned or they are trained.

And, as the information flows from output to input side with depth of the network the gradient of the error which is flown which flows from the output to the input side that goes on reducing and at a certain time that may even vanish. We will discuss about this vanishing or exploding gradient problem a bit later.

Then, the other challenge is the appropriate learning rate. We have said before that in every back propagation algorithm or gradient descent algorithm, you update your weights by an amount which is proportional to the gradient and while doing so, we have mentioned that there is an learning rate which is we have represented by a symbol say eta. So, if this learning rate is very high; that means, the parameters are updated quite fast or the step that you take for updation of the parameters, the step size becomes very high.

And, as a result there is a risk of missing the minima and you can overshoot the minima go to the other side. So, as a result you will have a number of oscillations before you can actually come to the minimum point of the loss or it may also possible that the system may actually diverge instead of converging to the minimum it will diverge. So, choice of appropriate learning rate is another challenge in deep learning in training of the deep neural network.

The other challenge is covariate shift. What is this covariate shift? You find that when we talked about the back learning algorithm or gradient descent algorithm, usually the kind of gradient descent version which is used is what is known as batch gradient descent or mini batch gradient descent, where you take a mini batch of the set of samples or the set of data which are given for the training purpose and you perform your gradient descent operation on the with the samples available in the mini batch ok.

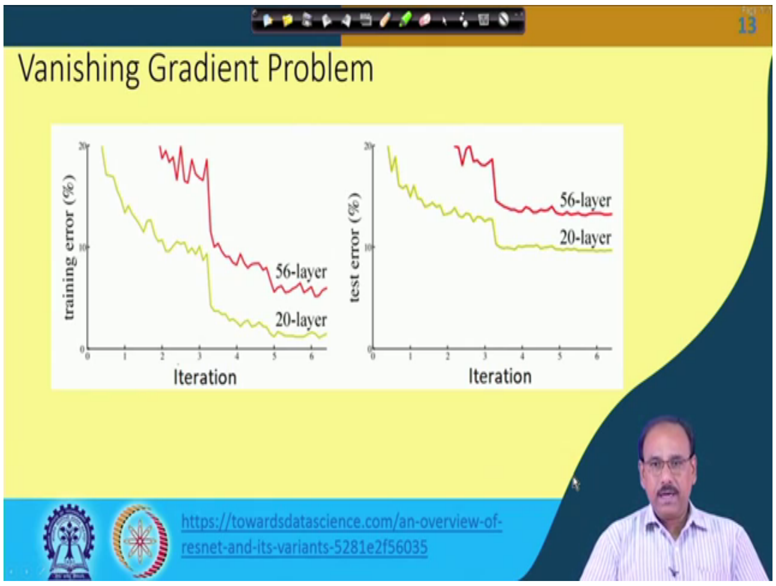
And, now when you shift from one mini batch to another mini batch the feature distribution of one mini batch may be widely different from the feature distribution from another mini batch, even if your objects or the targets remains the same. So, for example, if I want to distinguish between cat and say flower; in one mini batch I have the images of cats which are black and white and in another mini batch I have images of cats which are colored.

So, when you extend the features, when the convolutional neural network extract the features then the feature distribution with black images is different from the feature distributions with color images. So, though your objects remains the same that is in both the cases your database contains images of cats, but the distribution of features are widely different. And, as a result it becomes for the neural network to settle on to a set of parameters or it becomes difficult for the neural network to learn the features because the feature distribution from batch to batch changes.

So, we will also discuss about in our later lectures that how to take care of such covariate shift. And, similarly the other challenge that is quite common in while training the deep neural network is what is effective training. That means, how we can make the training operation of the neural network more efficient, when should we decide that the training should be stopped or how we can make the training faster. So, these are the various challenges in development of deep neural network or deep algorithms.

So, out of these two deep learning, I mean replication of data or augmentation of the training data and as well as the overfitting problems these are the things which we have already discussed in few of our previous lectures. Now, let us see what is this vanishing or exploding gradient problem before, we go for discussion of other neural network models like GoogleNet or ResNet or residual neural network.

(Refer Slide Time: 14:39)



## Vanishing Gradient Problem

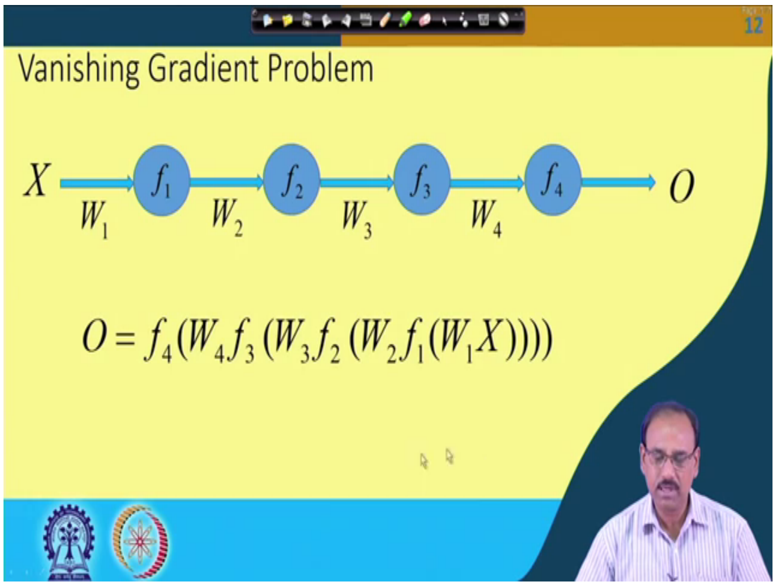
https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

So, what is this vanishing gradient or exploding gradient problem? So, it is expected that or it was believed that as the depth of the neural network increases, the capacity of the neural network also increases and the neural network learns more and more features and more and more representations. So, as a result the performance of the neural network should increase with the number of layers that you go on adding in your neural network or with the depth of the neural network the performance of the neural network should increase. But, the experiments have shown that it is not always true.

Say for example, here as we have shown in these two images that the test error and the training error that may in fact, increase as you increase the number of layers in your deep neural network. Say for example, here you find that with a network having 20 layers and with a network having 56 layers, during training operation though with number of iterations the error goes on reducing there is a trend of reducing error with the number of iterations, but you will find that as the number of layers is increased from 20 to 56 the training error has also increased substantially.

And, the same is also true with test error. When you test your train network, then also though with iterations the train say is that the test error goes on reducing, but as you have increased the number of layers from 20 to 20 fifth from 20 to 56 the test error has also increased. So, what is the problem? What is the lacuna in it? So, one of the reason has been attributed to the problem of vanishing gradient or exploding radiant.

(Refer Slide Time: 16:41)



## Vanishing Gradient Problem

$$\begin{tikzcd} X \arrow[r, "W_1"'] & f_1 \arrow[r, "W_2"'] & f_2 \arrow[r, "W_3"'] & f_3 \arrow[r, "W_4"'] & f_4 \arrow[r] & O \end{tikzcd}$$

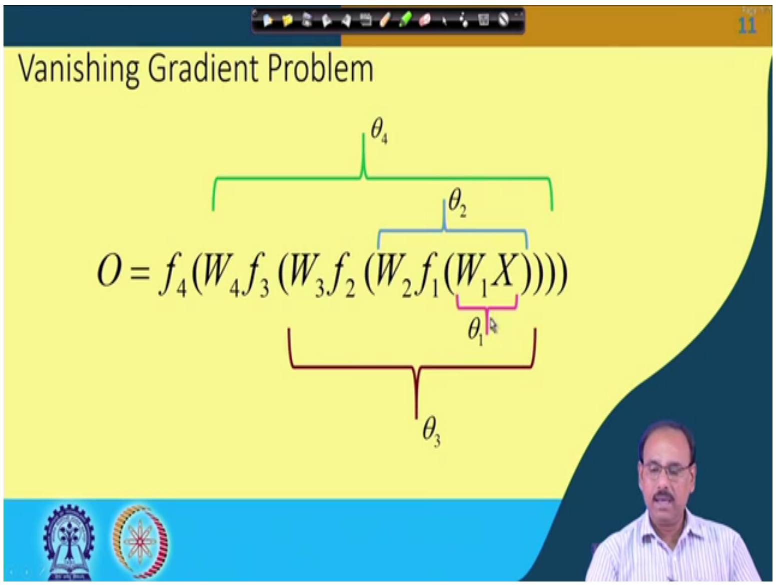
$$O = f_4(W_4 f_3(W_3 f_2(W_2 f_1(W_1 X))))$$

Let us see what is this vanishing gradient or exploding gradient. So, for expand this to explain this I have taken a simple network architecture having four different layers. So, this network having four different layers the first layer has an activation of f 1, the second layer has an activation f 2, third layer has an activation of f 3 and the fourth layer has an activation of f 4.

Similarly, the connection weights to the first layer it is W 1, weight between the first layer and the second layer is W 2, weight between the second layer and the third layer is W 3 and wait between the third layer and the fourth layer is W 4. And, finally, I have output from the fourth layer nodes. For simplicity we have assumed that all these are scalars; that means, W 1 is a scalar instead of a vector, W 2 also is a scalar, W 3 is a scalar, W 4 is a scalar and so on and this functions f 1, f 2, f 3, f 4 they are also scalar functions; that means, the output activation that you get is also a scalar value.

So, given this type of architecture you find that the output of this network which is now a function of the input X and all these weights W 1 to W 4. This I can write as output as f 4 of W 4 into f 3 of W 3 into f 2 of W 2 into f 1 of W 1 into x. So, this is your final output and this nature of the output function we have also seen we have also discussed when we discussed in our earlier lectures on the neural network.

(Refer Slide Time: 18:35)

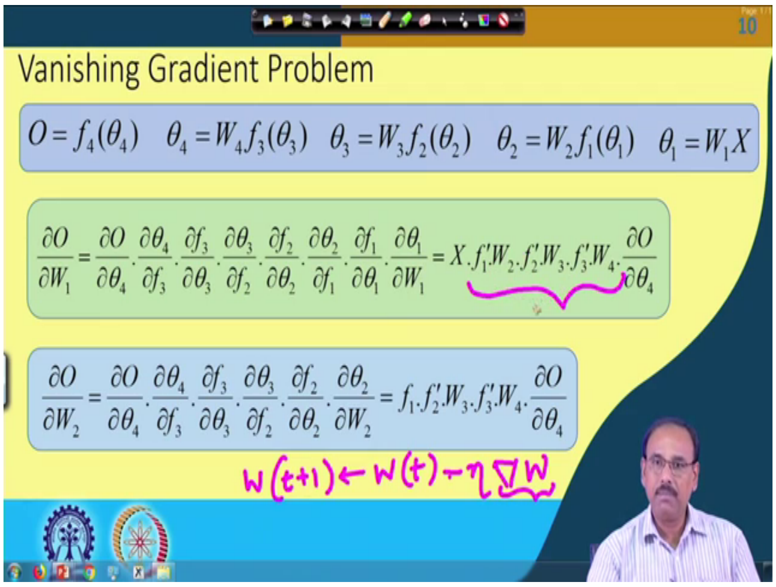


## Vanishing Gradient Problem

$$O = f_4(\underbrace{W_4 f_3(\overbrace{W_3 f_2(\underbrace{W_2 f_1(\underbrace{W_1 X}_{\theta_1})}_{\theta_3})}^{\theta_2})}^{\theta_4})$$

So, for mathematical convenience I put it this way: W 1 into X I put it as argument theta 1, W 2 into f 1 of W 1 into ax I put it as argument theta 2, then W 3 into f 2 of theta 2 I put it as argument theta 3 and then f 4 into f 3 of theta 3 I put it as argument theta 4. So, this is simply put as for mathematical convenience.

(Refer Slide Time: 19:07)



## Vanishing Gradient Problem

$$O = f_4(\theta_4) \quad \theta_4 = W_4 f_3(\theta_3) \quad \theta_3 = W_3 f_2(\theta_2) \quad \theta_2 = W_2 f_1(\theta_1) \quad \theta_1 = W_1 X$$

$$\frac{\partial O}{\partial W_1} = \frac{\partial O}{\partial \theta_4} \cdot \frac{\partial \theta_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial \theta_3} \cdot \frac{\partial \theta_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial \theta_2} \cdot \frac{\partial \theta_2}{\partial f_1} \cdot \frac{\partial f_1}{\partial \theta_1} \cdot \frac{\partial \theta_1}{\partial W_1} = X . \underbrace{f_1' W_2 . f_2' W_3 . f_3' W_4 .}\frac{\partial O}{\partial \theta_4}$$

$$\frac{\partial O}{\partial W_2} = \frac{\partial O}{\partial \theta_4} \cdot \frac{\partial \theta_4}{\partial f_3} \cdot \frac{\partial f_3}{\partial \theta_3} \cdot \frac{\partial \theta_3}{\partial f_2} \cdot \frac{\partial f_2}{\partial \theta_2} \cdot \frac{\partial \theta_2}{\partial W_2} = f_1 . f_2' W_3 . f_3' W_4 . \frac{\partial O}{\partial \theta_4}$$

$$W(t+1) \leftarrow W(t) - \eta \underbrace{\nabla W}$$

And, accordingly the definitions of this theta arguments and the output will be like this. Now, my output O becomes f 4 of theta 4. So, you just look over here the argument of f 4 is defined as theta 4. So, the final output that I get which is f 4 of theta 4. Similarly, theta

4 is f 3 W 4 into f 3 of theta 3; theta 3 is W 3 into f 2 of theta 2; theta 2 is W 2 into f 1 of theta one and theta 1 is W 1 into X.

Now, when I do the back propagation learning or for doing that what I do is I have to perform the gradient descent operation. Suppose, I wanted to update that weigh to W 1; so, in order to do this I have to minimize or I had to take the gradient step that minimizes the output error with respect to weight W 1. And, you remember that when we derived this relation when we derived this equation we had a term del of O with del of W 1 or del O del W 1.

And, now applying chain rule if I try to find out here that what is del O W del W 1 you find that the expression will come out to be X into f 1 dash, where f 1 dash is the derivative of f 1 into W 2 into f 2 dash or f 2 dash is again the derivative of f 2 into W 3 into f 3 dash into W 4 into del O del theta 4, where del or del theta 4 is the gradient of output O with respect to theta 4. Similarly, if I take del O del O del W 2 that is the gradient of output with respect to W 2 which will be useful when you try to update the weight W 2 and that you see over here this is product of f 1 f 2 dash W 3 f 3 dash W 4 into del O del theta 4.

So, just by looking at this you find that as you move towards the earlier layers for updating the weights in the earlier layers your number of terms in this product that goes on increasing and this product terms the number of terms in the product is the sequence of W's of the subsequent layers and the derivatives of the activations of the nodes of the subsequent layers and that is what you get over here.

Now, come to your situation, that if I assume that my activation function in every layer that is f 1, f 2, f 3, f 4 and so on they are sigmoidal function. So, if I use the activations to be sigmoidal functions then you remember that the derivative of a sigmoidal function if my if I represent my sigmoidal function as f then f prime is nothing, but f into 1 minus f and the value of this will be maximum when the argument is 0; that means, if my function is f X f dash X will be maximum at X equal to 0 and this maximum value is 1 by 4, you can just compute this and verify.

So, with sigmoidal functions this f 1 dash f 2 dash f 3 dash all of them will be 1 by 4 maximum, it may even be lesser than that. And, as we said that to initialize your network before you start training the weights are initialized at random and typically when you initialize the weights the weight values are from a distribution with mean 0 and the standard deviation of the variance equal to 1. So, all of these terms within this product, they are less than 1. So, as a result when you take the product of all of them the product will be even lesser and in fact, this product value goes on reducing exponentially.

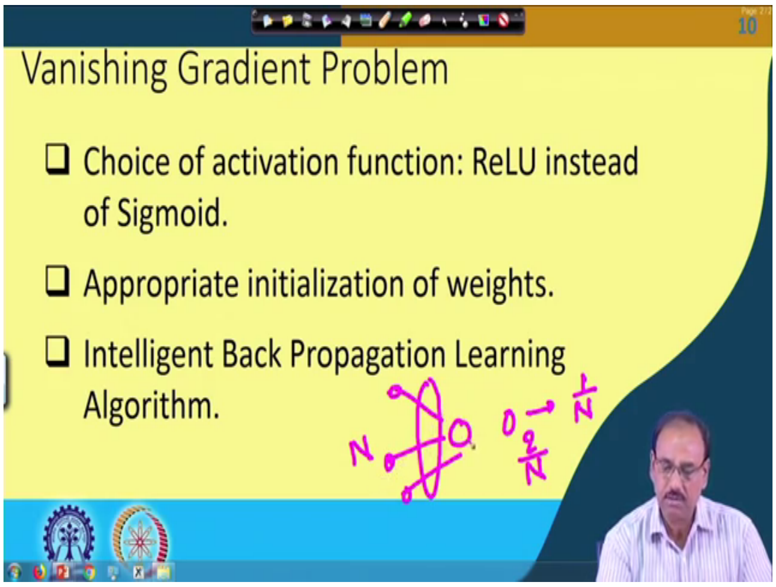
So, as a result as the number of terms in this product goes on increasing that is as we are moving towards the earlier layers of your deep neural network, the gradient term which reaches there almost vanishes because this gradient term or the product term goes on reducing exponentially. And as the gradient almost vanishes so, when I go for updation by the equation; so, what was our updation equation? Updation equation was a W t plus 1 was W t that is your parameter value at the previous parameter value minus eta times grad of W.

And, this term grad of W goes on using this reduces exponentially as you move towards the earlier layers and when this almost becomes 0, your W t plus 1 and W t it remains more or less same; that means, you hardly update your weight and this is what is the problem of vanishing gradient right. So, and this problem increases it becomes very very prominent with the depth of the network because as the depth of the network increases the terms in this product that also goes on increasing; reducing which reduces the gradient which reaches the earlier early layers of the network and as a result the training becomes more or less ineffective or there is no updation of the weights.

And, here we have shown these two expression just to emphasize that as we move towards the earlier layer the problem becomes more and more prominent. And, this is what is your vanishing gradient problem. Just the reverse is the exploding radiant problem you think of the situation that instead of every term in this product being less than 0, if they happen to be greater than 0. In that case that gradient will explode exponentially right that also you can do numerically.

So, if the gradient explodes, then when you are trying to update your weight the updation step will be very large indicating that it will lead to oscillation or even it may lead to divergence instead of converging your algorithm will actually diverge. So, these are the problems of vanishing gradient and exploding gradient. And, so, obviously, unless we take some measure to restrict or to arrest this vanishing or gradient vanishing or exploding gradient problem, naturally the network will not perform well. So, one of the ways in which this can be arrested is proper choice of your activation function.

(Refer Slide Time: 26:53)



## Vanishing Gradient Problem

- ❑ Choice of activation function: ReLU instead of Sigmoid.
- ❑ Appropriate initialization of weights.
- ❑ Intelligent Back Propagation Learning Algorithm.

So, as we have seen that in case of activation function if we use the sigmoidal activation function it is gradient is always than 1, maximum value of the gradient is 1 by 4. So, instead of sigmoidal activation function you can use the activation function ReLU or rectified linear unit for which the gradient is 1. And, that is the reason that ReLU has become very popular in modern neural networks than activation functions like sigmoidal or tan hyperbolic.

The other thing is by appropriate choice of the weight if the weight becomes less than 1, then we have vanishing gradient problem if it is greater than 1, we have exploding radiant problem or in other words your algorithm becomes unstable or it oscillates ok. So, when you choose the weights initially which are chosen at random with mean 0 and the standard deviation or the variance equal to 1, how do you choose the weights initialize the weights in a larger network?

So, a thumb rule says that if in any layer I have a node and the number of nodes from which this particular node is getting the input that is say N then while initializing these weights these weights should be initialized with mean 0, but the variance now should be 1 by N or in many cases they use the variance to be 2 by N. So, that is how I have to choose the weights at random and those randomly chosen weights are to be assigned to these weights from the previous layer to this layer.

So, weight initialization is another determining factor which can tackle or which can address the problem of vanishing gradient or exploding gradient. And, the other approach is obviously, by using intelligent back propagation learning algorithm or by carefully or intelligently designing your neural network.

So, with this we conclude our lecture today in our next lecture we will talk about the GoogleNet and the resNet. And, we will see that what measures GoogleNet or the resNet has taken to address this vanishing gradient problem.

Thank you.