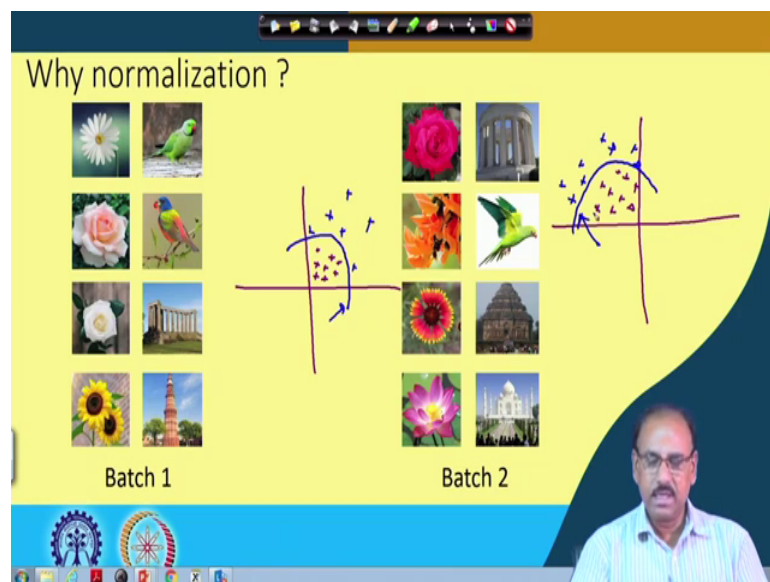


Deep Learning
Prof. Prabir Kumar Biswas
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture - 47
Batch Normalization- I

Hello, welcome back to the NPTEL online certification course on Deep Learning. Since our previous class, we are discussing about the normalization and in the last class we have discussed that what is the role of normalization, what we need normalization techniques, so that your classifier design can be faster or the training of the classifier can be faster and the classifier can be more stable.

(Refer Slide Time: 01:58)



Particularly, we have taken an example with the help of batch normalization techniques or why we are going for batch normalization is; the classifier training is or the learning process is in the form of batches. So, the algorithm which is mostly used is what is known as batch gradient descent or batch stochastic gradient descent approach.

So, where for training of the classifier or training of the deep neural network, you feed your training data into many batches where every batch contains may be say 10 training examples, 15 training examples, 100 training examples and so on that depends upon whether your batch size will be small or batch size is large, ok.

So, we are assuming that we are given these two different batches and the problem that we want to do address, so the classifier if classifies the flowers category from non-flower category. So, when you give the batches, the batches, every batch will contain some images from the flower some images from non-flower. So, these are the two different cases that have been shown here. On the left hand side we have batch 1 and on the, right hand side we have batch 2.

So, you find that the images of the flowers which are in the first batch they are not rich in colour, they are mostly whitish whereas, images of the flowers which are in the second batch or batch 2 they are very very rich in colour. So, as a result when you try to extract the features out of these flowers which are whitish in batch 1 and which are very very colourful in batch 2 the distribution of features of the flowers in batch 1 and the distribution of features of the flowers in batch 2 they are likely to be different.

So, as a result I can have a situation like this. So, in the first case, the flowers we can have the feature distribution something like this, whereas, non-flowers could have a distribution of features which are like this. So, these are the distribution features in the non-flower category. As a result you find that your classifier boundary will be something like this.

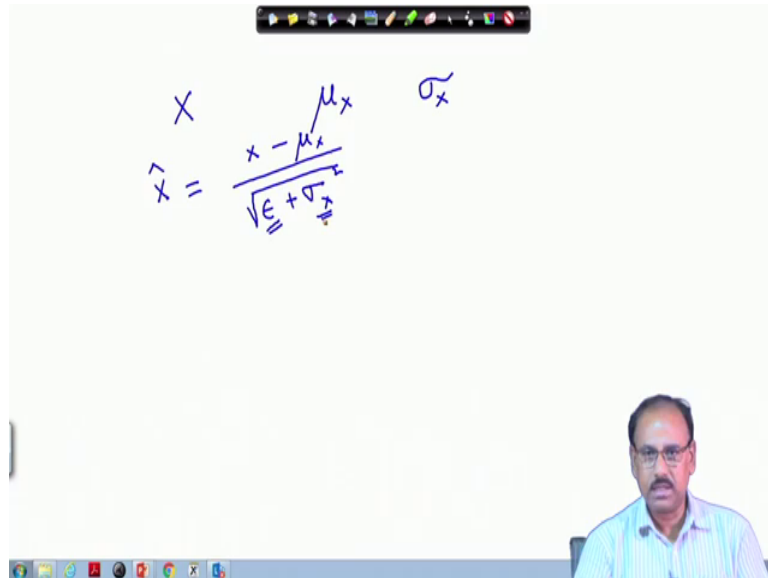
Whereas, in the second case again because now the flowers are more colourful, so it is quite possible that the distribution of features for the images belonging to flowers will be something like this whereas, the distribution of features of the images which would do not belong to the flower category may be something like this. So, as a result of that your classification boundary will be something like this.

So, now we find that though your images belonging to the same category of flowers, but because of their appearance the computed features may have different distribution. And as a result while training the classifier simply hops from one boundary to another boundary. In some cases it will decide this boundary, in some cases it will decide this boundary. So, as a result the time taken to train the classifier or the time taken to train your deep neural network becomes very large.

So, this can be avoided if we can somehow normalize the feature vectors, so that the distribution of all the feature vectors will belonging to the same class will be more or less

than. And the kind of normalization that we can apply in this case is a type of normalization that we have already discussed.

(Refer Slide Time: 05:09)

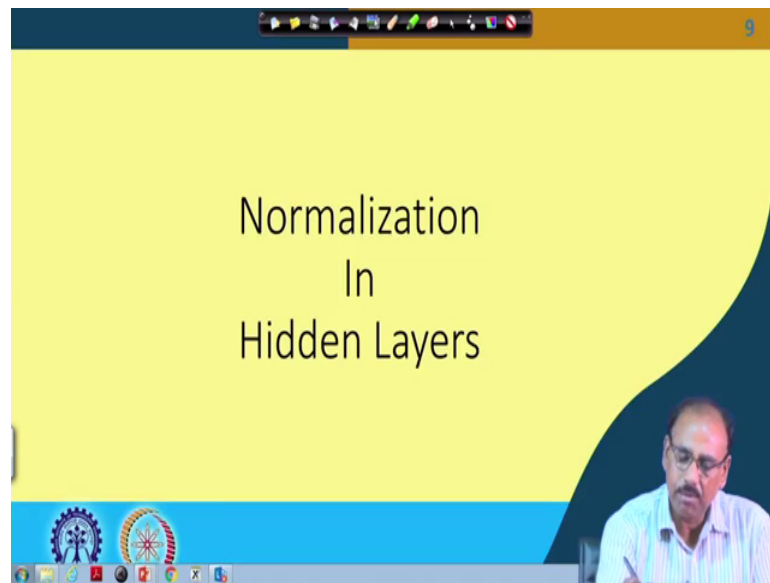

$$\hat{X} = \frac{x - \mu_x}{\sqrt{\epsilon + \sigma_x^2}}$$

Say for example, for every feature or the set of feature vectors belonging to a particular batch this is a set of feature vectors X , I can compute the mean of the feature vectors which is μ_X , I can also compute the standard deviation of the feature vectors which is σ_X , and then I can normalize this X as \hat{X} is equal to X minus μ_X upon square root of some epsilon plus σ_X square.

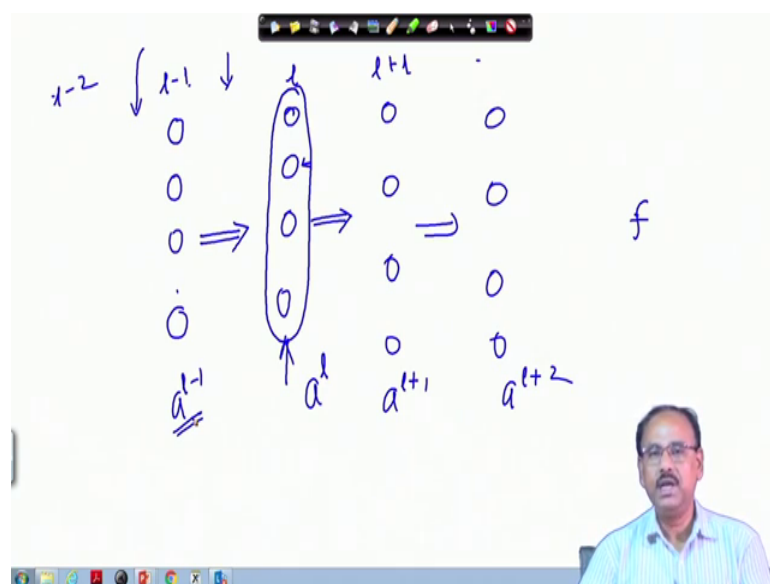
And this we have already discussed before that this epsilon is a very small positive quantity, this ensures that if this σ_X somehow becomes 0 I do not have a situation of a unstable division; that means, I do not get a case of division by 0. So, this is a sort of normalization that can be used. Now, you find that this sort of normalization what we have discussed is with respect to the input because we are feeding the images to the input layer.

This is applicable not only in the input layer this is also applicable in the hidden layers as well. Why should it be applicable to hidden layers? So, now, let us try to see that why do we need normalization even in hidden layers. So, for to discuss about that to see why you need normalization in the hidden layers, let us look at a typical architecture of a deep neural network.

(Refer Slide Time: 06:25)



(Refer Slide Time: 06:51)



So, the architecture of a deep neural network as we have already seen that it will be something like this that I have a number of layers, in every layer I have a number of neurons which is like this, right and from every layer to every other layer I have a set of parameters or set of weights. So, if this is my l -th layer this is my l minus first layer, this is l plus first layer and so on.

So, from every layer say l minus first layer the activations are fed to the l -th layer through a set of parameters or through a set of weights. So, every node in the l -th layer

gets an weighted sum of the activations of the previous layer and then on this weighted sum it performs another non-linear operation and the kind of non-linear operation function that we are considering in this case is ReLU non-linear function.

So, at the l -th layer you get an activation vector let me call it an activation vector say a_l , where it will be a_{l1} from the first neuron a_{l2} from the second neuron and so on. So, those are the components of this activation vector. And this activation vector is fed to the next layer to the that is l plus first layer that computes the activation l plus 1, which is fed to layer say l plus 2 which computes a_{l+2} and so on.

Now, during training at the final output I have some activations say O at the final layer, say O_f , so this is the final output that I get from the output layer, ok. To make it clear instead of O , let me use the term say my final output will be say some f at the final layer. And during back propagation what you do is you compute your error at the output and then following the gradient descent procedure in the backward pass you pass that gradient to the layers from the output side to the input side and while doing so, in every layer you go on updating the weight vectors or updating the parameter vectors. That is what you do.

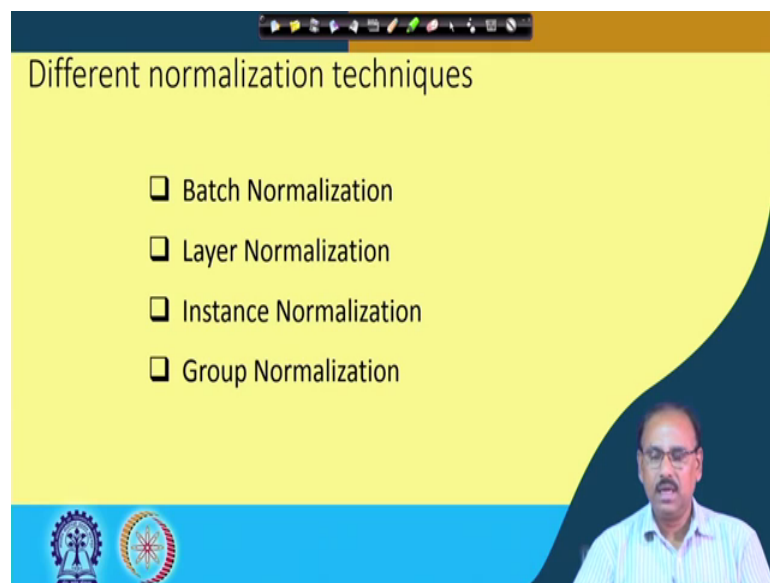
Now, when you come to the updation of or say come to this layer l only, you find that l the layer l gets activations from layer l minus 1 which are say activations a_{l-1} . And based on the distribution of a_{l-1} you adjust these weight vectors of layer l . Now, we find that if a_{l-1} is steady; that means, in every epoch the distribution of a_{l-1} remains the same, then learning of the layer l will not be any will not be a problem because the distribution of a_{l-1} which is coming from l minus first layer that remains the same.

But what happens? That during this training process this layer a_{l-1} is also updating its weights; that means, the weight vectors from layer l minus 2 to layer l minus 1 that is also being updated. The weight vectors from layer l minus 1 to layer l they are also being updated. So, as a result this a_{l-1} , the distribution of this may not remain same over the epochs. So, even if you are feeding the same input in the same batch the distribution of a_{l-1} the features which are computed at a_{l-1} may be different in different epochs. So, leading to the same problem of covariate shaped.

So, even here, even in the hidden layers or the internal layers I have to take some action so that this covariate shaped can be minimized. So, for minimization of this covariate shaped as we have seen before that I have to go for normalization of these weight vectors and in most of the cases what is tried is that you remap this vectors in such a way that the mean of all these feature vectors become 0 with a variance or standard deviation equal to 1.

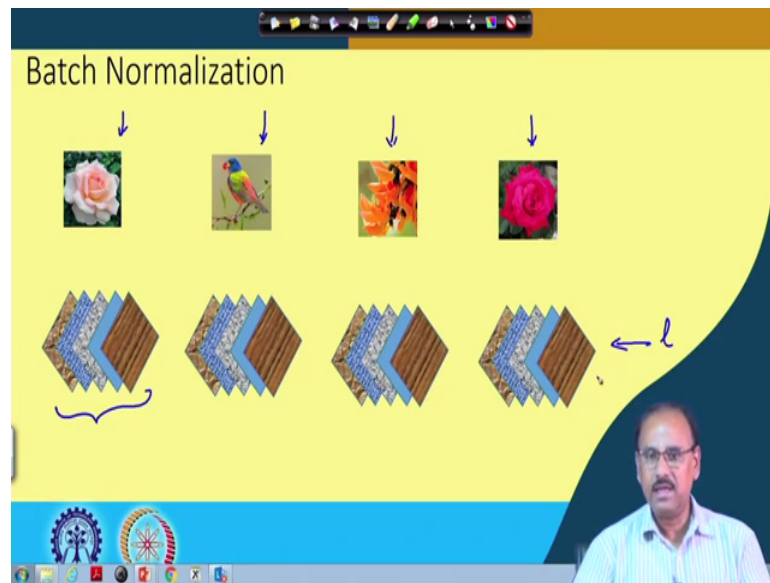
So, with this introduction that I need normalization not only at the input layer on the raw data, I also need normalization even at the hidden layer because the features which are computed from different layers even for the same input over different epoch or different training instances the distribution of those feature vectors may be different which I had to take some measure to erased that shape of the distribution. So, this is how the covariate shaped the problem of covariate shaped has to be addressed.

(Refer Slide Time: 12:19)



So, let us see that how we can do it. So, there are different ways in which this normalization can be done, one of the technique is what is known as batch normalization. You can also have what is known as layer normalization, you can have another technique known as instance normalization or you can also have a group normalization techniques. So, these are the different variants of the normalization techniques and we will see a bit later that the main difference is the way you compute the mean and standard deviation. So, first let us discuss about what is batch normalization.

(Refer Slide Time: 12:57)



So, in case of batches as we said earlier that you are feeding the inputs in batches, right you have the inputs from both the positive class as well as negative classes. And in every layer, so let us assume that we are now talking about some layer say l -th layer. In l -th layer, we have assuming that this is a convolution neural network, in l -th layer I have a number of convolution kernels.

So, depending upon the number of convolution kernels there will be different channels of features which are computed and all these feature channels when you concatenate together that becomes a feature map and this feature map is now then fed to the next layer during the forward pass. So, let us assume that in a particular batch, so these are the different examples, training examples which are provided in a particular batch for training of the deep neural network.

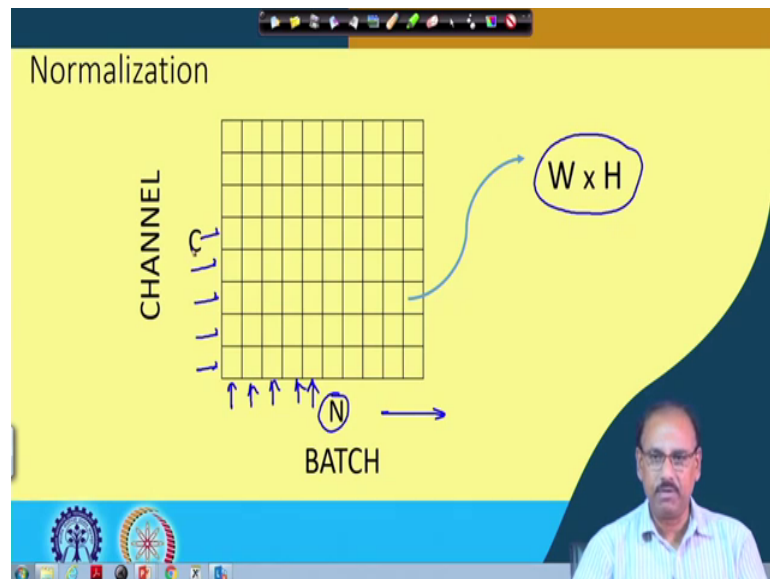
And in the n -th layer every convolution kernel forms a feature and let us assume that the number of convolution kernels in the l -th layer that we have is 5, so as a result I have 5 different feature components. Each of these feature components are of size say W by H or W is the width of the feature map and H is the height of the feature map.

So, finally, the feature map which goes to the next layer or l plus first layer that becomes a tensor. This tensor have 5 channels which is same as the number of convolution kernels that I have and the size of each of these channels depends upon how you compute the

convolution, whether it is with stride or without stride and how do you perform the pooling operation, ok.

As we said that pooling is nothing, but pooling eventually gives you a sort of dimensionality reduction or data reduction of the features that you compute. So, assuming over here that a particular batch has got say this 4 different images, training images, I have these 4 different feature maps, where every feature map has got certain number of channels and every channel has certain width and certain height. So, this is the kind of feature maps that you get which are to be fed to the layer l plus 1.

(Refer Slide Time: 15:50)

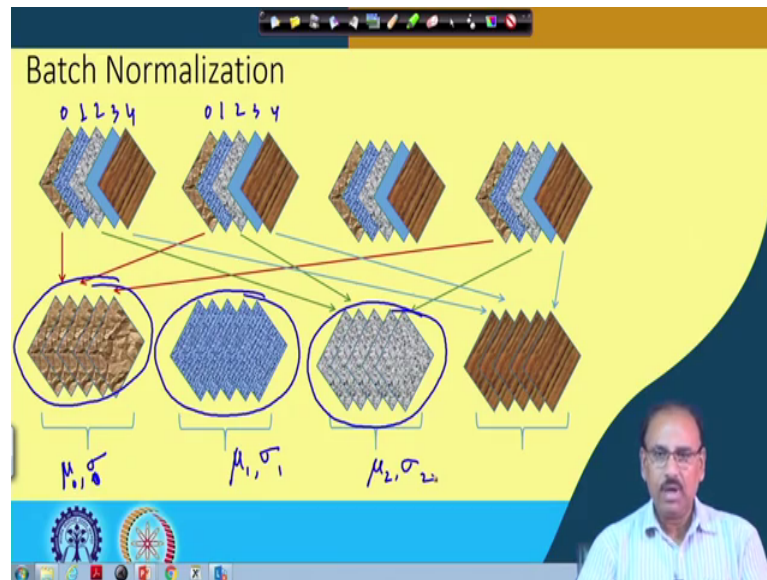


Now, what you do in case of batch normalization? In order to explain that let me arrange the feature maps or the channels in this form. So, over here in the horizontal direction what I put is the different training examples in a given batch say B . So, these are the training examples in a given batch. And in the vertical axis what we have put is the different channel indices, so obviously, the number of such channels that you have as we said is same as the number of convolution kernels.

And each of this channel has a size it has an width W and height H , so that is the size of every channel and all these channels being concatenated together gives over the feature map, ok. So, N actually gives you what is the batch size. I have N number of examples training examples in a particular batch and C is the number of channels and W and H is the width and height of every channel.

So, let us organize our feature map something like this. And this will help us to explain how the different types of normalizations, the batch normalization, instance normalization, layer normalization, group normalization and all that they work.

(Refer Slide Time: 17:14)



So, first we are talking about the batch normalization. In case of batch normalization, what you do is you collect all the identical channels from all the training examples in the same batch and group them together. So, here what we have done is, so if I put this as say channel number 0, this is channel 1, channel 2, channel 3, channel 4 and so on.

So, this is channel 0, channel 1, 2, 3, 4 and so on, you find that in this group we have made a group of channel 0s computed from all the training examples. This is a group of the features in channel ones from all the training examples. This is the group of features from channel 2 from all the training examples.

And then when you compute the mean and standard deviation, you compute one mean for this and one standard deviation for this, say this is μ_0 and standard deviation 0, I am putting it as subscript 0 because they are computed for from the 0th channel. This may be $\mu_1 \sigma_1$, this may be $\mu_2 \sigma_2$ and so on.

So, these two parameters μ and σ are computed from identical channels the same channels over all the training examples and that is how you compute the mean and sigma

in channel in the batch normalization technique. And using this mean and sigma values you go for normalization of the data. So, let us see how this is done.

(Refer Slide Time: 18:59)

The slide is titled "Batch Normalization" and contains the following mathematical expressions and a diagram:

$$x \in \mathbb{R}^{N \times C \times W \times H}$$

$$\mu_c = \frac{1}{NWH} \sum_{i=1}^N \sum_{j=1}^W \sum_{k=1}^H x_{icjk}$$

$$\sigma_c^2 = \frac{1}{NWH} \sum_{i=1}^N \sum_{j=1}^W \sum_{k=1}^H (x_{icjk} - \mu_c)^2$$

$$\hat{x} = \frac{x - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

To the right of the formulas is a grid diagram. The grid is 10 columns wide and 10 rows high. The label 'C' is placed to the left of the grid, and the label 'N' is placed below the grid. Blue arrows point from the variables x_{icjk} and $(x_{icjk} - \mu_c)^2$ in the formulas to the grid, indicating the corresponding dimensions.

So, basically when you are computing the mean of a particular channel this is your expression say $x_{i C j k}$. So, this x represents one feature element in channel C , at location $j k$ in that particular channel feature and the number of such channels or the number of training examples that we have is total N . So, μ_C that is the mean of a given channel is given by $x_{i C j k}$, where i and j , I said are the column and the row index within a single channel, ok.

And sorry j and k represents the row and column index within a given channel C and i is the index of the number of training examples that you have. So, the way you compute the μ_C , μ for a particular channel is given by this expression. In the same manner, you compute the variance that is σ_C^2 over following the same indices it becomes $x_{i C j k} - \mu_C$ square of that and you sum it over $i j$ and k and then we normalize with $N W$ and H and that becomes the variance for channel C .

And using this variance you go for normalization as we said before, so your normalized feature value \hat{x} now becomes $x - \mu_C$ upon square root of $\epsilon + \sigma_C^2$. So, this is your normalized value.

(Refer Slide Time: 21:07)

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

So, as a result if I can put it in another form also all the channels put together can be represented as a vector. So, what I do is, I compute the mean vector $\mu_{\mathcal{B}}$ over a batch which is given by x_i , i varying from 1 to m and then you normalize with respect to m . So, in this expression m is the number of examples that you have within the batch or the number of training examples within the batch.

In the same manner, you compute $\sigma_{\mathcal{B}}^2$ within a batch that is $\sigma_{\mathcal{B}}^2$ and then you normalize x_i with respect to this $\mu_{\mathcal{B}}$ and this $\sigma_{\mathcal{B}}$. So, your normalized value becomes \hat{x}_i which is $x_i - \mu_{\mathcal{B}}$ upon square root of $\epsilon + \sigma_{\mathcal{B}}^2$. So, this is what is your normalized value.

So, once you do that you find that all these normalized features that is \hat{x}_i or the collection of \hat{x}_i will have a mean value equal to 0 because from every x_i have subtracted $\mu_{\mathcal{B}}$ which is the main vector, ok. And then you have divided you have normalized with respect to the standard deviation which is $\sigma_{\mathcal{B}}$. So, as a result all \hat{x}_i we will have mean 0 and standard deviation 1.

Now, we find that there is one problem if on every data I map it to or I do some transformation this normalization, so that for all type of data your mean is 0, standard deviation is 1. Then you find that the data or the this new distribution, the redistributed data loses its class identity because every data now have the same distribution that is

mean 0 and standard deviation 1. So, which is not actually good for your classification purpose.

In classifier classification, every distribution should retain its class identity. So, in order to do that you are not only satisfied with the normalization with respect to mu and sigma, but what you need is you need some sort of reparameterization. So, this \hat{x}_i that you get this is now reparameterized. So, what you do is from this \hat{x}_i you map it to another data y_i , where y_i is $\gamma \hat{x}_i + \beta$; where this γ and β . So, now, γ becomes the standard deviation of this reparameterized data and β becomes the mean of this reparameterized data.

And that is what your batch normalized output. So, this batch normalized output, so this new distribution that we are generating that has two parameters one is γ , other one is β . And this both this β and γ are trainable or tuneable. So, while you train your neural network for the parameter vectors or weight vectors during gradient descent procedure at the same time you also train the neural network to tune these two new parameters which are γ and β .

And these two new parameters as they will be tuned with your input data during the training process, these two parameters will retain the class identification which will be subsequently helpful for classification of the data. So, how that is done?

(Refer Slide Time: 25:03)

Batch Normalization

$$\frac{\partial \ell}{\partial \hat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_B^2} = \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2}$$

$$\frac{\partial \ell}{\partial \mu_B} = \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_B)}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial \ell}{\partial \mu_B} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i \quad \checkmark \quad \gamma \leftarrow \gamma - \frac{\partial \ell}{\partial \gamma}$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \quad \checkmark \quad \beta \leftarrow \beta - \frac{\partial \ell}{\partial \beta}$$

So, you put that over here. So, what I need is if I want to tune these parameters γ and β during our gradient descent approach following the backpropagation learning technique; obviously, I need to get the gradient of the output error with respect to γ and β . And you find that as has been shown in these mathematical derivations. The derivations are here, I am not repeating them, you can verify that these derivations are correct.

So, through these derivations it has been shown that this re-parameter or the normalization and the reparameterization that we have done this is differentiable with respect to the parameters γ and β . And because it is differentiable, I can use the same gradient descent approach for tuning or for updating the γ and β values. So, for doing that what I need is over here this J is the loss function.

So, what I need to compute is $\frac{\partial J}{\partial \gamma}$ which is given by this, which is $\frac{\partial J}{\partial y_i}$ into x_i , take the summation over all the samples in the batch that is i is equal to 1 to m . Similarly, $\frac{\partial J}{\partial \beta}$ which is the gradient of the error with respect to β which is nothing, but $\frac{\partial J}{\partial y_i}$. So, again you find that you are following the chain rule of differentiation and you take the summation for i is equal to 1 to m .

So, once you have this then obviously, your update rule will be γ gets γ minus $\frac{\partial J}{\partial \gamma}$ and β gets β minus $\frac{\partial J}{\partial \beta}$. So, it is the same gradient descent rule that can be applied in the back propagation learning stage for update of these new parameters γ and β . And they will be finally, tuned and as a result what you are doing is you are normalizing the data at every layer whether it is the input layer or any of the hidden layers.

And the purpose of this normalization is as we have seen that because of the covariate shaped the distribution of data from batch to batch can vary widely, and the purpose of this normalization this batch normalization is that it tries to make sure that even if the data is a covariate shaped the amount of variance will not be match it will be within limit.

And because of this during the training process your classifier or the classification rule that you are generating through your deep neural network, the classifier will not hop much from one to another classifier classification boundary to another classification boundary and as a result your training process will be much faster.

So, in today's discussion what we have discussed is the batch normalization technique. In the subsequent classes, we will talk about other classification techniques. And as we have already said that the reference of other normalization techniques and as we have already said that the difference in this normalization techniques is basically how do you compute the mean and the standard deviation.

Thank you.