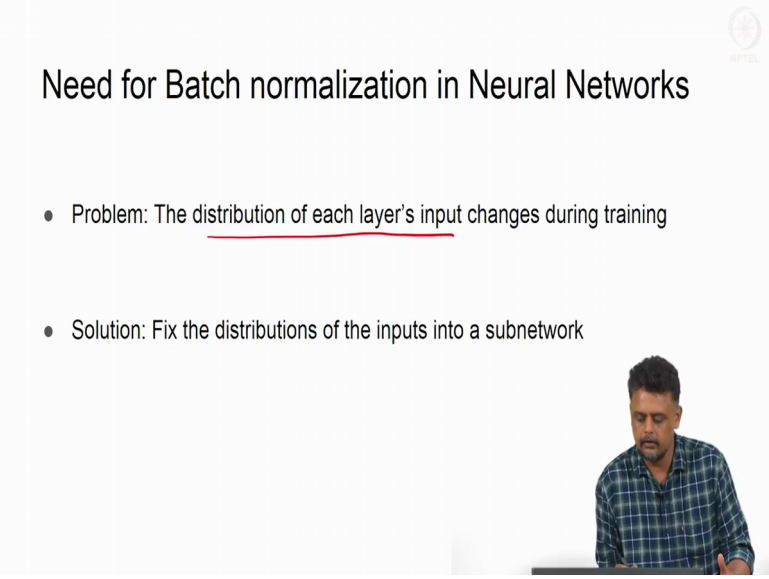
Machine Learning for Engineering and Science Applications
Dr. Ganapathy Krishnamurthi
Department of Engineering Design
Indian Institute of Technology Madras
Batch Normalizing

Hello and welcome back. So in this video, we will look at this technique called batch normalisation which helps in training the network, a deep neural network better and and it is kind of you can treat of it as a continuation from the data processing, preprocessing that we saw in the previous lecture, okay.

(Refer Slide Time 0:33)

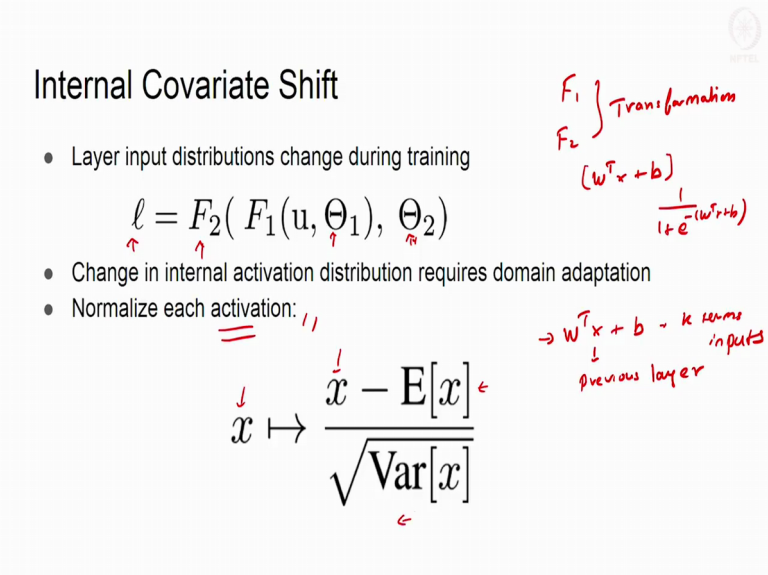


So what first what is, we will look at what is batch normalisation and then we will consider you know what is the problem here when you are training in deep neural network, so what happens, okay. So when we train a network, what happens is that the distribution of each layers input changes during training. We will see why that is in the next slide but we can see that as we train, because the weights keep changing, the input to a particular network, particular layer in the network would be changing dynamically.

So because if the weight changes drastically between 2 iterations, you have the same issue all right. And the solution is to somehow make sure that the distributions do not change too much in

the sense that the distributions of the inputs to the layer do not change too much okay. So let us see what do we mean by that and how we can address that problem okay.

(Refer Slide Time 1:26)



Okay. So let us consider this in a slightly more like in a functional form let us say okay. So F1 and F2 are some transformations. So what is a transform that happens in a neutral network in earlier? So in a layer, what happens? You have W transpose X plus B okay where X is the input to that layer. This is one transform that happens. And you would you pass it through a nonlinearity, so then you can get something like E raised to minus raised to minus right, you can do something like that.

So that is where transformation did not happen, right? So when W changes with every iteration, you can see that you know the inputs to the particular layer will also dramatically change. So you can think F1 and F2 as the transformation that happens to your inputs at every layer okay. So see you have to layers in succession, each layer categorised by theta by these parameters theta which is nothing but the weights and let us say another layer 2 characterised by this another set of weights, theta 2 right?

So once if as theta1 and theta2 keep changing, you can see that the, so if theta1 changes, then F1 will change. So the input to F2 changes right? Input to F2 will change. And if theta2 changes, then F2 itself will change, again L will change. okay. So if you have vary the input in other layer, then L will change. In the sense the change is of course, it is expected to happen because we are

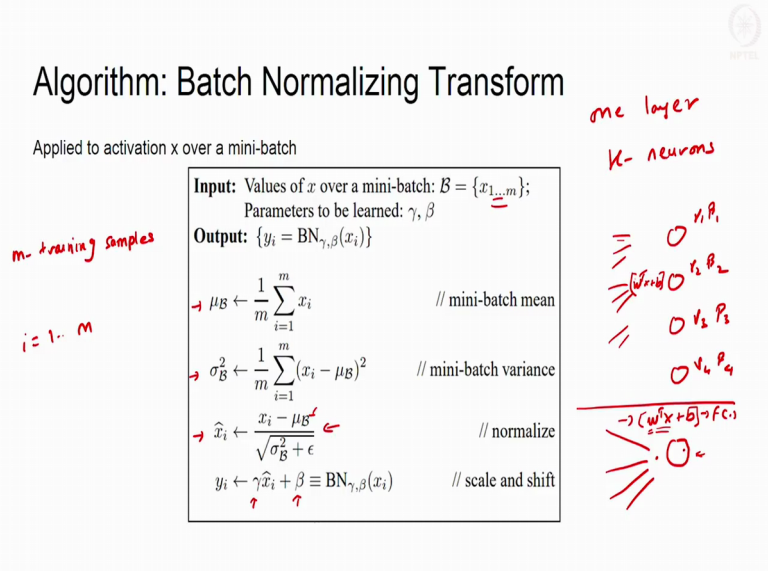
trying to estimate theta1 and theta2 if we think of F2 and F1 as the layers in a deep neural network.

But since these changes, if they are in the sense these changes are kind of random and large sometimes, then you have problems with conversions in deep neural (())(3:27) okay. So what we do to address this problem in a network is to normalise each activation, okay. So this is before we apply the nonlinearity, typically that is what is done is we can do, we always use X as to clarify the notations, we always use X to denote the input. In general, this is our training data input as what we, we usually use X for.

For the purposes of this video, think of X as the input to every layer. Okay. So every layer has a set of neurons and every neuron has an input coming into it. Right? And what is that input? That input coming into every neuron is W transpose X plus B, this is the output from previous layer, okay. So if there are K neurons in earlier, there will be K such terms right? K terms. Or K inputs. K terms or K inputs. So this is the input that is coming into A layer and what we do is so I have used X pretty much for everything, I have used the notation but then what we do is, we do X minus the mean or the expectation value of X divided by the variance of X.

This is what we saw earlier, this is your typical hex code normalisation but then how do we calculate expectation of X? What is this expectation of X? okay. So we will just go through the algorithm and then we will be very clear as to what, how this mean of X is calculated for every layer.

(Refer Slide Time 5:04)



Okay. So here is the algorithm, okay. We are considering one layer okay considering one layer. Okay. And let us say it has K neurons, it has K neurons and we are just trying to see how this batch normalising transform can be applied to that, okay. So if you have K neurons, right, you have this say K is 4, then we have inputs coming in from the previous layer, right? I am not going to draw it because it is too confusing.

So multiple inputs coming from the previous layer and then of course there is this W transpose, it is not the, they are fine transforms that we do. Okay. W transpose X plus B, right? And then followed by nonlinearity, that is the output of that particular layer. Okay. That is your activation, okay. So then once we have that, so how do we calculate this? So for every neuron, if we just take one neuron, okay. Let us just take one neuron. So we have all these inputs coming to that neuron okay with which you can calculate the layer combination, W transpose X plus B.

Then following, and then you apply the nonlinearity to that right, that is the output. Now what do we mean by calculating the mean of X? So this is one input, okay to that particular neuron. So what we do is, we consider a mini batch of M training samples. So there are M training samples in a mini batch, right? And we can once in the forward pass, we can actually forward pass all the M samples in succession okay and we can compute this W transpose X plus B for each mini batch.

So if we have M data points, then we will have M such calculated values for each neuron. Okay. So that for each activation, prior to passing it to a nonlinearity, you will have M such values corresponding to each data point in your mini batch. So this mean is calculated over the mini batch. So this is for A neuron right. So the A neuron and you have M input data points.

This neuron is let us say of the first or 2nd layer and but then when you do the forward pass for A neuron using the weights that have already been estimated randomly initialised, as you do the forward pass through the network, for every point in that M points in that mini batch you will have one linear combination, okay. So we will have M such linear combinations with which you will estimate a mean, (())(8:16) and of course once you subscribe that mean out, you can estimate the variance square or the (())(8:23) square or the mini batch variance.

And you will normalise every neuron with that every A. So for I equal to 1 to M over the over individual input training samples of that mini batch, you will calculate this normalised data point. Okay. And once you have done that, then we define 2 parameters, gamma and beta, again for every so this if there are 4 of them here, there will be a gamma 1, beta 1, gamma 2, beta 2, gamma 3, beta 3 and gamma 4, beta 4. So for every neuron, they will have two hyper parameters, gamma and beta and you will do this transformation.

So how are gamma and beta estimated? There estimated through backprop. Because all this is you can think of this as a linear layer in a network and that is how it is typically interpreted. So this linear layer is inserted between your fine transform which is the linear combination of your neuron activation from the previous layer and the nonlinearity you apply. Okay. So it is in between these 2 layers you have the batch normalisation layer, okay.

So what it does is that, so this is the transformation that helps to make sure that your data distribution in the sense that the distribution of your, activations of your that deactivation that you complete for every neuron does not shift too drastically, okay, they are confined to be within a certain distribution. So when this happens, then training is automatically faster and it converges faster.

So you can think of one example where this will work is when one of the Ws is too large and you know it might lead to saturation, we saw that, we talked about that earlier. So by doing this

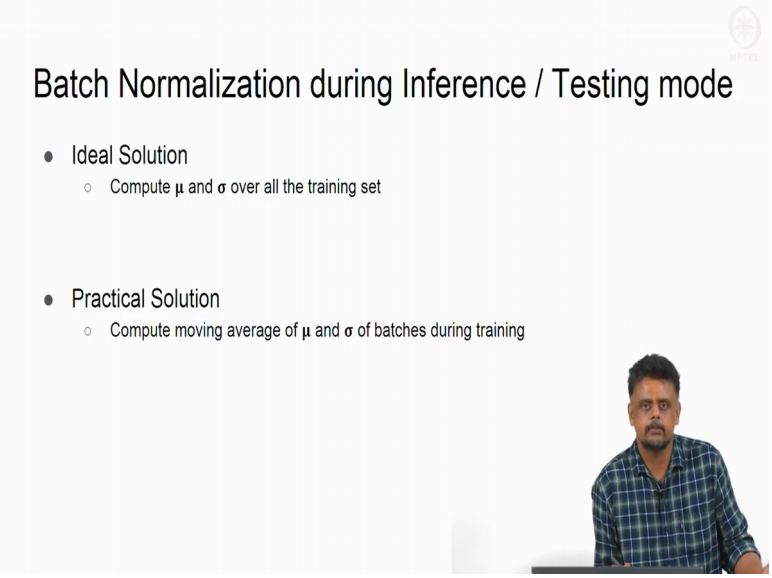
normalisation, you can prevent that from happening, also by making sure you can estimate gamma and beta. So you can also see that this is like a invertible confirmation and it can gamma and beta can be estimated so that YI can be just equal to XI okay. That is very easy to see (()) (10:41) okay.

So if the original calculated value is the one that is actually desirable, then gamma and beta can be, the network would estimate gamma and beta to be the inverse of this transformation that we did leading to a leading to identity okay. So this might sound cryptic but you should read the paper, I did post the paper up there and I have (())(11:01) okay. So just to recap once again, so for every neuron in the layer, you see that every neuron if this is a fully connected neural network, you can think that is the one that we are talking about in MLP, so every neuron in a particular layer, it gets inputs from the previous layer.

We call those, we denote those inputs by X okay. So and we are considering only one neuron at a time. So for every neuron, there is a linear combination of the activations from the previous layer, that is we that is what we call W transpose X plus B. Now for when you are doing training, there is a mini batch of data points, M data points. So do the forward pass and we calculate this W transpose X plus B for every data point in that mini batch and we do a mean and variance for that mini batch for that particular neuron, okay.

And then we scale the activation of that neuron for a particular data point, input data point by the calculated mean and standard deviation. Okay. And then of course we multiply it by this gamma and add by beta to get a transformed variable. So this gamma and beta and again estimated by back reputation. So remember that if there are K neurons in a (())(12:17) layer, you will have K such parameters. So it is the addition of K such parameters.

(Refer Slide Time 12:23)

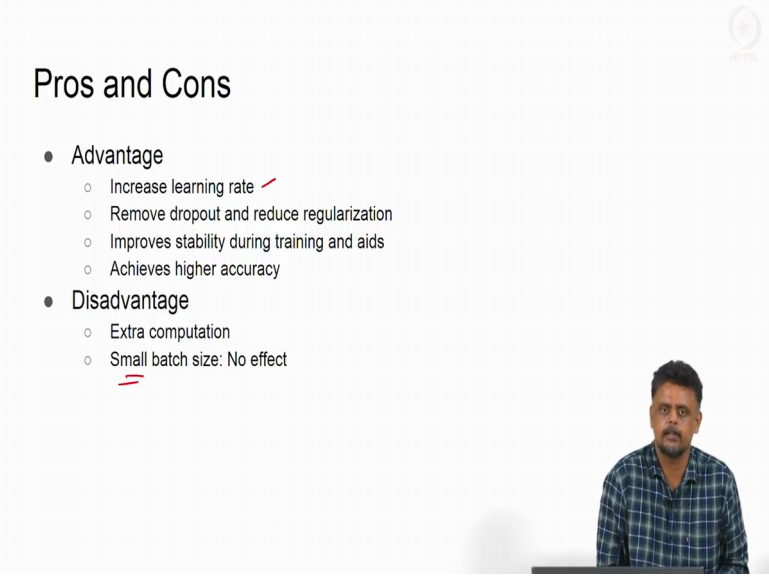


Once you have done training, so this is how it is trained. So for every layer, you will have gamma and beta and it will be estimated (())(12:27) training but do it by back propagation. Now once you have done training, how do you do the testing and inference? So then you still you need to remember for testing and inference, you would still have to you have to calculate this myu and Sigma, right? You have the gamma and beta but you still have to calculate myu and Sigma.

So what you do is for that, you can compute myu and Sigma over the entire training set for every layer for every neuron. Okay that is possible. And you have already converged on the upper plate values for the gamma and beta for every layer and for every neuron. Okay. There are places of computing myu and Sigma is running average, again you can do that as well as you know some exponentially varied average schemes are available but this is the way to, during testing you will calculate myu and Sigma for every layer, this myu and Sigma are for the activations of every layer, every neuron and every layer. Calculated by running it through the entire forward pass dataset. Okay.

That will be added computation or you can just maintain a running average during training. So each of these are fine. So that is one way of doing it.

(Refer Slide Time 13:34)



And one of the advantages, so apparently this the authors of this particular paper comment that it increases learning rate, so you cannot train with a high learning rate leading to fast convergence because sometimes you have high learning rates, you have large updates and sometimes get into saturation, that will not happen with this because you are doing this you are trying to constrain your activation values to lie within a range, typically that is what we are trying to do and that helps, okay.

It can also help you remove drop outs, so it acts kind of like a the regularisation effects of dropouts, advantages of dropouts apparently are also carried over by this batch normalisation, okay. Improves stability during training, same thing because if we have sometimes your activations can be very large, sometimes your weights can become very large leading to you know poor training and that can be taken care of by this batch normalisation. The extra computation burden is there because you are adding one more layer between you know extra layer is added before every set of neurons.

So that is the thing. And you need to have a significant backside. So we use a batch size of 1, 2 or 3, some of the large problems we require, memory constraints might, if your dataset is large, memory constraints might make you choose very small size datasets okay. And in that case there will be no effect. The statistical you know effects are lost by doing that. So then it is no point doing that for those kind of problems where you have very small batches. Reasonably large batch sizes, this will work okay.

So the one question that we have not addressed is the convolutional neural network. How do we do this in a convolutional neural network? Okay. It is a very interesting question. Actually the paper addresses that. The paper talks about how to calculate it. That would be a homework question, okay. So now I have given you the homework already. So for you to read the paper, I will upload the paper soon.

So read the paper and inside the paper, they do comment on how this particular batch normalisation can be implemented in a convolutional network, remember that, what is being described in this video is how it is implemented for a fully connected neural network. Okay. So that is all for batch normalisation. So that is we wanted to do these 2 videos together, basically data normalisation as well as batch normalisation because I think it helps you understand this better, okay thank you.