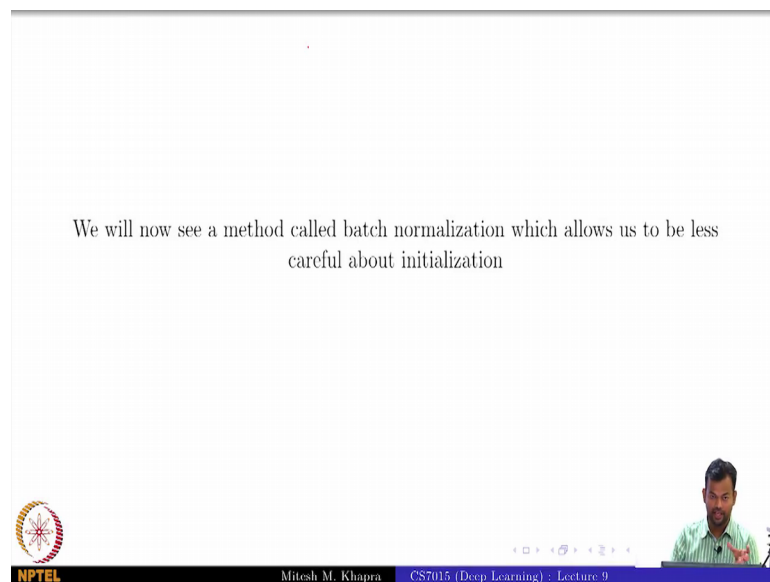


**Deep Learning**  
**Prof. Mitesh M Khapra**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology Madras**

**Module – 9.5**  
**Lecture – 09**  
**Batch Normalization**

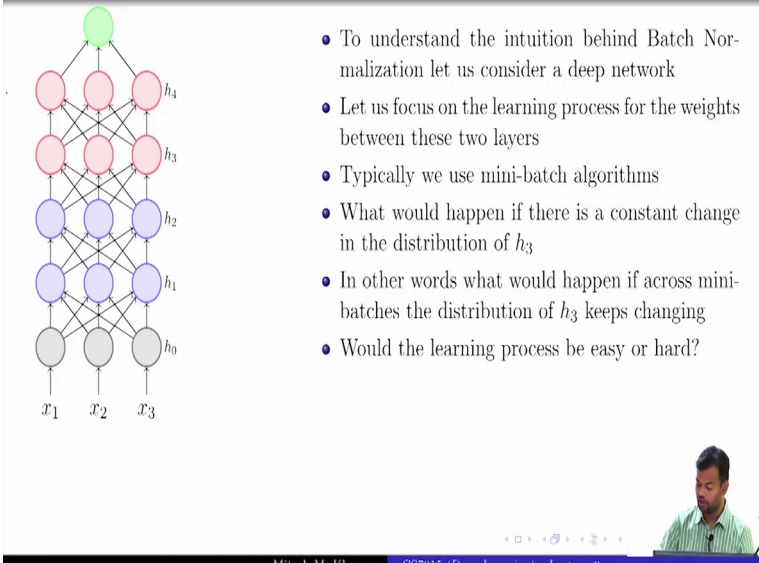
Now, we will end with something known as Batch Normalization, which is again almost a d facto standard at least in convolutional neural networks. So, if you are dealing with convolutional neural networks you will use something known as batch normalization.

(Refer Slide Time: 00:26)



So, let us see what it is, so this is again something which is some method which allows us to be less careful about initialization, so let us see why that happens.

(Refer Slide Time: 00:36)



- To understand the intuition behind Batch Normalization let us consider a deep network
- Let us focus on the learning process for the weights between these two layers
- Typically we use mini-batch algorithms
- What would happen if there is a constant change in the distribution of  $h_3$
- In other words what would happen if across mini-batches the distribution of  $h_3$  keeps changing
- Would the learning process be easy or hard?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, to understand the intuition behind this, let us consider a deep neural network and let us focus on the last two layers  $h_4$  and  $h_3$ ; Now, typically will use some mini-batch algorithm for training right, so we will use mini-batch version of gradient descent or mini-batch version of Adam or any of these algorithms right.

Now, what would happen if there is a constant change in the distribution of  $h_3$  no just think about the question that I am trying to ask you. So, as far as these two layers are concerned  $h_3$  is the input and  $h_4$  is the output it does not matter what has happened so far or in particular does not matter what  $x$  was whether it came from a normal distribution or whatever distribution right.

At this point my input is  $h_3$  and my output is  $h_4$ , now I am training it in mini batches what if across batches my distribution of  $h_3$  looks very different; what would happen, is it a good thing or a bad thing? It is a bad thing right. So, if you have training data right just think of this as I said just focus on this layer, if you have an input which is not following a fixed distribution is constantly changing during your training then that is always a bad thing right, because you try to adjust to one distribution and now again the distribution is completely changing. So, that always makes our training very very difficult right. So, if you have a very fluctu and distribution then a training is going to be hard ok, so that is the intuition that I want to build.

(Refer Slide Time: 02:06)

- It would help if the pre-activations at each layer were unit Gaussians
- Why not explicitly ensure this by standardizing the pre-activation?
$$\hat{s}_{ik} = \frac{s_{ik} - E[s_{ik}]}{\sqrt{\text{var}(s_{ik})}}$$
- But how do we compute  $E[s_{ik}]$  and  $\text{Var}[s_{ik}]$ ?
- We compute it from a mini-batch

So, now this could actually happen, so it would help if the pre activations at every layer are you need Gaussians because, for the input we made a case that will make the input as unit Gaussian right.

So, that things are very nice they are all coming all the inputs are coming from the same distribution, but we now realize that at every layer we have an input right it is not that the original input the only input even  $h_3$  is an input even  $h_4$  is an input and so on. So, why not ensure that at every layer your inputs or your  $h_1, h_2, h_3$  also is something, which looks like a Gaussian distribution which comes from a Gaussian distribution. Why not ensure that, that is the basic idea behind batch normalization and how do you do that is the following that you had computed this  $S_{ik}$  just as we had done in the derivation earlier right, so  $S_{ik}$  is one of these guys.

Now, if you do this what are you actually doing you just normalizing it right you are subtracting the mean and dividing by the variance, so that means you are making it zero mean unit variance and that is the intuition which I was trying to build that  $y$  naught at every layer have this good distribution which is zero mean unit variance. By even if you are feeding it multiple batches for that batch you will ensure that by this subtraction and division or the normalization process the data will become unit variance and zero mean ok. So, now at every batch the data is coming from the same distribution even if it was

originally from a distant different distribution, fine. But how do we compute this mean and variance?

So, did you understand the question that I am asking I am focusing on this  $S_i$  I want to subtract the mean of that  $S_i$ , how do I do that? So, the name gives it away batch normalization it cannot be more explicit than that. So, compute the mean for the current batch and the variance for the current batch and normalize your inputs or normalize the  $S_i$ 's according to that you get this. So, now end up with a situation where all your inputs at every layer across different mini-batches seem to come from the same distribution is it fine, the current batch, so you take the average value from the current batch right.

So, then it will become zero mean for that batch and unit variance for that batch and this you are ensuring for every batch. So, every independently every batch you are ensuring that it comes from a zero mean unit variance distribution right. So, overall the effect is that all the batches are coming from the same distribution no. So, at validation time you will compute the mean and variance from your entire data entire training data once after the training is done right.

So, now we will computed from a mini-batch and this is ensure that across mini-batches, now your input always comes from a zero mean unit variance distribution across all the layers.

(Refer Slide Time: 05:02)

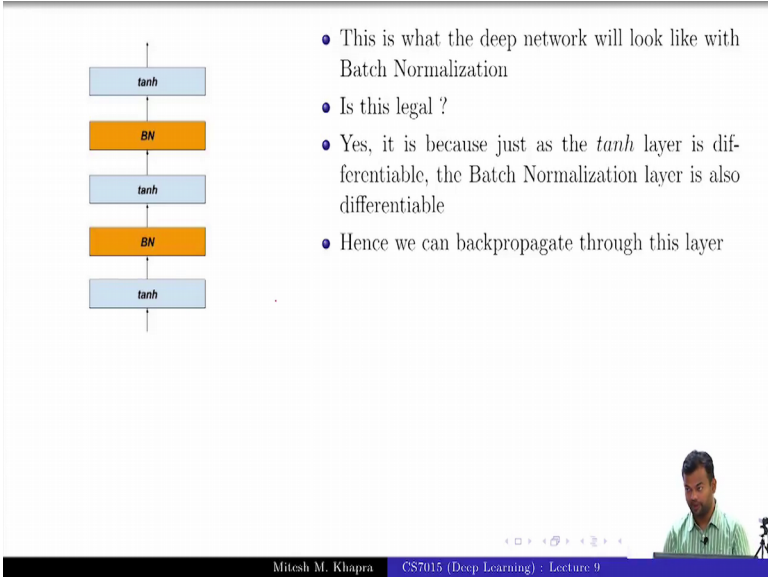
- This is what the deep network will look like with Batch Normalization
- Is this legal ?

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

This is what a deep network will look like with batch normalization right, so what will happen is you passed an input you computed this tan h, then you will have this batch normalization layer watch is what is the operation that the batch normalization is going to do this is the operation that it is going to talk ok. Everyone gets that and now it gives me a unit normalized distribution sorry it gives me a input coming from a zero mean unit variance distribution and then I pass it to the next layer again at a batch normalization layer.

So, after every layer you will actually add a batch normalization here, now my question is, is this legal? What is legal in this course anything that is differentiable right. So, you have to make sure that if we have added this operation it should be a differentiable operation. So, that you can come so now the gradients have to flow all the way here right, so that means I should be able to compute the gradients with respect to this. So, now this is one of my  $a_i$  and I should be able to compute  $\frac{dL}{da_i}$  with respect to something or rather the loss though of the loss function with respect to  $a_i$  by turns out that the operation, that you have done is actually differentiable.

(Refer Slide Time: 06:04)



- This is what the deep network will look like with Batch Normalization
- Is this legal ?
- Yes, it is because just as the  $\tanh$  layer is differentiable, the Batch Normalization layer is also differentiable
- Hence we can backpropagate through this layer

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

You can actually work that out and it is not important I am not going to derive it because, it is just yet another derivative that you will take, but it is a it should get the intuition from here right what you are doing is this simple operation and this just looks differentiable right.

So, the operation that you are doing is differentiable, so that is why you can add these batch normalization layers and you can back propagate through this layer. But now what is the catch here it somehow ties to the question that he was trying to ask, you are actually enforcing that all your are zero mean and unit variance right. So, this is again some sort of a constraint that you are enforcing right, what if that is not the best situation in which the network can learn; what if to distinguish between some classes it was ok. If the distribution was not same across all the batches they get this, they are enforcing a certain consider they are enforcing a certain condition on all the layers and all of them have to be zero mean and unit variance but that may not always be good.

(Refer Slide Time: 07:02)

$\gamma^k$  and  $\beta^k$  are additional parameters of the network.

- Catch: Do we necessarily want to force a unit gaussian input to the  $\tanh$  layer?
- Why not let the network learn what is best for it?
- After the Batch Normalization step add the following step:
 
$$y^{(k)} = \gamma^k \hat{s}_{ik} + \beta^{(k)}$$
- What happens if the network learns:
 
$$\gamma^k = \sqrt{\text{var}(x^k)}$$

$$\beta^k = E[x^k]$$
- We will recover  $s_{ik}$
- In other words by adjusting these additional parameters the network can learn to rec is more favourable

Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, they do something which is counterproductive, let us see what that is, why not let the network decide what is best for it. So, after the batch normalization layer so this is what a normalized  $S_{ik}$  was, after you have done that you compute a  $y^k$  and this is not the final output this is the output at the  $k$  th layer this is equal to this. Why do they do this and remember that gamma and beta are going to be learnable parameters, what are you doing actually you are again scaling it and shifting it this is the same as adjusting the variance and the mean right.

So, now what happens if the network learns the following you get back the  $S_{ik}$ , so you had taken  $S_{ik}$  and you had normalized it. But now if you allow these gammas and betas

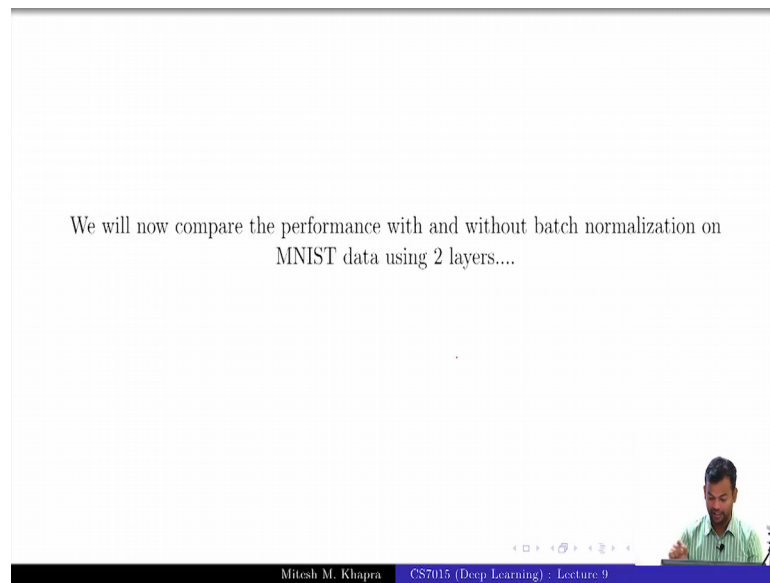
to be there in the network, then the network can decide that maybe at this layer I do not want this normalization I just want to stick to whatever output I was getting.

So, it could learn the gammas and betas in this way and ensure that you get back the unnormalized  $S$ , how many of you get this fine lot of you do not seem to get this; but I am pretty sure if you go back and look at it you will get it right. So, what is happening here is that is why I said it is counterproductive that you first forced it to make at unit mean and 0 variance and now you added no zero mean and unit variance and now you added this operation which is again a scaling and shifting operation. So, remember that when you make the data zero mean and unit variance that is exactly what you do you shift it, so that it become zero mean and you scale it. So, that it becomes unit variance.

So, you are again introducing parameters which again introduce the same flexibility that you could learn gamma and beta in such a way that you could get back the original data which was not normalized ok. So, if the network wants to learn that and if the network fees that is the right thing to do, then it has the flexibility to learn those parameters and you can recover  $S_i$ .

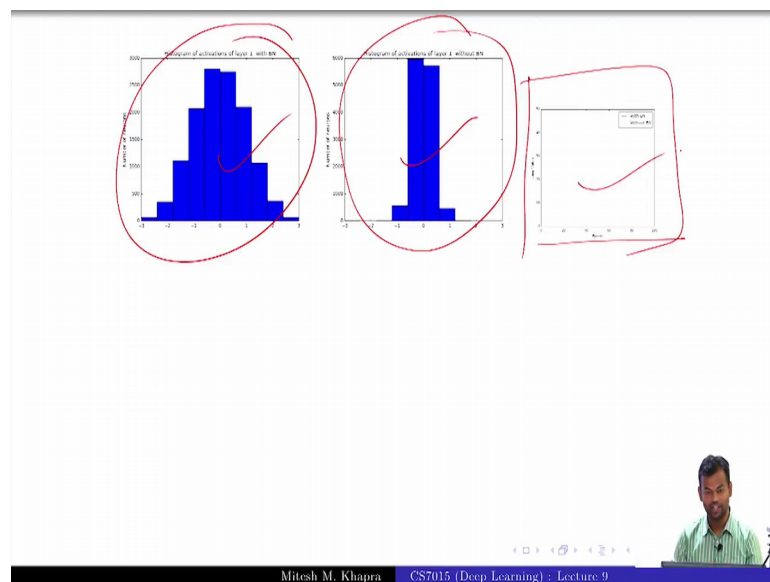
Yeah I think the rationale is that your first making is something which is more standard right and then from there trying to learn it instead of just trying to let it learn in the way. Do you get the difference between the two the first bringing it to all of these things to some standard value, which is between I mean which is the normal distribution and then from there allowing it to learn wherever it has to learn right that is the idea, but it could be the case that the other thing also works here.

(Refer Slide Time: 09:33)



So, now what we will do is we will compare the performance with and without batch normalization on MNIST data using 2 layers ok.

(Refer Slide Time: 09:40)



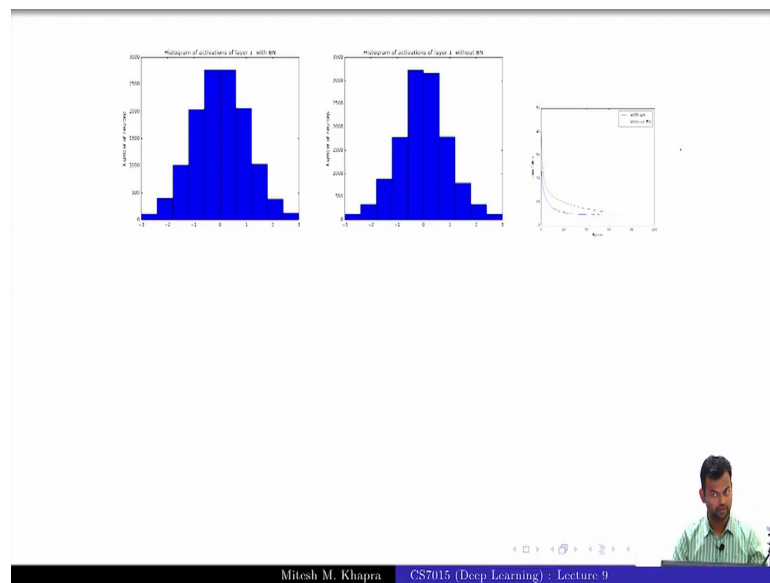
So, here in this figure what I am going to draw is the validation loss, am I no the training loss as I keep increasing the number of epochs and here I am showing you the histogram of the activation functions at layer 1. So, I have trained a deep feed forward neural network and I am showing you what do the activations look like at layer 1 with and without batch normalization. So, remember that we started with this intuition that



without batch normalization there would be this constant fluctuation and the data would seem to come from different distribution at every training instance right.

Whereas, with batch normalization you are ensuring at your data comes from zero mean unit variance distribution right and so that is one thing which I want to see another thing I want to see is that how does it affect training right. So, that is the animation that I am going to show you. So, focus on all these 3 things I do not know how you will do it, but focus on this, focus on this and focus on this with two eyes.

(Refer Slide Time: 10:49)



So, let us see to see what happened right, so this so now look at the focus on the leftmost figure. So, that does not seem to change much with respect to it is mean and variance right, but if you look at the middle figure that is constantly changing it is mean and variance right. And you see the effect on the training loss that the first one which was with batch normalization, that converges faster as compared to the second one right again an empirical result I am not really proving that this will always happen, this is what empirically we observed.

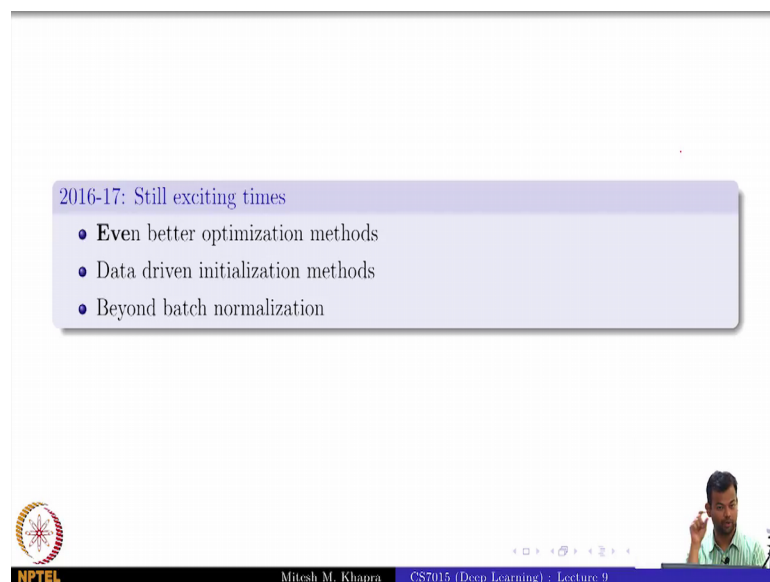
So, this was the story that we covered from 1986 to 2006 where back propagation was already it was already discovered, but was not working well and there was this park in 2006 that showed that we could do some things to make training really work for deep neural networks, but maybe that something is not sacrosanct. We could try different

things what we tried at that time was unsupervised free training which is almost nonexistent now.

But that lead that led to these thoughts that maybe this is because of optimization, generalization, regularization activation functions and so on right. So, there was a lot of research in these different areas and that led to a lot of developments which was better optimization algorithms, better regularization, better activation functions, better initializations and batch normalization right.

So, these a few concepts that you have seen in the past few lectures one being dropout and the other being weight initialization using this Xavier initialization or he initialization and this batch normalization right. This is something which is all prevalent right, so this is something that you will see in all deep neural networks that get trained definitely in convolutional neural networks and more often than not even in recurrent neural networks right. So, these are the two most popular types of neural networks. So, in both of these you will see that these ideas are regularly applied and they always lead to more stable training or better generalization right.

(Refer Slide Time: 12:47)



2016-17: Still exciting times

- Even better optimization methods
- Data driven initialization methods
- Beyond batch normalization

NPTEL Mitesh M. Khapra CS7015 (Deep Learning) : Lecture 9

So, now this was all which happened till 2016 or 17 what has happened still since then. So, there is still continuous research in designing better optimization methods, so as I said after Adam there was this Eve which did not become very popular, but there is still

people looking at better optimization methods and there is something which has been developed on Adam and came out in December last year.

Now, people have also started looking at data driven initialization methods right, so instead of having this fixed initialization which is drawn from a unit or just which is drawn from a normal distribution and then just divided by the square root of  $n$ . Why not think of data driven initialization methods that, so there are some works on that again not very popular because, most of the shelf things that you will try will not really do any data driven initialization.

But if you really think that you are stuck at some point then you could look at some of these works and see how they try to come up with initializations based on the data that you are dealing with and now after batch normalization there have been some other types of normalizations which have been proposed which seem to work better than batch normalization. But largely the stable configuration which has kind of prevalent is Adam in terms of optimization Xavier or he initialization in terms of initialization ReLU in terms of activation functions. What else is there batch normalization in terms of again regularization plus initialization and dropouts in terms of regularization right.

So, these are roughly the key terms that you will almost see in all the deeply living deep neural network people that you see right, you will always see when they describe the hyper parameters, they will say that this is how we initialized is this is the drop out that we use this is the batch normalization and the training algorithm more often than not is going to be Adam right.

So, they have seen some very crucial elements of training deep neural networks over the past 2 to 3 lectures right and now we will build on these and we will assume that this is what you are going to do. So, now when I talk about neural networks like convolutional neural networks and so on I not go back and tell you use Adam or use batch normalization or assume that you already know these things and you will try to train your networks using these tricks that we have your trick.

The first couple of lectures have been about tips and tricks for deep neural networks and from here on in the next lecture will move on to what to a right, because that is what you need for your assignment. So, in the next lecture we will do a word representations, so

that is essentially seeing an application of feed forward neural networks and from there on we will move on to convolutional neural network.