Fitting Batch Norm into a neural network

(SPEECH)
>>

(DESCRIPTION)
Text, Batch Normalization. Fitting Batch Norm into a neural network. Website, deep learning, dot, A.I.

(SPEECH)
So you have seen the equations for how to invent Batch Norm for maybe a single hidden layer.

Let's see how it fits into the training of a deep network.

(DESCRIPTION)
New slide, Adding Batch Norm to a network.

(SPEECH)
So, let's say you have a neural network like this, you've seen me say before that you can view each of the unit as computing two things.

First, it computes Z and then it applies the activation function to compute A.

And so we can think of each of these circles as representing a two step computation.

And similarly for the next layer, that is Z2 1, and A2 1, and so on.

So, if you were not applying Batch Norm, you would have an input X fit into the first hidden layer, and then first compute Z1, and this is governed by the parameters W1 and B1.

And then ordinarily, you would fit Z1 into the activation function to compute A1.

But what would do in Batch Norm is take this value Z1, and apply Batch Norm, sometimes abbreviated BN to it, and that's going to be governed by parameters, Beta 1 and Gamma 1, and this will give you this new normalize value Z1.

And then you fit that to the activation function to get A1, which is G1 applied to Z tilde 1.

Now, you've done the computation for the first layer, where this Batch Norms that really occurs in between the computation from Z and A.

Next, you take this value A1 and use it to compute Z2, and so this is now governed by W2, B2.

And similar to what you did for the first layer, you would take Z2 and apply it through Batch Norm, and we abbreviate it to BN now.

This is governed by Batch Norm parameters specific to the next layer.

So Beta 2, Gamma 2, and now this gives you Z tilde 2, and you use that to compute A2 by applying the activation function, and so on.

So once again, the Batch Norms that happens between computing Z and computing A.

And the intuition is that, instead of using the un-normalized value Z, you can use the normalized value Z tilde, that's the first layer.

The second layer as well, instead of using the un-normalized value Z2, you can use the mean and variance normalized values Z tilde 2.

So the parameters of your network are going to be W1, B1.

It turns out we'll get rid of the parameters but we'll see why in the next slide.

But for now, imagine the parameters are the usual W1.

B1, WL, BL, and we have added to this new network, additional parameters Beta 1, Gamma 1, Beta 2, Gamma 2, and so on, for each layer in which you are applying Batch Norm.

For clarity, note that these Betas here, these have nothing to do with the hyperparameter beta that we had for momentum over the computing the various exponentially weighted averages.

The authors of the Adam paper use Beta on their paper to denote that hyperparameter, the authors of the Batch Norm paper had used Beta to denote this parameter, but these are two completely different Betas.

I decided to stick with Beta in both cases, in case you read the original papers.

But the Beta 1, Beta 2, and so on, that Batch Norm tries to learn is a different Beta than the hyperparameter Beta used in momentum and the Adam and RMSprop algorithms.

So now that these are the new parameters of your algorithm, you would then use whether optimization you want, such as creating descent in order to implement it.

For example, you might compute D Beta L for a given layer, and then update the parameters Beta, gets updated as Beta minus learning rate times D Beta L. And you can also use Adam or RMSprop or momentum in order to update the parameters Beta and Gamma, not just creating descent.

And even though in the previous video, I had explained what the Batch Norm operation does, computes mean and variances and subtracts and divides by them.

If they are using a Deep Learning Programming Framework, usually you won't have to implement the Batch Norm step on Batch Norm layer yourself.

So the probing frameworks, that can be sub one line of code.

So for example, in terms of flow framework, you can implement Batch Normalization with this function.

We'll talk more about probing frameworks later, but in practice you might not end up needing to implement all these details yourself, knowing how it works so that you can get a better understanding of what your code is doing.

But implementing Batch Norm is often one line of code in the deep learning frameworks.

Now, so far, we've talked about Batch Norm as if you were training on your entire training site at the time as if you are using Batch gradient descent.

In practice, Batch Norm is usually applied with mini-batches of your training set.

(DESCRIPTION)
New slide, Working with mini-batches.

(SPEECH)
So the way you actually apply Batch Norm is you take your first mini-batch and compute Z1.

Same as we did on the previous slide using the parameters W1, B1 and then you take just this mini-batch and computer mean and variance of the Z1 on just this mini batch and then Batch Norm would subtract by the mean and divide by the standard deviation and then re-scale by Beta 1, Gamma 1, to give you Z1, and all this is on the first mini-batch, then you apply the activation function to get A1, and then you compute Z2 using W2, B2, and so on.

So you do all this in order to perform one step of gradient descent on the first mini-batch and then goes to the second mini-batch X2, and you do something similar where you will now compute Z1 on the second mini-batch and then use Batch Norm to compute Z1 tilde.

And so here in this Batch Norm step, You would be normalizing Z tilde using just the data in your second mini-batch, so does Batch Norm step here.

Let's look at the examples in your second mini-batch, computing the mean and variances of the Z1's on just that mini-batch and re-scaling by Beta and Gamma to get Z tilde, and so on.

And you do this with a third mini-batch, and keep training.

Now, there's one detail to the parameterization that I want to clean up, which is previously, I said that the parameters was WL, BL, for each layer as well as Beta L, and Gamma L. Now notice that the way Z was computed is as follows, ZL = WL x A of L - 1 + B of L. But what Batch Norm does, is it is going to look at the mini-batch and normalize ZL to first of mean 0 and standard variance, and then a rescale by Beta and Gamma.

But what that means is that, whatever is the value of BL is actually going to just get subtracted out, because during that Batch Normalization step, you are going to compute the means of the ZL's and subtract the mean.

And so adding any constant to all of the examples in the mini-batch, it doesn't change anything.

Because any constant you add will get cancelled out by the mean subtractions step.

So, if you're using Batch Norm, you can actually eliminate that parameter, or if you want, think of it as setting it permanently to 0.

So then the parameterization becomes ZL is just WL x AL - 1, And then you compute ZL normalized, and we compute Z tilde = Gamma ZL + Beta, you end up using this parameter Beta L in order to decide whats that mean of Z tilde L. Which is why guess post in this layer.

So just to recap, because Batch Norm zeroes out the mean of these ZL values in the layer, there's no point having this parameter BL, and so you must get rid of it, and instead is sort of replaced by Beta L, which is a parameter that controls that ends up affecting the shift or the biased terms.

Finally, remember that the dimension of ZL, because if you're doing this on one example, it's going to be NL by 1, and so BL, a dimension, NL by one, if NL was the number of hidden units in layer L. And so the dimension of Beta L and Gamma L is also going to be NL by 1 because that's the number of hidden units you have.

You have NL hidden units, and so Beta L and Gamma L are used to scale the mean and variance of each of the hidden units to whatever the network wants to set them to.

(DESCRIPTION)
New slide, Implementing gradient descent.

(SPEECH)
So, let's pull all together and describe how you can implement gradient descent using Batch Norm.

Assuming you're using mini-batch gradient descent, it rates for T = 1 to the number of many batches.

You would implement forward prop on mini-batch XT and doing forward prop in each hidden layer, use Batch Norm to replace ZL with Z tilde L. And so then it shows that within that mini-batch, the value Z end up with some normalized mean and variance and the values and the version of the normalized mean that and variance is Z tilde L. And then, you use back prop to compute DW, DB, for all the values of L, D Beta, D Gamma.

Although, technically, since you have got to get rid of B, this actually now goes away.

And then finally, you update the parameters.

So, W gets updated as W minus the learning rate times, as usual, Beta gets updated as Beta minus learning rate times DB, and similarly for Gamma.

And if you have computed the gradient as follows, you could use gradient descent.

That's what I've written down here, but this also works with gradient descent with momentum, or RMSprop, or Adam.

Where instead of taking this gradient descent update,nini-batch you could use the updates given by these other algorithms as we discussed in the previous week's videos.

Some of these other optimization algorithms as well can be used to update the parameters Beta and Gamma that Batch Norm added to algorithm.

So, I hope that gives you a sense of how you could implement Batch Norm from scratch if you wanted to.

If you're using one of the Deep Learning Programming frameworks which we will talk more about later, hopefully you can just call someone else's implementation in the Programming framework which will make using Batch Norm much easier.

Now, in case Batch Norm still seems a little bit mysterious if you're still not quite sure why it speeds up training so dramatically, let's go to the next video and talk more about why Batch Norm really works and what it is really doing.