Normalizing activations in a network

(DESCRIPTION)
Text, Batch Normalization. Normalizing activations in a network. Website, deep learning, dot, A.I.

(SPEECH)
In the rise of deep learning, one of the most important ideas has been an algorithm called batch normalization, created by two researchers, Sergey Ioffe and Christian Szegedy.

Batch normalization makes your hyperparameter search problem much easier, makes your neural network much more robust.

The choice of hyperparameters is a much bigger range of hyperparameters that work well, and will also enable you to much more easily train even very deep networks.

Let's see how batch normalization works.

(DESCRIPTION)
New slide, Normalization inputs to speed up learning.

(SPEECH)
When training a model, such as logistic regression, you might remember that normalizing the input features can speed up learnings in compute the means, subtract off the means from your training sets.

Compute the variances.

The sum of xi squared.

This is an element-wise squaring.

And then normalize your data set according to the variances.

And we saw in an earlier video how this can turn the contours of your learning problem from something that might be very elongated to something that is more round, and easier for an algorithm like gradient descent to optimize.

So this works, in terms of normalizing the input feature values to a neural network, alter the regression.

Now, how about a deeper model?

You have not just input features x, but in this layer you have activations a1, in this layer, you have activations a2 and so on.

So if you want to train the parameters, say w3, b3, then wouldn't it be nice if you can normalize the mean and variance of a2 to make the training of w3, b3 more efficient?

In the case of logistic regression, we saw how normalizing x1, x2, x3 maybe helps you train w and b more efficiently.

So here, the question is, for any hidden layer, can we normalize, The values of a, let's say a2, in this example but really any hidden layer, so as to train w3 b3 faster, right?

Since a2 is the input to the next layer, that therefore affects your training of w3 and b3.

So this is what batch norm does, batch normalization, or batch norm for short, does.

Although technically, we'll actually normalize the values of not a2 but z2.

There are some debates in the deep learning literature about whether you should normalize the value before the activation function, so z2, or whether you should normalize the value after applying the activation function, a2.

In practice, normalizing z2 is done much more often.

So that's the version I'll present and what I would recommend you use as a default choice.

So

(DESCRIPTION)
New slide, Implementing Batch Norm.

(SPEECH)
here is how you will implement batch norm.

Given some intermediate values, In your neural net, Let's say that you have some hidden unit values z1 up to zm, and this is really from some hidden layer, so it'd be more accurate to write this as z for some hidden layer i for i equals 1 through m.

But to reduce writing, I'm going to omit this [l], just to simplify the notation on this line.

So given these values, what you do is compute the mean as follows.

Okay, and all this is specific to some layer l, but I'm omitting the [l].

And then you compute the variance using pretty much the formula you would expect and then you would take each the zis and normalize it.

So you get zi normalized by subtracting off the mean and dividing by the standard deviation.

For numerical stability, we usually add epsilon to the denominator like that just in case sigma squared turns out to be zero in some estimate.

And so now we've taken these values z and normalized them to have mean 0 and standard unit variance.

So every component of z has mean 0 and variance 1.

But we don't want the hidden units to always have mean 0 and variance 1.

Maybe it makes sense for hidden units to have a different distribution, so what we'll do instead is compute, I'm going to call this z tilde = gamma zi norm + beta.

And here, gamma and beta are learnable parameters of your model.

So we're using gradient descent, or some other algorithm, like the gradient descent of momentum, or rms proper atom, you would update the parameters gamma and beta, just as you would update the weights of your neural network.

Now, notice that the effect of gamma and beta is that it allows you to set the mean of z tilde to be whatever you want it to be.

In fact, if gamma equals square root sigma squared plus epsilon, so if gamma were equal to this denominator term.

And if beta were equal to mu, so this value up here, then the effect of gamma z norm plus beta is that it would exactly invert this equation.

So if this is true, then actually z tilde i is equal to zi.

And so by an appropriate setting of the parameters gamma and beta, this normalization step, that is, these four equations is just computing essentially the identity function.

But by choosing other values of gamma and beta, this allows you to make the hidden unit values have other means and variances as well.

And so the way you fit this into your neural network is, whereas previously you were using these values z1, z2, and so on, you would now use z tilde i, Instead of zi for the later computations in your neural network.

And you want to put back in this [l] to explicitly denote which layer it is in, you can put it back there.

So the intuition I hope you'll take away from this is that we saw how normalizing the input features x can help learning in a neural network.

And what batch norm does is it applies that normalization process not just to the input layer, but to the values even deep in some hidden layer in the neural network.

So it will apply this type of normalization to normalize the mean and variance of some of your hidden units' values, z.

But one difference between the training input and these hidden unit values is you might not want your hidden unit values be forced to have mean 0 and variance 1.

For example, if you have a sigmoid activation function, you don't want your values to always be clustered here.

You might want them to have a larger variance or have a mean that's different than 0, in order to better take advantage of the nonlinearity of the sigmoid function rather than have all your values be in just this linear regime.

So that's why with the parameters gamma and beta, you can now make sure that your zi values have the range of values that you want.

But what it does really is it then shows that your hidden units have standardized mean and variance, where the mean and variance are controlled by two explicit parameters gamma and beta which the learning algorithm can set to whatever it wants.

So what it really does is it normalizes in mean and variance of these hidden unit values, really the zis, to have some fixed mean and variance.

And that mean and variance could be 0 and 1, or it could be some other value, and it's controlled by these parameters gamma and beta.

So I hope that gives you a sense of the mechanics of how to implement batch norm, at least for a single layer in the neural network.

In the next video, I'm going to show you how to fit batch norm into a neural network, even a deep neural network, and how to make it work for the many different layers of a neural network.

And after that, we'll get some more intuition about why batch norm could help you train your neural network.

So in case why it works still seems a little bit mysterious, stay with me, and I think in two videos from now we'll really make that clearer.