Backpropagation intuition (optional)

(SPEECH)
In

(DESCRIPTION)
Text, Deep learning dot AI. One hidden layer Neural Network. Backpropagation intuition (Optional). A man speaks to camera

(SPEECH)
the last video, you saw the equations for back propagation.

In this video, let's go over some intuition using the computation graph for how those equations were derived.

This video is completely optional.

So, feel free to watch or not.

You should be able to do the whole work either way.

So, recall that when we talk about logistic regression, we had this forward pass where we compute Z, then A and then the loss.

And

(DESCRIPTION)
Text, Computing gradients. Logistic regression. A flow chart with x, w, b feeding into z equals w raised to the t times x plus b. Next step, a equals sigma of z. Next step, L of a comma y

(SPEECH)
then to take the derivatives, we had this backward pass where we could first compute DA, and then go on to compute DZ, and then go on to compute DW and DB.

(DESCRIPTION)
Backwards arrow from L of a comma y to a equals sigma of z. D.a. underneath. Arrow back to previous step, z equals w to the t times x plus b, with dz underneath. Arrows back from there to dw and db.

(SPEECH)
So, the definition for the loss was L of A, Y equals negative Y log A minus one, minus Y times log one minus A.

So, if you are familiar with calculus and you take the derivative of this with respect to A, that would give you the formula for DA.

So, DA is equal to that.

And if we actually figure out the calculus you could show that this is negative Y over A plus one minus Y over one minus A.

You just kind of divide that from calculus by taking derivatives of this.

It turns out when you take another step backwards to compute DZ, we did work out that DZ is equal to A minus Y. I did explain why previously, but it turns out that from the chamber of calculus DZ is equal to DA times G prime of Z.

Where here G of Z equals sigmoid of Z is our activation function for this output unit in logistic regression, right?

So, just remember this is still logistic regression where we have X1, X2, X3 and then just one sigmoid unit and that gives us A, will gives us Y end.

(DESCRIPTION)
Flow chart with x1, x2, x3 flowing to sigma flowing to y hat, which is equal to a.

(SPEECH)
So, here are the activation function was a sigmoid function.

And as an aside, only for those of you familiar with the chamber of calculus the reason for this is because A is equal to sigmoid of Z.

And so, partial of L with respect to Z is equal to partial of L with respect to A times DA, DZ.

This is A is equal to sigmoid of Z, this is equal to DDZ, G of Z, which is equal to G prime of Z.

So, that's why this expression which is DZ in our code is equal to this expression which is DA in our code times G prime of Z.

And so this is just that.

So, that last derivation would made sense only if you're familiar with calculus and specifically the chamber from calculus.

But if not don't worry about it.

I'll try to explain the intuition wherever it's needed.

And then finally having computed DZ for this regression, we will compute DW which turns out was DZ times X and DB which is just DZ when you have a single training example.

So, that was logistic regression.

So, what we're going to do when computing back propagation for a neural network is a calculation a lot like this but only we'll do it twice because now we have not X going to an output unit, but X going to a hidden layer and then going to an output unit.

And so instead of this computation being sort of one step as we have here, we'll have you two steps here in this kind of a neural network with two layers.

(DESCRIPTION)
Neural network gradients

(SPEECH)
So, in this two layer neural network that is we have the input layer, a hidden layer and then output layer.

Remember the steps of a computation.

(DESCRIPTION)
Flow chart. X, w1, b1 feed into z1 equals w1x plus b1. Next step, a1 equals sigma of z1. W2 and b2 join the next step, z2 equals w2 times a1 plus b2. Next step, a2 equals sigma of z2. Final step, L of a2 comma y

(SPEECH)
First you compute Z1 using this equation, and then compute A1 and then you compute Z2.

And notice Z2 also depends on the parameters W2 and B2.

And then based on Z2, compute A2 and then finally that gives you the loss.

What backpropagation does is it will go backward to compute DA2 and then DZ2, and then you go back to compute DW2 and DP2, go backwards to compute DA1, DZ1 and so on.

We don't need to take the riveter as respect to the input X since the input X for supervised learning suffix.

We're not trying to optimize X so we won't bother to take the riveters.

At least, for supervised learning, we respect X. I'm going to skip explicitly computing DA2.

If you want, you can actually compute DA2 and then use that to compute DZ2 but, in practice, you could collapse both of these steps into one step so you end up at DZ2= A2-Y, same as before.

And, you have also, I'm going to write DW2 and DB2 down here below.

You have that DW2=DZ2*A1, transpose, and DB2=DZ2.

This step is quite similar for logistic regression where we had that DW=DZ*X except that now, A1 plays the role of X and there's an extra transpose there because the relationship between the capital matrix W and our individual parameters W, there's a transpose there, right?

Because W=[---] in the case of the logistic regression with a single output.

DW2 is like that, whereas, W here was a column vector so that's why it has an extra transpose for A1, whereas, we didn't for X here for logistic regression.

This completes half of backpropagation.

Then, again, you can compute DA1 if you wish.

Although, in practice, the computation for DA1 and the DZ1 are usually collapsed into one step and so what you'll actually implement is that DZ1=W2, transpose *DZ2, and then times an element Y's product of G1 prime of Z1.

And, just to do a check on the dimensions, right?

If you have a new network that looks like this, I'll put Y if so.

If you have N0, NX=N0 input features, N1 head in units, and N2 so far.

N2, in our case, just one output unit, then the matrix W2 is (N2,N1) dimensional, Z2 and therefore DZ2 are going to be (N2,N1) by one dimensional.

This really is going to be a one by one when we are doing binary classification, and Z1 and therefore also DZ1 are going to be N1 by one dimensional, right?

Note that for any variable foo and D foo always have the same dimension.

That's why W and DW always have the same dimension and similarly, for B and DB and Z and DZ and so on.

To make sure that the dimensions of this all match up, we have that DZ1=W2 transpose times DZ2 and then this is an element Y's product times G1 prime of Z1.

Matching the dimensions from above, this is going to be N1 by one=W2 transpose, we transpose of this so there's going to be N1 by N2 dimensional.

DZ2 is going to be N2 by one dimensional and then this, this is the same dimension as Z1.

This is also N1 by one dimensional so element Y's product.

The dimensions do make sense, right?

N1 by one dimensional vector can be obtained by N1 by N2 dimensional matrix times N2 by N1 because the product of these two things gives you an N1 by one dimensional matrix and so this becomes the element Y's product of two N1 by one dimensional vectors, and so the dimensions do match.

One tip when implementing a back prop.

If you just make sure that the dimensions of your matrices match up, so you think through what are the dimensions of the various matrices including W1, W2, Z1, Z2, A1, A2 and so on and just make sure that the dimensions of these matrix operations match up, sometimes that will already eliminate quite a lot of bugs in back prop.

All right. This gives us the DZ1 and then finally, just to wrap up DW1 and DB1, we should write them here I guess, but since I'm running of the space right on the right of the slight, DW1 and DB1 are given by the following formulas.

(DESCRIPTION)
dW1

(SPEECH)
This is going to be equal to the DZ1 times X transpose and

(DESCRIPTION)
db1

(SPEECH)
this is going to be equal to DZ.

You might notice a similarity between these equations and these equations, which is really no coincidence because X plays the role of A0 so X transpose is A0 transpose.

Those equations are actually very similar.

That gives a sense for how backpropagation is derived.

We have six key equations here for DZ2, DW2, DB2, DZ1,DW1 and D1.

Let me just take these six equations and copy them over to the next slide. Here they are.

(DESCRIPTION)
Summary of gradient descent. dz2 equals a2 minus y. dw2 equals dz2 times a1 transposed. db2 equals dz2. dz1 equals W2 transposed times dz2 times g1 prime of z1. dW2 equals dz1 times x transposed. db1 equals dz1.

(SPEECH)
So far, we have to write backpropagation, for if you are training on a single training example at the time, but it should come as no surprise that rather than working on a single example at a time, we would like to vectorize across different training examples.

(DESCRIPTION)
Vectorized implementation

(SPEECH)
We remember that for propagation, when we're operating on one example at a time, we had equations like this as was say A1=G1 of Z1.

(DESCRIPTION)
z1 equals w1 times x plus b1. a1 equals g1 of z1.

(SPEECH)
In order to vectorize, we took say the Zs and stacked them up in columns like this onto Z1M and call this capital Z.

(DESCRIPTION)
Capital Z1 is lowercase z1 1, lowercase z1 2, through to lowercase z 1 m

(SPEECH)
Then we found that by stacking things up in columns and defining the capital uppercase version of this, we then just had Z1=W1 X + B and A1=G1 of Z1, right?

(DESCRIPTION)
Capital Z1, capital W1, capital X, capital A

(SPEECH)
We define the notation very carefully in this course to make sure that stacking examples into different columns of a matrix makes all this work out.

It turns out that if you go through the math carefully, the same trick also works for backpropagation so the vectorize equations are as follows.

First, if you take these DZs for different training examples and stack them as the different columns of a matrix and the same for this and the same for this, then this is the vectorize implementation and

(DESCRIPTION)
all capital letters. dZ 2 equals A2 minus Y

(SPEECH)
then here's the definition for, or here's how you can compute DW2.

There

(DESCRIPTION)
d capital W2 equals 1 over m times d capital Z 2 times capital A1 transposed

(SPEECH)
is this extra 1/M because the cost function J is this 1/M of sum for Y = one through M of the losses.

(DESCRIPTION)
J equals 1 over m of sum for y equals 1 of the losses for y hat, comma y

(SPEECH)
When computing the riveters, we have that extra 1/M term just as we did when we were computing the wait up days for the logistic regression.

That's the update you get for DB2.

(DESCRIPTION)
d b 2 equals 1 over m times np dot sum of d capital Z2, axis equals 1, keepdims equals True

(SPEECH)
Again, some of the DZs and then with a 1/M and then DZ1 is computed as follows.

(DESCRIPTION)
d capital Z 1 equals capital W2 transposed times d capital Z2 times g1 prime of capital Z1

(SPEECH)
Once again, this is an element Y's product only whereas previously, we saw on the previous slide that this was an N1 by one dimensional vector.

Now,

(DESCRIPTION)
Underlines both d lowercase z1 and d capital Z1

(SPEECH)
this is a N1 by M dimensional matrix.

(DESCRIPTION)
Underlines capital W2 times d capital Z2 and g1 prime of capital Z1, the two parts of d capital Z of 1

(SPEECH)
Both of these are also N1 by M dimensional.

That's why that asterisk is element Y's product and then finally, the remaining two updates.

(DESCRIPTION)
d capital W 1 equals 1 over m dimtes d capital Z 1 times capital X transposed. db1 equals 1 over m times np dot sum of d capital Z1, axis equals 1, keepdims equals True

(SPEECH)
Perhaps it shouldn't look too surprising.

I hope that gives you some intuition for how the backpropagation algorithm is derived.

In all of machine learning, I think the derivation of the backpropagation algorithm is actually one of the most complicated pieces of math I've seen, and it requires knowing both linear algebra as well as the derivative of matrices to re-derive it from scratch from first principles.

If you are an expert in matrix calculus, using this process, you might prove the derivative algorithm yourself, but I think there are actually plenty of deep learning practitioners that have seen the derivation at about the level you've seen in this video and are already able to have all the very intuitions and be able to implement this algorithm very effectively.

If you are an expert in calculus, do see if you can derive the whole thing from scratch.

It is one of the very hardest pieces of math.

One of the very hardest derivations that I've seen in all of machine learning.

Either way, if you implement this, this will work and I think you have enough intuitions to tune and get it to work.

There's just one last detail I want to share with you before you implement your neural network, which is how to initialize the weights of your neural network.

It turns out that initializing your parameters, not to zero but randomly, turns out to be very important for training your neural network.

In the next video, you'll see why.