Gradient descent for Neural Networks

(SPEECH)
All

(DESCRIPTION)
Text, One hidden layer. Neural Network. Gradient descent for networks. Website, deep learning, dot, A.I.

(SPEECH)
right. I think this'll be an exciting video.

In this video, you'll see how to implement gradient descent for your neural network with one hidden layer.

In this video, I'm going to just give you the equations you need to implement in order to get back-propagation or to get gradient descent working, and then in the video after this one, I'll give some more intuition about why these particular equations are the accurate equations, are the correct equations for computing the gradients you need for your neural network.

(DESCRIPTION)
New slide, Gradient descent for neural networks.

(SPEECH)
So, your neural network, with a single hidden layer for now, will have parameters W1, B1, W2, and B2.

So, as a reminder, if you have NX or alternatively N0 input features, and N1 hidden units, and N2 output units in our examples.

So far I've only had N2 equals one, then the matrix W1 will be N1 by N0.

B1 will be an N1 dimensional vector, so we can write that as N1 by one-dimensional matrix, really a column vector.

The dimensions of W2 will be N2 by N1, and the dimension of B2 will be N2 by one.

Right, so far we've only seen examples where N2 is equal to one, where you have just one single hidden unit.

So, you also have a cost function for a neural network.

For now, I'm just going to assume that you're doing binary classification.

So, in that case, the cost of your parameters as follows is going to be one over M of the average of that loss function.

So, L here is the loss when your neural network predicts Y hat, right.

This is really A2 when the gradient label is equal to Y.

If you're doing binary classification, the loss function can be exactly what you use for logistic regression earlier.

So, to train the parameters of your algorithm, you need to perform gradient descent.

When training a neural network, it is important to initialize the parameters randomly rather than to all zeros.

We'll see later why that's the case, but after initializing the parameter to something, each loop or gradient descents with computed predictions.

So, you basically compute your Y hat I, for I equals one through M, say.

Then, you need to compute the derivative.

So, you need to compute DW1, and that's the derivative of the cost function with respect to the parameter W1, you can compute another variable, shall I call DB1, which is the derivative or the slope of your cost function with respect to the variable B1 and so on.

Similarly for the other parameters W2 and B2.

Then finally, the gradient descent update would be to update W1 as W1 minus Alpha.

The learning rate times D, W1.

B1 gets updated as B1 minus the learning rate, times DB1, and similarly for W2 and B2.

Sometimes, I use colon equals and sometimes equals, as either notation works fine.

So, this would be one iteration of gradient descent, and then you repeat this some number of times until your parameters look like they're converging.

So, in previous videos, we talked about how to compute the predictions, how to compute the outputs, and we saw how to do that in a vectorized way as well.

So, the key is to know how to compute these partial derivative terms, the DW1, DB1 as well as the derivatives DW2 and DB2.

So, what I'd like to do is just give you the equations you need in order to compute these derivatives.

(DESCRIPTION)
New slide, Formulas for computing derivatives.

(SPEECH)
I'll defer to the next video, which is an optional video, to go greater into Jeff about how we came up with those formulas.

So, let me just summarize again the equations for propagation.

So, you have Z1 equals W1X plus B1, and then A1 equals the activation function in that layer applied element wise as Z1, and then Z2 equals W2, A1 plus V2, and then finally, just as all vectorized across your training set, right?

A2 is equal to G2 of Z2.

Again, for now, if we assume we're doing binary classification, then this activation function really should be the sigmoid function, same just for that end neural.

So, that's the forward propagation or the left to right for computation for your neural network.

Next, let's compute the derivatives.

So, this is the back propagation step.

Then I compute DZ2 equals A2 minus the gradient of Y, and just as a reminder, all this is vectorized across examples.

So, the matrix Y is this one by M matrix that lists all of your M examples stacked horizontally.

Then it turns out DW2 is equal to this, and in fact, these first three equations are very similar to gradient descents for logistic regression.

X is equals one, comma, keep dims equals true.

Just a little detail this np.sum is a Python NumPy command for summing across one-dimension of a matrix.

In this case, summing horizontally, and what keepdims does is, it prevents Python from outputting one of those funny rank one arrays, right?

Where the dimensions was your N comma.

So, by having keepdims equals true, this ensures that Python outputs for DB a vector that is N by one.

In fact, technically this will be I guess N2 by one.

In this case, it's just a one by one number, so maybe it doesn't matter.

But later on, we'll see when it really matters.

So, so far what we've done is very similar to logistic regression.

But now as you continue to run back propagation, you will compute this, DZ2 times G1 prime of Z1.

So, this quantity G1 prime is the derivative of whether it was the activation function you use for the hidden layer, and for the output layer, I assume that you are doing binary classification with the sigmoid function.

So, that's already baked into that formula for DZ2, and his times is element-wise product.

So, this here is going to be an N1 by M matrix, and this here, this element-wise derivative thing is also going to be an N1 by N matrix, and so this times there is an element-wise product of two matrices.

Then finally, DW1 is equal to that, and DB1 is equal to this, and p.sum DZ1 axis equals one, keepdims equals true.

So, whereas previously the keepdims maybe matter less if N2 is equal to one.

Result is just a one by one thing, is just a real number.

Here, DB1 will be a N1 by one vector, and so you want Python, you want Np.sons.

I'll put something of this dimension rather than a funny rank one array of that dimension which could end up messing up some of your data calculations.

The other way would be to not have to keep the parameters, but to explicitly reshape the output of NP.sum into this dimension, which you would like DB to have.

So, that was forward propagation in I guess four equations, and back-propagation in I guess six equations.

I know I just wrote down these equations, but in the next optional video, let's go over some intuitions for how the six equations for the back propagation algorithm were derived.

Please feel free to watch that or not.

But either way, if you implement these algorithms, you will have a correct implementation of forward prop and back prop.

You'll be able to compute the derivatives you need in order to apply gradient descent, to learn the parameters of your neural network.

It is possible to implement this algorithm and get it to work without deeply understanding the calculus.

A lot of successful deep learning practitioners do so.

But, if you want, you can also watch the next video, just to get a bit more intuition of what the derivation of these equations.