Gradient Descent on m Examples

(SPEECH)
In

(DESCRIPTION)
Text, Basics of Neural Network Programming. Gradient descent of M, examples. Website, deep learning, dot, A.I.

(SPEECH)
a previous video, you saw how to compute derivatives and implement gradient descent with respect to just one training example for logistic regression.

Now, we want to do it for m training examples.

To get started, let's remind ourselves of the definition of the cost function J.

(DESCRIPTION)
New slide, Logistic regression on M, examples.

(SPEECH)
Cost- function w,b,which you care about is this average, one over m sum from i equals one through m of the loss when you algorithm output a_i on the example y, where a_i is the prediction on the ith training example which is sigma of z_i, which is equal to sigma of w transpose x_i plus b.

So, what we show in the previous slide is for any single training example, how to compute the derivatives when you have just one training example.

So dw_1, dw_2 and d_b, with now the superscript i to denote the corresponding values you get if you were doing what we did on the previous slide, but just using the one training example, x_i y_i, excuse me, missing an i there as well.

So, now you notice the overall cost functions as a sum was really average, because the one over m term of the individual losses.

So, it turns out that the derivative, respect to w_1 of the overall cost function is also going to be the average of derivatives respect to w_1 of the individual lost terms.

But previously, we have already shown how to compute this term as dw_1_i, which we, on the previous slide, show how to compute this on a single training example.

So, what you need to do is really compute these derivatives as we showed on the previous training example and average them, and this will give you the overall gradient that you can use to implement the gradient descent.

So, I know that was a lot of details, but let's take all of this up and wrap this up into a concrete algorithm until when you should implement logistic regression with gradient descent working.

So, here's what you can do: let's initialize j equals zero, dw_1 equals zero, dw_2 equals zero, d_b equals zero.

What we're going to do is use a for loop over the training set, and compute the derivative with respect to each training example and then add them up.

So, here's how we do it, for i equals one through m, so m is the number of training examples, we compute z_i equals w transpose x_i plus b.

The prediction a_i is equal to sigma of z_i, and then let's add up J, J plus equals y_i log a_i plus one minus y_i log one minus a_i, and then put the negative sign in front of the whole thing, and then as we saw earlier, we have dz_i, that's equal to a_i minus y_i, and d_w gets plus equals x1_i dz_i, dw_2 plus equals xi_2 dz_i, and I'm doing this calculation assuming that you have just two features, so that n equals to two otherwise, you do this for dw_1, dw_2, dw_3 and so on, and then db plus equals dz_i, and I guess that's the end of the for loop.

Then finally, having done this for all m training examples, you will still need to divide by m because we're computing averages.

So, dw_1 divide equals m, dw_2 divides equals m, db divide equals m, in order to compute averages.

So, with all of these calculations, you've just computed the derivatives of the cost function J with respect to each your parameters w_1, w_2 and b.

Just a couple of details about what we're doing, we're using dw_1 and dw_2 and db as accumulators, so that after this computation, dw_1 is equal to the derivative of your overall cost function with respect to w_1 and similarly for dw_2 and db.

So, notice that dw_1 and dw_2 do not have a superscript i, because we're using them in this code as accumulators to sum over the entire training set.

Whereas in contrast, dz_i here, this was dz with respect to just one single training example.

So, that's why that had a superscript i to refer to the one training example, i that is computerised.

So, having finished all these calculations, to implement one step of gradient descent, you will implement w_1, gets updated as w_1 minus the learning rate times dw_1, w_2, ends up this as w_2 minus learning rate times dw_2, and b gets updated as b minus the learning rate times db, where dw_1, dw_2 and db were as computed.

Finally, J here will also be a correct value for your cost function.

So, everything on the slide implements just one single step of gradient descent, and so you have to repeat everything on this slide multiple times in order to take multiple steps of gradient descent.

In case these details seem too complicated, again, don't worry too much about it for now, hopefully all this will be clearer when you go and implement this in the programming assignments.

But it turns out there are two weaknesses with the calculation as we've implemented it here, which is that, to implement logistic regression this way, you need to write two for loops.

The first for loop is this for loop over the m training examples, and the second for loop is a for loop over all the features over here.

So, in this example, we just had two features; so, n is equal to two and x equals two, but maybe we have more features, you end up writing here dw_1 dw_2, and you similar computations for dw_t, and so on delta dw_n.

So, it seems like you need to have a for loop over the features, over n features.

When you're implementing deep learning algorithms, you find that having explicit for loops in your code makes your algorithm run less efficiency.

So, in the deep learning era, we would move to a bigger and bigger datasets, and so being able to implement your algorithms without using explicit for loops is really important and will help you to scale to much bigger datasets.

So, it turns out that there are a set of techniques called vectorization techniques that allow you to get rid of these explicit for-loops in your code.

I think in the pre-deep learning era, that's before the rise of deep learning, vectorization was a nice to have, so you could sometimes do it to speed up your code and sometimes not.

But in the deep learning era, vectorization, that is getting rid of for loops, like this and like this, has become really important, because we're more and more training on very large datasets, and so you really need your code to be very efficient.

So, in the next few videos, we'll talk about vectorization and how to implement all this without using even a single for loop.

So, with this, I hope you have a sense of how to implement logistic regression or gradient descent for logistic regression.

Things will be clearer when you implement the programming exercise.

But before actually doing the programming exercise, let's first talk about vectorization so that you can implement this whole thing, implement a single iteration of gradient descent without using any for loops.