Explanation for Vectorized Implementation

(SPEECH)
In

(DESCRIPTION)
Text, One hidden layer Neural Network. Explanation for vectorized implementation. Website, deep learning, dot, A.I.

(SPEECH)
the previous video, we saw how with your training examples stacked up horizontally in the matrix x, you can derive a vectorized implementation for propagation through your neural network.

Let's give a bit more justification for why the equations we wrote down is a correct implementation of vectorizing across multiple examples.

(DESCRIPTION)
New slide, Justification for vectorized implementation.

(SPEECH)
So let's go through part of the propagation calculation for the few examples.

Let's say that for the first training example, you end up computing this x1 plus b1 and then for the second training example, you end up computing this x2 plus b1 and then for the third training example, you end up computing this 3 plus b1.

So, just to simplify the explanation on this slide, I'm going to ignore b.

So let's just say, to simplify this justification a little bit that b is equal to zero.

But the argument we're going to lay out will work with just a little bit of a change even when b is non-zero.

It does just simplify the description on the slide a bit.

Now, w1 is going to be some matrix, right?

So I have some number of rows in this matrix.

So if you look at this calculation x1, what you have is that w1 times x1 gives you some column vector which you must draw like this.

And similarly, if you look at this vector x2, you have that w1 times x2 gives some other column vector, right?

And that's gives you this z12.

And finally, if you look at x3, you have w1 times x3, gives you some third column vector, that's this z13.

So now, if you consider the training set capital X, which we form by stacking together all of our training examples.

So the matrix capital X is formed by taking the vector x1 and stacking it vertically with x2 and then also x3.

This is if we have only three training examples.

If you have more, you know, they'll keep stacking horizontally like that.

But if you now take this matrix x and multiply it by w then you end up with, if you think about how matrix multiplication works, you end up with the first column being these same values that I had drawn up there in purple.

The second column will be those same four values.

And the third column will be those orange values, what they turn out to be.

But of course this is just equal to z11 expressed as a column vector followed by z12 expressed as a column vector followed by z13, also expressed as a column vector.

And this is if you have three training examples.

You get more examples then there'd be more columns.

And so, this is just our matrix capital Z1.

So I hope this gives a justification for why we had previously w1 times xi equals z1i when we're looking at single training example at the time.

When you took the different training examples and stacked them up in different columns, then the corresponding result is that you end up with the z's also stacked at the columns.

And I won't show but you can convince yourself if you want that with Python broadcasting, if you add back in, these values of b to the values are still correct.

And what actually ends up happening is you end up with Python broadcasting, you end up having bi individually to each of the columns of this matrix.

So on this slide, I've only justified that z1 equals w1x plus b1 is a correct vectorization of the first step of the four steps we have in the previous slide, but it turns out that a similar analysis allows you to show that the other steps also work on using a very similar logic where if you stack the inputs in columns then after the equation, you get the corresponding outputs also stacked up in columns.

Finally, let's just recap everything we talked about in this video.

(DESCRIPTION)
New slide, recap of vectorizing across multiple examples.

(SPEECH)
If this is your neural network, we said that this is what you need to do if you were to implement for propagation, one training example at a time going from i equals 1 through m. And then we said, let's stack up the training examples in columns like so and for each of these values z1, a1, z2, a2, let's stack up the corresponding columns as follows.

So this is an example for a1 but this is true for z1, a1, z2, and a2.

Then what we show on the previous slide was that this line allows you to vectorize this across all m examples at the same time.

And it turns out with the similar reasoning, you can show that all of the other lines are correct vectorizations of all four of these lines of code.

And just as a reminder, because x is also equal to a0 because remember that the input feature vector x was equal to a0, so xi equals a0i.

Then there's actually a certain symmetry to these equations where this first equation can also be written z1 equals w1 a0 plus b1.

And so, you see that this pair of equations and this pair of equations actually look very similar but just of all of the indices advance by one.

So this kind of shows that the different layers of a neural network are roughly doing the same thing or just doing the same computation over and over.

And here we have two-layer neural network where we go to a much deeper neural network in next week's videos.

You see that even deeper neural networks are basically taking these two steps and just doing them even more times than you're seeing here.

So that's how you can vectorize your neural network across multiple training examples.

Next, we've so far been using the sigmoid functions throughout our neural networks.

It turns out that's actually not the best choice.

In the next video, let's dive a little bit further into how you can use different, what's called, activation functions of which the sigmoid function is just one possible choice.