Gradient Checking Implementation Notes

(DESCRIPTION)
Text, Setting up your optimization problem. Gradient Checking implementation notes. Website, deep learning, dot, A.I.

(SPEECH)
In the last video you learned about gradient checking.

In this video, I want to share with you some practical tips or some notes on how to actually go about implementing this for your neural

(DESCRIPTION)
New slide, Gradient checking implementation notes.

(SPEECH)
network.

First, don't use grad check in training, only to debug.

So what I mean is that, computing d theta approx i, for all the values of i, this is a very slow computation.

So to implement gradient descent, you'd use backprop to compute d theta and just use backprop to compute the derivative.

And it's only when you're debugging that you would compute this to make sure it's close to d theta.

But once you've done that, then you would turn off the grad check, and don't run this during every iteration of gradient descent, because that's just much too slow.

Second, if an algorithm fails grad check, look at the components, look at the individual components, and try to identify the bug.

So what I mean by that is if d theta approx is very far from d theta, what I would do is look at the different values of i to see which are the values of d theta approx that are really very different than the values of d theta.

So for example, if you find that the values of theta or d theta, they're very far off, all correspond to dbl for some layer or for some layers, but the components for dw are quite close, right?

Remember, different components of theta correspond to different components of b and w.

When you find this is the case, then maybe you find that the bug is in how you're computing db, the derivative with respect to parameters b.

And similarly, vice versa, if you find that the values that are very far, the values from d theta approx that are very far from d theta, you find all those components came from dw or from dw in a certain layer, then that might help you hone in on the location of the bug.

This doesn't always let you identify the bug right away, but sometimes it helps you give you some guesses about where to track down the bug.

Next, when doing grad check, remember your regularization term if you're using regularization.

So if your cost function is J of theta equals 1 over m sum of your losses and then plus this regularization term.

And sum over l of wl squared, then this is the definition of J.

And you should have that d theta is gradient of J with respect to theta, including this regularization term.

So just remember to include that term.

Next, grad check doesn't work with dropout, because in every iteration, dropout is randomly eliminating different subsets of the hidden units.

There isn't an easy to compute cost function J that dropout is doing gradient descent on.

It turns out that dropout can be viewed as optimizing some cost function J, but it's cost function J defined by summing over all exponentially large subsets of nodes they could eliminate in any iteration.

So the cost function J is very difficult to compute, and you're just sampling the cost function every time you eliminate different random subsets in those we use dropout.

So it's difficult to use grad check to double check your computation with dropouts.

So what I usually do is implement grad check without dropout.

So if you want, you can set keep-prob and dropout to be equal to 1.0.

And then turn on dropout and hope that my implementation of dropout was correct.

There are some other things you could do, like fix the pattern of nodes dropped and verify that grad check for that pattern of [INAUDIBLE] is correct, but in practice I don't usually do that.

So my recommendation is turn off dropout, use grad check to double check that your algorithm is at least correct without dropout, and then turn on dropout.

Finally, this is a subtlety.

It is not impossible, rarely happens, but it's not impossible that your implementation of gradient descent is correct when w and b are close to 0, so at random initialization.

But that as you run gradient descent and w and b become bigger, maybe your implementation of backprop is correct only when w and b is close to 0, but it gets more inaccurate when w and b become large.

So one thing you could do, I don't do this very often, but one thing you could do is run grad check at random initialization and then train the network for a while so that w and b have some time to wander away from 0, from your small random initial values.

And then run grad check again after you've trained for some number of iterations.

So that's it for gradient checking.

And congratulations for coming to the end of this week's materials.

In this week, you've learned about how to set up your train, dev, and test sets, how to analyze bias and variance and what things to do if you have high bias versus high variance versus maybe high bias and high variance.

You also saw how to apply different forms of regularization, like L2 regularization and dropout on your neural network.

So some tricks for speeding up the training of your neural network.

And then finally, gradient checking.

So I think you've seen a lot in this week and you get to exercise a lot of these ideas in this week's programming exercise.

So best of luck with that, and I look forward to seeing you in the week two materials.