Normalizing inputs

(DESCRIPTION)
Text, Setting up your optimization problem. Normalizing inputs. Website, deep learning, dot, A.I.

(SPEECH)
When training a neural network, one of the techniques that will speed up your training is if you normalize your inputs.

Let's see what that

(DESCRIPTION)
New slide, Normalizing training sets.

(SPEECH)
means.

Let's see if a training sets with two input features.

So the input features x are two dimensional, and here's a scatter plot of your training set.

Normalizing your inputs corresponds to two steps.

The first is to subtract out or to zero out the mean.

So you set mu = 1 over M sum over I of Xi.

So this is a vector, and then X gets set as X- mu for every training example, so this means you just move the training set until it has 0 mean.

And then the second step is to normalize the variances.

So notice here that the feature X1 has a much larger variance than the feature X2 here.

So what we do is set sigma = 1 over m sum of Xi**2.

I guess this is a element y squaring.

And so now sigma squared is a vector with the variances of each of the features, and notice we've already subtracted out the mean, so Xi squared, element y squared is just the variances.

And you take each example and divide it by this vector sigma squared.

And so in pictures, you end up with this.

Where now the variance of X1 and X2 are both equal to one.

And one tip, if you use this to scale your training data, then use the same mu and sigma squared to normalize your test set, right?

In particular, you don't want to normalize the training set and the test set differently.

Whatever this value is and whatever this value is, use them in these two formulas so that you scale your test set in exactly the same way, rather than estimating mu and sigma squared separately on your training set and test set.

Because you want your data, both training and test examples, to go through the same transformation defined by the same mu and sigma squared calculated on your training data.

So, why do we do this?

Why do we want to normalize the input features?

(DESCRIPTION)
New slide, Why normalize inputs?

(SPEECH)
Recall that a cost function is defined as written on the top right.

(DESCRIPTION)
Cost function formula: J, W, B, equals, 1 over M, y, hat top to the I, Y to the I.

(SPEECH)
It turns out that if you use unnormalized input features, it's more likely that your cost function will look like this, it's a very squished out bowl, very elongated cost function, where the minimum you're trying to find is maybe over there.

But if your features are on very different scales, say the feature X1 ranges from 1 to 1,000, and the feature X2 ranges from 0 to 1, then it turns out that the ratio or the range of values for the parameters w1 and w2 will end up taking on very different values.

And so maybe these axes should be w1 and w2, but I'll plot w and b, then your cost function can be a very elongated bowl like that.

So if you part the contours of this function, you can have a very elongated function like that.

Whereas if you normalize the features, then your cost function will on average look more symmetric.

And if you're running gradient descent on the cost function like the one on the left, then you might have to use a very small learning rate because if you're here that gradient descent might need a lot of steps to oscillate back and forth before it finally finds its way to the minimum.

Whereas if you have a more spherical contours, then wherever you start gradient descent can pretty much go straight to the minimum.

You can take much larger steps with gradient descent rather than needing to oscillate around like like the picture on the left.

Of course in practice w is a high-dimensional vector, and so trying to plot this in 2D doesn't convey all the intuitions correctly.

But the rough intuition that your cost function will be more round and easier to optimize when your features are all on similar scales.

Not from one to 1000, zero to one, but mostly from minus one to one or of about similar variances of each other.

That just makes your cost function J easier and faster to optimize.

In practice if one feature, say X1, ranges from zero to one, and X2 ranges from minus one to one, and X3 ranges from one to two, these are fairly similar ranges, so this will work just fine.

It's when they're on dramatically different ranges like ones from 1 to a 1000, and the another from 0 to 1, that that really hurts your authorization algorithm.

But by just setting all of them to a 0 mean and say, variance 1, like we did in the last slide, that just guarantees that all your features on a similar scale and will usually help your learning algorithm run faster.

So, if your input features came from very different scales, maybe some features are from 0 to 1, some from 1 to 1,000, then it's important to normalize your features.

If your features came in on similar scales, then this step is less important.

Although performing this type of normalization pretty much never does any harm, so I'll often do it anyway if I'm not sure whether or not it will help with speeding up training for your algebra.

So that's it for normalizing your input features.

Next, let's keep talking about ways to speed up the training of your new network.