Regularization

(DESCRIPTION)
Text, Regularizing your neural network. Regularization. Website, deep learning, dot, A.I.

(SPEECH)
If you suspect your neural network is over fitting your data.

That is you have a high variance problem, one of the first things you should try per probably regularization.

The other way to address high variance, is to get more training data that's also quite reliable.

But you can't always get more training data, or it could be expensive to get more data.

But adding regularization will often help to prevent overfitting, or to reduce the errors in your network.

So let's see how regularization works.

(DESCRIPTION)
New slide, Logistic regression. Formula, min, W, B, underneath, J, W, B.

(SPEECH)
Let's develop these ideas using logistic regression.

Recall that for logistic regression, you try to minimize the cost function J, which is defined as this cost function.

Some of your training examples of the losses of the individual predictions in the different examples, where you recall that w and b in the logistic regression, are the parameters.

So w is an x-dimensional parameter vector, and b is a real number.

And so to add regularization to the logistic regression, what you do is add to it this thing, lambda, which is called the regularization parameter.

I'll say more about that in a second.

But lambda/2m times the norm of w squared.

So here, the norm of w squared is just equal to sum from j equals 1 to nx of wj squared, or this can also be written w transpose w, it's just a square Euclidean norm of the prime to vector w.

And this is called L2 regularization.

Because here, you're using the Euclidean normals, or else the L2 norm with the prime to vector w.

Now, why do you regularize just the parameter w?

Why don't we add something here about b as well?

In practice, you could do this, but I usually just omit this.

Because if you look at your parameters, w is usually a pretty high dimensional parameter vector, especially with a high variance problem.

Maybe w just has a lot of parameters, so you aren't fitting all the parameters well, whereas b is just a single number.

So almost all the parameters are in w rather b.

And if you add this last term, in practice, it won't make much of a difference, because b is just one parameter over a very large number of parameters.

In practice, I usually just don't bother to include it.

But you can if you want.

So L2 regularization is the most common type of regularization.

You might have also heard of some people talk about L1 regularization.

And that's when you add, instead of this L2 norm, you instead add a term that is lambda/m of sum over of this.

And this is also called the L1 norm of the parameter vector w, so the little subscript 1 down there, right?

And I guess whether you put m or 2m in the denominator, is just a scaling constant.

If you use L1 regularization, then w will end up being sparse.

And what that means is that the w vector will have a lot of zeros in it.

And some people say that this can help with compressing the model, because the set of parameters are zero, and you need less memory to store the model.

Although, I find that, in practice, L1 regularization to make your model sparse, helps only a little bit.

So I don't think it's used that much, at least not for the purpose of compressing your model.

And when people train your networks, L2 regularization is just used much much more often.

Sorry, just fixing up some of the notation here.

So one last detail.

Lambda here is called the regularization, Parameter.

And usually, you set this using your development set, or using [INAUDIBLE] cross validation.

When you a variety of values and see what does the best, in terms of trading off between doing well in your training set versus also setting that two normal of your parameters to be small.

Which helps prevent over fitting.

So lambda is another hyper parameter that you might have to tune.

And by the way, for the programming exercises, lambda is a reserved keyword in the Python programming language.

So in the programming exercise, we'll have lambd, without the a, so as not to clash with the reserved keyword in Python.

So we use lambd to represent the lambda regularization parameter.

So this is how you implement L2 regularization for logistic

(DESCRIPTION)
New slide, Neural network.

(SPEECH)
regression.

How about a neural network?

In a neural network, you have a cost function that's a function of all of your parameters, w[1], b[1] through w[L], b[L], where capital L is the number of layers in your neural network.

And so the cost function is this, sum of the losses, summed over your m training examples.

And says at regularization, you add lambda over 2m of sum over all of your parameters W, your parameter matrix is w, of their, that's called the squared norm.

Where this norm of a matrix, meaning the squared norm is defined as the sum of the i sum of j, of each of the elements of that matrix, squared.

And if you want the indices of this summation.

This is sum from i=1 through n[l-1].

Sum from j=1 through n[l], because w is an n[l-1] by n[l] dimensional matrix, where these are the number of units in layers [l-1] in layer l.

So this matrix norm, it turns out is called the Frobenius norm of the matrix, denoted with a F in the subscript.

So for arcane linear algebra technical reasons, this is not called the l2 normal of a matrix.

Instead, it's called the Frobenius norm of a matrix.

I know it sounds like it would be more natural to just call the l2 norm of the matrix, but for really arcane reasons that you don't need to know, by convention, this is called the Frobenius norm.

It just means the sum of square of elements of a matrix.

So how do you implement gradient descent with this?

Previously, we would complete dw using backprop, where backprop would give us the partial derivative of J with respect to w, or really w for any given [l].

And then you update w[l], as w[l]- the learning rate times d.

So this is before we added this extra regularization term to the objective.

Now that we've added this regularization term to the objective, what you do is you take dw and you add to it, lambda/m times w.

And then you just compute this update, same as before.

And it turns out that with this new definition of dw[l], this new dw[l] is still a correct definition of the derivative of your cost function, with respect to your parameters, now that you've added the extra regularization term at the end.

And it's for this reason that L2 regularization is sometimes also called weight decay.

So if I take this definition of dw[l] and just plug it in here, then you see that the update is w[l] = w[l] times the learning rate alpha times the thing from backprop, +lambda of m times w[l].

Throw the minus sign there.

And so this is equal to w[l]- alpha lambda / m times w[l]- alpha times the thing you got from backpop.

And so this term shows that whatever the matrix w[l] is, you're going to make it a little bit smaller, right?

This is actually as if you're taking the matrix w and you're multiplying it by 1-alpha lambda/m.

You're really taking the matrix w and subtracting alpha lambda/m times this.

Like you're multiplying matrix w by this number, which is going to be a little bit less than 1.

So this is why L2 norm regularization is also called weight decay.

Because it's just like the ordinally gradient descent, where you update w by subtracting alpha times the original gradient you got from backprop.

But now you're also multiplying w by this thing, which is a little bit less than 1.

So the alternative name for L2 regularization is weight decay.

I'm not really going to use that name, but the intuition for it's called weight decay is that this first term here, is equal to this.

So you're just multiplying the weight metrics by a number slightly less than 1.

So that's how you implement L2 regularization in neural network.

Now, one question that [INAUDIBLE] has asked me is, hey, Andrew, why does regularization prevent over-fitting?

Let's look at the next video, and gain some intuition for how regularization prevents over-fitting.