Dropout Regularization

(SPEECH)
In

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

(SPEECH)
addition to L2 regularization, another very powerful regularization techniques is called "dropout." Let's see how that works.

(DESCRIPTION)
New slide, Dropout regularization.

(SPEECH)
Let's say you train a neural network like the one on the left and there's over-fitting.

Here's what you do with dropout.

Let me make a copy of the neural network.

With dropout, what we're going to do is go through each of the layers of the network and set some probability of eliminating a node in neural network.

Let's say that for each of these layers, we're going to- for each node, toss a coin and have a 0.5 chance of keeping each node and 0.5 chance of removing each node.

So, after the coin tosses, maybe we'll decide to eliminate those nodes, then what you do is actually remove all the outgoing things from that no as well.

So you end up with a much smaller, really much diminished network.

And then you do back propagation training.

There's one example on this much diminished network.

And then on different examples, you would toss a set of coins again and keep a different set of nodes and then dropout or eliminate different than nodes.

And so for each training example, you would train it using one of these neural based networks.

So, maybe it seems like a slightly crazy technique.

They just go around coding those are random, but this actually works.

But you can imagine that because you're training a much smaller network on each example or maybe just give a sense for why you end up able to regularize the network, because these much smaller networks are being trained.

Let's look at how you implement dropout.

(DESCRIPTION)
New slide, Implementing dropout, Inverted dropout.

(SPEECH)
There are a few ways of implementing dropout.

I'm going to show you the most common one, which is technique called inverted dropout.

For the sake of completeness, let's say we want to illustrate this with layer l=3.

So, in the code I'm going to write- there will be a bunch of 3s here.

I'm just illustrating how to represent dropout in a single layer.

So, what we are going to do is set a vector d and d^3 is going to be the dropout vector for the layer 3.

That's what the 3 is to be np.random.rand(a).

And this is going to be the same shape as a3.

And when I see if this is less than some number, which I'm going to call keep.prob.

And so, keep.prob is a number.

It was 0.5 on the previous time, and maybe now I'll use 0.8 in this example, and there will be the probability that a given hidden unit will be kept.

So keep.prob = 0.8, then this means that there's a 0.2 chance of eliminating any hidden unit.

So, what it does is it generates a random matrix.

And this works as well if you have factorized.

So d3 will be a matrix.

Therefore, each example have a each hidden unit there's a 0.8 chance that the corresponding d3 will be one, and a 20% chance there will be zero.

So, this random numbers being less than 0.8 it has a 0.8 chance of being one or be true, and 20% or 0.2 chance of being false, of being zero.

And then what you are going to do is take your activations from the third layer, let me just call it a3 in this low example.

So, a3 has the activations you computate.

And you can set a3 to be equal to the old a3, times- There is element wise multiplication.

Or you can also write this as a3* = d3.

But what this does is for every element of d3 that's equal to zero.

And there was a 20% chance of each of the elements being zero, just multiply operation ends up zeroing out, the corresponding element of d3.

If you do this in python, technically d3 will be a boolean array where value is true and false, rather than one and zero.

But the multiply operation works and will interpret the true and false values as one and zero.

If you try this yourself in python, you'll see.

Then finally, we're going to take a3 and scale it up by dividing by 0.8 or really dividing by our keep.prob parameter.

So, let me explain what this final step is doing.

Let's say for the sake of argument that you have 50 units or 50 neurons in the third hidden layer.

So maybe a3 is 50 by one dimensional or if you- factorization maybe it's 50 by m dimensional.

So, if you have a 80% chance of keeping them and 20% chance of eliminating them.

This means that on average, you end up with 10 units shut off or 10 units zeroed out.

And so now, if you look at the value of z^4, z^4 is going to be equal to w^4 * a^3 + b^4.

And so, on expectation, this will be reduced by 20%.

By which I mean that 20% of the elements of a3 will be zeroed out.

So, in order to not reduce the expected value of z^4, what you do is you need to take this, and divide it by 0.8 because this will correct or just a bump that back up by roughly 20% that you need.

So it's not changed the expected value of a3.

And, so this line here is what's called the inverted dropout technique.

And its effect is that, no matter what you set to keep.prob to, whether it's 0.8 or 0.9 or even one, if it's set to one then there's no dropout, because it's keeping everything or 0.5 or whatever, this inverted dropout technique by dividing by the keep.prob, it ensures that the expected value of a3 remains the same.

And it turns out that at test time, when you trying to evaluate a neural network, which we'll talk about on the next slide, this inverted dropout technique, there is there is line to are due to the green box at dropping out.

This makes test time easier because you have less of a scaling problem.

By far the most common implementation of dropouts today as far as I know is inverted dropouts.

I recommend you just implement this.

But there were some early iterations of dropout that missed this divide by keep.prob line, and so at test time the average becomes more and more complicated.

But again, people tend not to use those other versions.

So, what you do is you use the d vector, and you'll notice that for different training examples, you zero out different hidden units.

And in fact, if you make multiple passes through the same training set, then on different pauses through the training set, you should randomly zero out different hidden units.

So, it's not that for one example, you should keep zeroing out the same hidden units is that, on iteration one of grade and descent, you might zero out some hidden units.

And on the second iteration of great descent where you go through the training set the second time, maybe you'll zero out a different pattern of hidden units.

And the vector d or d3, for the third layer, is used to decide what to zero out, both in for prob as well as in that prob.

We are just showing for prob here.

Now,

(DESCRIPTION)
New slide, Making predictions at test time.

(SPEECH)
having trained the algorithm at test time, here's what you would do.

At test time, you're given some x or which you want to make a prediction.

And using our standard notation, I'm going to use a^0, the activations of the zeroes layer to denote just test example x.

So what we're going to do is not to use dropout at test time in particular which is in a sense.

$Z^1= w^1.a^0 + b^1.$

$a^1 = g^1(z^1 Z).$

$Z^2 = w^2.a^1 + b^2.$

$a^2 =...$

And so on. Until you get to the last layer and that you make a prediction y^.

But notice that the test time you're not using dropout explicitly and you're not tossing coins at random, you're not flipping coins to decide which hidden units to eliminate.

And that's because when you are making predictions at the test time, you don't really want your output to be random.

If you are implementing dropout at test time, that just add noise to your predictions.

In theory, one thing you could do is run a prediction process many times with different hidden units randomly dropped out and have it across them.

But that's computationally inefficient and will give you roughly the same result; very, very similar results to this different procedure as well.

And just to mention, the inverted dropout thing, you remember the step on the previous line when we divided by the cheap.prob.

The effect of that was to ensure that even when you don't see men dropout at test time to the scaling, the expected value of these activations don't change.

So, you don't need to add in an extra funny scaling parameter at test time.

That's different than when you have that training time.

So that's dropouts.

And when you implement this in week's premier exercise, you gain more firsthand experience with it as well.

But why does it really work?

What I want to do the next video is give you some better intuition about what dropout really is doing.

Let's go on to the next video.