Understanding Dropout

(SPEECH)
Drop

(DESCRIPTION)
Regularizing your neural network. Understanding dropout. Website, deep learning, dot, A.I.

(SPEECH)
out does this seemingly crazy thing of randomly knocking out units on your network.

Why does it work so well with a regularizer?

Let's gain some better

(DESCRIPTION)
New slide, Why does drop-out work? Text, Intuition: Can't rely on any one feature, so have to spread out weights.

(SPEECH)
intuition.

In the previous video, I gave this intuition that drop-out randomly knocks out units in your network.

So it's as if on every iteration you're working with a smaller neural network, and so using a smaller neural network seems like it should have a regularizing effect.

Here's a second intuition which is, let's look at it from the perspective of a single unit. Let's say this one.

Now, for this unit to do his job as for inputs and it needs to generate some meaningful output.

Now with drop out, the inputs can get randomly eliminated.

Sometimes those two units will get eliminated, sometimes a different unit will get eliminated.

So, what this means is that this unit, which I'm circling in purple, it can't rely on any one feature because any one feature could go away at random or any one of its own inputs could go away at random.

Some particular would be reluctant to put all of its bets on, say, just this input, right?

The weights, we're reluctant to put too much weight on any one input because it can go away.

So this unit will be more motivated to spread out this way and give you a little bit of weight to each of the four inputs to this unit.

And by spreading all the weights, this will tend to have an effect of shrinking the squared norm of the weights.

And so, similar to what we saw with L2 regularization, the effect of implementing drop out is that it shrinks the weights and does some of those outer regularization that helps prevent over-fitting.

But it turns out that drop out can formally be shown to be an adaptive form without a regularization.

But L2 penalty on different weights are different, depending on the size of the activations being multiplied that way.

But to summarize, it is possible to show that drop out has a similar effect to L2 regularization.

Only to L2 regularization applied to different ways can be a little bit different and even more adaptive to the scale of different inputs.

One more detail for when you're implementing drop out.

Here's a network where you have three input features.

This is seven hidden units here, seven, three, two, one.

So, one of the parameters we had to choose was the cheap prop which has a chance of keeping a unit in each layer.

So, it is also feasible to vary key prop by layer.

So for the first layer, your matrix W1 will be three by seven.

Your second weight matrix will be seven by seven.

W3 will be seven by three and so on.

And so W2 is actually the biggest weight matrix, because they're actually the largest set of parameters would be in W2 which is seven by seven.

So to prevent, to reduce over-fitting of that matrix, maybe for this layer, I guess this is layer two, you might have a key prop that's relatively low, say zero point five, whereas for different layers where you might worry less about over-fitting, you could have a higher key prop, maybe just zero point seven.

And if a layers we don't worry about over-fitting at all, you can have a key prop of one point zero.

For clarity, these are numbers I'm drawing on the purple boxes.

These could be different key props for different layers.

Notice that the key prop of one point zero means that you're keeping every unit and so, you're really not using drop out for that layer.

But for layers where you're more worried about over-fitting, really the layers with a lot of parameters, you can set the key prop to be smaller to apply a more powerful form of drop out.

It's kind of like cranking up the regularization parameter lambda of L2 regularization where you try to regularize some layers more than others.

And technically, you can also apply drop out to the input layer, where you can have some chance of just maxing out one or more of the input features.

Although in practice, usually don't do that that often.

And so, a key prop of one point zero was quite common for the input there.

You can also use a very high value, maybe zero point nine, but it's much less likely that you want to eliminate half of the input features.

So usually key prop, if you apply the law, will be a number close to one if you even apply drop out at all to the input there.

So just to summarize, if you're more worried about some layers overfitting than others, you can set a lower key prop for some layers than others.

The downside is, this gives you even more hyper parameters to search for using cross-validation.

One other alternative might be to have some layers where you apply drop out and some layers where you don't apply drop out and then just have one hyper parameter, which is a key prop for the layers for which you do apply drop outs.

And before we wrap up, just a couple implementational tips.

Many of the first successful implementations of drop outs were to computer vision.

So in computer vision, the input size is so big, inputting all these pixels that you almost never have enough data.

And so drop out is very frequently used by computer vision.

And there's some computer vision researchers that pretty much always use it, almost as a default.

But really the thing to remember is that drop out is a regularization technique, it helps prevent over-fitting.

And so, unless my algorithm is over-fitting, I wouldn't actually bother to use drop out.

So it's used somewhat less often than other application areas.

There's just with computer vision, you usually just don't have enough data, so you're almost always overfitting, which is why there tends to be some computer vision researchers who swear by drop out.

But their intuition doesn't always generalize I think to other disciplines.

One big downside of drop out is that the cost function J is no longer well-defined.

On every iteration, you are randomly killing off a bunch of nodes.

And so, if you are double checking the performance of grade and dissent, it's actually harder to double check that you have a well defined cost function J that is going downhill on every iteration.

Because the cost function J that you're optimizing is actually less.

Less well defined, or is certainly hard to calculate.

So you lose this debugging tool to will a plot, a graph like this.

So what I usually do is turn off drop out, you will set key prop equals one, and I run my code and make sure that it is monotonically decreasing J, and then turn on drop out and hope that I didn't introduce bugs into my code during drop out.

Because you need other ways, I guess, but not plotting these figures to make sure that your code is working to greatness and it's working even with drop outs.

So with that, there's still a few more regularization techniques that are worth your knowing.

Let's talk about a few more such techniques in the next video.