Using an appropriate scale to pick hyperparameters

(DESCRIPTION)
Text, Hyperparameter tuning. Using an appropriate scale to pick hyperparameters. Website, deep learning, dot, A.I.

(SPEECH)
In the last video, you saw how sampling at random, over the range of hyperparameters, can allow you to search over the space of hyperparameters more efficiently.

But it turns out that sampling at random doesn't mean sampling uniformly at random, over the range of valid values.

Instead, it's important to pick the appropriate scale on which to explore the hyperparamaters.

In this video, I want to show you how to

(DESCRIPTION)
New slide, Picking hyperparameters at random.

(SPEECH)
do that.

Let's say that you're trying to choose the number of hidden units, n[l], for a given layer l.

And let's say that you think a good range of values is somewhere from 50 to 100.

In that case, if you look at the number line from 50 to 100, maybe picking some number values at random within this number line.

There's a pretty visible way to search for this particular hyperparameter.

Or if you're trying to decide on the number of layers in your neural network, we're calling that capital L.

Maybe you think the total number of layers should be somewhere between 2 to 4.

Then sampling uniformly at random, along 2, 3 and 4, might be reasonable.

Or even using a grid search, where you explicitly evaluate the values 2, 3 and 4 might be reasonable.

So these were a couple examples where sampling uniformly at random over the range you're contemplating, might be a reasonable thing to do.

But this is not true for all hyperparameters.

(DESCRIPTION)
New slide, Appropriate scale for hyperparameters.

(SPEECH)
Let's look at another example.

Say your searching for the hyperparameter alpha, the learning rate.

And let's say that you suspect 0.0001 might be on the low end, or maybe it could be as high as 1.

Now if you draw the number line from 0.0001 to 1, and sample values uniformly at random over this number line.

Well about 90% of the values you sample would be between 0.1 and 1.

So you're using 90% of the resources to search between 0.1 and 1, and only 10% of the resources to search between 0.0001 and 0.1.

So that doesn't seem right.

Instead, it seems more reasonable to search for hyperparameters on a log scale.

Where instead of using a linear scale, you'd have 0.0001 here, and then 0.001, 0.01, 0.1, and then 1.

And you instead sample uniformly, at random, on this type of logarithmic scale.

Now you have more resources dedicated to searching between 0.0001 and 0.001, and between 0.001 and 0.01, and so on.

So in Python, the way you implement this, is let r = -4 * np.random.rand().

And then a randomly chosen value of alpha, would be alpha = 10 to the power of r.

So after this first line, r will be a random number between -4 and 0.

And so alpha here will be between 10 to the -4 and 10 to the 0.

So 10 to the -4 is this left thing, this 10 to the -4.

And 1 is 10 to the 0.

In a more general case, if you're trying to sample between 10 to the a, to 10 to the b, on the log scale.

And in this example, this is 10 to the a.

And you can figure out what a is by taking the log base 10 of 0.0001, which is going to tell you a is -4.

And this value on the right, this is 10 to the b.

And you can figure out what b is, by taking log base 10 of 1, which tells you b is equal to 0.

So what you do, is then sample r uniformly, at random, between a and b.

So in this case, r would be between -4 and 0.

And you can set alpha, on your randomly sampled hyperparameter value, as 10 to the r, okay?

So just to recap, to sample on the log scale, you take the low value, take logs to figure out what is a.

Take the high value, take a log to figure out what is b.

So now you're trying to sample, from 10 to the a to the b, on a log scale.

So you set r uniformly, at random, between a and b.

And then you set the hyperparameter to be 10 to the r.

So that's how you implement sampling on this logarithmic scale.

(DESCRIPTION)
New slide, Hyperparameters for exponentially weighted averages.

(SPEECH)
Finally, one other tricky case is sampling the hyperparameter beta, used for computing exponentially weighted averages.

So let's say you suspect that beta should be somewhere between 0.9 to 0.999.

Maybe this is the range of values you want to search over.

So remember, that when computing exponentially weighted averages, using 0.9 is like averaging over the last 10 values.

kind of like taking the average of 10 days temperature, whereas using 0.999 is like averaging over the last 1,000 values.

So similar to what we saw on the last slide, if you want to search between 0.9 and 0.999, it doesn't make sense to sample on the linear scale, right?

Uniformly, at random, between 0.9 and 0.999.

So the best way to think about this, is that we want to explore the range of values for 1 minus beta, which is going to now range from 0.1 to 0.001.

And so we'll sample the between beta, taking values from 0.1, to maybe 0.1, to 0.001.

So using the method we have figured out on the previous slide, this is 10 to the -1, this is 10 to the -3.

Notice on the previous slide, we had the small value on the left, and the large value on the right, but here we have reversed.

We have the large value on the left, and the small value on the right.

So what you do, is you sample r uniformly, at random, from -3 to -1.

And you set 1- beta = 10 to the r, and so beta = 1- 10 to the r.

And this becomes your randomly sampled value of your hyperparameter, chosen on the appropriate scale.

And hopefully this makes sense, in that this way, you spend as much resources exploring the range 0.9 to 0.99, as you would exploring 0.99 to 0.999.

So if you want to study more formal mathematical justification for why we're doing this, right, why is it such a bad idea to sample in a linear scale?

It is that, when beta is close to 1, the sensitivity of the results you get changes, even with very small changes to beta.

So if beta goes from 0.9 to 0.9005, it's no big deal, this is hardly any change in your results.

But if beta goes from 0.999 to 0.9995, this will have a huge impact on exactly what your algorithm is doing, right?

In both of these cases, it's averaging over roughly 10 values.

But here it's gone from an exponentially weighted average over about the last 1,000 examples, to now, the last 2,000 examples.

And it's because that formula we have, 1 / 1- beta, this is very sensitive to small changes in beta, when beta is close to 1.

So what this whole sampling process does, is it causes you to sample more densely in the region of when beta is close to 1.

Or, alternatively, when 1- beta is close to 0.

So that you can be more efficient in terms of how you distribute the samples, to explore the space of possible outcomes more efficiently.

So I hope this helps you select the right scale on which to sample the hyperparameters.

In case you don't end up making the right scaling decision on some hyperparameter choice, don't worry to much about it.

Even if you sample on the uniform scale, where sum of the scale would have been superior, you might still get okay results.

Especially if you use a coarse to fine search, so that in later iterations, you focus in more on the most useful range of hyperparameter values to sample.

I hope this helps you in your hyperparameter search.

In the next video, I also want to share with you some thoughts of how to organize your hyperparameter search process.

That I hope will make your workflow a bit more efficient.