Train : Dev : Test sets

(DESCRIPTION)
Text, Setting up your M.L. application. Train/Dev/test sets. Website, deep learning, dot, A.I.

(SPEECH)
Welcome to this course on the practical aspects of deep learning.

Perhaps now you've learned how to implement a neural network.

In this week you'll learn the practical aspects of how to make your neural network work well.

Ranging from things like hyperparameter tuning to how to set up your data, to how to make sure your optimization algorithm runs quickly so that you get your learning algorithm to learn in a reasonable time.

In this first week we'll first talk about the cellular machine learning problem, then we'll talk about randomization.

And we'll talk about some tricks for making sure your neural network implementation is correct.

With that, let's get started.

Making good choices in how you set up your training, development, and test sets can make a huge difference in helping you quickly find a good high performance neural network.

(DESCRIPTION)
New slide, Applied M.L. is a highly iterative process.

(SPEECH)
When training a neural network you have to make a lot of decisions, such as how many layers will your neural network have?

And how many hidden units do you want each layer to have?

And what's the learning rates?

And what are the activation functions you want to use for the different layers?

When you're starting on a new application, it's almost impossible to correctly guess the right values for all of these, and for other hyperparameter choices, on your first attempt.

So in practice applied machine learning is a highly iterative process, in which you often start with an idea, such as you want to build a neural network of a certain number of layers, certain number of hidden units, maybe on certain data sets and so on.

And then you just have to code it up and try it by running your code.

You run and experiment and you get back a result that tells you how well this particular network, or this particular configuration works.

And based on the outcome, you might then refine your ideas and change your choices and maybe keep iterating in order to try to find a better and a better neural network.

Today, deep learning has found great success in a lot of areas.

Ranging from natural language processing to computer vision to speech recognition to a lot of applications on also structured data.

And structured data includes everything from advertisements to web search, which isn't just Internet search engines it's also, for example, shopping websites.

Already any websites that wants deliver great search results when you enter terms into a search bar.

To computer security, to logistics, such as figuring out where to send drivers to pick up and drop off things, to many more.

So what I'm seeing is that sometimes a researcher with a lot of experience in NLP might try to do something in computer vision.

Or maybe a researcher with a lot of experience in speech recognition might jump in and try to do something on advertising.

Or someone from security might want to jump in and do something on logistics.

And what I've seen is that intuitions from one domain or from one application area often do not transfer to other application areas.

And the best choices may depend on the amount of data you have, the number of input features you have through your computer configuration and whether you're training on GPUs or CPUs.

And if so, exactly what configuration of GPUs and CPUs, and many other things.

So for a lot of applications I think it's almost impossible.

Even very experienced deep learning people find it almost impossible to correctly guess the best choice of hyperparameters the very first time.

And so today, applied deep learning is a very iterative process where you just have to go around this cycle many times to hopefully find a good choice of network for your application.

So one of the things that determine how quickly you can make progress is how efficiently you can go around this cycle.

(DESCRIPTION)
New slide, Train/dev/test sets.

(SPEECH)
And setting up your data sets well in terms of your train, development and test sets can make you much more efficient at that.

So if this is your training data, let's draw that as a big box.

Then traditionally you might take all the data you have and carve off some portion of it to be your training set.

Some portion of it to be your hold-out cross validation set, and this is sometimes also called the development set.

And for brevity I'm just going to call this the dev set, but all of these terms mean roughly the same thing.

And then you might carve out some final portion of it to be your test set.

And so the workflow is that you keep on training algorithms on your training sets.

And use your dev set or your hold-out cross validation set to see which of many different models performs best on your dev set.

And then after having done this long enough, when you have a final model that you want to evaluate, you can take the best model you have found and evaluate it on your test set.

In order to get an unbiased estimate of how well your algorithm is doing.

So in the previous era of machine learning, it was common practice to take all your data and split it according to maybe a 70/30% in terms of a people often talk about the 70/30 train test splits.

If you don't have an explicit dev set or maybe a 60/20/20% split in terms of 60% train, 20% dev and 20% test.

And several years ago this was widely considered best practice in machine learning.

If you have maybe 100 examples in total, maybe 1000 examples in total, maybe after 10,000 examples.

These sorts of ratios were perfectly reasonable rules of thumb.

But in the modern big data era, where, for example, you might have a million examples in total, then the trend is that your dev and test sets have been becoming a much smaller percentage of the total.

Because remember, the goal of the dev set or the development set is that you're going to test different algorithms on it and see which algorithm works better.

So the dev set just needs to be big enough for you to evaluate, say, two different algorithm choices or ten different algorithm choices and quickly decide which one is doing better.

And you might not need a whole 20% of your data for that.

So, for example, if you have a million training examples you might decide that just having 10,000 examples in your dev set is more than enough to evaluate which one or two algorithms does better.

And in a similar vein, the main goal of your test set is, given your final classifier, to give you a pretty confident estimate of how well it's doing.

And again, if you have a million examples maybe you might decide that 10,000 examples is more than enough in order to evaluate a single classifier and give you a good estimate of how well it's doing.

So in this example where you have a million examples, if you need just 10,000 for your dev and 10,000 for your test, your ratio will be more like this 10,000 is 1% of 1 million so you'll have 98% train, 1% dev, 1% test.

And I've also seen applications where, if you have even more than a million examples, you might end up with 99.5% train and 0.25% dev, 0.25% test.

Or maybe a 0.4% dev, 0.1% test.

So just to recap, when setting up your machine learning problem, I'll often set it up into a train, dev and test sets, and if you have a relatively small dataset, these traditional ratios might be okay.

But if you have a much larger data set, it's also fine to set your dev and test sets to be much smaller than your 20% or even 10% of your data.

We'll give more specific guidelines on the sizes of dev and test sets later in this specialization.

(DESCRIPTION)
New slide, Mismatched train/test distribution.

(SPEECH)
One other trend we're seeing in the era of modern deep learning is that more and more people train on mismatched train and test distributions.

Let's say you're building an app that lets users upload a lot of pictures and your goal is to find pictures of cats in order to show your users.

Maybe all your users are cat lovers.

Maybe your training set comes from cat pictures downloaded off the Internet, but your dev and test sets might comprise cat pictures from users using our app.

So maybe your training set has a lot of pictures crawled off the Internet but the dev and test sets are pictures uploaded by users.

Turns out a lot of webpages have very high resolution, very professional, very nicely framed pictures of cats.

But maybe your users are uploading blurrier, lower res images just taken with a cell phone camera in a more casual condition.

And so these two distributions of data may be different.

The rule of thumb I'd encourage you to follow in this case is to make sure that the dev and test sets come from the same distribution.

We'll say more about this particular guideline as well, but because you will be using the dev set to evaluate a lot of different models and trying really hard to improve performance on the dev set.

It's nice if your dev set comes from the same distribution as your test set.

But because deep learning algorithms have such a huge hunger for training data, one trend I'm seeing is that you might use all sorts of creative tactics, such as crawling webpages, in order to acquire a much bigger training set than you would otherwise have.

Even if part of the cost of that is then that your training set data might not come from the same distribution as your dev and test sets.

But you find that so long as you follow this rule of thumb, that progress in your machine learning algorithm will be faster.

And I'll give a more detailed explanation for this particular rule of thumb later in the specialization as well.

Finally, it might be okay to not have a test set.

Remember the goal of the test set is to give you a unbiased estimate of the performance of your final network, of the network that you selected.

But if you don't need that unbiased estimate, then it might be okay to not have a test set.

So what you do, if you have only a dev set but not a test set, is you train on the training set and then you try different model architectures.

Evaluate them on the dev set, and then use that to iterate and try to get to a good model.

Because you've fit your data to the dev set, this no longer gives you an unbiased estimate of performance.

But if you don't need one, that might be perfectly fine.

In the machine learning world, when you have just a train and a dev set but no separate test set.

Most people will call this a training set and they will call the dev set the test set.

But what they actually end up doing is using the test set as a hold-out cross validation set.

Which maybe isn't completely a great use of terminology, because they're then overfitting to the test set.

So when the team tells you that they have only a train and a test set, I would just be cautious and think, do they really have a train dev set?

Because they're overfitting to the test set.

Culturally, it might be difficult to change some of these team's terminology and get them to call it a trained dev set rather than a trained test set.

Even though I think calling it a train and development set would be more correct terminology.

And this is actually okay practice if you don't need a completely unbiased estimate of the performance of your algorithm.

So having set up a train dev and test set will allow you to integrate more quickly.

It will also allow you to more efficiently measure the bias and variance of your algorithm so you can more efficiently select ways to improve your algorithm.

Let's start to talk about that in the next video.