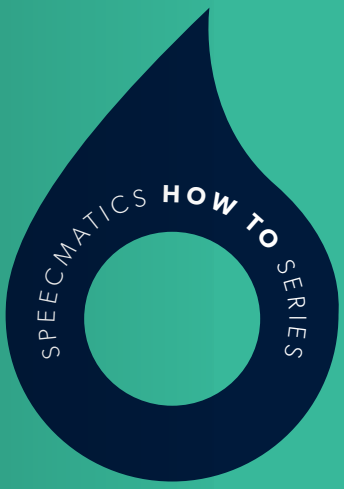
SPEECHMATICS



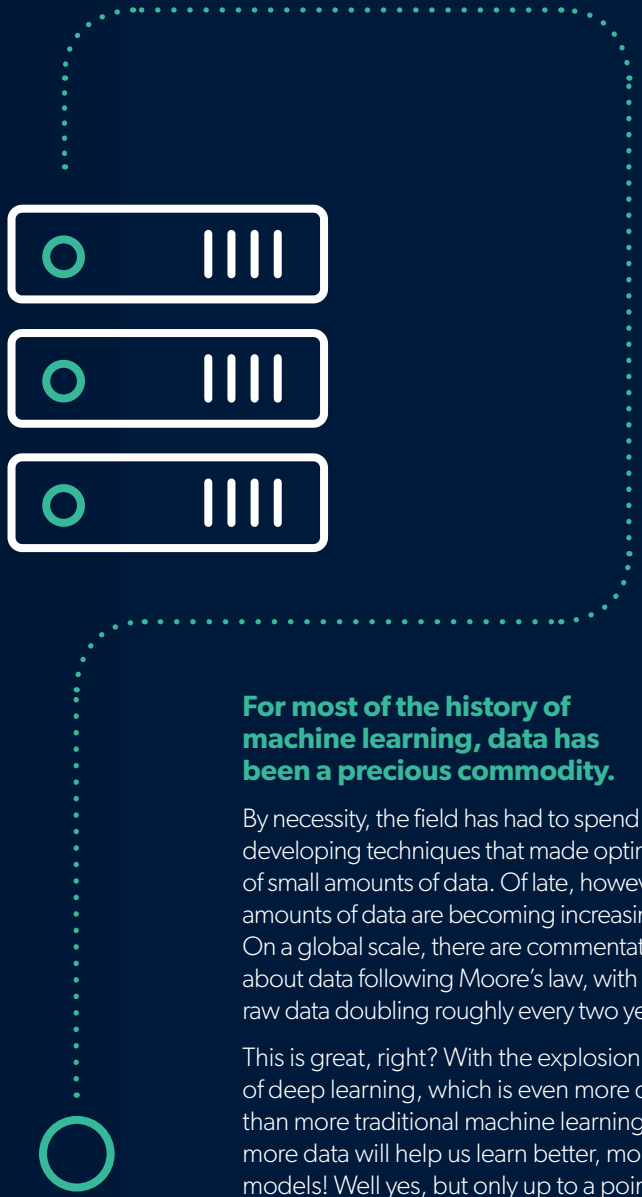
HOW TO

make the most of data surplus

WWW.SPEECHMATICS.COM

Part 1

What to do with all that data?



For most of the history of machine learning, data has been a precious commodity.

By necessity, the field has had to spend time developing techniques that made optimal use of small amounts of data. Of late, however, large amounts of data are becoming increasingly available. On a global scale, there are commentators talking about data following Moore's law, with the amount of raw data doubling roughly every two years.

This is great, right? With the explosion in the use of deep learning, which is even more data hungry than more traditional machine learning methods, more data will help us learn better, more nuanced models! Well yes, but only up to a point.

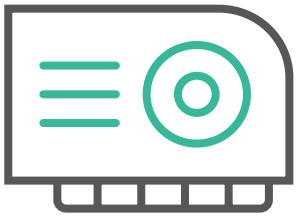
The CPUs cannae take it, Captain!

The immediate problem is that of computational resource. The demands of modern deep learning systems have already meant that traditional CPUs are insufficient for the task of training models. There are also signs that Moore's law for computational resource is slowing down, with physical limitations starting to have an impact and costs have been rising (see Rock's law) even as compute gains are released.

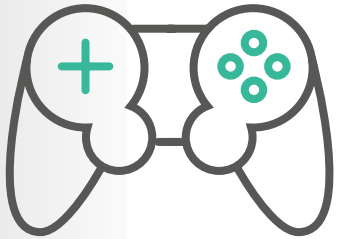
The computational limits have been circumvented to some degree by mass parallelism of parts of model training by using GPUs-graphics cards developed originally to produce cutting-edge video game visuals but now repurposed as the core engine driving the latest AI revolution. There are now even custom build tensor processing units that Google has made available on their cloud platforms, with academic papers reporting results of models built on hundreds of these devices.

EXPERT OPINION

Even hardware accelerants have their limits.



However, even these hardware accelerants have their limits. Amdahl's law tells us that speeding up one individual part of your compute pipeline will start to have diminishing returns fairly rapidly, so even if more speculative technologies such as quantum computing come along, we will still have a bottleneck somewhere (disk I/O, or data transfer from standard CPU to the hardware accelerator are often candidates, in our experience). At some point, the sheer mass of data available to an AI engineer will overwhelm their available compute resources. I won't predict exactly when that will happen-predictions on technological topics tend to come back and haunt their creators-but I believe it will come.



So far, I have just talked about training.

At inference time, more problems crop up. In order to make better use of more data, typically we increase our model sizes, so we have more parameters to better learn the data. That then leads to a bigger computational footprint when those models are in use. At a time when AI is trying to break into lower specification and low powered devices in the Internet of Things and mobile phones, larger models that require more power are not welcome.

Does all that data even help?

The previous discussion has the implicit assumption that more data leads to better models and artificial intelligence systems. This has led to the presumption that always throwing more data at a problem will lead to better results. However, that is not necessarily the case. If your data is of poor quality, adding more of it may actually harm your performance as your model will learn irrelevant or even incorrect associations.

Alternatively, if your data is of high quality but all very similar to each other, then adding more of it will not help – your model will have learned one particular pattern very well, but slightly atypical examples at run time will confuse it.

KEY POINT

This is a key point, so we'll state it again: **More data does not always lead to improved performance**.

To continue advancing the field we are going to have to start thinking about solutions to our excess of data. I will cover three possibilities. The first is how to train on all of that data (Single Pass Training), the second is how to reduce your data down to just the examples that are most important (Filtering out rubbish) and the third is how to use an excess of data in one domain to improve performance in a different domain (Domain Adaptation).

Single Pass Training

The simplest solution to having too much data to train on is to break the paradigm of training to convergence and where progress is measured in terms of the number of epochs completed. An epoch is a full pass over all your available data, updating your parameters as you go along. Typically, you use many epochs to converge your model towards an optimal solution, where convergence is defined as occurring when an epoch no longer improves your model on some validation dataset compared to the previous epoch.

If you have very large amounts of data, this level of iteration to convergence over your training set may not be possible within your hardware and time constraints. This is starting to become the norm in machine translation for example, where users are talking in terms of number of hours or days a model has been trained for, rather than the number of epochs. To make this work practically, you need to have regular checkpointing, rather than waiting until the end of an epoch as was historically standard practice. This means outputting interim models at regular points – perhaps after every few hours or after a set number of pieces of data have been processed – then benchmarking those models against a validation set to track progress to convergence and also whether any hyperparameters need to be dynamically adjusted, such as learning rate decay.

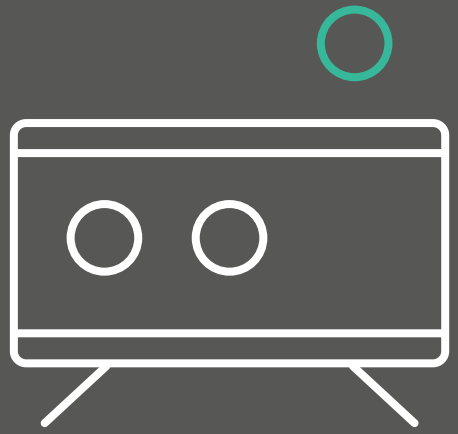
These changes are not yet available 'off the shelf' in most machine learning toolkits. In the main, the paradigm of training multiple epochs over your data still rules. You may need to either hack the code to make it possible or adjust how you deal with your data-perhaps chunking your data into pieces and feeding in one at a time, 'tricking' your model into thinking each one is a full epoch.

In the extreme, this can be pushed to single pass training, where your models only ever see any individual piece of data once. This can work if your data is all of equal quality and relevance to your use case and there are no particular constraints on the order you train over it.

However, it is rarely the case that all of your data is of equal quality and relevance for your use case, which brings us to the next couple of tricks.

Part 2

Filtering out rubbish



The truth is that a lot of your data is probably bad data. That is not your fault. You did your best.

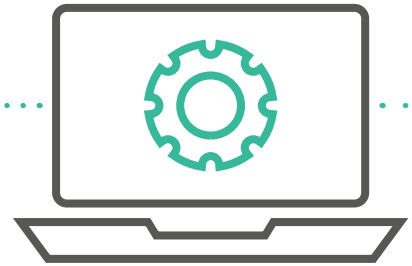
But sometimes your HTML parser barfs on a particular malformed website. Or an angry user abuses your web feedback form. These things happen, but regardless of cause, you don't want to train on that particular piece of data. If you could filter out all that rubbish you would not only reduce the amount of computational resource you need to build models on it, you would also end up with better models as they won't be misled by training on bad examples.

This is becoming a big enough task, that this year's World Machine Translation conference had a corpus filtering competition for machine translation data. The task was to take 1 billion words of supposedly parallel English-German data that had been roughly scraped from the web and automatically filter it down to smaller corpus sizes. I entered that competition as part of a team and have some tips coming out the back of it.

First off, start simple.

You don't necessarily need to break out Tensorflow and start training deep convolutional Siamese networks to discriminate your good from your bad data immediately. Our first step was to devise a bunch of simple rules that quickly eliminated the worst examples. The nature of these rules will depend on your particular use case, so use your common sense and old-fashioned eyeballing of the data.

For our machine translation corpus, for example, we could pretty quickly see that some sentences were not correct translations of one another simply because their lengths were very different, so we eliminated any sentence pairs for which one side was much longer than the other. Other rules were similar – we used edit distance to check for non-translation, simple regex to check digits matched, an off the shelf language identifier to eliminate badly labelled data – and quickly we had reduced our corpus to less than a fifth of its original size.



Once you have reduced your corpus using simple rules, there may be more to be gained by more intelligent methods. One of the more straightforward methods is to use your final task as a way to filter your training data. By this I mean train a model to solve whatever problem you are attempting to solve and then use that model on your training data to identify good and bad data examples.

AN EXPLAINATION BY EXAMPLE

This is probably best explained by example. Let's assume we are building an ant photograph classifier. We have lots of photographs of different ants, each labelled with the species of the ant in the picture. However, we notice that some of the photographs have been mislabelled, and some are not even of ants at all! Once we have used simple rules to eliminate the worst offenders, we are left with a smaller dataset that we still believe to have some bad data in it. So, we build an ant photograph classifier on the mixed data we have. We may choose to make this a smaller than normal model or build on a randomly selected subset of the data to save time and resources. We then apply that classifier on our source data, applying probabilistic labels to each photograph. There are two types of data we may want to remove.

The first is that for which the classifier disagrees very strongly with the data label. In this case, the data label is probably incorrect (perhaps the photograph is of a termite rather than an ant?) and so that data should be removed. We should note though that we do have to be careful that we aren't removing correctly labelled examples that happen to be unusual (for example an Army Ant photographed from below when most of the images are from above). These unusual examples are extremely valuable to keep – some form of tradeoff often needs to take place here which you will need to tune for your own use case.

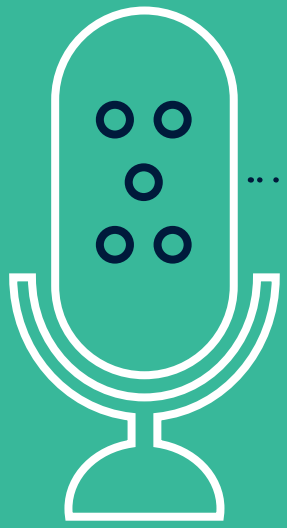
The second is that you might actually want to remove some of the examples for which the classifier very strongly agrees with the data.

In this case training on that particular example is probably not adding much to the overall model, and by removing it you will be allowing your model to generalise better, by instead emphasising the diversity of your dataset. Keeping just the examples that the classifier already does well on will not give the best overall performance in real world tests.

So far, the discussion has considered pre-processing the data, but the technique of removing low/high classifier confidence examples is also starting to be used on the fly during training. Recent papers have shown great benefit in adapting which data you train on as you go along. Between epochs, you could check which of your training data you are struggling on and overemphasise those in the next epoch, whilst down-weighting those that the model is already very confident in.

Curriculum learning is a recent advance that takes this a step further and orders the training examples you present to your model training, with simple ones coming earlier in the training regime and more difficult ones later. This can be compared to how we learn at school or university – starting from more simple, general examples then gradually moving to more nuanced and difficult tasks as we learn more. The early examples lead to generalised abstractions, which are then useful in understanding and modelling the more complex or unusual examples later in the curriculum.

Part 3



Domain adaptation

It may be that your large amounts of data are just not the type of data you think you need. Don't despair!

There are ways you may be able to use it anyway. In my primary field of speech recognition, this is a common occurrence – I may have lots of recordings of people speaking on broadcast news, for example, but few examples of people talking on a telephone, which may be the area I really want to target. In this case, you need to leverage that larger 'out-of-domain' corpus by taking the smaller 'in-domain' corpus as a reference.

In the simple case, this may just mean paying attention to any metadata you have. If you have data prelabelled in different categories, you may be able to simply select data that is close enough to your target domain that way. However, this throws away a lot of the use that the larger pool of out-of-domain data might offer.

The next step – which we at Speechmatics regularly use in our language modelling – is domain filtering. This means taking your small in-domain corpus and filtering your larger out-of-domain corpus against it to find the subset of the data that is most similar. For language modelling we do this by entropy filtering. Entropy filtering works by building small language models on both your in and out-of-domain corpora. Then you measure how well each model performs when modelling every sentence in the out-of-domain corpus. Those sentences for which the difference between these two measures is below a certain threshold are kept and the others discarded. This then leads to a smaller corpus which actually gives better results than using the entire out-of-domain corpus, as well as a smaller training footprint.

An even further step is to map your out-of-domain data into the same space as your in-domain data.

Correlation Alignment (CORAL) is a 'frustratingly easy' first technique you could use. It's called 'frustratingly easy' because it can be implemented in as little as 4 lines of MATLAB code and outperforms many more complex methods! The core algorithm can be understood as first whitening your out-of-domain data then, re-colouring it with the covariance of the in-domain data and has been shown to be surprisingly effective in fields such as object recognition from images.

Other methods have also been shown to be effective – such as subspace alignment, where both in and out-of-domain data are mapped to a shared subspace and that mapping is iteratively repeated until the distance between the two datasets is minimised. The performance is typically improved by use of 'anchor' examples which can be used to measure how well the two datasets have mapped onto one another. Facebook's MUSE uses similar techniques to map vectoral representations of words in different languages onto one another, with improved performance by providing an initial seed dictionary of translations.

Finally, if you have already trained a model for a different task using your large dataset, you can use transfer learning. This means training a model for a particular task and/or domain and then using that model as the basis for your target task or domain. In the simplest case, this means training a model on your out-of-domain dataset then retraining the weights on your in-domain dataset, essentially using the larger dataset to produce sensible initial values for your final model.

A really interesting example of transfer learning I found recently took this even further. The task was discrimination of audio clips into classes, with not much data available to train on. Rather than try to leverage audio data, they took the interesting step of stepping really far out-of-domain – into image recognition. They took spectrographic representations of their audio signals, turned them into images with a green tint, then used a pre-trained and very accurate image classifier as the basis for the audio classifier! The system trained to classify images was able to break the spectrogram images into useable sub features sufficiently well to massively improve performance over just using the limited dataset. This really shows that if you have sufficient data in any task, it may be worthwhile to leverage it in completing any other limited data tasks you have.

Conclusions

We all know data is hugely valuable to those of us training artificial intelligence systems of any kind. Too much data can feel like a burden, either in training time, hardware requirements or simply output model size. However, I hope that over our three-part series I have shown you some tips for how to deal with that data. Whether that be reducing the data size into a highly efficient subset or using the data to bootstrap models in radically different domains and tasks…

…I hope you can learn to love your big data again.


SPEECHMATICS