Draft Version

MACHINE LEARNING YEARNING

Technical Strategy for AI Engineers, In the Era of Deep Learning

ANDREW NG

deeplearning.ai

Machine Learning Yearning is a
deeplearning.ai project.

Training and testing on different distributions

36 When you should train and test on different distributions

Users of your cat pictures app have uploaded 10,000 images, which you have manually labeled as containing cats or not. You also have a larger set of 200,000 images that you downloaded off the internet. How should you define train/dev/test sets?

Since the 10,000 user images closely reflect the actual probability distribution of data you want to do well on, you might use that for your dev and test sets. If you are training a data-hungry deep learning algorithm, you might give it the additional 200,000 internet images for training. Thus, your training and dev/test sets come from different probability distributions. How does this affect your work?

Instead of partitioning our data into train/dev/test sets, we could take all 210,000 images we have, and randomly shuffle them into train/dev/test sets. In this case, all the data comes from the same distribution. But I recommend against this method, because about 205,000/210,000 ≈ 97.6% of your dev/test data would come from internet images, which does not reflect the actual distribution you want to do well on. Remember our recommendation on choosing dev/test sets:

> Choose dev and test sets to reflect data you expect to get in the future and want to do well on.

Most of the academic literature on machine learning assumes that the training set, dev set and test set all come from the same distribution.[1] In the early days of machine learning, data was scarce. We usually only had one dataset drawn from some probability distribution. So we would randomly split that data into train/dev/test sets, and the assumption that all the data was coming from the same source was usually satisfied.

[1] There is some academic research on training and testing on different distributions. Examples include "domain adaptation," "transfer learning" and "multitask learning." But there is still a huge gap between theory and practice. If you train on dataset A and test on some very different type of data B, luck could have a huge effect on how well your algorithm performs. (Here, "luck" includes the researcher's hand-designed features for the particular task, as well as other factors that we just don't understand yet.) This makes the academic study of training and testing on different distributions difficult to carry out in a systematic way.

But in the era of big data, we now have access to huge training sets, such as cat internet images. Even if the training set comes from a different distribution than the dev/test set, we still want to use it for learning since it can provide a lot of information.

For the cat detector example, instead of putting all 10,000 user-uploaded images into the dev/test sets, we might instead put 5,000 into the dev/test sets. We can put the remaining 5,000 user-uploaded examples into the training set. This way, your training set of 205,000 examples contains some data that comes from your dev/test distribution along with the 200,000 internet images. We will discuss in a later chapter why this method is helpful.

Let's consider a second example. Suppose you are building a speech recognition system to transcribe street addresses for a voice-controlled mobile map/navigation app. You have 20,000 examples of users speaking street addresses. But you also have 500,000 examples of other audio clips with users speaking about other topics. You might take 10,000 examples of street addresses for the dev/test sets, and use the remaining 10,000, plus the additional 500,000 examples, for training.

We will continue to assume that your dev data and your test data come from the same distribution. But it is important to understand that different training and dev/test distributions offer some special challenges.

37 How to decide whether to use all your data

Suppose your cat detector's training set includes 10,000 user-uploaded images. This data comes from the same distribution as a separate dev/test set, and represents the distribution you care about doing well on. You also have an additional 20,000 images downloaded from the internet. Should you provide all 20,000+10,000=30,000 images to your learning algorithm as its training set, or discard the 20,000 internet images for fear of it biasing your learning algorithm?

When using earlier generations of learning algorithms (such as hand-designed computer vision features, followed by a simple linear classifier) there was a real risk that merging both types of data would cause you to perform worse. Thus, some engineers will warn you against including the 20,000 internet images.

But in the modern era of powerful, flexible learning algorithms—such as large neural networks—this risk has greatly diminished. If you can afford to build a neural network with a large enough number of hidden units/layers, you can safely add the 20,000 images to your training set. Adding the images is more likely to increase your performance.

This observation relies on the fact that there is some x —> y mapping that works well for both types of data. In other words, there exists some system that inputs either an internet image or a mobile app image and reliably predicts the label, even without knowing the source of the image.

Adding the additional 20,000 images has the following effects:

1. It gives your neural network more examples of what cats do/do not look like. This is helpful, since internet images and user-uploaded mobile app images do share some similarities. Your neural network can apply some of the knowledge acquired from internet images to mobile app images.

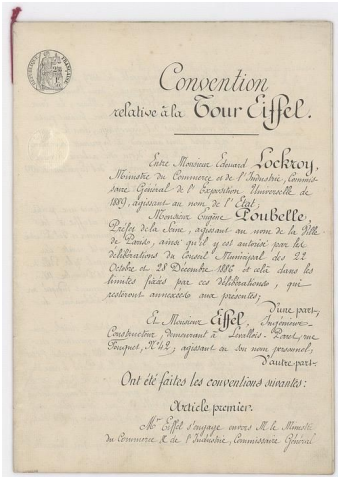
2. It forces the neural network to expend some of its capacity to learn about properties that are specific to internet images (such as higher resolution, different distributions of how the images are framed, etc.) If these properties differ greatly from mobile app images, it will "use up" some of the representational capacity of the neural network. Thus there is less capacity for recognizing data drawn from the distribution of mobile app images, which is what you really care about. Theoretically, this could hurt your algorithms' performance.

To describe the second effect in different terms, we can turn to the fictional character Sherlock Holmes, who says that your brain is like an attic; it only has a finite amount of space. He says that "for every addition of knowledge, you forget something that you knew before. It is of the highest importance, therefore, not to have useless facts elbowing out the useful ones."[2]

Fortunately, if you have the computational capacity needed to build a big enough neural network—i.e., a big enough attic—then this is not a serious concern. You have enough capacity to learn from both internet and from mobile app images, without the two types of data competing for capacity. Your algorithm's "brain" is big enough that you don't have to worry about running out of attic space.

But if you do not have a big enough neural network (or another highly flexible learning algorithm), then you should pay more attention to your training data matching your dev/test set distribution.

If you think you have data that has no benefit,you should just leave out that data for computational reasons. For example, suppose your dev/test sets contain mainly casual pictures of people, places, landmarks, animals. Suppose you also have a large collection of scanned historical documents:


Convention
relative à la Tour Eiffel.

These documents don't contain anything resembling a cat. They also look completely unlike your dev/test distribution. There is no point including this data as negative examples, because the benefit from the first effect above is negligible—there is almost nothing your neural network can learn from this data that it can apply to your dev/test set distribution. Including them would waste computation resources and representation capacity of the neural network.

[2] *A Study in Scarlet* by Arthur Conan Doyle

38 How to decide whether to include inconsistent data

Suppose you want to learn to predict housing prices in New York City. Given the size of a house (input feature x), you want to predict the price (target label y).

Housing prices in New York City are very high. Suppose you have a second dataset of housing prices in Detroit, Michigan, where housing prices are much lower. Should you include this data in your training set?

Given the same size x, the price of a house y is very different depending on whether it is in New York City or in Detroit. If you only care about predicting New York City housing prices, putting the two datasets together will hurt your performance. In this case, it would be better to leave out the inconsistent Detroit data.[3]

How is this New York City vs. Detroit example different from the mobile app vs. internet cat images example?

The cat image example is different because, given an input picture x, one can reliably predict the label y indicating whether there is a cat, even without knowing if the image is an internet image or a mobile app image. I.e., there is a function f(x) that reliably maps from the input x to the target output y, even without knowing the origin of x. Thus, the task of recognition from internet images is "consistent" with the task of recognition from mobile app images. This means there was little downside (other than computational cost) to including all the data, and some possible significant upside. In contrast, New York City and Detroit, Michigan data are not consistent. Given the same x (size of house), the price is very different depending on where the house is.

[3] There is one way to address the problem of Detroit data being inconsistent with New York City data, which is to add an extra feature to each training example indicating the city. Given an input x—which now specifies the city—the target value of y is now unambiguous. However, in practice I do not see this done frequently.

39 Weighting data

Suppose you have 200,000 images from the internet and 5,000 images from your mobile app users. There is a 40:1 ratio between the size of these datasets. In theory, so long as you build a huge neural network and train it long enough on all 205,000 images, there is no harm in trying to make the algorithm do well on both internet images and mobile images.

But in practice, having 40x as many internet images as mobile app images might mean you need to spend 40x (or more) as much computational resources to model both, compared to if you trained on only the 5,000 images.

If you don't have huge computational resources, you could give the internet images a much lower weight as a compromise.

For example, suppose your optimization objective is squared error (This is not a good choice for a classification task, but it will simplify our explanation.) Thus, our learning algorithm tries to optimize:

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} \left(h_\theta\left(x\right) - y\right)^2 + \sum_{(x,y)\in \text{InternetImg}} \left(h_\theta\left(x\right) - y\right)^2$$

The first sum above is over the 5,000 mobile images, and the second sum is over the 200,000 internet images. You can instead optimize with an additional parameter β:

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} \left(h_\theta\left(x\right) - y\right)^2 + \beta \sum_{(x,y)\in \text{InternetImg}} \left(h_\theta\left(x\right) - y\right)^2$$

If you set β=1/40, the algorithm would give equal weight to the 5,000 mobile images and the 200,000 internet images. You can also set the parameter β to other values, perhaps by tuning to the dev set.

By weighting the additional Internet images less, you don't have to build as massive a neural network to make sure the algorithm does well on both types of tasks. This type of re-weighting is needed only when you suspect the additional data (Internet Images) has a very different distribution than the dev/test set, or if the additional data is much larger than the data that came from the same distribution as the dev/test set (mobile images).