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.

40 Generalizing from the training set to the dev set

Suppose you are applying ML in a setting where the training and the dev/test distributions are different. Say, the training set contains Internet images + Mobile images, and the dev/test sets contain only Mobile images. However, the algorithm is not working well: It has a much higher dev/test set error than you would like. Here are some possibilities of what might be wrong:

1. It does not do well on the training set. This is the problem of high (avoidable) bias on the training set distribution.
2. It does well on the training set, but does not generalize well to previously unseen data *drawn from the same distribution as the training set*. This is high variance.
3. It generalizes well to new data drawn from the same distribution as the training set, but not to data drawn from the dev/test set distribution. We call this problem **data mismatch**, since it is because the training set data is a poor match for the dev/test set data.

For example, suppose that humans achieve near perfect performance on the cat recognition task. Your algorithm achieves this:

- 1% error on the training set
- 1.5% error on data drawn from the same distribution as the training set that the algorithm has not seen
- 10% error on the dev set

In this case, you clearly have a data mismatch problem. To address this, you might try to make the training data more similar to the dev/test data. We discuss some techniques for this later.

In order to diagnose to what extent an algorithm suffers from each of the problems 1-3 above, it will be useful to have another dataset. Specifically, rather than giving the algorithm all the available training data, you can split it into two subsets: The actual training set which the algorithm will train on, and a separate set, which we will call the “Training dev” set, that we will not train on.

You now have four subsets of data:

- Training set. This is the data that the algorithm will learn from (e.g., Internet images + Mobile images). This does not have to be drawn from the same distribution as what we really care about (the dev/test set distribution).
- Training dev set: This data is drawn from the same distribution as the training set (e.g., Internet images + Mobile images). This is usually smaller than the training set; it only needs to be large enough to evaluate and track the progress of our learning algorithm.
- Dev set: This is drawn from the same distribution as the test set, and it reflects the distribution of data that we ultimately care about doing well on. (E.g., mobile images.)
- Test set: This is drawn from the same distribution as the dev set. (E.g., mobile images.)

Armed with these four separate datasets, you can now evaluate:

- Training error, by evaluating on the training set.
- The algorithm's ability to generalize to new data drawn from the training set distribution, by evaluating on the training dev set.
- The algorithm's performance on the task you care about, by evaluating on the dev and/or test sets.

Most of the guidelines in Chapters 5-7 for picking the size of the dev set also apply to the training dev set.

41 Identifying Bias, Variance, and Data Mismatch Errors

Suppose humans achieve almost perfect performance (≈0% error) on the cat detection task, and thus the optimal error rate is about 0%. Suppose you have:

- 1% error on the training set.
- 5% error on training dev set.
- 5% error on the dev set.

What does this tell you? Here, you know that you have high variance. The variance reduction techniques described earlier should allow you to make progress.

Now, suppose your algorithm achieves:

- 10% error on the training set.
- 11% error on training dev set.
- 12% error on the dev set.

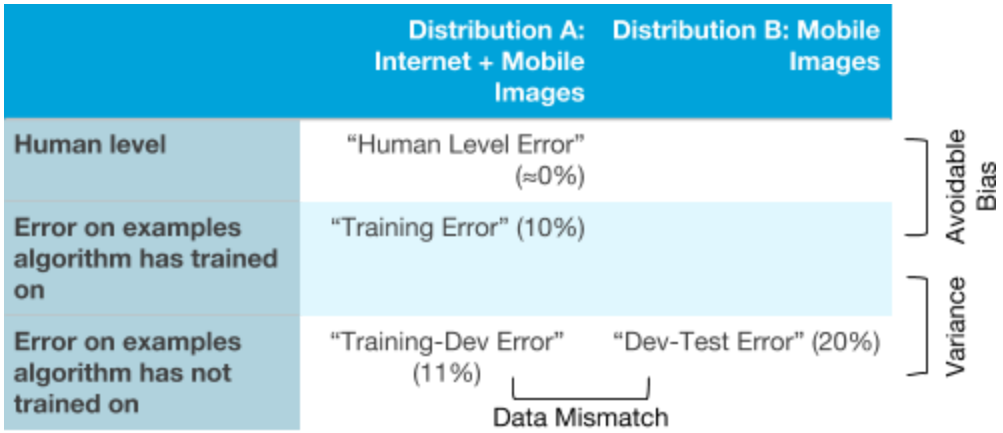
This tells you that you have high avoidable bias on the training set. I.e., the algorithm is doing poorly on the training set. Bias reduction techniques should help.

In the two examples above, the algorithm suffered from only high avoidable bias or high variance. It is possible for an algorithm to suffer from any subset of high avoidable bias, high variance, and data mismatch. For example:

- 10% error on the training set.
- 11% error on training dev set.
- 20% error on the dev set.

This algorithm suffers from high avoidable bias and from data mismatch. It does not, however, suffer from high variance on the training set distribution.

It might be easier to understand how the different types of errors relate to each other by drawing them as entries in a table:



Continuing with the example of the cat image detector, you can see that there are two different distributions of data on the x-axis. On the y-axis, we have three types of error: human level error, error on examples the algorithm has trained on, and error on examples the algorithm has not trained on. We can fill in the boxes with the different types of errors we identified in the previous chapter.

If you wish, you can also fill in the remaining two boxes in this table: You can fill in the upper-right box (Human level performance on Mobile Images) by asking some humans to label your mobile cat images data and measure their error. You can fill in the next box by taking the mobile cat images (Distribution B) and putting a small fraction of into the training set so that the neural network learns on it too. Then you measure the learned model's error on that subset of data. Filling in these two additional entries may sometimes give additional insight about what the algorithm is doing on the two different distributions (Distribution A and B) of data.

By understanding which types of error the algorithm suffers from the most, you will be better positioned to decide whether to focus on reducing bias, reducing variance, or reducing data mismatch.

42 Addressing data mismatch

Suppose you have developed a speech recognition system that does very well on the training set and on the training dev set. However, it does poorly on your dev set: You have a data mismatch problem. What can you do?

I recommend that you: (i) Try to understand what properties of the data differ between the training and the dev set distributions. (ii) Try to find more training data that better matches the dev set examples that your algorithm has trouble with.[1]

For example, suppose you carry out an error analysis on the speech recognition dev set: You manually go through 100 examples, and try to understand where the algorithm is making mistakes. You find that your system does poorly because most of the audio clips in the dev set are taken within a car, whereas most of the training examples were recorded against a quiet background. The engine and road noise dramatically worsen the performance of your speech system. In this case, you might try to acquire more training data comprising audio clips that were taken in a car. The purpose of the error analysis is to understand the significant differences between the training and the dev set, which is what leads to the data mismatch.

If your training and training dev sets include audio recorded within a car, you should also double-check your system's performance on this subset of data. If it is doing well on the car data in the training set but not on car data in the training dev set, then this further validates the hypothesis that getting more car data would help. This is why we discussed the possibility of including in your training set some data drawn from the same distribution as your dev/test set in the previous chapter. Doing so allows you to compare your performance on the car data in the training set vs. the dev/test set.

Unfortunately, there are no guarantees in this process. For example, if you don't have any way to get more training data that better match the dev set data, you might not have a clear path towards improving performance.

[1]There is also some research on "domain adaptation"—how to train an algorithm on one distribution and have it generalize to a different distribution. These methods are typically applicable only in special types of problems and are much less widely used than the ideas described in this chapter.

43 Artificial data synthesis

Your speech system needs more data that sounds as if it were taken from within a car. Rather than collecting a lot of data while driving around, there might be an easier way to get this data: By artificially synthesizing it.

Suppose you obtain a large quantity of car/road noise audio clips. You can download this data from several websites. Suppose you also have a large training set of people speaking in a quiet room. If you take an audio clip of a person speaking and "add" to that to an audio clip of car/road noise, you will obtain an audio clip that sounds as if that person was speaking in a noisy car. Using this process, you can "synthesize" huge amounts of data that sound as if it were collected inside a car.

More generally, there are several circumstances where artificial data synthesis allows you to create a huge dataset that reasonably matches the dev set. Let's use the cat image detector as a second example. You notice that dev set images have much more motion blur because they tend to come from cellphone users who are moving their phone slightly while taking the picture. You can take non-blurry images from the training set of internet images, and add simulated motion blur to them, thus making them more similar to the dev set.

Keep in mind that artificial data synthesis has its challenges: it is sometimes easier to create synthetic data that appears realistic to a person than it is to create data that appears realistic to a computer. For example, suppose you have 1,000 hours of speech training data, but only 1 hour of car noise. If you repeatedly use the same 1 hour of car noise with different portions from the original 1,000 hours of training data, you will end up with a synthetic dataset where the same car noise is repeated over and over. While a person listening to this audio probably would not be able to tell—all car noise sounds the same to most of us—it is possible that a learning algorithm would "overfit" to the 1 hour of car noise. Thus, it could generalize poorly to a new audio clip where the car noise happens to sound different.

Alternatively, suppose you have 1,000 unique hours of car noise, but all of it was taken from just 10 different cars. In this case, it is possible for an algorithm to "overfit" to these 10 cars and perform poorly if tested on audio from a different car. Unfortunately, these problems can be hard to spot.

To take one more example, suppose you are building a computer vision system to recognize cars. Suppose you partner with a video gaming company, which has computer graphics models of several cars. To train your algorithm, you use the models to generate synthetic images of cars. Even if the synthesized images look very realistic, this approach (which has been independently proposed by many people) will probably not work well. The video game might have ~20 car designs in the entire video game. It is very expensive to build a 3D car model of a car; if you were playing the game, you probably wouldn't notice that you're seeing the same cars over and over, perhaps only painted differently. I.e., this data looks very realistic to you. But compared to the set of all cars out on roads—and therefore what you're likely to see in the dev/test sets—this set of 20 synthesized cars captures only a minuscule fraction of the world's distribution of cars. Thus if your 100,000 training examples all come from these 20 cars, your system will "overfit" to these 20 specific car designs, and it will fail to generalize well to dev/test sets that include other car designs.

When synthesizing data, put some thought into whether you're really synthesizing a representative set of examples. Try to avoid giving the synthesized data properties that makes it possible for a learning algorithm to distinguish synthesized from non-synthesized examples—such as if all the synthesized data comes from one of 20 car designs, or all the synthesized audio comes from only 1 hour of car noise. This advice can be hard to follow.

When working on data synthesis, my teams have sometimes taken weeks before we produced data with details that are close enough to the actual distribution for the synthesized data to have a significant effect. But if you are able to get the details right, you can suddenly access a far larger training set than before.