Error analysis

Error analysis:

- Get 100 mislabeled dev set examples
- Count up how many of them are of one particular class

Knowing that you can take decisions on which class mislabelling you should tackle. This process gives you an estimate of how worthwhile it might be to work on each of the different categories of errors and it five syou a sense of the best options to pursue (maybe blurry images, images of one particular category being mislabeled as another one, etc)

Incorrectly labeled data: It turns out that deep learning algorithms are quite robust to random errors in the training set; if the errors are reasonably random, then it's probably okay. Nevertheless, deep learning algorithms are less robust to systematic errors.

Correcting incorrect dev/test sets: It might only be worthwhile doing it after evaluating the impact using error analysis. If you decide to do it some guidelines are:

- Apply same process to dev and test sets to make sure they continue to come from the same distribution.
- Consider examining examples the algorithm got right as well as the ones it got wrong.
- Train and dev/test data may now come from slightly different distributions. (this might be okay)

Build the first system quickly and then iterate: Set up dev/test set and metrics. Build an initial system quickly and then use bias/variance analysis and error analysis to prioritize next steps.

Mismatched training and dev/test sets

You might want to (purposely) use train and dev/test sets with different distributions since sometimes you can't gather enough data to train your system and you might get data from different sources to make a richer training.

Bias and Variance with mismatched data distributions When training and testing data comes from different distributions you can no longer draw the conclusion that the algorithm is not generalizing if your dev error ir much bigger than the train error.

For this problem is important to create a *training-dev* set that has de same distribution as the training (but it's not used for training). Looking at this error you can see if you have a generalization problem (your training-dev error is similar to the dev error) or a distribution problem (you training-dev is small so the problem is because of the data mismatching)

Addressing data mismatch Carry out manual error analysis and try to make the training data more similar or so that it resambles the kind of data that produced the mistakes.

Transfer learning

One of the most powerful ideas in deep learning is that sometimes you can take knowledge the neural network has learned from one task and apply that knowledge to a separate task. This is knonw as transfer learning.

The basic idea is to take a neural network that was trained for a particular task, remove the last layer and train it for the new (similar) task. Usually this is called *finetuning*.

When does transfer learning makes sense? Transfer learning makes sense when you have a lot of data for the problem you're transferring from and usually relatively less data for the problem you're transferring to. In general:

- Task A ans task B have the same input x.
- You have a lot mode data for task A than for task B.
- Low level features from A could be helpfull for learning B.

Multi task learning

So whereas in transfer learning, you have a sequential process where you learn from task A and then transfer that to task B. In multi-task learning, you start off simultaneously, trying to have one neural network do several

things at the same time. And then each of these task helps hopefully all of the other task.

When does multi-task learning makes sense?

- Training on a set of tasks that could benefit from havind shared lower level features.
- Usually: Amount of data you have for each task is quite similar.
- Can train a big neural network to do well on all tasks.

End-to-end deep learning

There have been some data processing systems, or learning systems that require multiple stages of processing. And what end-to-end deep learning does, is it can take all those multiple stages, and replace it usually with just a single neural network.

Pros and cons of end-to-end deep learning

- Pro: Let the data speak
- Pro: Less hand-designing of components needed.
- Con: May need large amounts of data.
- Con: Excludes potentially useful hand-designed components.

Key question: Do I have sufficient data to learn a function od the complexity needed to map x to y?