Introduction to ML strategy

When working in a aplication of DL you might have a lot of ideas as to how to improve your system

- Collect more data or a more diverse training set.
- Train the algorithm longer or with different optimizers.
- Try bigger or smaller networks.
- Try $L2$ or dropout regularization.
- Change the network architecture (activation functions, hidden units).

And the problem is that if you choose poorly, it is entirely possible that you end up spending six months charging in some direction only to realize after six months that that didn't do any good.

We would like to have a number of strategies, that is, ways of analyzing a machine learning problem that will point you in the direction of the most promising things to try.

Orthogonalization

Chain of assumptions in supervised learning:

- Fit training set well on cost function (human-level performance). If something fails here you might train a bigger network or switch to a different optimization algorithm
- Fit dev set well on cost function. If it fails here you might try regularization techniques
- Fit test set well on cost function. If it fails here you might need a bigger dev set.
- Performs well in real world. If it fails here you should go back and try changing your dev set or cost function.

Setting up a goal

Single number evaluation metric

Whether tuning hyperparameters, or trying out different ideas for learning algorithms, you'll find that your progress will be much faster if you have a single real number evaluation metric that lets you quickly tell if the new thing you just tried is working better or worse than your last idea.

As an example, suppose you are training two classifiers and at evaluation time one performs better in terms of precision and the other one performs better in terms of recall, which one would you choose? It's better to choose a metric that combines this two (as the F1 score) so that you can focus just in that.

Satisficing and Optimizing metric

It's not always easy to combine all the things you care about into a single row number evaluation metric. In those cases it's sometimes useful to set up satisficing as well as optimizing matrix.

As an example, consider a problem where you want to maximize your accuracy but the model has to run quickly, you will create the following meeting.

$$cost = \max_{\text{running time } \leq 100} \{\text{accuracy}\}$$

In this case we say we are optimizing the accurazy, satisfiying the running time.

Train/dev/test sets

Set distributions:

- Dev and test sets should have the same distributions (randomly shuffle before selecting dev/test sets).
- Choose dev and test set to reflect data you expect to get in the future and consider important to do well on.

Set sizes:

In traditional ML, the 60-20-20 % split was a rule of thumb for train/dev/test sizes. Right now it might not be the case, if you have a milllion observations a 98/1/2 % split might be sufficient to the the validation and training and let's you use more data to train the model.

Rule of thumb: set the test set to be big enough to give hugh confidence in the overall performance of the system, it can be 10k or 100k. Also consider that it might not be necessary to have a test set so you can only use the train and validation sets.

1 Comparing to human-level performance

1.1 Avoidable bias

We talked about how you want your learning algorithm to do well on the training set but sometimes you don't actually want to do too well and knowing what human level performance is, can tell you exactly how well but not too well you want your algorithm to do on the training set.

We suppose that the human-level error is a good proxy for the bayes optimal error, so knowing the human error we can take decisions on what to do.

As an example suppose your classifier has a training and dev error really distinct from your human-level error, then you would like to reduce variance. Now supose the training error is similar to the human-level error, then it doesn't make much sense to putting efford in reducing bias and it might be better to focus in reducing variance.

The avoidable bias is defined as the diference between the human-level error and the train error.

1.2 Surpassing human-level performance

What happens when the train and dev errors are better thean the human-level error? should you focus on reducing bias or variance? It's not clear because we don't really know which is the bayes error. That's why the machine learning process slows down after surpassing human-level performance.

Problems where ML significantly surpasses human-level performance:

- Online advertising
- Product recommendations
- Logistics
- Loan approvals
- Some speech, image recognition and medical aplications

1.3 Summary to improve model performance

If the avoidable bias is big and you want to reduce bias:

- Train bigger model
- Train longer and with better optimization algorithms
- Change architecture
- Hyperparameter search

If the avoidable bias is small and you want to reduce variance:

- Get more data
- Regularization