2025-08-20T21-07-47
Tags: DLS

DLS C3 week 1

ML strategy

When improving a DL system, often there are many ideas to try. There's a risk in this—choosing the wrong direction can waste months of work.

Orthogonalization

An effective ML practitioner knows what to tune for each performance issue—they use orthogonalization.

Orthogonalization means designing controls that affect only one factor. Orthogonal means at 90° implying independence.

For supervised learning to succeed, four things need to hold true.

1. Fit train set well on J
2. Fit dev set well on J
3. Fit test set well on J
4. Perform well on production

Like tuning a TV, each problem should have dedicated knobs to fix it.

If train set is poor:	• make bigger network. • apply better optimizer (e.g., Adam).	If dev set is poor:	• get bigger train set. • apply regularization. • find better generalization methods.
If test set is poor but dev set is good:	• get bigger dev set. • increase regularizationn.	If production performance is poor but test set is good:	• change the dev set. • change cost function.

Early stopping is not a bad technique, a lot of people use it. But, doesn't follow orthogonalization:

- It fit the train set less well.
- Simultaneously, it improve dev set performance.

Single number evaluation metric

Progress will be faster if one has a single real number evaluation metric that tells if the latest approach is better or worse than the last.

Precision	• Of images predicted as cats, how many are truly cats? (in %)	Recall	• Of all true cat images, how many were correctly recognized? (in %)

There's a problem: Imagine two classifiers: one has better recall, the other has better precision. It's unclear which is classifier is better overall.

- The solution is to combine precision and recall into a single number evaluation metric called F1 score.

F1 score is defined as the harmonic mean of precision (P) and recall (R):

$$\mathrm{F1} = \frac{2}{P^{-1} + R^{-1}}$$

A well-defined dev set and a single *real* number evaluation metric speeds up the iterative process of choosing a model.

If there are multiple error rates in four markets: US, China, India, and the rest of the world. Tracking multiple error rates leads to difficult decision making.

- The solution is to still track per-region perfomrnace.
- Still compute the average error across the four markets.

Choose a single number evaluation metric as the primary decision driver.

Satisficing and optimizing metrics

It's not always easy to combine all factors into a single number evaluation metric. When this happens, set up *satisficing* and *optimizing* metrics.

In addition to accuracy, one might care about the running time. For instance:

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	95 ms

Choose a classifier that maximizes accuracy but subject a running time that is less than 100 ms.

- Accuracy is the optimizing metric.
- Running time is the satisficing metric.

For N metrics, it's reasonable to have:

- 1 optimizing
- $N-1$ satisificing metrics

Example: wake word detection system

- metrics of concern are accuracy and false positives.
- optimizing metric is the accuracy (maximize correct wake-up rate)
- satisficing metric is the false positives (e.g., at most 1 false positive for every 24 hours).

Train/Dev/Test distributions

Dev and test sets must come from the same distribution. If not, months of optimization on the dev set may not generalize well to the test set.

Setting up the dev set and a single number evaluation metric should be followed.

If we have data from the different regions, say: U.S., U.K., other European countries, South America, India, China, other Asian countries, Australia.

- The solution is to randomly shuffle all data into the dev and test set. This will make the dev and test set come from the same distribution.

But before all that, remember to choose a dev and test set that reflect data you expect to get in the future and consider important to do well on.

It's alright to have a different distribution in the train set than the dev and test set.

Size of the dev and test sets

Machine learning rule of thumb for 100-10000 examples:

- 70% train, 30% test
- 60% train, 20% dev, 20% test

But now we're working with much larger data set sizes, say a million examples:

- 98% train, 1% dev, 1% test

The purpose of test set: evaluates how good the final system after finishing development.

Guideline: set the test set to be big enough to give high confidence in the overall performance of the system.

Unless there's a need to have a very accurate measure of final system performance, millions of test set examples is not needed.

If you're tuning on a test set, it's effectively a dev set.

Not recommended to not have a test set when building a system. It is reassuring to have a seperate test set to get an unbiased estimate of performance before shipping. Unless, you have a very large dev set that won't overfit.

Rule of thumb:

- dev set should be big enough for its purpose.
- test set should be big enough to give high confidence to evaluate overall performance.
- dev and test set should be smaller than 30% combined.
- focus on the purpose of each set, not rigid percentages.

When to change dev and test sets and metrics?

Change evaluation metric when it no longer ranks algorithms in the correct order.

Misclassification error formula:

$$\text{Error} = \frac{1}{m_{\text{dev}}} \sum_{i=1}^{m_{\text{dev}}} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

- counts misclassified examples equally.

Weighted error metric:

$$\text{Error} = \frac{1}{\sum_i w^{(i)}} \sum_{i=1}^{m_{\text{dev}}} w^{(i)} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

If unsatisfied with old error metric, define a new one that better captures preferences. This is an orthogonalized step.

1. Define right metric.
2. Figure how to do well on that metric.

Defining and optimizing are seperate knobs to tune.

Given 2 algorithms, both with high-quality dev/test set:

- Algorithm A with 3% error.
- Algorithm B with 5% error.

Say, in deployment, algorithm B performs better on user-uploaded images—blurry, lower quality, or poorly framed.

Reason: dev/test set distribution does not match real-world data.

Solution: change the metric and/or the dev/test set.

Avoid: running for too long without any evaluation metric and dev set.

Recommendation: Set up an evaluation metric and dev set quickly—even if imperfect. If it's not good, just change it.

Why human-level performance?

Why compare performance to humans?

- Advances in deep learning allows algorithms to be competitive with humans.
- Workflow is natural when comparing or mimicking human-level performance.

Working on a machine learning task over time:

1. Progress is rapid as you approach human-level performance.
2. After approaching human-level performance, progress slows down.
3. Eventually, performance achives *Bayes optimal error*.

The Bayes optimal error is the best possible mapping of $x \rightarrow y$. It is the lowest possible error and cannot be surpassed.

Why does progress slow down after human-level performance?

- Human-level performance for many tasks is not far from Bayes optimal error.
- As long as performance is worse than human-level performance you can:
    - get labeled data from humans.
    - performance *manual error analysis*.
    - get a better analysis of bias and variance.

Avoidable bias

Your learning algorithm should perform well, but not too well on the train set.

Human-level performance provides a benchmark on how well (but not too well) the learning algorithm should be.

Case 1:

- Human-level error: 1%
- Train error: 8%
- Dev error: 10%

Huge gap between train error and human-level error. Reduce bias.

Case 2:

- Human-level error: 7.5% due to noisy data.
- Train error: 8%
- Dev error: 10%

The algorithm is doing fine on the train set. Reduce variance instead.

Main points:

- In reality, $\text{Bayes error} > 0$. Human-level error can be thought of as a proxy for Bayes error. But remember, $\text{Human-level error} \geq \text{Bayes error}$.
- Reducing the train error below the Bayes means overfitting.
- Avoidable bias: $\text{Train error} - \text{Bayes error}$
- Variance: $\text{Dev error} - \text{Train error}$
- If avoidable bias is large focus on reducing bias.
- If variance is large focus on reducing variance.

Understanding human-level performance

Human-level error gives us a way of estimating Bayes error.

Medical imaging example:

- Untrained human: 3% error
- Typical doctor: 1% error
- Experienced doctor: 0.7% error
- Team of experienced doctors: 0.5% error

Bayes error is the lowest possible error—0.5% for this case.

The definition of human-level performance *goal*-dependent:

- for bias-variance analysis: use the best possible human performance.
- for deployment: surpass typical human performance.

Case 1:

- train error: 5%
- dev error: 6%
- human-level error: $0.5 - 1\%$
- avoidable bias: $4 - 4.5\%$
- variance: 1%

Bias dominates, focus on reducing bias.

Case 2:

- train error: 1%
- dev error: 5%
- human-level error: $0.5 - 1\%$
- avoidable bias: $0 - 0.5\%$
- variance: 4%

Variance dominates, focus on reducing variance.

Case 3:

- train error: 0.7%
- dev error: 0.8%
- human-level error: 0.5%
- avoidable bias: 0.2%
- variance: 0.1%

0.5% was used because it is the Bayes error. If 0.7% was used instead, it would miss that bias is the larger issue.

Bias is slightly bigger, focus reducing bias.

Key takeaway:

- For tasks with Bayes error is near zero (e.g., cat classification), it is fine to compare train error to 0%.
- For tasks with high noise (e.g., speech recognition in noisy audio), use an accurate Bayes estimate to calculate avoidable bias and variance. This allows us to decide whether to focus on reducing bias or variance.

Surpassing human-level performance

For both of the cases below:

- team of humans error: 0.5%
- single human error: 1%

Case 1

- train error: 0.6%
- dev error: 0.8%
- avoidable bias: 0.1%
- variance: 0.2%

Case 2

- train error: 0.3%
- dev error: 0.4%
- avoidable bias: $?$
- variance: 0.2%

Case 2 has a lower error than the human-level error:

- human intuition becomes harder to rely on.
- direction is now less clear—but still possible to improve.

Problems where ML significantly surpasses human-level performance:

- structured data
- non-natural perception problems
- huge amount of data

With enough data, even natural perception tasks can be surpassed.

Improving your model performance

Two fundamental assumptions for supervised learning:

- You can fit the train set well—low avoidable bias.
- The train set performance generalizes well to the dev/test set—low variance.

What to do when avoidable bias or variance is an issue?

Avoidable bias

- bigger model
- train longer
- better optimization
- different NN architecture
- hyperparameter search
- decrease regularization

Variance

- more data
- regularization
- different NN architecture
- hyperparameter search