Deep Learning Specialization: Strategy

Why ML strategy?

→ If results are not satisfying, there are LOTS of things you could try.

→ Many things will take time / be expensive

ML Strategy: WHAT should you try to improve?
Effective ML engineers are those who know this.
↳ much like in software!

ORTHOGONALISATION

- Decompose potential actions into orthogonal direction
→ Know what actions to take based on the problem you have.

4 core assumptions in ML:

1. Fit training set on cost function well
   - More powerful network (layers, units, BN, activations, ...)
   - Better optimization (Algo, LR, longer runs, more compute

2. Fit dev set on cost function well
   - Regularization
   - Bigger training set

3. Fit test set on cost function well
   - Bigger dev set?

4. Perform well in real world.
   - Different cost function?
   - Bigger dev set

Single number evaluation metric

- Use a single number to evaluate & compare models.
  - => Makes selecting among multiple candidates much easier
  - => Faster to iterate and shoot for a target.

Ex: Precision + Recall vs F1 score
↳ Harmonic mean of P & R.

Optimizing vs Satisficing metrics

Optimizing: Maximize/minimize this number
Satisficing: Guard rails. Meet a certain threshold.
Ex: Accuracy vs Running time

* When it is hard to fit all metrics you care about into a single metric, make one optimizing and the others satisficing.

Train / dev / test set distributions

Make sure dev, test (and prod) come from same distribution!

-> otherwise moving the target!

Examples:

- Dev set US, UK, Europe, Test: India, China, Asia
- Dev: Middle income zip codes Test: low income zip
- Dev: Cat pics from internet Test: Cat pics from users

Guideline: Choose dev and test set to reflect data you expect and is important to do well on.

Dev & Test set sizes

- Dev set big enough to detect differences in algos you train.
- Test set big enough to give high confidence in perf of system.

When to change your cost function

- If it does not capture well what you really want to achieve
- If doing well on metric in dev and test does not mean doing well in your application

→ change it!

Comparing to human level performance.

- Bayes error: Error on best possible performance
  → likely > 0%
- Human level performance:
  → Our best proxy on a lot of tasks
  → Humans are quite good at a lot of tasks, esp. natural perception.
- While worse than humans, you can
  → Get labeled data from humans
  → Gain insight from manual error analysis
    → Why did a person get this right?
  → Better analyse bias / variance

Avoidable Bias

Cat classification	Error
Humans (proxy for Bayes)	0.5 %
	7.5% avoidable bias
Training error	8 %
	2% variance
Dev error	10%

Human Level Error

Ex:

- Typical human: 3%
- Typical doctor: 1%
- Expert doctor 0.7%
- Team of experts 0.5%

↳ Proxy for Bayes error

Surpassing human level performance

→ Not impossible
→ Can be hard, depending on problem
Examples
- Online advertising
- Product recommendations
- Logistics (prod. transit time)
- Loan approvals

→ lots of structured and quantitative data

Improving your model performance

Two fundamental assumptions of supervised learning:

1) You can fit training set well
   $\boxed{\Rightarrow \text{low avoidable bias}}$
2) Training set performance generalizes well to the dev & test set
   $\boxed{\Rightarrow \text{low variance}}$

Reducing (avoidable) Bias and Variance

Bayes error (≈ Human level)

$\updownarrow$ Avoidable bias $\leftarrow$
- Bigger model
- Longer / better optimization
- NN architecture / hyperparameters
- Features (!)

Training error

$\updownarrow$ Variance
- More data
- Regularization: L2, Dropout, Data augm.
- NN arch. / Hyperparam search

Dev Error

Week 2: Error Analysis

Q: Should you work on making your cat classifier better on dogs?

→ Evaluate ROI!

Look at 100 mislabled examples → how many are dogs. Is it worth it?

- Evaluate multiple ideas in parallel (e.g. dogs, big cats, blurry pictures, ...)
- Classification spreadsheet.

Cleaning up wrong lables

Q: Should clean up wrong lables in training set?

A: DL algorithms robust to random errors in training set. (Systematic errors are a problem)

On dev set: Impact analysis: Will fixing this help selecting between two different algos?

- Apply same method of correcting to <u>dev</u> and <u>test</u> set. These need to come from the same distribution!

<u>Strategy:</u> <u>Build first system quickly</u> then iterate!

<u>Training & testing on different distributions</u>

→ Can be ok. But algo might struggle on data it has not seen at all

→ Can augment ~~the~~ train dataset?

  → be careful when synthesizing. Can work but ensure it's not too monotonic.

How to assess bias / variance when training & testing on different sets?

- → Train-dev set: - Same distribution as train set, but used for testing
  - → Shows variance of model on distribution it was trained on
  - → Asses: overfitting or learning doesn't generalize?

Bayes error

} avoidable bias

Training error

} variance

Training-dev error

} data mismatch

Dev error

} overfitting the dev set

Test error

	Train data		Real data	
Human level	Human level	4%	6%	
		3%		avoidable bias
Error on data trained on	Train error	7%	6%	
		3%		variance
Error on data **not** trained on	Train-dev error	10%	Dev/Test error	15%

data mismatch 5%

Addressing data mismatch

- Carry out error analysis to understand difference between training & dev/test sets
- Make training data more similar

Transfer Learning

2023-07-05

or maybe add a few layers

Swap out last layer of an already trained neural net and train (only last, or all) on new task.

Depending on how much new data you have (for target task) train only last or all layers.

little data — lots of data

Makes sense if:

- Input data to the tasks is the same
- You have (a lot) more samples for the task you're transferring from.
- Tasks are similar / some reason to suspect things will transfer. Low level features from Task A could be helpful for Task B.

Multi task learning

- Learn multiple tasks at the same time, e.g. multi-object classification.

- Multi-output layer.

Loss function: Sum over individual losses

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4}\mathcal{L}(y_j^{(i)}, \hat{y}_j^{(i)})$$

- If you have some examples where you only have one / a few things labeled, then don't add the loss for those examples.

Where it makes sense

- The different tasks could benefit from similar low-level features
- Amount of data for each task is ~ similar
- Can train NN big enough to handle all tasks.