Advice for Applying ML

Repeatedly making good decisions can save a ton of time.

Debugging a learning alg.

- Get more training data
- Smaller!
- Additional features, polynomial features
- increase/decrease learning rate λ

There are some diagnostics that will advise.

Evaluating a model

- Plotting hard in $n > 2$ dims
- Compare J_{train} vs. J_{test} (regression) or count $y \neq \hat{y}$ (classification) to know whether train vs. test model is generalizable

Model Selection and Cross-Validation

Note that when using J_{test} to compare models (e.g. poly degree), you're essentially adding another feature based on test data!

→ Split into three sets
- train
- test
- validate (cross-val) (dev set)

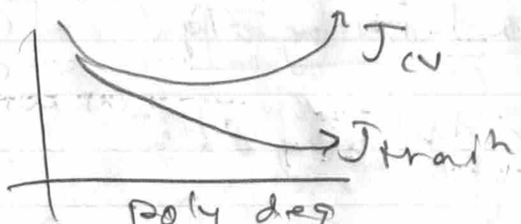
- J_{train} to fit model
- Eval model w/ J_{cv}, choose the best
- Estimate error w/ J_{test}

(Applies to NNs too, comparing architectures)

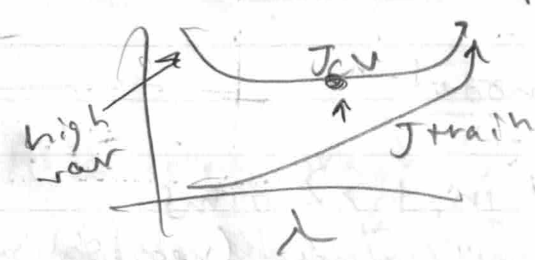
KEY: don't use test data to build any model!!

Bias and Variance

High bias = underfit. doesn't capture patterns
High variance = overfit. captures noise



Regularization λ
- Cross val can help pick λ (reg. param.) to reduce variance



Performance Baseline

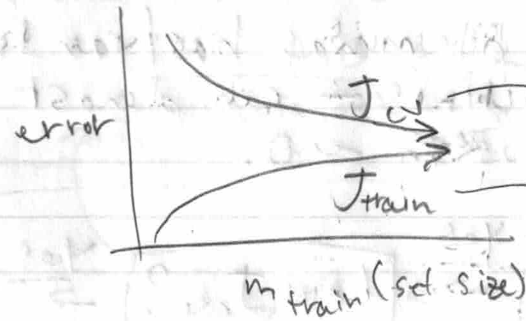
Note that human-level performance for a task can be pretty high! Compare train/test/cv error to this baseline error.

→ What level of error can we reasonably get to?

- Human perf
- Competing algs
- Experience-based guess

High Var	High Bias
Baseline/train diff low	Train/CV diff low
Baseline/CV diff high	CV/Baseline diff high

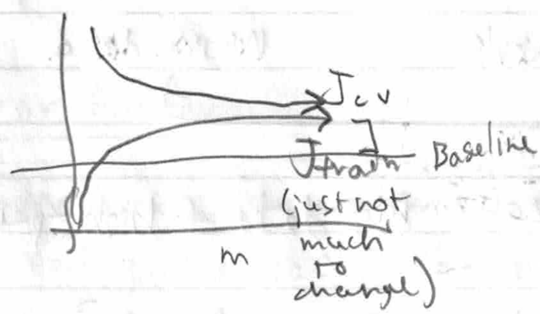
Learning Curves



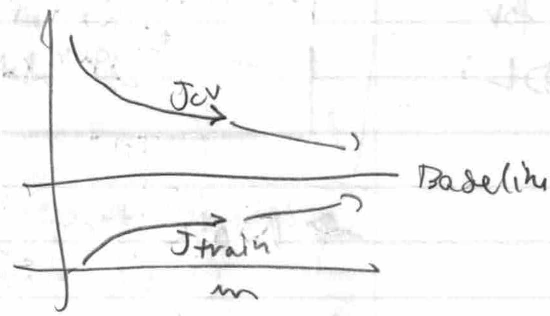
J_{cv} — better model w/ more data

J_{train} — harder and harder to fit all examples!

High Bias Model



High Var Model



"What if we get way more data??"

High Bias → Probably won't help! Note the error curves flatten out.

High Var → Can help!

☆ What to try next?

High Bias

- Add new features
- Add poly features
- Decrease λ (reg. param)

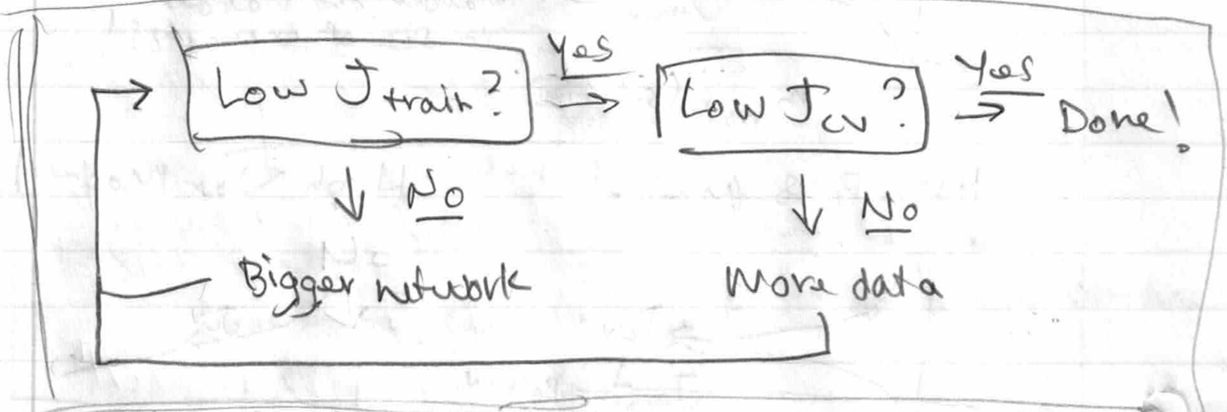
High Var

- Get more training data
- Try fewer features
- Increase λ

Bias/Variance for NNs

Traditional ML dilemma: bias/var tradeoff.
NNs: low-bias machines — can almost always get bias to ~0.

Recipe for DL:



→ Note this needs big data & high-perf HW!

Can a large NN increase variance?
→ Not with regularization!
(but it can be slower...) ($$)

ML Dev Process

Iterative loop:

Choose Arch
(model, data, etc)

Train Model

Diagnostics
(bias, var, error analysis)

Error Analysis

- Manually inspect misclassified examples
- Categorize based on common traits

May lead to:

- Need more data
- Need new features
- Craft special features
- Add heuristic transforms (eg. misspellings)

Adding Data

① Of all types? Or ② directed by error analysis?

③ Data augmentation? Modify existing training examples to get new ones. (E.g. rotating etc. images, add bg data distortions, ...)
(not random noise)

AI = Code + Data

Conventional model-centric approach → most work is here

Data-centric approach → most work is here

Data Engineering has become more impt!

Transfer Learning

Not much data? Use model trained on other data.

Eg. For digit recognition, take layers N-1 of a general img recog. model and replace output layer w/ new one, then train that network. (So pre-trained layers are the starting point.)

Supervised Pre-Training

Fine-Tuning

Option 1: Only train output layer params
Option 2: Train all params from starting point.

How does it work??
- Lower layers detect low-level patterns
- These patterns may "transfer" since they're generic

→ Can use for task w/ very few examples!

Common for GPT-3, etc.

Full Cycle

1. Scoping + project definition
2. Define + collect data
3. Train model (incl. error analysis, iterative improvements)
4. production deployment (incl. monitoring & maintenance)

loop

Deployment

- Impl. model in "Inference Server" w/API
- Rest of app uses via API
- Software Eng. problem!
  - reliability + efficiency
  - Scaling
  - logging + monitoring
  - model updates

MLOps

Fairness, Bias, Ethics

Bad things happen! — laundering bias; reinforcing neg. stereotypes

Adverse use cases — Deepfakes; Fake content; Fraud, etc.

No such thing as a checklist to follow.

Some guidelines though:

- Diverse team to brainstorm possible issues
- Lit search on industry-specific guidelines
- Audit pre-deployment
- Mitigation planning (e.g. rollbacks) and monitoring

Skewed Datasets

For datasets where pos/neg (or whatever) ratio is very unbalanced, standard metrics (e.g. accuracy) don't work well.

e.g. always predict "not a fossil"

Precision and Recall

Confusion Matrix

		Actual Class 1	Actual Class 0
Predicted class	1	TP 15	FP 5
	0	FN 10	TN 70

25 vs. 75 (not so skewed)

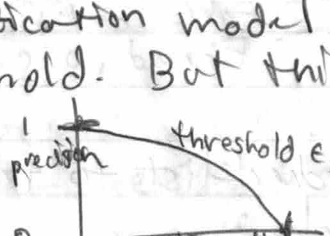
$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

This will detect a "$y=0$" classifier.

Trading off Precision and Recall

Let's say we want to increase precision in a classification model, and we increase the prob. threshold. But this reduces recall! (And vice versa)



You have to do this yourself most of the time!

But: can use $\boxed{F_1 \text{ score} = \frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)} = 2\frac{PR}{P+R}}$ This prioritizes the lower of $\{P, R\}$.