STRUCTURING ML PROJECTS:

i) Orthogonalization: Tune different knobs for solving different problems.

ii) Single Real number evaluation metric:

* Precision, Recall

F1 score $\Rightarrow \frac{2}{1/P + 1/R}$ "Harmonic mean"

iii) Optimizing vs satisficing metric:

Set one optimizing metric & many more satisficing metric.

iv) Chose dev set that reflects the data you expect to get in the future.

v) Size and of dev & Test sets:

1% 1%

vi) Change dev/test sets & metrics.

Step i) define a metric

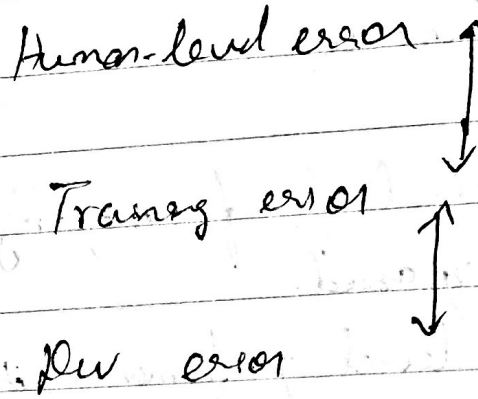
Step ii) do well on the metric

vii) Why human-level performance?
It is more doable now.
Bayes optimal error: theoretical for for mapping X to Y that can be never surpassed.
After surpassing human-level performance, it is difficult to improve till Bayes error and it takes lotta time.
To get to human level performance:
- Get data
- Manual error analysis
- Analyse bias/variance. You should know how to reduce bias/variance.

viii) Avoidable bias?
The difference b/w Bayes error & Training error.
Also Training - dev error = variance.
Try to reduce variance.

ix) Human-level performance?
Focus on bias or variance - reduction techniques depending upon the training & dev set errors.

Train error	1%	5%
Dev error	4%	10%
	high bias	high variance.



x) Surpassing human-level performance +
DL systems have been able to surpass human level perf due to training on large data.

xi) Improving model performance +
* Fit training set pretty well
~ Avoidable bias
> Training bigger n/w
* Training set performance generalizes pretty well to dev/test set.
~ Variance
> Regularization / get more training data.

AVOIDABLE BIAS : > Train bigger model
> Train longer / better optim. algos
> NN arch / hyperparameter search.

VARIANCE :-> MORE DATA

> Regularization

> NN arch / hyperparameters search

STRUCTURING ML PROJECTS II

i) Carrying out error analysis

Take ~100 mislabelled examples and find out where to focus on.

> Fix pictures on dogs / not?

Maintain spreadsheet to know where to focus on.

ii) Cleaning up incorrectly labelled data (I.L.D)

DL algos are robust to random errors but not systematic errors (white dogs = cats) in train set.

If there is I.L.D in dev set and it makes a significant diff relabelling, the re-label it, else don't.

	✗	✓	
Dev set error →	10%	2%	✓ - worthwhile
errors due to I.L.D →	0.6%	0.6	
" " to other →	9.4%	1.4%	

Correcting incorrect dev/test set examples +

* Apply same process to dev & test set to make sure they continue to come from same distribution.

* Examine right & wrong examples for mislabelling
* Train may have slightly different distribution than dev/test. Can fix that (data mismatch).

iii) Build an initial system quickly and then iterate:
- set up dev/test set & metric
- Build initial system
- use bias/variance analysis & error analysis to prioritize next steps.

* Except there is prior experience / large academic literature.

iv) Training & Testing on different distributions:
Cat example: web page (HD) - 200,000 // Mobile app - 10k

210,000 cats → shuffle →

Train	dev	test
205,000	2500	2500

$\frac{200}{201}$ = 95% → 5% mobile uploads

5% in dev & test ⇒ 119 images.

(very less)

(focusses more on web page images)

wrong ✗ due to

iii) mobile upload

Train	Test	Dev
200,000 + 5,000 (mob)	2.5k	2.5k

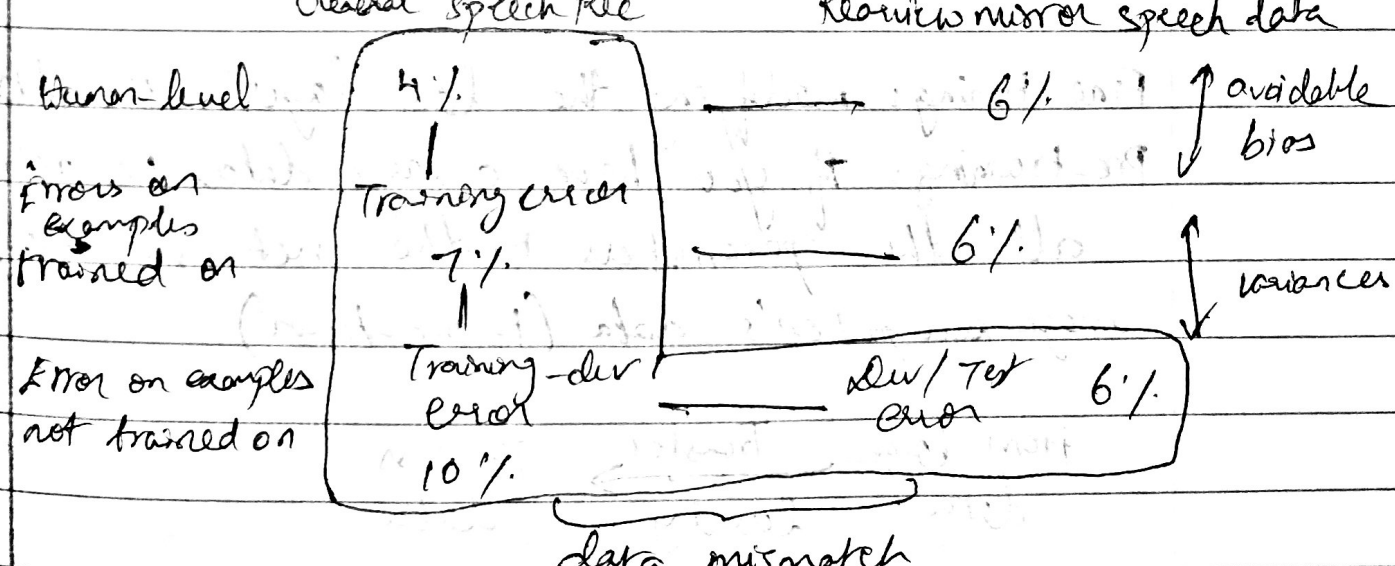
v) Bias & Variance with mismatched data distributions:

If Training error is less; but dev error is more, then is it a data mismatch or a variance problem.

To find out, we introduce train-dev set (comes from distribution of ~~dev~~ train set)

T.E	1%	1%	} variance
T.D.E	9%	1.5%	} data mismatch.
D.E	10%	10%	

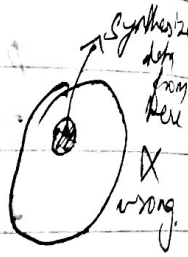
General formulation:



vi) Addressing data mismatch

* Do manual error analysis to find out how dev and train sets are different.
  - Car noise in dev
  - wrong numbers pronounced in dev.

* Do artificial data synthesis.
  - generating from a smaller subset of data makes the model to overfit



vii) Transfer learning:

Apply the learnings of one task to another task.

Fine-tuning: modifying the last layer's weights (less data)
Pre-training: If you have a large data, we retrain all the parameters in the network using image recognition's data (to pre-train)

From large data → Transfer learning → To less data.

Transfer from A → B

> Task A, B have same i/p – x

> You have more data for Task A than Task B

> Low level features from A could help B.

Transfer learning won't work when, you have less data for A and slightly more data for B.

ix) Multitask learning:

Learning to perform well on multiple tasks.

↳ objects detection on images.

When makes sense?

> tasks have shared low-level features

> Amt. of data for each task is similar

> Big enough NN than 1 NN for each task.

x) End-to-End deep learning:

Pros:

> let data speak

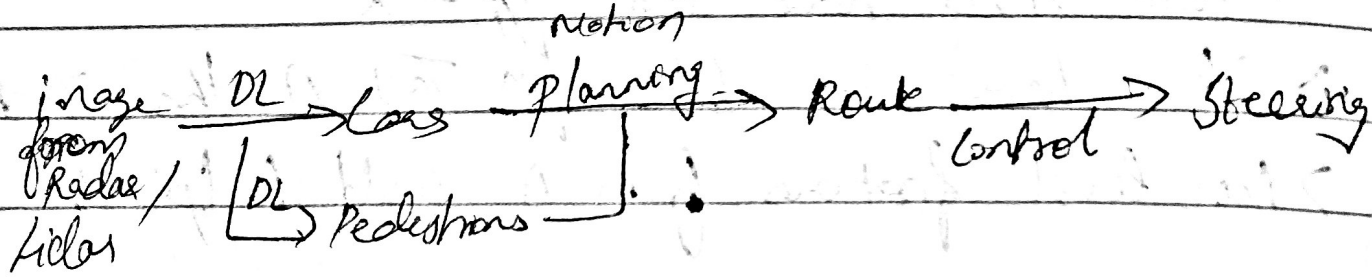
> less hand-designed components needed.

Cons:

> needs large amount of data

> Excludes potentially useful hand-designed components

key question: Have enough data?



* Use DL to learn individual components
* Carefully choose X → Y depending on what task you can get data for.

image → Steering - X wrong