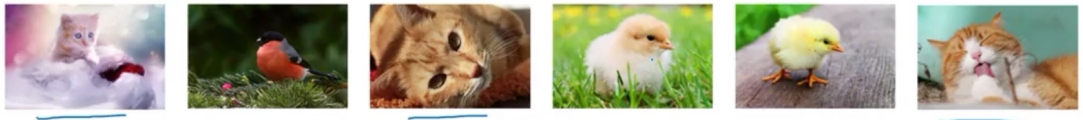
Course 3: Structuring Machine Learning Projects

week 1 Machine Learning Strategy (1)

why ML strategy?

Motivating example

e.g. Cat classifier



90% accuracy ↳ not good enough ↳ want to improve!

Ideas:

- Collect more data
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
  - Activation functions
  - # hidden units
  - ...

Andrew Ng

which one to pursue? → ML Strategy

Orthogonalisation

clear about what to tune, in order to try to achieve one effect

↳ orthogonalisation

By having orthogonal controls (not coupled with other goal) that are ideally aligned with the things we want to control → easier to tune.

Chain of assumption in ML

For a supervised learning system to do well, we usually need to tune the knobs of our system to make sure 4 things hold true

① Fit training set well on cost function
- need to at least fit training set well → pass some acceptability assessment
  - ≈ human level performance
- ↳ 1 knob to adjust, so algorithm can fit well on training set

② Fit dev set well on cost function
- ↳ separate set of knobs to tune model
  - ↳ regularization
  - ↳ bigger data set

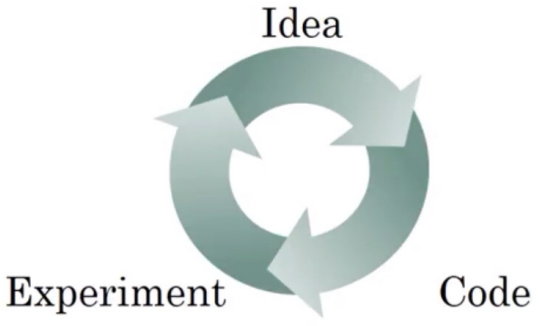
③ Fit test set well on cost function
- ↳ bigger dev set

④ Performs well in real world
- ↳ change dev set or cost function
- ∴ if model performs well on test set but not real world, either way wrong dev set distribution / cost func not doing the right thing

When training NN, maybe not using Early Stopping
- ↳ it affects fitting training set (fit less well)
  - and fitting dev set (fit better)
    - ↳ less orthogonalize

This course is about what to tune, to solve problems that limit the performance of ML in various phase above.

Single number evaluation metric



95% chance it really is a cat

from all cats pic in the set, classifier spills out 90% of them

Classifier	Precision	Recall	
A	95%	90%	92.4%
B	98%	85%	91.0%

F1 score

usually a trade-off

loosely:

Precision: of the examples that our classifier recognises as cats, what percentage are actually cats?

$$\text{Precision} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

Recall: of all the image that are really cats, what percentage are correctly recognized by our classifier

$$\text{Recall} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

with 2 evaluation metric, it may be hard to pick best model

↳ a metric that combines recall & precision → F_1 score

$$F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}}$$ harmonic mean

⇒ dev set + single real number evaluation metric

↳ speed up iterating

percentage %

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

hard to track 4 metrics
use average instead

Satisficing and Optimising metric

It's not always easy to combine all the things we care into a single real number evaluation metric

↳ setup satisficing and optimising metrics

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

say we care about accuracy (can be F_1 score) and running time

e.g. combine the 2 metrics

cost = accuracy − 0.5x run_time

maximise accuracy, but subject to running time ≤ 100 ms

accuracy ↑ optimising metric

running time ↑ satisficing metric ← just have to be good enough beyond which we don't care

Generally, if we have N metrics that we care, can pick 1 of them to optimise, then (N-1) to be satisficing.

↳ as long as it reach some threshold
↳ we don't really care

Another example: Detection of wake words / Trigger words

Care about accuracy

false positive (falsely triggers)

↳ maximise accuracy, subject to ≤1 false positive / day

Train / dev / test distribution

Cat classification dev/test sets

↳ development set, hold out cross validation set

Regions:

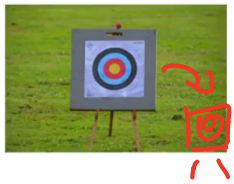
- US
- UK
- Other Europe
- South America

Dev

- India
- China
- Other Asia
- Australia

Test

bad idea
↳ Dev and Test set from different distribution



dev set + metric

different direction on test set

dev set ⟺ hold out cross validation set
also called

↳ Randomly shuffle into dev/test

↳ after trained multiple model, use this dev set to pick one

↳ then improve on dev set performance

↳ use test set to do final evaluation

Guideline:

Choose a dev set and test set to reflect data we expect to get in the future and consider important to do well on.

Dev set and test set should come from same distribution.

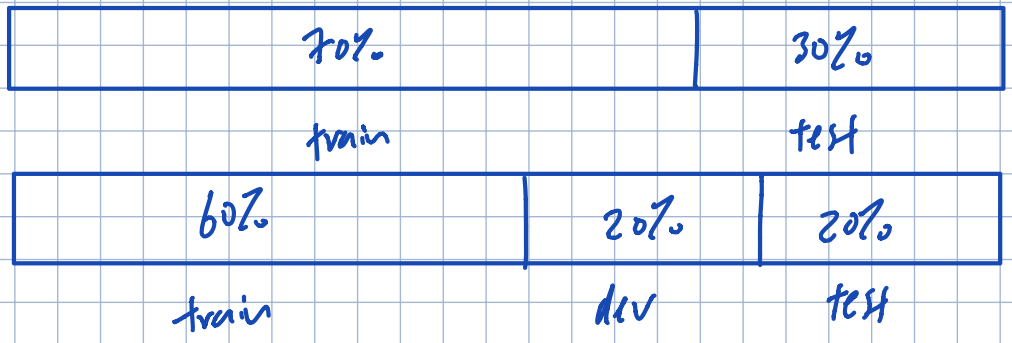
Setting Dev set + metric: define what target we aiming

test set (same distribution): help to aim final target

training set: how well we hit the target

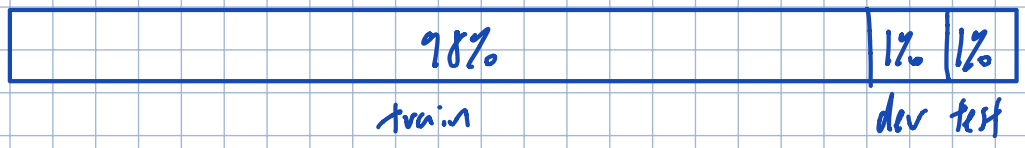
size of the Dev and Test set

old way of splitting Data



Reasonable when data are smaller

But in modern era, if we have 1 mil example (10,000 examples)



Size of test set:

Set your test set to be big enough to give high confidence in the overall performance of our system

When to change dev/test sets and metrics

e.g. Cat dataset examples

Metric: classification error

Algorithm A: 3% error ⟶ but also with porn pic ⟶ unacceptable

Algorithm B: 5% error ⟶ no porn image

↳ metric + Dev: Prefer A

↳ User: Prefer B

Error: $\frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{y_{pred}^{(i)} \neq y^{(i)}\}$ ← just count misclassified examples

$\frac{1}{\sum_i w^{(i)}}$; $y_{pred}^{(i)}$ = predicted value (1/0)

↳ problem: treat porn/non-porn images equally

↳ can add weight term: $w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn} \end{cases}$

If we find the evaluation metric is not giving the correct rank order preference for what is actually better algorithm, then it's time to think of a new metric

* Goal of evaluation metric: accurately tells us, which model is better.

Orthogonalization for cat pictures: anti-porn

1. So far we've only discussed how to define a metric to evaluate classifiers. ← place the target — Step 1
2. Worry separately about how to do well on this metric. — Step 2
   ↳ Aim accurately

Another example

Algorithm A: 3% error

✓ Algorithm B: 5% error ←

→ Dev/test

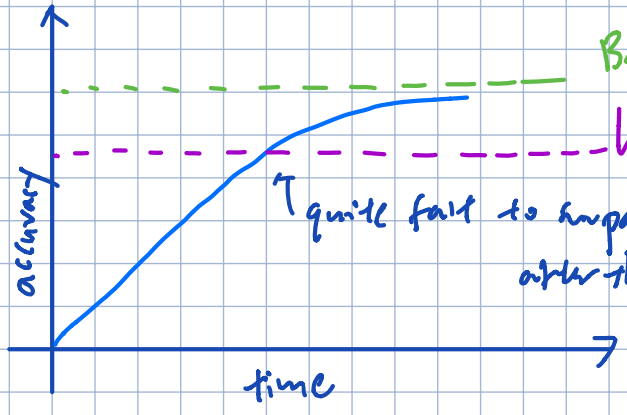
→ User images

If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

Why human-level performance?

① Advancement in Deep Learning, ML algorithm are suddenly much better → more feasible in a lot of application areas to be competitive with human level performance

② workflow of design and building ML system is much more efficient.



↳ very best func map $X \to Y$

↑ quite fast to surpass human, after that slows down

↳ reason:

① human level, for many tasks, not very far from Bayes's error

② so long your performance is lower than human level, there are tools we can use to improve. However we surpass human level

As long as ML is worse than human, we can:

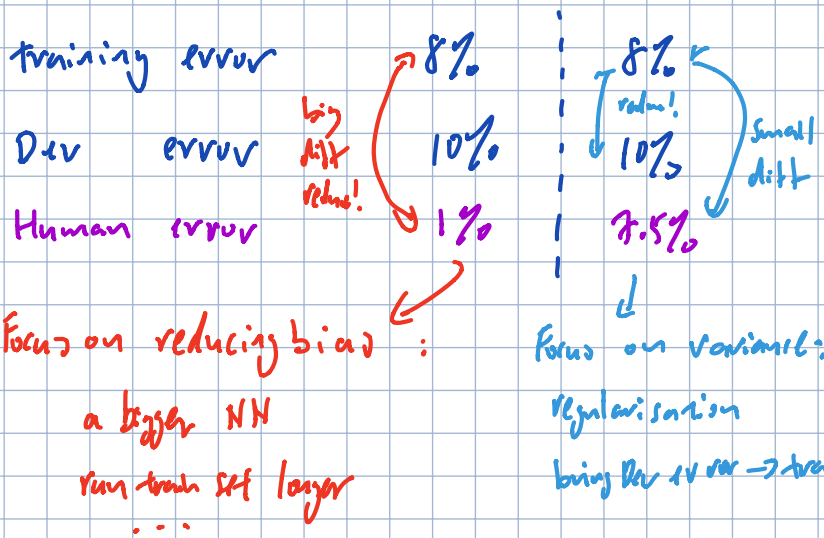
→ Get labeled data from human

→ Gain insight from manual error analysis: why did a person get this right?

→ Better analysis of bias / variance

Avoidable bias

Cat classification example



Can think human level error as an estimate for Bayes error

Avoidable Bias: Difference between Bayes error and the training error

variance : Difference between training error and dev error

understanding human level performance

Human-level error as a proxy for Bayes error

Medical image classification example:

Suppose:

(a) Typical human ..... 3 % error

(b) Typical doctor ..... 1 % error

(c) Experienced doctor ..... 0.7 % error

(d) Team of experienced doctors .. 0.5 % error

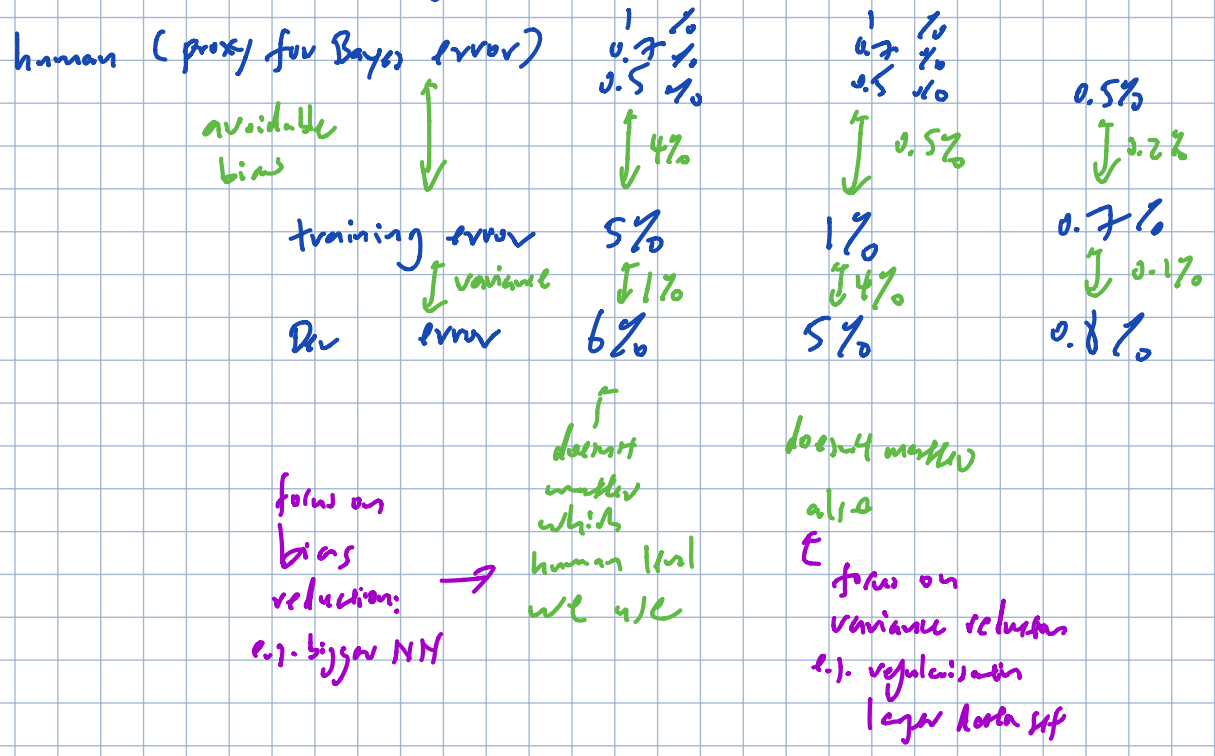
What is "human-level" error?

↳ Bayes error ≤ 0.5%

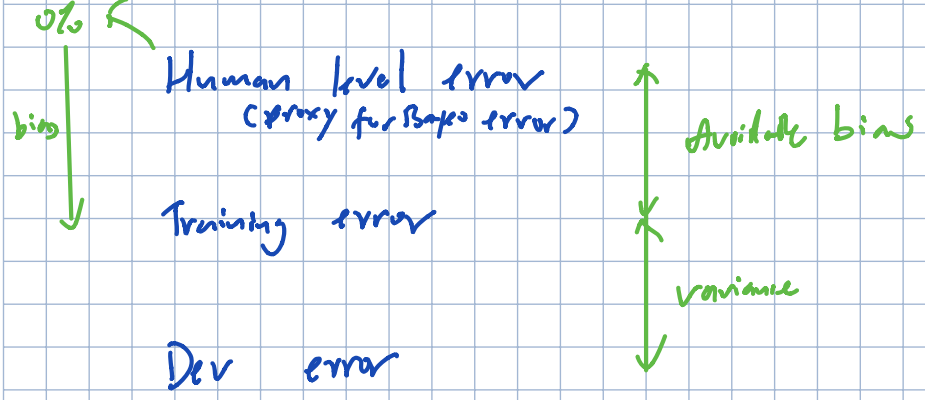
↳ 0.5% : ~ Bayes error

need to ask what is the purpose of defining human-level error

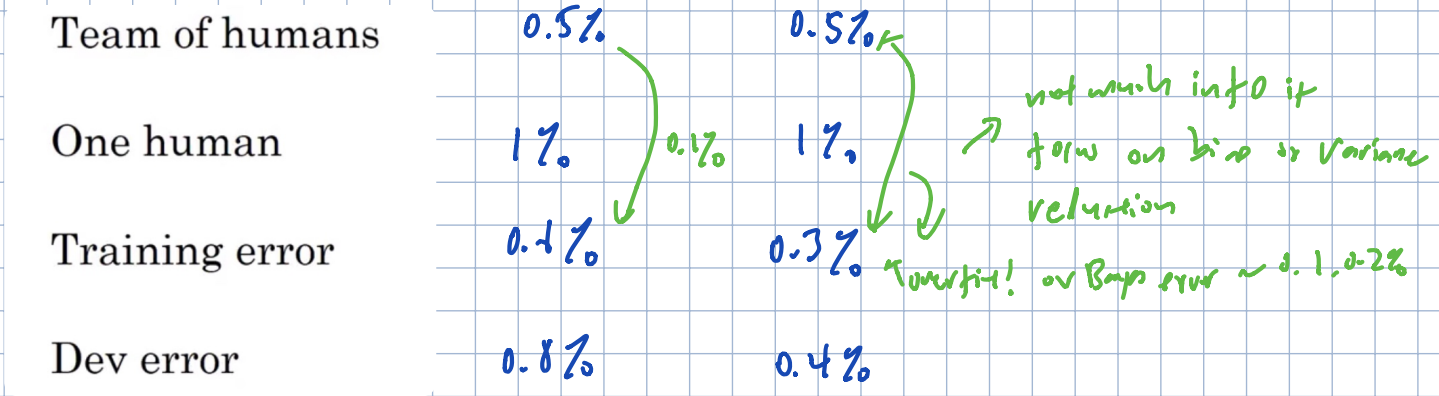
e.g. medical imaging diagnosis



Summary



Surpassing human-level performance



Problems where ML significantly surpasses human-level performance

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals

Structured data → need from DB

Not natural perception → not CV, NLP, etc

lots of data

- Speech recognition
- Some image recognition
- Medical
  - ECG, Skin cancer, ...

Andrew Ng

Improving your model performance

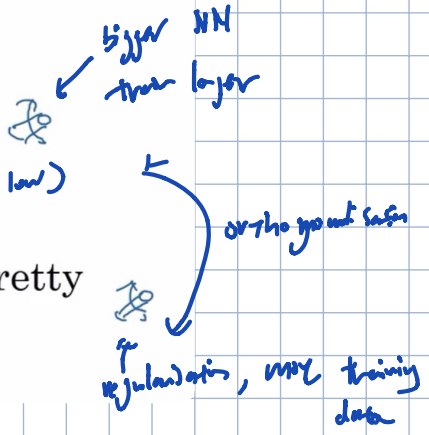
The two fundamental assumptions of supervised learning

1. You can fit the training set pretty well.

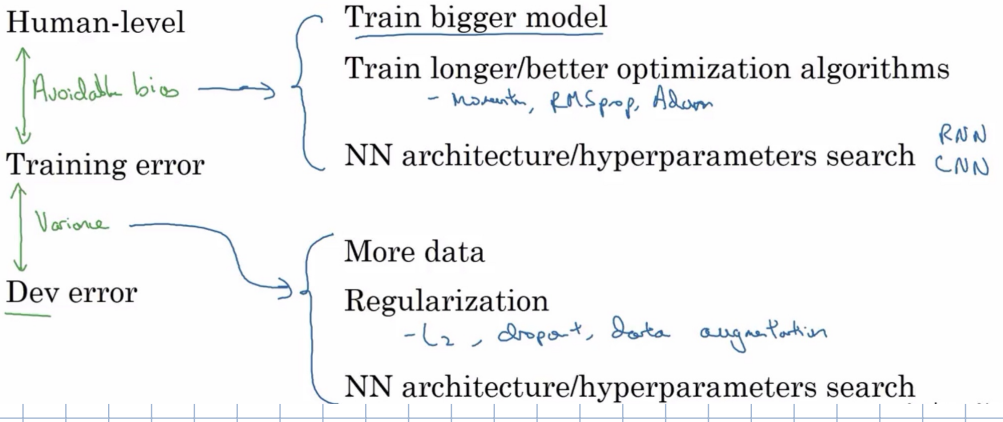
   ~ Avoidable bias (low)

2. The training set performance generalizes pretty well to the dev/test set.

   ~ Variance (low)



Reducing (avoidable) bias and variance



Week 2: Machine Learning Strategy (2)

Carrying out error analysis

If our learning algorithm is not yet at the performance of a human, we can manually examine mistakes that our algorithm is making
↳ error analysis

Look at dev examples to evaluate ideas

90% accuracy ← cat classifier
10% error
← misclassify dogs as cats!

Should you try to make your cat classifier do better on dogs? ←

Error analysis:
- Get ~100 mislabeled dev set examples. ← examine manually
- Count up how many are dogs.

↳ say 5% are dogs → 10% error to 9.5% error
if we solve the dog problem
↳ 5% relative decrease in error
from 10% → 9.5%

↳ "ceiling" ——————→

say 5% are dogs → more confidence to start on a dog problem
if we solve this. error 10% → 5%

Evaluate multiple ideas in parallel

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats ⇐
- Fix great cats (lions, panthers, etc..) being misrecognized
- Improve performance on blurry images ←

dev set misclassified examples

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮		
% of total	8%	43%	61%	12%	

ceiling for improvement

cleaning up incorrectly labeled data

By right, labeled data map $X \longrightarrow y$, what if y is mislabeled?

Incorrectly labeled examples

if we found incorrectly labeled data, what we should do?

we first consider training set

DL algorithms are quite robust to random errors in the training set as long as the errors are not too far from random → probably ok to just leave the error. → actual % of error is not too high

DL algorithms are less robust to systematic errors.

e.g. consistently labels white dogs as cats → classifier will learn to classify all white colored dogs as cats.

How about dev / test set?

Error analysis

during error analysis add in

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

Overall dev set error 10%

Errors due incorrect labels ..... 0.6% ← 6% of 10%

Errors due to other causes ..... 9.4% ←

↳ relatively small room for improvement

↳ maybe not worth the time

other example:

2%

0.6% ← now it is a larger fraction $0.6/2 \approx 30\%$

1.4%

↳ it is much more worth while to fix label!

Recall goal of dev set is to help us select between (e.g.) 2 classifier A or B.

Correcting incorrect dev/test set examples (guideline)

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong. (98-1) (2%)
- Train and dev/test data may now come from slightly different distributions.

↳ actually ok, Learning algorithm are quite robust to this

this has to be the same!

Building your first system quickly, then iterate

Speech recognition example

various directions

- Noisy background
  - Café noise
  - Car noise
- Accented speech
- Far from microphone
- Young children's speech
- Stuttering (uh, ah, um, ...)
- ...

- Set up dev/test set and metric
- Build initial system quickly
- Use Bias/Variance analysis & Error analysis to prioritize next steps.

Training and testing on different distributions

DL need a lot of data, sometimes not all training data come from the same distribution as dev/test set.

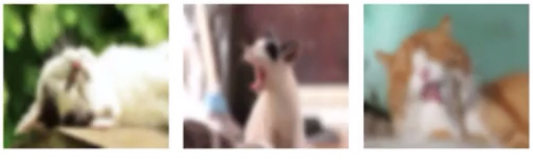
Cat app example

Data from webpages

≈ 200,000

I get from internet as more training data

Data from mobile app



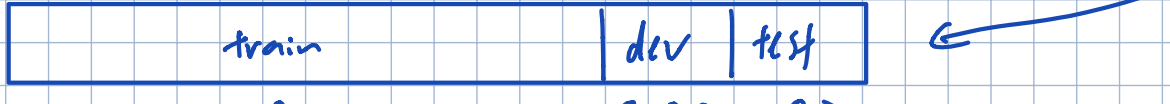
≈ 10,000

we care about this eventually

↳ what user uploaded pic will be predicted in the app

want to use both, how?

option 1: put both data set together. so 210,000 pics, then shuffle



advantage: all train/dev/test come from same distribution

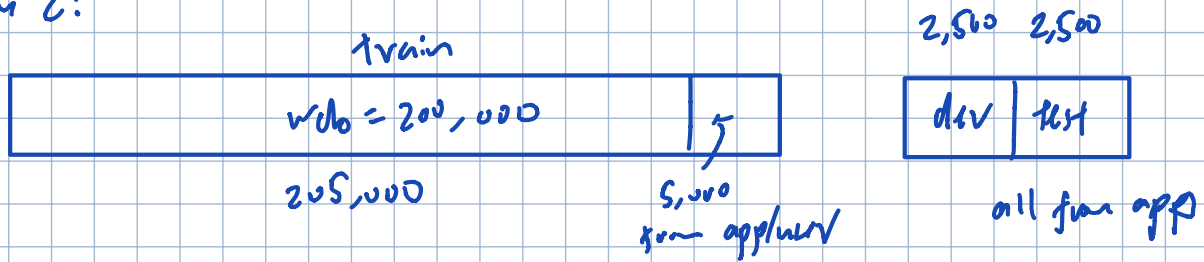
disadvantage (huge!): dev set (2,500) mostly come from internet, rather than app user (you actually care about).

Out of 2,500, 2381 from internet

↳ spent most time on wrong target! (diff between dev and actual app data!)

* This option is bad!

Option 2:



advantage: dev/test set align with final product

disadvantage: train has different distribution with dev/test set.

Option 2 will have better performance in the long term.

Speech recognition example

Speech activated rearview mirror

Training	audio | transcript
Purchased data	x, y
Smart speaker control	
Voice keyboard	
...	
500,000 utterances	

Dev/test
Speech activated rearview mirror (contains more street addresses)
20,000

5K each

actual speech from rearview mirror

Bias and Variance with mismatched data distributions

The way we analyse bias and variance changes when our training set comes from a different distribution than our dev/test set.

Cat classifier example

Assume human get ≈ 0% error

To carry out error analysis, look at

training error	1%
Dev error	10%

If they come from same distribution → large variance problem (overfit)

If they come from different distribution → can't draw the same conclusion
Maybe training set is easy/dev set is hard

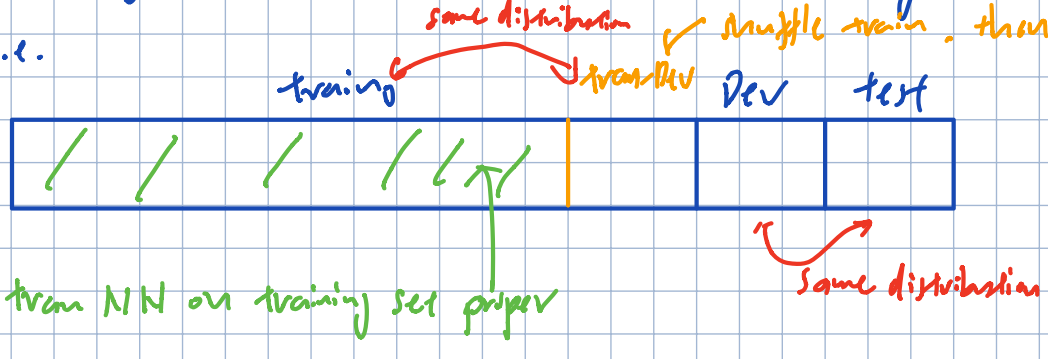
2 things changed from train set → dev set

① algorithm saw data in train set but not in dev set } hard to say where
② distribution in dev set is different } 9% error from

In order to tease out these 2 effects, define a new piece of data:

Training-dev set: Same distribution as training set, but not used for training

i.e.



e.g. training error 1% } variance problem (overfit)
train-dev error 9%
Dev error 10%

e.g. training error 1% } low variance (not overfit, can generalize)
train-dev error 1.5%
Dev error 10% } due to data mismatch (distribution)

e.g. training error 10% → avoidable bias problem (model too simple)
train-dev error 11%
Dev error 12%
(recall human error ~0%)

e.g. training error 10% → avoidable bias problem } 2 issues
train-dev error 11%
Dev error 20% } due to data mismatch

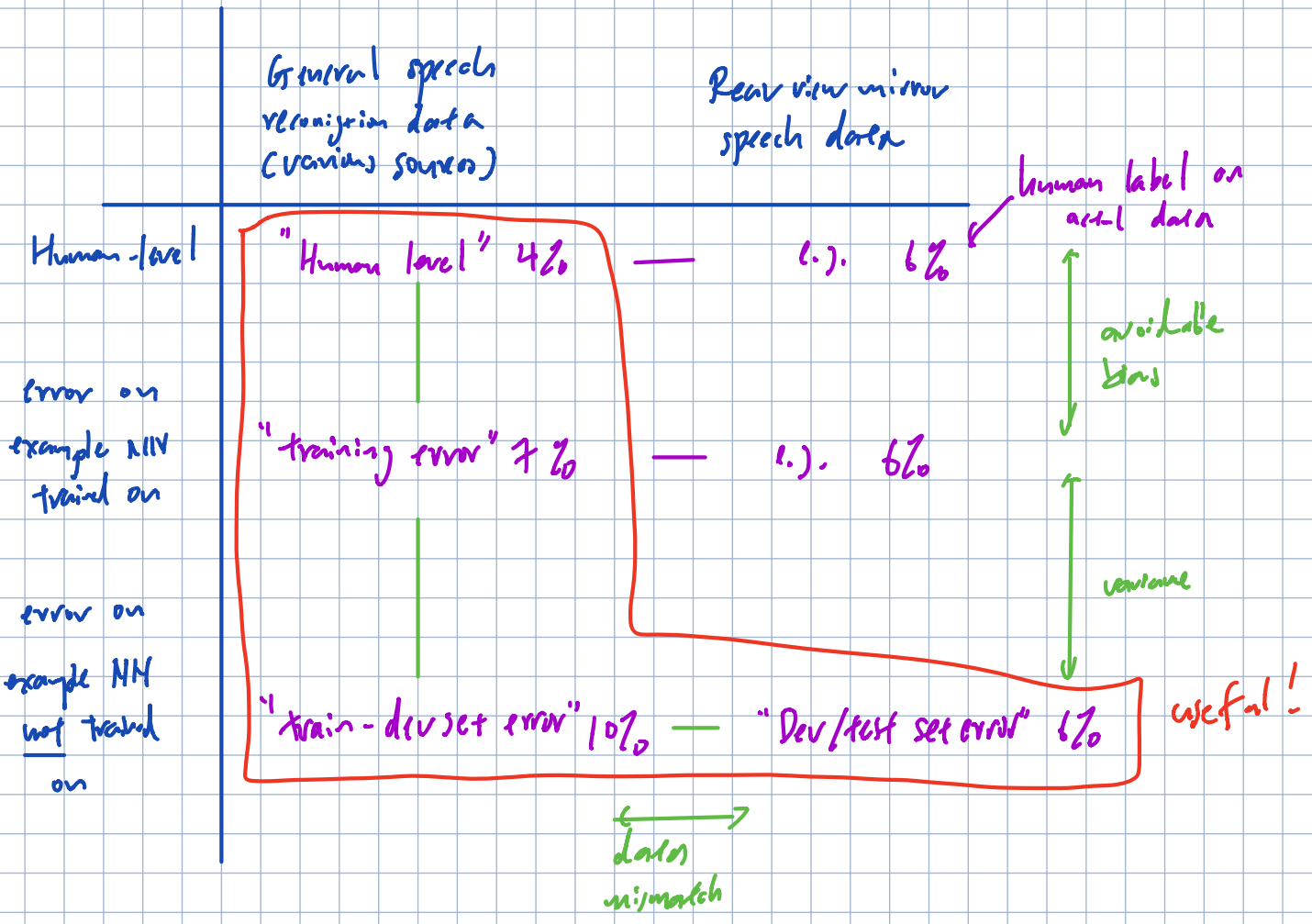
Bias/variance on mismatched training and dev/test sets

Key quantity:

Human-level error	4%	4%
training set error	7% ↕ avoidable bias	7% } train data hard
train-dev error	10% ↕ variance (generalize)	10%
dev set error	12% ↕ data mismatch	6% } dev/test data easy
test set error	12% ↕ degree of overfitting to the dev set	6%

More general formulation

e.g. Rearview mirror speech recognition



Addressing Data Mismatch

Don't have systematic solutions, can try a few things

Addressing data mismatch

- Carry out manual error analysis to try to understand difference between training and dev/test sets

  E.g. noisy – car noise street numbers

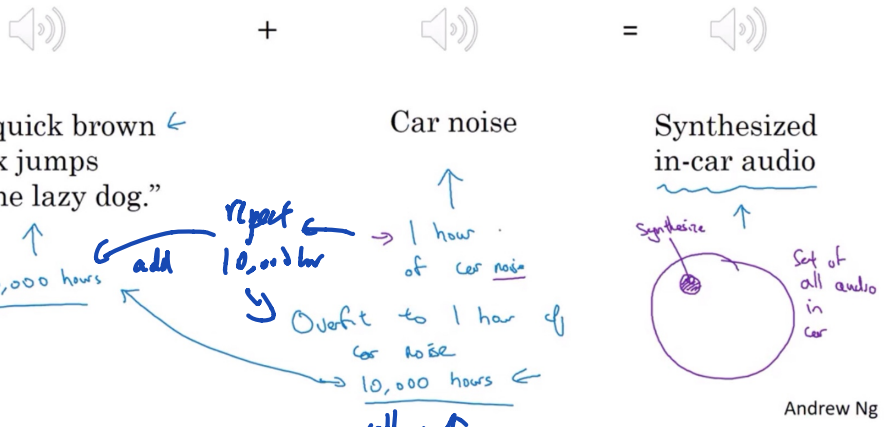
- Make training data more similar; or collect more data similar to dev/test sets

  E.g. Simulate noisy in-car data

  → to dev/test set

Can try artificial data synthesis

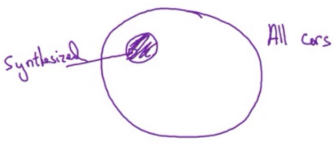
Artificial data synthesis



Andrew Ng

Car recognition:

≈20 cars



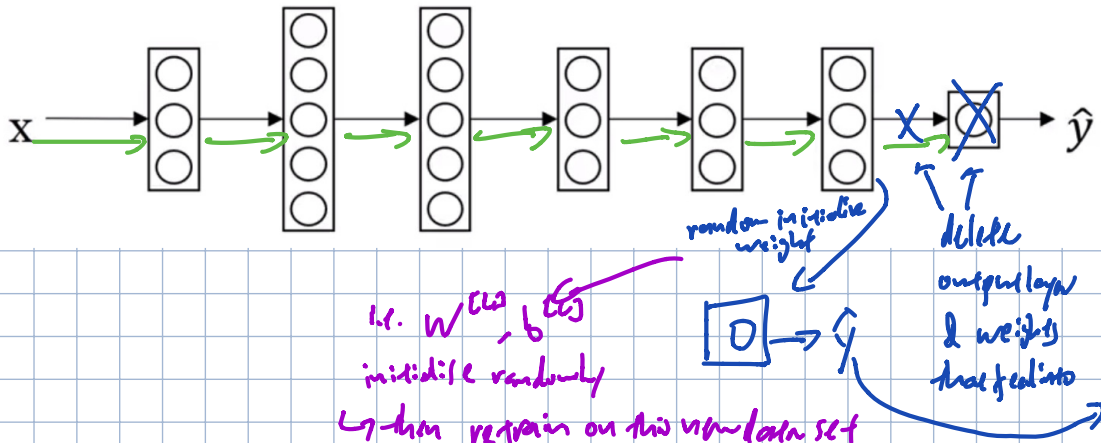
Andrew Ng

Learning from multiple tasks: transfer learning

Sometimes we can take knowledge of a NN that learned from 1 task, apply that knowledge to a separate task.

radiology image (X, y) diagnosis

image (X, y) object



if radiology data is small, might keep the previous layer weights and only train the last layer.

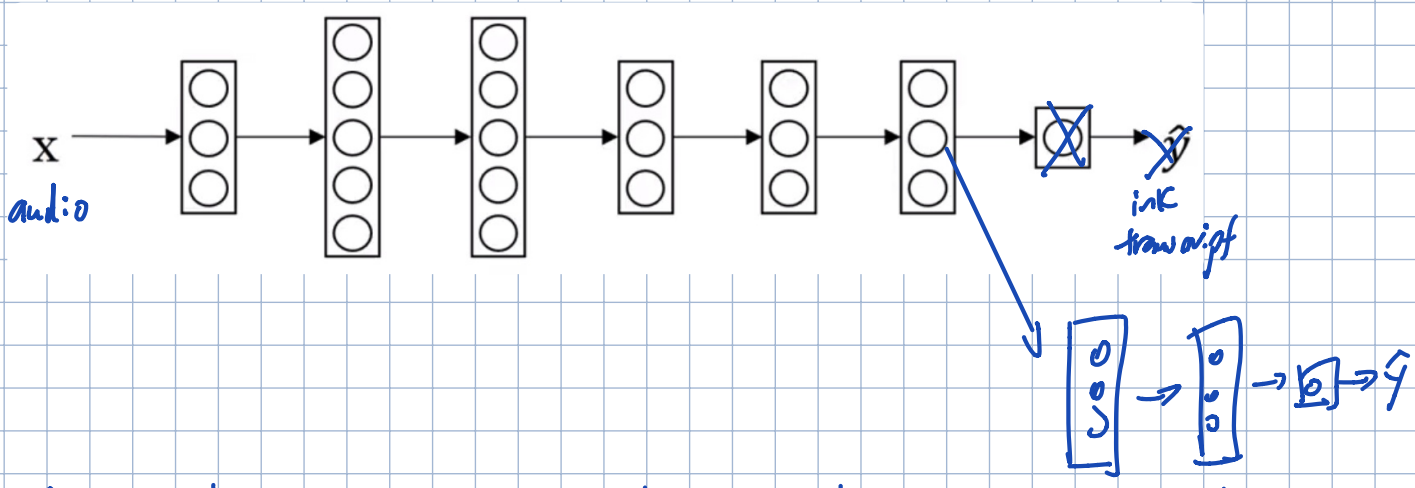
if we have enough data, can retrain all the layers.

Usually the initial (X, y) is called pre-training

train on (X, y) is called fine-tuning

A lot of the low level features, such as detecting edges, detecting curves, detecting positive objects, learned from large set of image recognition, can help our learning algorithm do better in radiology diagnosis.

Another example



Transfer learning make sense when we have a lot of data for the problem you're transferring from and usually less data for the problem you're transferring to.

When transfer learning makes sense

Trans from A -> B

- Task A and B have the same input x. images / audio
- You have a lot more data for Task A than Task B.
- Low level features from A could be helpful for learning B.

Multi-task learning

transfer learning : sequential process

multi-task learning: start off simultaneously

Simplified autonomous driving example

$x^{(i)}$

Pedestrians
Cars
Stop signs
Traffic lights
⋮

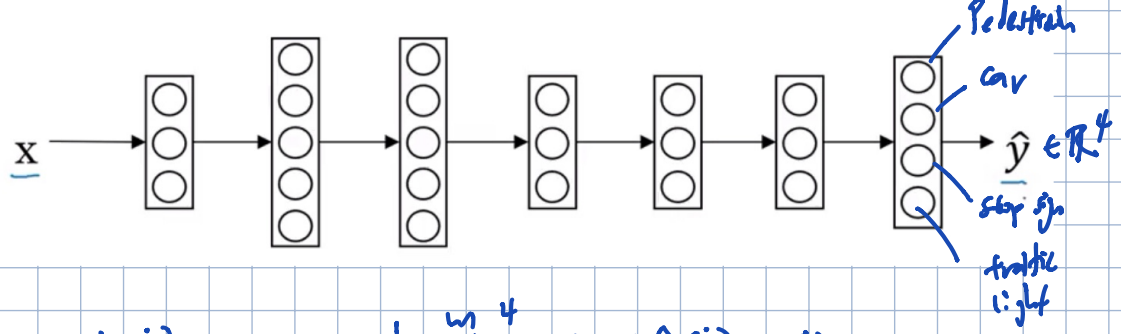
$y^{(i)}$ $(4,1)$

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix}$$

detect 4 things at the same time.

$$Y = \begin{bmatrix} | & | & | & & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \\ | & | & | & & | \end{bmatrix}$$

Neural network architecture



$\hat{y} \in \mathbb{R}^4$

Loss: $\hat{y}^{(i)}$ $(4,1)$

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

sum only over values of j with 0/1 label

↑ T diff from binary classification

↳ usual logistic loss

$$-y_j^{(i)} \log \hat{y}_j^{(i)} - (1 - y_j^{(i)}) \log (1 - \hat{y}_j^{(i)})$$

unlike softmax regression: (1 image 1 class/label)

this one image can have multiple class/labels.

↳ this multi-task learning

↳ 1 NN solving 4 problems

also works with unlabeled data

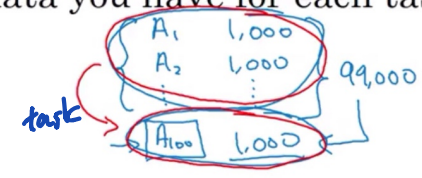
$\sum$ for j omit

$$Y = \begin{bmatrix} 1 & 0 & \cdots \\ ? & ? & \\ ? & ? & \end{bmatrix}$$

When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.

A 1,000,000 ↓ B 1,000



- Can train a big enough neural network to do well on all the tasks.

End-to-End deep learning

speech recognition example

X audio → (MFCC) extract feature → (ML) phonemes → words → transcript → Y

End-to-End

audio → transcript

Need a lot of data to work well

3,000 h of data works well on traditional pipeline,

but need 10,000 → 100,000 h of data for end-to-end pipeline

Also, sometimes we have more data for individual sub-task

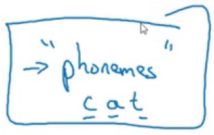
but not many data that satisfy all steps together.

whether to use end-to-end learning

Pros and cons of end-to-end deep learning

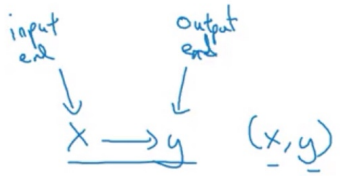
Pros:

- Let the data speak
- Less hand-designing of components needed



Cons:

- May need large amount of data
- Excludes potentially useful hand-designed components



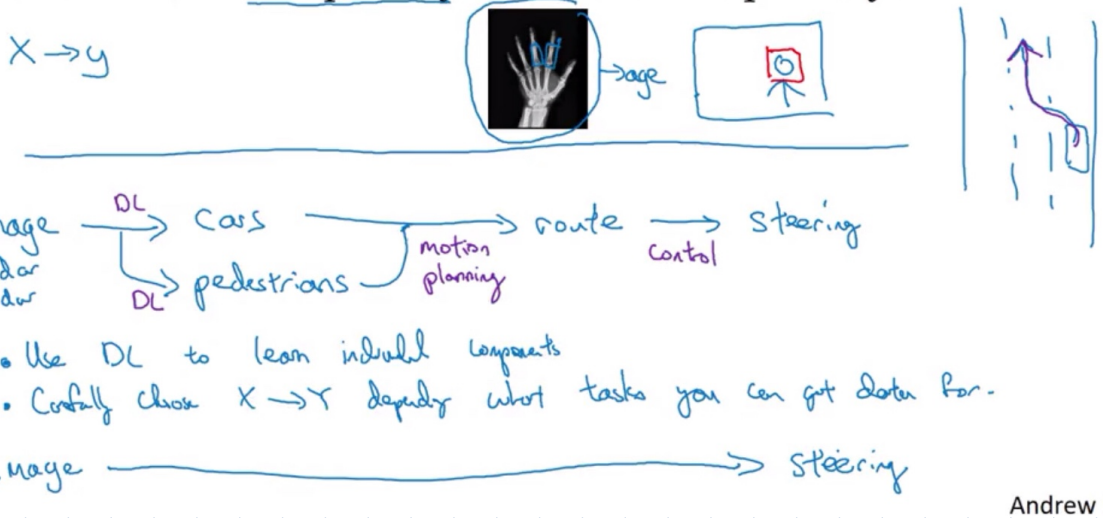
Data ↔ Hand-design

2 source of knowledge of ML

Andrew Ng

Applying end-to-end deep learning

Key question: Do you have sufficient data to learn a function of the complexity needed to map x to y?



- Use DL to learn individual components
- Carefully choose X → Y depending what tasks you can get data for.

image → steering

Andrew Ng