Week 2

Error Analysis

Carrying out error analysis:

Error Analysis - manually examining the mislabelled data by a ML model. We can use this to analyze which idea is worth our time.

Consider an example where the cat classifier is classifying certain dogs as cats.

→ Get ~100 mislabeled dev set examples

→ Count up how many are dogs.

i) If 5/100 are dogs, Then error will only reduce from 10% → 9.5%.

∴ Not worth your time

ii) If 50/100 are dogs, Then error will reduce from 10% → 5%.

∴ Worth your time

Evaluating multiple ideas in parallel:

Ideas for cat detection,

→ Fix pics of dogs being recognised as cats

→ Fix great cats (lions, panthers, etc) being misrecognised

→ Improve performance on blurry images

→ Fix instagram filters that are ruining recognition

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

These are worth working on

Cleaning up incorrectly labeled examples, is it worth the time?

→ DL algorithms are quite robust to random errors (if the data is mislabelled randomly - like one white dog) in the training set but not to systematic errors (like all white dogs labelled as cats - the algo will learn that white dogs are cats).

→ Suppose we add a column 'incorrectly labeled' to this

and it has 6%. Then,

		after reducing other causes
Overall dev set error	10%	2%
Errors due to incorrect labels ...	0.6%	0.6%
Errors due to other causes ...	9.4%	1.4%

Its more worthwhile to solve this

Since it is 30% of the errors, it is worthwhile to solve

Correcting incorrect dev/test set examples,

→ 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

→ Train and dev/test data may now come from slightly different distributions.

Build your first system quickly, then iterate

There are many ideas you can work on like in speech recognition - Noisy background, accent, far from microphone, etc.

A good strategy is to build something quick and then iterate:

→ Set up dev/test set and metric

→ Build initial system quickly

→ Use bias/variance analysis & error analysis to prioritize next steps.

Mismatched training and dev/test set

Training and testing on different distributions,

Consider cat app example:

(good quality)
Data from webpages

$\approx$ 200 k

(blurry - bad quality)
Data from mobile phone

$\approx$ 10K

↑

This is what users upload and we care about this

Then we split is as:

Train	Dev	Test
200K web + 5K app	2.5 k app	2.5 k app

This way we set the target to what our users will be uploading

How to evaluate if training data and dev/test data come from different distribution?

Assuming human error $\approx$ 0% error, lets say we get

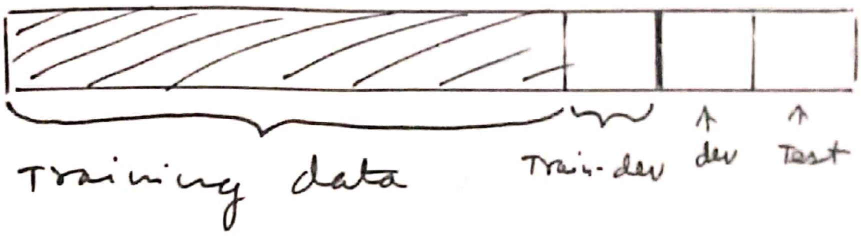
Training error ... 1%
Dev error ... 10%

} 9%

This 9% error is a result of variance and data mismatch (training data may have been more easy to recognise than dev data - like website cats are easy training data but app cats are hard dev data)

So to find out if its variance or data mismatch we use a training-dev set - same distribution as training set, but not used for training.

~~Example:~~



Examples,

1. Variance Problem

Training error	1%	Variance
Train-dev error	9%	
Dev error	10%	

2. Data Mismatch Problem

Train ~~data~~ error	1%	
Train-dev error	1.5%	Data mismatch
Dev error	10%	

3. Bias Problem

Human error	0%	Avoidable bias
Train error	10%	
train-dev error	11%	
Dev error	12%	

4. Bias & Data Mismatch error

Human error	0%	Avoidable bias
Train error	10%	
Train-dev error	11%	Data mismatch
Dev error	20%	

Bias/variance on mismatched training & dev/test sets,

Human level	4%	Avoidable bias	4%	It has decreased here since dev/test set is easier than train set
Training set error	7%	Variance	7%	
Train-dev error	10%	Data Mismatch	10%	
Dev error	12%	Degree of overfitting to dev set	6%	
Test error	12%		6%	

It may not always increase like this

More general formulation,

Data gathered from older projects →	General ~~Good~~ speech recognition	New project ↓ Rearview mirror speech data
Human level	"Human level" 4%	6%
	↕ Avoidable bias	
Error on examples trained on	"Training error" 7%	6%
	↕ Variance	
Errors on examples <u>not</u> trained on	"Train-dev error" 10%	"dev/test error" 6%
	⟷ Data Mismatch	

Addressing data mismatch,

- Carry out manual error analysis to try to understand difference between training and dev/test sets
  Eg. noisy - car noise

- Make training data more similar; or collect more data similar to dev/test sets
  Eg. Simulate noisy 'in car' data

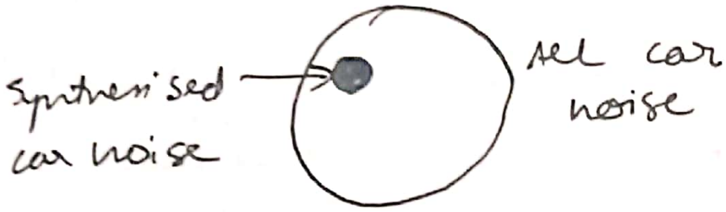
One way to do this is through artificial data synthesis.

Clear Voice + Car Noise = Synthesised in-car audio

10,000 hours 1 hour

The problem with this is, we may end up overfitting to the 1 hour of car noise.

Although car noise seems the same to us, 1 hour of the car noise we have may only be a small subset of all car noise



This is similar when people use computer generated images or video games to get more data of cars. (The video game may only have 20 models of cars

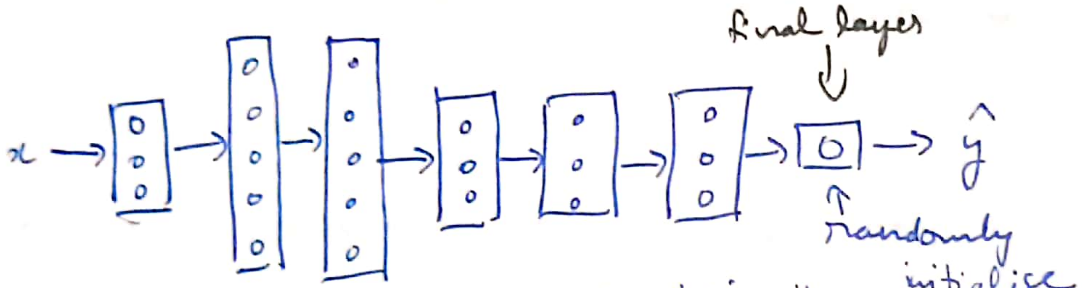
So the model will be better if we train on synthesised data made using 10,000 hours of car noise

Learning from multiple tasks

Transfer learning:

When we use a pre-trained model to train on new data (like using a ~~cts~~ cat classifier model to train a ~~xed~~ x-ray classifier).

We do this by using the old NN but removing the final layer. Instead we use a new randomised final layer and train the model using the new data



Use new (x, y) data to train it

When to use transfer learning,

→ Task A and B have same input x (like cat & x ray are both images)

→ You have a lot more data for Task A (cat) than Task B (x-ray)

→ Low level features from A (like edges and corners) could be helpful for learning B.

Multi-task Learning:

Where we design a neural network that performs multiple tasks.

Example, an autonomous driving car:

	$y^{(i)}$ - (4,1)
Pedestrian	0
cars	1
Stop sign	1
Traffic lights	0
⋮	

$$Y = \begin{bmatrix} | & | & | & & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \\ | & | & | & & | \end{bmatrix} \quad - (4, m)$$

This is different from softmax regression (the multiclass regression where an image has multiple labels) - in that it is either a pedestrian or a car or a stop sign.... but here a pedestrian and a car can be in an image (we are multitasking by recognising a car & pedestrian).

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

we sum over only values of j with 0/1 label

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

Multitask learning labels

- it can have incomplete data and it won't matter since the loss will only take into account the 0/1 labels and not ?

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

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

End-to-End Deep Learning

You can sometimes replace a multiple set of tasks into one neural network.

Example:

① audio → feature → Phonemes → Words → Transcript

② audio ——————————————————→ Transcript
END TO END DL

But for this you need a lot of data.

However in some cases its better to have multiple tasks.

Exa: In face recognition, you can have one NN to identify where the face is in a picture, crop and centre it, and then have another NN to identify the person's face.

Pros and cons of end-to-end DL,

pros:
- → Let the data speak
- → Less hand-designing of components needed

cons:
- → May need large amounts of data
- → Excludes potentially useful hand-designed components

The key question is,
Do you have sufficient data to learn a function of the complexity neededdto map x to ?

Example, in face recognition its hard to get a lot of data of every possible image ~~where~~ of a person's face position, so it makes more sense to first make it learn to detect where the face is and then detect the face's identity.