Week 1 Intro to ML strategy

Tuesday, 31 July, 2018 13:45

Why ML strategy?

Tuesday, 31 July, 2018 14:04

Learn how to get our machine learning projects working more quickly & efficiently.

MOTIVATING EXAMPLE:

We have a cat classifier w/ a 90% accuracy. We have a few ideas on how to improve the algorithm.

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

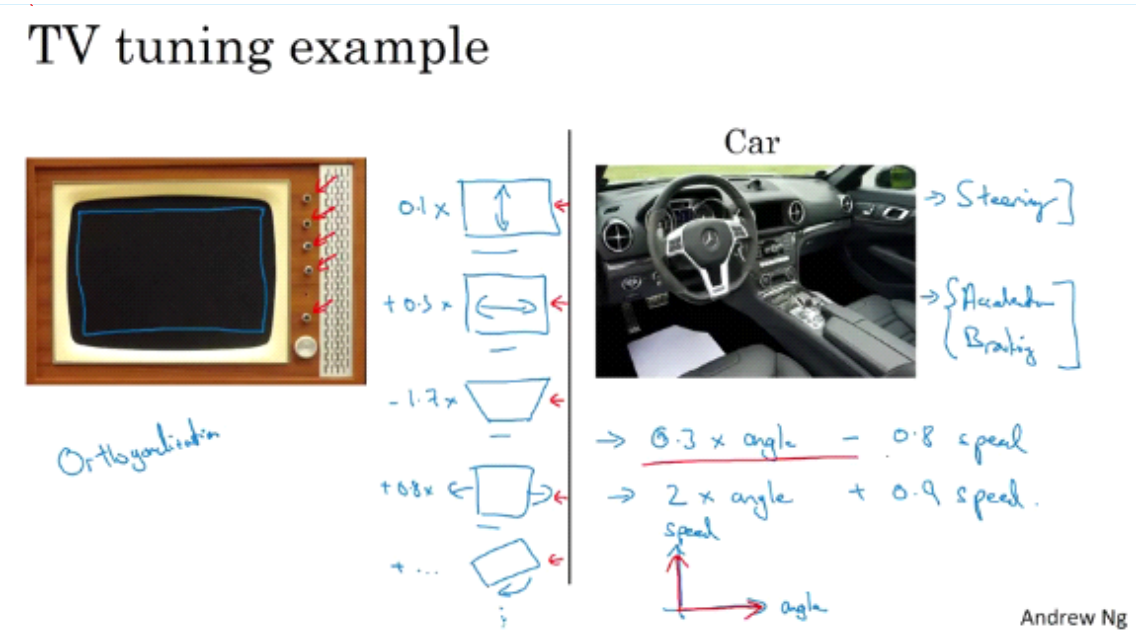
Screen clipping taken: 2018-07-31 15:34

Which path should we choose? A poor choice is going to waste our time.

Orthogonalization

Wednesday, 01 August, 2018 18:35

Orthogonalization refers to decoupling parameters in such a way that they can be tuned independently of each other.



Screen clipping taken: 2018-08-01 19:29

We want to have a knob affect only one property.

CHAIN OF ASSUMPTIONS OF ML

1) Fit training set well on cost function (≈ human-level performance)
   - IMPROVED BY: bigger network, Adam optimization, ..., early stopping
2) Fit dev set well on cost function
   - IMPROVED BY: Regularization, Bigger training set improves generalization, early stopping
3) Fit test set well on cost function
   - IMPROVED BY: Bigger dev set

↓

4) Performs well in real world.

(we have happy cat pic app users)

Change dev set or

Change the Cost function (we are not measuring the right thing)

Tend not to use early testing as it couples both train set & dev set performance.

Single Number Evaluation Metric

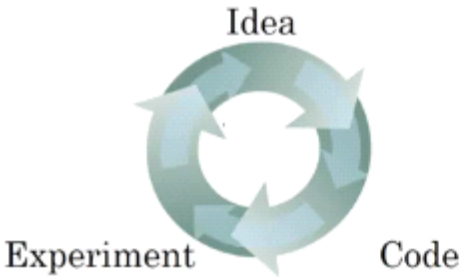
Wednesday, 01 August, 2018 19:31

Setting up our goal is more easily done if we have a single number to optimize.

i.e. a single real number that tells us if what we are trying is working better or worse than the previous idea.

MOTIVATING EXAMPLE

Machine Learning is a very iterative process:



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

F1 score
92.4%
91.0%

Screen clipping taken: 2018-08-01 19:35

Precision: of examples recognized as cats how many (percentage wise) are cats?

Recall: of all the examples that are cats how many (percentage wise) are correctly recognized as cats.

There is a tradeoff b/w precision & recall.

Problem is if Classifier A does better on Precision & Classifier B does better on recall we don't know which classifier is better.

Instead we can use an "average of both" called a F_1 score.

$$F_1 = \frac{2}{\frac{1}{\quad} + \frac{1}{\quad}}$$ ("harmonic mean")

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$ ("harmonic mean")

↳ slightly more effective than arithmetic mean.

To many ML projects having a:

- Well defined dev set (which we use to measure precision & recall)

&

- Single # evaluation metric.

allows us to speed up iterations to improve the ML algorithm

ANOTHER EXAMPLE

We are developing a cat classifier in 4 different geographical regions.

Algorithm	US	China	India	Other
A	3%	7%	5%	9%
B	5%	6%	5%	10%

← errors achieved in different geographies.

Screen clipping taken: 2018-08-01 19:48

It is tricky to decide if algorithm A or algorithm B is superior by looking at the performance in 4 different regions.

We should also compute the average and use it as the metric determining performance

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%

← best algorithm

Screen clipping taken: 2018-08-01 19:51

Satisfying & Optimizing metrics

Wednesday, 01 August, 2018 19:53

ANOTHER CAT CLASSIFICATION EXAMPLE:

optimizing ↓ satisficing ↓

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

this classifier has the highest accuracy while satisfying the ≤100ms criteria.

Screen clipping taken: 2018-08-02 19:16

In this case we care about both accuracy and running time

- cost = accuracy − 0.5 × running time.

alternatively we could choose a classifier that

- maximize accuracy subject to running time is ≤ 100ms.

in this case we say accuracy is a MAXIMIZING METRIC

running time is a SATISFICING METRIC

it has to be good enough and after that we don't care.

IN GENERAL:

Given N metrics choose 1 to optimize and (N-1) to satisfice.

accuracy (how often it wakes up when someone says the trigger word)

{

false positive rate (how often it wakes up when no trigger word is spoken)

So we want to maximize accuracy subject to having ≤ 1 false positives in 24 hr

optimizing metric

satisficing metric.

How to set Training/Dev/Test Sets

Thursday, 02 August, 2018 19:44

Setting train / dev / test sets properly can speed up training.

How to set-up Dev / Test sets

dev set: development set (aka hold out cross validation set)

- **Working Example A:**

Train multiple classifier on training set & use the dev set to evaluate the performance of each classifier until we find a classifier we are happy with that we then test on the test set.

We are building a cat detection classifier for multiple regions in the world:

Regions:
- US
- UK
- Other Europe
- South America
- India
- China
- Other Asia
- Australia

Dev

Test

Screen clipping taken: 2018-08-02 19:52

How should we set our dev set & test set.

- OPTION 1

Choose the 1st 4 for Dev set and last 4 for test set

This is a bad idea as the Dev set data and test data come from different distributions.

- OPTION 2

Choose the data for Dev & Test set so they come from the same distribution.

Randomly shuffle the data and assign it to dev or test set.

- **Working Example B:**

ML Team optimizing classifier for determining loan application approval.

$x \longrightarrow y$

borrower features — will repay loan

Training done on loan approvals of medium income zip codes.

+

Test done on loan approvals of low income zip codes.

⇓

classifier failed

(wasted ~ 3 months)

CONCLUDING GUIDELINE:

Choose dev / tests from the same distribution of data that

- is considered important to do well on
- you expect to get in the future

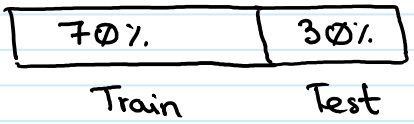
This will allow for effective learning iterations and determines what target the classifiers are trying to hit.

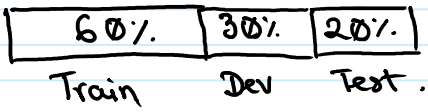
Note: The way we choose the train set determines how well we can hit this target

Size of Dev & Test sets

Friday, 03 August, 2018 14:40

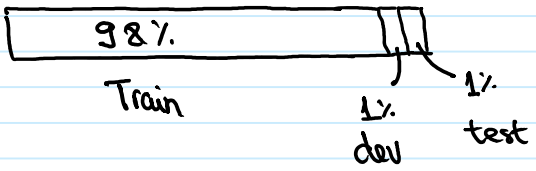
- Old way of splitting data:





This was a sensible rule when we had 100; 1,000 or even 10,000 examples.

- In the modern era of Big Data & Deep learning when we have 1,000,000 examples a more common split is.



much smaller fraction for dev & train sets.

SIZE OF TEST SET

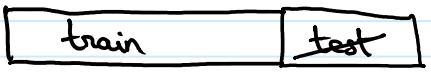
The purpose of the test set is to evaluate how well the final system performs after we finished training it.

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

Actually for some applications we don't need a high confidence in the performance of the system.

→ in this case we don't need a test set as we use the

dev set to iterate performance improvements.

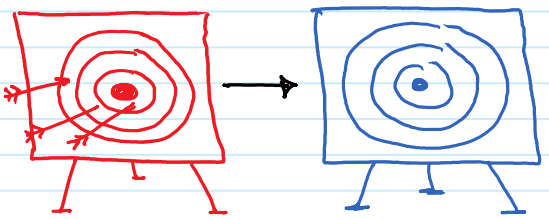


dev (some people wrongly call this test set)

When to change Dev/Test sets & evaluation metrics

Friday, 03 August, 2018 14:56

Having a dev set & evaluation metrics is similar to setting a target for the team to aim at.



Sometimes part way through the project we realize we put the target in the wrong spot. Then we should move the target.

CAT DATASET EXAMPLE:

- To start with we choose "classification error" as the evaluation metric.
- But in this case we notice we are allowing a significant number of pornographic images to be classified as cats.

$$\text{Classification Error} = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

I counts the # of misclassified examples.

$y_{pred}^{(i)}$: predicted value (0 or 1)

Metric: classification error

Algorithm A: 3% error → also shows more pornographic images

✓ Algorithm B: 5% error → higher error but no pornographic images.

Screen clipping taken: 2018-08-03 15:10

→ In this case Algorithm B is better as it doesn't show any

pornographic images even though it has higher error rate.
(worse evaluation metric).

Metric + Dev Set predict Algorithm A
Us + Users prefer Algorithm B

When the evaluation metrics mispredict the ranked order of the prefered algorithm it is a sign that we need to change our evaluation metrics (or the dev/test sets)

The problem with our "classification error" evaluation metric is that it treats all mismatches equally but what we want is to not mislable any pornographic images.

One way to improve the "classification error" is to add a large penalty when the algorithm makes a misclassification error using a pornographic image.

change the denominator so the average is still b/w 0 & 1.

$$\text{New Classification Error} = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathbb{I}\{y_{pred}^{(i)} \neq y^{(i)}\}$$

$$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-porn} \\ 10 & \text{if } x^{(i)} \text{ is porn.} \end{cases}$$

(in this case we penalize a porn misclassification 10x more than non-porn).

NOTE: To implement this weighing we need to go through our dev & test

set and label porn images.

- TAKE AWAY POINT:

  The reason for the evaluation metric is to allow us to choose the better classifier for our application given a set of multiple classifiers.

  - so if we are not satisfied with our evaluation metric we shouldn't keep using it and try to find a new one that better captures our preferences of what we consider a better classifier.

ORTHOGONALIZATION FOR CAT PICTURES: ANTI-PORN

Until now we have only discussed how to define metrics to evaluate classifiers.
→ they help us rank order classifiers that perform at different levels.

Another discussion should be about how to do well on this metric.
→ minimizing:

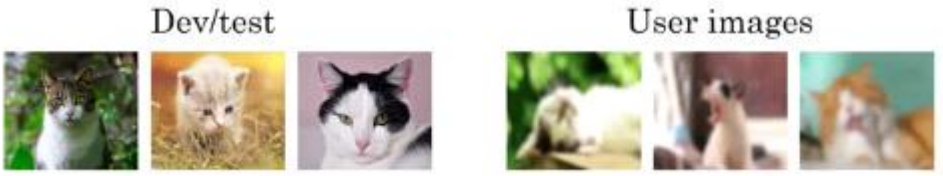
$$J = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m} w^{(i)} \mathcal{L}(\hat{y}^{(i)}, y^{(i)})$$

Keeping these discussions separate is an example of ORTHOGONALIZATION.
→ don't couple "placing the target" with "aiming accurately at the target"

ANOTHER EXAMPLE:

well shot:
- good lighting & framing, high resolution

poorly shot:
- blurry, low resolution, low lighting.



Screen clipping taken: 2018-08-03 16:13

The problem in this case is that even though the metric perfoms well on the dev/test set it perfoms poorly on the user set (real world usage)

→ so in this case we need to change our test / dev set (and maybe even our eval. metric).

The dev/test set is not predictive of the real world usage.

RECAP:

- Set an eval metric that's "good enough"
- Make sure dev/test sets are from same distribution
- Make sure test set is predictive of real world usage.
- Iterate
- Update eval metric or dev / test sets as needed

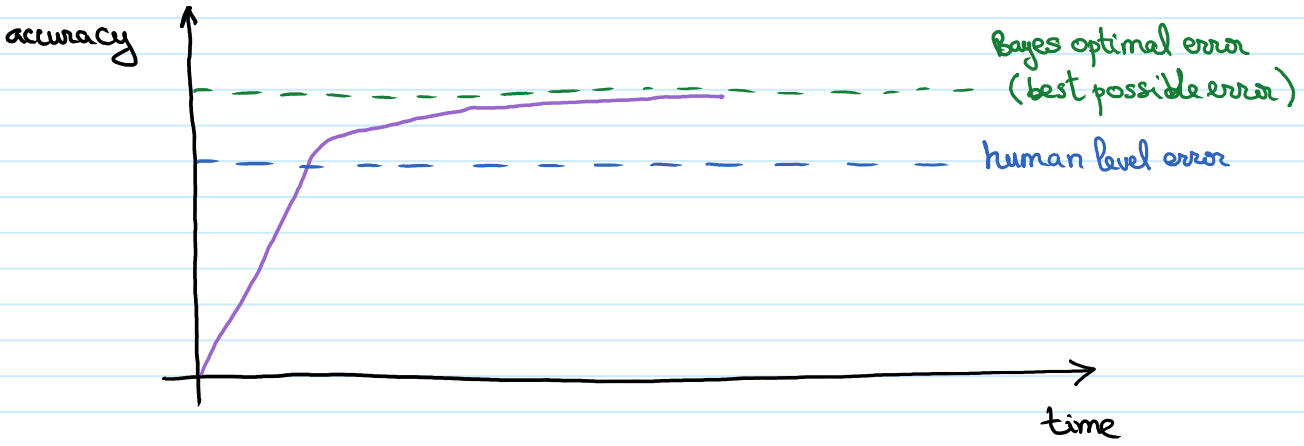
Why human-level performance?

Friday, 03 August, 2018 16:29

Why compare to human-level performance?

→ as ML has improved it has reached levels that it allows its performance to be competitive with human performance

→ in domains that use ML the largest advantages (so far) are obtained for tasks that mimic human activities.



Bayes optimal error (Bayes error)

→ there is no way for a function mapping from X → Y to surpass this level of accuracy.

EXAMPLE:

	X	→ Y	
Speech recognition:	audio clip	transcription	Some audio is so noisy that it can't be transcribed
Cat recognition:	image	cat	image is so blurry that it is impossible to determine if a cat is

in the picture.

Learning progress is rapid until accuracy reaches human level performance after which it slows down dramatically.

→ CAUSE 1: Human level performance in many cases is close to Bayes optimal error so there is little room for improvement.

→ CAUSE 2: When performance is below human level there are many tools available that can improve performance.

TOOLS TO IMPROVE PERFORMANCE FOR SUB-HUMAN LEVEL

- Get labeled data from humans.
- Gain insights from manual error analysis.
  - → why did humans get it right but the algorithm got it wrong?
- Better analysis of bias / variance.
  - → Knowing what the human bias/variance is helps improve performance.

Once the algorithm is doing better than humans these tools are not easily applied.

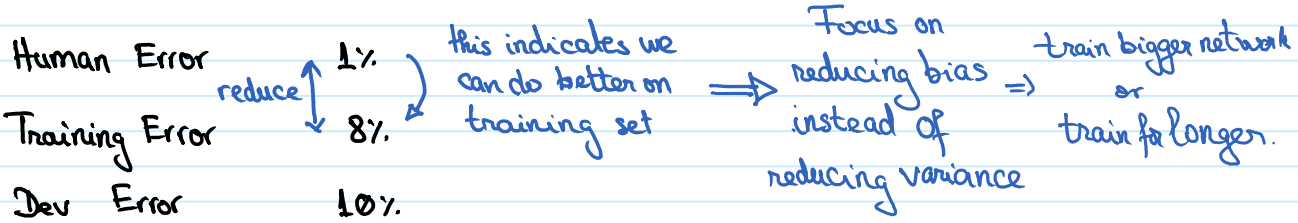
Avoidable Bias

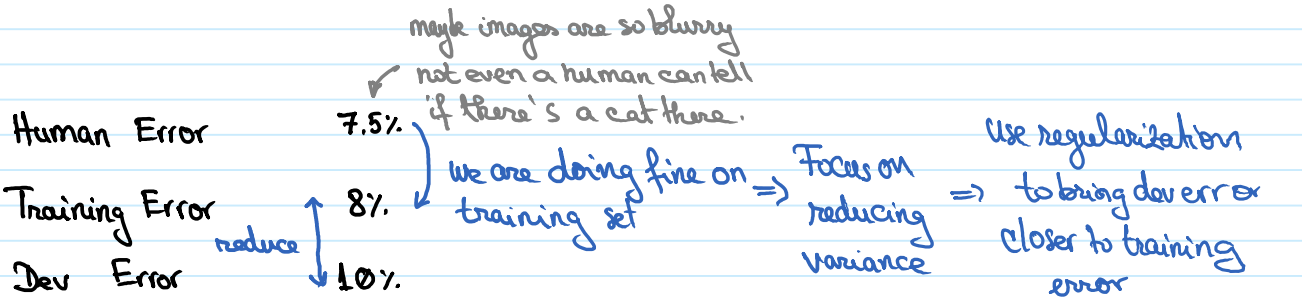
Friday, 03 August, 2018 17:00

We want our learning algorithm to do well on the training set. But we don't want it to do too well (overfitting). Knowing what human level performance is can prevent overfitting.

CAT CLASSIFICATION EXAMPLE:

Which tactic should we focus on: bias reduction or variance reduction?





In earlier discussions we were assuming Bayes error to be ~ 0%.

As a proxy for Bayes error Human-level error for vision tasks is a good proxy.

→ humans perform very well at vision classification.

→ though by definition Human-level error is worse than Bayes error.

Not common but useful terminology: (ideal error)

Avoidable bias: Difference b/w Bayes error and Training error.

Variance: Difference b/w Training error and Dev error.

Understanding human level performance

Friday, 03 August, 2018 20:02

What is the most useful definition of human-level performance?

→ human-level error is useful to be used as a proxy for Bayes error (the lowest level of error achievable given the data)

HUMAN-ERROR AS PROXY FOR BAYES-ERROR

MEDICAL EXAMPLE:

How should we define human-level error? is it the error of a typical human?

Medical image classification example:

Suppose:

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

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

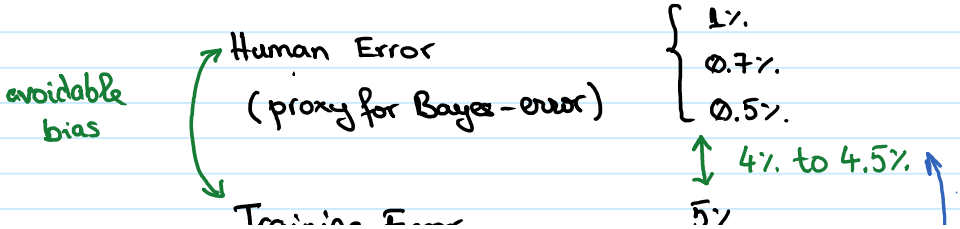
(c) Experienced doctor ..... 0.7 % error

(d) Team of experienced doctors .. 0.5 % error → we know Bayes error ≤ 0.5

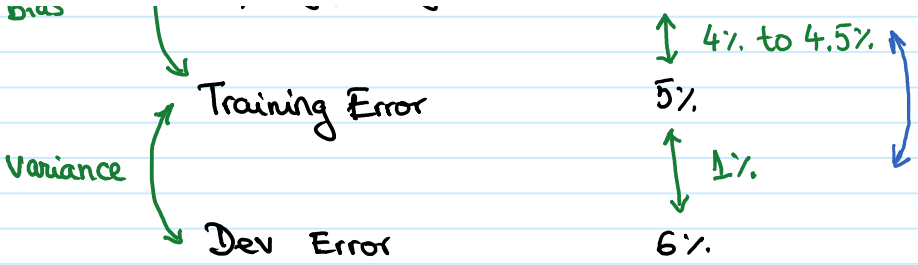
so for purposes of determining whether there is room to minimize avoidable bias this is the useful definition of human-level error.

Screen clipping taken: 2018-08-03 20:07

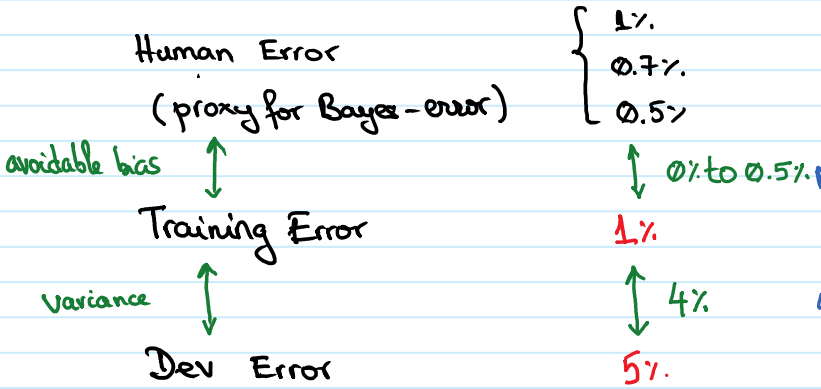
ERROR ANALYSIS EXAMPLE



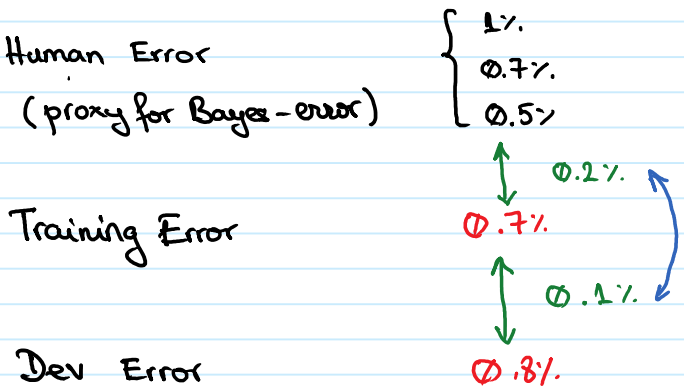
In this example it doesn't matter which definition of human-level error we use



number which definition of human-level error we use the avoidable bias (4% to 4.5%) is larger than variance so focus on that (i.e. train larger network)



again it doesn't matter which definition of human-level we use the dev error is the largest and we should focus on addressing variance reduction (i.e. regularization)



WE ARE DOING REALLY WELL

In this case we must use the best human-level error to correctly determine whether to focus on reducing avoidable bias or variance

REMEMBER

- We use Human-level error as a proxy for Bayes error (the ideal base level error which is not always 0 (it depends on how noisy the data is).

- Use (Human-level error - Training error)

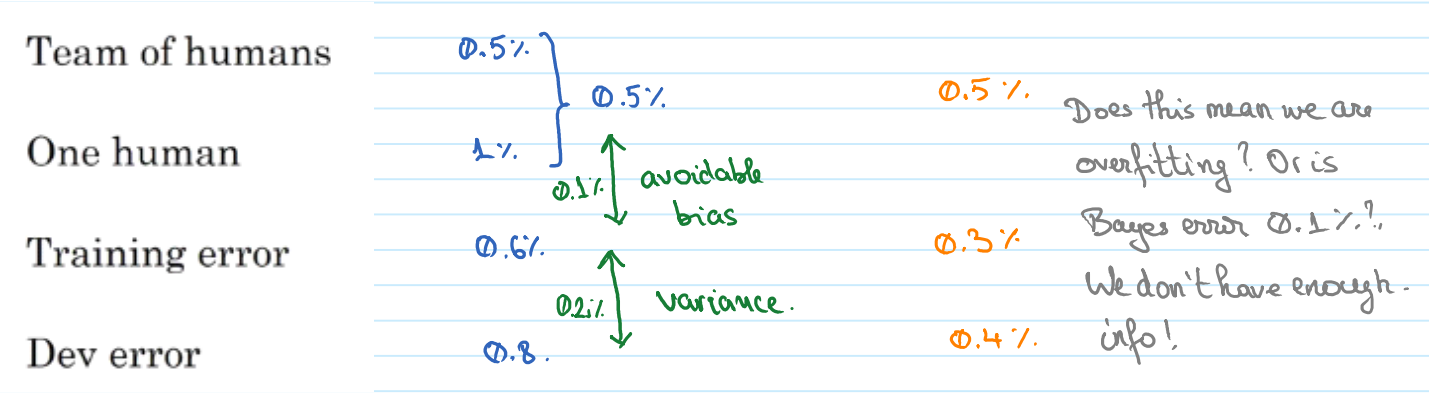
  vs

  (Training error - Dev error)

  to determine whether to focus on reducing avoidable bias or variance.

Surpassing human-level performance

Saturday, 04 August, 2018 16:29



Screen clipping taken: 2018-08-04 16:34

After the classifier used surpasses human-level performance we can no longer get a higher limit for Bayes error. (the ideal error) so we don't know what to focus on (bias or variance reduction) and progress slows down.

Additionally once we surpass human-level performance we can no longer use human intuition to improve performance.

→ We can still make progress but slower.

DOMAINS WHERE ML SURPASSES HUMAN-LEVEL PERFORMANCE

- Online advertising.
- Product recommendations.
- Logistics (predicting how long it takes to go from A to B)
- Loan approvals.

All these examples learn from structured data - data where we know what the previous decision was and its outcome.

These are not natural perception domains (i.e. computer vision or speech recognition or natural language perception)

In the examples above the ML analyzed more data than any human could have done.

However there are some problems in the natural perception domain where ML has surpassed human-level performance:

- some speech recognition
- some image recognition
- Medical: ECG analysis, skin cancer detection

but this is harder as human-level error in this domain is close to Bayes-error.

Improving model performance

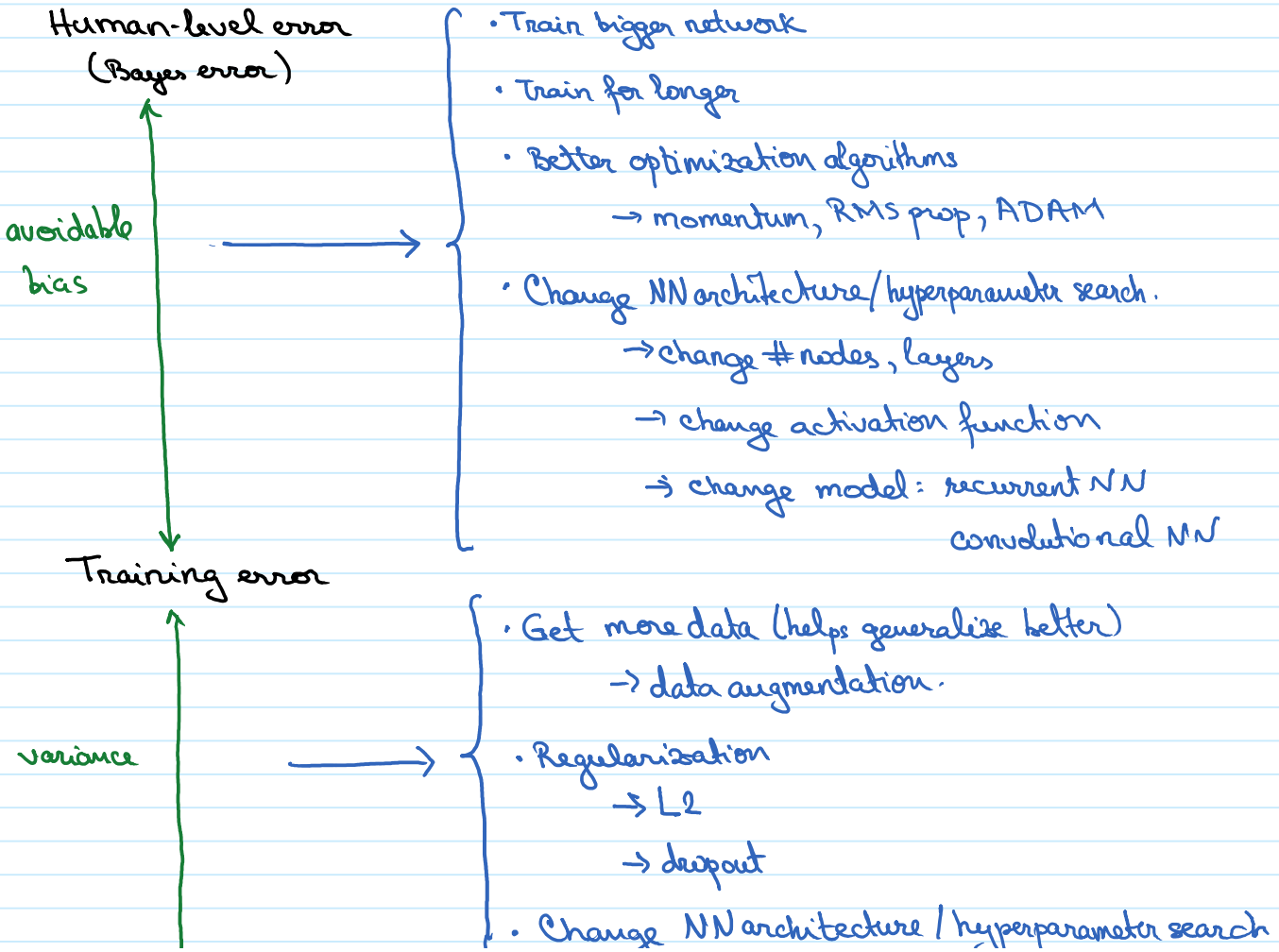
Saturday, 04 August, 2018 16:51

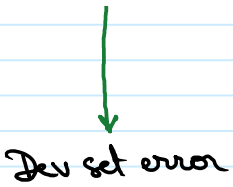
ASSUMPTIONS:

1. We can fit the training set well (aka achieve low available bias)
   → improved by training bigger network, training for longer.

2. Train set performance generalizes well to dev / test set performance.
   (aka variance is not bad)
   → improved by regularization, getting more data

TECHNIQUES TO IMPROVE PERFORMANCE





→ overfit

- Change NN architecture / hyperparameter search

Week 2 ML strategy 2

Sunday, 05 August, 2018 15:36

Error Analysis

Sunday, 05 August, 2018 15:38

Manually reviewing why the ML algorithm hasn't reached human-level performance is called Error Analysis.

REVIEW DEV EXAMPLES TO EVALUATE IDEAS

Cat classifier performance

accuracy : 90% <=> 10% error.

Notice that the cat classifier recognizes some dogs as cats.

Screen clipping taken: 2018-08-05 18:51

Want to improve dog performance:

→ collect more dog pictures

→ design features that are dog specific.

QUESTION: Should we address the dog problem?

How do we assess if it is worth our effort to solve the dog problem?

ERROR ANALYSIS PROCEDURE

- Get ~100 mislabeled dev set examples (all types)
- Count manually how many of them are pictures of dogs.

★ Say 5% of the 100 mislabeled images are pictures of dogs.

→ so our 10% error will go down to 9.5%.

best case (if we fix the dog problem perfectly)

→ this gives us a "ceiling" performance improvement

★ Say 50% of the 100 mislabeled images are pictures of dogs.

→ so our error could go down from 10% to 5%.

which might be worthwhile.

So in 5 to 10 minutes we can evaluate if a single idea is worth pursuing.

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

Screen clipping taken: 2018-08-05 19:03

Create a table as below and classify manually each picture

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

Screen clipping taken: 2018-08-05 19:04

As we classify we might notice other classes in this case we should

add another column to our table to account for this new group. (i.e. Instagram filters)

- This procedure will give us a sense of what optimization is worth pursuing.
- It also provides an upper limit on the performance improvement expected if this optimization is resolved.

Cleaning up incorrectly labeled data

Sunday, 05 August, 2018 19:13

Data used to feed the neural net is comprised of input X & output Y.

What do we do if we determine some of the output Ys are incorrect?
→ is it worth while fixing these labels.

CAT CLASSIFIER WORKING EXAMPLE

$$Y = \begin{cases} 1 & \text{is cat} \\ 0 & \text{is not cat} \end{cases}$$

Screen clipping taken:
2018-08-05 19:17

★ TRAIN SET MISLABELED DATA

ML algorithms are relatively good at dealing with a small percentage of RANDOMLY mislabeled data in the training set

→ so if you don't have time to fix the labels and there's not too many of them and they are not systematic (i.e. the mislabeled data consistently label white dogs as cats) then the ML algorithm will probably still work.

★ DEV / TEST SET MISLABELED DATA

Add one extra column during Error Analysis to identify Incorrect Labels

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%	

Screen clipping taken: 2018-08-06 22:24

So in this case the Error is not a result of misclassification of the ML algorithm → it is just the wrong label was assigned.

As with other causes of Error if the % total errors associated with this type of error is significant it is worthwhile to go in and fix the dev/test set, otherwise it's probably a waste of time.

Andrew Ng recommends looking at:

6% of 10% 30% of 2%

Overall dev set error	10%	2%
Errors due to incorrect labels	0.6%	0.6%
Errors due to other causes	9.4%	1.4%
	probably not worthwhile fixing mislabels	more important to fix mislabels

OTHER CASES WHEN DEV SETS NEED CLEAN-UP:

Goal of Dev Set is to help rank classifiers so we can select "the best one".

If two classifiers have error rates of:

A: 2.1%.

B: 1.9%.

and there are 0.6% mislabeled data in the dev sets we can no longer know if classifier B is better than A intrinsically or because of the mislabeled data.

In this case we should clean-up the Dev Set.

CORRECTING MISLABELED DEV/TEST SET EXAMPLES

1) Whatever correction process is used for the dev set we need to make sure to use it on the test set also → this maintains the same data distribution b/w dev & test sets.

REMEMBER: Dev Set indicates where the target that needs to be achieved is.

2) Consider both examples that the classifier (algorithm) got right as well as examples that it got wrong.

→ If we fix only the ones it got wrong we bias the algorithm

- White dog mislabeled as cat → algo labels as dog ("WRONG" CALL)
- White dog mislabeled as cat → algo labels as cat ("RIGHT" CALL)

This is not easily done because if the classifier is any good there are

many fewer examples it will get wrong than it gets right.

i.e. for a 98% accuracy classifier

2% of examples are wrong
98% ——v—— right
} 39x more correct examples to check for.

3) If we don't also correct the train set for mislabeled data (maybe it's too large for the clean-up to be feasible) the train and dev/test sets now come from different distributions.

→ this is OK

- IT IS VERY IMPORTANT THAT DEV & TEST SETS COME FROM THE SAME DISTRIBUTIONS.

Build first system quickly then iterate

Thursday, 09 August, 2018 14:53

TLDR: Majority of developers overthink the problem when starting work in a new domain.

SPEECH RECOGNITION WORKING EXAMPLE

There are many directions we could prioritize:

- Noisy background
  - Café noise
  - Car noise
- Accented speech
- Far from microphone (for field speech recognition)
- Young children's speech (different pronounciation, vocabulary and word order)
- Stuttering & nonsensical phrases (i.e. um, uh, ah)
- ...

Screen clipping taken: 2018-08-09 14:56

In general for any ML project there are many dozen directions that would improve system performance. So how do we pick which direction to pursue.

- BUILD SYSTEM QUICKLY & THEN ITERATE

1) Set up dev/test sets and evaluation metrics
   (set-up a target somewhere)

target placement

2) Build initial system (quick & dirty)
   - find training set
   - define architecture
   - train it.

and try to understand how we are performing

3) Use :

- Bias & Variance analysis
- Error analysis

to prioritize next improvement steps to pursue.

i.e. if error analysis indicates for field issues we'll focus on techniques to improve that.

- ADVANTAGES:
  - We can quickly look at Bias & Variance to prioritize what to do next
  - Look at Error Analysis to determine which direction is the most promising.

- APPLIES LESS TO:
  - Working in a domain where we have significant prior experience
  - There's prior art that can be reviewed for this domain

(i.e. face recognition → large body of research)

Training & Testing on different distributions

Thursday, 09 August, 2018 15:10

Deep Learning has a huge hunger for data. This causes developers to scour the Digital for any data they can find that would potentially fit in the TRAINING SET.

→ even when this train data doesn't come from the same distribution as the DEV/TEST SET

So more & more TRAIN SETS don't use the same distribution as DEV/TEST SETS.

CAT APP EXAMPLE:

PROFESSIONAL DATA

well lit & framed, sharp focus, hi res.

Data from webpages

≈ 200,000

AMATEUR DATA

blurrier, not well lit, not well framed, low res

Data from mobile app

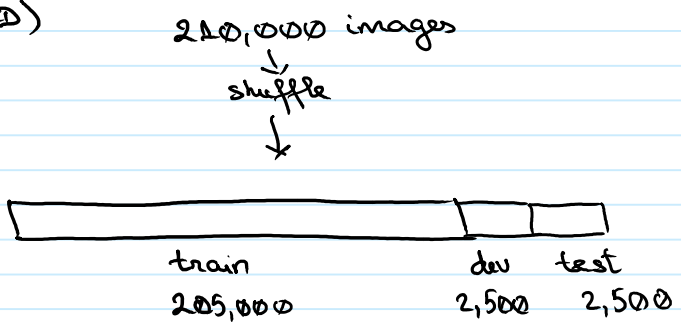
≈ 10,000

We care about our user pictures but we don't have that many

Screen clipping taken: 2018-08-09 15:16

OPTION 1: Combine everything and shuffle:

(NOT RECOMMENDED)



- Advantage: All train, dev, test sets use the same distribution

- Disadvantage: ★ HUGE — Most of the 2,500 images in the dev set are from the webpage crawler instead of from the mobile app (which is in the end what we care about)

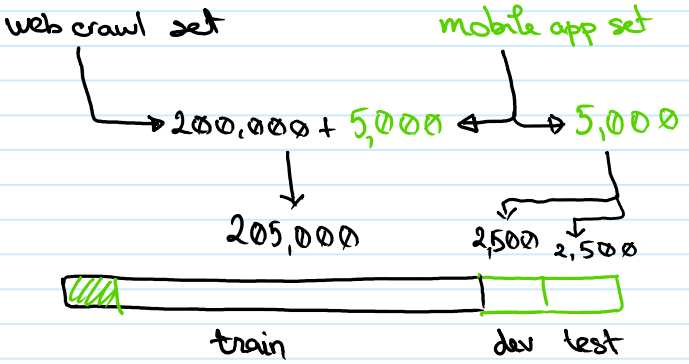
$\frac{10,000}{210,000} \times 2500 = 119$ from mobile app (on average)

$2,500 - 119 = 2,381$ from web crawling

- Setting the Dev Set is determining where we are setting the target

! → we are setting the target to optimize for web crawling obtained data (not useful) instead of the end user mobile app data

OPTION 2: (RECOMMENDED)



- ADVANTAGE: In this case both the dev & test set use the mobile app distribution

→ set the ML to perform well at aiming at the mobile data set target.

- DISADVANTAGE: Different data distributions used for

(Not as detrimental) dev/test sets ≠ train set.

SPEECH ACTIVATED REAR VIEW MIRROR EXAMPLE

has speech recognition

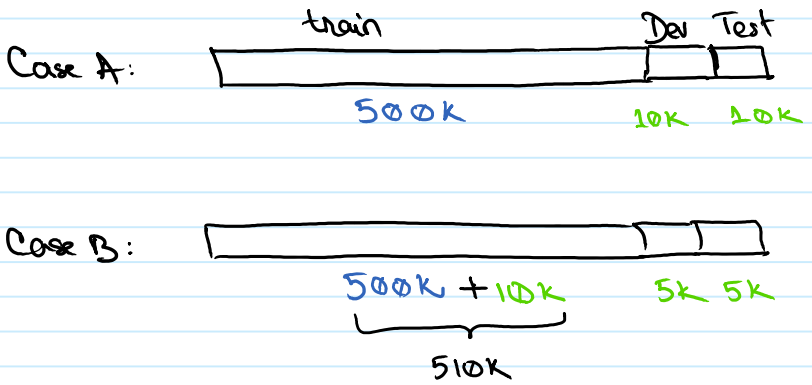
"Rear view mirror, find nearest gas station"

Screen clipping taken: 2018-08-09 15:42

Say we have a lot of speech recognition data from previous work but none from the rear view mirror microphone.

Example of splitting the data:

TRAINING SET	DEV / TEST SET
• Purchased set (can buy datasets where x: audio data, y: transcribed data)	• Speech activated rear view mirror microphone. → much narrower data set (users only ask for road related topics: gas stations, traffic, etc) → much smaller data set
• Smart speaker control	
• Voice activated keyboards	(THIS IS THE DATA WE CARE ABOUT)
• etc.	
Total: 500,000 utterances	Total: 20,000 utterances.



This results in a much larger training set than just the 20k utterances from rear view mirror microphone.

QUESTION: Is it necessary to use all data available?

ANSWER: it depends.

Bias & Variance of mismatched distributions

Thursday, 09 August, 2018 15:55

The way we analyse bias & variance changes when the train data and dev/test data come from different distributions.

CAT CLASSIFIER EXAMPLE

Train set



Dev/Test set



Assume Human-error ≈ Bayes error ≈ 0% error.

Error analysis involves looking at

Training error ..... 1%.

↓ 9%.

Dev error ..... 10%.

Confounding problem when moving from Train set to Dev Set Forward pass

1) Algorithm analyzes new data that it didn't see yet in Dev Set
2) Distribution of data is different b/w Train & Dev Set

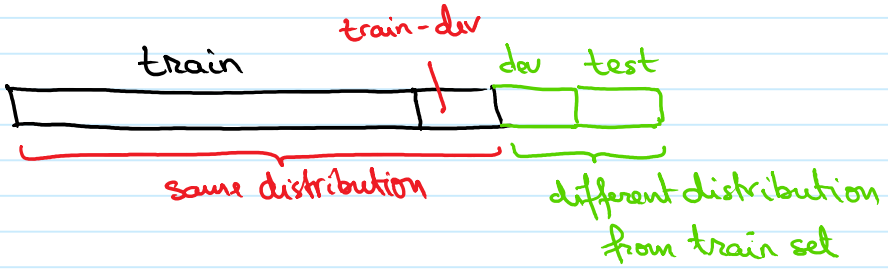
- Normally when we use same train dev/test distribution we say we have a large variance problem (we are not generalizing well).
- However when we draw from different distributions for train set vs dev/test set we can believe we might be doing ok on the dev set and it's just that the train set was easy (images were crisp, well light well framed) while the dev set is a more difficult set (blurry images etc)

→ We don't know how much of the difference in training vs dev error is due to a new data set or a different distribution of data

- To decouple the new vs different distribution of data we create a new data set:

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

Screen clipping taken: 2018-08-09 17:33



Train error:	1%	1%
Train-dev error:	9%	1.5%
Dev set error:	10%	10%
	variance problem	data distribution problem

This tells us we have a variance problem as we controlled for the data distribution

This tells us our variance is OK but the error jumps when we go to dev set so we have a data mismatch problem.

Human (Bayes):	0%.	
Train error:	10%.	10%.
Train-dev error:	11%.	11%.
Dev set error:	12%.	20%.
	This scenario indicates we have a high avoidable bias problem as our train error is not in the ballpark of Bayes error.	Now we have 2 problems 1) high avoidable bias 2) data mismatch problem but variance is OK.

SUMMARY

Key quantities to review when train set distribution doesn't match the dev/test set distribution are.

- Human level error — 4% — avoidable bias
- Training set error — 7% — variance
- Train-dev set error — 10% — data mismatch problem
- Dev set error. — 12% — degree of overfitting to dev set. (maybe increase size of dev set)

OPTIONAL • Test-set error — 12%

- In practice the errors don't always improve as we go up the hierarchy of errors. For example
  - Human level error 4%
  - Training set error 7%
  - Train-dev set error 10%

} Train set distribution

- Training set error 1% } Train set distribution
- Train-dev set error 10% }
- Dev set error 6% } Dev/test set distribution
- Test-set error 6% }

In this case the training set data is much harder than dev/test set data.

GENERALIZED ANALYSIS

Let's use the Rear View Mirror example

Error \ Data sets	General Speech Recognition (previous owned data)	RearView mirror speech Data (new data)
Human-level	"Human level": 4%	6% (ask humans to review recordings of rearview mirror data)
Error on examples NN has trained on	Training error: 7%	6% (put some rear view mirror data in train set and check error only on this subset of data)
Error on examples NN has not trained on	Training-dev error: 10%	Dev error: 6% or Test error: 6%

avoidable bias

variance

this indicates rearview mirror data is harder for humans

data set mismatch

normal path to analyze train vs dev/test set distribution differences.

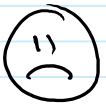
So now our error analysis can indicate 3 causes:

- avoidable bias
- variance

- data set mismatch

How do we address data set mismatch problems?

→ there aren't many good ways to go about it.



Addressing data mismatch

Thursday, 09 August, 2018 18:26

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

Screen clipping taken: 2018-08-09 18:30

To avoid overfitting the test set technically for error analysis we should only look at the dev set (and not at the test set)

EXAMPLE REARVIEW MIRROR SPEECH RECOGNITION

- Review sound data to get insight into the differences.
  - notice that in the dev set the background is more noisy as there are many car related sounds overlapping the voice
  - in the dev set there are many more errors related to street numbers (as there are more queries related to street addresses).

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

Screen clipping taken: 2018-08-09 18:38

Eg. • If we find car noise background is a major source of errors we can simulate noisy in car data. (artificial data synthesis)

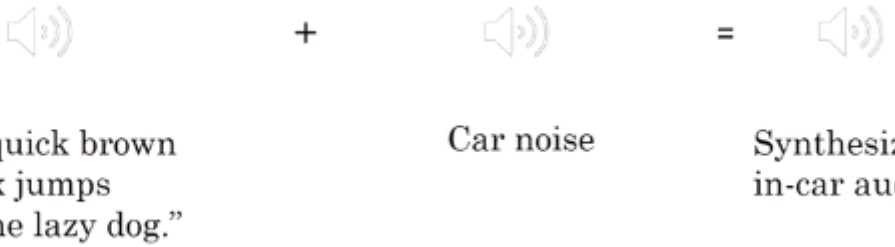
• if we find numbers are over represented in the dev-set add more

examples of people saying numbers in the train set.

This is not a systematic process it is just a guideline.

ARTIFICIAL DATA SYNTHESIS:

- REAR VIEW MIRROR SPEECH RECOGNITION EXAMPLE



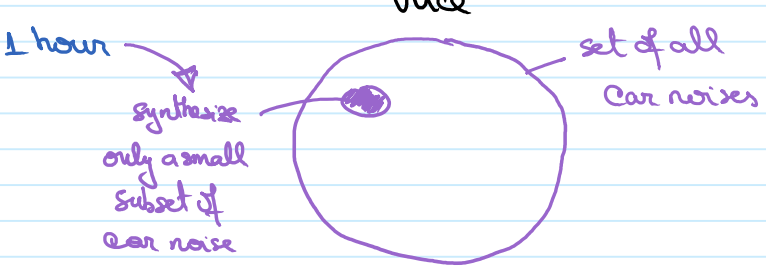
Screen clipping taken: 2018-08-09 18:43

We can create data that is similar to actual data by combining subsets of data.

- CAVEATS:

Human Voice + Car noise = Synthesized human in car voice

10,000 hours



We can repeat the 1 hour car noise 10,000 times to match the

human voice data set.

→ to the human ear this sounds OK.

→ algorith is highly likely to overfit to the car noise.

The challenge is that the human senses cannot tell the differences between different types of car noise.

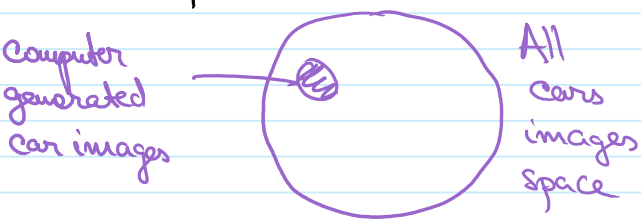
- CAR RECOGNITION EXAMPLE:



Screen clipping taken: 2018-08-09 18:59

Why can't we use computer generated images of the cars to create our train data set?

Again to the human eye this looks fine but we might only capture a small subset of the "all cars" data space



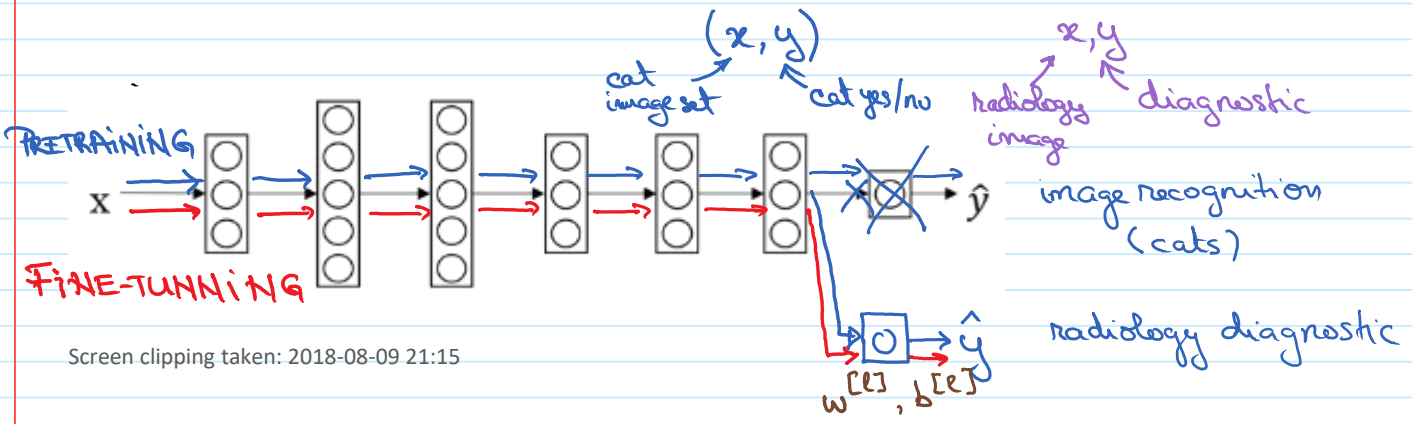
In particular use video games to acquire data
but games are limited to the # of cars available in the game which is a small subset of all cars in reality.

Transfer learning

Thursday, 09 August, 2018 21:12

Apply knowledge learned for one task and apply it (or part of it) to a different task.

For a learning task where there are only a small # of examples available find a related task that has a large set of examples available train a NN on this related task and then use the NN to learn the initial task.



Screen clipping taken: 2018-08-09 21:15

delete the last output layer and its associated weights and then create a new set of randomly initialized weights for the last layer and have that output radiology results.

So:

1) Train the network on Data Set 1 (cats) (x, y)
2) Swap in a new Data Set 2 (radiology images) (x, y) and initialize the last layer weights randomly and retrain the network.

There are a couple of options on how to retrain the NN w/ radiology data

A: For small radiology data sets retrain only the weights of

the last layer $(w^{[l]}, b^{[l]})$

B: For large radiology data set retrain all the weights of the NN

HEURISTIC: The data set determines how many layers from the end get trained.

Small: last & maybe 2nd to last layer

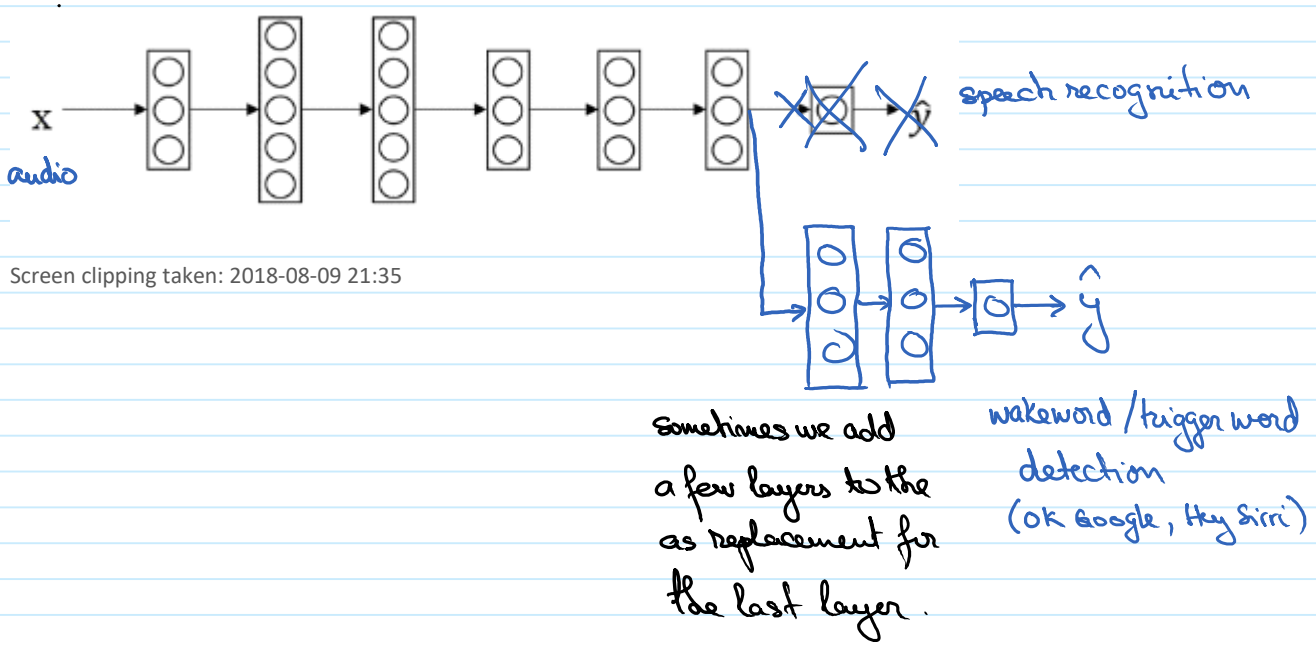
Large: all layers.

The initial phase of training the NN (using cat images) is called:
PRE-TRAINING.

The 2nd phase of training (using the radiology data) is called
FINE-TUNNING.

This works b/c the detection of low level features is similar for all images.

- SPEECH RECOGNITION SYSTEM EXAMPLE:



Screen clipping taken: 2018-08-09 21:35

WHEN TO USE TRANSFER LEARNING?

- When we have a lot of data for the problem we are transfering from and relatively little data for the problem we are transfering to.

	IMAGE RECOGNITION	AUDIO DETECTION
SOURCE PROBLEM:	cat detection 1,000,000 examples used to learn many low level features	speech recognition 10,000 hours
DESTINATION PROBLEM:	radiology diagnostic 100 examples (x-rays + diagnostic)	wakeword detection 1 hour.

- It doesn't make sense to use transfer learning when the opposite is true:

SRC: Cat detection 100 images

DST: radiology diagnostic 100 images

↳ each image in the DST set is much more valuable than the images in the SRC set for training a good radiology diag system.

SUMMARY

It is useful to use transfer learning when

Task A → Task B

- Task A has same inputs as Task B (same type of x)
- You have a lot more data for Task A than Task B

- We suspect low level features from Task A apply to low level features for Task B.

Multitask learning

Thursday, 09 August, 2018 22:04

Trying to learn from multiple tasks at the same time.

(as compared to transfer learning which is a sequential process)

We have one NN trying to learn several things at the same time.

SIMPLIFIED AUTONOMOUS DRIVING EXAMPLE

$x^{(i)}$

Screen clipping taken: 2018-08-09 23:03

Needs to detect:

$y^{(i)}$ → $(4,1)$

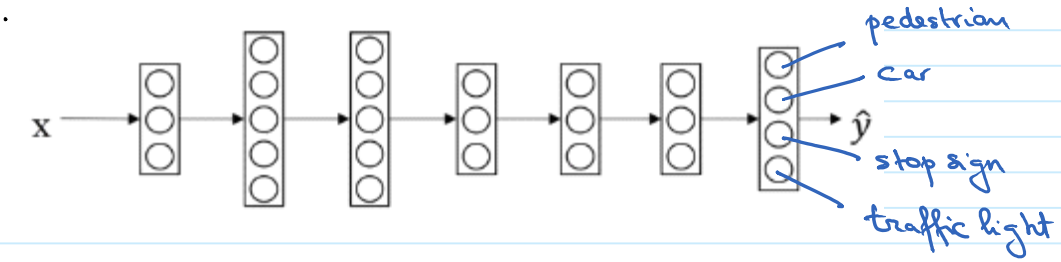
Pedestrians	0
Cars	1
Stop signs	1
Traffic lights	0
⋮	

for the image to the left

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

$(4, m)$

(normally Y would be a $(1, m)$ → when detecting a single feature)



Screen clipping taken: 2018-08-09 23:07

Loss for this 4 dimensional output $\hat{y}^{(i)}$:

Loss for this 4 dimensional output $\hat{y}^{(i)}$:

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

→ standard logistic loss

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

this is slightly different as we are summing over the four output dimensions

NOTE: Unlike Softmax - which assign a single label to an image (image is car xor image is stop sign).
Multitask assigns multiple labels to an image.

By minimizing this loss function we are performing Multitask Learning
→ we are looking at each image and are solving 4 different problems.
→ we could have trained 4 different NN instead
→ but b/c the early features for this type of classification are common to all the 4 NN this is more efficient. (better performance)

It turns out Multitask Learning works even if not all images have all the features labeled (i.e. missing the stop sign)

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

partially labeled — fully — checking only for presence of cars.

labeled labeled for presence of cars.

We will smarten the loss function so it only sums for dimensions that are labeled

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

Sum only over values of j with 0 or 1 labels. (omit terms with "?")

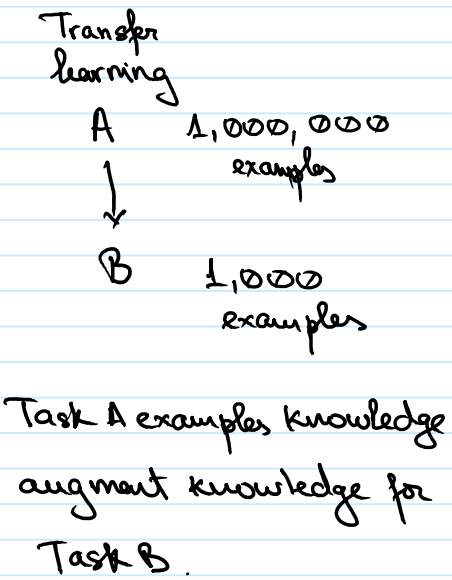
WHEN TO USE MULTITASK LEARNING?

- Training on a set of tasks that could benefit from having shared lower-level features.

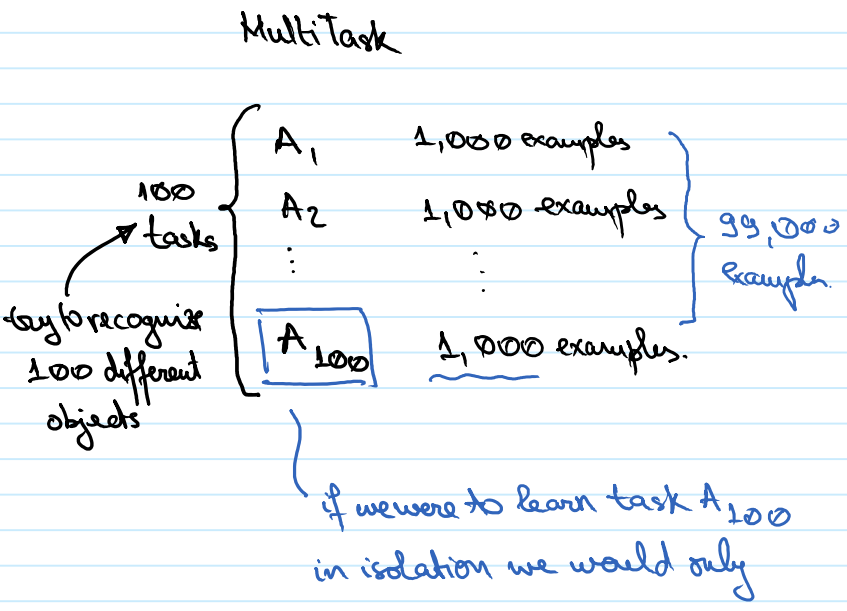
Screen clipping taken: 2018-08-09 23:24

- Usually: Amount of data you have for each task is quite similar.

Screen clipping taken: 2018-08-09 23:25



Task A examples knowledge augment knowledge for Task B.



if we were to learn task A_{100} in isolation we would only

have 1,000 examples to learn from

↓

by training on the other 99 tasks we use an additional 99,000 examples to augment the example for task A 1000.

This works only when the sum of all other tasks examples is much larger than the number of examples for a single task

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

Screen clipping taken: 2018-08-09 23:34

If using on small NN Multitask Learning hurts performance (researcher Rich Caruana showed this a while ago)

The alternative is to train a NN for each task.

Multitask learning is less often used than transfer learning.

Exception is in Computer Vision (object detection)

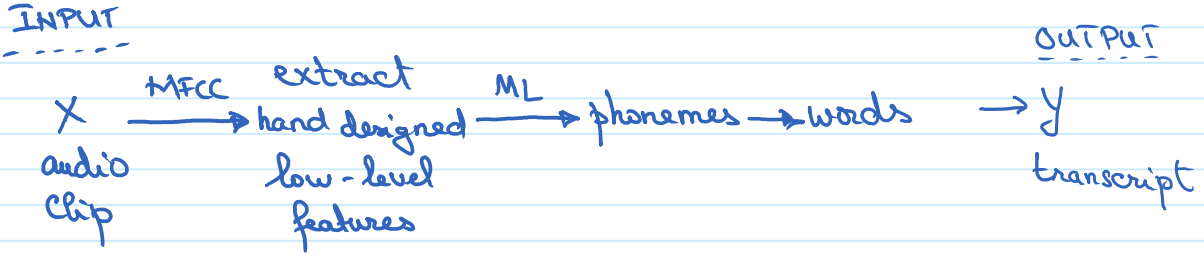
What is end-2-end Deep Learning?

Friday, 10 August, 2018 13:51

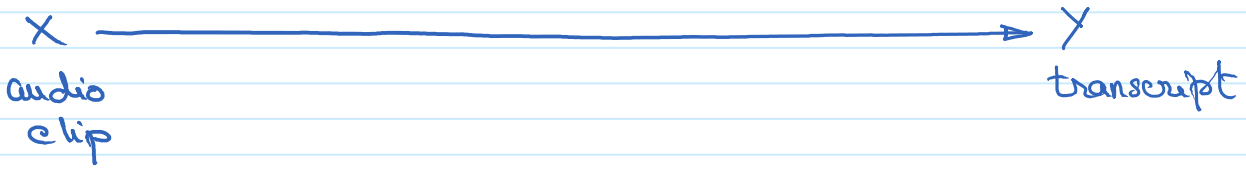
For a learning process that requires multiple stages of processing E2EDL takes all those stages and combines them into a single NN

SPEECH RECOGNITION EXAMPLE:

- TRADITIONAL ARCHITECTURE



- E2E DL ARCHITECTURE



Note: Many researchers had spent most of their careers building sub stages for the traditional architecture. When E2EDL arrived on the scene there was a lot of push-back and skepticism from those researchers on how well it will perform
→ this obsoleted many years of research.

- E2EDL requires a lot of data to work well
→ if we only have 3,000 hours of audio the TRADITIONAL

pipeline works very well

→ if we have 10,000h to 100,000h of audio the E2EDL works better.

→ for intermediate amounts of data a hybrid aproach works better

audio ⟶ phonemes ⟶ . . . ⟶ transcript

FACE RECOGNITION EXAMPLE:

[Image courtesy of Baidu]

Screen clipping taken: 2018-08-10 14:08

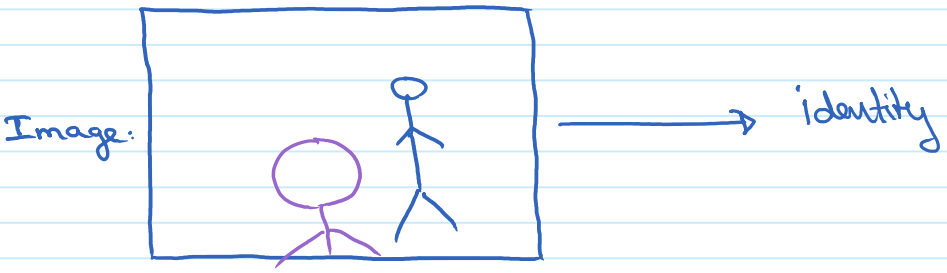
At turnstile entry points in a building use a camera to identify personnel and allow them to enter.
This can replace RFID badges.

HOW DO WE BUILD A SYSTEM LIKE THIS?

- Aproach A: Pure E2EDL

  Analyze the raw image from the camera and determine identity

  X Y

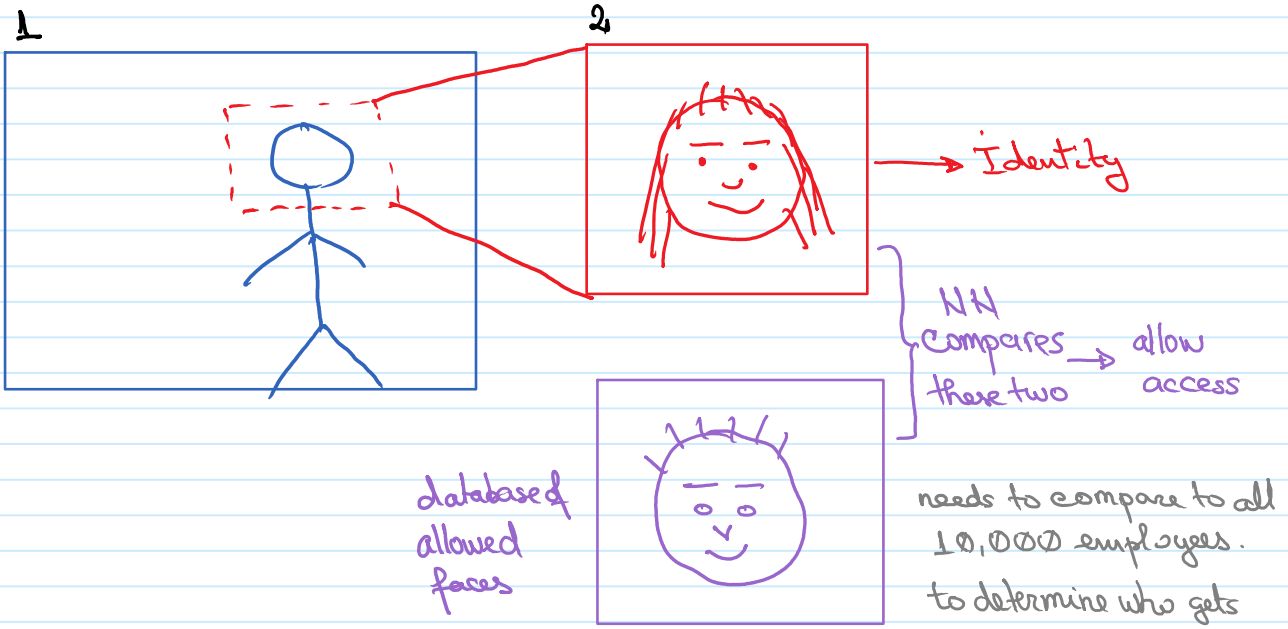


This is not ideal as the person can approach the camera in many different ways

→ head looking away, really close to the camera (big head) far from the camera, etc.

- Approach B: 2-step.

1) Analyze the image to identify the location of the face and then crop the face.
2) Feed the face into the NN to estimate the person identity



WHY DOES APPROACH B WORK BETTER?

1) Each of the two problems we are solving is much simpler.

2) We have a lot of data for each of the two subtasks

→ we have a lot of labeled data for finding faces in images

→ we have 100Ds M of people faces with associated identities.

In contrast we have very little labeled data of general image with person's identity associated with it.

So in practice a pure E2EDL approach doesn't work as well as a 2-step approach like above.

→ if we had more data maybe E2EDL would work better.

MACHINE TRANSLATION EXAMPLE:

- Traditional approach:

English → text analysis → ... → French

As we have access to large sets of labeled data (x, y)

x: English, y: French.

- E2EDL approach:

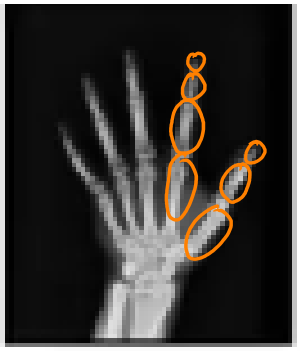
English ⟶ French

ESTIMATE CHILD AGE FROM X-RAY:

This is used to determine if a child is growing normally (less often used for crime investigation)

Traditional approach:

Image → ① segment out bones → ② bones length → lookup → age



Screen clipping taken: 2018-08-10 14:40

Image (1) → segment out bones → bones length (2) → lookup → age

E2EDL:

Image ——————————→ age

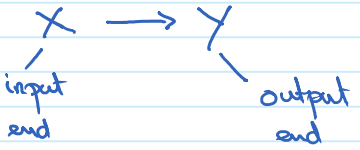
Until now the E2EDL approach works worse as there is not enough data to train.

SUMMARY:

When E2EDL works it simplifies the architecture of the project dramatically but E2EDL does not work all the time.

When to use End-2-end Deep Learning?

Friday, 10 August, 2018 14:46

PROS	CONS
• "Let the data speak" → the NN will figure out the statistics of the data rather than use human preconceived notions i.e. language processing: humans use concept of phonemes "c a t" it is not obvious why a NN should be forced to use phonemes. • Less hand-design of substages and special features is needed	• Requires large amounts of data  • Excludes potentially useful hand-designed components → even though hand-designed components are looked down upon by ML practioners when we have small amounts of data hand-design components are a way to manually inject knowledge into the system.

A learning algorithm has two sources of knowledge:

1) Data

2) Manually designed components / features

When we don't have much data we can compensate by using hand-design components.

→ though they might hold back performance.

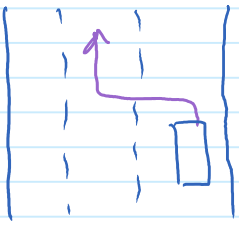
KEY QUESTION:

Do we have sufficient data to "learn" a function of the complexity needed to map x to y?

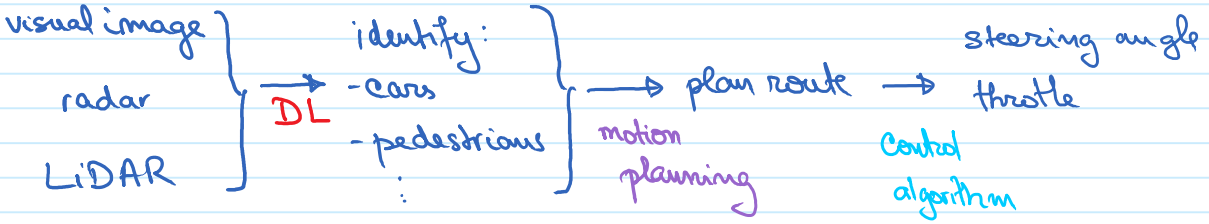
→ we don't know exactly what "complexity needed" means.

Low complexity	high complexity
Find bones in an X-ray image	Determine age from an X-ray image
Find faces in an image	

CAR DRIVING EXAMPLE:



- NON-E2EDL APPROACH:



In this case ML is only applied for a subset of the stages of the process.

→ this depends on which components we can get data for.

It is exciting to think we can use E2EDL for the entire process. but this is not currently a feasible approach for the current state of Deep Learning