Structuring ML Projects

Why ML Strategy? → save time (and not lose months...)

→ Cat classifier → 90% Accuracy → want to be better!

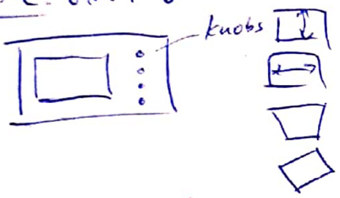
↳ Ideas how to improve

- more data
- more diverse training set
- train longer with GD
- try Adam instead of GD
- try bigger network
- try smaller network
- try dropout
- try L2 regularization
- NN architecture
  - activation functions
  - # hidden units

} how can we measure our improvement?

Orthogonalization → "which HP to tune to get which effect?"

example: old TV

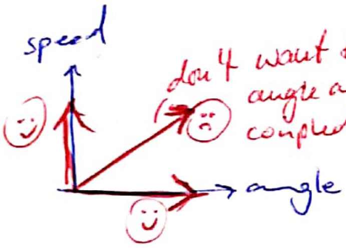


90°

or car
steering
accelerating
braking

} we want each "knob" or steering feature to have an isolated effect (on the TV's picture or on the car's steering angle (or decoupled effect)

↳ Orthogonalization!



don't want to have angle and speed coupled in a car!

Chain of assumptions in ML (for a supervised system) (Orthogonalization process)

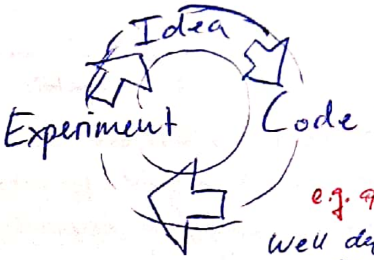
1) Fit training set well on cost function (≈ human-level performance) (want one knob to adjust) → tune algorithm well to fit on training set → bigger NN, better optimizer...
↓
2) Fit dev set well on cost function → set of knobs → Regularization → Bigger training set
↓
3) Fit test set well on cost function → Bigger dev set (overtuned the former dev set)
↓
4) Performs well in real world → change either dev set or cost function

→ not use early stopping → affecting multiple things at the same time

Using a single number evaluation metric

Precision → what % was classified correctly? (of examples)

Recall → what % is actually true, will be correctly recognized by our classifier



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

→ often a trade-off

e.g. after HP change

F1 Score
92.4%
91.0%

~ average of P. and R.

$F1 = \left(\frac{2}{\frac{1}{P}+\frac{1}{R}}\right)$ "Harmonic mean"

Well defined Dev Set + Single real number eval metric

↳ speeds up iterative process of ML!

normal average is also quick to evaluate

Satisficing and Optimizing metric → When a single value evaluation metric is not possible

ex1

optimizing metric

satisficing metric

Classifier	Acc.	Run time
A	90%	80ms
B	92%	95ms
C	95%	1500ms

N metrics: 1 optimizing
N−1 satisficing

- maximize accuracy
- subject to run time ≤ 100 ms

satisficing has only to do sufficiently

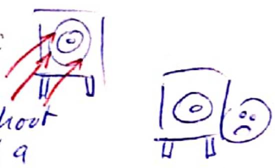
ex2: wake/trigger words
→ Alexa, OK Google, "Hey Siri"...
Interested in accuracy
+ # false positives (wake up without command)

maximize accuracy — optimizing metric
subject to ≤ 1 false positive / 24 hours — satisficing metric

Train/dev/test distributions

classification dev/test sets
hold out cross validation set

Analogy:



learn to shoot arrows at a target, → then ask model to "shoot" at a different target with same accuracy

ex1:
Regions
- US
- UK
- oth. Europe
- South America
- India
- China
- oth. Asia
- Australia

Dev: US, UK, oth. Europe, South America
Test: India, China, oth. Asia, Australia

bad because they are from different distributions!! Teams lose time by doing well on dev set only to realize that test set will behave completely different

Solution → randomly shuffle into dev/test sets!
↳ same distribution, mixed together

ex2:
loan approval: dev set was optimized to medium income ZIP codes
↳ was then tested on low income ZIP codes

↳ Guideline

same distribution

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

Size of the dev and test sets

70% train	30% test

60% train	20% dev	20% test

m 100 – 10.000

Tr 98% DT 1% 1% # m ~1.000.000

Size of test set
→ big enough to give high confidence in overall performance

Sometimes OK to have only train+dev set and no test set

When to change dev/test sets and metrics

ex: Metric: classification error
Algorithm A: 3% error → let through pornographic images
" B: 5% error → lets through no porno images
→ B is preferred although has higher error

adjust metric

$$\text{Error} = \frac{1}{\sum_i w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{y^{(i)}_{pred} \neq y^{(i)}\}$$

"weight" $w^{(i)}$ $\begin{cases} 1 & \text{if } x^{(i)} \text{ is no-porn} \\ 10\text{–}100 & \text{if } x^{(i)} \text{ is porn} \end{cases}$

Orthogonalization

→ So far only how to define a metric to evaluate classifier

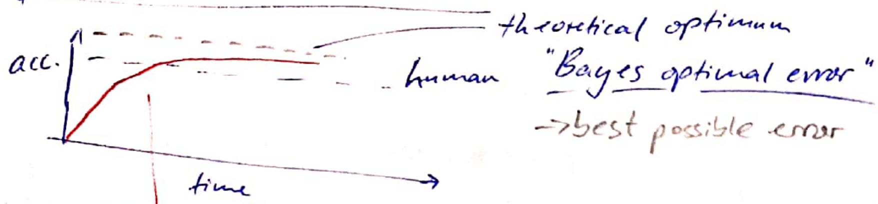
→ Worry separately about how to do well on this metric

If doing well on metric + dev/test set doesn't correspond to doing well on your application, change your metric and/or dev/test set

sharp images for training/dev/test but user images are blurry (cat classifier)

DL spec // course III // week 1

Why human-level performance?



progress fast until surpass human-level performance

Why compare to human-level performance

- Humans are quite good at a lot of tasks. As long as ML is worse we can:
  - → get labeled data from humans
  - → gain insight from manual error analysis: why did a person get this right?
  - → Better analysis of bias/variance

Avoidable Bias

Cat classifier

Humans 1%
train error 8%
dev error 10%
} try ~~reducing~~ reducing bias

focus on bias

different case

human 7.5% (maybe pics blurry)
8% — 0.5% avoidable bias!
10% ↕ 2% variance

variance reduction techniques such as regularization or getting more data

focus on variance here

Human level error as a proxy (estimate) for Bayes error

$\boxed{\text{diff: Bayes} - \text{train-error} = \text{"Avoidable bias"}}$

Understanding human-level performance

HLE as proxy for Bayes error

Medical image classification example:

	error
normal person	3%
" doctor	1%
experienced doctor	0.7%
team of experienc. doctors	0.5%

What is → "human-level" error?

Bayes error ≤ 0.5%

avoidable bias / variance

Human (proxy for Bayes) 1% / 0.7% / 0.5%
training error 5% } 4-4.5% → higher, so focus on bias reduction techniques → bigger NN
Dev error 6% } 1%

H.e. 1% / 0.7% / 0.5% } 0.1%
tr.er 1%
Dev 5% } 4% → use variance reduction techniques! (regularization or bigger training examples)

important case

here it matters which human error we take

Human error 0.5%
tr. err. 0.7% } 0.2% → twice as big! ↳ so we know that we actually can do better!
dev. err. 0.8% } 0.1%

→ works until surpassing human-level performance

Surpassing human-level performance

Team humans	0.5%	avoidable bias 0.2%	0.5%
one human	1%		1%
training error	0.6%	variance 0.2%	0.3% ← does this mean we overfitted the model or are we actually above/better than human error?
dev error	0.8%		0.4%

examples where ML significantly > human level performance

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

} learned from structured data (not natural perception) (Lots of data)

humans are very good at natural perception:
- speech recog.
- image "
- medical
- ...

Improving your model performance

2 fundamental assumptions of supervised learning

1. You can fit training set well → low avoidable bias

2. training set performance generalizes pretty well to dev/test set → Variance not too bad

Human-level
↕ avoidable bias → { bigger model; train longer / better optimization – momentum, RMSprop, Adam; NN architecture / HP search, try others like RNN / CNN
training-error
↕ variance
dev-error → { more data; Regularization → L2, dropout, data augmentation; NN architecture / HP search

---

training set 10'000'000 images $y = \begin{cases} 0 & \text{no bird} \\ 1 & \text{bird} \end{cases}$
→ what is evaluation metric?
→ how structure data?

Error Analysis

Look at dev examples to evaluate ideas

↳ get ~ 100 mislabeled dev set examples
↳ count how many are wrong (like dogs for a cat classifier)
↳ get "ceiling" → from 100: is it 5% or 50% ?

Evaluate multiple ideas in parallel

Ideas for cat detection:
- Fix pics of dogs being recognized as cats
- Fix great cats (lions, panthers..) being misrecognized
- Improve performance on blurry images

Image	Dog	Big Cat	blurry	Comment	Instagram
1	✓			Pitbull	✓
2			✓	lion	✓
3		✓	✓	rainy day at zoo	
⋮	⋮	⋮	⋮		⋮
% of total	8%	43%	61%		12%

Cleaning up incorrectly labeled data

→ What to do? → Training set

"DL algorithms are robust to random errors in the training set", less robust to systematic errors

↳ go to table

Image	Dog	great cat	blurry	Incorrect labeled	Comment
				✓	...
	8%	43%	61%	6%	

1) overall dev set error .... 10% error
2) Errors due incorrect labels .... 0.6% error
3) Errors due to other causes ...... 9.4% error

case 2: 2% 0.6%

→ Goal of dev set is to help you select between two classifiers A & B

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

sometimes difficult to do → 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 first system quickly, then iterate (not overthink at the beginning)

Speech recognition example
- → Noisy background
  - Café noise
  - Car noise
- → Accent
- → Far from microphone
- → Young children's speech
- → Stuttering
- ⋮

↳ 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

Cat app example

(good images) Data from webpages ≈ 200'000
↳ :(is not the distribution we want

(blurry images) Data from mobile app ≈ 10'000 → we want this to perform well

210'000
↓ shuffle randomly

Option 1: | train | dev | test | → now all from same distribution
205'000 | 2500 | 2500

:(disadvantage: A lot of dev set examples will come from the webpage distribution which we do not care about
2381 – web :(
119 – app :)

Option 2: train: | train → web 200'000 | 5000 | dev | test |
205'000 | 5000 from app | 2500 | 2500 from app

training-dev set

→ train distribution is now different from dev/test distribution

example 2: Speech recognition
speech activated rear view mirror

training (audio clip x, transcript y)
Purchased data
Smart speaker
Voice keyboard
⋮ 500'000 utterances

Dev/test
Speech activated rearview mirror
→ 20'000 → dev test

train 500k | 10'000 | 10'000
or | 510'000 | 5k | 5k |

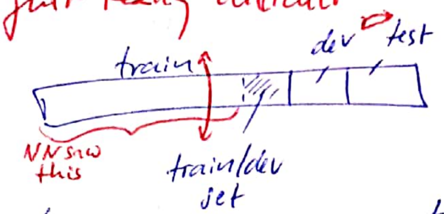
Bias and variance with mismatched data distributions

Cat classifier   Assume human ≈ 0% error

Training error 1%
Dev error 10%
9%

→ normally we'd assume just high variance but maybe the set is just really difficult

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



↳ train training set now on train/dev-set

→ train. error 1%
train-dev error 9%
dev error 10%

} then we know we have Variance problem

		human err. 0%	hum. er. 0%
train error	1%	10% (avoidable bias)	10% (avoidable bias!)
train-dev error	1.5% (data mismatch)	11%	11% (small variance)
dev error	10%	12%	20% } data mismatch!

→ Key quantities

Human error	4%	avoidable bias	4%	
training set error	7%	variance	7%	} evaluated on training set distribution
train/dev-set-error	10%	data mismatch	10%	
dev error	12%	degree of overfitting to dev set	6%	} evaluated on dev/test set distribution
test error	12%		6%	

If dev set is overfitted get bigger dev-set!

More general formulation (speech recogn.)

	General speech recognition	Rearview mirror speech data
Human level performance	"Human-level" 4%	6%
Err. examples that NN has trained on	"train error" 7%	6%
Err. example that NN has not trained on	"train-dev set error" 10%	"Dev/test error" 6%

avoidable bias
variance
data mismatch

Addressing data mismatch

If training set is from other distribution than dev and test set and error analysis shows data mismatch problem:

→ Carry out manual error analysis to try to understand difference between training and dev/test set
  → speech recognition with noise

→ Make training data more similar, or collect more data similar to dev/test sets

e.g. simulate noisy in-car data

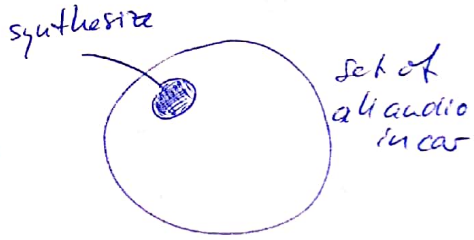
Artificial data synthesis

"The quick brown fox jumps over the lazy dog" + Car noise = synthesized in-car audio

→ has every letter in it

10.000 hrs

1 hr
→ risk to overfit to car noise



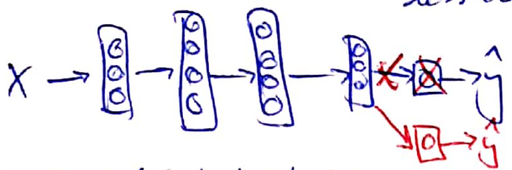
other example

car recognition + put a bounding box around it

Learning from multiple tasks

Transfer learning

→ makes sense when enough data where training is done and less data where we are transferring to

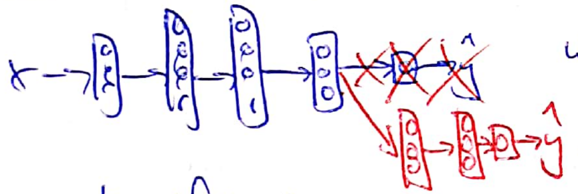


(image recognition) 100'000'000 images
↓
(medical (...)) radiology images → 100 images

→ to do: ∘ delete last layer + the weights feeding into it
∘ create a new set of randomly initialized weights
∘ swap data set, now with radiology images
∘ re-train NN (only new W, b or all layers if there are enough data)

„pre-training" NN has learned how images look like and can
„fine-tuning" now be applied to a different data set

other example: speech recognition



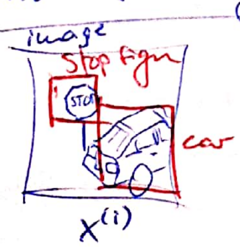
wake-word detection 10'000 h
↓
→ 1h

when transfer learning makes sense A → B

→ Task A and B have the same input x (images or audio)
→ You have a lot more data for Task A than for task B
→ Low level features from A could be helpful for learning B

Multi-task learning

autonomous driving example



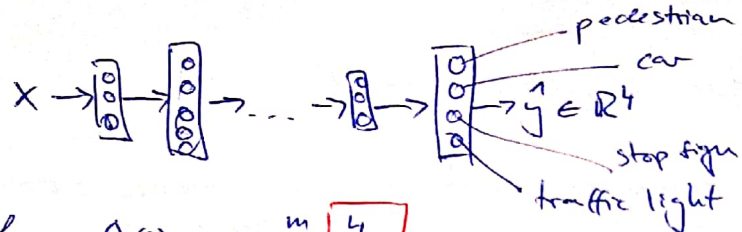
$$y^{(i)} \quad (4,1)$$

Pedestrians	0
cars	1
stop signs	1
traffic lights	0

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

(4, m)

↓



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

sum only over j with label for 0 or 1

↳ usual logistic loss

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

Unlike softmax regression:

One image can have multiple labels! → multi-task learning

(could also train 4 NNs, but one NN will have better performance)

also works for $Y = \begin{bmatrix} 1 & 1 & \cdots & 0 & \cdots & ? \\ 0 & 1 & & 1 & & ? \\ ? & ? & & 1 & & 1 \\ ? & ? & & 0 & & ? \end{bmatrix}$ missing labels...

When multi-task learning makes sense

→ Training on a set of tasks that could benefit from having shared low-level features

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

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

End-to-end learning

Speech recognition

pipeline

"cat"

better for smaller dataset

conventional: $X \xrightarrow{MFCC}$ features $\xrightarrow{ML}$ phonemes → words → y ~3'000h OK

audio transcript

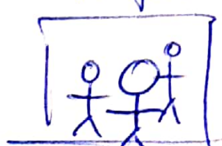
end-to-end DL: audio $\xrightarrow{NN}$ transcript ~10'000 ... 100'000h OK

better for larger dataset

Face recognition:

(to let someone in a facility → security gate)

Image (x)



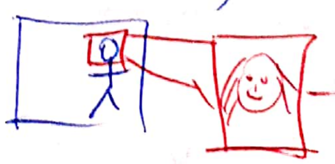
→ y identity

problem: approach gate from different angles + distance

2 steps performs better

instead: 1) face

2) identity



→ identity

why works better?

→ have lot of data

(is compared to employee database)

more examples

Machine translation

English → text analysis → ... → French

English ————————→ French ✓ (many data)

Estimating child's age:

bone image

Image ① → bones ② → age more promising

Image ————→ age ☹

Pros and cons of end-to-end DL

Pros: → Let the data speak $x \to y$ "phonemes" c a t

→ Less hand-designing of components needed

Cons: → May need large amount of data

→ Excludes potentially useful hand-designed components

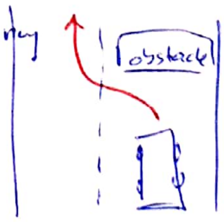
Apply end-to-end DL

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

example

image, radar, lidar $\xrightarrow{DL}$ cars → route → steering control

$\xrightarrow{DL}$ pedestrians (motion planning)



→ Use DL to learn components

→ Carefully chose $X \to Y$ depending what tasks you can get data for

image ————————→ steering ☹