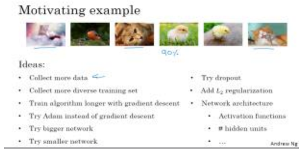
3-1_Machine Learning

3.1-1 Machine learning

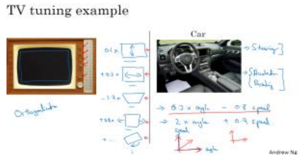
- machine learning strategy
  way of analizing problem, ->point in the direction of the most promising things to try.



3.1-2 Orthogonalization

--**Purpose**: too many hyperparamters to tune, need to know which one to tune for specific problem

Orthogonalization or orthogonality is a system design property that assures that modifying an instruction or a component of an algorithm will not create or propagate side effects to other components of the system. It becomes easier to verify the algorithms independently from one another, it reduces testing and development time.



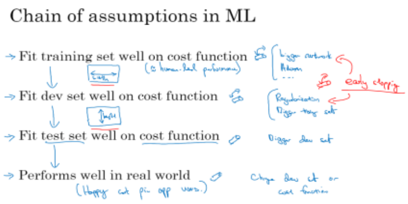
When a supervised learning system is design, these are the 4 assumptions that needs to be true and orthogonal.

1. Fit training set well in cost function
performance on the training set needs to pass some acceptability assessment. e.g: compare to human error.
- If it doesn't fit well, the use of a bigger neural network or switching to a better optimization algorithm (e.g.Adam)might help.
(note:
- bigger neural work: better architecture , to deal training set feature
- better optimization: faster optimizing time, faster to converge.)

2. Fit development set well on cost function
- If it doesn't fit well, regularization or using bigger training set might help.
(note:
- Regularization: for overfitting training set, -->small weight
- Bigger training set: more representative of dev set)

3. Fit test set well on cost function
- If it doesn't fit well, the use of a bigger development set might help, as when does well on dev set, not well on test set, probably means overtuned dev set.
(note: dev & test set are same distribution, while training set may different distrubtion.-->if fit dev set well, but not in test set-->dev set may too small , could not representting test set)

4. Performs well in real world
- If it doesn't perform well, the development dev, test set is not set correctly or the **cost function** is not evaluating the right thing.



Note:
dropout is a good techni, yet no recommended to use as difficult to think about, because it will influence cost fucntion in traning set and dev set; stop training on training set ealier, which often done to improve performance on dev set.
it doesn't mean that drop out is bad,, you can use it if you want. But when you have more orthogonalized controls, such as these other ones above , then it just makes the process of tuning your network much easier.
(note: orthogonalized controls: intro. one method then only influence one of algorithm performance: on training set, dev set, test set, real word;)

3.1-3 Single number evaluation metric

1. Using a single number evaluatino metric

Purpose: quick efficient tell the idea just tried is better or not.
To choose a classifier, a well-defined development set and an evaluation metric speed up the iteration process.

>1.Precision: of the reusult (image) identified as true (cat), what percentage actually is true (cat):
Of all the images we predicted y=1, what fraction of it have cats?

Precision (%) = *True positive number of predicted positive x 100 = True positive / (True positive + False positive) x 100*

>2. Recall : Of all the images that actually have cats, what fraction of the classfifier did correctly identifying have cats?

Recall (%) = True positive number of predicted / actually positive in training set x 100 = True positive / (True positive + False negative)--all positive in training set x 100

Note:
1. offen need to tradeoff precision an recall: e.g: many feature (strictions) added to be identified as cat-->high precision, but will cause low recall (if could not meet all features identified not cat);
2. with two evaluation metric, it is difficult to know how to quickly pick one of the multiple models.--->a new evaluation combine these two metric.

2. F1 score
consider both precision and recall: F1-score, a harmonic mean, combine both precision and recall.
F1-Score=2/ (1/p+1/r)
F1-Score is not the only evaluation metric that can be use, the average, for example, could also be an indicator of which classifier to use.

Note:
Measure presion and recall on a **well-defined dev set**, and compute a single number evaluation metric-->quickly tell if a classfier is better or not -->speed up iteration process of improving machine learning algorithm

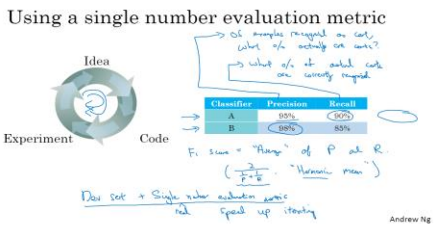
3. Another example: average perforamnce on all test set

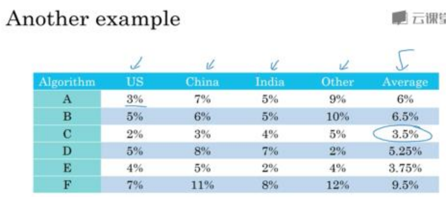
e.g algorithm A/B/C/D have diffrent perforance on different region (US, China...), compare these algorithm performance

average algorithm A/B/C/D perforamnce on all test set/region and then compare different algorithm average perforamnce.

average performance could also consider : for quick judge which classfier is better

Summary:
Machine learning is experimental method , have an idea, implement it try it out, and check this idea helpes or not by having a single number evaluation metric , this can really improve your efficiency or the efficiency of your team in making those decisions





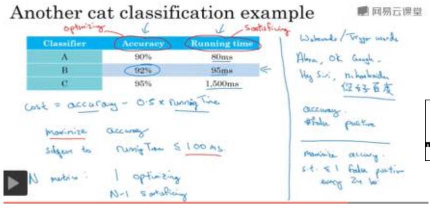
3.1-4 Satisficing and optimizing metric

Purpose: It's not always easy to combine all the things you care about into a single row number evaluation metric. In those cases it's sometimes useful to set up satisficing as well as optimizing matrix.

1. Classification example:
care about classfier accuracy (F1 or other evaluatio metric) while also care about classfier running time.

--> method of considering mutliple evalutions
>1. could combine these two evaluation together accuray and running time and create a new evaluation metric, e.g: cost =accuracy -0.5 * running time; artificial formula

>2. Maximize one evaluation metric (accuracy), but subject to other evaluation metricx (running time)
right e.g: as long as running time is less 10 ms is ok, while accuracy higher, better-->classfier B.



2. Satisficing and optimizing metric
--Purpose: quick efficient judge when there are many metricl to evalute classfier performance
>1. satisficing metric: just need to meet one spefic value, and beyond that do not really care.
>2. optimizing metric: to maximize this evaluation metric: the higher the metric value means classsifier better (choose highest optimizing metric in the metrics that meet satisficing spec.)

recommendation: when there are multi concerns/ evaluation metrics, set one metric as optimizing metric (wan to as well as possible on), one or more as satisficing metric (have to meet some threshold, and do not care beyong that)
e.g: wake up system: optiming metric: accuracy when people did say wake up word, machie wake up; satisficing metric: number of false positive every 24h (machine wake up while peop did not say wake up word).

Note: It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.

3.1-5 Setting development and test set

Item purpose: the way you set up your training dev, or development sets and test sets, can have a huge impact on how rapidly you or your team can make progress on building machine learning application. -----> maximize work efficiency

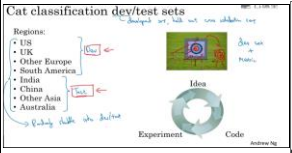
1. Cat classfication dev/test sets:

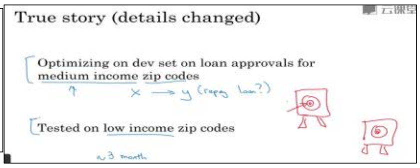
>1. Workflow in machine learning:
Try many ideas-->train up different modles on the training set -->use dev set to evaluate the trained different ideas and pick one.-->keep iterating to improve dev set performance, till have satisficed cost.(train in dev set with picked hyperparameter?)-->evaluate on test set.

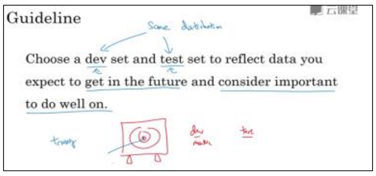
>2. Dev set: development set/ hold out cross validation set
Setting up the training, development and test sets have a huge impact on productivity. It is important to choose the dev and test sets from the same distribution and it must be taken **randomly** from all the data.

>>>1. principle: dev set:
setting dev set+ singal real number evaluation metric: is like placing a target to aim at, as: once set dev set + the metric, team can iterate very quickly, try different ideas, run experiments and very quickly use dev set and metric to evaluate classfier and try to pick the best one.
 -->machine learning good at shooting different arrows into target and iterating to get closer and closer to hitting the bullseyes. so doing well on the dev set, on the evaluation metric.
(note: dev set + single number evaluation metric function:
 -1 dev set: same distribution of test set, reprezentative of object in real world;
  same algorithm will have different performance on different data/object in real world-->dev set is to see algorithm performance on specify object in the real world.
 -2 single number evaluation metric: reprezentative of algorithm performance we want on that object. different metric could reprezents different performance: running time, F1, or none
   -->need to reprezent performance we cared/expected on the algorithm result.
   -->different signle number evaluation metric, may result in choosing different algorithm/classfire
dev set + singlue number evaluation metric--> algorithm 'specific performance' on 'specific object')

>>>2. dev set and test set different distribution:
 -->months of work spending optimizing to dev set, is not giving good performance on test set. is like: seting a target by dev set and metric -->try to aim closer and closer to the bull's eye during iterating -->test in different test data: move target to another place.
(note: like apply algorithm on different object, whose feature is also different-->algorithm parameter need to optimize for adapting these new data features).







2. Guideline for dev and test set setting

>1. Take all data and randomly, shuffled into the dev set and test set, (then have same distribution)
>2. Choose a development set and test set to reflect data you expect to get in the future and consider important to do well on.

So, whatever type of data expect to get in the future, and once you do well on, try to get data that looks like that. And, whatever that data is, put it into both your dev set and your test set. Because that way, you're putting the target where you actually want to hit and you're having the team innovate very efficiently to hitting that same target, hopefully, the same targets well.

summarize:
setting up the dev set, as well as the validation metric, is really defining what target you want to aim at(like evaluate an algorithm 's 'specfic' performance on 'specific objcet'). And hopefully, **by setting the dev set and the test set to the same distribution, you're really aiming at whatever target you hope your machine learning team will hit.**

3. Training set
 the way to choose training set affect how well you can actually hit the target (choose by dev set). ??
(> training set (input + label): algorithm parametered trained, algorithm per se is determined here;
 >dev set+ singular number evaluation metrix :: how algorithm performed on this 'specific' function on 'specifc' product.
)

3.1-6 Size of the development and test sets

The guidelines to help set up dev and test sets are changing in the Deep Learning era.

1. Old way of splitting data:

>1. data set is middle size:
training: 70% +test: 30% or training 60% +20% dev +20% test

>2. if data is much large: much smaller data percentage for dev and test, Much higher fraction of training set (e.g. 99%)

2. Test set:

>1. purpose: test set: evaluate how good the system is (evaluate final cost bias);

>2. test set size: Guideline: set test set big enough to **give high confidence** in the overall performance of system.
So, unless need to have a very accurate measure of how well your final system is performing, maybe you don't need millions of examples in a test set, and maybe 10,000 examples gives you enough confidence to find the performance , and this could be much less than, 30% or 20% of the overall data set, depend on how much data you have.

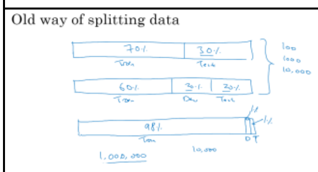
>3. No test set:
Not having a test set might be okay:
>>>1. for some applications, maybe don't need a high confidence in the overall performance of final system., and all need is a train and dev set.

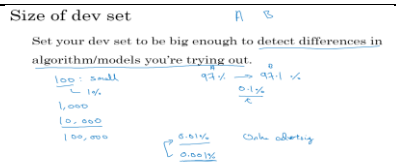
>>>2.Have a very large dev set so that think won't overfit the dev set too badly. Maybe it's not totally unreasonable to just have a train, dev set.

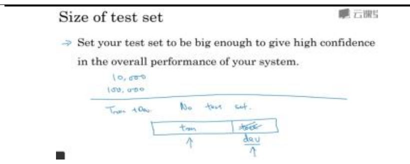
yet, recommend: have a separate test set to get an unbiased estimateion of algorithm performance.

3. Rule of thumb:
>1. set the dev set to big enough for its purpose: helps evaluate different ideas and pick this up better algorithm
(set dev set big enough to detect differences in algorithm/models you are trying out.)

>2. purpose of test set is to help you evaluate your final classfier. You just have to set your test set big enough for that purpose, and that could be much less than 30% of the data.







Summarize:
1. purpose:
test set: evaluate how good the system is (evaluate final cost bias);
dev set: evaluate different ideas, choose which classfier is better;
training set: feed data for training each classfier model

2. data set size is choosen acc. to it's purpose.
 may no needed, for some applications do not need high confidence in the overall performance of the system. then only training set and dev set is enough . or have very large dev set, make sure will not overfit dev set too badly, then not unreasonably to just have a train dev set (not recommend , test set better have to have confidence).
 --dev set: if actually tuning to the set , then this set is dev set instead of test set .
 --test set, will get unbiased estimate of how well the performance will be.

3. Guidelines
• Set up the size of the test set to give a high confidence in the overall performance of the system.
• Test set helps evaluate the performance of the final classifier which could be less 30% of the whole data set.
• The development set has to be big enough to evaluate different ideas.(and better big enough incase overfit dev set.)

3.1-7 Orthogonalizaiton

Have a dev set and evaluation metric is like placing a target somewhere for your team to aim at. But sometimes partway through a project you might realize you put your target in the wrong place. In that case you should move your target.
(note: singular number evaluation metric on dev set, should reflect real concerned performance in real world . while dev set-representative real world here.)

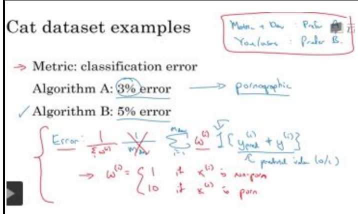
1. Example:
evaluation metric and user have different choice.

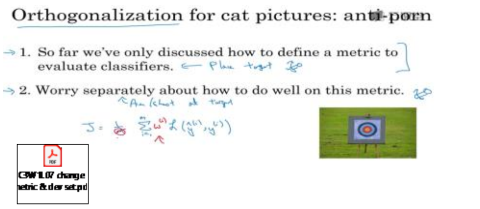
>1. Problem
E.g in right pic : classfier A high less error, yet could have badly false true (pornagraphic pic judge as true) which is not definately allowed , while classfier B will not let this happen. classfier B will be choosen even has higher error than A.

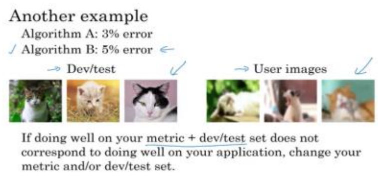
Classfier A have better value on evaluation metric, yet is worse algortihm: evaluation metric choose A, while user choose B.
-->evalation metric could no longer correctly rank ordering preferences between algorithms.matrix))
this is a sign to change target: evaluation metric or dev or test set.
(note: evaluation result not match concerns in real world-->something wrong with representative of real world (dev set) or representative of real world concenrns (singular number evaluation matrix)

>2. classfier error calculation:
in right e.g: sum y^ != y over all examples, divide by m : misclassified examples number fraction.

Evaluation metric problem: is error set / evaluation metric take equally/same weight for normal pic, and pornagraphic pic, while do not want pornographic mislabed as cat.-->update to add diffrent weight on error for normal pic and pornagraphic pic (e.g 10 size weight of normal pic).







2. Define a new evaluation metric or dev se, or test set:

>1. when evaluation metric could not correct rank order preference for what is actually better algorithm.
If not satify with old error metric then try to define a new one that better capture preferances/orderin in terms of what actually a better algorithm.

(-->signular number evaluation matrix; more representative of real concerned performance
--> better evaluation result/ordering of different algorithm,)

4. Another example: dev data diffrent distribution from user

e.g: classfier B higher error on dev, yet better usage for user, while user example is different distribution from dev set.
-->change dev set to have same distribution as user image.

Guideline : if doing well on metric and dev sets or dev and test sets' distribution,yet does not correspond to doing well on the application actually care about, then change target: evaluation metric and / or dev/ test set.
in this case do not change training set?

Above is actually an example of an orthogonalization: where take a machine learning problem and break it into distinct steps.

3. Orthogonalization: in machine learing:
machine learning split into two steps:
>1. Define metric to evaluate classifiers./captures what you want to do.
(note: evaluation matrix :
>>1. defined only acc. to real world concerns: need to be representative of real world concerns
>>2. only affect rank/ordering of different trained algorithms, no other infuence on algorithm per se
)

>2. worry seperately how to do well on this metric.
(note: algorithm performance per se:
>>1. only determined by trained parameters that trained in training set.
>> 2. has no influence/affect on others like evaluaton matrix defination)

Use target analogy: the first step is to place the target as a completely separate problem.
Think of it as a separate step to tune in terms of how to do well at this algorithm, how to aim accurately or how to shoot at the target. Defining the metric is step one and you do something else for step two.

In terms of shooting at the target, maybe learning algorithm is optimizing some cost function, minimizing some of losses on your training set, or also use modified loss function ..(e.g. cost function, weight, hyperparmaters modify)

Summary:
Having an evaluation metric and the dev set allows you to much more quickly make decisions about which Algorithm is better. It really speeds up how quickly you and your team can iterate.- ??
So recommend: , even if you can't define the perfect evaluation metric and dev set, just set something up quickly and use that to drive the speed of your team iterating. And if later down the line you find out that it wasn't a good one, you have better idea, change it at that time, it's perfectly okay.

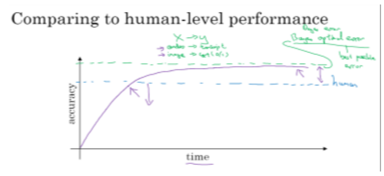
But recommend against for the most teams is to run for too long without any evaluation metric and dev set up because that can slow down the efficiency of what your team can iterate and improve your algorithm.

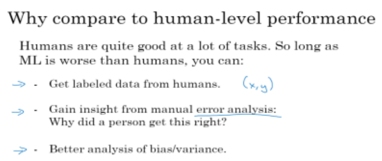
dev set and evalutation, how faster algorithm iteratin efficency?-choose better ideas/fatures/models
cost function used to penalting predicted error by updating parameter, -->evaluation on dev set is only used to pick up hyperamter? also different models -ideas, features ,etc.

3.1-8 Compare with Human

1. Comparing the machine learning systems to human level performance

1.1 Reason:
>1. Due to advances in deep learning, machine learning algorithms are suddenly working much better and so it has become much more feasible in a lot of application areas for machine learning algorithms to actually become competitive with human-level performance.
>2. Second, it turns out that the workflow of designing and building a machine learning system, is much more efficient when trying to do something that humans can also do.





1.2 Examples:

for many machine learning tasks:
>1. progress/accuracy tends to be relatively rapid as you approach human level performance.

>2. But when algorithm surpasses human-level performance , progress and accuracy actually slows down: maybe it keeps getting better but the slope/speed of how rapid the accuracy's going up, often slows down.

progress often slows down when you surpass human level performance because:
>>>1. human level performance is for many tasks not that far from Bayes' optimal error. (e.g:People are very good at looking at images and telling if there's a cat or listening to audio and transcribing it).
So, by the time you surpass human level performance maybe there's not that much head room to still improve.

>>>2. so long as performance is worse than human level performance, then there are actually certain tools could use to improve performance that are harder to use once surpassed human level performance.

-Get labeled data from humans.
Ask humans to label examples so that can have more data to feed learning algorithm.

- Gain insight from manual error analysis: why did a person get this right?
So long as humans are still performing better than any other algorithm, you can ask people to look at examples that your algorithm's getting wrong, and try to gain insight in terms of why a person got it right but the algorithm got it wrong.
- get a better analysis of bias and variance

>3. And the hope is it achieves some theoretical optimum level of performance:
As you keep training the algorithm over time,, maybe bigger and bigger models on more and more data, the performance approaches but never surpasses some theoretical limit- which is called the Bayes optimal error.

So Bayes optimal error: think of this as the best possible error; is the very best theoretical function for mapping from x to y, that can never be surpassed.

Summary:
so long as algorithm is still doing worse then humans then you have these important tactics for improving your algorithm. Whereas once your algorithm is doing better than humans, then these three tactics are harder to apply.
So, this is maybe another reason why comparing to human level performance is helpful, especially on tasks that humans do well , and why machine learning algorithms tend to be really good at trying to replicate tasks that people can do and kind of catch up and maybe slightly surpass human level performance.

3.1-9 Avoidable bias

Expect learning algorithm to do well on the training set but sometimes do not actually want to do too well (overfitting)and knowing what human level performance is, can tell you exactly how well but not too well you want your algorithm to do **on the training set.**

1. Example:
Assumption: Consider human level preformance as estimation of bayes optimal error (reasonable esp. for computer vision task: as human pretty good at computer vision and so whatever human can do is may be not too far from bayes error)
training error 8%, dev error 10%

>case 1:human error 1% (assume it as bayer error)
there's a huge gap between how well algorithm does on your training set versus how humans do , shows that algorithm isn't even fitting the training set well.
So in this case would focus on reducing bias: e.g train a bigger neural network or run training set longer

>case2: human error 7.5% (assume it as bayer error)
not much head room for training error to improve, and may overfit the training set-->think accessable.
instead there are more room between training error and dev error 2%, to improve.-->variance reduction tactics (overfit training set; --> regulation, more training data)

summarize:
>1. By definition, human level error is worse than Bayes error because nothing could be better than Bayes error, but human level error might not be too far from Bayes error.

>2. Surprising thing saw in this example: depending on what human level error is or really this is really approximately Bayes error or so we assume it to be, but **depending on what we think is achievable,** with the same training error and dev error in these two cases, we decided to focus on bias reduction tactics or on variance reduction tactics.

Learning human level performance could tell how well and not too well algorithm expected to do in training set.

bias error: difference between training error and zero.
Note: no problem to use when bayes close to 0 (e.g where human is exremly good at, human error is also close to 0). but when bayes error not close to 0, need to consider to avoidable bias for better judging which error to focus (bias or virance)

avoidable bias: difference between bayes error or approximation of bayes error and training error.
note: training errror could not do better than bayes error unless overfit.

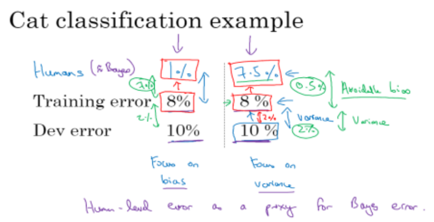
algorithm vairance problem: difference between training error and dev error. algorithm ability to generalize from training set to dev set.

Avoidable bias: difference between Bayes error or approximation of Bayes error and the training error.
So what you want is maybe keep improving your training performance until you get down to Bayes error but you don't actually want to do better than Bayes error.
Algorithm can not actually do better than Bayes error unless overfitting.

Avoidable variance: difference between training error and the dev error.
A measure of algorithm variance problem. -algorithm generalization ability: from training set to dev set.
(note: for good algorithm generalization:
>> 1. algorithm architecuture per se: fit training set well , not overfitting (otherwise these 'over' weights will casue more error on dev /test set)
>>2. training set feature should cover dev/test set feature, --.>large training set

if could not generalize good-->algorithm architure overfitting training set (regularization) or/and training set feature could not cover dev set)

Summary:
understanding human level error, understanding your estimate of Bayes error really causes you in different scenarios to focus on different tactics, whether bias avoidance tactics or variance avoidance tactics. There's quite a lot more nuance in how you factor in human level performance into how you make decisions in choosing what to focus on.



3.1-10 human level performance

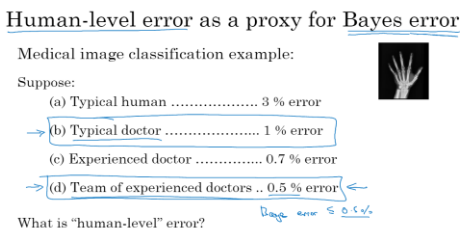
1. Human-level error as a proxy for Bayes error

>1. human level performance defination: as a proxy or an estimation of bayes error

2. Bayes error: best possible error any function could even now or in the furture could ever achieve.
for tasks human can do very well, can use human error as proxy of bayes error.
e.g medical image classficiation example: get different human error from different person/team.
bayes error is less than any human error, --->bayes error <=0.5%

find human error: :
1. one of the most useful ways to think of human error is, as a proxy or an estimation of bayes error
so bayes error is less than human error achieved currently.

2. depending on the purpose of defining human level error, human error could be different values.
e.g: if goal is to surpass signal human, then human error could set 1%; if goal is a proxy of bayers error, then human error could set 0.5%



2. Error analysis example

-->human error getted: 1%-common person, 0.7%-signal doctor, 0.5%-doctor team

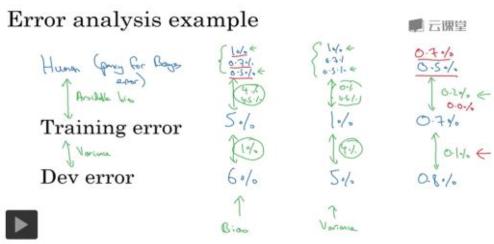
>.case1: training error 5%, dev error 6%
in this case, no matter use which human error as bayes error, avoidable bias is high, -->focus on bias issue.

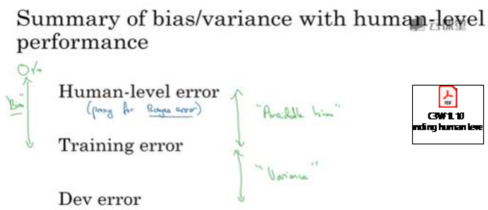
>case2: training error 1%, dev error 5%
in this case, no matter use which human error as bayes error, avoidable variance is high, -->focus on variance issue.

>case3: training error 0.7%, dev error 0.8%
in this case , really matter to use bayer error as 0.5%. as different proxy of bayer error (1% / 0.7% / 0.5%)will direct to focus on different issue or both issue.

Making progress in a machine learning problem gets harder as achieve or approach human-level performance: might not know how far away algorithm are from Bayes error. And therefore no idea how much you should be trying to reduce aviodable bias. might be very difficult to know if you should be trying to fit your training set even better.

this problem arose only when you're doing very well on your problem already. When algorithm further away human-level performance, it was easier to target your focus on bias or variance.





Summary:

>1. human-level error
can use human-level error as a proxy or as a approximation for Bayes error, **for a task that humans can do quite well.**

>2. Training error:
-->Avoidable bias: difference between your estimate of Bayes error and training error.
Tells you how much avoidable bias is a problem

>3. Dev error:
-->Avoidable variance: difference between training error and dev error.
Tells how much variance is a problem: algorithm's ability to generalize from the training set to the dev set.

>4. Bayes error: 0 or not 0?

>>> 1.Sometimes Bayes error is non zero: sometimes it's just not possible for anything to do better than a certain threshold of error:
e.g: examples where the data is noisy, like speech recognition on very noisy audio where it's just impossible sometimes to hear what was said and to get the correct transcription. For problems like thist, having a better estimate for Bayes error can help you better estimate avoidable bias and variance

>>>2.for cases that humans are near perfect for that, then Bayes error is also near perfect for that. So that actually works okay take Bayes error as zero.

Recap:
having an estimate of human-level performance gives an estimate of Bayes error, allows to more quickly make decisions to focus on reducing algorithm bias or variance.
These techniques will tend to work well until surpass human-level performance, where upon you might no longer have a good estimate of Bayes error that still helps you make this decision really clearly.

Note: in deep learning, there are more and more tasks we're actually able to surpass human-level performance.

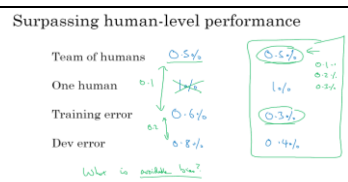
3.1-11_surpass human level performance

1.Surpasssing human-level performance

e.g:
Human error : 0.5%; training error 0.3%, dev error 0.4%.
No idea bayes error : 0.1, 0.2, or 0.3%, etc...

-->no enough info. to tell focus on bias or variance, slow down the efficiency where should make progress.

once surpass human level performance : bias is close to 0 or negative, while viarance is also very small
--hard/slow down to improve as:
1. no ennoug info. to judge whether should foucs on bias or viarance. (no idea bayes error value,)
machine learning could still improve, but no idea which direction to go now.



2. Probemls ML surpasses human-level perforamnce

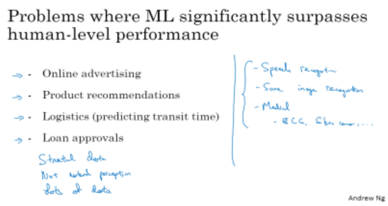
>1. No natural perception problem:

like advertising, product recommendations, logistics, loan approval, etc. ML surpassed singal human .

Machine learning Supassed area: **based on structured data:**
Where have a huge database for user info., action,etc. access to these huge data is key point.

in these tasks, there are teams that have access to huge amounts of data, these applications have probably looked at far more data of that application than any human could possibly look at. And so, that's also made it relatively easy for a computer to surpass human-level performance.
The fact that there's so much data that computer could examine, so it can better find statistical patterns than even the human can not.

Note: human tend to be very good in natural perception taks, harder for ML to surpass human-level performance on this task.



>2. othes areas not based on structured data: natural perception (human good at),
Speech recognition systems, some computer vision , language processing, some medical task surpass singal person.
there are cases in deep learning, surpass human in natural perception.

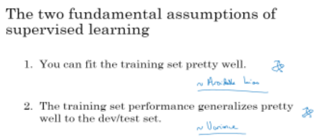
Surpassing human-level performance is often not easy, but given enough data there've been lots of deep learning systems have surpassed human-level performance on a single supervisory problem.

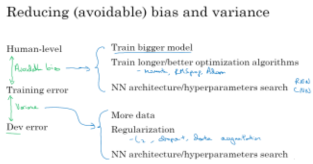
3.1.-12 improving model performance

Improving your model performance
1. The two fundamental assumptions of supervised learning to do things well
>1. Fitting training set pretty well.
which means achieve low avoidable bias;

>2. Training set performance generalizes well to the development set and test set.
which means low acceptable variance.

in spirit of orthogonation, avoidable bias could tuned speretely by training bigger network or longer.
sperate set to reduce viarance difference: regularization, or getting more training data.





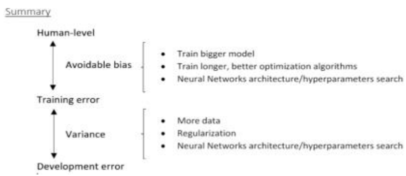
2. Reducing (avoidable) bias and variance

>1. Avoidable bias:
looking at the difference between training error and your proxy for Bayes error and just gives you a sense of the avoidable bias: just how much better do you think you should be trying to do on your training set.
Technique could use for bias: (focus on model /algorithm per se: architecture, hyperameters, optimizer-faster converge)
- training a bigger model (layers, hidden units),
-training longer, better optimization algorithms (momentum, RMS , Adam)
-change the neural networks architecture (RNN, CNN))or try various hyperparameters search, .

>2. Avoidable variance:
look at the difference between dev error and your training error, as an estimate of how much of a variance problem you have: how much harder you should be working to make your performance generalized from the training set to the dev set that it wasn't trained on explicitly.
Technique could use for variance:
->> Bigger training data set: (better cover dev /test set features)
getting more data to train on, help generalize better to dev set data that algorithm did not see--> mainly influence dev error

-..Regularization: drop out, data agumentation (regularization mainly influence dev error)
('modify' algorithm per se for overfiiing training set-->penalt these 'over' weights)

->> Change the neural networks architecture or try various hyperparameters search.
('modify' algorithm per se: --reduce 'overfitt' , & choose hyperamters that adapt to dev set better (training set feature not fully cover dev set)).-->influence training error & dev error.



3-2_Machine Learning

3.2_1 carrying out error analysis

Error analysis:
If machine learning worse than human level performance , process: examining mistakte algorithm is doing manually can give insight what to do next. this proces is erro analysis.
(now is avoidable bias problem: need to improve architecture : hidden layers , unit numbers etc. erreor analysis to help how to work on improving achriteture-input fetures, training or dev or test set, etc.

1. use error on dev set: not examples on traini g set,as dev set more representative of algorithem error- do not see examples before.
2. variance not be affected by this.-should look at examples error on test set??
input feature imroved should also improve J_dev?
error analysis help to find if working on one idea helps or not).

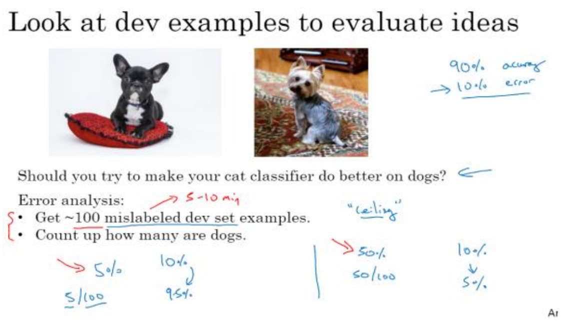
1. Example:

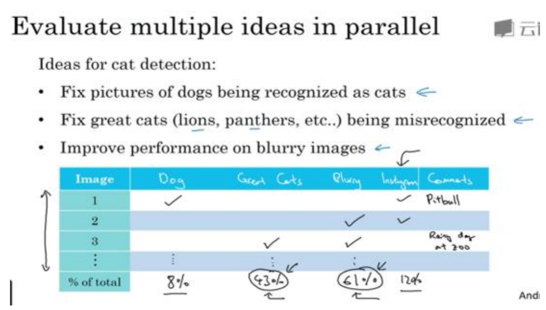
e.g: there are dogs mislabled as cat-->should start a project focus on dog problem?
whether spending months of time sovling this could improve performance?

Analysis process will help get to know whether this direction /this idea could be worth effort or not **by estimate ceiling performance.**
Ceiling performance: the best case/how well could working on the specific problem helps.

Error analysis:
>1. get-100 mislabeled dev set examples.
>2.: count up how many are dogs. exame the 100 manually, cout them up one at a time to see how many of these 100 mislabeled examples in dev set are actually dog pictures.
If only 5%, then fix dog problem could only reduce 5% dev error. still have 95% error left in dev set.
If 50% of 100 examples is dog, could be much more optimistic about spending time on dog problem.

Manual work (simple counting procedure) in building applied systems, can save a lot of time to decide what's the most promising direction to focus on.





2. Evaluate multiple ideas in parallel

Error analysis : could evaluate whethe ror not a single idea is worth working on. coul also evaluate multiple ideas in parallel.

Error analysis table: row- each misrecognized example in dev set; colums-category error concerned

>1. Get certain volume examples that algorithm has misrecognized on dev set,
>2. Go through each of the examples(listed in row) manually: for each example,
check if the error caused by the each of the error categories that would like to work on for improving perfromance-listed in columns.

>3. Check precentage of each catagory error's attribute to dev error.(each column's error summary)
go down each column and count up percentage of images have a check mark (judged that misrecognitation caused by this category error) in that column.

-->Pecentage conclusion give esitmation how worthwhile it might be to work on each of these different catagories of error.
Note:
1. could add more ideas/errors (column) in analysis table while part way through the process.
2. examples used are: dev set examples that algorithm has misrecognized.

Summary:

To carry out error analysis, should find a set of mislabeled examples in your **dev set.** And look at the mislabeled examples for false positives and false negatives. And just count up the number of errors that fall into various different categories.

During this process, you might be inspired to generate new categories of errors, you can create new categories during that process. But by counting up the fraction of examples that are mislabeled in different ways, often this will help you prioritize, or give you inspiration for new directions to go in.

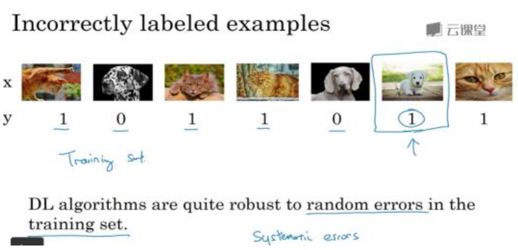
3.1.-2clean up incorrectly labled data

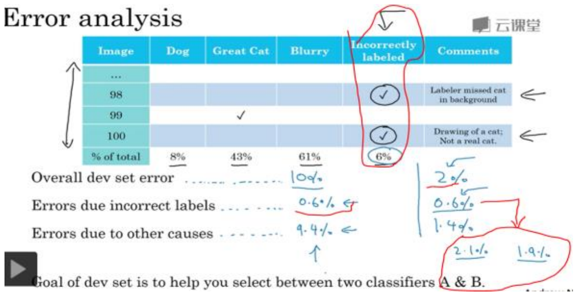
1. Incorrectly labeled example:

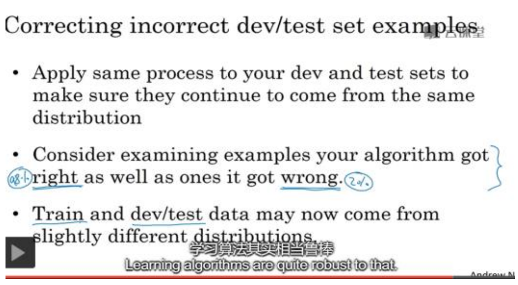
incorrectly labled data: in training data/dev set/test set, human label (y) assigned to this data is incorrect.
does it worth a while to fix these incorrectly labled data?

1.1. Training set: incorrectly labeled
Algorithm quite robust to **random errors** in training set. (note : but less robust to system errors)
So long as these errors/incorrectly labled data are not too far from reasonable random (sometimes laber did not pay attention or accidently randomly hit the wrong key in keyboard), it probable ok to leave these error and not spend time to fix them. as long as total training data is big enough, the actualy error percentage is maybe not too high.

-->system errors: e.g;: laber consistly making wrong label, (not random)







2. Incorrectly labed in dev set/test set
-->in error analysis metric, add one more column for 'incorrectly labed error', -->count number of examples in this error, to **judge ceiling improvment by fiximg this problem**.
(labeled data in dev is for evaluating different ideas improve algorithm performance or not/which model is better.
if data mislabeled, check how much this affect dev set purpose: influence on ability to tell difference of models' performance on dev set
Do error analysis.)

E.g: take 100 example in dev set where output is disagree with labed data, then count up percentage in all errors in metrics.-->check significance to your ability to estimate algorithm on dev set/test set, then judge if spend time to fix error label problem. (if this incorrectly label on dev set does not influence picking better algorithm based on their performance on dev set, then could leave these examples alone)

2.1: analysis process:
>1rst: look overall dev set error:
> 2nd: error due to incorrect label (In the dev set error)
>3rd: error due to all the other causes (in dev set error)
if 2nd error is big percentage of dev set error, should fix incorrect lable.
(influence the ability to tell models apart based on their performance on dev set.
if dev set is big enough, fraction of mislabeled data is very small, then may could like training set, robust to these mislabeled dev data.)

Note:
goal of dev set is help to select better classfier, if incorrect error influence choice (classfiers' dev set very close, and error due to incorrect label could influence comparision): e.g classifier A dev set error 1.9%, B is 2.1%. yet there are 0.6 % dev set error caused by incorrect label.-->need go to fix incorrect label

(note: decide direction to work on based on error on dev set:
>1. error catergory percentage on dev error examples-->get impression of algorithm ceiling performance for fixing this error;
>2. error catergory influence on dev set purpose: e.g: even if one error catergory has small percentage on dev error, yet this could influence singular number evaluation matrix/dev error to make decisons which model/idea/hyperatmer is better , then need to fix this error .)

3. correct incorrect label in dev/test set

>1. Apply same process to dev set and test set at same time, to make sure same distribution.
Dev set set target, and algorithm optimized based on this target then generalize to test set.
works more efficient to dev and test set come from the same distribution.
if go to fix something on the dev set, apply same process to the test set.

>2. consider examining examples algorithm get right as well as one get wrong:
Easy to examne wrong output, and judge need to fix or not. but if only fix wrong example, may end up with more bias estimates of algorithm error.-->need doubl check what algorithm get right. (fixing wrong examples may end up make algorithm previous right output become wrong)

2nd item not always done: as if classfier is very accurate, the right examples is much more than wrong examples, very easy to fix wrong sampels while it takes longer to validate right out examples.-->just keep in mind of this item, to consider.

>3. may decide only to fix incorrect labe in dev/test set, not in training set
As algorithm is more robust to random error in training set and training set is much more larger than dev/test set , traing set may not do correction.-->traing set & dev set/test come from different distribution, while deep learning algorithm quite robust to this different distribution.

Note:
super important: dev set and test set come from same distribution;
reasonable : training set distribution different from dev set.
(note:
>>1. if these correction done in dev/test set (20%, 20% data) is for quite small volume example, -->data distribution differe in dev and training is much smaller percentage of training data volume--yet training set have same percentage error...not small percentage error in training set, need to correct still??
>>2. training set still should cover all dev/test/real world object feature) for algorithm generalize to dev set)

Summary:
1. In building practical systems, often there's also more manual error analysis and more human insight that goes into the systems than sometimes deep learning researchers like to acknowledge.

2. To understand what mistakes algorithm is making, actually go in and look at the data and try to counter the fraction of errors. because a small number of hours of counting data can really help you prioritize where to go next.

Error analysis is a very good use of your time when you are trying to decide what ideas or what directions to prioritize things.

3.1-3: build first system quckly then iterate

1. Build new system

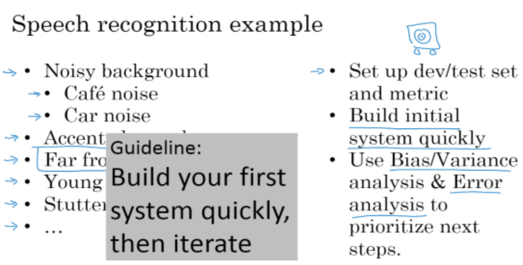
when building fresh new system there are many direction and lots of things could priortize.
Problem/Challenge is how to choose a direction to focus on.

1.1 Recommendation for building brand new machine learning application:

Recommendation: Build new system quicliy and then iterate:
>1. set up dev/test set and metric:
set target first and if wrong could move this target later. just set a target somewhere.

>2. Build initial machine learning system quickly:
find the traing set and train and see, and start to see how well performed in dev set.

>3. use bias / viarance analysis or error anlaysis to priotirize next step.(e.g. far from microphone to focus on)
(note: seems
1. bias/variance analysis: is checking algorithm underfitting or overfitting;-find higher level, general algorithm problem;
2. error analysis: is for checking if one idea works or not. could help underfitting or overfitting or may help both issue.)



Summarize:
Do not think to much/overthink/too complicated, build simple system. initial system's value is to have some trained system to allow you to localize bias/viarance, to figure out all the most worthwile direction could go.

note: this adivce applies less strongly if working on an application area in which you have significant prior experience, or there are acadamic literature could draw on for pretty much same problem you are building.

3.1-4: training and testing in different distribution

Deep learning algorithms have a huge hunger for training data. They just often work best when you can find enough label training data to put into the training set.

This has resulted in many teams sometimes taking whatever data you can find and just shoving it into the training set just to get it more training data. Even if some of this data, or even maybe a lot of this data, doesn't come from the same distribution as your dev and test data. . (happens when expamples from wanted/cared distribution is not big engough) and algorithm performed better (than only use small real cared data).

(note: even training and dev/test set come from different distributation, in this case , training set still cover dev/test set feature/distritutioin-->algorithm trained could generalize to dev/test set)

So in a deep learning era, more and more teams are now training on data that comes from a different distribution than your dev and test sets.

1. Example: cat classification
user image: a, really care about, which come from mobile app, less professionally shot,less volume. (too small volume to train algorithm)
data from web.: b more professionally taken, huge numbers of images. different distribution from user image.

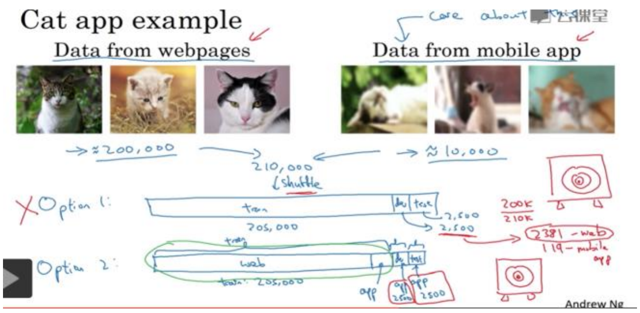
Concerns: final algorithm does well on user image.

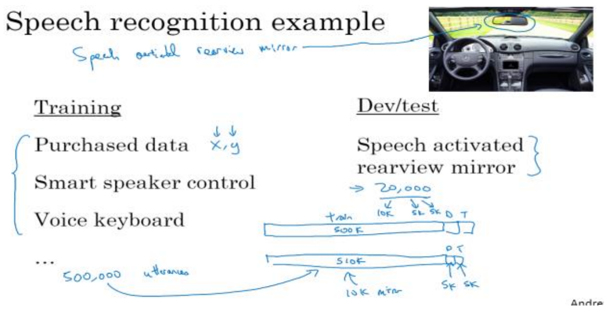
1.1 Seting training , dev, test set:
> option1: mix a & b, and random shuttle into training set and dev, test set
---> Advantage: training , dev, test set all come from same distribution;
---> Disadvantage: dev and test set percentage from b is too small, far from target (data b).

not recommended: as target set by dev data, most of which is not really want.--> targe come from different distribution than what really cared.

> option2: dev & test set all come from distrubiton data really cared b. rest data in b mix with a to use as training set.
--> Advantage: aiming to the target really want
-->Disadvantage: traning set and dev set distribution diffrent, but over the long term algorithm work well.

this split of data into train, dev, test set, will get better performance over the long term.
(training set: mix a & b-part of real care data: hope with large mounts of different distribution data a, could cover real cared data b feature;
a need to be large as one example of it could provide much less info. than real cared data example.)





2. Speech recogniation example

Training set: data accumulated from working on other speech problems, eg.: purchased data, voice keyboard...

Dev /test set: smaller data, actually come from a speech activated in this application. Very different from training data.

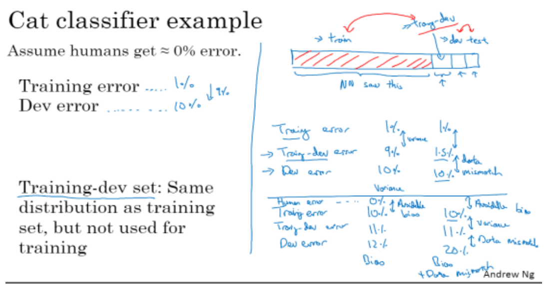
Dev and test set: all come from atually cared data: activated by this applicaiton; could put all or part of cared data into dev/test set and rest into training set.
Training set: data from other applications and part of actually cared data.

Get much bigger training set for algorithm than using only cared data, get algorithm performance better.
Note: not always need to use all the data have.

3.1-5 bias and viarance with mismatch different distribution

way of analyxing bias and viarance changes when traing set and dev set come from different distribution.

1. Example: training set and dev set has different distribution



now have three errors: bias, viarance, and data mismatch

e.g: bayes error ~~human error is 0%, training error is 1%, dev error is 10%.

> 1 If training set and dev set are same distribution, -->focus on viarance error.
>2. if training set and dev set different distribution -->hard to come conclusion which direction to go, as:
dev error higher than train error, maybe because dev set is much more harder to identify than training set.

when training set and dev set come from different distribution, two changes in dev test here:
>1. algorithm did not see dev set before (contribute to viarance error);
>2. different distribution.-->contribute to mismatch error.

2. Method for distinguishing two changes contribution to difference of dev error and training error:

Add training-dev set: randomly shiftting training set and then carve out a piece of training set to be training-dev set.

>0: human error/bayers error

>1.training error:

>2. training dev error:
different between training dev and training error-->viarance problem, algorithm generalization ability. (algorithm did not see training dev set before)
-->training dev set did not go backward propagation.

>3. dev error:
different between training dev and dev error-->data mismatch problem

algorithm did not trained on data from training dev or dev set, while these two set come from different distributions. if algorithm does well on training dev, but not well on dev set,-->algorithm has learned to do well on a diffrent distribution than really care about-dismatch problem.

>4. test error:
difference between test error and dev error-->overfit degree to the dev set.(need to find bigger dev set)
test and dev set come from same distribution.

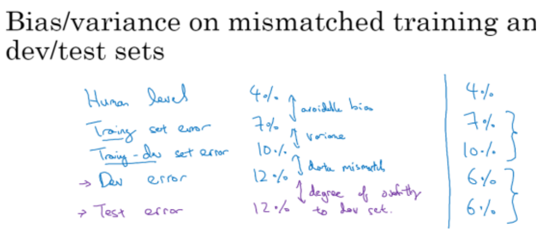
note: do not develop on test set as not want to overfit.

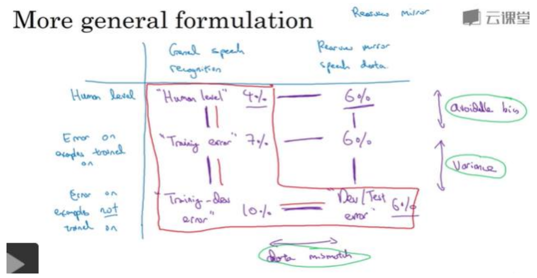
3. Examples:

huaman level error, training error, training-dev error, dev error, test error, normally becomes bigger from human level error to test error.

There are cases, error do not increase:
human level: 4%, training error 7%, training-dev error 10%, dev error 6%, test error 6%.

this case may be: training set is more harder to distinguish than dev/test set.
-->need more general formulation for analysis.





4. More general formulation:
Fill in table :
colums: different data set: training data (data from general speech reccognition), dev/test data (really cared data: rear mirror speech data).
rows: human-level performance, algorithm error on trained example (training error), algorithm error on examples not trained on (-->training dev error, dev/test error)

note: do not trained -do not go through backward prop on this data set.

Analysis table:
first row: human-level error on training data,
second row: training error
3rd row: training -dev error; dev/test error.
-->get to know avoidable bias, variance, and mismatch error.
----

4. More general formulation:
----
could also get the other two error:
human-level error on dev/test data (could label dev/test set and measure how good humans are at this task): e.g 6%
error on examples trained-dev/test set: could get y puting dev/test set in training set so algorithm learns on it as well, then measure the error on that subset of the data.:6% in right e.g-->algorithm doing quite well on this distribution of data as same as human error on this data-6%.

This formula -->
no only get fouces direction(bias, viarance, data mismatch)-red rectangular, but also get insight for other features:
E.g righ pic: human error 4% less in training data than dev/test set, may means it harder to distinguish 'rear mirror speech data' even for human.
(note: above e.g:
1. human error on training set < dev/test set-->may be dev /test set is harder to recognize;-->distribution problem
2. avoidable bias 3%-->may need focus on algorithm architecure/learning time
3. avoidbalbe varainte 3%-->may need focus on overfitting: regulation
4. Mismatch error -4%: -->may need to foucs on different distribution problem-may means dev set are easier to recognize?)

for data mismatch problem, no systematic way to address.

summary:

Using training data that can come from a different distribution than dev and test set, this could give you a lot more data and therefore help the performance of your learning algorithm.

But instead of just having bias and variance as two potential problems, now have this third potential problem, data mismatch. and It turns out that unfortunately there are super systematic ways to address data mismatch, but there are a few things you can try that could help.

3.1-6 addressing mismatch

1. Addressing data mismatch

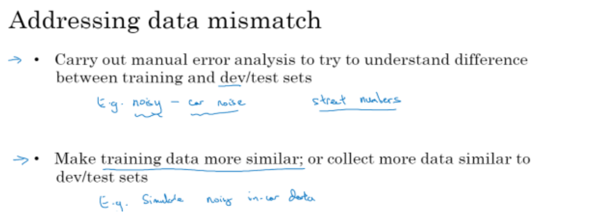
No systematic method. Could try:
1.1 carry out manual error analysis to try to understand differances between traning and dev/test set.

Note: incase of overfit, should manually look at only dev set not test set.

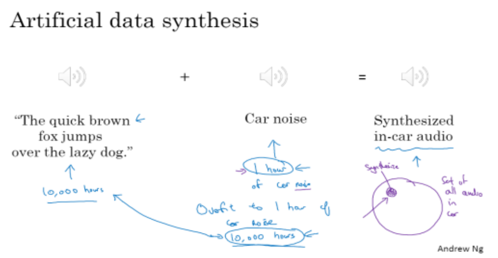
>1. understand catergary errors in dev set vs training-dev set
>2. get insight the feature how dev set different /harder than training-dev set.
(note: look into errors on dev set, and erros on training -dev?
1. errors on dev set:-->categories 1 that matter
2. errors on training -dev set-->catergories 2 that matter
compare these two catergires to find distribution difference on dimensions that matter?
)

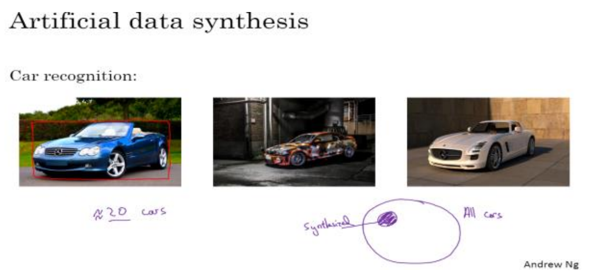
1.2: . Make training set more similar, or collect more datat similar to dev/test set.

e.g if more noise/new word in dev set, add these , combine to training set.
Technique could use: aritificial data synthesis-saving time and closer to dev set distribution/feature;



Note:
this is a rough guideline for things could try, not a systematic process and it's no guarantee that you get the insights you need to make progress.
But this manual insight**-understanding difference in distribution**, **+ together with making the data more similar on the dimensions that matter ,** this often helps on a lot of the problems.





2. Artificial data synthesis;

note : there is risk if repeated frequently combine one small/short data/feature into training set, neural network may overift this feature/data while human seems no problem/could not realize the diffrance .
as this small data/noise maybe just very small subset of the data/noise occured in cared situation /space.
(note: overfit subset noise: consequence:
could eliminate noise combined in training set very well, yet could not generalize well to dev/test set: could not eliminate well noise in dev/test set-noise space feature is not covered by training set noise)

2.1: Speech recognatin example
e.g: repeated 1000times of 1h noise into 1000h speech record may overfit noise. while this 1 hour noise maybe only very small subset of really cared noise space. better use long time nose to combine into 1000h training set.

use unrepeatable noice added in training data may could help get better algorithm performance.

2.2: example_car recognition;

use artificial image to train algorithm,
eg. training set has only 20 distinct cars, then algorithm may overfit these 20 cars, and it's difficult for a person to easily tell that you are really convering such a tiny subset of the sets of all possible cars.
(trained algorithm captured car feature in training set, while real word car, has more feature than training set car-->algorithm could not genralize to dev/test set well)

Summary:

If there is a data mismatch problem, do error analysis, or look at the training set, or look at the dev set to try this figure out/ gain insight into how these two distributions of data might differ.

And then find some ways to get more training data that looks a bit more like your dev set.
One of the ways is artificial data synthesis, which does work, esp. in speech recognition.

But while using artificial data synthesis, just be cautious and bear in mind whether or not you **might be accidentally simulating data only from a tiny subset of the space of all possible examples** (human may hard to recognize the difference of subset and all possible space).

3.1-7 Transfer learning

One of the most powerful ideas in deep learning is that sometimes you can take knowledge the neural network has learned from one task and apply that knowledge to a separate task.

e.g: maybe you could have the neural network learn to recognize objects like cats and then use that knowledge or use part of that knowledge to help you do a better job reading x-ray scans. This is called transfer learning.

1. Transfer learning:

1.1 Examples:
e.g: 1: transfer algorithm from cat image recognition to x-ray pic recogniton;
image features like curve, line, edge. etc. learned from lower layer, could help /use for new task x-ray pic feature capturing.

e.g 2: speech recognation algorithm transfer to 'wake up' system.
speech recognation algorithm trained to output text based audio input.
take out algorithm output layer, and create not just a single new output , but could also create several new layers to neural network to try to predict label y.

1.2 Transfer learning process:
>1. Remove algorithm outlayer (also parameters in last layer)
>2. Create a new set of randomly initialized parameters just for the last layer, and have that output new task.
>3. Retrain the neural network on this new data set.

1.2: Retrain modified neural network on new data set:

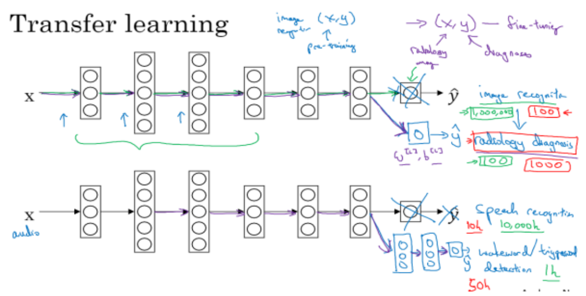
>>1. if only have small new data, could retrain neural network on last one /two layer weight, and output on new training set. or remove last layer and add more layers and train on new training set. while keep other layers' parameters fixed.

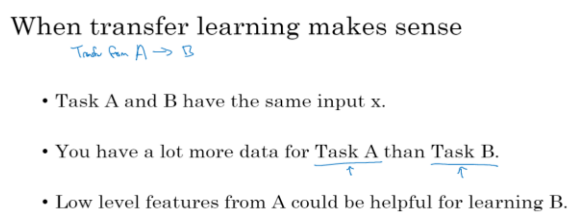
Aappliable as: lot of the lower layer featured/knowledge (such as detecting image features: edges, curves, positive objects, etc), learned from a very large previous task data, might help algorithm do better in new task-learn fast or use less data.
(note: lower layer features/knowledge learned:
>>1. architecture: actually is the layer setup-hidden units number etc + trained parameters
>>2. these layer architerure & trained parameters: could commonly used in other task's examples for capturing same kind of feature/knowldege
>>3, for new task, algorithm lower layer need to capture this kind of feature

like conv compuation on computer vision: same filter (lower layers) used in different image position (different examples here)

-->use trained lower layer architure & parameter only when:
1. new task need same feature/knowldege in lower layer
2. new examples: should same type with trained examples: so lower layer architure could be commonly used
)

>>2. If have lots of data, could retrain all parameters in the network-called fine tuning (pre-tuning: is training on previous data , get weight pre-trainined).





2. Transfer learning make sense when:
>1. Both task A, B have same input X (catagory): both are pic. or video, etc.
(note: condition to share lower layer architecture & weights/could use transfer learning)

>2. have a lots more data for the problem that is transfering from than for the problem is transferring to.
(note: motivation for using tranfer learning)

new task data is more valuable for new task, usually need a lot more data for Task A, as each example from task A is less valuable for new task than new task example.
(note: and hope these large less valuable data could cover new real cared data feature)
If opposite, new data much more than old data, then use new data training system, old training data less valuable and will not gain much from that.

> 3. Low level features from task A could be helpful for task B.
a lot of knowlege/features learned from previous task can be transferred and really help new task even if have small data for new task.
(note: condition for using tranfer learning
>>>1.architecture and trained weights could capture similar feature on new task examples <--same input X (catagory)
>>>2. these feature/knowledge caputured is heplfull for new task)
)

Summary:
Transfer learning has been most useful if you're trying to do well on some Task B, usually a problem where you have relatively little data.
So for example, in radiology, you know it's difficult to get that many x-ray scans to build a good radiology diagnosis system. So in that case, you might find a related but different task, such as image recognition, where you can get maybe a million images and learn a lot of load-over features from that, so that you can then try to do well on Task B on your radiology task despite not having that much data for it.

When transfer learning makes sense, It does help the performance of your learning task significantly.

(note: seems kind of like breakdown problem?
e.g: no enough data for end to end task A-->break down to sub-task B, C that has engough data seperatedly.)

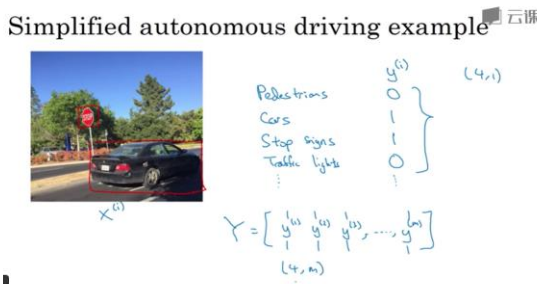
3.1-8 mulit task learning

So whereas in transfer learning, you have a sequential process where you learn from task A and then transfer that to task B. In multi-task learning, you start off simultaneously, trying to have one neural network do several things at the same time. And then **each of these task helps hopefully all of the other task.**

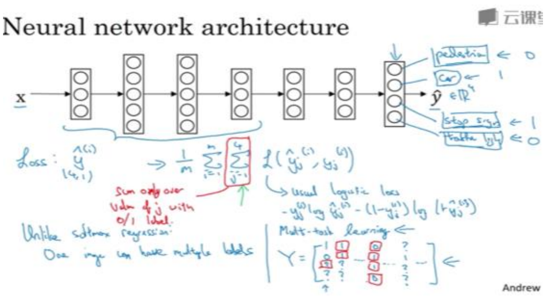
1. Example_autonomous driving

input image--->multiple output: there is car, pedestrian, traffic light..
Labeled output Y and predicted output Y^: dimension (n, m) n: output number for each example; m:: example number
(note: for classfication problem: each calssification category is one classfier-->output have n classfire for each example)

train aglorithm to predict matrix Y^.



2. Neural network architecture



>1. output: multi output-multi units (classfier) in output layer.
>2. Loss for one example: sum loss of each output unit over all units/dimension,:
each output unite loss , use logistic loss-(as each units now is a classfier-there are this class in input or not)
>3. Loss for all example: average loss sum of singal example over all examples.
>4. differance with softmax:
Softmax assign a single label to single example (different output value differnt possibility, max possiblity is actual label output) ,-cost is y_i* log (y^_i) (i-ith output unit)
While multi-taks have multi labeles to single example (for each input, go through all output classes: check if there are theses classes in input)

Multi task learning: start off simutanuesly trying to have one neural network do sevral things at the same time.-output have high dimension (as long as output is high dimension-loss function summarize in all dimension-->multi-task.)
function loss: sum loss in all dimension, then average on all training sample.

(note: similar with word embedding skim-gram model: k negative : output is 'vocabl size' classifier.

2.1 Mulit task learning advantage:

Alternative for mulit-task: trained several neural network seperated for each class.

Mulit task learning advantage:
>1. if low level features in neural network could shared in the multi tasks, -->trained one neural network to do multi tasks result in better performance.
(note:
note: if classfires for multi tasks need same low level feature-->using one neural network to capture these low level feature and shared in different classfire :
>>1.1 is more time efficient (if training set is same for multi task network and seperated singal task network: different task have same training set)
>>1.2: better performance (if training set for multi task is the sum of all training set for signal task network: different task has different training set):
as now training set is much bigger for each classfier and cover all seperate signal task's training set/data feature)

> 2.Multi-task neural network also works better, when some of training set only have some of the objets(lableds): labels missed in training data: e.g. 1 pic have 1or 2 , or unclear/missed label.-->in this case could still do learning on these training set.

in this case, still could train neural network on these training set, while loss function: sum in dimension (classes) then will only sum over classes that have label-labeled output 1 or 0, omit loss of the class that has no label.
(note: how could this be advantage?--->larger training set, more robust to these examples without lable-will not be used in backprop.
in this case-->do learning on only labeled classes of these training set;
while for seperated classfier, also learn on only labeled classed of these training set; if no label for one classfier of some training set, then just omit these training set on that classfire, but will continue used for other classfier which class these example is labeled and this classfier also continue with examples that labeled this class)

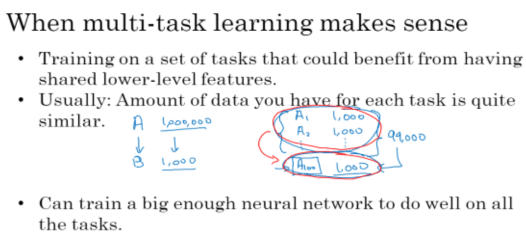
3. Multi task learning make sense when:
>1. Training on a set of tasks that could benifit from having shared low-level features.
e.g in autonomous driving, predict if there is pedestrian, car, ..from one image, these tasks should have similiar features-as these are all features of roads
(note: condition to use multi task learning)

>2. Less of a hard and fast rule, so is not always true:
usuallly: Amount of data you have for each task is similar.

In task transfer, new task have much smaller data B than origianl task A. all the knowledge learned from task A could really help augment the much smaller data set for task B.
In multi task: e.g 100 task and 1000 example for each task. if do the 100-task on isolation , then only have 1000 examples to train, performance not well. but by training on other 99 other tasks-99,000 examples, could give a lot of knowledge to augment this 100-th task.
symmetrically, every one of the other 99 tasks data can provide some data/knowledge that help every one of the other tasks in the list of 100 tasks.

if focus on any one task, for that to get a big boost from multi-task learning , the other tasks in aggregate need to have quite a lot more data than for that one task. And so one way to satisfy that is if a lot of tasks like we have in this example on the right, and if the amount of data you have in each task is quite similar.
But the key really is that if you already have 1,000 examples for 1 task, then for all of the other tasks you better have a lot more than 1,000 examples if those other task are meant to help you do better on this final task.

>3. make better sense when can train a big enough neural network to do well on all tasks.
(note: condition:
can make the network for multi task bigger, trained with bigger training set-->
combined bigger traiing set feature /info provided is bigger able to train more complicated



3. Multi task learning make sense when:

Note:
1. alternative for multi task learning: seperate neural network for each task.
2. it has been found only in one case where mulit task learning not better than seperated neural network, -->because multi task neural network not big enough.
But if you can train a big enough neural network, then multi-task learning certainly should not or should very rarely hurt performance. And hopefully it will actually help performance compared to if you were training neural networks to do these different tasks in isolation.
3. multi task less frequently used than task transfer, while yet might use more in computer vision-object detection.

Summarize:

multi-task learning enables you to train one neural network to do many tasks and this can give you better performance than if you were to do the tasks in isolation.

Now one note of caution, in practice transfer learning is used much more often than multi-task learning. A lot of machine learning problems have a relatively small data set, then transfer learning can really help. Where if you find a related problem but you have a much bigger data set, you can train in your neural network from there and then transfer it to the problem where we have very low data.

And maybe the one exception is computer vision object detection: training a neural network to detect lots of different objects. And that works better than training separate neural networks and detecting the visual objects.

3.1-9 end to end deep learning

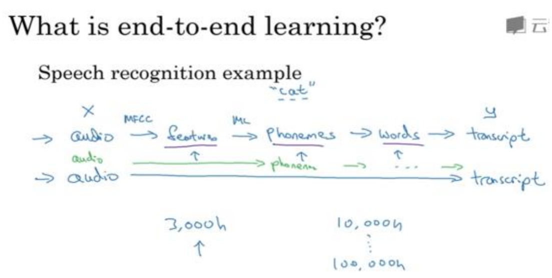
there have been some data processing systems, or learning systems that require multiple stages of processing. And what end-to-end deep learning does, is it can take all those multiple stages, and replace it usually with just a single neural network.

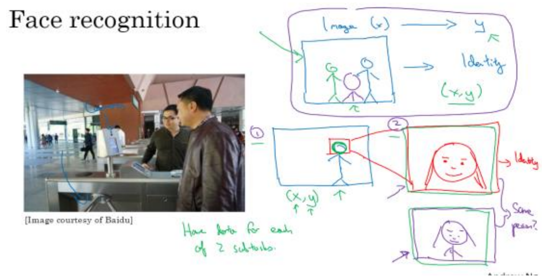
1. Example_speech recognition
input x: audio,
outputY: trasnscript tex.

> tradition way-pipeline stages: first recognize audito feature------>use machine learning to find phonemes------>words---------->output: transcript.
End to end learning: train huge neura network, with input------------> and then directly output.: taking training set, learn function from input to output.
bypassing a lot of these intermediate steps.

end-to-end learning challenge: need a lot of data before algorithm works well.
e.g when data is small-3000h audio, then traditon way works very well/even better, but when have 10,000-100,000h data, end-to-end approach suddenly starts to work really well.

2. When to use end to end
>1. If have a smaller data set, the more traditional pipeline approach actually works just as well. Often works even better.
>2. And you need a large data set before the end-to-end approach really shines.
>3. If have a medium amount of data, then there are also intermediate approaches: where maybe you input audio and bypass the features and just learn to output the phonemes of the neural network, and then at some other stages as well. So this will be a step toward end-to-end learning, but not all the way there.





3. Example_Face recognition:

Purpose:
A camera looks at the person approaching the gate, and if it recognizes the person then the turnstile automatically lets them through.

Algorithm buidling:
method1: input----->directly output identity.
Not the best approach, as person could occure in camara in different distance and direciton-->person position and scale in the image changes.

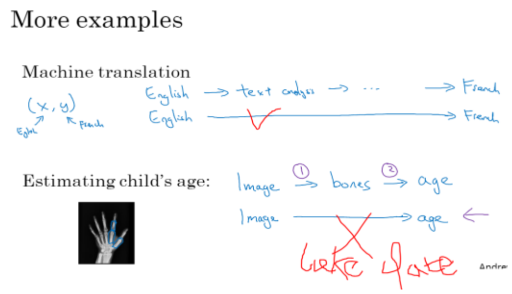
Method2: multi-step approach:
>1. first, algorithm detect person face postion
>2. zoom in to that part of the image, and crop so person's face is centered.
>3. feed crop image to neural network to estimate the person's identity.
(take two images to compare if are same person)

Note:
Instead of trying to learn everything on one step, by breaking this problem down into two simpler steps, first is figure out where is the face. And second, is look at the face and figure out who this actually is.
This second approach allows the learning algorithm or really two learning algorithms to solve two much simpler tasks and results in overall better performance.

Two step approach works better, because:
>1. Each of the two problems is actually much simpler.
>2. Have a lot of data for each of the two sub-tasks.
In particular, there is a lot of data you can obtain for face detection, for task one over here, . And then separately, there's a lot of data for task two as well: okay, leading companies have hundreds of millions of pictures of people's faces.

But in contrast, if try to learn everything at the same time, there is much less data (X, Y) Where X is image like this taken from the turnstile, and Y is the identity of the person.

So because you don't have enough data to solve this end-to-end learning problem, but you do have enough data to solve sub-problems one and two, in practice, breaking this down to two sub-problems results in better performance than a pure end-to-end deep learning approach.



4. More examples

Machine translation: end to end learning well due to large data

Estimating child's age:
Image----->segment out bones--->age: works well
Image------------------------------------>age: does not work well due to lack engough data

Summary:

So an end-to-end deep learning can work really well and it can really simplify the system and not require you to build so many hand-designed individual components. But also it doesn't always work.(only due to lack data)

3.1-10 whether to use end-end learning

1. Pros and cons of end-to-end deep learning

1.1. Pros:

>1. let data speek:

>>>1. if have enough X,Y data then whatever is the most appropriate function mapping from X to Y, if train a big enough neural network, hopefully the neural network will figure it out.

>>>2. And by having a pure machine learning approach, neural network learning input from X to Y may be more able to capture whatever statistics are in the data, rather than being forced to reflect human preconceptions (-->most appropriate function).

let algorithm learn whatever it want to learn than force to learn some 'features', overall performance might be better.
(learn 'best' features and mapping functions)

e.g speech recognation, phonemes are an artifact created by human linguists in traditional method.
while end-to end let algorithm learn whatever it wants to learn-overall performance might be better.

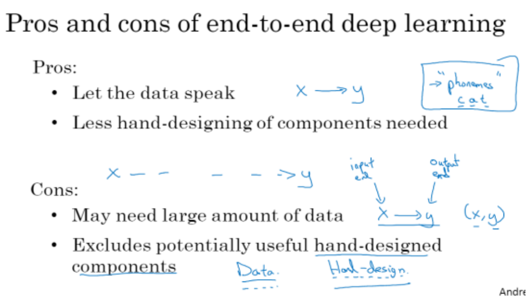
>2.Less hand-designed components needed: simply design work flow.
Don't need to spend a lot of time in hand designing features, hand designing these intermediate representations.

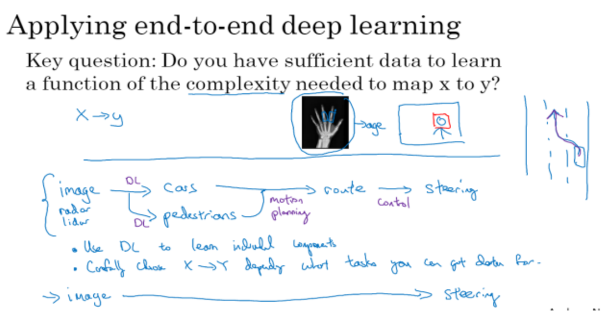
1.2. Cons:
>1. May need large data:

To learn this X to Y mapping directly, might need a lot of data of X, Y : X is the input end of the end-to-end learning and Y is the output end.
So need all the data X Y with both the input end and the output end, in order to train these systems.
And this is why we call it end-to-end learning value as well because you're learning a direct mapping from one end of the system all the way to the other end of the system.: X   (Input) ------------------->Y(Output)

>2. excluded potentially useful hand-designed components.

If don't have a lot of data, then learning algorithm doesn't have that much insight it can gain from data . And so hand designing a component can really be a way for you to inject manual knowledge into the algorithm, and that's not always a bad thing.





1.3 Summarize:
A learning algorithm has two main sources of knowledge: data and whatever you hand design, be it components, or features, or other things.

>1. When have a ton of data it's less important to hand design things;
>2. when you don't have much data, then having a carefully hand-designed system can actually allow humans to inject a lot of knowledge about the problem into an algorithm deck and that should be very helpful if well designed.

So one of the downsides of end-to-end deep learning is that it excludes potentially useful hand-designed components. And hand-designed components could be very helpful if well designed. They could also be harmful if it really limits your performance, such as if you force an algorithm to think in phonemes when maybe it could have discovered a better representation by itself.

So hand-designed components is kind of a double edged sword that could hurt or help, but it does tend to help more when you're training on a small training set.

2. Applying end-to-end deep learning

Whether to use end-to-end learnign:
Key question: If have sufficient data to learn a function of the complexity needed to map labeld input X and labeld output Y.
E.g: auto drive: input pic. Around-->steering angle.
Pure end to end not appliable as not engough data.
Multi-step method: used instead.
Image caputured------>detected cars, pedestrians, ..--------->Plan route: path---------->Steering