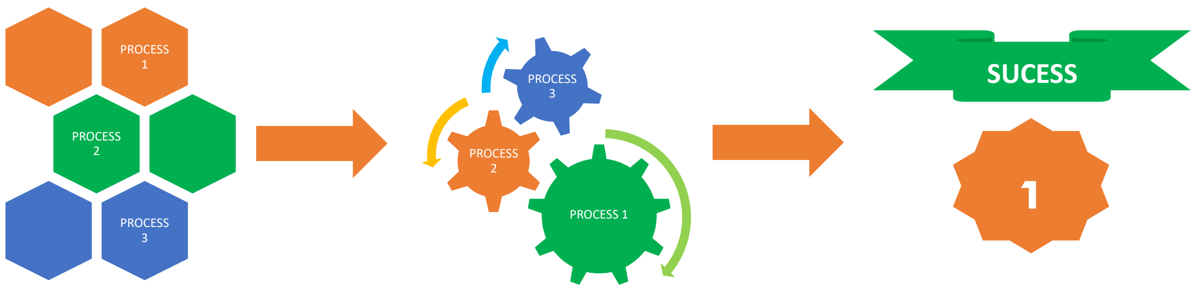
DEEP LEARNING SPECIALIZATION

COURSERA-DEEPLEARNING.AI-ANDREW NG

These are notes taken from [Deep Learning Specialization]() by deeplearning.ai in [Coursera](). All the screenshots shown here are taken from the course videos. I would advise you to first watch the videos of the course in order to completely understand the notes.

COURSE 3

Structuring Machine Learning Projects



About this Course (By Andrew Ng)

You will learn how to build a successful machine learning project. If you aspire to be a technical leader in AI, and know how to set direction for your team's work, this course will show you how.

Much of this content has never been taught elsewhere, and is drawn from my experience building and shipping many deep learning products. This course also has two "flight simulators" that let you practice decision-making as a machine learning project leader. This provides "industry experience" that you might otherwise get only after years of ML work experience.

After 2 weeks, you will:

- Understand how to diagnose errors in a machine learning system, and
- Be able to prioritize the most promising directions for reducing error
- Understand complex ML settings, such as mismatched training/test sets, and comparing to and/or surpassing human-level performance
- Know how to apply end-to-end learning, transfer learning, and multi-task learning

This is the third course in the Deep Learning Specialization.

WEEK 1 – ML Strategy (1)

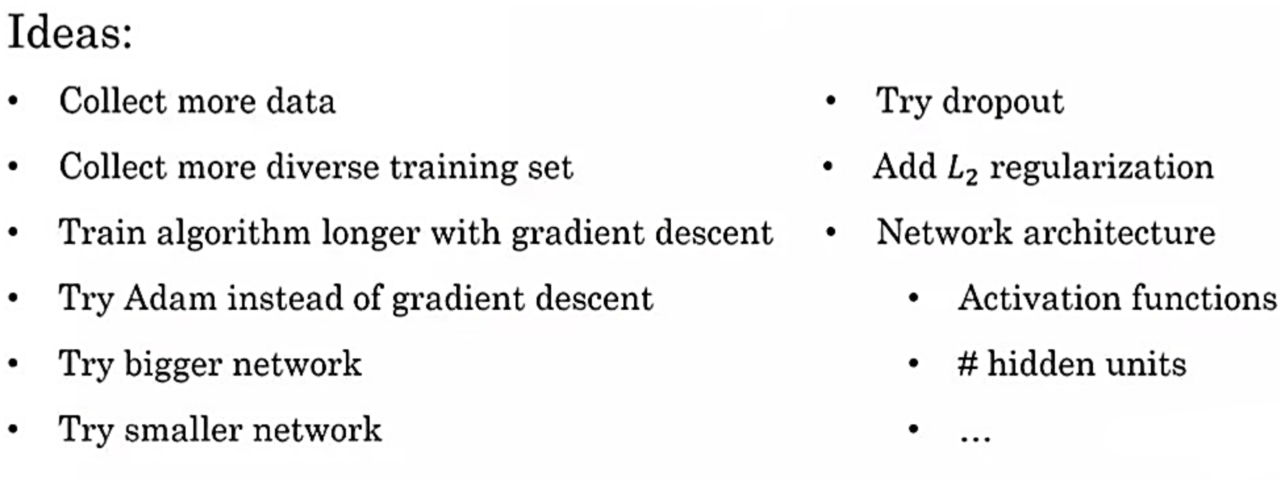
Key Concepts

- Understand why Machine Learning strategy is important
- Apply satisficing and optimizing metrics to set up your goal for ML projects
- Choose a correct train/dev/test split of your dataset
- Understand how to define human-level performance
- Use human-level perform to define your key priorities in ML projects
- Take the correct ML Strategic decision based on observations of performances and dataset

Why ML Strategy

Motivating example

Let's say we are working on a cat image classifier and we got our model to perform at 90% accuracy. We would like to improve our model's performance and these are the areas that we think we should improve on:



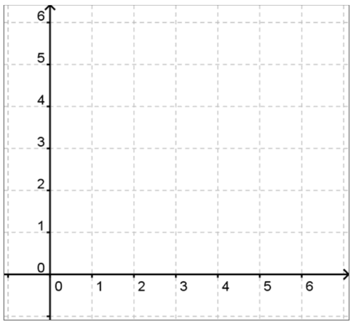
Ideas:

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

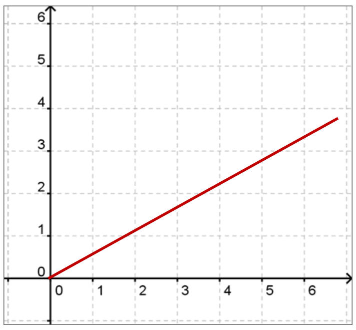
These are the ideas that we want to try out in order to improve our modes performance. But one problem that we encounter if we do not have a clear strategy about which direction to go in is the possibility of spending days or weeks or even months behind one idea and trying to improve our model based on that. It might end up as success but if it doesn't then it would cost you lot of time. That's where a good ML Strategy plays a big role.

Orthogonalization

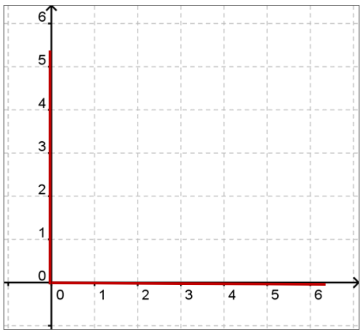
Imagine we have two function of a TV to tune. Let's represent then in two axis x and y



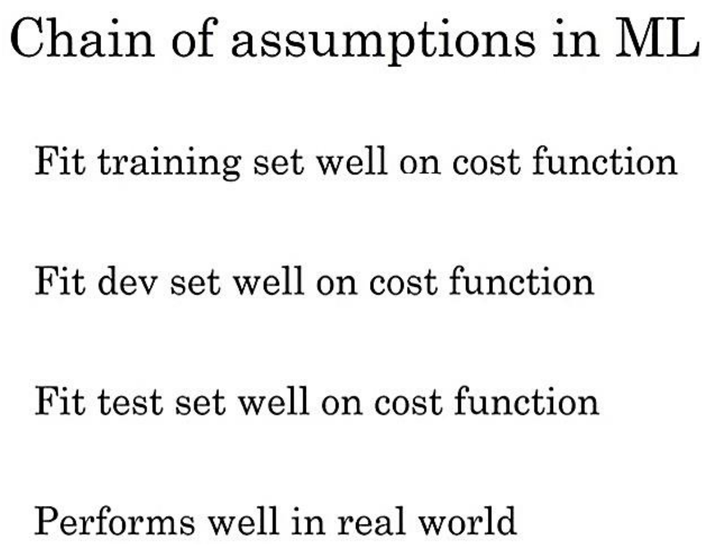
Now let's imagine we have been tuning the functions of TV in a way that it would affect 0.3 times x value and 0.7 times y value. Which might be represented like shown below.



We can see that here we are unable to tune the two things separately. Orthogonalization means the idea of tuning all the functions in a way that tuning one of them won't affect the other one. Which means in orthogonalization if we tune x then it would change values on x axis only and if we tune y it would change the values on y axis only.



This can be applied in ML as well. Imagine that we are training an ML model. We would have the following assumption to determine if our model is performing as per requirement.



Chain of assumptions in ML

Fit training set well on cost function

Fit dev set well on cost function

Fit test set well on cost function

Performs well in real world

Let's imagine each one of these assumptions as a function that we need to tune.

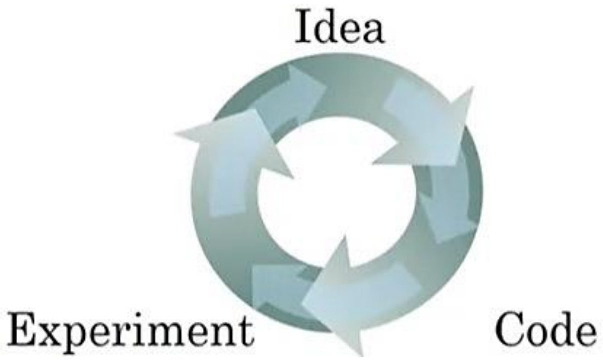
1. If our model is not performing well on the training data then we might want to choose a better architecture with more layers on network or we might want to train for more times or might want to use optimizer like Adam if we are not using it already.

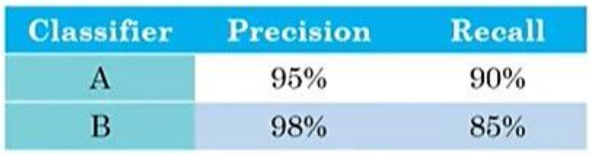
2. If our model is working well on training set but poor on dev set then we might want to do regularization or collect more data for training set.
3. If our model is working well on both training and dev set but not on test set then we might want to give more data to dev set.
4. If our model works well in all these sets but not on real world problems then we might want to consider changing our cost function which allow us to train the right parameters. Or we can change the distribution of our dataset to more realistic ones.

If we use early stopping instead of regularization then it is affecting both the dev set and training set so we might use the property of orthogonalization.

Using single number evaluation metrics

Let's say we are building an app that classifies cat pictures. We might use more than one classifier with changes in hyperparameters or training data to train our model. In this case one way to know which classifier does best is to find the precision and recall.





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

Precision: Out of all the examples classified as cats how many are actually cats.

Recall: Out of all the true cats how many we predicted correctly.

AP - Actual +ve , PP - Predicted +ve

AN - Actual -ve , PN - predicted -ve

if PP - AP = 0 - All are predicted correctly

PP - AP > 0 - There are False +ves

PP - AP < 0 - There are False -ves

Predicted True +ve (PTP) = P.P - F P

$$\text{Precision (P)}: \frac{PTP}{PP} = \frac{PTP}{PTP+FP} \Big\} \text{FP- False +ve}$$

$$\text{Recall (R)}: \frac{PTP}{AP} = \frac{PTP}{PTP+FN} \Big\} \text{FN- False-ve}$$

$$\text{F1 Score} = \frac{2}{\frac{1}{P}+\frac{1}{R}} \Big\{ \text{Harmonic Mean}$$

When we have more than two classifiers it would be difficult to use precision and recall to identify the best one. Instead it would be good to have a single evaluation metric like the F1 score which takes the average of both precision and recall.

We can see for F1 score we have used harmonic mean instead of arithmetic mean.

The harmonic mean H of the positive real numbers $x_1, x_2, \ldots, x_n$ is defined to be

$$H = \frac{n}{\frac{1}{x_1}+\frac{1}{x_2}+\cdots+\frac{1}{x_n}} = \frac{n}{\sum_{i=1}^{n}\frac{1}{x_i}} = \left(\frac{\sum_{i=1}^{n} x_i^{-1}}{n}\right)^{-1}.$$

The harmonic mean is the appropriate mean if the data is comprised of rates.

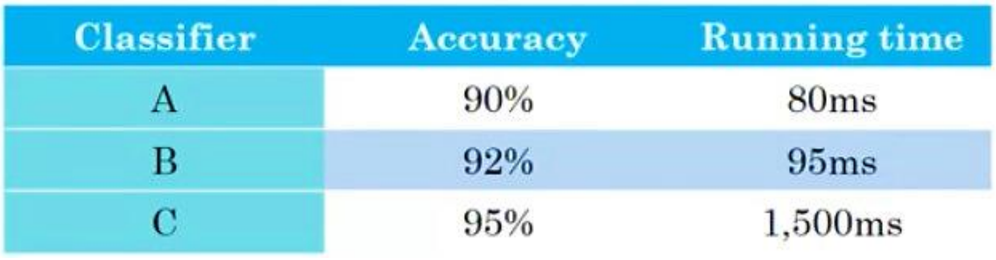
Let’s take another example where you want to know the error in predicting cats by using the app in different countries like given below.

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

Here we can see that if we are comparing different classes performance in different countries it would be difficult to understand which one is better as there can be lot of classes and lots of countries. So in this case to we can take average performance of a particular class across all the countries and use that to choose which one is the best class.

Satisficing & Optimizing metrics

Let's say we are evaluating our model's performance based on two metrics, its accuracy which can be an F1 score and its runtime as shown below.



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

Now we know that accuracy is something that we cannot compromise on and it's something that we should optimize to get the maximum value for. So, we would choose accuracy as the optimizing metric here and would put a limit to the value of running time let's say <=100ms. And we call that a satisficing metric because it is having lesser importance than the accuracy and anything below 100ms run time even if its 10 or 50 it would be hardly a problem.

In general, if there are N metrics to look at while choosing the best classifier it is always good to put 1 or few as the optimizing metric one that you have to definitely optimize and others as satisficing metrics would speed up the process of choosing the best classifier.

Train/Dev/Test set distributions

When we are training our model and testing it on dev set and tuning some evaluation metrics, we should consider these things:

Choose a dev set and test set to reflect data you

expect to get in the future and consider important

to do well on.

And also choose the test and dev set from the same distribution of data.

Choosing size of Train/Dev/Test set

***Check page 4 of course 2 notes**

https://www.linkedin.com/posts/deepak-jose-60953b186_deep-learning-specialization-course-activity-6695337510232559616-MCc4

When to change Dev/Test set

Let's say we are making an application that classifies images to cats and non-cat. We have two classifiers and we calculated their error percentage and found that Algorithm A does best as it has less error. But when we tested both these on real user images, we found out that Algorithm B does better than A even when having more error percentage. This could be due to the fact that we trained our model using the images available from the internet which are of really high quality but the images that the users upload might be of lower quality. In this case we should consider changing our Dev/Test set.

Another scenario is when you have two classifiers which classifies cat images and classifier A has less error compared to B but when we tested both on users, we found that classifier B is letting in some pornographic images that is totally unacceptable. So, in this case even when classifier B has higher error, we would choose this as it won't misclassify a cat image with a pornographic one. So, in this case we should change our evaluation metric which allowed the pornographic image to get in.

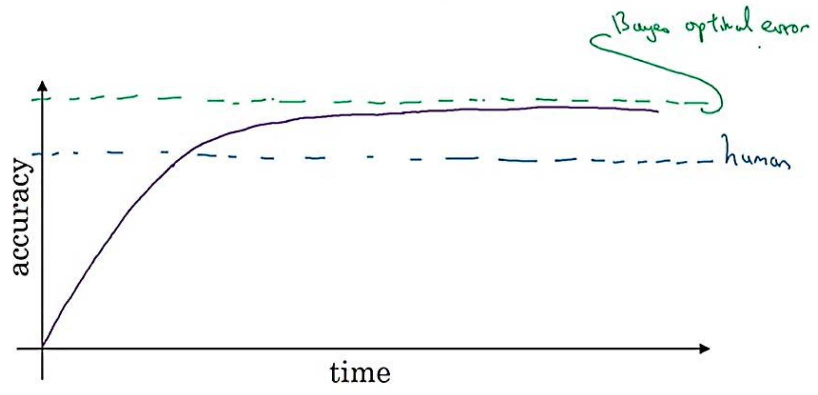
We can do this by adding a limit to our error function like this:

Error: $\frac{1}{\sum_i \omega^{(i)}} \sum \omega^{(i)} L(y_{pred}^{(i)} \neq y^{(i)})$

Where $\omega^{(i)}$ = 1 if $x^{(i)}$ is non-porn and 10 if it is porn.

In this case the cost would be high if the image is porn.

Why comparing to human level performance?



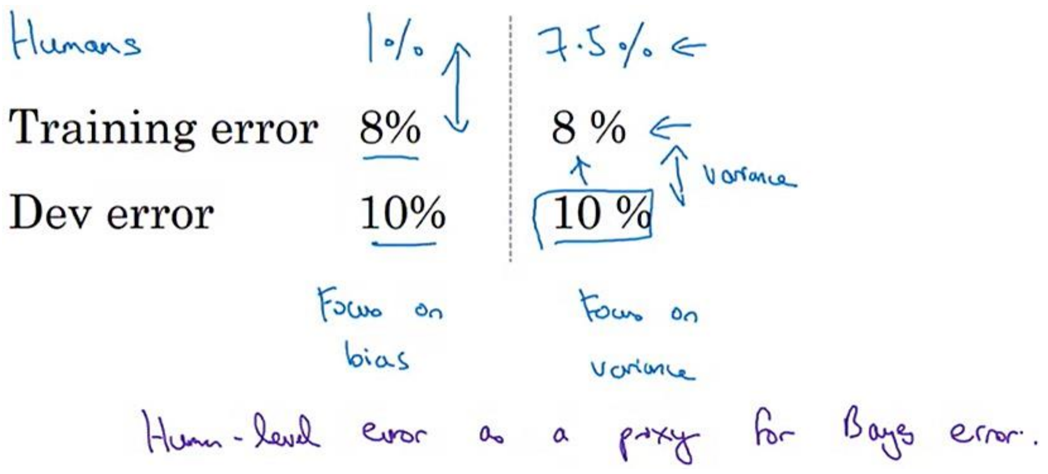
When training our model there is a minimum error that we can never improve upon which is called the Bayes optimal error. This is slightly above the human level performance. The reason why we compare our models to human level performance is because after achieving human level performance our model can only improve a slight bit towards the Bayes optimal error. But before human level performance we have lots of tools to improve our model to get closer to human level.

Humans are quite good at a lot of tasks. So long as ML is worse than humans, you 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

Imagine we are doing a cat classification and we got the following training and development errors.

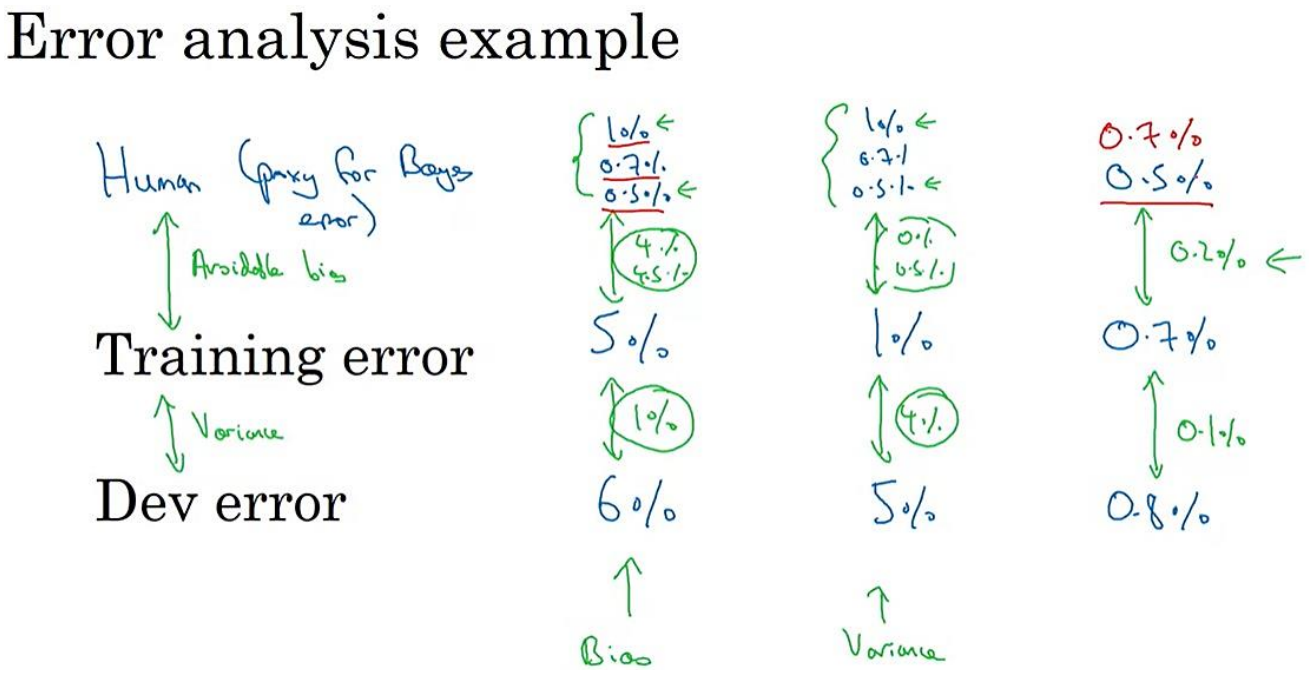


Humans	1%	7.5%
Training error	8%	8%
Dev error	10%	$\boxed{10\ \%}$

variance
Focus on bias
Focus on variance

Human-level error as a proxy for Bayes error.

Here we are considering human level performance as approximately equal to Bayes error. In the first case we have human error of 1%. We can clearly say tat in this case we need to improve our training to reduce the bias of 7%. In the second case let's say our human error is about 7.5%, here we can see that there is merely any room for improvement. But we can still improve the performance of our development set by reducing error by 2%. So, in first case we can focus on bias and second case we should focus in reducing variance.

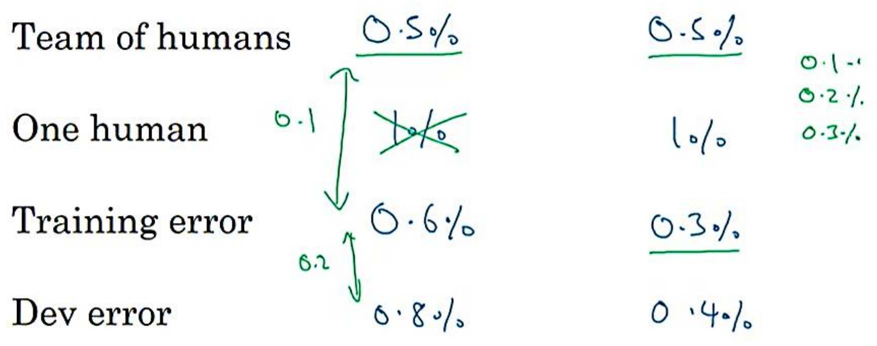


In this error analysis example, we are comparing 3 situations to identify what approach should we do. In the first case our training error is 5%, dev error is 6% but the human error which is a proxy of Bayes error is 1,0.7 or 0.5% therefor its clearly a case of bias and we should work to reduce it.

In the second case we can see that training error is 1% which is closer to human error but dev error is 5% which is 4% higher than training error. Therefor here we should focus on reducing variance.

In the third case we have training error 0.7 and dev error 0.8% here if we take 0.5% as the human error then we can say that there is room for improvement and we should focus on reducing avoidable bias. But in this case if we take 0.7 as the Bayes error then we can see that avoidable bias is 0 and we should reduce the 0.1 percent variance.

Surpassing human level performance



Team of humans	0.5%	0.5%	0.1%
One human	~~1%~~	1%	0.2%
Training error	0.6%	0.3%	0.3%
Dev error	0.8%	0.4%	

Here in the first example we have team of humans getting 0.5%error and our training set getting 0.6% error and dev set 0.6% error. Here we have an almost clear idea to improve our model to reduce variance and maybe reduce the bias of 1%.

But in the second example shown left we have 0.3% error on our training set and 0.4 on dev set which clearly surpasses the error of team of humans. Here it is not clear that whether we should decrease error further. Because once we surpass human level performance then we can't rely on human intuitions to improve our model.

Problems where ML significantly surpasses human-level performance

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

These are some of the areas where ML models can surpass human performances as these tasks rely on data that is structured and computers are far better at understanding structured data than humans. Also in recent times computers were able to surpass humans in tasks like computer vision and speech recognition by a small margin.

Improving Models Performance

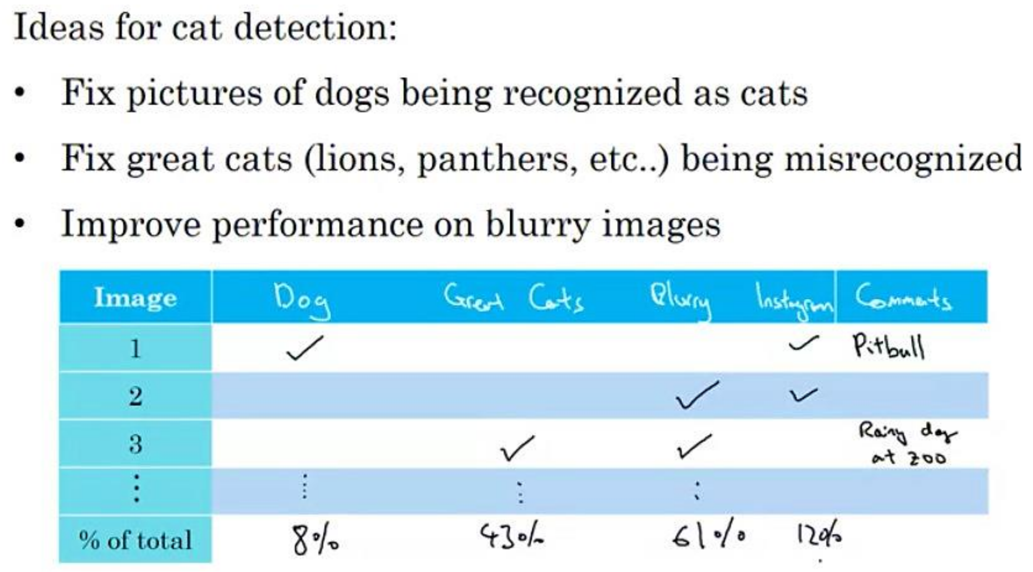
***Reducing bias and variance – see page 5 to 8 of course 2 notes**

https://www.linkedin.com/posts/deepak-jose-60953b186_deep-learning-specialization-course-activity-6695337510232559616-MCc4

WEEK 2 - ML Strategy (2)

Carrying out error analysis

Imagine we are training a cat classifier and your model has got 10% error. You selected 100 of the misclassified images from dev set and analyse it to find which ones are misclassified the most. Before we start analysing, we had few ideas on what to change to improve our model.



Ideas for cat detection:

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

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

Here from the analysis we can see that if we focus on fixing correct recognition of dog images by adding more dog images or so it would only reduce 8% of the wrong classifications. On the other hand, if we improve the detection of great cats more and blurry images too it would be a great improvement for our model.

Cleaning up incorrectly labelled data

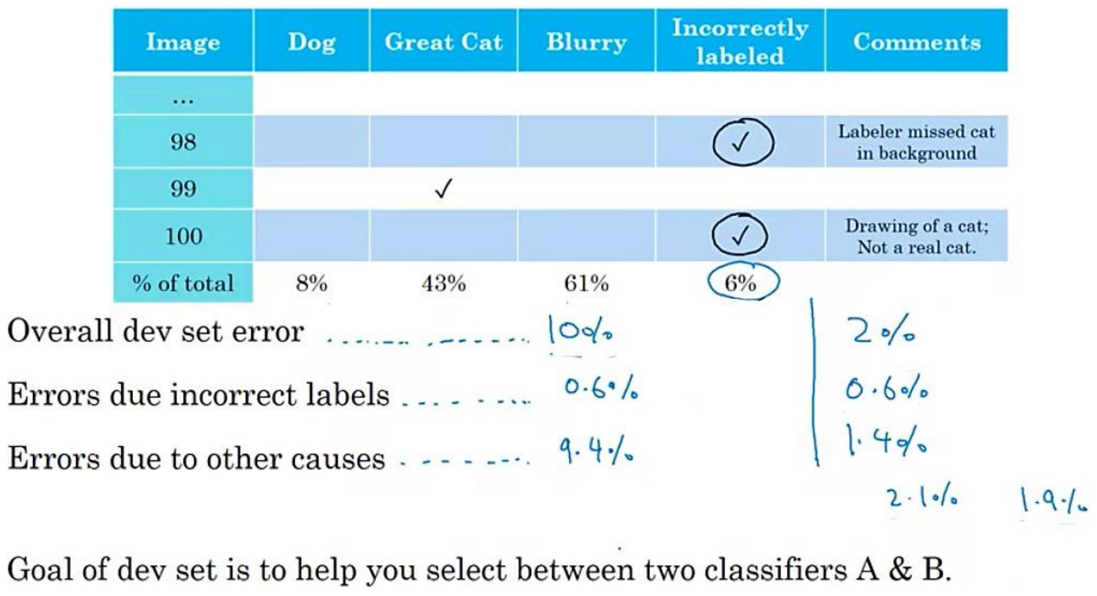


Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
...					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

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

2.1% 1.9%

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

Let's say our model has 10% error and we found out that some of the errors are actually due to the model identifying an image correctly as cat whereas we have mislabelled it to be some other animal. So we would again analyse the dev set to get all the misclassified images and analyse which images are misclassified most. In our case only 6% of the error is due to mislabelled data. So, if we try to reduce that we would only reduce the error to 9.4%.

So, we focussed on improving on other errors and let's say now our model has just 2% error. But still we have 0.6% of that 2% error due to mislabelled data. In this case improving on mislabelled data can decrease our error by 30%.

Tips when correcting dev/test set

- When we correct dev/test set make sure that we do the same process on both the dev and test sets to make the distributions same.
- If you have time then also consider examining the data that you got correct too for finding wrong labels.
- Training set might not need to go through the little changes that you made on test and dev sets.

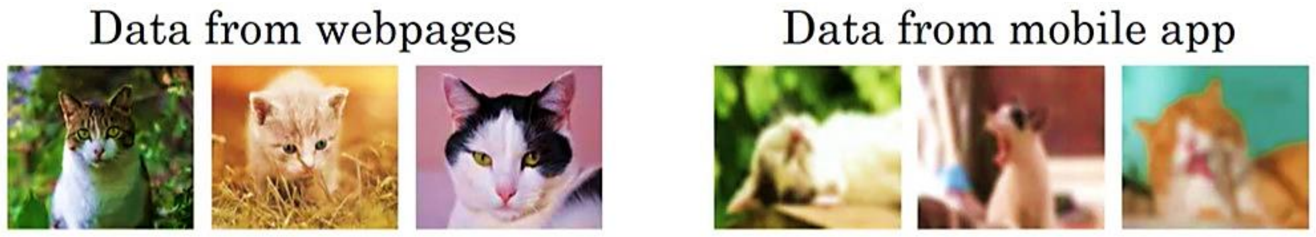
Build your first model quick

One of the guidelines for creating a model is to create something quick first time and then evaluate it using bias variance analysis and other error analysis to find what to do next.

But if you already have worked on a similar project before or if you have a guideline on how to make it perfect like an academic paper or something then doing it perfectly by taking time is okay.

Training and Testing on different distributions

Let's take an example of building a cat classifier mobile app. You have only about 2,00,000 images form the web and about 10,000 images form mobile phones.



Now these two are from different distributions. But you can't train your model using just 10,000 images. So, there are two options you can try:

- Option 1: You can add all the data together and shuffle them and split them into train/test and dev sets. But in this case out of the 2,10,000 images only few would be present in the dev and test sets. So, we won't get an accurate result using this dataset.
- Option 2: So, we would add al the images we collected from we to the training set and then put maybe about 5000 images to the training set and then split the next 5000 and add 2500 in each dev and test sets.

Bias and Variance with mismatched data distributions

Let's imagine we are doing cat classification and our data distributions are different for training and dev/test sets. So, we want to know how to correctly evaluate the errors in dev/test set and training set compared to **human level performance which is ≈ 0.**

Data Set	Avoidable Bias(%)	Variance(%)	Data Mismatch(%)	All three(%)	Dev/Test error < Training error(%)
Training Set	10	1	1	10	7
Training-Dev Set	11	9	1.5	11	10
Dev/Test Set	12	10	10	20	6

*In the last case take human error as 4%

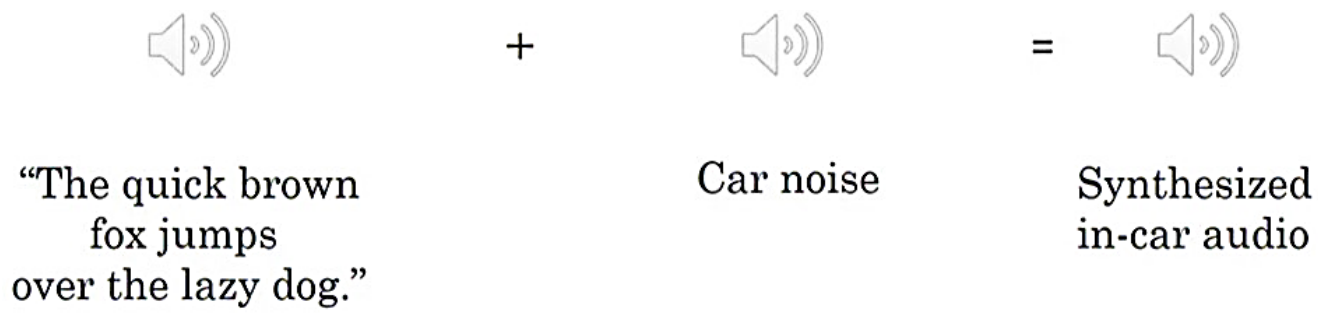
Also, in an additional case if dev error is also less but the test error is more, then it has overfit the dev set and we might have to get more data for the dev set.

How to deal with data mismatch

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

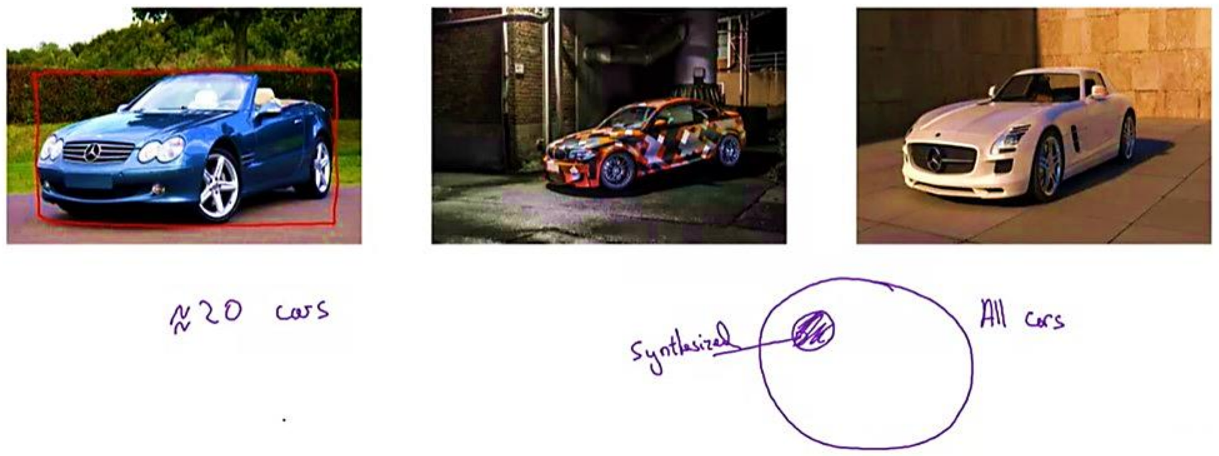
- Make training data more similar; or collect more data similar t dev/test sets

So inured to carry out the second step we need more data that is from the same distribution that we intent to put our application on. One thing we can do is to collect more data from these sources. But That would be a pretty difficult and time-consuming task. An alternate way is to get data by doing artificial data synthesis. Let's take an example of building a speech recognizing mirror for the car. We have almost 10,000 hours of speech data covering almost all the letters and a 1-hour data of car noise inside the car. What we do is we combine these two data to produce 10,000 hours of speech including the car noise.



One problem with this is that we are taking just 1 hour of car noise and putting the same over 10,000 hours of speech. So the car noise that we have is a very small subset of the all possible car noises that can happen in real life.

Car recognition:



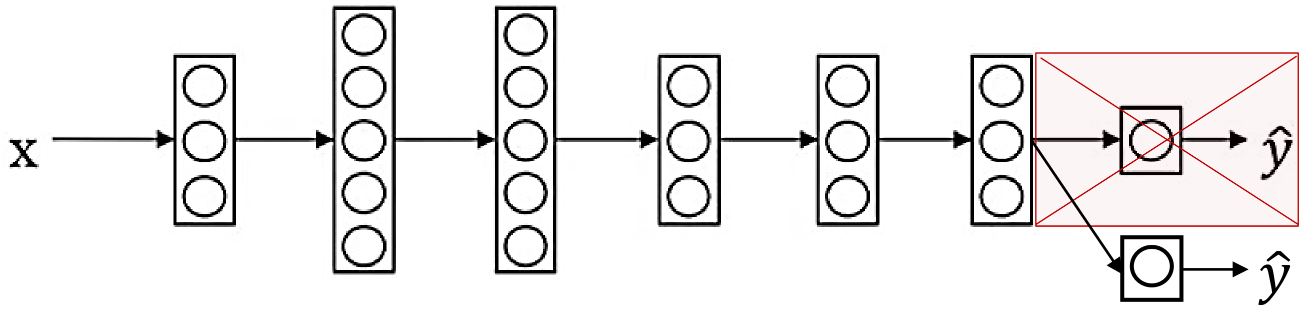
Like in this example of a car recognition if we take synthetic data from say a car game with high graphics which has about 2 different cars. Then If we use that as synthetic data then we are taking a very small subset of a huge different types of cars available in real world.

Transfer Learning

Taking knowledge that neural network gained from one task and use it to help for another task.

How it works?

Let's say we have a model that we trained for image recognition and we want to use that for transfer learning to train a model for radiology diagnosis. So, what we do is we take the model and would replace the output layer including the weight with a new non-trained layer and new weights and would either train this last layer with the new data or we train the entire model again and fine tune it to get the desired output.



- This might not work well when the model that we already have is trained on data less than what we have for the new task.
- If we have more data for the first task and less for the second transfer learning works pretty well.
- It works because in doing similar tasks let's say like our first task is for building a speech recognition system and second one is to wake word detector. So, by using the speech recognition model it I already trained on identifying human voices and letters. So, it would be learning faster on the new task which also would include human voices.

When transfer learning makes sense

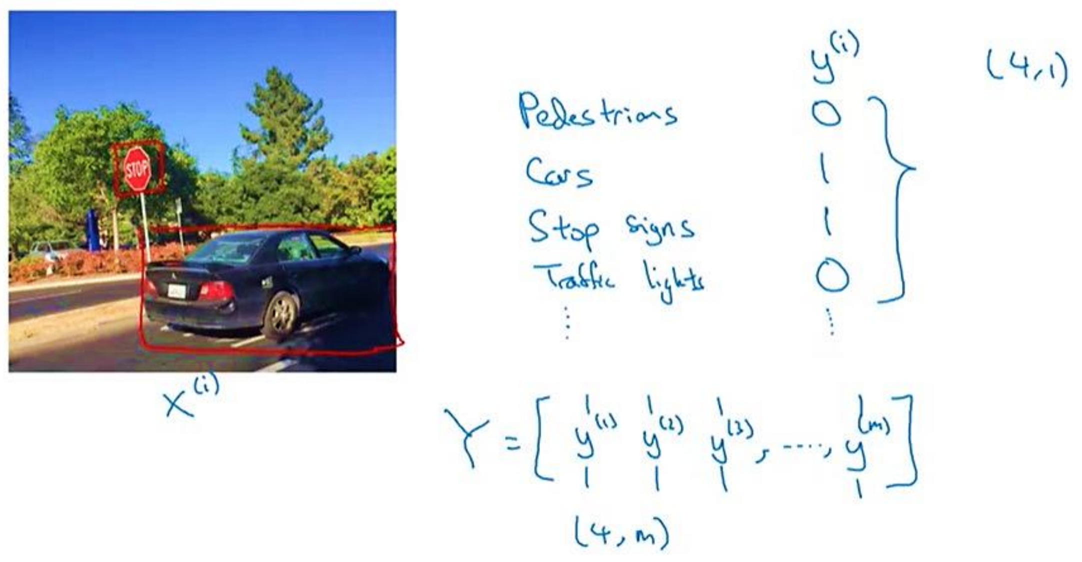
- Task A and B have the same input x. {Same Type of Input}
- You have a lot more data for Task A than Task B.
- Low level features from A could be helpful for learning B.

Multi-Task Learning

In this type of learning we use one neural network to learn to do multiple similar type tasks. For example, in an object detection application we have to detect objects like pedestrian, signs, cars, traffic light etc. all from one image or from one frame. So, in this case we can use multitask learning where we are using the same neural network to label all these in the same image. This type of learning is mostly used in computer vision often than in other areas.

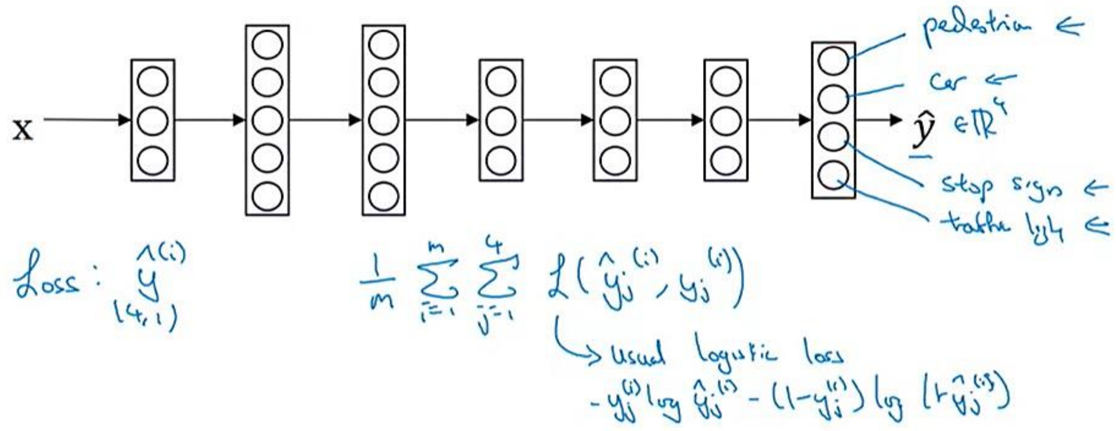
So here instead of giving each of the object's probabilities of occurrence, we give each object 1 if it is found in the image and 0 if it is not found in the image. So unlike in Softmax where we give just one label for one image here, we label all the objects in the same image in one image.

Simplified autonomous driving example



Our output matrix here will be of shape (4, m). 4 features to identify and m examples.

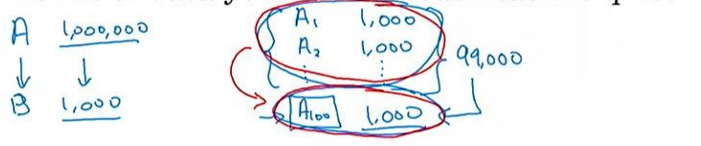
Neural network architecture


x
pedestrian
car
$\hat{y} \in \mathbb{R}^4$
stop sign
traffic light
Loss: $\hat{y}^{(i)}$ $(4,1)$
$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} \mathcal{L}(\hat{y}_j^{(i)}, y_j^{(i)})$
Usual logistic loss
$-y_j^{(i)}\log \hat{y}_j^{(i)} - (1-y_j^{(i)})\log(1-\hat{y}_j^{(i)})$

We can see that the loss function we are using now add up the loss of all 4 feature recognitions.

When multi-task learning makes sense

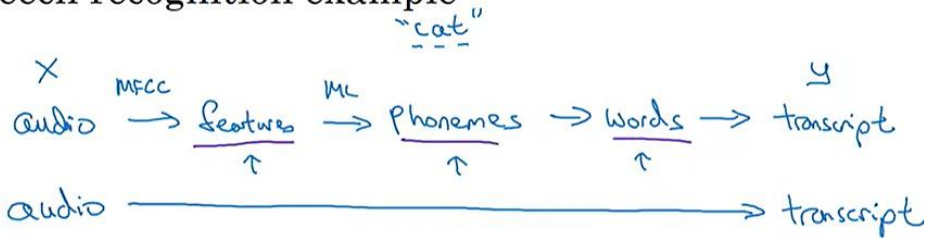
- Training on a set of tasks that could benefit from having shared lower-level features.
- Usually: Amount of data you have for each task is quite similar.



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

End-to-End Deep Learning

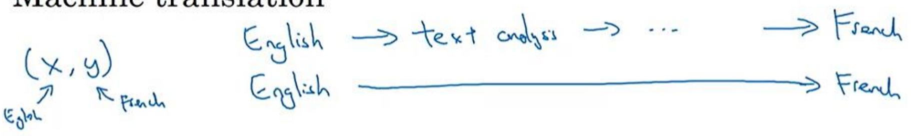
Speech recognition example



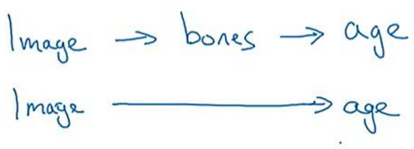
Let's take the example of speech recognition to explain this. Here we have an example of making transcript of an audio. The normal approach is to identify the features first, then phonemes, then words and the full transcript. But if we do It with end to end DL we would directly try to transcript the audio by feeding in previously done examples.

Now we will see two areas to know where we can and where we can't use this mode of learning.

Machine translation

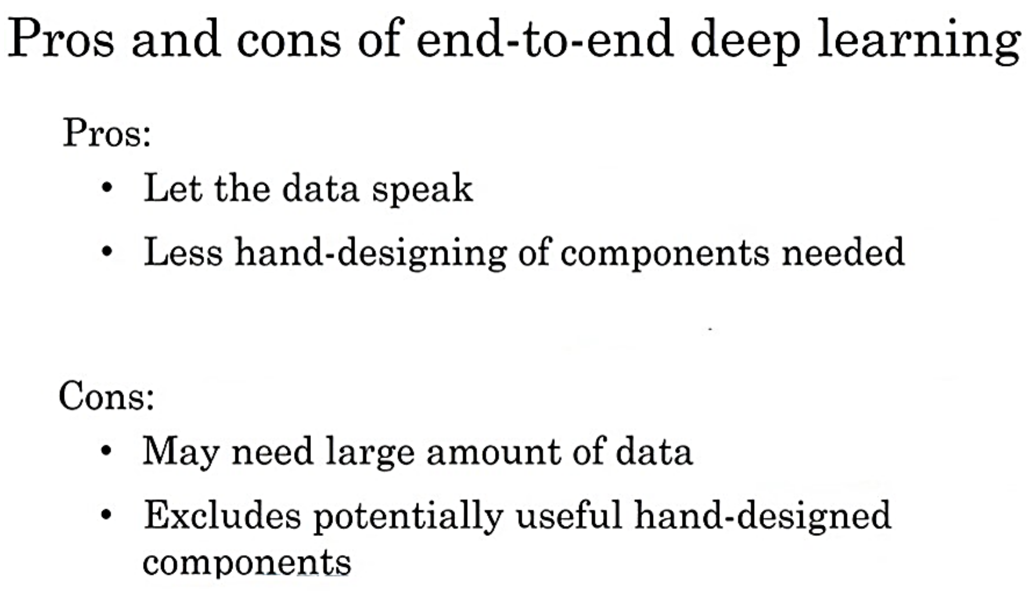


Estimating child's age:



Here in the first case we are trying to translate English to French. In this case we can use end-to-end approach because here we could get large amount of data in which English to French translations are there.

But in the second case we are trying to identify the age of a child from the x-ray image of his hand. So, in this case it would be really difficult to identify the age from looking juts at his hands without measuring the finger lengths and all. So here end-to-end approach won’t be efficient.



Pros and cons of end-to-end deep learning

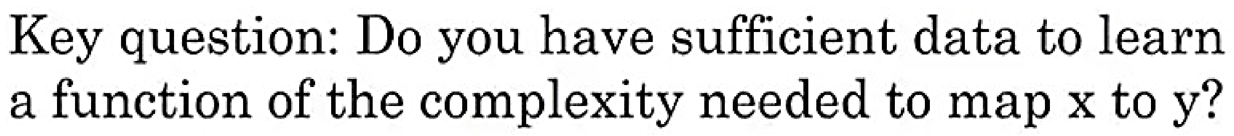
Pros:

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

Cons:

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

Let the data speak means if we let the neural network to match between the input and output itself without giving lot of human inputs in some tasks then it would reduce human caused biases and might be more efficient way of learning.



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