Week#2: ML Strategy (2)

Error Analysis:-

① Carrying out Error Analysis:-

If you are trying to achieve what humans can do and your training algorithm is not yet at the level of humans then manually examining mistakes your algorithm is making can give you insights about what to do next.

Example:- Assume you are training a Cat Classifier and you have achieved **90% accuracy** or **10% error** on your dev set.

→ This is much worse than you're hoping to do

Your team may be, try to analyze those examples which are misclassified and see that it is classifying dogs also as Cats, and proposes that, we should make our algorithm better by employing certain proces for dogs.

→ **Now the qeetion is, Should you focus and go on to start a project focused on dog problem?**

→ Solving this problem could take several month to solve. Is this worth the effort

→ Now here is the error analysis that you could do in order to identify, whether your idea is the one you should employ or not.

Error Analysis:-

- Get ~100 mislabeled dev set examples. (Randomly)
- Count up how many are dogs.

Suppose:- 5% of picked mislabeled examples are dogs.

→ Now that means, even if you enplog dog solving solution after getting done with model training only you'll get 5 more correct images out of 100.

→ In other words if only 5% of your error are dog images then your error will decrease down to 9.5% from 10% even after a lot of hardwork.

→ Now this shows that all the effort you would do is just not worth it.

- This analysis gives you the upper bound OR ceiling for your dog problem solution, should you decide to enplog it.

Now Consider another scenerio

Suppose out of 100 picked randomly examples, 50% are dogs.

→ Now in this case enploying a solution to your dog problem will decrease the error from 10% to 5% and you might think that, this might be worth your effort.

Sometimes you can also evaluate multiple ideas in parallel doing error analysis.

Example:-

Ideas for cat detection:

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

→ during your error analysis you could go like this:-

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

here we are dealing with images which are misrecognized from dev set.

So if 1st misrecognized image is Dog, put a check mark there and may be command and it's type.

→ Now count up the images attributed to certain class of image.
e.g.: 8% are dogs.

→ Sometimes partway through the process you encounter another class of Image messing up your classifier like images with instagram filters.

→ Conclusion of this analysis gives you an estimate that, how worthwhile it might be to work on certain problems.

e.g.: like in this example there are two problems which we encounter the most like "great Cats" and "blurry images" So two different sets of teams can work on these two different problems individually.

→ Sometimes when going through dev set you notice that, some of your examples are mislabeled. Now what you do about that? Let's discuss.

Cleaning up incorrectly labeled data:-

→ Data comprises of input X and output label Y.

Now what if you find that, some of the labels are incorrect, what should you do?

First let's consider the training set:-

Turns out that, DL algorithms are quite robust to random errors in the training set. means that, sometimes people don't pay attention eg they hit the wrong key during putting labels. So, these sort of scenerios are ok!. as long as your dataset is big enough.

→ DL algorithms are less robust to systematic errors.

e.g: if your labaur who puts labels on images consistantly put wrong label on white dogs then it might be a problem, because your algorithm will be trained on a systematic error.

→ Now, how about incorrectly labelled examples on the dev/test set.

In this scenario you'll make another column in the error analysis table of "incorrectly labelled".

→ for example you go through your dev/test set and you find out that, the % of error there is 6%. Now is it worthwhile to go through your data and fix up this incorrectly labelled examples.

★ now if you observe or think that, it would make a significant difference then go ahead and do it. if not then it might not be the best usage of your time.

- **Example:-**

3 numbers to look at and decide if it is worth going in and ameliorate the situation.

- Overall dev set error ---- 10% * following previous example.
- Errors due to incorrect labels 0.6%
- Errors due to other causes:- 9.4%

→ So in this example selling 0.6% error straight will not be the best of choices as compared to 9.4% error.

another example:-

Let's say you bring down your overall error to 2% (dev)

But error due to incorrect labels is still 0.6%

Errors due to other causes now :- 1.4%

Now $\frac{0.6}{2} \times 100 = 30\%$. So such a high fraction of error due to incorrect labels seems worthwhile to ameliorate.

→ **Goal of dev set is to help you classify between 2 classifiers A & B.**

Suppose A has 2.1% dev error and B has 1.9% dev error. So, even then you cannot believe model B because 0.6% of dev set error is due to mislabelled examples, then this is a good reason to go and fix against incorrect labels.

Here are some additional guidelines for Correcting incorrect dev/test set examples:-

- Apply same process to your dev and test sets to make sure that they come from same distributions.
- Consider examining examples your algorithm got right as well as ones it got wrong.

  if we only check wrong examples then it is possible that you end up with more bias estimate of the error of your algorithm.

- Train and dev/test set may now come from slightly different distributions.

Build your first system quickly the iterate :-

→ It is often good to building your model quickly then iterate. Let's see what I mean.

Example:- Let's say you are building a speech recognition system :-

There are actually a lot of things you can do and ~~post~~ prioritize e.g — there are specific techniques to make your system robust to noisy background e.g café noise or Car noise.

- Accented speech
- Far from microphone ⇒ far field noise.
- Young children's speech
- Stuttering ⇒ uh, Ahhh..., um.....

..... There are many other thing too that you could do to improve your algorithm.

Not only this in ML applications there are atleast 50 other techniques / directions to follow which could improve your model.

Now how do you decide which one is the best?

→ What does it mean is that,

- Set up dev/test set and metric.
- Build initial system quickly.
- Use Bias/Variance analysis & Error analysis to prioritize next steps.

Now implementing these steps can be quite messy. So, follow this approach only if you are dealing in the area where you have worked before.

Mismatched training and dev/test set:-

DL work best when given a lot of training data. But this doesn't mean that you can take data from any distribution and shove it into the training set.

Let's say you are building a mobile app

In this app users will upload pictures from mobile and see whether the uploaded image is Cat or not.

So, there are two sources of data that you can get:-

- Data from web → (legible quality)
- Data from mobile app → (Low Quality)

→ Let's say your app don't have a lot of following:- So say, till now you have 10,000 images from mobile app.

→ whereas from webpages you can download as images as you want.

What you care about:- Your final system does well on mobile's pics

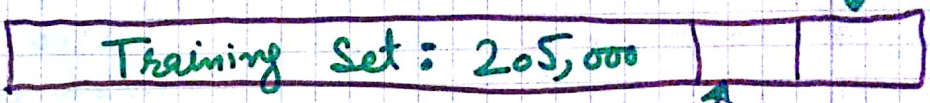
→ mobile app dataset is too low.

→ But on the flipside, 200000 imgs. isn't the distribution you want.

What to do in this situation?

Option 1:-

Merge all data together, shuffle them and seperate training, dev and test set from them.



It has advantages but also disadvantages.

Advantage:-

- All of the data will be from the same distribution.

Disadvantage:-

- In the dev set, a lot of images will be from the webpages' images data rather than what you actually care about (mobile app).

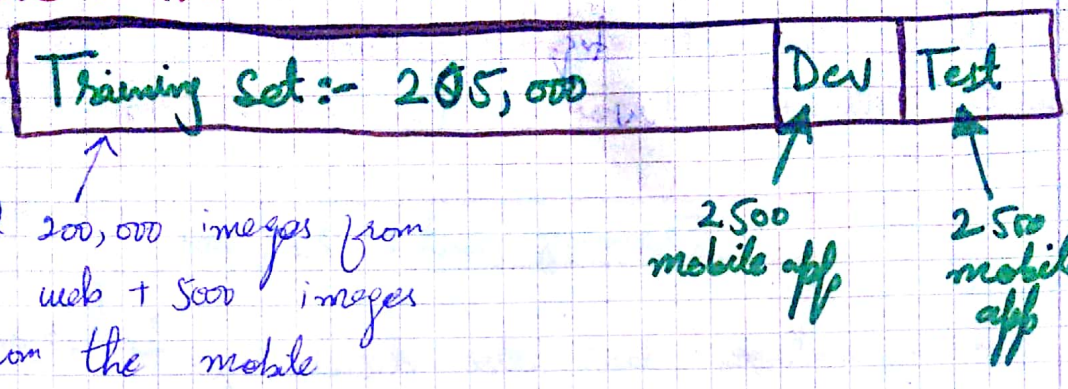
Now $\frac{200,000}{210,000}$ is from webpages so out of 2500 the expected value for mobile app images in dev and test set is 119, which is too less.

→ As we have established that, setting up a dev set is like aiming at a particular target. Now with this distribution, you are implying that most of the time the target should be images from webpages.

So, option 1 should not be followed

bad Idea!

Option #2:



Advantage :- Now you are aiming at the target which you actually want to hit at.

Disadvantage :- Now your training distribution is different from dev and test set distribution.

→ It turns out that, this type of split will get you better performance for the long term.

Let's look at another example :-

Speech recognition example :-

Let's say you are building a speech activated rear-view mirror for cars :-

Now, how can you get data for the training of this product?

→ Let's say, you have worked on a speech recognition system for a long time and you have data from other speech recognition systems, but not for this product.

Here is how you'll classify your data :-

Training Data

- Took all the speech data you have.
- Purchased data.
- Smart speaker Control.
- Voice keyboard.
- .....
- May be you have 500,000 utterances.

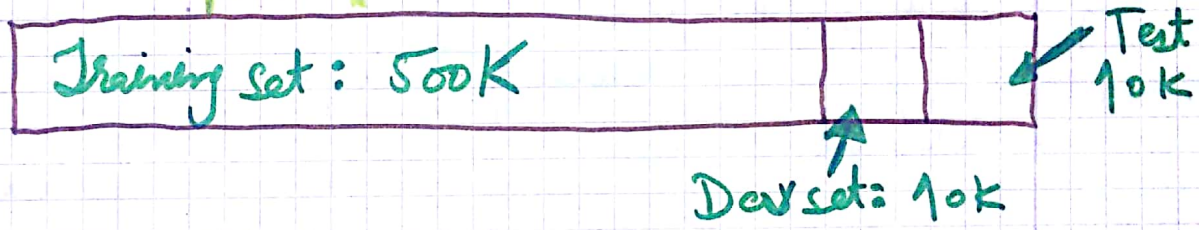
Dev/Test

May be you have much smaller data set that actually come from speech activated rearview mirror. (20,000)

This distribution of data will be very different from the one on the left.

(This is what you care about)

Here's how you would distribute it:



Another way is to halve Dev/test data and put one half in training set and again split the second half into dev and test set evenly.

Bias and Variance with mismatched data distribution:-

The way you analyze bias and variance changes when your training and dev/test set come from different distributions. Let's see how:

→ Let's analyze a Cat classification example and assume humans get near perfect performance for this:

Cat classifier example:-

Assume humans get ≈ 0% error.

So Bayes' optimal error is near Zero here.

→ To carry out error analysis you usually look at the "training error" and "dev error".

Lets say.

	Example
Training error :	1%
Dev Error :	10%

if training data distribution is same as dev set then your algorithm has a variance problem bcz, your model isn't generalizing well on dev set.

In setting where training and dev data come from different distribution, you can no longer safly draw earlier conclusion. In particular May be it is doing fine, it's just that training data was easy may be dev set is much harder So, may be there isn't a variance problem.

→ So, problem with this analysis is that, when you went from training error to dev error, two thing changed. **One:** algorithm saw data in training set but not in the dev set. **Two:** Distribution of data in the dev set is different.

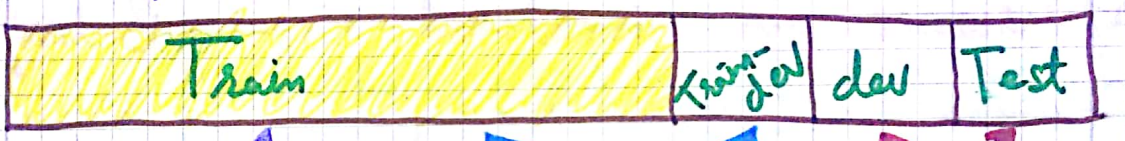
That's why it is harder to distinguish how much of this 9% increment in error is due to the fact that algorithm didn't see the data in the dev set (**Variance problem**) and how much of it is because dev set is different.

In order to tease out these two effects we are gonna define new piece of data which is called

Training-dev set. ⇒ Same distribution as training set, but not used for training.

Here is what I mean.

- Keep previous setting in mind, we will have training set from one distribution and dev/test test from different distribution. Now what will do is randomly shuffle training set and carve out some piece of it and call it train-dev set



Train your network only on this

Same distribution

Same distribution.

Error Analysis:-

- Training error: 1%
  Train-dev Error: 9%
  Dev-Error: 10%

  So what you can conclude from this is that, as Training data and Train-dev data is from same distribution So, you have a **Variance problem.**

So, the model isn't generalizing well.

Let's look at different example:-

- Training Error: 1%
  Train-dev Error: 1.5%
  Dev Error: 10%

  Now as you leap from training data to train-dev data, error doesn't increase that much. But upon going to Dev data Error shoots up to 10%. So this is **Data Mismatch problem.**

Let's go through few more examples:-

Bayes' Error ≈ 0%

Training Error: 10%
Train-Dev Error: 11%
Dev Error: 12%

} Now as you are doing much worse than human level, so you have an avoidable bias problem.

Another Example:-

Training Error: 10%
Train-Dev Error:- 11%
Dev Error:- 20%

} It looks like this example has two issues. Avoidable bias is quite high. Variance is quite small. But Data-mismatch is quite large.

Bias/Variance on mismatched training & dev/test sets

Human Level: 4% → Avoidable Bias
Training set Error: 7% → Variance
Training-Dev set Error: 10% → Data Mismatch
Dev Error 12%
Test Error:- 12% → Degree of over-fitting to dev set.

↳ Same distribution

The only way for 'dev error' and 'Test error' to have a huge gap is because if you somehow manage to overfit your dev set. In that case you need to redevelop by getting more dev data.

↳ Usually this trend goes up.

Let's consider one Example where this trend doesn't go up.

Human Level:	4%
Training set Error:	7%
Training-Dev set Error:	10%
Dev Error:	6%
Test Error:	6%

Sometimes when your dev/test setting is much easier than it can actually go down

There's even more general formulation for this analysis that might be helpful.

General formulation:-

So, let me motivate this using the speech activated rear-view mirror Example.

	General speech Recognition	Rear-view mirror speech recognition
Human Level	"Human Level": 4%	6% ★
Error on Examples N.N trained on	"Training Error": 7%	6% ★
Error on Examples N.N. not trained on	"Training-Dev Error": 10%	"Dev/Test Error": 6%

Avoidable Bias

Variance

Data Mismatch

★ Take rear-view mirror data and ask humans to classify it.

★ Take rear-view mirror data and pass it through network by putting it in the training set.

This data is enough to point you in pretty promising direction

→ Finally, how do you address data mis-match?

Honestly there aren't many systematic ways to address this problem but there are some ways which you can follow to address this issue.

Addressing Data Mismatch:-

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

> To avoide overfitting the test set, technically you should only look at the Dev set for manual error analysis

Example:- If you are building the speech recognition system for rearview mirror app. You might listen to examples in the dev set and try to figure out how your dev set is different than test set.

e.g: . It is noisy e.g car noise
. It is mis-recognizing street numbers

→ when you have insight into these types of errors or about your dev set, what you can do is find ways to make both data somewhat similar.

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

e.g. if noise is the problem then you can
→ Simulate noisy in-car data.
or if the problem of street numbers arises then
→ you can go around and get data of people speaking numbers. and add this to your training set.

→ So, in order to make training data more similar to dev/test data, one of the things you could try is **Artificial Data Synthesis.**

Let's discuss this in terms of development of

Ex.1 Voice recognition rear-view mirror activation.

"The quick brown fox jumps over the lazy dog"

Car noise

Synthesized in car audio

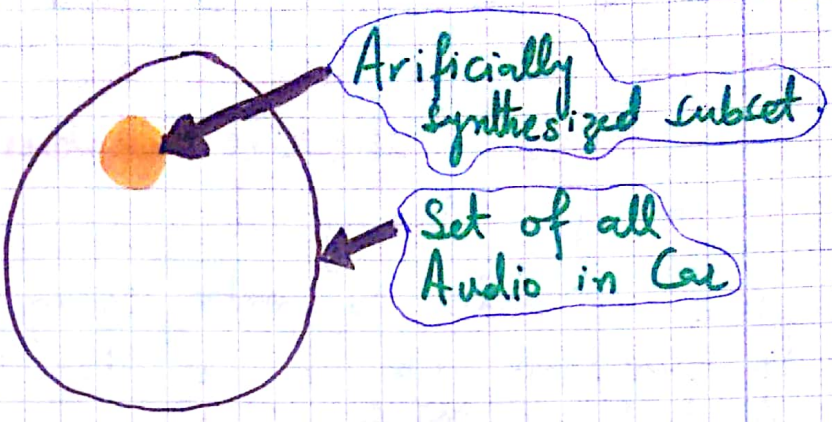
• → Lik this you can produce other effects like "Reverberation"

There is however a note of caution here.

→ Let's say you have 10,000 hrs of data that was recorded against a quite background. Further you have 1 hr of car noise.

→ One approach is repeat this 1 hrs car noise 10,000
• times and mix it with 10,000 hrs of data.

→ To human ear that's fine but there is a chance that your algorithm will overfit to this 1 hrs of data. It like this:-



So, it can happen that, your model might over-fit to this subset.

→ It's not gaurenteed but collecting 10,000 hrs of car noise and mixing it with 10,000 hrs voice data can help you improve your performance.

Ex. 2:- Let's say you are building an autonomous driving system and for that car recognition system is required.

→ One thing that people recommend is that: by utilizing computer graphics, synthesize many cars' images.

→ Above figure also applies here where it is a high probability that, you might over-fit your model to small subset of simulated cars' images because in games and stuff you have limited design for realistic cars, but in reality you have a lot more.

→ Artificial synthesized data does work, but be cautious that, you are not simulating data from the little subset of the whole car noise set.

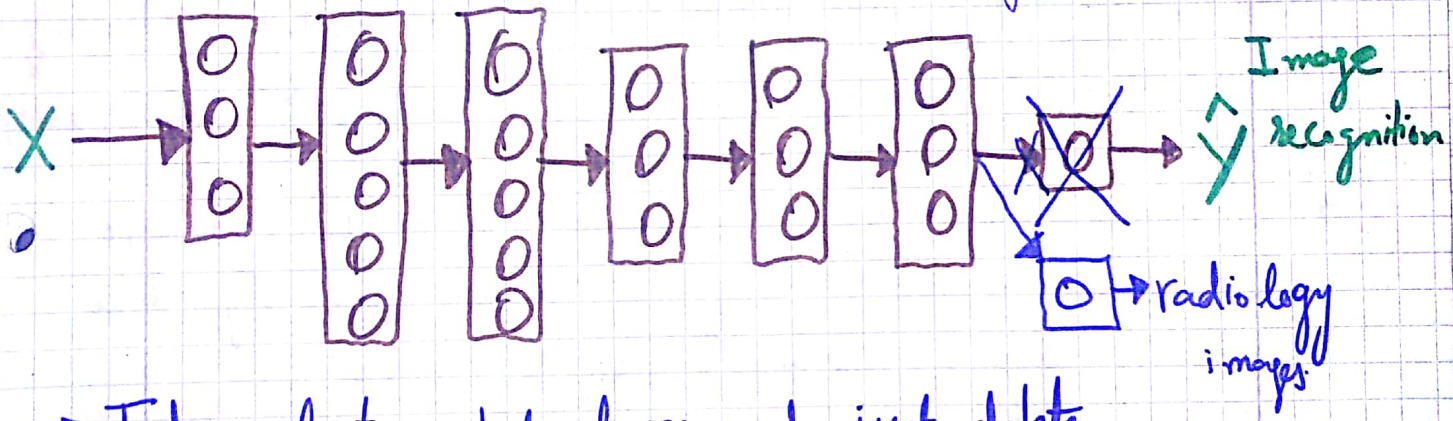
Learn from multiple tasks :-

Transfer Learning :-

Sometimes you can transfer knowledge from one application towards other application.

i.e. using the learned praneters and hyperparameters from one application and use them in other separate application.

→ e.g: training a model which recognises cats and use that model to read X-ray scans



→ Take last output layer and just delete that, delete also the weights speeding into that output layer. Initialize those weights randomly and compute the output.

→ To implement transfer learning, what you do is swap in the new dataset (X, Y) where now these are radiology images (X) and diagnosis (Y).

→ Now retrain the neural network by initializing $W^{[L]}$ and $b^{[L]}$ randomly and retain on new radiology dataset.

→ You have a couple of options here:-

→ If you have a small dataset then just retrain the last layer's parameters, but if your dataset is huge, you could retrain all the layers.

→ Now if you train all the layers then, the training of previous data is called :- **Pre-training** and training on radiology images is called **Fine-tuning**.

→ This can be very powerful technique, because lots of low level thing are done previously like detecting edges, circular objects e.t.c.

Another example :-

Let's say now you have speech data as input to same neural network and output transcript.

→ Now you want to develop voice triggered system.

→ Again do the same, delete the last layer and weights going into it.

→ Sometimes you just don't initialize weights randomly and compute the last layer. Sometimes you attach different layers. like 1, 2 or more than that to compute the output.

And then depending on size of data, train your newly created layers or whole network.

When transfer learning makes sense?

→ It makes sense when you have a lot of data for the problem you are transferring from and relatively less data for the problem you are transferring to.

e.g.: if your earlier voice recognition has 10,000 hrs of data you train your model. It will learn to recognize human voice and sounds. So, that learning can be transferred to voice triggering system even if you have 1 hr of data for that.

→ If we have the opposite case, Transfer Learning wouldn't make sense.

→ Both tasks should have the same input.

→ Low level features from task 1 should be transferred to task 2.

Multi-task Learning:-

In multi-task learning, you start off simultaneously trying to have one neural network do several things at the same time.

Example: Simplified autonomous driving Example:-

→ It needs to detect several different things like:-

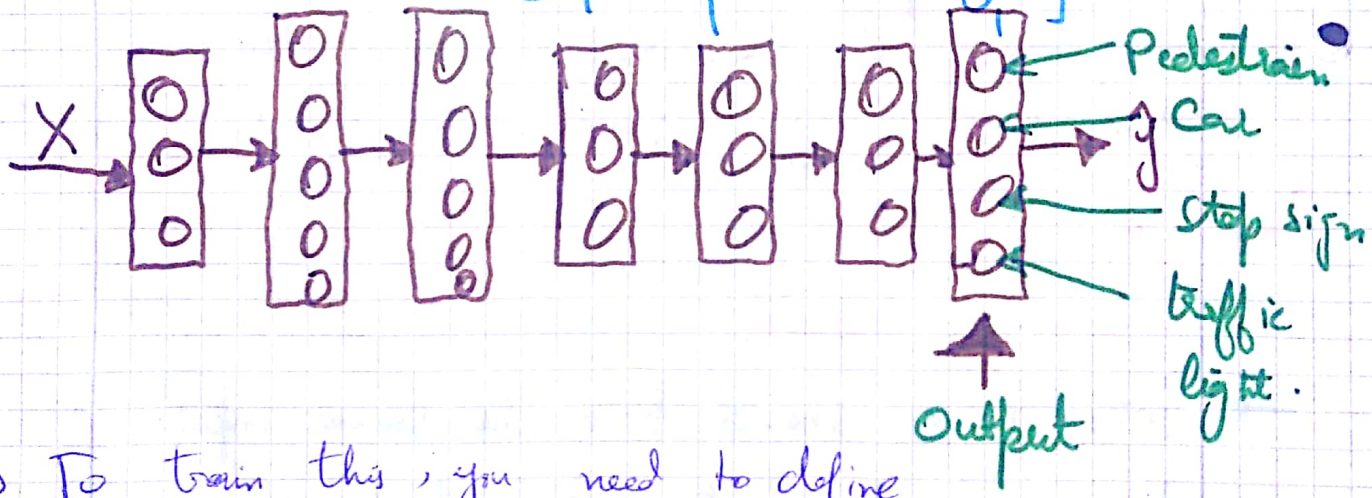
- Pedestrains
- Other Cars
- Stop signs
- Traffic Lights

In this example you would have four labels.

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}^T \in \mathbb{R}^{4\times 1}$$

→ Considering a picture and following is the output given above

$$Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix} \in \mathbb{R}^{4\times m}$$



→ To train this, you need to define Loss.

Loss: $\hat{y}^{(i)}_{(4,1)}$

$$\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} \underbrace{\mathcal{L}\left(\hat{y}_j^{(i)}, y_j^{(i)}\right)}_{\text{usual logistic loss}}$$

Unlike Softmax regression:

→ One image can have multiple labels.

→ This is called Multi-task Learning. Because, what you are doing is building a single neural network that is processing each image and basically solving 4 problems.

→ One other thing you can do is train 4 seperate neural networks instead of training one to do 4 things.

→ But the thing is, if you find that, some of the earlier features in the neural network can be shared between these different types of objects, then training one neural network to do 4 thing actually yields better performance.

→ It can happen that your dataset have images where they have some labels and some are absent like this

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

← Even then you can train the algorithm

And the way you train is that, in the cost function the way you sum over j will be changed. Now you will sum only those values of j with 0/1 label.

When does multi-task learning make Sense?

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

e.g: You have 100 tasks to solve and each task has 1000 images. Now if you do one task at a time 1000 images are very low for training a model. But if you do multi-task learning 99,000 other images can give your progress a boost.

Not very hard and fast rule.

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

Transfer learning is used much more often than Multi-task Learning.

End-to-End Deep Learning:-

What is End-to-End Deep Learning?

- Most recent development.

→ There are some data processing system or learning systems which require multiple stages of processing what end-to-end deep learning does is take all those steps and replace it with just one single network.

Example:- Speech recognition example

Goal here is mapping from X (audio) → Y (transcript)

Pipeline Approach

X (audio) → (MFCC) features → (ML) Phonemes → words → transcript

Traditionally this requires many stages of processing.

→ Now what end-to-end deep learning does is that you input the audio clip and have directly your transcript.

audio ——————————→ Transcript

One of the challenges of end-to-end deep learning is that you might need a lot of data.

→ when you have a smaller dataset like 3000 hrs of data; traditional pipeline work better. It's only that, when you have a large amount of data like 10,000 hrs → 100,000 hrs of data, suddenly end-to-end approach work best.

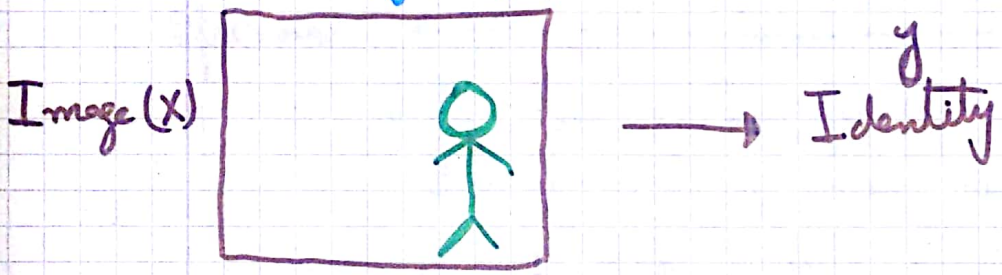
→ when you have ~~minimum~~ medium amount of data you can also bypass the features and go straight to Phonemes step then so on.....

Face Recognition :-

- Installed at many places
- What it does is that, camera takes the picture of the objects whatever that comes in front of it and if it detects a face it opens the gate itself and lets the person through.

Now how do you build this kind of system?

One thing you could do is that, look at the image camera is snapping and try to learn a mapping from there.



Turns out this is not the best approach because person can come from different angles, ~~from~~ image, closer that and so on....

Best approach today is multi-step approach.

1. First detect where the person's face is.
2. Having detected the person's face, then you zoom in to the person's face.
   - Crop that image so that person's face is centered.
   - That cropped image is fed into neural network then.

2 step Approach.

Why this 2-step approach works better?

① Each step's problem is simple.

② Having a lot of data for each of 2 subtasks.

→ If you go just to solve the randomly taken image then it won't work better because there is not enough data for that kind of distribution.

More Examples:-

- Machine Translation:-

  English → Text-analysis → ..... May steps ..... → French

  Because we have lot of pairs for many sentences for English → French translation, End-to-End learning work better in this scenario.

- Estimation of Child's age:-

  → Look at the x-ray of the hand of child and estimate its age.

Non-end-to-end approach:

Image → classify bones size → look up table → estimate.

End-to-End approach:

Image ——————→ age

This approach doesn't work as well today just because there isn't enough data to train this task.

→ Non-end-to-end fashion does work better.

Whether to use end-to-end Deep Learning:-

Let's see some of the pros and cons of end to end deep learning so that you can come up with some of the guidelines.

Pros:-

- Let the data speak. $X \rightarrow Y$
  - ↳ Can give you deeper insight regarding statistical analysis. e.g Phonemes in speech recognition.
- Less hand designing of components needed.
  - ↳ Simplify design work flow

Cons:-

- May need large amount of data. (X, Y)
- Excludes potentially useful hand-designed components. e.g components like feature extraction from speech in speech recognition.

So, when applying end-to-end approach you need to ask

Key Question:- Do you have sufficient data to learn a function of the complexity needed to map X to Y?

e.g: end-to-end seems legible in face recognition for cropped centered images

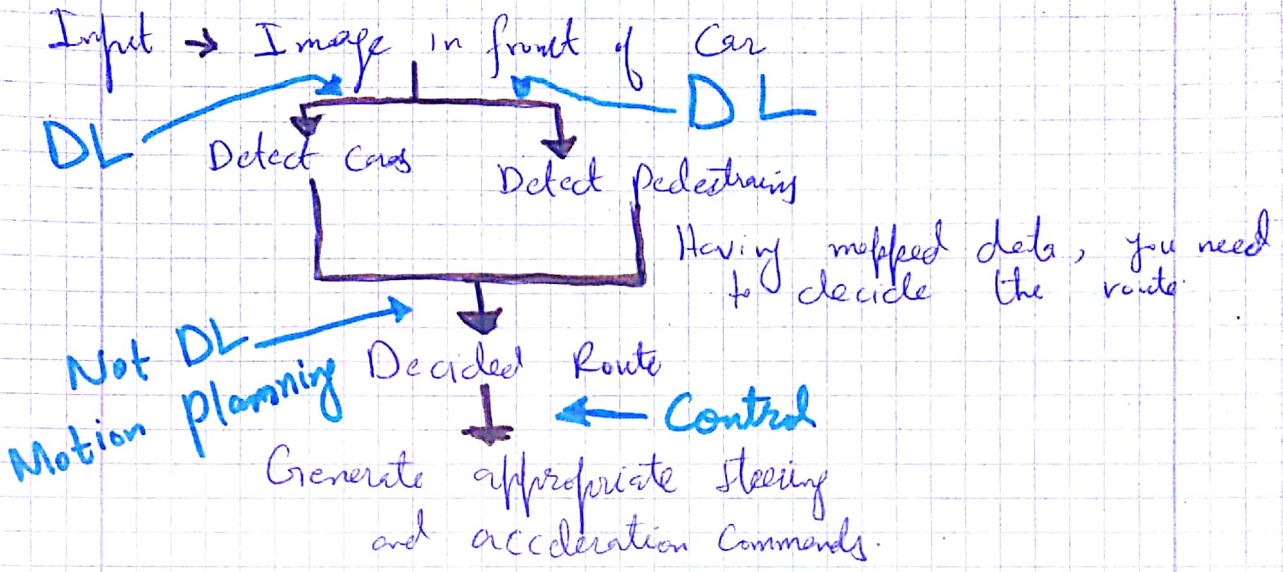
but

for age prediction for a child by looking at the X-ray is bad problem using end to end approach.

Autonomous Car driving :-

How to build a car that drives itself.

→ Not end-to-end deep learning approach.



- Use DL to learn individual components.
- Carefully chose X → Y depending what tasks you can get data for.