**Carrying out error analysis**

Manually examining mistakes that the algorithm is making can give you insights into what to do next. This process is called error analysis.

Working on cat classifier and have achieved 90% accuracy or 10% error. A team member notices it is categorizing some dogs as cats. So we do some adjustments in order to make our cat classifier do better on dogs.

Question is should you start a project that focuses on dog problem?

Here is an error analysis procedure that lets you quickly tell whether or not it is worth the effort?

**Error analysis**

Get 100 mislabelled dev set examples and count how many are dogs. Suppose it turns out 5% are pictures of dogs. If only 5% of the error are dog pictures, then the best you can hope to do is get down from 10% error to 9.5% error which is a 5% decrease in error. And so you decide that this is not the best use of time.

Suppose we look at 100 mislabelled examples and 50 of them are actually dog images. In this case if you actually solve the dog problem the error would go down from 10% to 5%.

Here we have evaluated single idea using error analysis, sometimes you can also evaluate multiple ideas in parallel doing error analysis. We have several ideas for improving the cat detector.

The several criteria each occupy a column whereas rows occupy the number of images we have to manually check. Then put a check mark corresponding to matching criteria. Then we count up the percentage of each of the categories related to total images. Error analysis helps you to make prioritization decisions and understand how promising different approaches are to work on.

**Cleaning up Incorrectly Labelled Data**

You going through the data and find some of the output labels Y incorrect. Is it worth your while to go in and fix some of the labels?

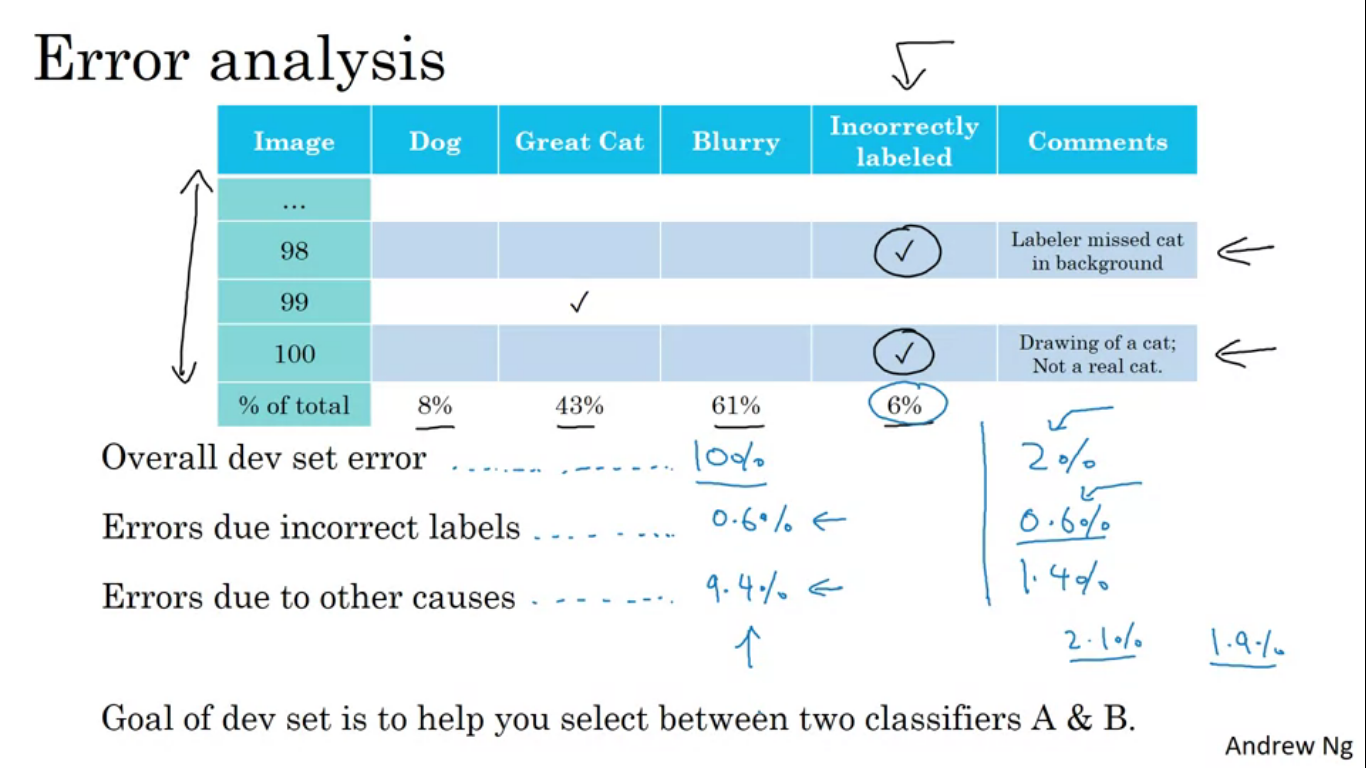
In the cat classification example y=1 for cat and y=0 for non cats. Mislabelled examples if the algorithm outputs wrong value of Y. Incorrectly labelled is when the data in the training/dev/test labelled by human is actually incorrect.

Deep learning examples are quite robust to random error in the training set. If the errors are random it is ok to leave the errors as they are and not too much time fixing them. Deep learning algorithms are less robust to systematic error for eg: if your labeller consistently labels white dogs as cats then that is a problem because your classifier will learn to classify all white coloured dogs as cats.

**Effect of incorrect example in dev/test set**

Add one extra column so you can also add up where the label Y was incorrect. Sometimes our classifier disagrees because the label was wrong, but the classifier was right.

Is it worthwhile to fix up the 6% of incorrectly labelled examples?



If it makes a significant difference to your ability to evaluate algorithms to your dev set, the spend the time in fixing the labels, but if it doesn’t make a significant difference to use dev set to evaluate classifiers, then it might not be the best use of your time.

Now the overall errors are down to 2%, so error due to incorrect labels is still 0.6% and error due to other causes now reduces to 1.4%, so it makes 30% of images incorrectly labelled.

**Correcting incorrect dev/test set examples**

Apply same process to dev and test set to make sure they continue to come from same distribution.

Consider examining examples your algorithm got right as well as ones it got wrong.

The reason above is not always done because if your classifier very accurate then it is getting very fewer things wrong than right. So difficult to validate

Train and dev/test data may now come from different distributions.

Build your first system quickly and then iterate

Building a speech recognition system, there are many challenges, how do you fix which of these to focus on? To solve this, first quickly setup a dev/test metric, build initial Ml system quickly, and see how well to do on dev/test set and evaluation metric.

Next use bias/variance analysis and error analysis to prioritize next steps. Error analysis gives a specific problem then it gives a good reason to focus on techniques to address the problem.

Training and testing on different distributions

In the deep learning era more and more teams are now training on data that comes from a different distribution than your dev and test set.



Designing a cat classifier where the users upload from the cell phone. Two sources of data ,one from mobile, other from the web download a lot of hd images of cats. Finally we care about that the classification from the mobile app usage is more accurate. Now we have small data from the mobile images and more data from the one we have crawled from the web. So we have two distributions.

One thing to do put both types of images together and randomly shuffle them into train test dev set. The advantage is that all train/dev/test come from the same distributions, disadvantage is in the dev set a lot of the examples are from the web crawled. So training set will be 205000 examples and dev and test set will have 2500 examples each. As the disadvantage says out of 2500,2381 will be coming from web pages and 119 will come from mobile app uploads. And by this we are saying spend most of the time optimizing the webpage images which is really not what we want.

The other options will be that training set will have all 200000 images from the web and then add in 5000 images from the mobile app, so our dev and test set would be all mobile app images. Disadvantage here is that now our training distribution is different from dev and test set but it turns out that this split will give you better performance in the long run.

Another example, speech activated rear view mirror of the car. For training we can have all the data we have while solving other speech related problems. Let’s say we have 500000 utterances, and 20000 from speech activated rear view mirror. Set the training set to be 510000 clips from previous sources and dev and test set could be 5000 utterances each that’s drawn from actual speech activated rear view mirror.

**Bias and Variance with mismatched data distributions**

In the cat classification example humans get near perfect accuracy. Training error is 1% and dev error is 10%, if the dev data came from the same distribution as the training set so looking at this value we have a large variance problem. But in case the training data and dev data come from diff distributions we can’t say this. It can be that training error was easy and dev set was much harder. So here two things changed at a time, algorithm saw data in training set and not dev set distribution of data in dev set is different. To understand these two effects , we create a training dev set- it has same distribution as training set but is not used for training.

So we randomly shuffle the training set and randomly carve out a subset to be training dev set. Training and training dev set have the same distribution. But now NN will just run on the training set proper. To carry out error analysis we will look at training set, training dev set as well as dev and test set.

Training error=1%

Training dev error=9%

Dev error=10%

When you went from training data to training dev data, the error went up a lot. This tells you that you have a variance problem. Since the training and training dev come from the same distribution.

In another example let’s say that training error is 1%, training dev error is 1.5%, dev error is 10%. Pretty low variance problem but this is a data mismatch problem.

In another example, train error is 10%, training dev is 11% and dev error is 12% , human level proxy for baye’s error is 0%. Here we really have a avoidable bias problem.

In another example, if the training error is 10% , training dev error is 11% and dev error is 20%. So it has avoidable bias quite high, variance here is small but data mismatch is quite large. So here we have avoidable bias problem as well as well as data mismatch problem.

So key quantities to look at are –

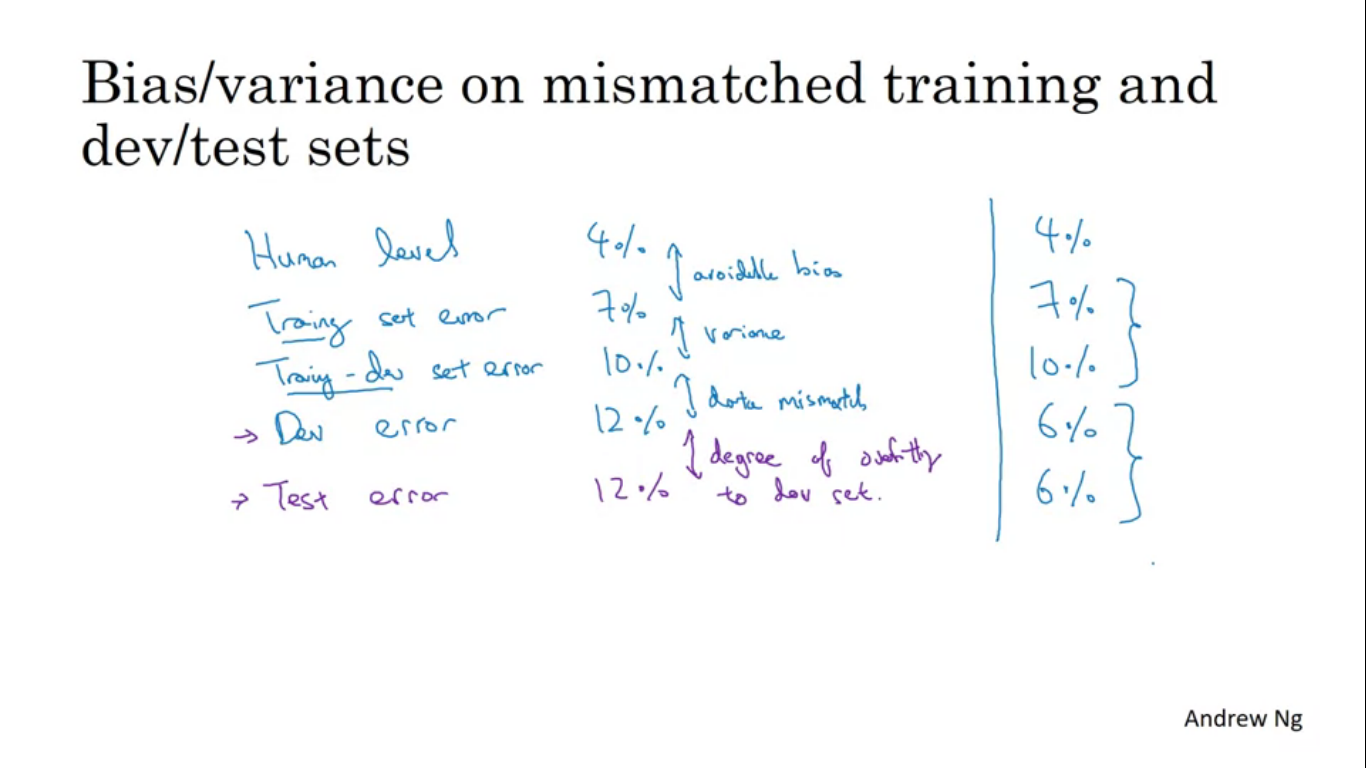
Human level error

Training set error

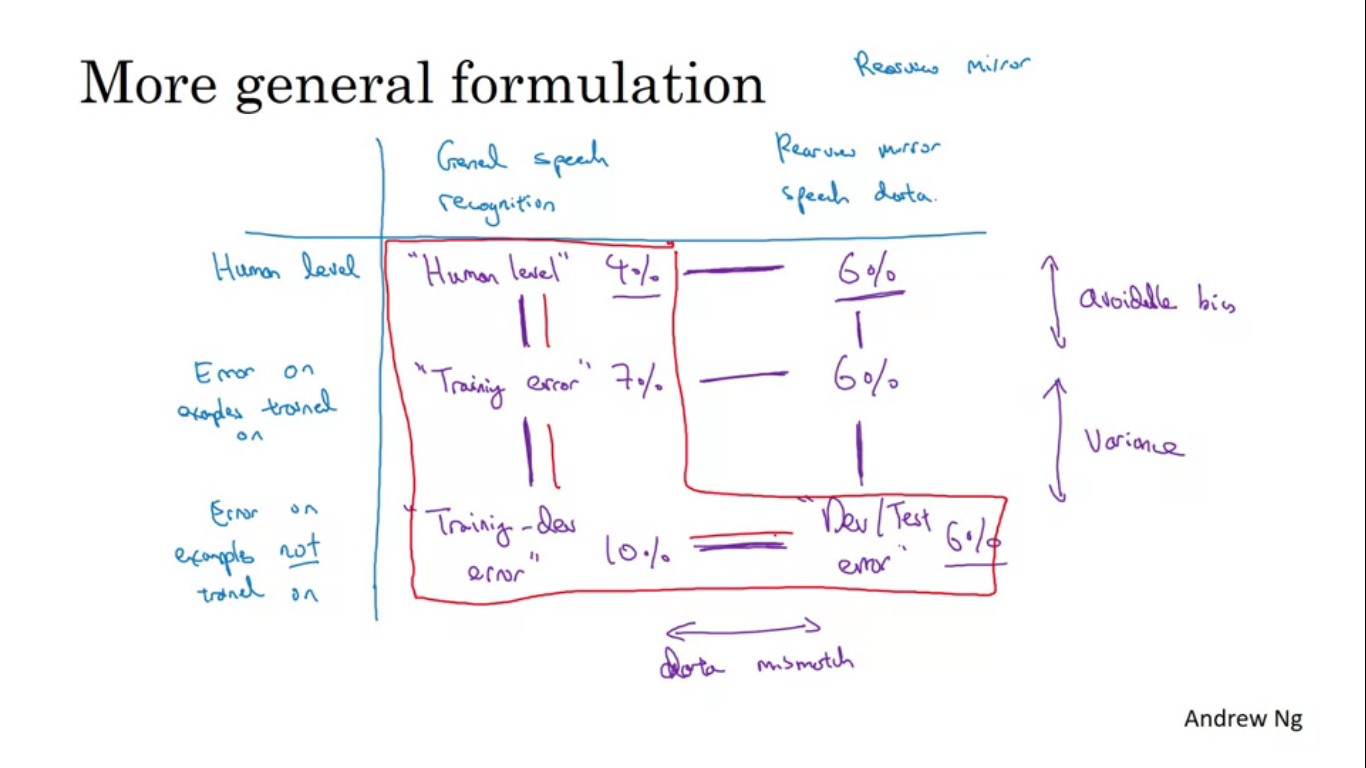
Training dev set error

Dev set error

Depending upon the differences as above, we can get various problems.



If dev set and test set errors are too far apart, then it is the case of over fit the dev set and we need to get more data for the dev set.



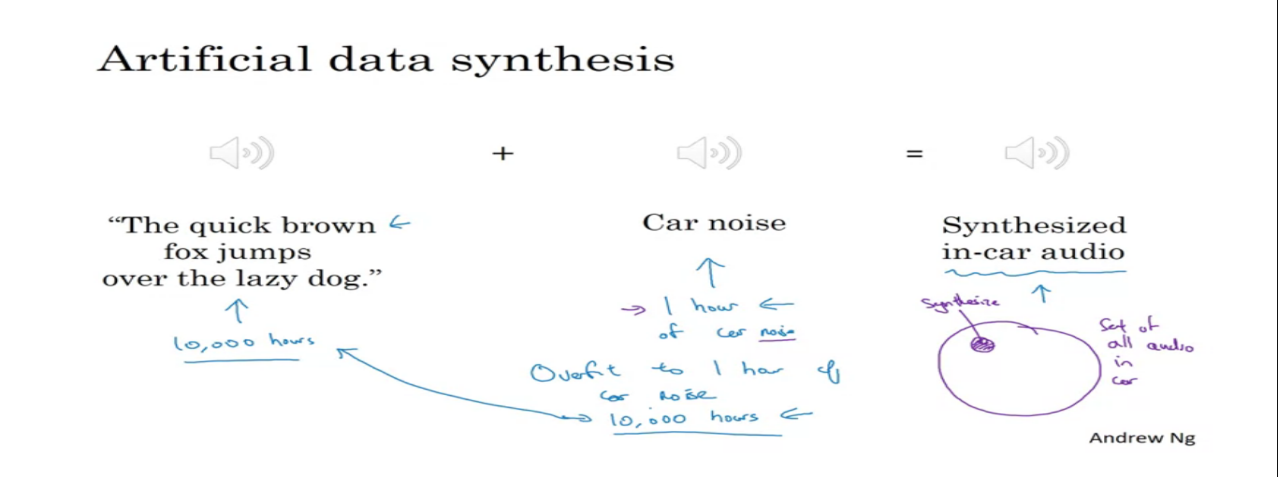
Sometimes if the dev test set distribution is much easier then the dev error and test error can go down

**Addressing Data Mismatch Problem**

Carry out manual error analysis and try to understand difference between training set and the dev/test set. Example : if you are building a speech activated rear view mirror application, you try to figure out how your dev set is different from the training set. A lot of dev set examples are very noisy and there is a lot of car noise.

Next we can try to make the training data more similar or try to collect more data similar to dev and test set. If we have problem with the street number we can have a data of more people speaking out numbers and add it to training set.

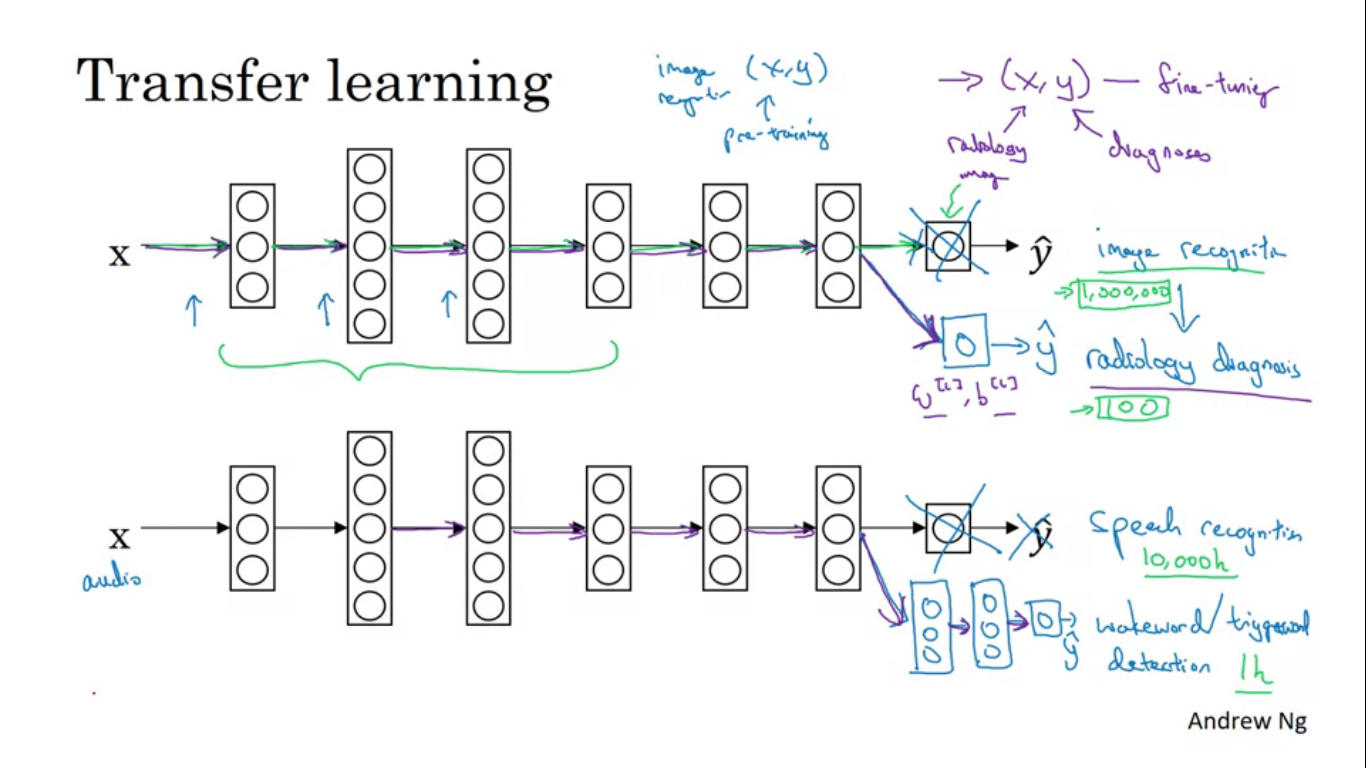
To make training data more similar to dev set



Artificial data synthesis - We have recorded a large amount of clean audio without the car background noise and get the background noise of car, and if we take these two and join them together we can then get the sound as in the noisy car. A bit of caution though, let’s say we have 10000 hours of clean audio and 1 hour of car noise, so one thing we can try is take this 1 hour of car noise and repeat it 10000 times. But there is a chance that learning algorithm will over fit 1 hour of car noise.

By synthesising pictures we could train a pretty good computer vision system for detecting cars. From the ser of all cars we synthesise a very small subset of cars then to the human eye maybe the synthesised images look fine but you might over fit to this small subset you are synthesising. From a video game we can get a huge number of cars, but if the video game has 20 unique cars then the video game looks fine. But the world has a lot more than 20 unique designs of cars, so NN will probably over fit to these 20 cars. If you think you have a data mismatch problem then error analysis is recommended. Try to gain insights into how these two distributions of data might differ.

**Transfer learning**



Sometimes you can take knowledge that the NN has learned from one task and apply that knowledge to a separate task.

Let’s say you have trained your network on image recognition, if we want to transfer the task to something as radiology diagnosis.

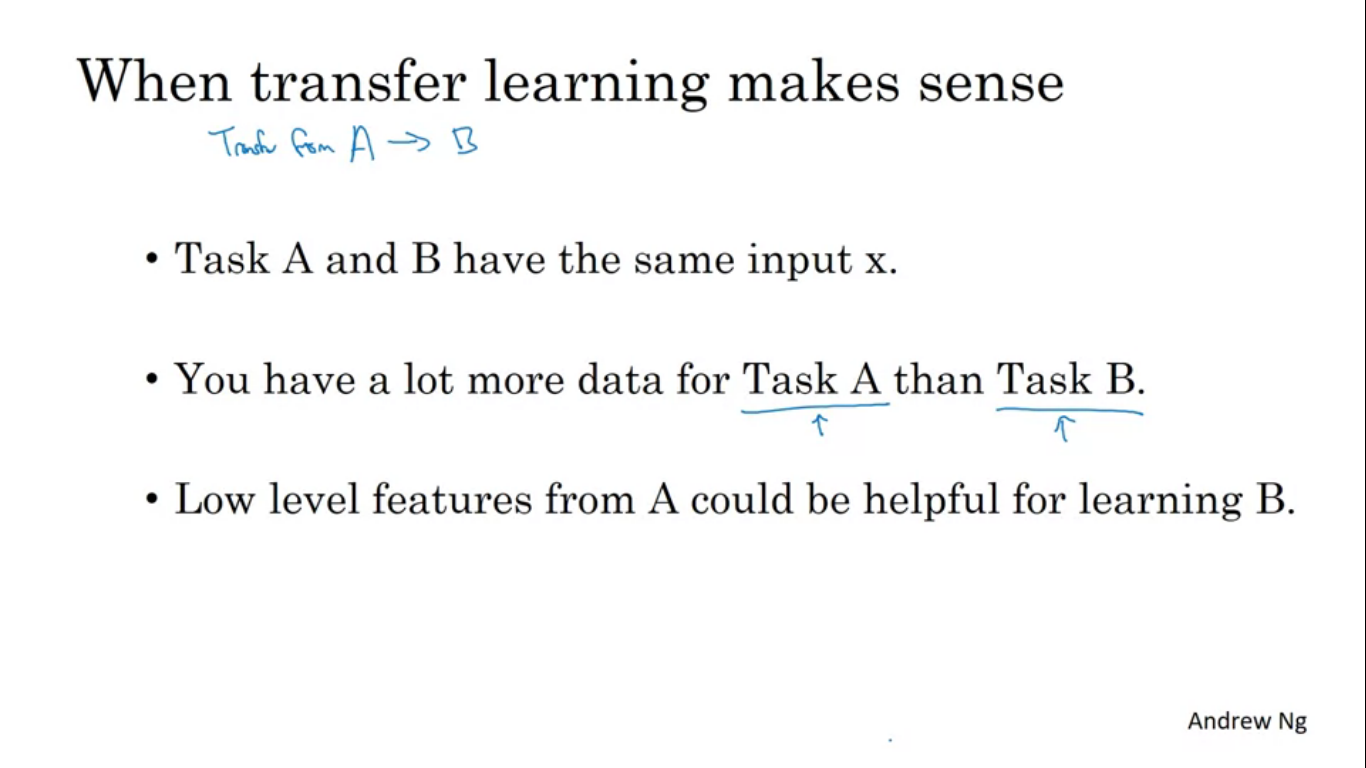
Take the last output layer of the NN and delete it long with the weights feeding into the last output layer. Now create a new set of randomly initialized weights just for the last layer.

If you have a small radiology data, we might want to train only the weights of the last layer and keep the rest of parameters fixed.

If you have enough data, you can also retrain all the layers of the rest of the NN. Initial phase of training of the NN for some other task is called pre-training. And then when you train the complete NN and update the weights according to new data, that is called as fine tuning.

Taken knowledge from image recognition and transferred it to radiology diagnosis.

Similarly in another example, let’s say that we are training speech recognition where input is audio clips and Y is some transcripts.



Transfer learning makes sense when you have a lot of data for the problem you are transferring from and relatively less data you are transferring to. 1 million examples for image recognition task, and only 100 examples for radiology diagnosis.

One place where transfer learning doesn’t work is when you have 100 images for image recognition and 1000 images for radiology diagnosis. The value of 1 image of x ray is much more than the value of 1 image of image recognition.

When transfer learning makes sense

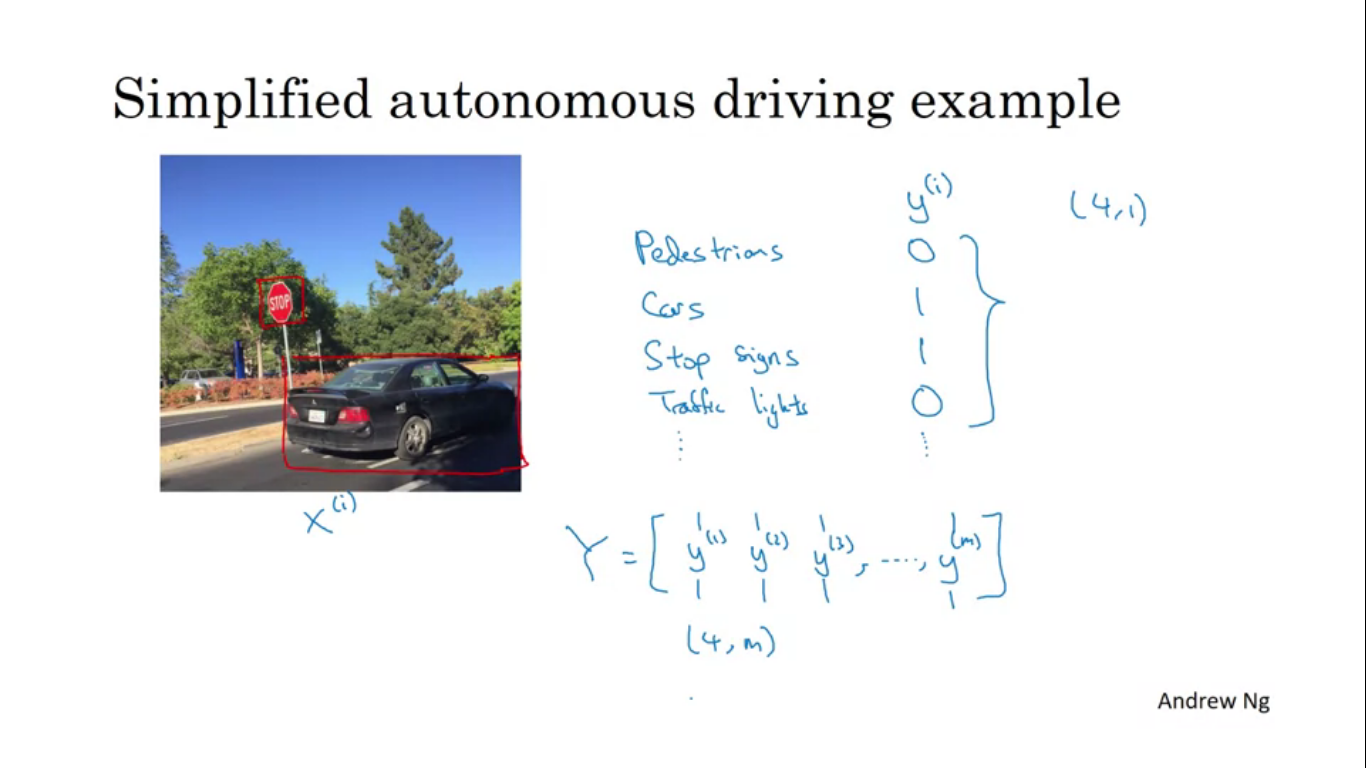
**Multi task Learning**

In transfer learning we have a sequential process, where we learn from task A and transfer that to task B, in multi task learning we try to do several things at the same time. Each of these tasks help all the other tasks.

Example – we are creating a self driving car, it would need to detect several different things like pedestrians, other cars, stop signs, traffic lights. So for every input image we have four labels.

Y is now a 4Xm matrix whereas previously when Y was a single number this was 1Xm matrix.

Rest all is provided in the images.



Softmax regression assigns single label to a single example, here one image can have multi labels. For each image we have multiple classes of objects.

If we train a NN to minimize this cost function we are carrying out multi task learning. Training 1 NN to do four things works out better performance than 4 NN doing 1 thing separately



It can also happen some images are fully labelled and some other examples are partially labelled, we can still train our algorithm to do four tasks at the same time. To train in the loss function we sum only over the values of j with a 0/1 label.

When does multi task learning makes sense

Training on a set of task that can benefit from having shared lower level features

The amount of data you have for each task is quite similar.

Train a big enough NN to do well on all the tasks.

**End to end deep Learning**

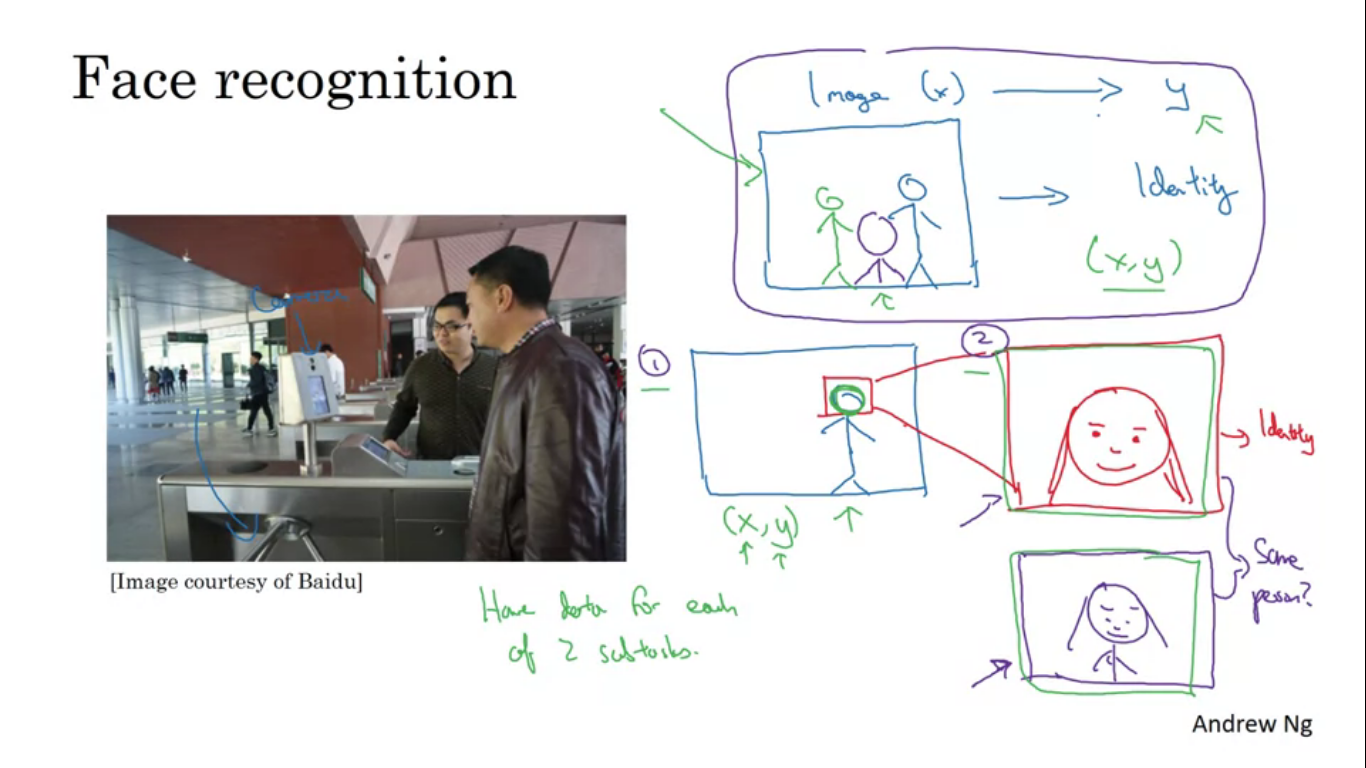
There are some processes which require multiple stages of data processing, end to end deep learning replaces all those stages with a single NN.

Speech recognition example where input X is audio clip and Y is transcript. Traditionally speech recognition has many stages like feature extraction using MFCC an algorithm for extracting certain set of hand designed features for audio. Having extracted some low level features we apply ML algorithm to find phonemes int he audio clip. Phonemes are the basic unit of sound. Combine together phonemes to form different words and stringing the words together form transcripts of the audio clip.

In end to end deep learning just input the audio clip and have it directly output the transcript.

Challenge in end to end deep learning is we need a lot of data before it works well.

Face recognition system



One piece of software to detect a person’s face , after detecting the face , zoom in the face and crop so that person’s face is centered. The cropped image is then sent to NN to learn the person’s identity. Breaking up a process into two steps has two advantages, one is the problem is broken into relatively simpler problems and second is we have a lot of data for each of the two subtask.

Pros and cons of end to end learning

End to end learning lets the data speak

Less hand designing of components needed

Cons

May need a large amount of data

Excludes potentially useful hand designed components. When you have a lot of data it is less important to hand design things, but if you don’t have enough data then having a carefully hand designed system can actually allow humans to inject a lot of knowledge about the problem.

Key question for end to end learning

Do you have enough data to learn a function of the complexity needed to map from X to Y?

How do you build a car that drives itself?

Non end to end deep learning approach

Image of what is in front of the car. To drive safely we need to detect other cars and pedestrians. After that we need to plan our own route. To execute the route we need to generate acceleration and braking and steering commands.

