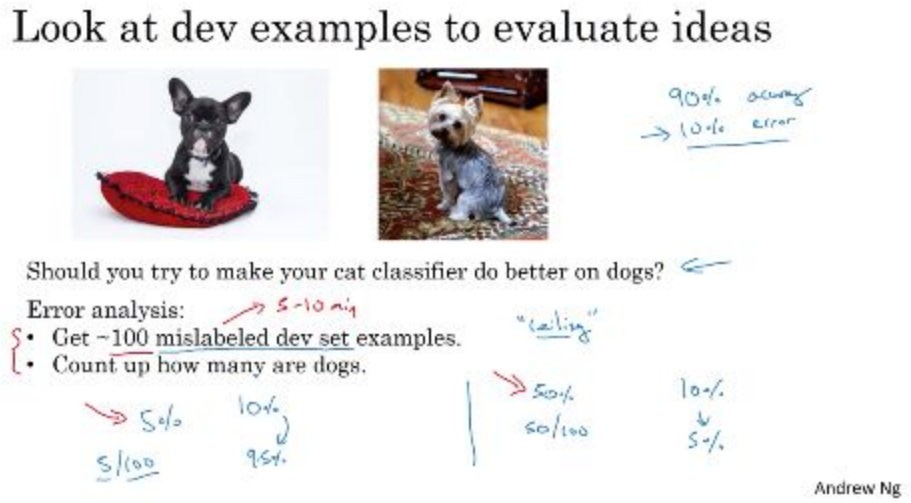
Andrew Ng's Structuring Machine Learning Projects

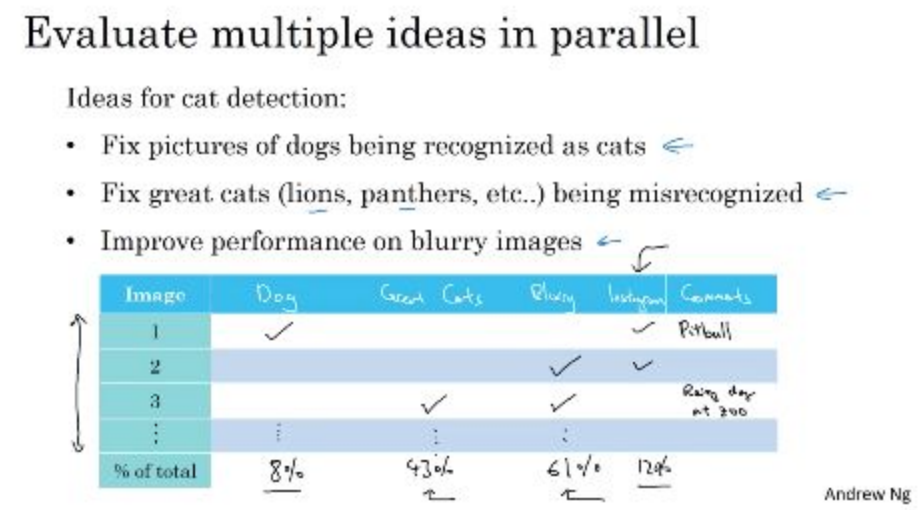
Week 2: ML Strategy 2

Carrying out error analysis

- What do we do if our learning algorithm is not yet at human-level performance? One way to decide what to do next is to carry out error analysis. Let's illustrate it with an example: suppose our cat classifier has 10% error and one idea to make it better is to try to make it identify dogs better (some dogs look like cats). Assuming we have 100 mislabeled dev set examples, how do we know the dog idea is worth spending time on? In the first example we see that only 5 examples were mislabeled as dogs so the best we can do is reduce our error from 10% to 9.5%; doesn't seem like a lot of return if we invest many months on improving dog recognition. In the 2nd example below we have 50 mislabeled dog images so we can reduce our error from 10% to 5%; this case has a better ROI and looks like it is worth pursuing.



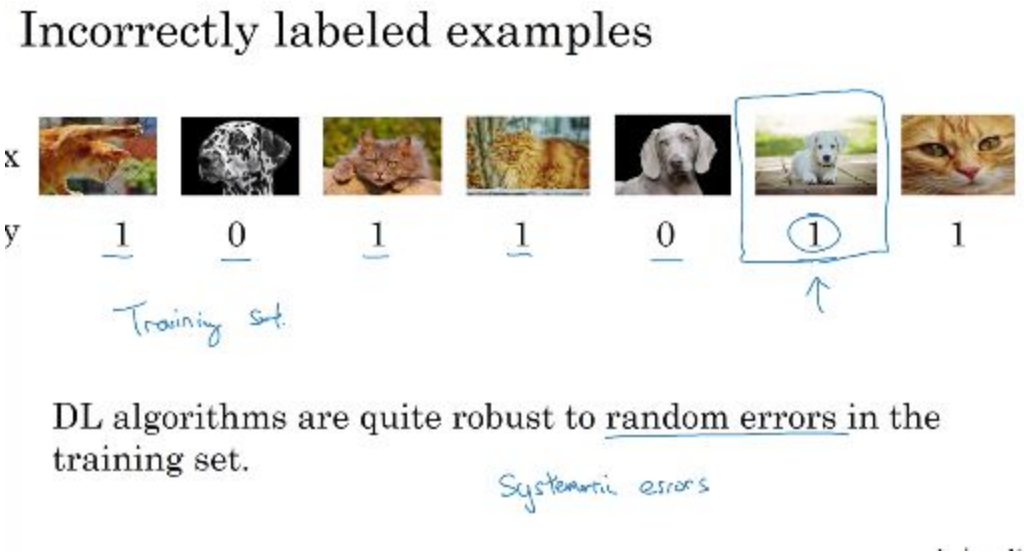
- In the previous example we were evaluating one idea. It is possible, however, to evaluate multiple ideas at once. Professor Ng uses a spreadsheet like the one shown below where the rows are the mislabeled images and the columns are the categories to what those images belong. We then do a percentage calculation to figure out which ideas are most promising in terms of improving our algorithm's performance. In the example below, great cats and blurry seem good ideas to pursue.



Summarizing, we carry out error analysis by taking the mislabeled examples from the dev set and classifying them on the cause of the misclassification. We then do a % calculation to figure out which ideas are worth pursuing.

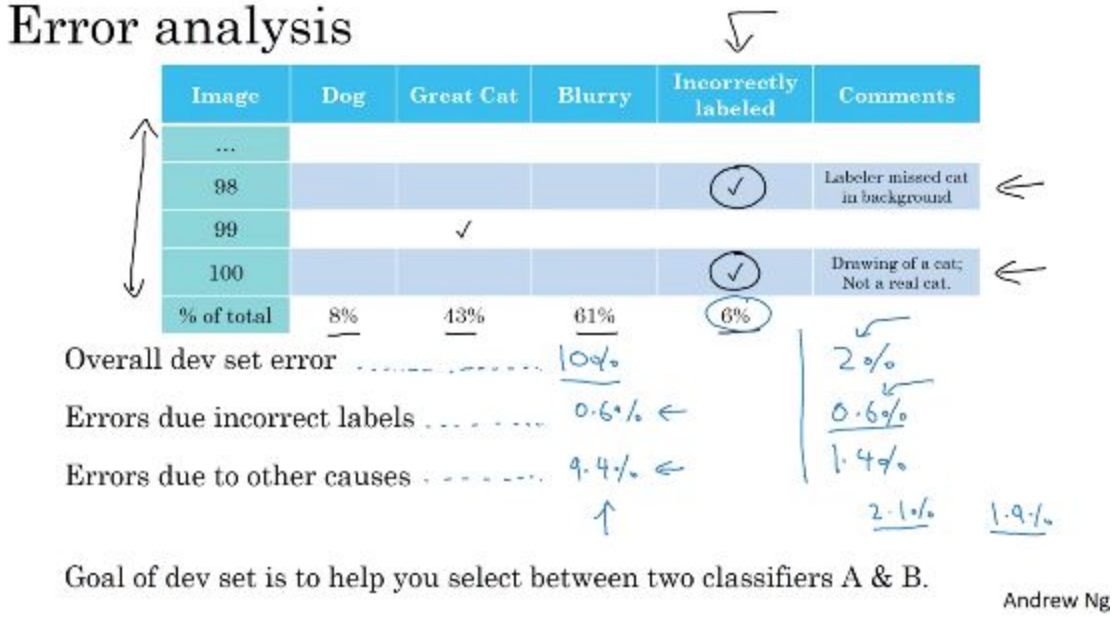
Cleaning out incorrectly labeled data

- Incorrectly labeled examples are examples where the label is incorrect as the one shown below. In the case below the label should be a 0. What do we do in such a case? It turns out deep learning algorithms are resilient to random errors in the training set; they are not resilient to systematic errors.

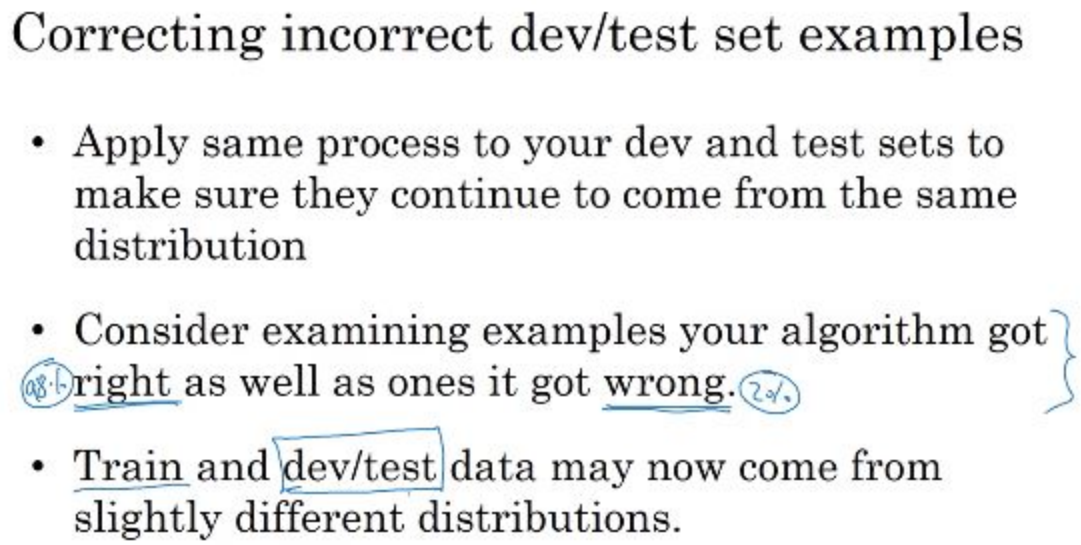


- What do we do if the incorrectly labeled data is in the dev set or test set? What we do is we take it into account during the error analysis: we add another column to our table so we can quantify the % of misclassified examples that had an incorrectly labeled example. Then what? Depends on a couple of things. We need to look at more numbers:

the overall dev set error, the error due to incorrect labels and error due to other cases. In the first example below the error due to incorrect labels (0.6%) is very small compared to the error due to other causes so we are probably better off focusing on some of the other causes. In the 2nd example below things change because now our overall dev set error is 2% and our error due to incorrect labels (0.6%) is a big percentage of the overall error. In this case it makes sense to try to fix the incorrect labels.

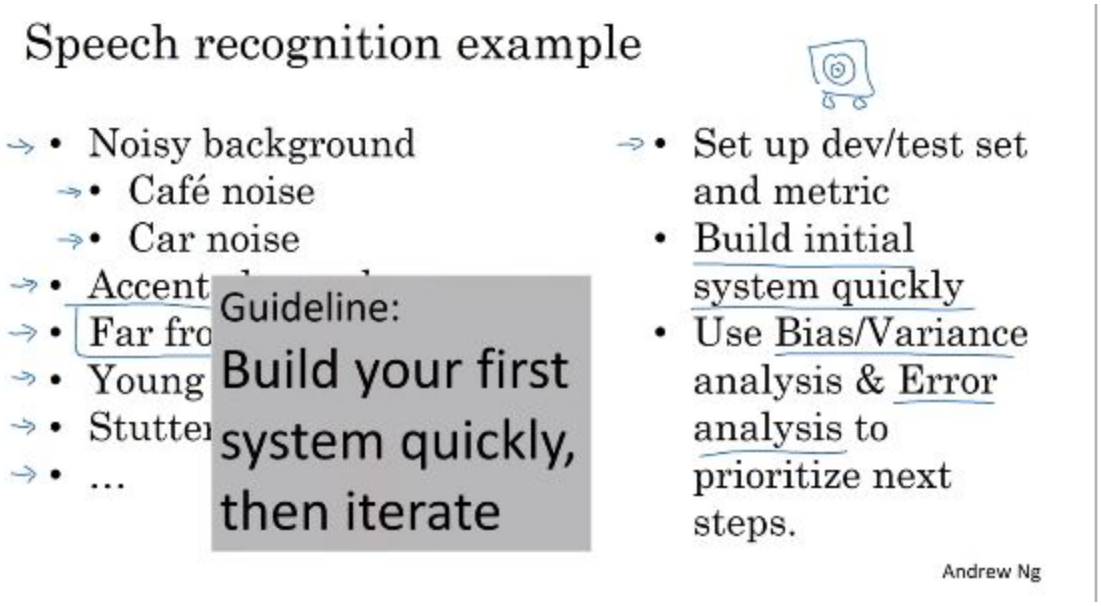


- How do we fix these incorrect labels? Here are some guidelines: apply the same process to the test and dev sets so they continue to be from the same distribution, consider examining examples we got right and wrong and, finally, our train and dev/test sets may now come from different distributions (this is fine and we are going to see why in a future session).



Build the first system quickly and iterate

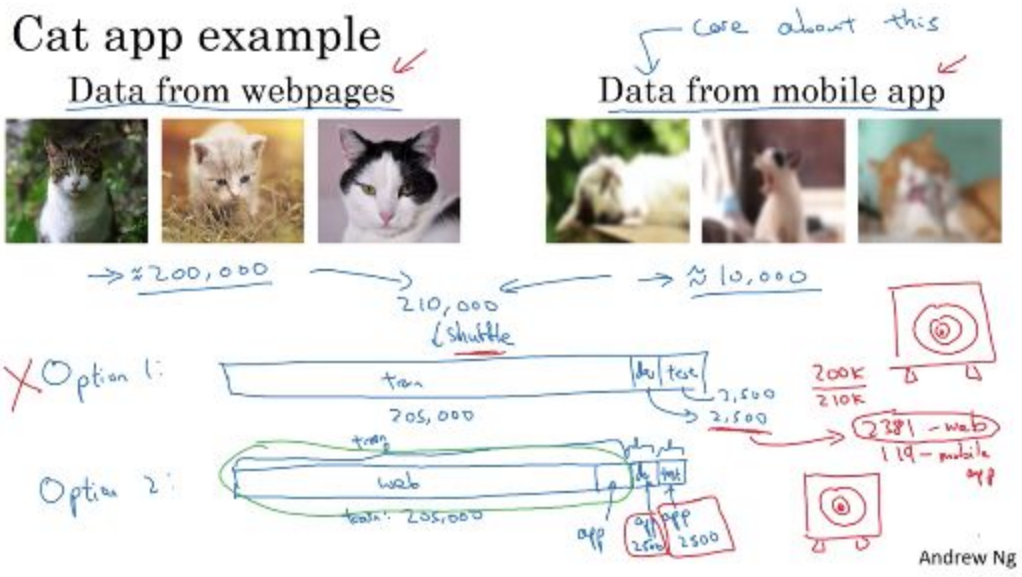
- The main guideline here is to build our first system quickly and iterate. A deep learning problem will have many avenues to explore. The speech recognition example below list a few of the relevant avenues for that particular problem. So what we do is:
    - Set up our dev/test set and a metric
    - Build the first system quickly
    - Use bias/variance and error analysis to decide what to do next.



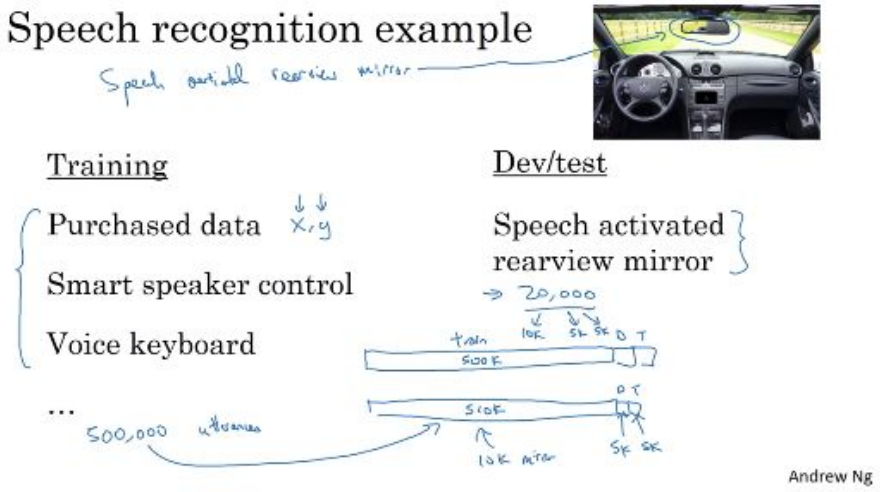
Training and testing on different distributions

- How do train and test with examples from different distributions? Let's see how we do it by looking at the case below. In the example below we want to build a learning algorithm that recognizes cats in a mobile app. However, we only have a data set of 10,000 mobile app examples. If we crawl the web for cat images we will get a large dataset, say 200,000 images. What do we do then? We have 2 options:
    - Option 1 is to randomly shuffle the 2 datasets into a training set, a dev set and test set. Let's assume our training set will be 205,000 images, the dev set 2,500 images and the test set 2,500 images. The advantage of this approach is that both training, dev and test sets come from the same distribution. The disadvantage is that most of our dev and test set examples are going to come from the web images (~2,381) so we are not really testing for the final objective (mobile app cat images).
    - Option 2 is to have a training set of 200,000 web images + 5,000 mobile app images, a dev set of 2,500 mobile app images and a test set of 2,500 mobile app images. The advantage here is that we are aiming for what we really want,

meaning we are testing for the end goal (mobile app images). This choice gives us better performance and is the one recommended by Professor Ng.



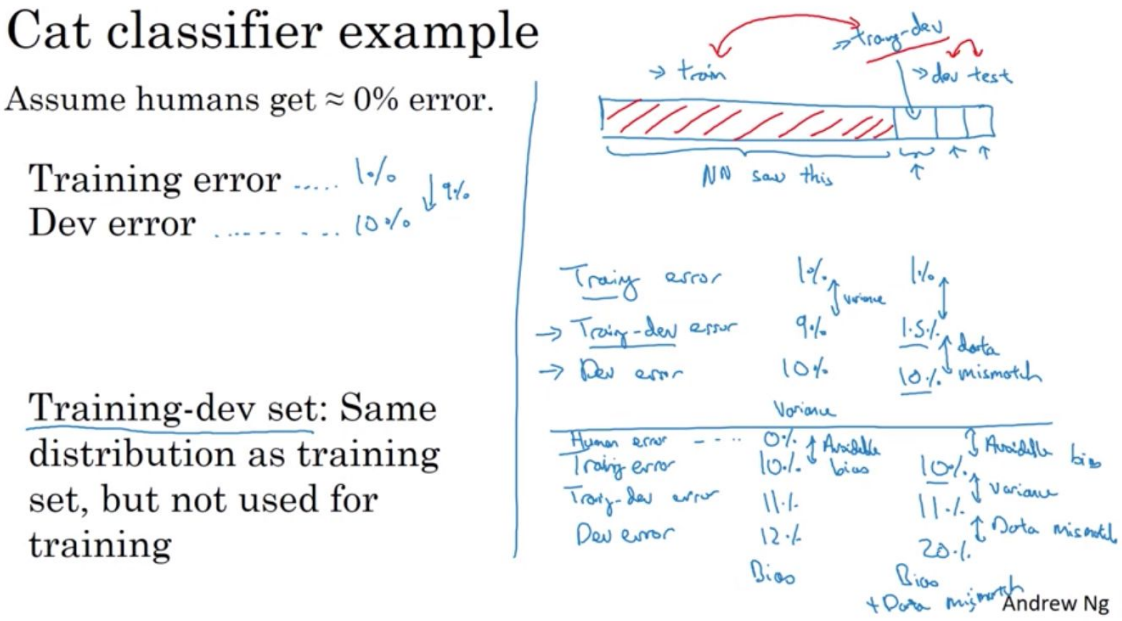
- Here's another example, this time it is a speech recognition example. In this case we have a speech-enabled rear view mirror and we have 20,000 examples that come from people talking to the rearview mirror. We also have 200,000 utterances that we have acquired through various means. How do we create our sets from this data? Similar to the recommendation above.
    - The first option is to have a training set with the 200,000 utterances from different sources and split the 20,000 rearview mirror examples in the dev and test sets.
    - The second option is to have a training set of the 200,000 utterances from different sources + 5,000 rearview mirror examples. The remaining rearview mirror examples are given to the dev set (2,500) and test set (2,500).



Both of the options above are reasonable options for how to deal with different distributions.

Bias and variance with mismatched data distributions

- How do we deal with bias and variance when the training set comes from a different distribution than the dev and test sets? Let's use an example to see what we do. We have a cat classifier example where the human error ~ 0%, the training error is 1% and the dev error is 10%. If all the sets came from the same distribution then we have a variance problem but if the sets came from different distributions then we cannot draw that conclusion anymore. The problem here is that 2 things happened when we went from training to dev error: our classifier is seeing the dev set for the first time and the dev set comes from a different distribution so we don't really know what of the 2 things is causing the big difference in error rates.

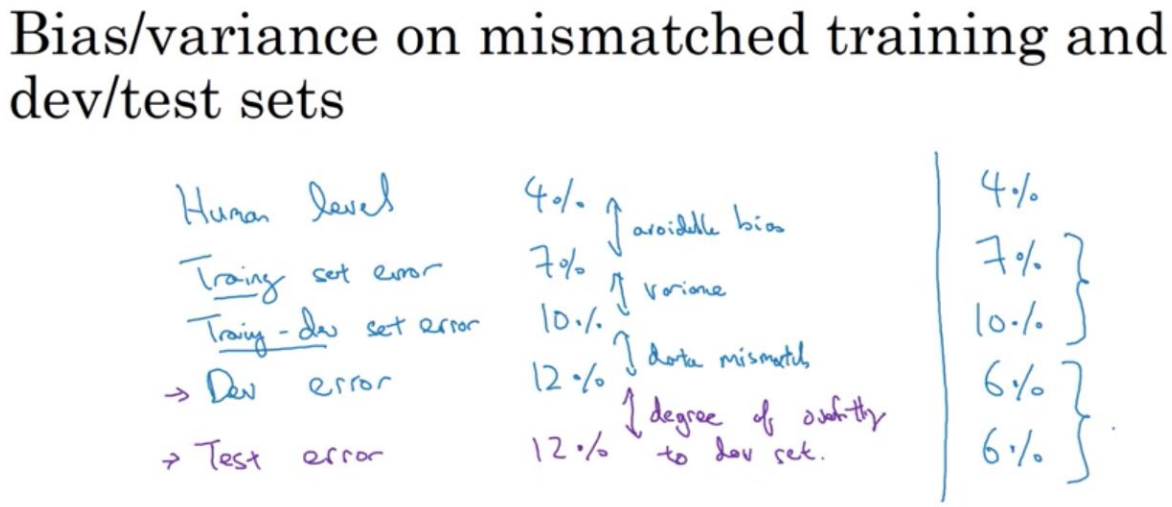


What we do to answer this question is to create a training-dev set which uses the same distribution as the training set but it is not used for training. We randomly shuffle the training set and reserve a piece of it to be the training-dev set; now the training and the train-dev set error come from the same distribution. We then train our neural network on the training set and test it in the train-dev set. A couple of examples show us what we mean: In the first example above we have a training error of 1%, a train-dev error of 9% and a dev error of 10%; we then say we have a variance problem. In the second example we have the same 1% error in the training set, 1.5% in the train-dev error and 10% in the dev error; in this case we say we have a data mismatch problem. On the 3rd example we have a training error of 10%, train-dev error of 11% and a dev error of 12%; in this case we have an avoidable bias problem because the difference between the human error and

the train error is the biggest. Finally we have an example where we still have a 10% training error, an 11% train-dev error and a 20% dev error; in this case we have 2 problems: avoidable bias (big difference from human error to training error) and data mismatch (big difference between train-dev error and dev error).

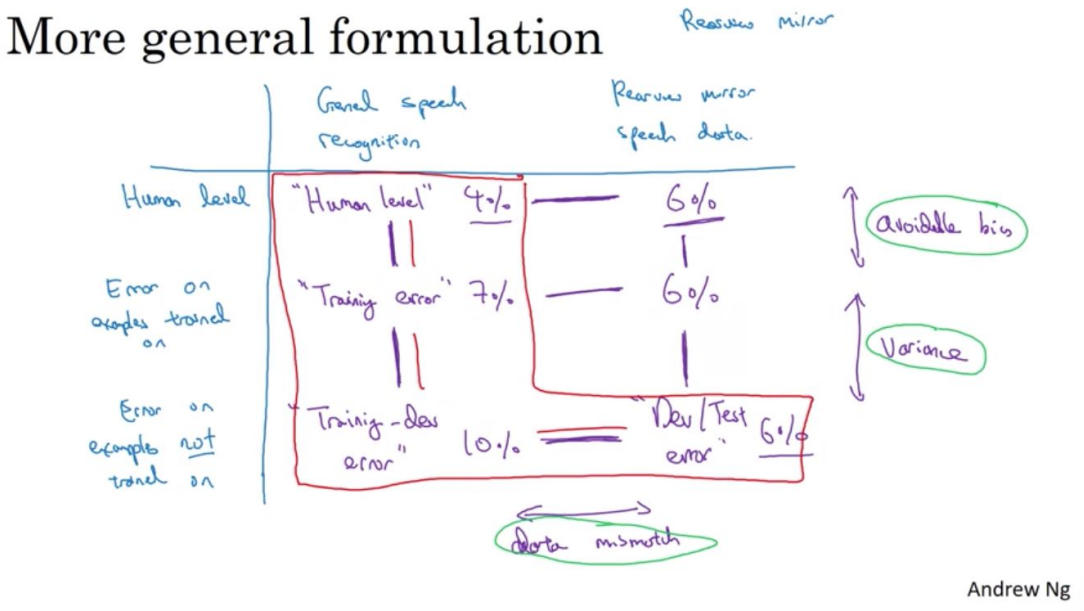
- How do we generalize these examples? We start by getting the error rates of 5 sets
  - Human level performance
  - Training set error
  - Train-dev set error
  - Dev set error
  - Test error

The difference between human level and training set error is the avoidable bias, the difference between training set error and train-dev set error is the variance, the difference between train-dev set error and dev set error is data mismatch and, finally, the difference between dev set error and test error is the degree of overfitting to the dev set.



In some cases we might see the error rates go down and shown in the 2nd example above. That is because the data comes from different distributions. One distribution, i.e. the training and train-dev sets, has harder data than the other distribution, the dev and test sets.

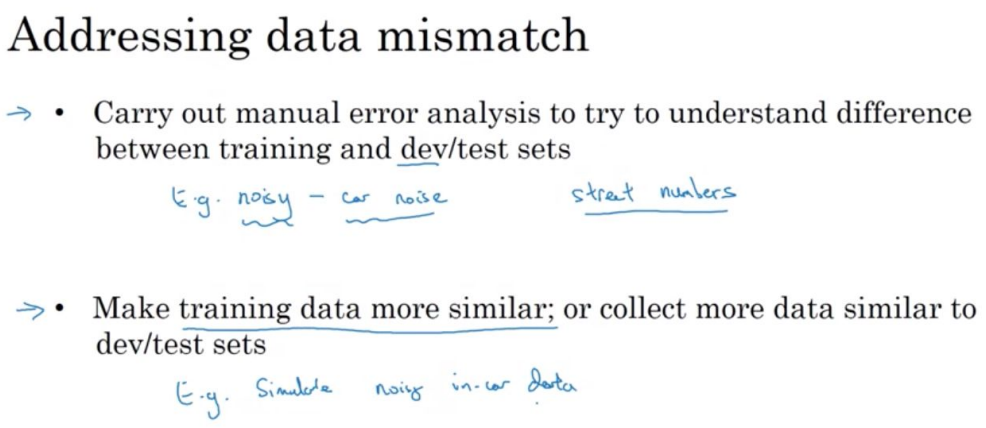
- The generalization is then to create a table like the one shown below (using our speech-enabled rearview mirror example). On the rows we have the different set errors and on the columns we have the different distributions (where the data came from). So far we have been using the avoidable bias, the variance and the data mismatch differences. Adding the missing pieces (the human error on rearview mirror speech data and having some rearview mirror examples that we train on) gives us additional insights but, in general, we are fine using the 3 differences we have talked about.



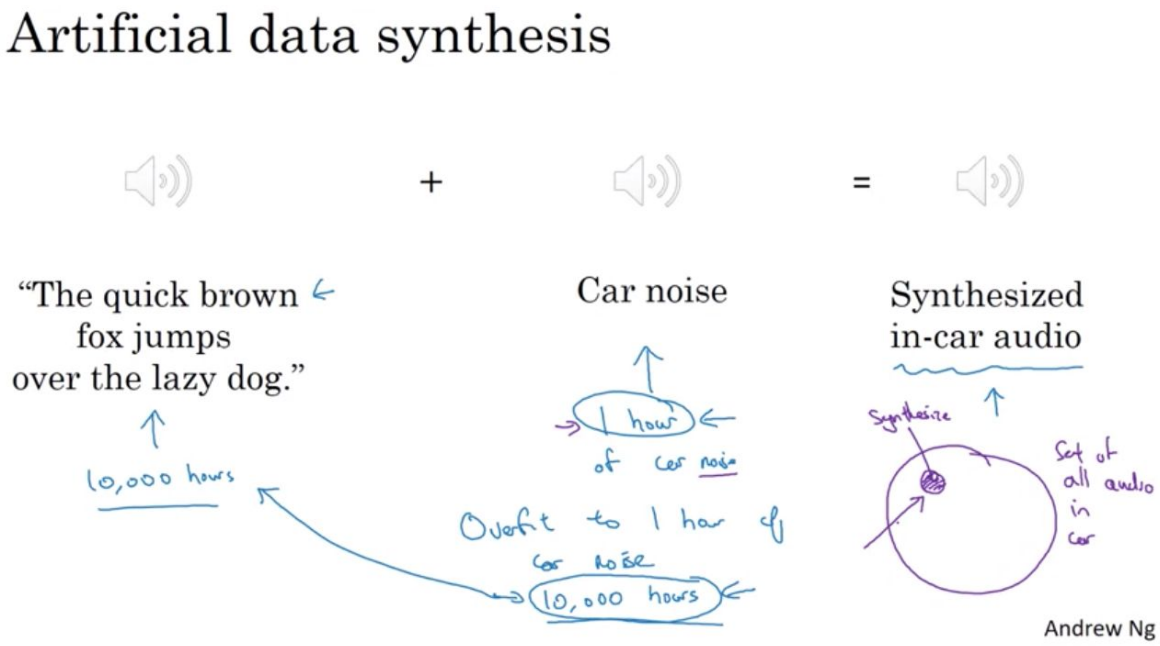
Data mismatch is new kind of thing to look at for us, before we only worried about bias and variance. In the next session we will see what are the things we can do to address data mismatch.

Addressing data mismatch

- What do we do if we have data mismatch? A couple of things: first we do manual error analysis to understand the difference between the training and dev/test sets. It could be a million things (i.e. car noise or trouble recognizing numbers for our speech-enabled rearview mirror). Once we have some insights on what the problem is we will try to make the training data more similar by adding data similar to the dev/test sets.

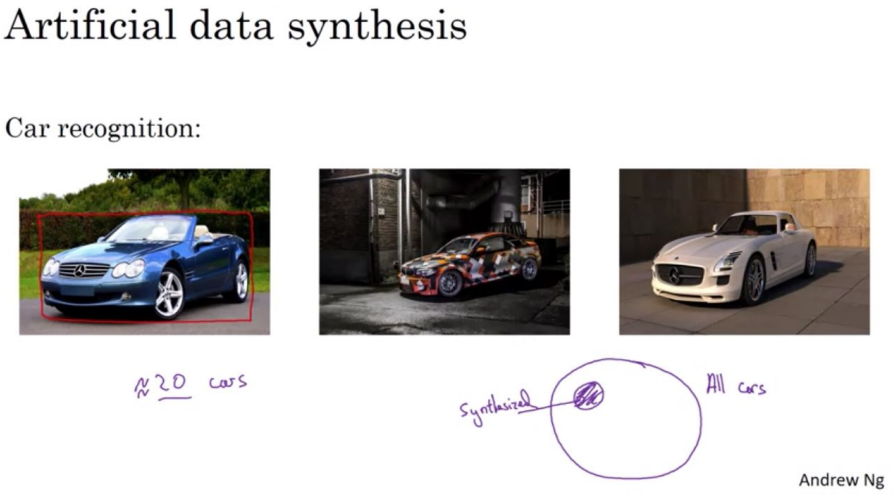


- One thing we can try to solve the problem above is artificial data synthesis. This is a process where you take 2 separate relevant audio files (a person speaking a sentence and car noise) and combine them in one synthesized audio file and we then use that file in our training.



One problem we might run into is that we might have a lot of hours for one audio file and not many hours for the other audio type. One way to solve this problem is to just use the existing audio file and replicate it multiple times (i.e. we take a 1 hour car noise audio file and we replicate it 10,000 times to match the 10,000 hours we have of people talking). A problem with this process is that we might overfit for a small part of the whole universe of car noise audio since we are just using 1 file that is 1 hour long.

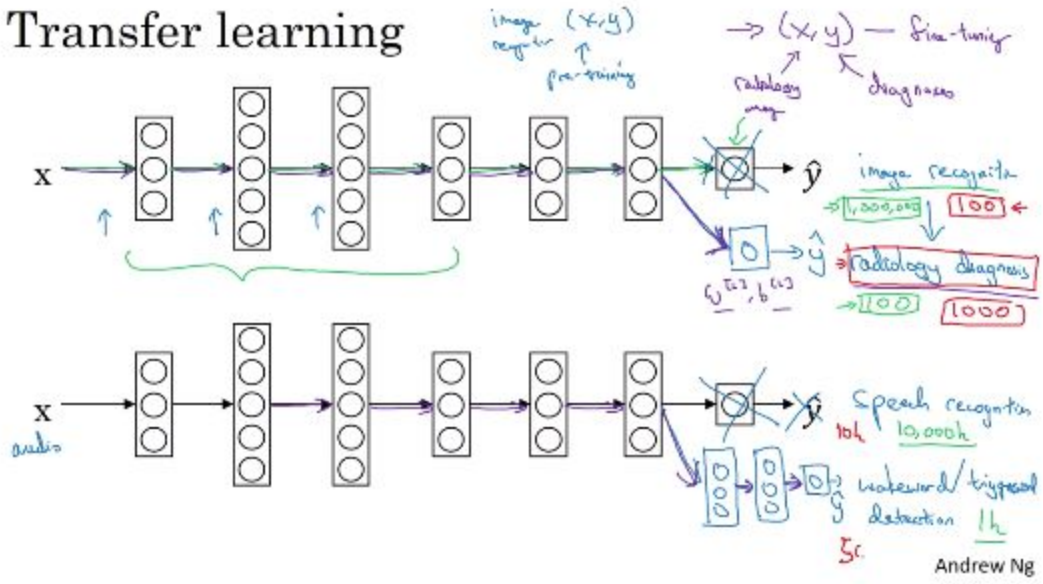
- Another example of artificial data synthesis is below, in this case for a car recognition problem.



If we don't have many car images one way to get more is to use computer graphics to generate more (like the ones shown above). Another way is to use a video game with cars and take images from there. The problem we have is the same as before: we might be overfitting to a very small part of the overall universe of cars. Summarizing, artificial data synthesis works but we have to be careful of the overfitting problem.

Transfer learning

- Transfer learning is taking one neural network that has been trained with one set of data and apply that knowledge to another task. Below we have two examples. In one example we trained a neural network to recognize cats and then we are going to use it (with some modifications) to do radiology diagnosis. What we do in this case is we create a new output node that outputs yhat. The second example is similar: we take a neural network that has been trained using speech recognition data and then we apply that knowledge to a wake word/trigger word detection.



Why is this a good idea? Because the learnings a neural network does in the early layers for the original task is transferable to the new task. If we have a lot of data for the new task we can retrain all the layers, if we don't we just retrain the last layer of the neural network. When does it make sense to use transfer learning? When we have a lot of data for the original task and not a lot of data for the new task. If the opposite is true then it does not make sense to use transfer learning because the new task's data samples are more valuable than the small number of samples we have from the previous task.

- Summarizing, transfer learning makes sense when
    - Task A and B have the same input x. In the previous examples we had images as input for the first example and audio as input for the second example.

- We have a lot more data for task A than for task B.
- When the low level features of task A could be helpful for learning task B.

When transfer learning makes sense

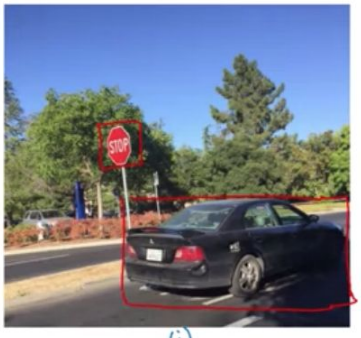
Trans fer A → B

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

Multitask learning

- What is multi-task learning? It's when a neural network is trying to do several things at the same time. Let's illustrate it with an example: Below is an image that is an input for a simplified autonomous driving example. In this case we want our algorithm to identify 4 things: pedestrians, cars, stop signs and traffic lights so in this case $y^{(i)} = \{0,\ 1,\ 1,\ 0\ \}$ and is a (4,1) vector. The Y matrix is then a (4,m) matrix with each $y^{(i)}$ example being a (4,1) vector,

Simplified autonomous driving example



$x^{(i)}$

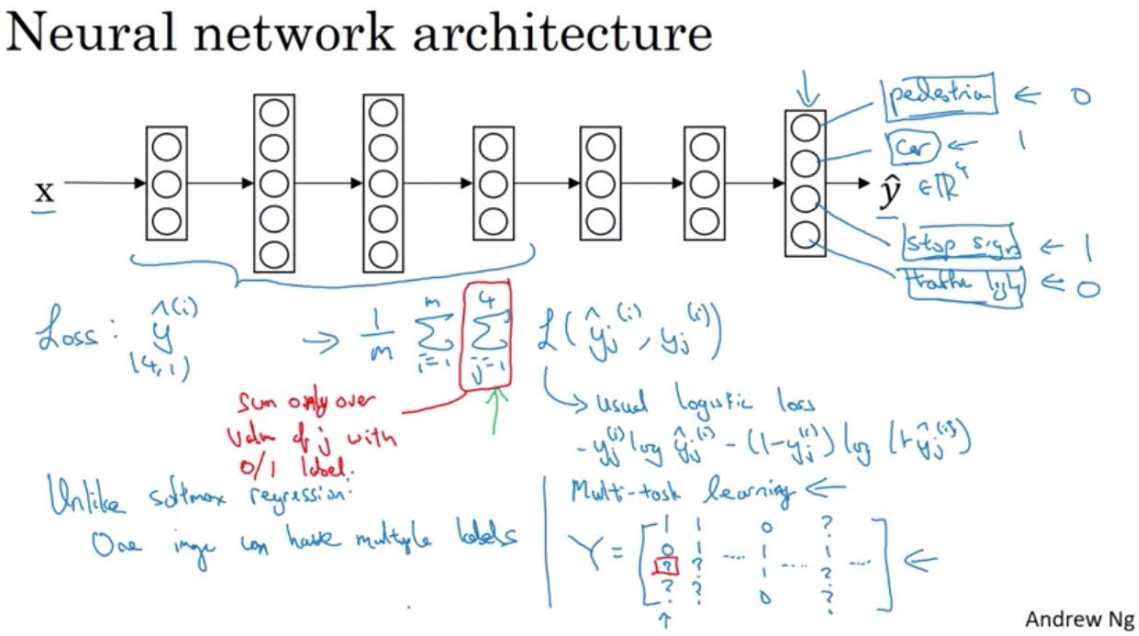
$y^{(i)}$ (4,1)

Pedestrians 0
Cars 1
Stop signs 1
Traffic lights 0

$Y = \begin{bmatrix} | & | & | & & | \\ y^{(1)} & y^{(2)} & y^{(3)} & \cdots & y^{(m)} \\ | & | & | & & | \end{bmatrix}$ (4, m)

Andrew Ng

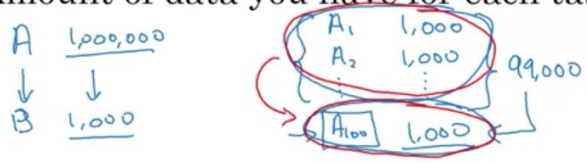
- Now we are going to train a neural network to predict all these y values. Below is a neural network whose output layer has 4 outputs: pedestrian, car, stop sign and traffic light. The loss function is expressed as $1/m * \sum_{i=1}^{m} \sum_{j=1}^{4} L(yhat_j^{(i)}, y_j^{(i)})$ and that loss function is the logistic loss function: $-y_j^{(i)} log(yhat_j^{(i)}) - (1 - y_j^{(i)}) log(1 - yhat_j^{(i)})$. Our neural network is different from softmax regression because one image can have multiple labels, in softmax regression you could only have one label per image. Finally, we can use partially labeled datasets (datasets where we don't have values for all 4 things) here. The only thing we do to account for them is to only sum up the examples where we have a 1 or 0 value. Another track we could have taken is to train 4 neural networks instead of 1 but if the earlier features can be shared between these different objects we will find out that performance is better for one neural network.



- When does multi-task learning make sense? In the following cases: training on a set of tasks that could benefit from having shared lower-level features, when the amount of data between tasks is similar as shown in the 100 task example below and when we can train a big enough neural network that can do well in all tasks. In practice transfer learning is used more frequently than multi-task learning; the one exception is computer vision object detection.

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.



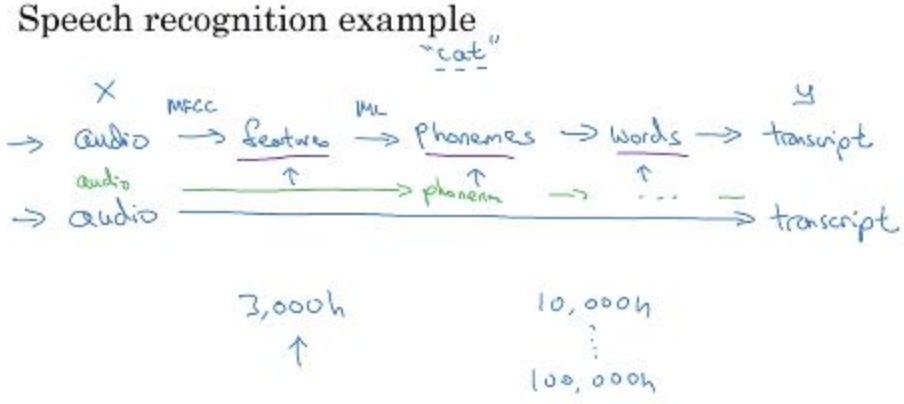
- Can train a big enough neural network to do well on all the tasks.

End to end Deep Learning

- What is end to end deep learning? It is taking a learning algorithm that has multiple stages of data processing and replacing them with a single neural network. Let's look at the example below. In this example we have a speech recognition algorithm that has an audio file as an input, has 3 intermediate stages and outputs a transcript y. It can be replaced with a neural out that outputs a transcript. If we have a small amount of data it is better to use the traditional way. If we have a large amount of data (say, 100,000 hours) it's better to use the end to end approach.
- t

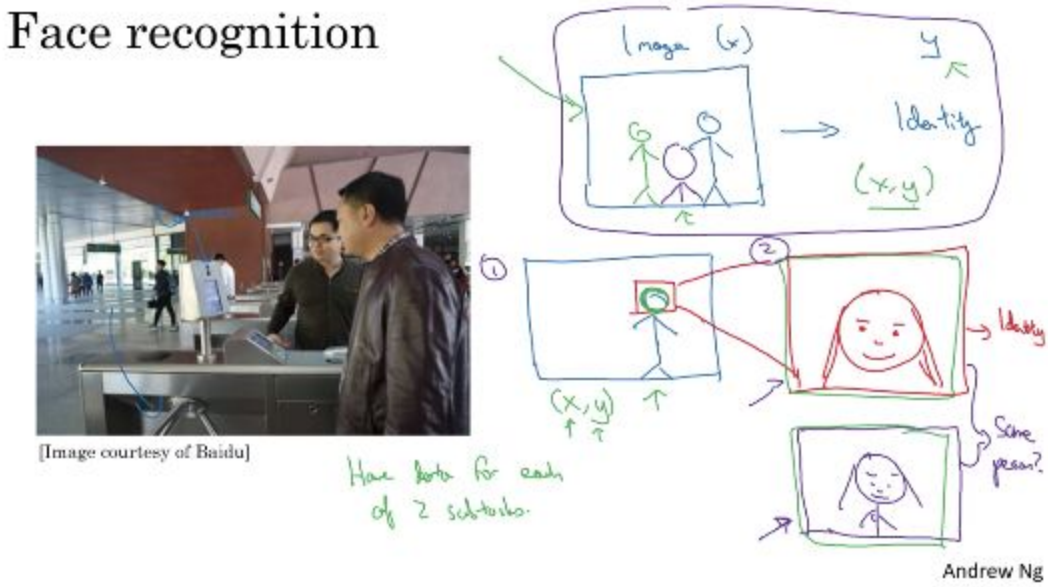
What is end-to-end learning?

Speech recognition example

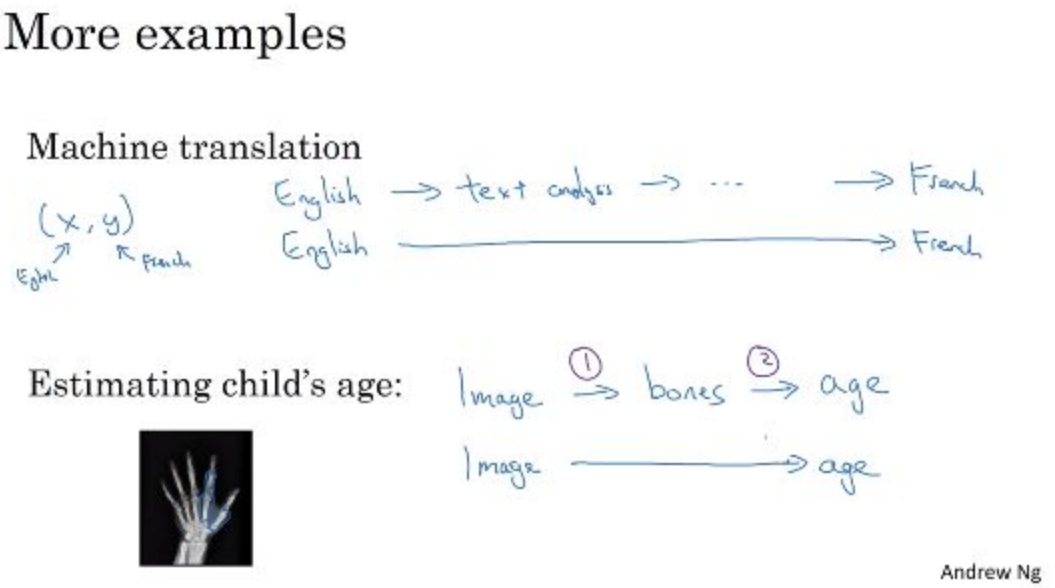


- Let's look at a face recognition example: we have a turnstile with a camera and its job is to recognize faces and if they belong to an employee then allow them to enter the building. We could use the end to end approach and take the original image x and then try to find identity y but it turns out that is not the best approach. In this case it is better

to break the problem in 2 steps: the first step recognizes the face in the original image, crops it and then feeds that image to the 2nd step. The 2nd steps compares that image to the list of faces of the company employees and if it finds a match grants access. Why does the multi step approach works better? Because it turns out we have a lot of data for each of the 2 steps and we don't have a lot of data for the end to end approach.

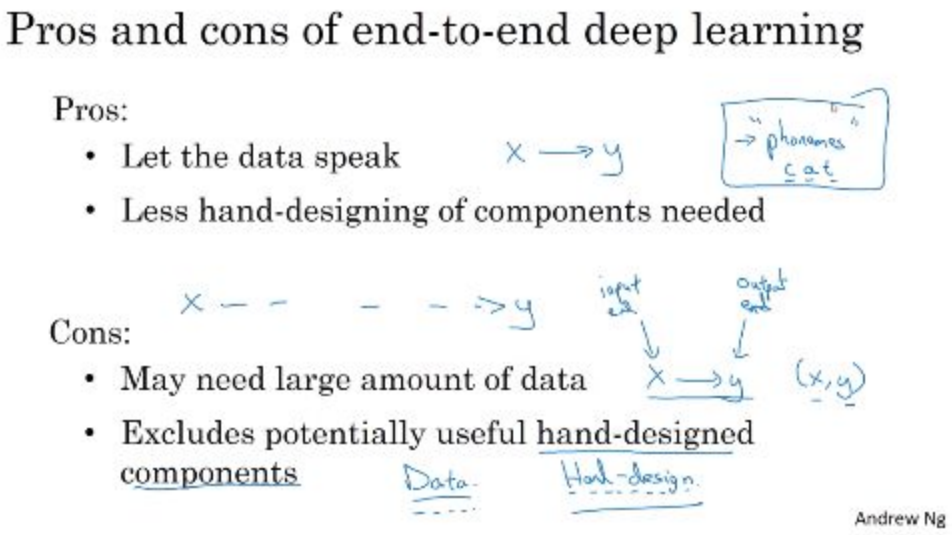


- Let's take a look at a couple of more examples: the first one is machine translation where the end to end approach works very well because we have a lot of data that matches an english sentence to a french sentences. The second example is different: it is estimating a child's age given an x ray of his/her hand. In this case the multi step approach works better because we have more data for each step and we don't have a lot of data for the end to end approach.



Whether to use End to end Deep Learning

- What are the pros and cons of end to end deep learning. The pros are that it lets the data speak and there is less of a need for hand designed components. The cons are that it may need a large amount of data and it may exclude potentially useful hand designed components.



- The key question is then if we have enough data to learn a function of the complexity needed to map x to y. A final example is an autonomous driving example. In this case the end to end approach does not work for the same reason as before: we have more data for the individual steps than for the end to end approach.