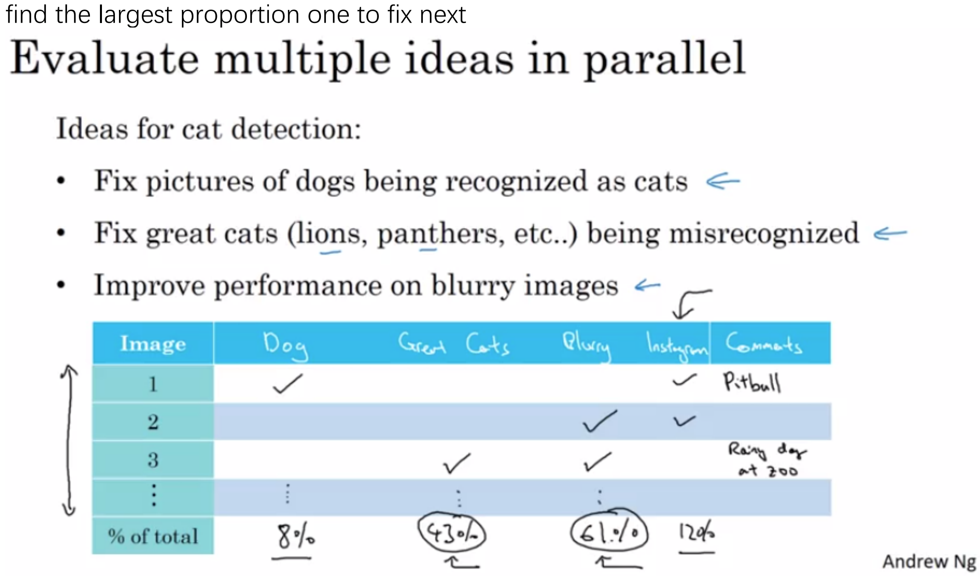
deep Learning_c3_week2

Created: 19/02/19 07:21 **Updated:** 19/02/19 09:00

- # error analysis
    - ## manually examining mistakes
        - process
            - randomly get some mislabeled examples from dev set.
            - count up what kind of mistake took place. (use a spreadsheet to help)
            - find the largest proportion one to fix next

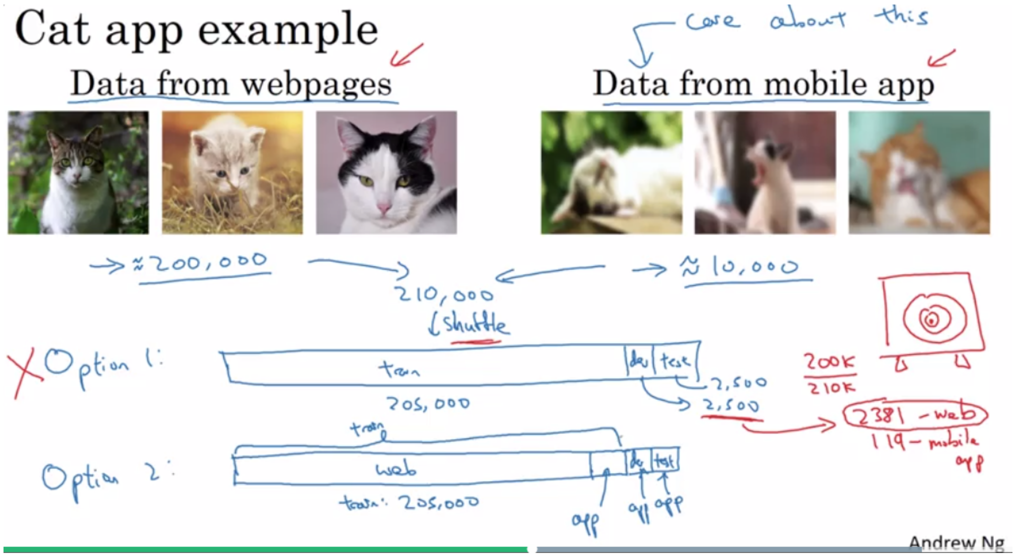


- ## clean up incorrectly labeled data
    - if in the **training** set
        - **random** mistake: let it go
            - DL algorithm is quite robust to random errors.
        - systematic mistake: carefully check and correct it.
    - if in the dev/test set
        - add a **"mislabeled" column** to error analysis spreadsheet to see its influence.
            - EX.

Image	Dog	Great Cats	blurry	Mislabeled	Comments
1	√				
2	√		√		
3					
4		√			
....					
% totals	8%	43%	61%	6%	

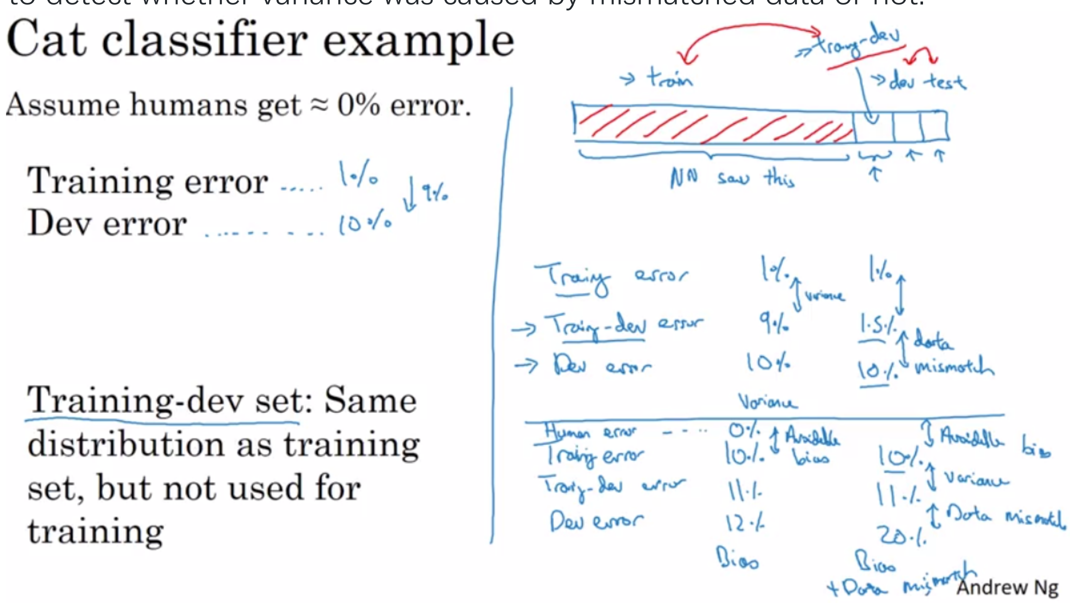
        - guidelines
            1. apply same process to dev and test set to make sure they still come from same distribution.
            2. it could be OK for a train set come from slightly different distribution.
    - build your first system quickly, then iterate

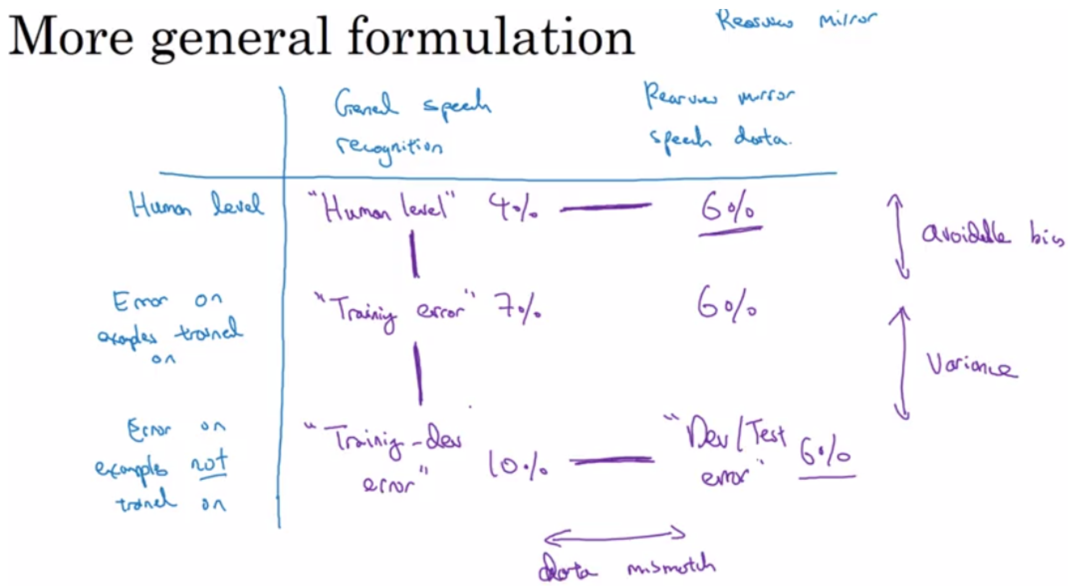
- # mismatched training and dev/test set
    - why: **the hunger for more data**. Sometimes we don't have enough data that exactly match our task. We may use other data to expand our data set. (like data by crawling from internet)
    - ## how to split data set:

- 

- use our **original task data as dev/test set**, then take some of the dev/test set examples and add them to the training set.
  - Advantages: the distribution you care about is your **target** now.
  - Disadvantage: the distributions in training and dev/test sets are different. But you will get a better performance over a long time.

bias and variance with mismatched data distributions

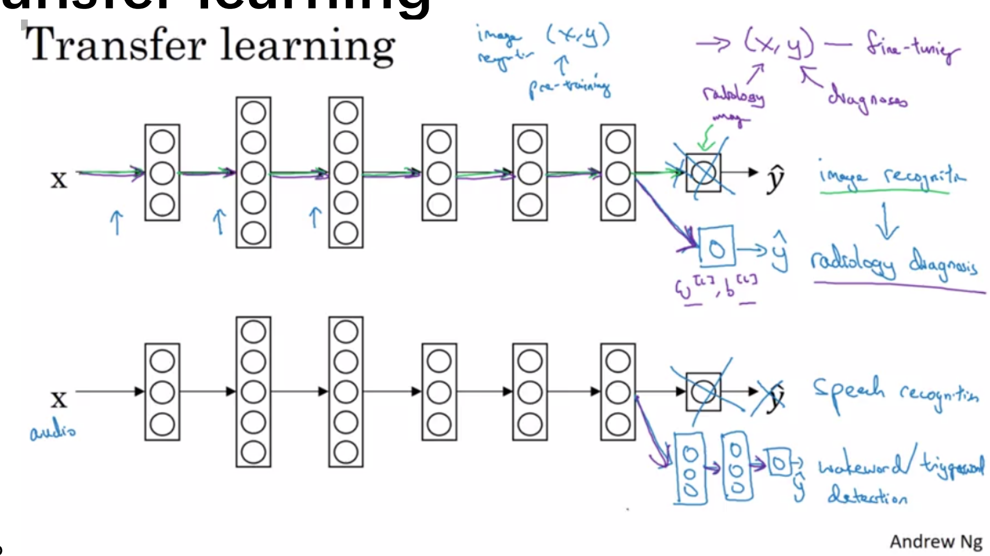
- analysis
  - create a new set called train-dev set as a random subset of the training set (so it has the same distribution)
  - use it to detect whether variance was caused by mismatched data or not.
  - 



- Conclusions:
  - Human-level error (proxy for Bayes error)
  - Train error
    - Calculate `avoidable bias = training error - human level error`
    - If difference is big then it is **Avoidable bias** problem.
  - Train-dev error
    - Calculate `variance = training-dev error - training error`
    - If difference is big then it is high **variance** problem.
  - Dev error
    - Calculate `data mismatch = dev error - train-dev error`
    - If difference is much bigger then train-dev error its **Data mismatch** problem.
  - Test error
    - Calculate `degree of overfitting to dev set = test error - dev error`
    - Is the difference is big (positive) then maybe you need to find a bigger dev set (dev set and test set come from the same distribution, so the only way for there to be a huge gap here, for it to do much better on the dev set than the test set, is if you somehow managed to overfit the dev set).

- address data mismatch
  - unfortunately, there is no systematic way to solve it.
  - guideline
    - using error analysis to understand the difference between training and dev/test sets.
    - Make training data more similar, or collect more data similar to dev/test sets.
      - Ex. **artificial data synthesis.**

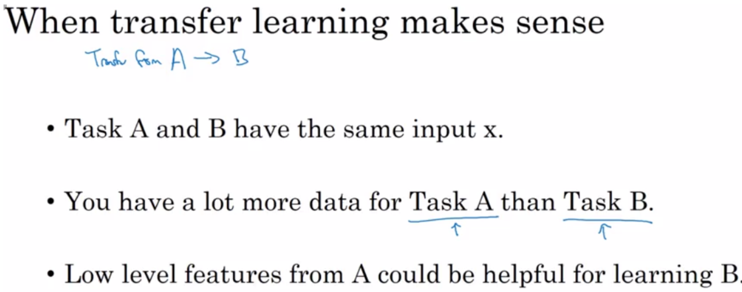
Transfer learning

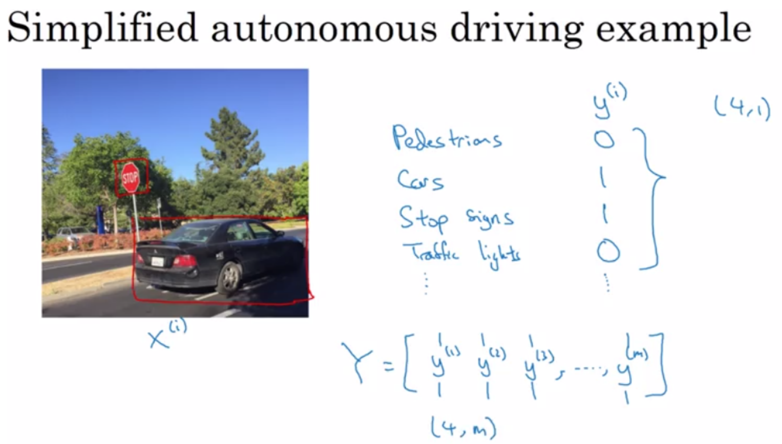


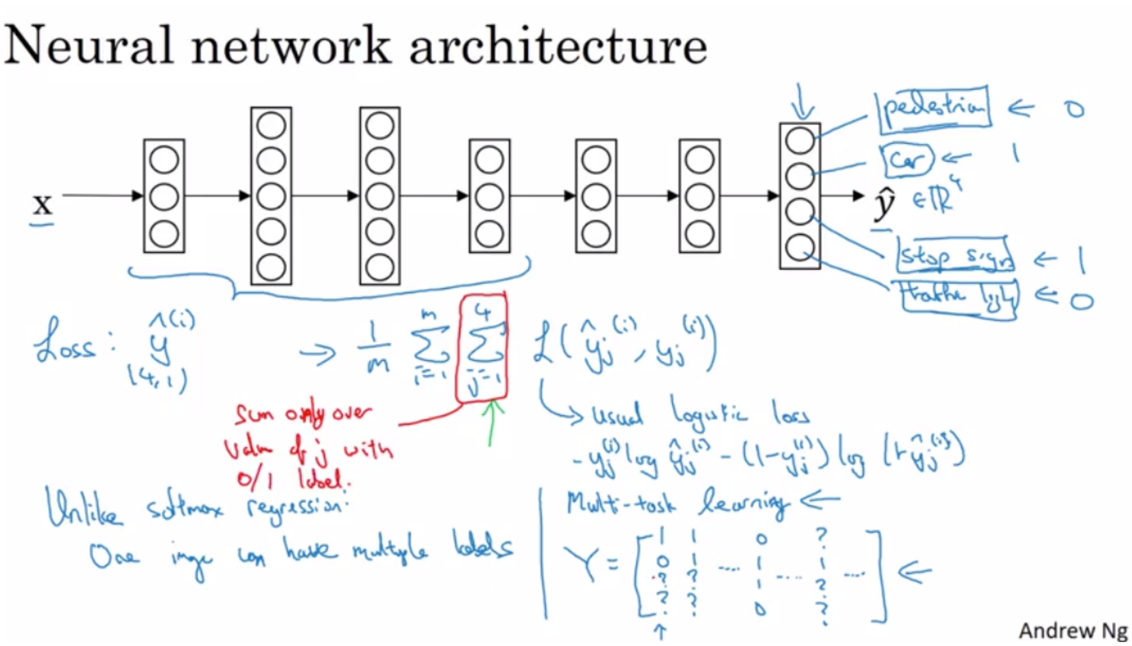
- To do transfer learning, delete the last layer of NN and it's weights and:

i. Option 1 **fine-tuning** :
   1. if you have a small data set - keep all the other weights as a fixed weights.
   2. Add a new last layer(-s) and initialize the new layer weights.
   3. feed the new data to the NN and learn the new weights.

ii. Option 2 **pretraining**:
   1. if you have **enough data** you can **retrain** all the weights.

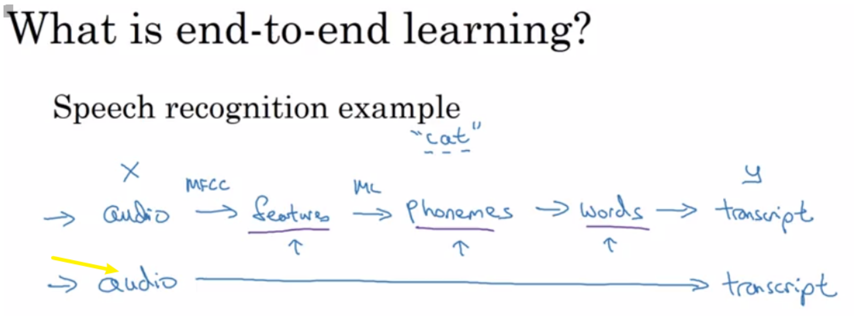
- 

- # multi-task learning
  - not like softmax regression, multi-task learning recognize several things at the same time.
  - Ex.
    - 

  - 

  - Multi-task learning makes sense:
    i. Training on a set of tasks that could benefit from having **shared lower-level** features.
    ii. Usually, **amount** of data you have for each task is quite **similar**.
    iii. Can train a **big enough** network to do well on all the tasks.
  - If you can train a big enough NN, the performance of the multi-task learning compared to splitting the tasks is better.
  - Today transfer learning is used more often than multi-task learning.

- # End-to-end learning

- 

- Pros
  - Let the data speak. NN learning input from X to Y capture whatever statistics are in the data, rather than being forced to reflect human preconceptions.
  - Less hand-designing of components needed.
- Cons
  - need a large amount of data.
  - Excludes potentially useful hand-design components (it helps more on the smaller dataset).
- Applying
  - Key question: Do you have **sufficient data** to learn a function of the **complexity** needed to map x to y?
  - Use ML/DL to learn some **individual** components.
  - When applying supervised learning you should carefully choose what types of X to Y **mappings** you want to learn depending on what task you can get data for.