**Week 2 – Notes**

**Error Analysis**

**Carrying Out Error Analysis**

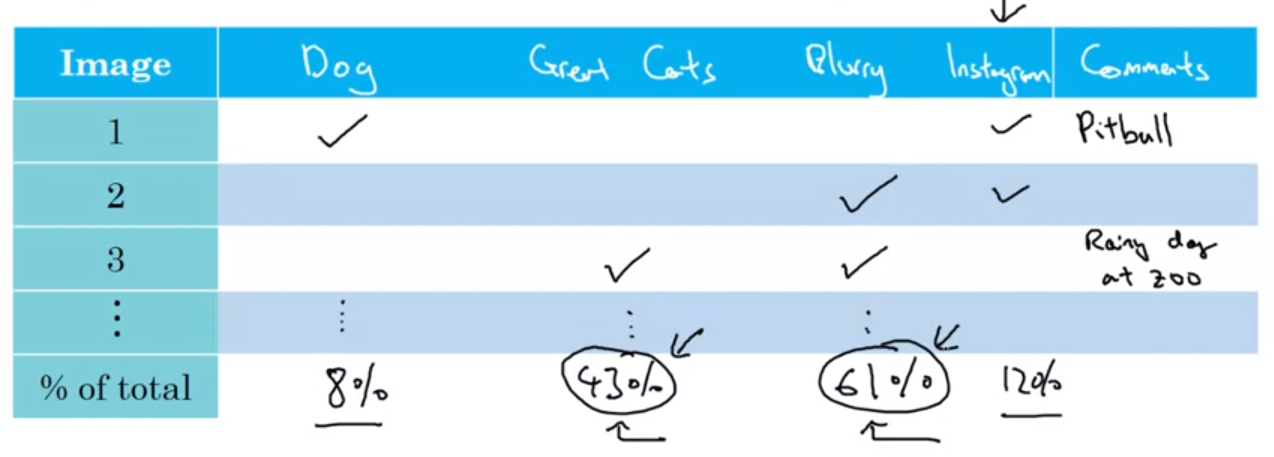
Before trying to fix the error rate by going down one path, you should perform an error analysis

Get ~100 dev set examples that were mislabeled by the classifier

Count how many include the category you want to fix (e.g.: optimize on dogs while building a cat detector)

For example, let’s say that the error is 10% and you find 5 / 100 mislabeled examples that are dogs => the ceiling (maximum improvement) is of 0.5%

If you have multiple improv. ideas, create a table for all of them and perform the error analysis



In this way, based on a 5–10-minute effort, you can take an informed decision

**Cleaning Up Incorrectly Labeled Data**

Usually, deep learning algorithms are quite robust to random errors in the training set and avoids learning from them

However, this error should not be systematic (the labeler mislabeled only white dogs as cats) because the algorithm will learn that white dogs should be labeled as cats

When it comes to the dev data set it’s more sensible

Let’s assume that the overall dev set error is 10%; if the error due to incorrect labels is 0.6% it’s not that much, considering that 9.4% represent errors from other causes

However, if the overall dev set error is 2% and the incorrect labels represent 0.6%, then this would restrict a proper evaluation of different models (the uncertainty of the quality of models is too high)

If we want to correct the incorrect labels of dev and test:

Apply the same process to these data sets so that they continue to come from the same distrib.

Also, consider examining examples your algorithm got right as well

If you correct only dev and test data sets, they can have slightly different distributions compared to the train data set

**Build your First System Quickly, then Iterate**

For a new problem you don’t know with what edge cases to start, so the best approach for new problems is to:

Set up dev / test set and metric

Build initial systems quickly

Use Bias / Variance analysis and Error analysis to prioritize next steps

If you deal with well-known problems, with a lot of literature, then you can start by building something more complex

More teams are overthinking compared to the ones which underestimate the problem (and then build too simple solutions)

**Mismatched Training and Dev/Test Set**

**Training and Testing on Different Distributions**

Let’s day you build a classifier for a mobile app and you have 2 types of images: from webpages (200k) and from mobile app (10k)

You can create your train / dev / test sets in 2 ways:

1. Combine the data sources, shuffle the images and create the sets
2. Use all web images + 5k from users for training, and the rest of 5k from users for dev and test

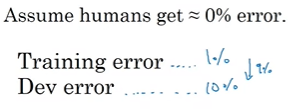
The first case isn’t ideal because you optimize your model to predict only a small portion of images that actually will be seen in production

The second option is better, because you optimize your model to classify images that actually will be in production, even though the train and dev / test distributions are different; we rely on the model’s capability to generalize; additionally, there’s the option to use all images from user only for dev and test sets

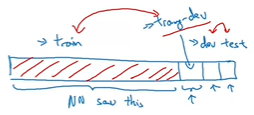
**Bias and Variance with Mismatched Data Distributions**

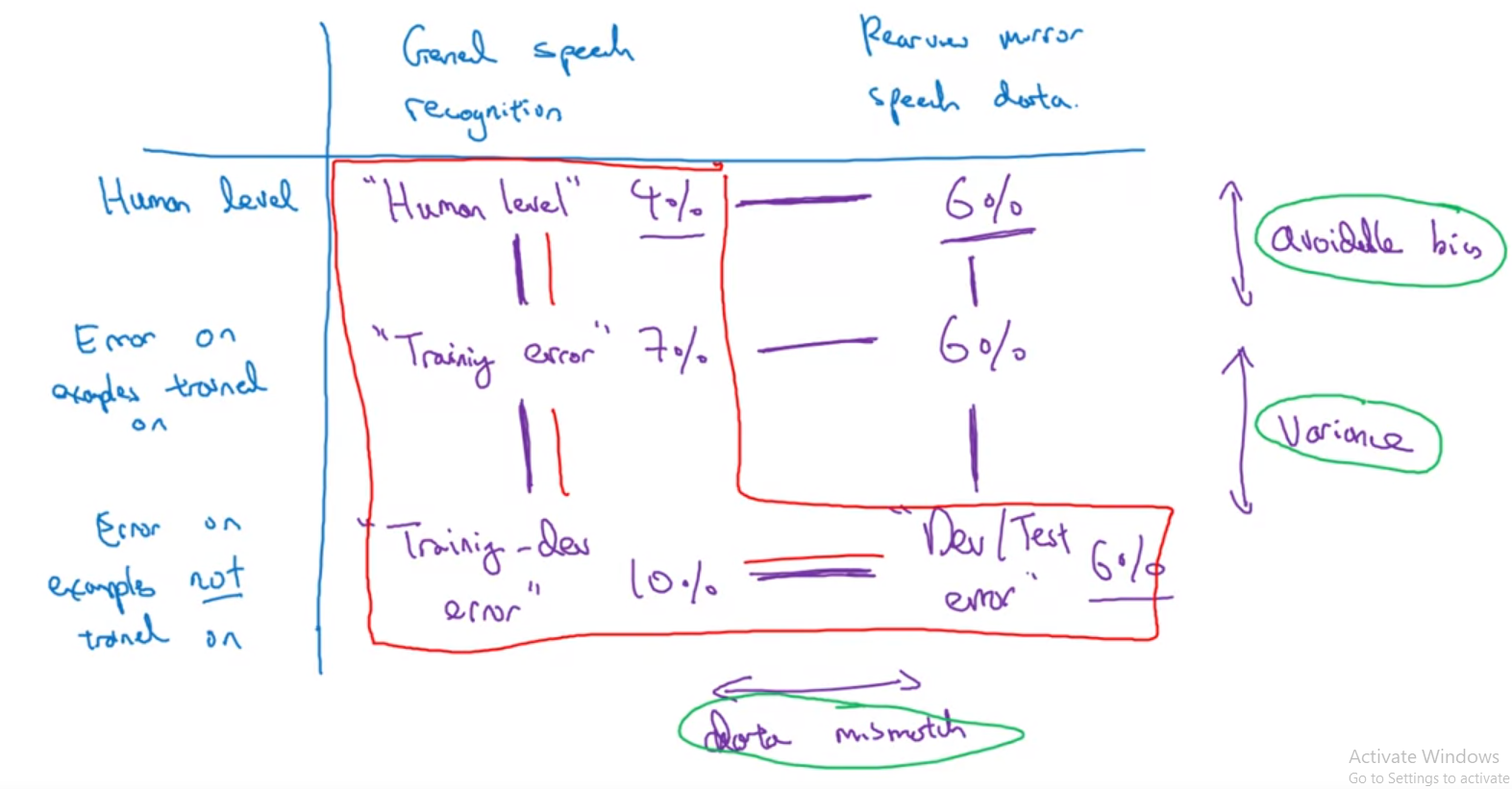
Let’s say you train a classifier on a train set and test the performance on a dev set that has a different distribution

When checking the results, you don’t know whether this is a variance problem or a data mismatch one (training data is easier than the dev one)



To solve this unknown issue, you have to create a training-dev set, which is a part of the train set; train the model on the train and test it on the train-dev, dev and test





Diagnosing the problems:

Training error >> human error => avoidable bias

Training dev error >> training error => variance problem

Dev error >> training dev error => data mismatch problem

Test error >> dev error => overfitting the dev set

**Addressing Data Mismatch**

There aren’t systematic ways to solve this issue

You have to carry out manual error analysis to try to understand difference between training and dev/test sets (e.g.: dev/test sets have a noisy background noise)

Make training data more similar by augmenting it or collect more data similar to dev/test sets

For example, let’s say we have 10k hours of speech in the training set and we want to augment it to include background noise, but we have only 1 hour of background noise => out of the possible background noises, we add only a subset, therefore, the network may overfit the augmented data; in this case, would be better to collect more background noise before augmenting the data

Takeaway: pay attention when you augment your data, so that it includes many cases, not only a small subset

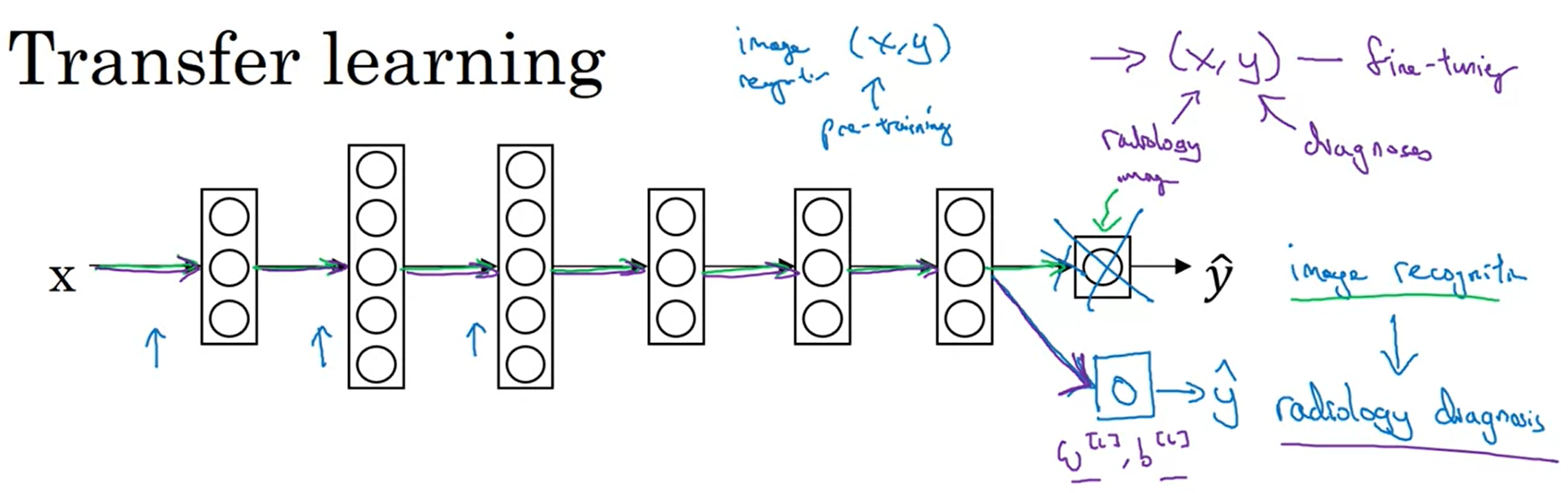
**Learning from Multiple Tasks**

**Transfer Learning**

Let’s say you want to train a classifier to label x-ray images

You would want to use a pre-trained classifier that recognizes images because it has a lot of knowledge about detecting features

You need to replace the last layer and then randomly initialize the weights which are fed into it



If you have a huge data set, then retrain the entire network and consider the pre-trained weights as the initialized values

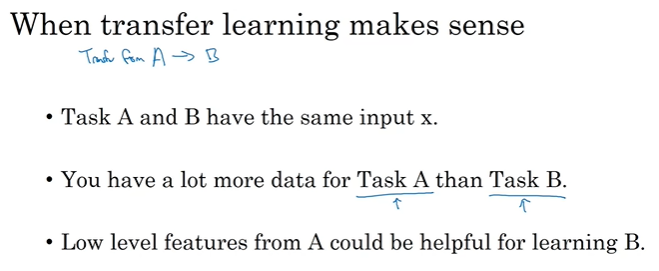
If you have a small set, keep all the weights expect the ones of the last 1 or 2 layers

This process of tuning the weights for a specific task is called fine-tuning

Additionally, instead of just replacing the last layer, you can add layers at the end of the network

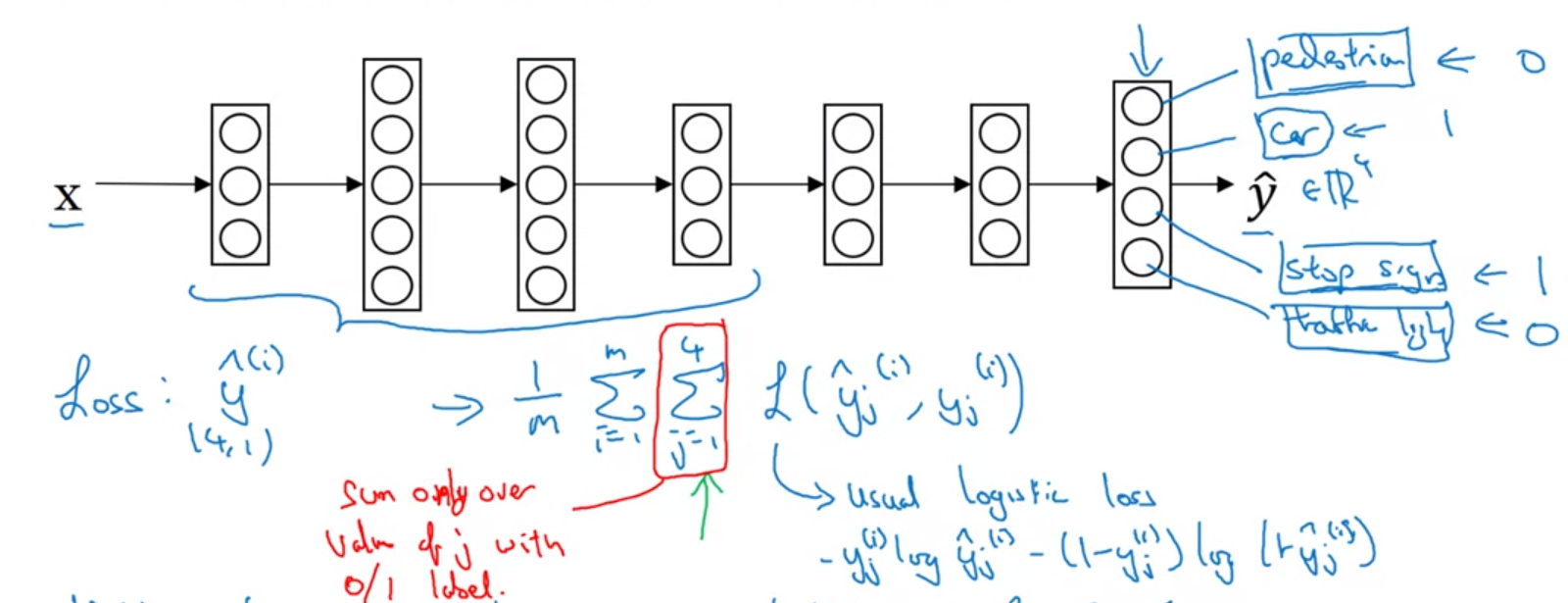
Transfer learning is useful when you have a much smaller data set for fine-tuning compared to the one used for pre-training

Transfer learning isn’t useful in the other case, because the data for the fine-tuning task is much more valuable compared to the one used for pre-training



**Multi-task Learning**

Train a network that has multiple outputs

These models are extremely common in computer vision, for example, train a model which predicts if in the image are pedestrians, cars, etc. (all at once) 

The cost function is the sum of a simple logistic loss (binary cross entropy loss in our case) for each output value

It’s interesting that even if we don’t have for each image a label for each output, we can still train the network because in the loss function we omit that output for which we don’t have a label

Multi-task learning makes sense when:

Training on a set of tasks that could benefit from having shared lower-level features; in this case is better to have only one network than several models specialized on different tasks

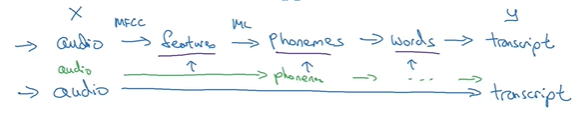
The amount of data you have for each task is similar (for example instead of training 100 models to solve 100 classification problems on 1,000 for each task, it would be better to solve these problems at once; thus, the last task can benefit from the 99k images of the first 99 tasks)

Can train a big enough NN to do well on all tasks (the only way a multi-task model is worse than multiple single-task models is when the first one isn’t big enough)

**End-to-end Deep Learning**

**What is End-to-end Deep Learning?**

Traditionally, a system like a speech recognizer is implemented in several steps, but when there is a lot of data available, we can use an end-to-end approach in which you predict the transcript based only on the input audio



For a face recognition system, if there isn’t enough data to solve a hard problem like detecting if a person is in the data base, from a picture of the whole scene, it’s better to split the problem in several steps like: finding the face and just then to check the identity by comparing the face image with the ones from the data base

Thus, we have to be careful when choosing to use an end-to-end approach or it’s better to split the problem in multiple steps that can also be solved by using DL

Some problems cannot be solved with an end-to-end approach; for example, detecting the child’s age based on an x-ray image of his hand

That’s because we don’t have sufficient pairs of images and predictions and the problem is too hard to solve; better is to detect the bones and then based on statistics of their size to predict he child’s age

**Whether to use End-to-end Deep Learning**

Pros:

Let the data speak: A possible problem of the traditional approach is that phonemes are extracted, that perhaps do not represent the best way of generating transcripts; sometimes is better to let the NN to extract the useful features by itself

Less hand-designing of components needed

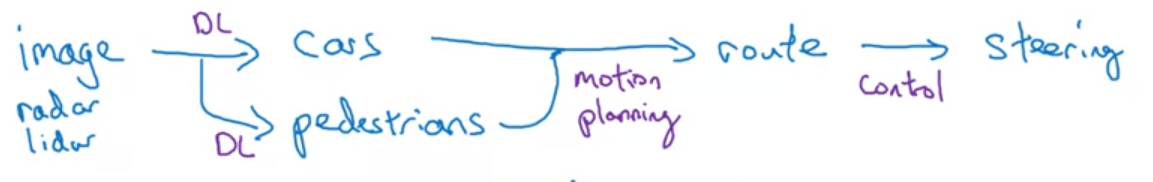
Cons:

May need large amount of data

Excludes potentially useful hand-designed components: which are harder to extract automatically from small data sets

Apply this approach based on the following question: do you have sufficient data to learn a function of the complexity needed to map x to y?

For instance, building an autonomous car based only on images is much harder than splitting the problem in multiple steps, some of them being solvable by using DL



**Assignment**

In the multi-task learning approach, the activation function used in the last layer is the Sigmoid

Synthetic images shouldn’t be added to the dev or test sets because they don’t represent our target in a completely accurate way