

Addressing data mismatch

This is a general guideline to address data mismatch:

- Perform manual error analysis to understand the error differences between training, development/test sets. Development should never be done on test set to avoid overfitting.
- Make training data or collect data similar to development and test sets. To make the training data more similar to your development set, you can use artificial data synthesis. However, it is possible that if you might be accidentally simulating data only from a tiny subset of the space of all possible examples.

Bias and variance with mismatched data distributions

Example: Cat classifier with mismatch data distribution

When the training set is from a different distribution than the development and test sets, the method to analyze bias and variance changes.

	Classification error (%)					
	Scenario A	Scenario B	Scenario C	Scenario D	Scenario E	Scenario F
Human (proxy for Bayes error)	0	0	0	0	0	4
Training error	1	1	1	10	10	7
Training-development error	-	9	1.5	11	11	10
Development error	10	10	10	12	20	6
Test error	-	-	-	-	-	6

Scenario A

If the development data comes from the same distribution as the training set, then there is a large variance problem and the algorithm is not generalizing well from the training set.

However, since the training data and the development data come from a different distribution, this conclusion cannot be drawn. There isn't necessarily a variance problem. The problem might be that the development set contains images that are more difficult to classify accurately.

When the training set, development and test sets distributions are different, two things change at the same time. First of all, the algorithm trained in the training set but not in the development set. Second of all, the distribution of data in the development set is different.

It's difficult to know which of these two changes what produces this 9% increase in error between the training set and the development set. To resolve this issue, we define a new subset called training-development set. This new subset has the same distribution as the training set, but it is not used for training the neural network.

Scenario B

The error between the training set and the training-development set is 8%. In this case, since the training set and training-development set come from the same distribution, the only difference between them is the neural network sorted the data in the training and not in the training development. The neural network is not generalizing well to data from the same distribution that it hadn't seen before

Therefore, we have really a variance problem.

Scenario C

In this case, we have a mismatch data problem since the 2 data sets come from different distribution.

Scenario D

In this case, the avoidable bias is high since the difference between Bayes error and training error is 10 %.

Scenario E

In this case, there are 2 problems. The first one is that the avoidable bias is high since the difference between Bayes error and training error is 10 % and the second one is a data mismatched problem.

Scenario F

Development should never be done on the test set. However, the difference between the development set and the test set gives the degree of overfitting to the development set.

General formulation

Bayes error



Avoidable Bias

Training set error



Variance

Development - Training set error



Data mismatch

Development set error



Degree of overfitting to the development set

Test set error

Build system quickly, then iterate

Depending on the area of application, the guideline below will help you prioritize when you build your system.

Guideline

1. Set up development/ test set and metrics
 - Set up a target
2. Build an initial system quickly
 - Train training set quickly: Fit the parameters
 - Development set: Tune the parameters
 - Test set: Assess the performance
3. Use Bias/Variance analysis & Error analysis to prioritize next steps

Multi-task learning

Multi-task learning refers to having one neural network do simultaneously several tasks.

When to use multi-task learning

- 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 tasks

Example: Simplified autonomous vehicle

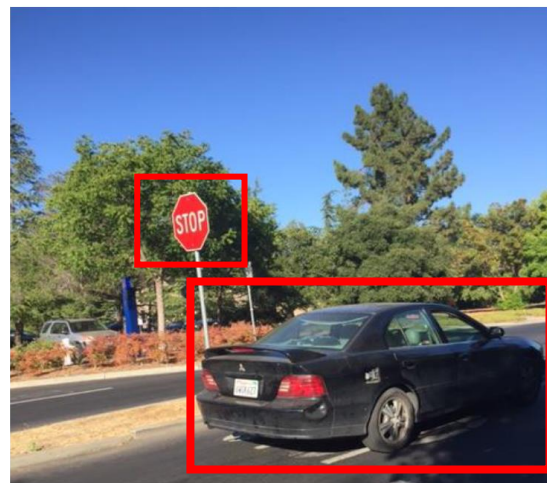
The vehicle has to detect simultaneously several things: pedestrians, cars, road signs, traffic lights, cyclists, etc. We could have trained four separate neural networks, instead of train one to do four tasks. However, in this case, the performance of the system is better when one neural network is trained to do four tasks than training four separate neural networks since some of the earlier features in the neural network could be shared between the different types of objects.

The input $x^{(i)}$ is the image with multiple labels

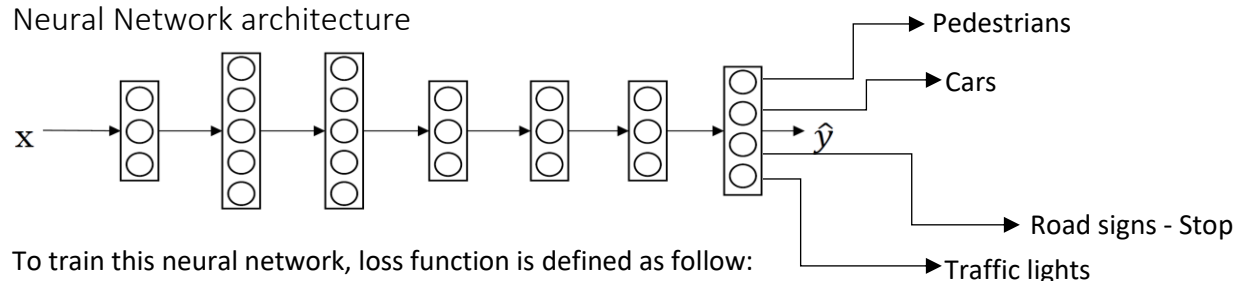
The output $y^{(i)}$ has 4 labels which are represents:

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \begin{matrix} \text{Pedestrians} \\ \text{Cars} \\ \text{Road signs - Stop} \\ \text{Traffic lights} \end{matrix}$$

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



Neural Network architecture



To train this neural network, loss function is defined as follow:

$$-\frac{1}{m} \sum_{i=1}^m \sum_{j=1}^4 \left(y_j^{(i)} \log(\hat{y}_j^{(i)}) + (1 - y_j^{(i)}) \log(1 - \hat{y}_j^{(i)}) \right)$$

Also, the cost can be compute such as it is not influenced by the fact that some entries are not labeled.

Example:

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

Training and testing on different distributions

Example: Cat vs Non-cat

In this example, we want to create a mobile application that will classify and recognize pictures of cats taken and uploaded by users.

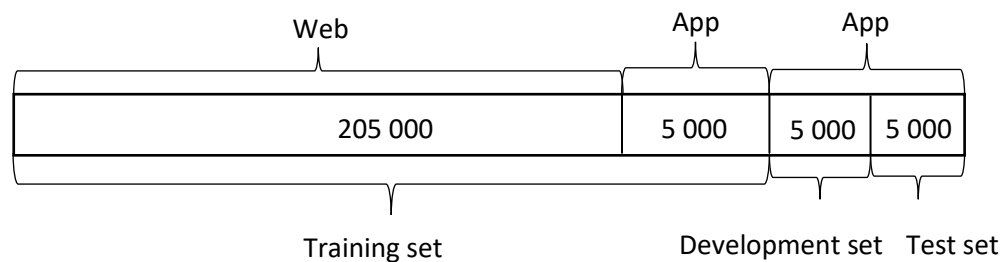
There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded 200 000 pictures where cat's pictures are professionally framed and in high resolution.

The problem is that you have a different distribution:

- 1- small data set from pictures uploaded by users. This distribution is important for the mobile app.
- 2- bigger data set from the web.

The guideline used is that you have to choose a development set and test set to reflect data you expect to get in the future and consider important to do well.

The data is split as follow:



The advantage of this way of splitting up is that the target is well defined.

The disadvantage is that the training distribution is different from the development and test set distributions. However, this way of splitting the data has a better performance in long term.

Transfer Learning

Transfer learning refers to using the neural network knowledge for another application.

When to use transfer learning

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

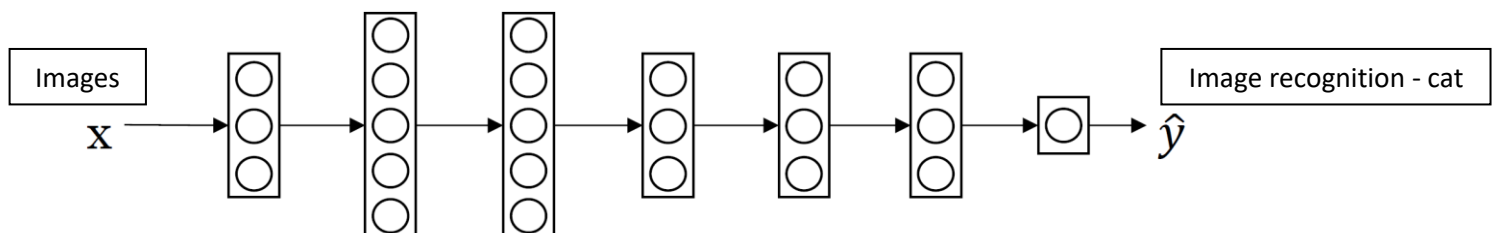
Example 1: Cat recognition - radiology diagnosis

The following neural network is trained for cat recognition, but we want to adapt it for radiology diagnosis. The neural network will learn about the structure and the nature of images. This initial phase of training on image recognition is called pre-training, since it will pre-initialize the weights of the neural network. Updating all the weights afterwards is called fine-tuning.

For cat recognition

Input x : image

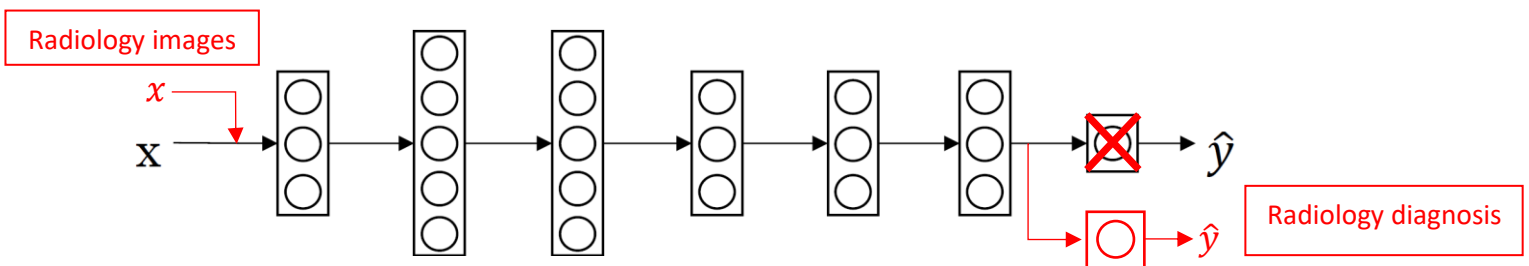
Output y – 1: cat, 0: no cat



Radiology diagnosis

Input x : Radiology images – CT Scan, X-rays

Output y : Radiology diagnosis – 1: tumor malign, 0: tumor benign



Guideline

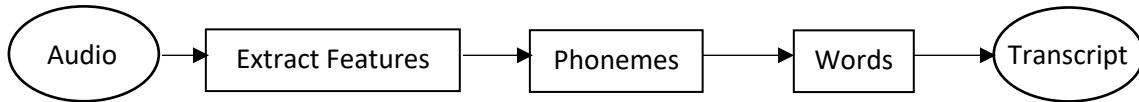
- Delete last layer of neural network
- Delete weights feeding into the last output layer of the neural network
- Create a new set of randomly initialized weights for the last layer only
- New data set (x, y)

What is end-to-end deep learning

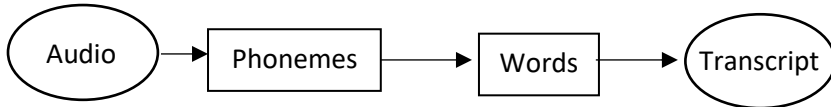
End-to-end deep learning is the simplification of a processing or learning systems into one neural network.

Example - Speech recognition model

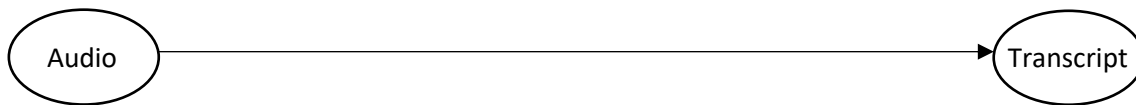
The traditional way - small data set



The hybrid way - medium data set



The End-to-End deep learning way – large data set



End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc.

Whether to use end-to-end deep learning

Before applying end-to-end deep learning, you need to ask yourself the following question: Do you have enough data to learn a function of the complexity needed to map x and y ?

Pro:

- Let the data speak
 - By having a pure machine learning approach, the neural network will learn from x to y . It will be able to find which statistics are in the data, rather than being forced to reflect human preconceptions.
- Less hand-designing of components needed
 - It simplifies the design work flow.

Cons:

- Large amount of labeled data
 - It cannot be used for every problem as it needs a lot of labeled data.
- Excludes potentially useful hand-designed component
 - Data and any hand-design's components or features are the 2 main sources of knowledge for a learning algorithm. If the data set is small than a hand-design system is a way to give manual knowledge into the algorithm.