Carrying out Error Analysis

An example: A cat-vs-non-cat classifier reaches 90% accuracy and 10% error.

My supervisor points out that there are dogs misclassified as cats when testing.

And he asks me to make this cat classifier do better on dogs.

Hehe. Should I listen to him and retrain a better cat classifier?

How do I know it's worth my effort? Use Error Analysis.

Models think it's a cat, but it's a dog actually.

Error Analysis:

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs. Eg: 5 out of 100 are dogs. → Don't retrain specifically on dogs. Not worth it.
  50 out of 100 are dogs. → Worth the effort. 50% improvement.

Evaluate multiple ideas in parallel.

Ideas for cat detection:

- Fix pictures of dogs being recognized as cats.
- Fix great cats (lions, panthers, etc...) being misrecognized.
- Improve performance on blurry images.

Image	Dogs	Great Cats	Blurry	Instagram Filters	...
1	✓			✓	
2		✓	✓		
3		✓	✓		
⋮					
% of total	8%	43%	61%	12%	

This table shows that I should put my effort on Great Cats and Blurry images as it can improve the performance the most.

If I spend time on Dogs images, the optimal improvement is only 8% which is not much.

Incorrectly labeled examples in training set.

x							
y	1	0	1	1	0	1	1

Incorrect Label

If there is random error in the training data, is it worth fixing the data?

DL algorithms are quite robust to random errors in the training set. Notice that if it's systematic error, such as all white dogs are labeled as cats, then it's a problem.

Error analysis in Dev/Test set.

Now, the question is that is it worthwhile going in to fix up this 6% of incorrectly labeled data.

Image	Dog	Great Cat	Blurry	Incorrectly labeled	Comments
…					
98				✓	Labeler missed cat in background
99		✓			
100				✓	Drawing of a cat; Not a real cat.
% of total	8%	43%	61%	6%	

Overall dev set error

Assume 10% error

Errors due incorrect labels 0.6% $\frac{0.6}{10}$=6% error

Errors due to other causes 9.4%

Assume 2% error

Errors due incorrect labels 0.6% $\frac{0.6}{2}$=30% error could have been avoided. It's too much. I should fix it.

Errors due to other causes 1.4%

Goal of dev set is to help you select between two classifiers A & B.

Andrew Ng

Some guidelines to be considered when correcting incorrect Dev/Test set examples.

Recall target planning and shooting.

- Apply some process to your dev and test sets to make sure they continue to come from the same distribution.
- Consider examing examples my algorithm got right as well as ones it got wrong.
- Train and Dev/Test data may now come from slightly different distributions.

Consider correcting training data as well.

Build your First System Quickly, then Iterate

Speech Recognition example:

- Noisy background
  - Cafe noise
  - Car noise
- Accented Speech
- Far from microphone
- Young children's speech
- Stuttering

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.

Mismatched Training and Dev/Test Set

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.

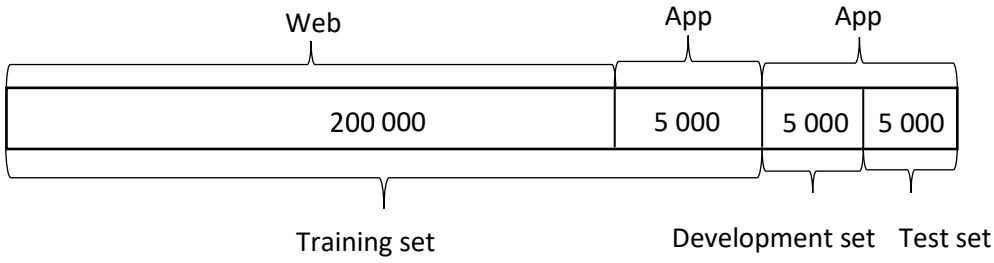
There are two sources of data used to develop the mobile app. The first data distribution is small, 10 000 pictures uploaded from the mobile application. Since they are from amateur users, the pictures are not professionally shot, not well framed and blurrier. The second source is from the web, you downloaded 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 best option:

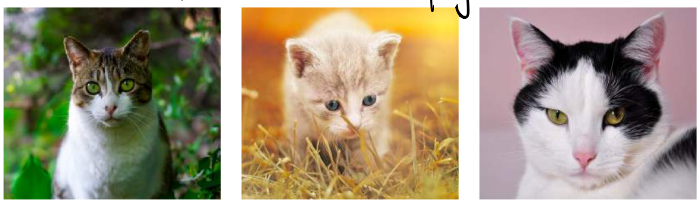


The advantage of this way of splitting up is that the target is well defined.

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.

What we really care about.

Data from Webpages



Data from Mobile App

The bad choice: Combine 200,000 images from webpages and 10,000 images from mobile app, and shuffle randomly.

200000 + 10000 = 210000 → shuffle and split randomly.

Train	Dev	Test
205000	2500	2500

In Dev set, there are 2381 images from web and 119 images from mobile. It's a BAD Dev set. Images from web are not important AT ALL for Dev/Test/APP. Thus, putting web images in Dev doesn't make any sense. The metric of Dev set will be dominated by web images.

Definition of Training-Development Set:
It has the same data distribution as training set, but not used for training.

Bias and variance with mismatched data distributions

Example: Cat classifier with mismatch data distribution Cat images from Web and Mobile APP

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 (High Variance Problem)

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 (Training Data and Dev/Test Data from mismatched distribution)

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

Scenario D (High Bias Problem)

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 (The Dev and Test set are actually easier data.)

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 (Human-Level 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

Remember the Dev set and the test set always have the same data distribution. If overfitting the Dev set, we can get more Dev set data.

More General Formulation: (Using Rearview Mirror as an example.)

Rearview Mirror

	General Speech Data	Rearview Mirror Data
Human Level	"Human level" 4%	6%
Error on examples trained on	"Traing error" 7%	6%
Error on examples Not trained on	"Traing-Dev" 10% error	"Dev/Test Error" 6%

Data Mismatch

Training:
- Purchased Data
- Smart Speaker Control
- Voice Keyboard

Dev/Test
- Speech Activated Rearview Mirror

Addressing Data Mismatch

1. Carry out manual error analysis to try to understand difference between training and dev/test sets.

   E.g., noisy in-car data in Dev set.

2. Make training data more similar; or collect more data similar to dev/test sets.

   E.g., simulate noisy in-car data

Artificial Data Synthesis:

Clean audio + Car noise = Synthesized in-car audio

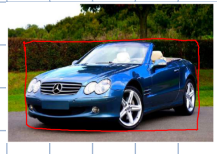
10000 hours 1 hour

Notice that using only 1 hour car noise could result to overfitting to this 1 hour car noise.

Thus, it will be better to use unique 10000 hours real world car noise to synthesize data.

Another example: Car Recognition

More variety of cars shall be synthesized to avoid overfitting.



Transfer Learning

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

When to use transfer learning

When NOT to use transfer learning:
If any one of three is untrue.

- 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

Task A: 10^6 images or 10000h speech
Task B: 10^3 images or 10 hours speech

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.

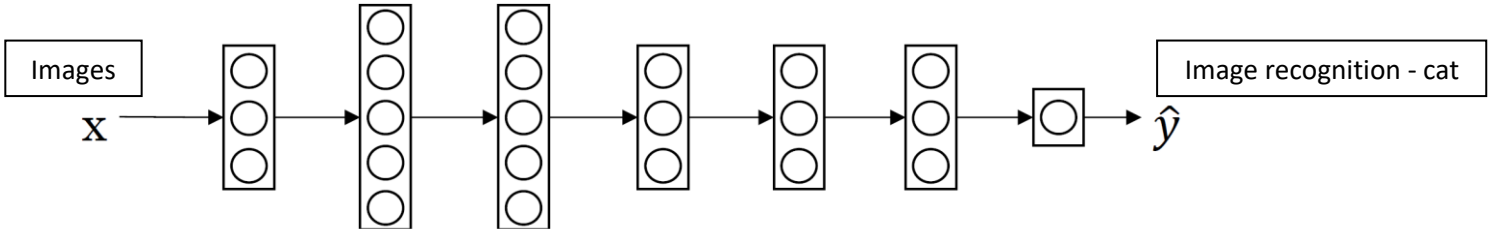
Task A

For cat recognition

Pre-Training

Input x: image

Output y – 1: cat, 0: no cat



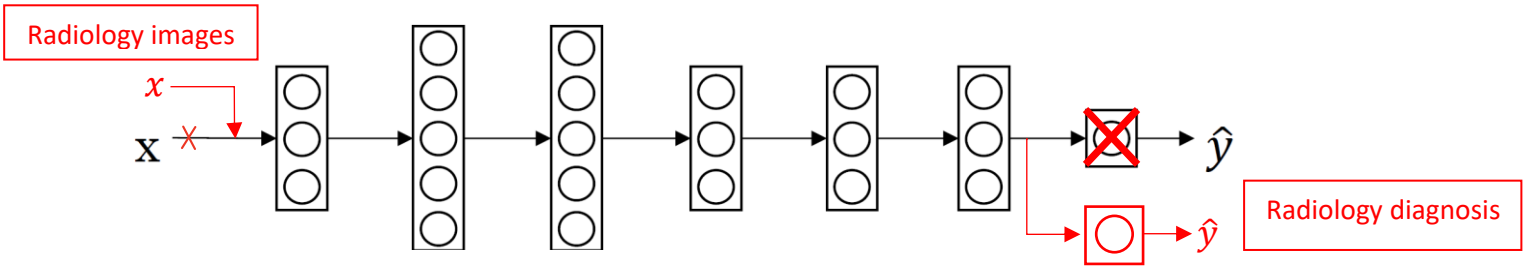
Task B

Radiology diagnosis

Fine-Tuning

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

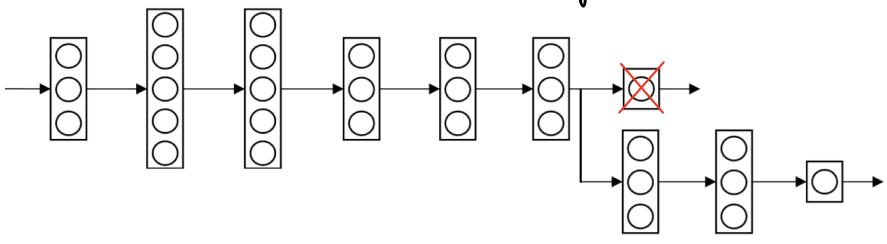
Output y :Radiology diagnosis – 1: tumor malign, 0: tumor benign



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)

More layers can also be added in Transfer Learning.



A_1 10^3 samples
A_2 10^3 samples
⋮
A_{100} 10^3 samples

Eg, there are 100 tasks $(A_1, \dots, A_{100})$. Each task has 1000 samples. The number of samples might look few for each task, but Andrew says that other 99 tasks can provide knowledge to help the current task.

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

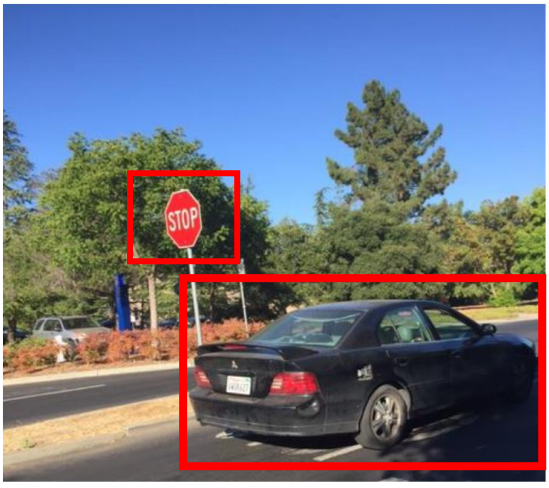
Example: Simplified autonomous vehicle

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:

$$y^{(i)}_{(4,1)} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Pedestrians
Cars
Road signs - Stop
Traffic lights

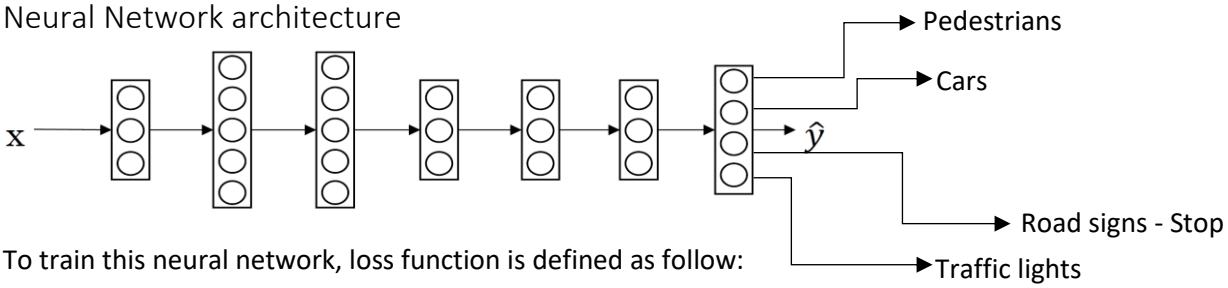


$$Y = \begin{bmatrix} | & | & | & | \\ y^{(1)} & y^{(2)} & y^{(3)} & y^{(4)} \\ | & | & | & | \end{bmatrix}$$

$Y = (4, m)$

$Y = (4,1)$

Neural Network architecture



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

Can the softmax layer be used in the last output layer?

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

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

We can sum only over value of j with 0/1 label, ignore '?'.

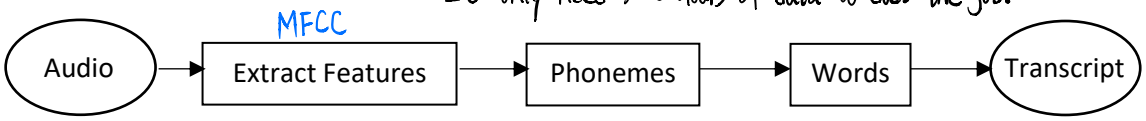
$$Y = \begin{bmatrix} 1 & 0 & ? & ? \\ 0 & 1 & ? & 0 \\ 0 & 1 & ? & 1 \\ ? & 0 & 1 & 0 \end{bmatrix}$$

What is end-to-end deep learning

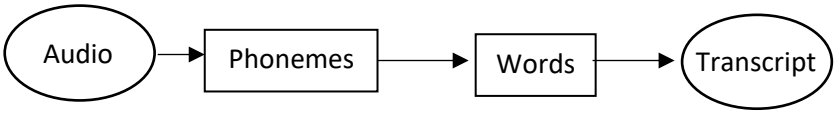
End-to-end deep learning is the simplification of a processing or learning systems into one neural network. This is insanely unreal. How is it possible?

Example - Speech recognition model

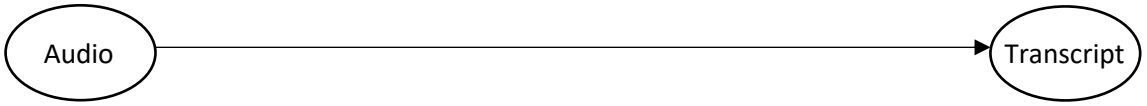
The traditional way - small data set It only need 3000 hours of data to does the job.



The hybrid way - medium data set



The End-to-End deep learning way – large data set 10^5 hours

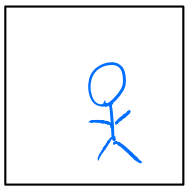


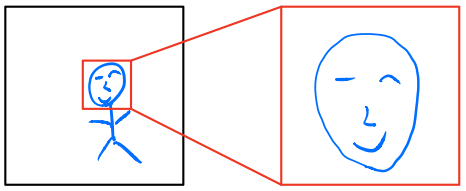
End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc.

I think end-to-end learning is not efficient. It needs 100 to 1000 times or more of data to achieve what traditional method can achieve. So, the hybrid way is the best way. We just need to figure out which step can be replaced by deep learning.

E.g., deep learning-based stereo dense matching algorithm is way better than the traditional method (Semi Global Matching)

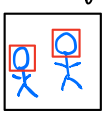
An example: Recognize a person in image. There are two options.

1.  Given an image, tell identity.

2.  Break the problem into two steps. First find the face. Then tell the identity.

Option 2 performs better than option 1. Why?

- Break a hard problem into simpler subproblems. Divide and Conquer.
- There are more data for individual subproblems than the data for the original problem.

 Where is the face?

 Are they the same person?

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. Eg, "c a t" phonemes are artifacts created by human linguists. We shouldn't force our algorithm to think in phonemes. ML algorithms are free to learn whatever representation it wants to learn rather than forcing itself to use phonemes as representation.
- 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.

In general, more data is beneficial to algorithms.

Hand-design features are double-edge sword. It can give meaning information to algorithms, but we also want algorithms to find useful features by itself as long as there are sufficient data.

The key question: Do we have sufficient data to learn a function of the complexity needed to map X to Y?

Define the mapping function X to Y. Then assess the complexity.

Another example: Estimating child's age with radiology image.

Option 1: Image → Bones → Age. Divide and Conquer

Option 2: Image → Age. end-to-end