

Structuring Machine Learning Projects

Jane Huang 04/01/2018


Table of Contents

Week 1	1
1. Motivating examples.....	1
2. Orthogonalization	1
3. Using a single number evaluation metric	2
4. Satisficing and optimizing metric	3
5. Training, development and test distributions	4
6. Size of the development and test sets.....	4
7. When to change development/test sets and metrics	5
8. Why human-level performance?	6
9. Avoidable bias.....	7
10. Understanding human-level performance.....	8
11. Surpassing human-level performance	9
12. Improving your model performance.....	9
Week 2:	10
1. Carrying out error analysis.....	10
2. Cleaning up incorrectly labelled data	11
3. Build system quickly, then iterate.....	12
4. Training and testing on different distributions.....	12
5. Bias and variance with mismatched data distributions.....	14
6. Addressing data mismatch.....	16
7. Transfer Learning	17
8. Multi-task learning.....	18
9. What is end-to-end deep learning	19
10. Whether to use end-to-end deep learning	20

Week 1

1. Motivating examples

Ideas:

- Collect more data 
- Collect more diverse training set
- Train algorithm longer with gradient descent
- Try Adam instead of gradient descent
- Try bigger network
- Try smaller network
- Try dropout
- Add L_2 regularization
- Network architecture
 - Activation functions
 - # hidden units
 - ...

Andrew Ng

2. Orthogonalization

Orthogonalization or orthogonality is a system design property that assures that modifying an instruction or a component of an algorithm will not create or propagate side effects to other components of the system. It becomes easier to verify the algorithms independently from one another, it reduces testing and development time.

When a supervised learning system is design, these are the 4 assumptions that needs to be true and orthogonal.

1. Fit training set well in cost function

- If it doesn't fit well, the use of a bigger neural network or switching to a better optimization algorithm might help.

2. Fit development set well on cost function

- If it doesn't fit well, regularization or using bigger training set might help.

3. Fit test set well on cost function

- If it doesn't fit well, the use of a bigger development set might help

4. Performs well in real world

- If it doesn't perform well, the development test set is not set correctly or the cost function is not evaluating the right thing.

3. Using a single number evaluation metric

To choose a classifier, a well-defined development set and an evaluation metric speed up the iteration process.

Example : Cat vs Non- cat

y = 1, cat image detected

Predict class \hat{y}	Actual class y	
	1	0
1	True positive	False positive
0	False negative	True negative

Precision

Of all the images we predicted y=1, what fraction of it have cats?

$$\text{Precision (\%)} = \frac{\text{True positive}}{\text{Number of predicted positive}} \times 100 = \frac{\text{True positive}}{(\text{True positive} + \text{False positive})} \times 100$$

Recall

Of all the images that actually have cats, what fraction of it did we correctly identifying have cats?

$$\text{Recall (\%)} = \frac{\text{True positive}}{\text{Number of predicted actually positive}} \times 100 = \frac{\text{True positive}}{(\text{True positive} + \text{False negative})} \times 100$$

Let's compare 2 classifiers A and B used to evaluate if there are cat images:

Classifier	Precision (p)	Recall (r)
A	95%	90%
B	98%	85%

Let's compare 2 classifiers A and B used to evaluate if there are cat images:

Classifier	Precision (p)	Recall (r)
A	95%	90%
B	98%	85%

In this case the evaluation metrics are precision and recall.

For classifier A, there is a 95% chance that there is a cat in the image and a 90% chance that it has correctly detected a cat. Whereas for classifier B there is a 98% chance that there is a cat in the image and a 85% chance that it has correctly detected a cat.

The problem with using precision/recall as the evaluation metric is that you are not sure which one is better since in this case, both of them have a good precision et recall. F1-score, a harmonic mean, combine both precision and recall.

$$F1-Score = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

Classifier	Precision (p)	Recall (r)	F1-Score
A	95%	90%	92.4 %
B	98%	85%	91.0%

Classifier A is a better choice. F1-Score is not the only evaluation metric that can be use, the average, for example, could also be an indicator of which classifier to use.

4. Satisficing and optimizing metric

There are different metrics to evaluate the performance of a classifier, they are called evaluation matrices. They can be categorized as satisficing and optimizing matrices. It is important to note that these evaluation matrices must be evaluated on a training set, a development set or on the test set.

Example: Cat vs Non-cat

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1 500 ms

In this case, accuracy and running time are the evaluation matrices. Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example, is the satisficing metric which mean that the metric has to meet expectation set.

The general rule is: (If there are N metrics that you care about, then only choose one as optimizing metrics, all others as satisficing metric).

$$N_{metric} \cdot \begin{cases} 1 & \text{Optimizing metric} \\ N_{metric} - 1 & \text{Satisficing metric} \end{cases}$$

5. Training, development and test distributions

Setting up the training, **development and test sets** have a huge impact on productivity. It is important to choose the development and test sets from **the same distribution** and it must be taken randomly from all the data.

Guideline

Choose a development set and test set to reflect data you expect to get in the future and consider important to do well.

True Story (not recommended)

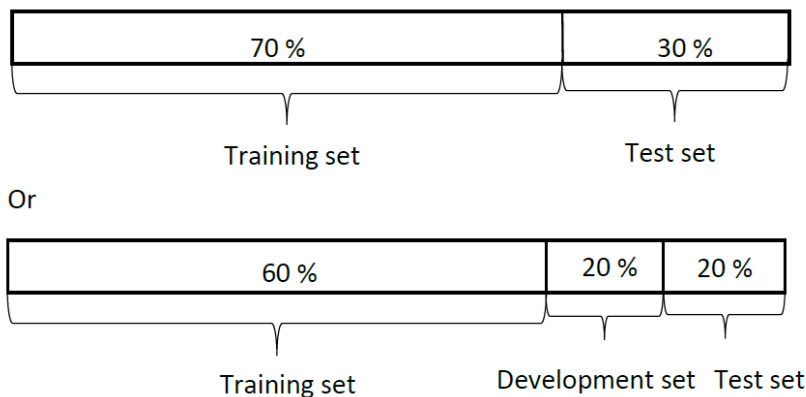
- Optimizing on dev set on loan approvals for medium income zip codes
- Tested on low income zip codes

6. Size of the development and test sets

Sample size example: 100,1000,10000

Old way of splitting data

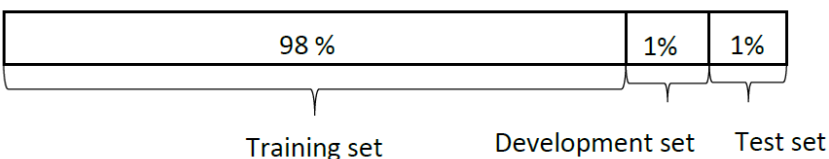
We had smaller data set therefore we had to use a greater percentage of data to develop and test ideas and models.



Sample size example: 1000000

Modern era – Big data

Now, because a large amount of data is available, we don't have to compromise as much and can use a greater portion to train the model.



Guidelines

- Size of dev set

Set your dev set to be big enough to detect differences in algorithm/models you're trying out. The development set has to be big enough to evaluate different ideas.

- Size of test set:
 - Set up the size of the test set to give a high confidence in the overall performance of the system.
 - Test set helps evaluate the performance of the final classifier which could be less 30% of the whole data set.
 - For some applications, maybe you don't need a high confidence in the overall performance of your final system. Maybe all you need is a train and dev set, not having a test set might be okay.

7. When to change development/test sets and metrics

Example: Cat vs Non-cat

A cat classifier tries to find a great amount of cat images to show to cat loving users. The evaluation metric used is a classification error.

Algorithm	Classification error [%]
A	3%
B	5%

It seems that Algorithm A is better than Algorithm B since there is only a 3% error, however for some reason, Algorithm A is letting through a lot of the pornographic 色情 images.

Algorithm B has 5% error thus it classifies fewer images but it doesn't have pornographic images.

From a company's point of view, as well as from a user acceptance point of view, Algorithm B is actually a better algorithm. The evaluation metric fails to correctly rank order preferences between algorithms. The evaluation metric or the development set or test set should be changed.

The misclassification error metric can be written as a function as follow:

$$Error : \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

This function counts up the number of misclassified examples.

The problem with this evaluation metric is that it treats pornographic vs non-pornographic images equally. One way to change this evaluation metric is to add the weight term $w^{(i)}$.

$$w^{(i)} = \begin{cases} 1 & \text{if } x^{(i)} \text{ is non-pornographic} \\ 10 & \text{if } x^{(i)} \text{ is pornographic} \end{cases}$$

The function becomes:

$$Error : \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \mathcal{L}\{\hat{y}^{(i)} \neq y^{(i)}\}$$

Another example

Algorithm A: 3% error

Algorithm B: 5% error

If the pictures in your dev/test set is clear, but the user images are blurry.

If doing well on your metric + dev/test set does not correspond to doing well on your application, change your metric and/or dev/test set.

Guideline

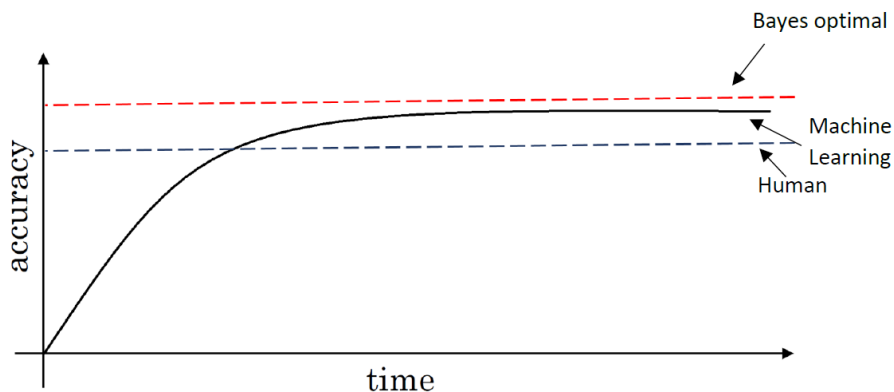
1. Define correctly an evaluation metric that helps better rank order classifiers
2. Optimize the evaluation metric

8. Why human-level performance?

Today, machine learning algorithms can compete with human-level performance since they are more productive and more feasible in a lot of application. Also, the workflow of designing and building a machine learning system, is much more efficient than before.

Moreover, some of the tasks that humans do are close to “perfection”, which is why machine learning tries to mimic human-level performance.

The graph below shows the performance of humans and machine learning over time.



- Bayes optimal error

- Humans
- Machine Learning

Machine learning progresses slowly when it surpasses human-level performance. One of the reason is that human-level performance can be close to Bayes optimal error, especially for natural perception problem.

Bayes optimal error is defined as the best possible error. In other words, it means that any functions mapping from x to y can't surpass a certain level of accuracy.

Also, when the performance of machine learning is worse than the performance of humans, you can improve it with different tools. They are harder to use once it surpasses human-level performance.

These tools are:

- Get labeled data from humans
- Gain insight from manual error analysis: Why did a person get this right?
- Better analysis of bias/variance.

9. Avoidable bias

By knowing what the human-level performance is, it is possible to tell when a training set is performing well or not.

Example: Cat vs Non-Cat

	Classification error (%)	
	Scenario A	Scenario B
Humans	1	7.5
Training error	8	8
Development error	10	10

In this case, the human level error as a proxy for Bayes error since humans are good to identify images. If you want to improve the performance of the training set but you can't do better than the Bayes error otherwise the training set is overfitting. By knowing the Bayes error, it is easier to focus on whether bias or variance avoidance tactics will improve the performance of the model.

Scenario A (focus on reducing bias)

There is a 7% gap (avoidable bias) between the performance of the training set and the human level error. It means that the algorithm isn't fitting well with the training set since the target is around 1%. To resolve the issue, we use bias reduction technique such as training a bigger neural network or running the training set longer.

Scenario B (focus on reducing variance between 8% and 10%)

The training set is doing good since there is only a 0.5% difference with the human level error. The difference between the training set and the human level error is called avoidable bias. The focus here is to reduce the variance since the difference between the training error and the development error is 2%. To resolve the issue, we use variance reduction technique such as regularization or have a bigger training set.

There is more room to reduce the 2% variance than the 0.5% bias error

10. Understanding human-level performance

Human-level error gives an estimate (proxy) of Bayes error.

Example 1: Medical image classification

This is an example of a medical image classification in which the input is a radiology image and the output is a diagnosis classification decision. What is “human-level” error?

	Classification error (%)
Typical human	3.0
Typical doctor	1.0
Experienced doctor	0.7
Team of experienced doctors	0.5

The definition of human-level error depends on the purpose of the analysis, in this case, by definition the Bayes error is lower or equal to 0.5%.

Example 2: Error analysis

	Classification error (%)		
	Scenario A	Scenario B	Scenario C
Human (proxy for Bayes error)	1	1	0.5
	0.7	0.7	
	0.5	0.5	
Training error	5	1	0.7
Development error	6	5	0.8

Scenario A

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 4%-4.5% and the variance is 1%. Therefore, the focus should be on bias reduction technique.

Scenario B

In this case, the choice of human-level performance doesn't have an impact. The avoidable bias is between 0%-0.5% and the variance is 4%. Therefore, the focus should be on variance reduction technique.

Scenario C

In this case, the estimate for Bayes error has to be 0.5% since you can't go lower than the human-level performance otherwise the training set is overfitting. Also, the avoidable bias is 0.2% and the variance is 0.1%. Therefore, the focus should be on bias reduction technique.

Summary of bias/variance with human-level performance

- Human - level error – proxy for Bayes error
- If the difference between human-level error and the training error is bigger than the difference between the training error and the development error. The focus should be on bias reduction technique
- If the difference between training error and the development error is bigger than the difference between the human-level error and the training error. The focus should be on variance reduction technique

11. Surpassing human-level performance

Example1: Classification task

	Classification error (%)	
	Scenario A	Scenario B
Team of humans	0.5	0.5
One human	1.0	1
Training error	0.6	0.3
Development error	0.8	0.4

Scenario A

In this case, the Bayes error is 0.5%, therefore the available bias is 0.1% , and the variance is 0.2%.

Scenario B

In this case, there is not enough information to know if bias reduction or variance reduction has to be done on the algorithm. It doesn't mean that the model cannot be improve, it means that the conventional ways to know if bias reduction or variance reduction are not working in this case.

There are many problems (not a nature perception task, lots of data) where machine learning significantly surpasses human-level performance, especially with structured data:

- Online advertising
- Product recommendations
- Logistics (predicting transit time)
- Loan approvals

Other problems that machine learning may surpass human-level performance

- Speech recognition
- Some image recognition
- Medical: ECG, skin cancer, radiology task

12. Improving your model performance

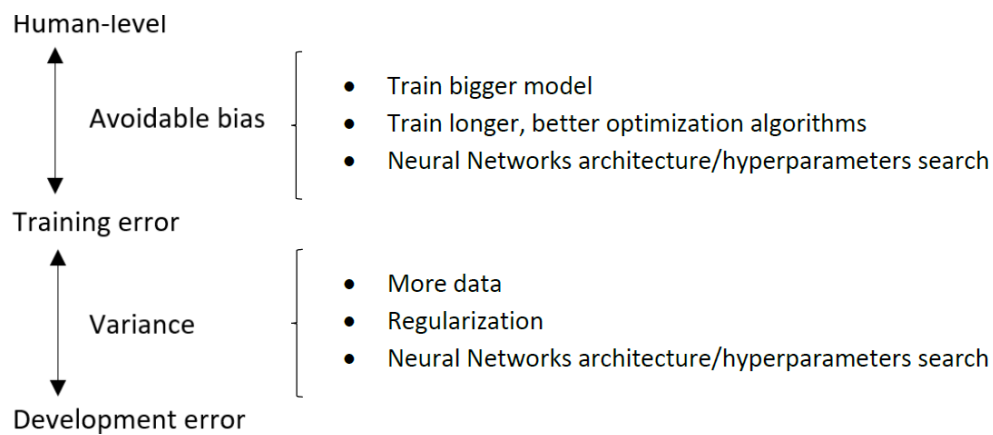
The two fundamental assumptions of supervised learning

There are 2 fundamental assumptions of supervised learning. The first one is to have a low avoidable bias which means that the training set fits well (achieve low avoidable bias). The second one is to have a low or acceptable variance which means that the training set performance generalizes well to the development set and test set (to achieve lower variance).

- If the difference between human-level error and the training error is bigger than the difference between the training error and the development error, the focus should be on bias reduction technique which are training a bigger model, training longer (for deep learning, add momentum, add RMSprop, use better algorithm like Adam) or change the neural networks architecture (try RNN, CNN) or try various hyperparameters search.
- If the difference between training error and the development error is bigger than the difference between the human-level error and the training error, the focus should be on variance reduction technique which are bigger data set, regularization (L2, dropout,

data augmentation) or change the neural networks architecture or try various hyper parameters search.

Summary



Week 2:

1. Carrying out error analysis

Let at dev examples to evaluate ideas

Should you try to make your cat classifier do better on dogs?

Error analysis:

- Get ~100 mislabeled dev set examples.
- Count up how many are dogs.

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	Dog	Great cats	Blurry	Instagram	Comments
1	V			V	
2				V	
3			V		Rainy day at zoo
...		V	V		
% of total	8%	43%	61%	12%	

As you're part way through this process, sometimes you notice other categories of mistakes. So, for example, you might find that Instagram style filter, those fancy image filters, are also messing up your classifier. In that case, it's actually okay, part way through the process, to add another column like that. For the multi-colored filters, the Instagram filters, and the Snapchat filters. And then go through and count up those as well, and figure out what percentage comes from that new error category.

The conclusion of this process gives you an estimate of how worthwhile it might be to work on each of these different categories of errors. For example, clearly in this example, a lot of the mistakes we made on blurry images, and quite a lot on were made on great cat images. And so the outcome of this analysis is not that you must work on blurry images. This doesn't give you a rigid mathematical formula that tells you what to do, but it gives you a sense of the best options to pursue. It also tells you, for example, that no matter how much better you do on dog images, or on Instagram images. You at most improve performance by maybe 8%, or 12%, in these examples. Whereas you can to better on great cat images, or blurry images, the potential improvement. Now there's a ceiling in terms of how much you could improve performance, is much higher. So depending on how many ideas you have for improving performance on great cats, on blurry images. Maybe you could pick one of the two, or if you have enough personnel on your team, maybe you can have two different teams. Have one work on improving errors on great cats, and a different team work on improving errors on blurry images.

So to summarize, to carry out error analysis, you should find a set of mislabeled examples, either in your dev set, or in your development set. And look at the mislabeled examples for false positives and false negatives. And just count up the number of errors that fall into various different categories. During this process, you might be inspired to generate new categories of errors, like we saw. If you're looking through the examples and you say gee, there are a lot of Instagram filters, or Snapchat filters, they're also messing up my classifier. You can create new categories during that process. But by counting up the fraction of examples that are mislabeled in different ways, often this will help you prioritize. Or give you inspiration for new directions to go in. Now as you're doing error analysis, sometimes you notice that some of your examples in your dev sets are mislabeled. So what do you do about that? Let's discuss that in the next video.

2. Cleaning up incorrectly labelled data

Deep learning algorithms are quite robust to random errors in the training set, less robust to systematic errors.

Error analysis:

Image	Dog	Great cats	Blurry	Incorrectly labeled	Comments
...					
98				V	Labeler missed cat in background
99		V			
100				V	Drawing of a cat; not a real cat
% of total	8%	43%	61%	6%	

Scenario 1: No, it is not worth your time

Overall dev set error: 10%

Errors due incorrect labels: $6\% \times 10\% = 0.6\%$

Errors due to other causes: 9.4%

Scenario 2: Yes, it may worth your time

Overall dev set error: 2%

Errors due incorrect labels:0.6%

Errors due to other causes: 1.4%

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

My advice is, if it makes a significant difference to your ability to evaluate algorithms on your dev set, then go ahead and spend the time to fix incorrect labels. But if it doesn't make a significant difference to your ability to use the dev set to evaluate cost buyers, then it might not be the best use of your time.

Correcting incorrect dev/test set examples

- Apply same process to your dev and test sets to make sure they continue to come from the same distribution
- Consider examining examples your algorithm got right as well as ones it got wrong.
- Train and dev/test data may now come from slightly different distributions. (it is okay)

3. Build system quickly, then iterate

Speech recognition example:

- Noisy background
 - Café noise
 - Car noise
- Accented speech
- Far from microphone
- Young children's speech
- Stuttering: uh, ah, um....
- ...

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

4. Training and testing on different distributions

Example: Cat vs Non-cat

Data from webpages



Data from mobile app



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:

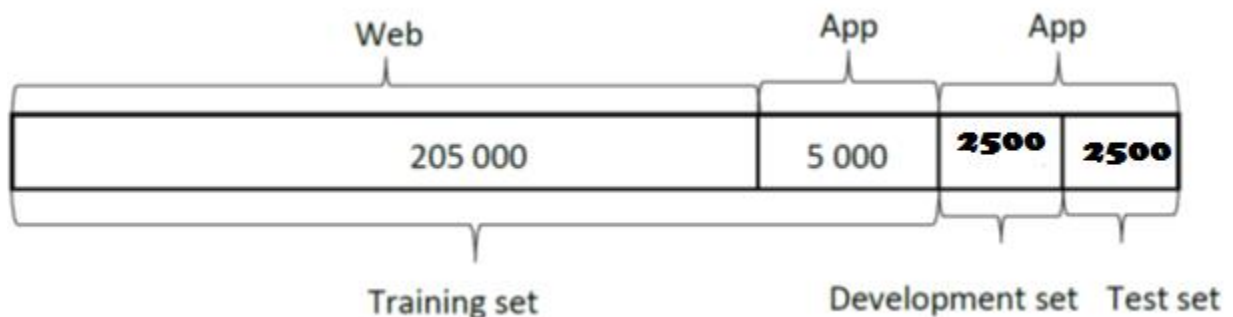
$200,000 + 10,000 = 210,000$

Option 1: (not recommended)

after shuffle, train 205000, dev 2500, test 2500

Big disadvantage: Among 2500, only 2381 from web, only 119 from model app

Option 2: (recommended)



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.

Another example: Speech activated rearview mirrors for a car



Training

- Purchased data from various speech recognition problems
- Smart speaker control
- Voice keyboard (500k)

Dev/Test

- Speech activated rearview mirrors (smaller datasets, very different distributions)
- (20k)

Train	Dev	Test
500k + 10 k	5 k	5k

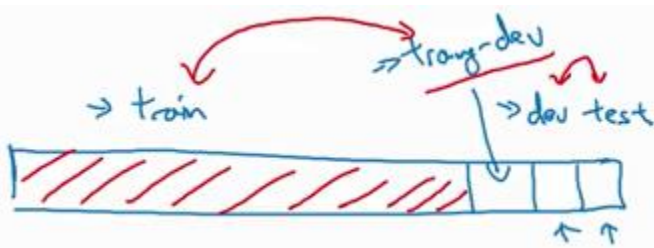
5. Bias and variance with mismatched data distributions

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

Training-dev set: same distribution as training set, but no used for training.



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 %.

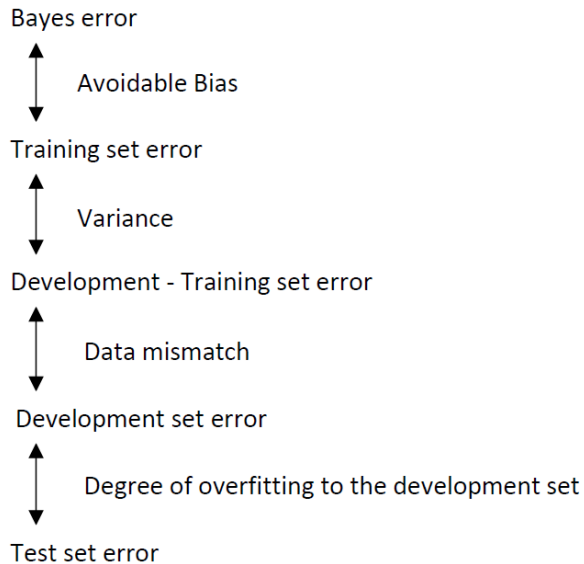
Scenario E

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



	General speech recognition	Rearview mirror speech data	
Human level	"human level" 4%	6%	Avoidable bias
Error on examples trained on	"training error" 7%	6%	
Error on examples not trained on	"training-dev error" 10%	"Dev/test error" 6%	Variance
	Data mismatch		

6. 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.

e.g.: noisy – car noise

Street number

- 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.

e.g. Simulate noisy in-car data

Artificial data synthesis

"The quick brown fox jumps over the lazy dog."

+ Car noise = Synthesized in-car

7. Transfer Learning

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

When to use transfer learning

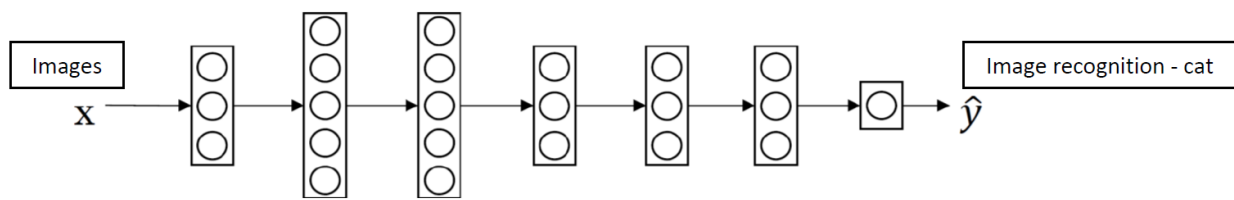
- Task A and B have the same input x
- A lot more data for Task A than Task B
- Low level features from Task A could be helpful for Task B

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

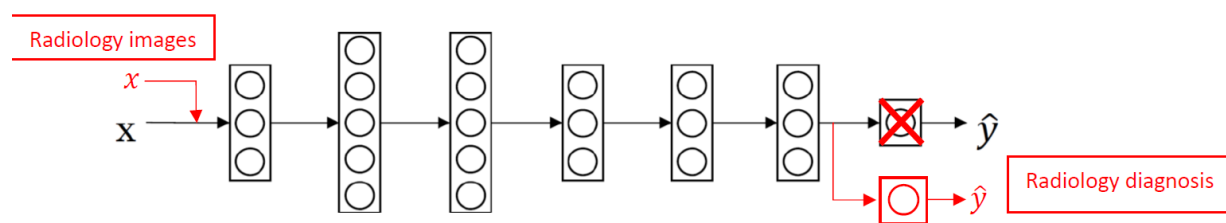


Image recognition: 1,000,000

Radiology diagnosis: 100

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)

And the rule of thumb is maybe if you have a small data set, then just retrain the one last layer at the output layer. Or maybe that last one or two layers.

But if you have a lot of data, then maybe you can retrain all the parameters in the network. And if you retrain all the parameters in the neural network, then this initial phase of training on image recognition is sometimes called **pre-training**, because you're using image recognitions data to pre-initialize or really pre-train the weights of the neural network.

And then if you are updating all the weights afterwards, then training on the radiology data sometimes that's called **fine tuning**. So you hear the words pre-training and fine tuning in a deep learning context, this is what they mean when they refer to pre-training and fine tuning weights

8. 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

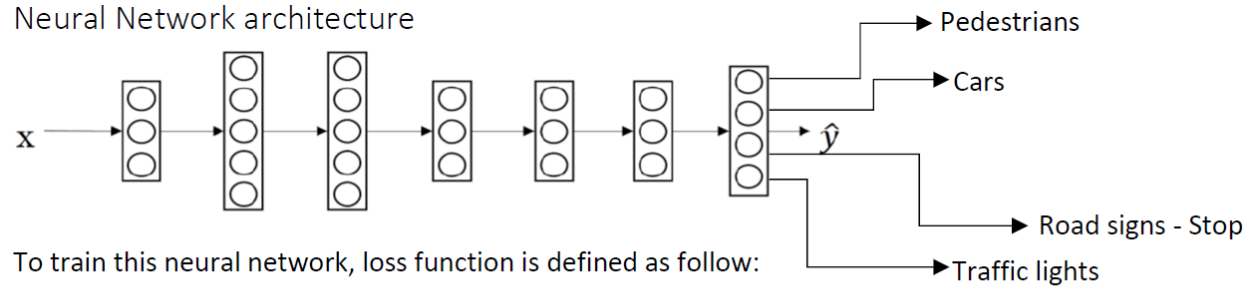
The output $y^{(i)}$ has 4 labels which are represents:

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

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



Neural Network architecture



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

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

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

9. What is end-to-end deep learning

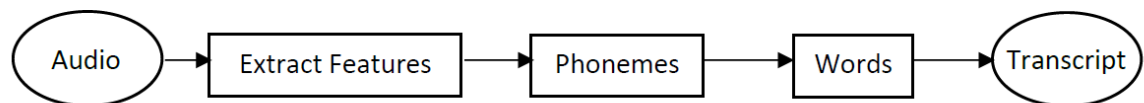
End-to-end deep learning is the simplification of a processing or learning systems into one neural network.

Example - Speech recognition model

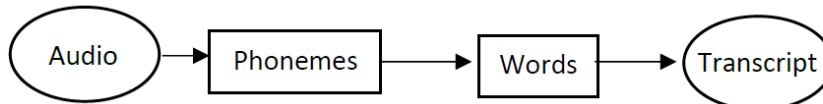
X: audio

Y: transcript

The traditional way - small data set



The hybrid way - medium data set



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



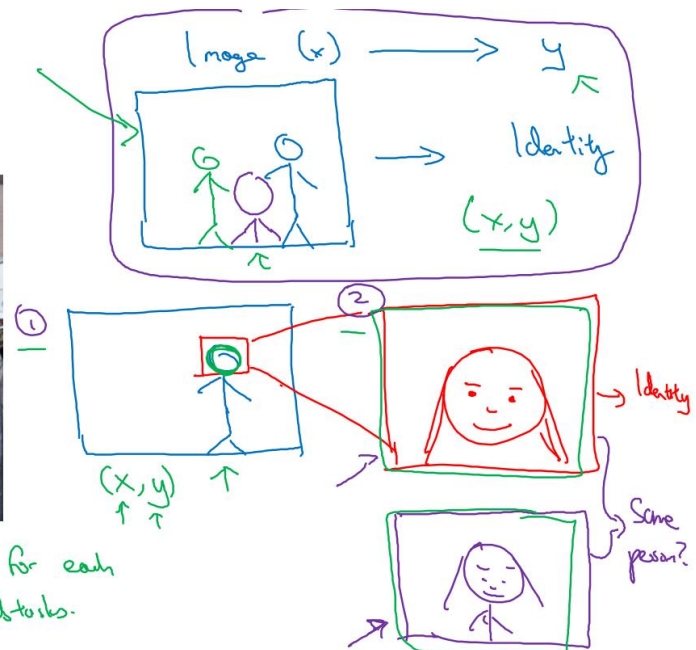
End-to-end deep learning cannot be used for every problem since it needs a lot of labeled data. It is used mainly in audio transcripts, image captures, image synthesis, machine translation, steering in self-driving cars, etc.

Face recognition



[Image courtesy of Baidu]

Have data for each of 2 sub-tasks.



More examples:

Machine translations

X: English

Y: French

English \rightarrow text analysis \rightarrow \rightarrow French

English \rightarrow French (better)

Estimating child's age

Image \rightarrow bones \rightarrow age (better)

Image \rightarrow age

10. 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

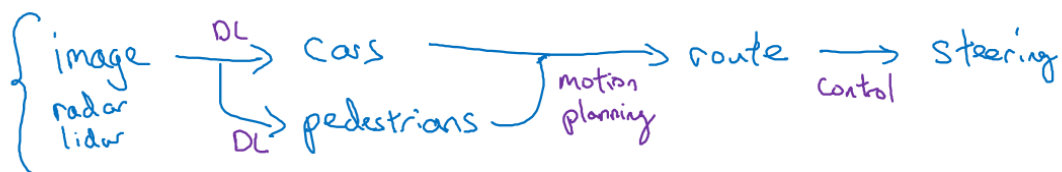
data set is small than a hand-design system is a way to give manual knowledge into the algorithm.

Key question: Do you have sufficient data to learn a function of the complexity needed to map x to y ?

I don't have a formal definition of this phrase, complexity needed, but intuitively, if you're trying to learn a function from X to Y , that is looking at an image like this and recognizing the position of the bones in this image, then maybe this seems like a relatively simple problem to identify the bones of the image and maybe they'll need that much data for that task.

Or given a picture of a person, maybe finding the face of that person in the image doesn't seem like that hard a problem, so maybe you don't need too much data to find the face of a person. Or at least maybe you can find enough data to solve that task, whereas in contrast, the function needed to look at the hand and map that directly to the age of the child, that seems like a much more complex problem that intuitively maybe you need more data to learn if you were to apply a pure end-to-end deep learning approach. So let me finish this video with a more complex example.

Another example: autonomous driving (end to end deep learning is less promising)



- Use DL to learn individual components
- Carefully choose $x \rightarrow y$ depends on tasks you can get data for