Deep Learning Specialization : Course3-4-5

Student
Riahi LOURIZ

Teacher
Andrew Ng

September 11, 2018

Contents

1 Course 3 Structuring Machine Learning Projects

About this Course :
You will learn how to build a successful machine learning project. If you aspire to be a technical leader in AI, and know how to set direction for your team's work, this course will show you how.

Much of this content has never been taught elsewhere, and is drawn from my experience building and shipping many deep learning products. This course also has two "flight simulators" that let you practice decision-making as a machine learning project leader. This provides "industry experience" that you might otherwise get only after years of ML work experience.

After 2 weeks, you will:
- Understand how to diagnose errors in a machine learning system, and
- Be able to prioritize the most promising directions for reducing error
- Understand complex ML settings, such as mismatched training/test sets, and comparing to and/or surpassing human-level performance
- Know how to apply end-to-end learning, transfer learning, and multi-task learning

I've seen teams waste months or years through not understanding the principles taught in this course. I hope this two week course will save you months of time.

This is a standalone course, and you can take this so long as you have basic machine learning knowledge. This is the third course in the Deep Learning Specialization.
Learning Objectives :

1. *Understand why Machine Learning strategy is important*
2. *Apply satisficing and optimizing metrics to set up your goal for ML projects*
3. *Choose a correct train/dev/test split of your dataset*
4. *Understand how to define human-level performance*
5. *Use human-level perform to define your key priorities in ML projects*
6. *Take the correct ML Strategic decision based on observations of performances and dataset*

1.1 Week 1

1.1.1 Introduction to ML Strategy

Why ML strategy ? :
Motivating example :

Suppose you have a cat classifier ML model with an accuracy of 90%, which is not enough for you. But you have some ideas to improve it, like :

- collect more data
- Collect more diverse training set
- Train algorithm longer with gradient descent
- try adam instead of gradient descent
- Try bigger networks
- try smaller network
- try dropout
- Add $L2$ regularization
- Network Architecture :
  - Activation functions
  - number of hidden untis
  - etc

As you can see, there are a lot of ideas in order to improve to accuracy of your machine learning model. But how to choose efficiently the most appropriate one without wasting in all directions ?
$\Rightarrow$ **Machine Learning Strategy**.

Orthogonalization : to be clear-eyed about what to tune in order to try to achieve one effect.

Examples of orthogonalization :

1. Example 1 : TV tunning

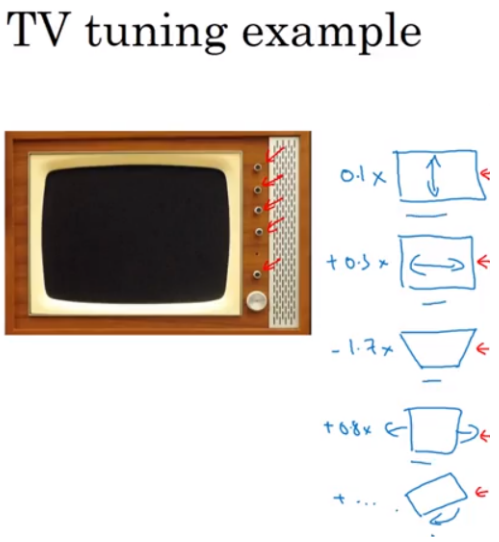

Figure 1: Tv tunning example

→ Each knob do only one thing :
- knob for adjusting how tall the image is
- knob for adjusting rotation
- etc

2. Exampl 2 : Car
   A car has three main items for control :
   (a) Steering wheel
   (b) Acceleration
   (c) Breaking

   - Having only one knob for each control is more efficient and easy than having a knob that can adjust two controls at the same time.
   - This idea of having one knob by control is called orthogonalization.

Chain of assumptions in Machine Learning :
For a supervised learning system to do well, you usually need to tune the knobs of your system to make sure that four things hold true :

1. fit training set well on cost function
2. Fit dev set well on cost function
3. Fit test set well on cost function
4. Performs well in real world

- Anderw Ng tends not to use early stoping. Because it is a knob that simultaneously affects how well you fit the training set, and also improve the dev set performance.

1.1.2 Setting up your goal

Single number evaluation metric :
→ It is recommended to set a single real number evaluation metric in order to evaluate the problem
Example 1 :
Let's recall the following metrics used in classification :

$$Recall = \frac{TP}{TP+FN}$$
$$Precision = \frac{TP}{TP+FP}$$

Let's see an example where it is confusing to choose one model :

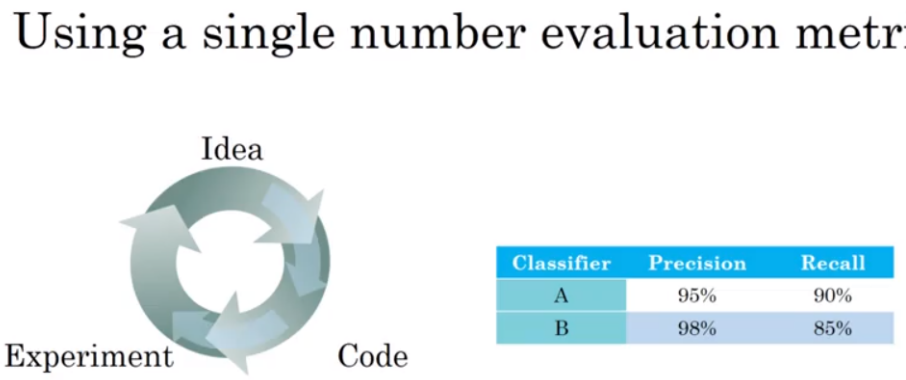

Figure 2: Recall and Precision as metrics

→ In the above table, if we choose Recall & Precision as metrics, we can not know which model is better. That's why it is recommended to keep one single number evaluation metric in order to quikly pick up the best model.

→ One idea to overcome our last issue is to combine Precision and Recall in metric Called F1-score :

$$F_1 - score = \frac{2}{\frac{1}{P}+\frac{1}{R}} = \frac{2*PR}{P+R} \text{ (harmonic mean)}$$

Note : sometimes it is more reasonable to not combine them but only use one of them. You can visit my repository in github to learn more about evaluation for classification tasks : Evaluation for classification probelms in Machine learning.

Example 2 : Cat app for cat lovers in four geographics

Algorithm	US	China	India	Other
A	3%	7%	5%	9%
B	5%	6%	5%	10%

Figure 3: Algorithm's error for cat app

→ It would be maybe good to keep tracking if all errors for all zones. But it is not convenient to choose between A and B quikly.
→ What if you have many models. It is more difficult to make a decision.

Algorithm	US	China	India	Other
A	3%	7%	5%	9%
B	5%	6%	5%	10%
C	2%	3%	4%	5%
D	5%	8%	7%	2%
E	4%	5%	2%	4%
F	7%	11%	8%	12%

Figure 4: Algorithm's error for cat app

→ By computing average errors, it is more easy to choose a model.

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

Figure 5: Algorithm's average error for cat app

→ In this case it is easy to pick up the algorithm C.

Satisficing and optimizing metric :
→ It is not always easy to combine all the things you care about into one single row number
→ To do so, we will set up a satisficing & optimizing matrix
Example 1 : Cat classification

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

Figure 6: Satisficing and optimizing metric

- We have two metrics : accuracy and running time
- Choosing formula like : $cost = accuracy - 0.5 * running_{time}$ seems not to be good at all
- Instead, we can set :
  - Accuracy as an optimizing
  - Running time as satisficing (example : having running time $\leq 100ms$)
- More generally : for N matrix
  - 1 metric for optimizing
  - N-1 metrics for satisficing

Example 2 : Wakewords/Trigger words
→ examples of wakewords : Alexa, OK google, etc
→ we can set as metrics the following :

- Accuracy : optimizing
- False positive : satisficing metric (by having almost one false positive each 24 hours)

Train/dev/test distributions :
→ The way you set up your dev/test sets can have a huge impact in the team work
→ Let's have an example : suppose we have data for cat classification for the below regions :

- US
- UK
- other Europe
- South America
- India
- China
- Other Asia
- Australia

→ Then suppose we have set the first four regions as dev set, and the rest as test set ⇒ This would be a very bad idea, because the two datasets do not come from the same distribution.
→ To de well, we can randomly shuffle into dev/test.
Example 2 :True story (details changed)

- Optimizing on dev set on loan approvals for medium income zip codes
- tested on low income zip codes ⇒ bad results (wasting ≈ 3 months to re-do a lots of work).

→ **Guidelines** :

- Choose a devset and test set to reflect data you expect you get in the future and consider important to do well
- dev/test sets should come from the same distribution

Size of the dev and test sets :
→ The size of dev/test sets are changing in deep learning era
→ Old way of splitting data :

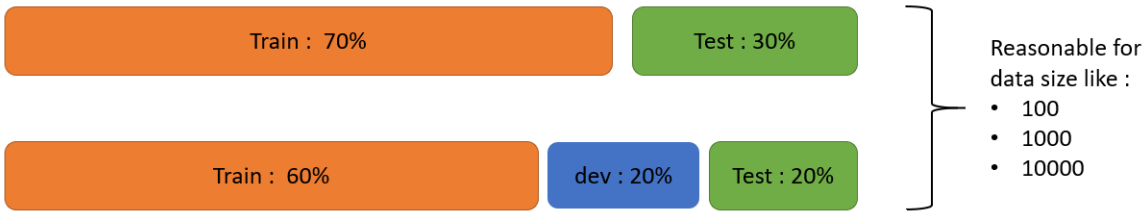

Figure 7: Splitting data for old era of machine learning

→ In deep learning era :

Figure 8: Splitting data for the era of deep learning

→ Size of test set : set your test set to be big enough to give high confidence in the overall performance of your system
→ Assure to have Train/dev/test sets, not only Train/dev

When to change dev/test sets and metrics :
Cat dataset example : :
Let's say that we are working on a cat classification error : we got two classifiers with the errors as following :

- Algorithme A : 3% error ← let pornographic pictures
- Algorithme B : 5% error ← does not let pornographic pictures

→ So from a metric/dev set point of view : the model A is the best one.
→ But from you/users point of view : the model B is the best.
→ In this case what we should do is to change the metric :

- We know in the beginning that error metric counts only missclassification pictures :

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} \hat{1}(y_{pred}^{(i)} \neq y^{(i)})$$

- To change it, we could do the following :

$$Error = \frac{1}{\sum_i w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} \hat{1}(y_{pred}^{(i)} \neq y^{(i)})$$

where :

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

Orthogonalization for cat pictures : anti-porn :

1. So far we've only discussed how to define a metric to evaluate a classifiers
2. Worry separatly on how to do well on this metric
3. Example :

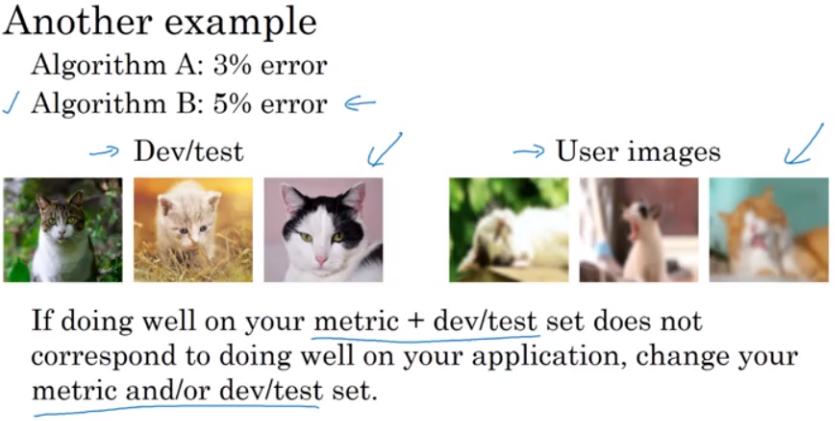

Figure 9: Orthogonalization for cat classifier : anti-porn

1.1.3 Comparing to human-level performance

Why human level performance ? :
→ Comparing to human level performance :

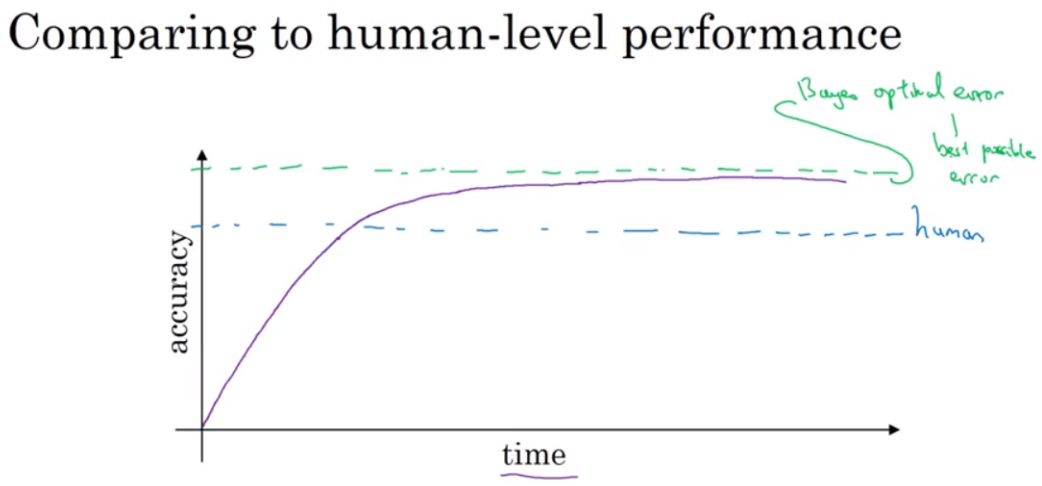

Figure 10: Comparing to human level performance

1. Bayes optimal error : the best possible error. It can never be surpassed unless you are overfitting.
2. The perfect level of accuracy may not to be 100% (Example : blurry pictures for cat that no one and nothing can tell what it is exactly).
3. There are two reasons for why progress often slows down when you surpass human level performance :
   (a) Human level performance for many tasks not that far from Bayes optimal error.
   (b) So long as your performance is worse than human performance level, then there are certain tools you could use to improve performance that are harder once you have surpassed human level performance.

→ Why compare to human level performance : Human are quit good at a lot of tasks. So long as Machine Learning is worse than humans, you can :

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

Avoidable bias : Cat classification example
→ 1-case : let's suppose that cat pictures are good enough to have a human error of 1% :

- Human error ($\approx$ Bayes error) $\rightarrow$ 1%
- Training error $\rightarrow$ 8%
- Dev error $\rightarrow$ 10%

In this case the most important gap is about human and training error $\Rightarrow$ we must focus to improve training error $\Rightarrow$ Tactics for bias reduction.

$\rightarrow$ 1-case : let's suppose that cat pictures are blurry enough to have a human error of 7.5% :

- Human error ($\approx$ Bayes error) $\rightarrow$ 7.5%
- Training error $\rightarrow$ 8%
- Dev error $\rightarrow$ 10%

In this case the most important gap is about training/dev errors $\Rightarrow$ we must focus to improve dev error $\Rightarrow$ Tactics for variance reduction.

Terminology :

1. Avoidable bias = human error - training error
2. Variance = dev error - Training error

Understanding human level performance :
$\rightarrow$ Human level error as a proxy for bayes error :
Let's take Medical image classification example and suppose the following :

- (a) Typical human $\Rightarrow$ 3% error
- (b) Typical doctor $\Rightarrow$ 1% error
- (c) Experienced doctor $\Rightarrow$ 0.7% error
- (d) Team of experienced doctors $\Rightarrow$ 0.5%

$\Rightarrow$ What is human level error ? is it 3%, 1%, 0.7% or 0.5% ?
To answer to this question, we must have in mind that human level error is a proxy for the bayes error (= the best error we can achieve).
$\Rightarrow$ Bayes error $\leq$ 0.5%
$\Rightarrow$ We can choose 0.5% as human level performance
$\Rightarrow$ But When choosing human-level performance, it has to be chosen in the terms of what you want to achieve with the system.
$\Rightarrow$ You might have multiple human-level performances based on the human experience. Then you choose the human-level performance (proxy for Bayes error) that is more suitable for the system you're trying to build.
$\rightarrow$ Error Analysis example :

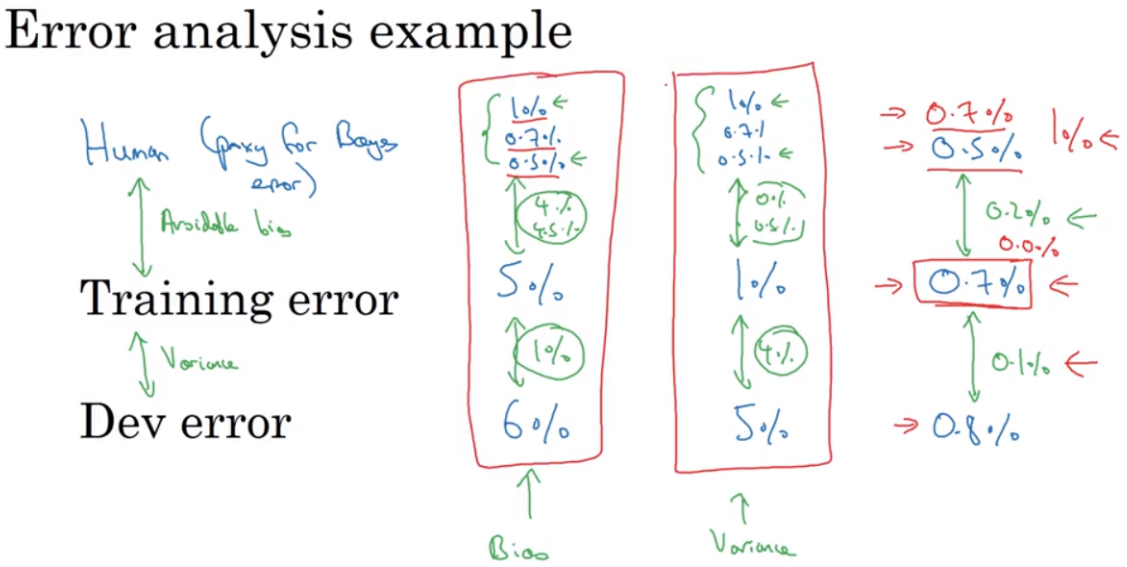

Figure 11: Error Analysis example

→ Summary of bias/variance with human level performance :

- human-level error (proxy for Bayes error)
  - Calculate avoidable bias = training error - human-level error
  - If avoidable bias difference is the bigger, then it's bias problem and you should use a strategy for bias resolving.
- training error
  - Calculate variance = dev error - training error
  - If variance difference is bigger, then you should use a strategy for variance resolving.

Surpassing human level performance :
Let's study the following cases :
→ Case 1 : suppose that we have for some application the following errors :

- Team of human : 0.5%
- one human : 0.1%
- Training error : 0.6%
- Dev error : 0.8%

In this case the avoidable bias = 0.6% -0.5% = 0.1% $\leq$ 0.2% = 0.8%-0.6%
So there will be a focus on variance reduction.

→ Case 2 : suppose that we have for some application the following errors :

- Team of human : 0.5%
- one human : 1%
- Training error : 0.3%
- Dev error : 0.4%

This case is harder, because we do not know if we have overfitted data by 0.2% or maybe base error is 0.1%, 0.2% ...? ⇒ we can not answer giving the above information.

→ Problems where ML significantly surpasses human level performance :

- Online advertising
- Product recommendation
- Logistics (predicting transit time)
- loan approvals

- All the previous applications are used with structured data. ⇒ not natural perception problems.
- Human tend to be very good in natural perception task like : speech recognition, NLP, etc
- It is harder for computers to surpass human level performance in natural perception task
- But today there are some computers that can surpass human level performance in some natural perception problems : speech recognition, image recognition, medical application (ECG, diagnosing skin cancer, narrow radiology task)

Improving your model performance :

1. The two fundamental asssumptions of supervised learning:
   - You can fit the training set pretty well. This is roughly saying that you can achieve low avoidable bias.
   - The training set performance generalizes pretty well to the dev/test set. This is roughly saying that variance is not too bad.
2. To improve your deep learning supervised system follow these guidelines:
   - Look at the difference between human level error and the training error - avoidable bias.
   - Look at the difference between the dev/test set and training set error - Variance.
   - If avoidable bias is large you have these options:
     - Train bigger model.
     - Train longer/better optimization algorithm (like Momentum, RMSprop, Adam).
     - Find better NN architecture/hyperparameters search.

- If variance is large you have these options:
  - Get more training data.
  - Regularization (L2, Dropout, data augumentation).
  - Find better NN architecture/hyperparameters search.

1.2 Week 2

1.2.1 Error Analysis

Carrying out error analysis :
→ Look at dev examples to evaluate ideas :
Let's say that you have developped a cat classifier model, and you get 90Let's assume also that you found out that your classifier misclassifies some dogs pictures as cats.
Should you try to make your classifier do better on dogs ?
Before jumping to the answer, we should perform an Error Analysis, then we can decide :

- Get ≈ 100 misclassified dev set examples
- count up how many are dogs
- If only 5% are dogs pictures ⇒ our error will go down from 10% to 9.5% (5%(10) = 0.5 so 10-0.5 = 9.5)
- If 50% are dogs pictures ⇒ our error will go down from 10% to 5% (50% (10) = 5 so 10 - 5 = 5) ⇒ It is a good choice to make our model to better on dogs

→ Evaluate multiple ideas in parallel :
Let's say that we have the following ideas for the cat classifier :

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

To carry out those 3 ideas, what you could do is the following :

Image	Dog	Great Cats	Blurry	Instagram	Comments
1	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
⋮	⋮	⋮	⋮		
% of total	8%	43%	61%	12%	

Figure 12: Evaluate multiple ideas in parallel for error analysis

In this example, we should work on blurry images to improve performance. This procedure safes a lot of time for team working on a such project.

Cleaning up incorrectly labeled data :
→ Incorrectly labeled examples :

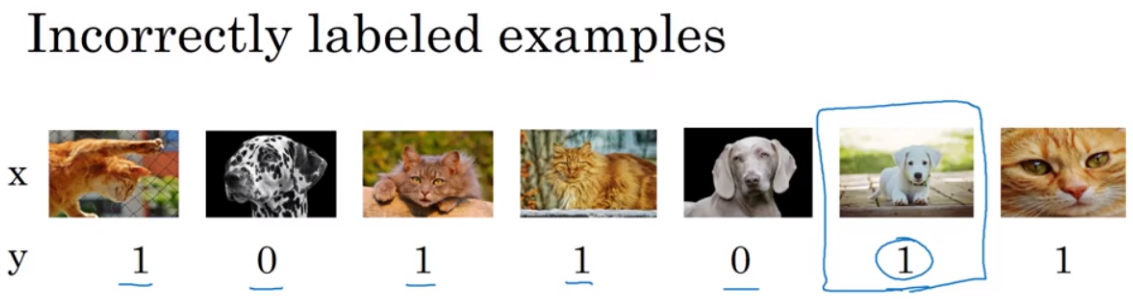

Figure 13: Incorrectly labeled examples : Cat dataset

If you found out that your dataset present some incorrectly labeled data, what can you do ?

- On training data : Deep Learning algorithms are quite robust to random errors in the training set. But they are less robust to systematic errors (example : all white dogs are mislabeled as cats)
- On dev/test datasets : to handle incorrectly labeled data, you add one extra column during error analysis :

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%	

Figure 14: handling incorrect labels on dev and test sets

Is it worthwhile going in to try to fix up this 6% of incorrectly labeled examples ?
If it make a significant difference to your ability to evaluate algorithms on your dev set, then go ahead and fix incorrect labels. Let's look at these two examples to clarify better our statement :

– example 1 :

    * Overall dev error : 10%
    * Error due to incorrect labels : 0.6%
    * Error due to other causes : 9.4%

  ⇒ not worth to fix incorrect labels

- example 2 :
    * Overall dev error : 2%
    * Error due to incorrect labels : 0.6%
    * Error due to other causes : 1.4%

  ⇒ Yes it is worth to fix incorrect labels

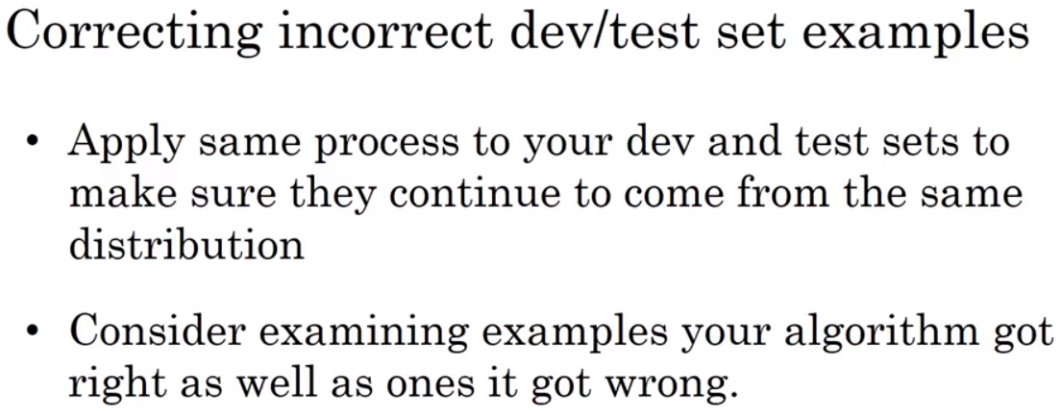

Figure 15: Conclusion

The above figure shows that train and dev/test may now come from slightly different distributions. It is Ok but the most important thing is to have dev/test sets come from the same distribution.

Build your first system quickly, the iterate :
→ Speech recognition example :
There are many things you could do to improve your speech recognition system :

- Noisy background : cafe noise, car noise
- Accented speech
- Far from microphone
- Young Childern's speech

- Stuttering (uh, oh, umm, ..)

The challenge is how to pick one of those to focus on.
To be efficient in a new Machine Learning Project, build your first system quickly, then iterate :

- Set up dev/test and metric (set your target)
- Build initial system quickly
- Use Bias/Variance analysis and Error Analysis to prioritize next steps.

1.2.2 Mismatched training and dev/test set

Training and testing on different distributions :

→ Today many teams do training with Deep Learning in a training set that has a different distribution as dev/test sets.
→ Example 1 : Cat app

Figure 16: Data mismatch example

And you care about your final system to do well on mobile app pictures.
- Let's say we have get 10000 pictures from mobile app and 200 000 from webpages
- You can not train only using the mobile pictures (small dataset for deep learning)
- Having 200000 from webpages is good, but the dilemma is those pictures come from a different distribution.

What can you do ?

- Option 1 : combine 200000 with the 10000 then shuffle them randomly into train/dev/test sets (205 000, 2500, 2500).
  Advantage : train/dev/test all come from the same distribution.
  Disadvantage : is what the most pictures for dev/test will come from webpages, which we do not care about in the final app.

- Option 2 : Choose the training set to be the 200,000 images from the internet along with 5000 images from your mobile app pictures. The 5000 remaining images will be split equally in dev and test sets.

→ Example 2 : Speech recognition system (Speech activated rearview mirror for a car)

Let's say that we have the following data :

- Training data : purchased data, smart speaker control, voice keyboard ⇒ 500000
- Dev/test : Speech activated rearview mirror ⇒ 20 000

So the best option to split data is to choose the training set to be the 500,000 images from the training data along with 10 000 images from your dev/test set. The 10 000 remaining images will be split equally in dev and test sets.

Bias and Variance with mismatched data distributions :

→ Cat classifier example :
- Assume humans gets ≈ 0% error (= Bayes error)
- Training error : 1%
- Dev error : 10%

In ths case, we would draw a conclusion that there is a problem of variance (the model does not generalize well on the dev set). But if the two set do not come form the same distribution, our conclusion is no longer correct.
To tease out if it is a problem of variance or distribution, we will define a new subset of data : **Training-dev set : Same distribution as training set, but not used for training**.

Let's say now that we have the following errors :

- Train error : 1%
- Train-dev error : 9%
- Dev error : 10%

⇒ Variance problem

- Train error : 1%
- Train-dev error : 1.5%
- Dev error : 10%

⇒ data mismatch problem

- human error : 0%

- Train error : 10%
- Train-dev error : 11%
- Dev error : 12%

⇒ bias problem

- human error : 0%
- Train error : 10%
- Train-dev error : 11%
- Dev error : 20%

⇒ bias problem + data mismatch

→ Recap on Bias/Variance on mismatched training and dev/test sets :

- Training error - human error = avoidable bias
- training-dev error - training error = variance
- dev error - training-dev error = data mismatch
- test error - dev error = degree of overfitting to dev set.

→ More general formulation :

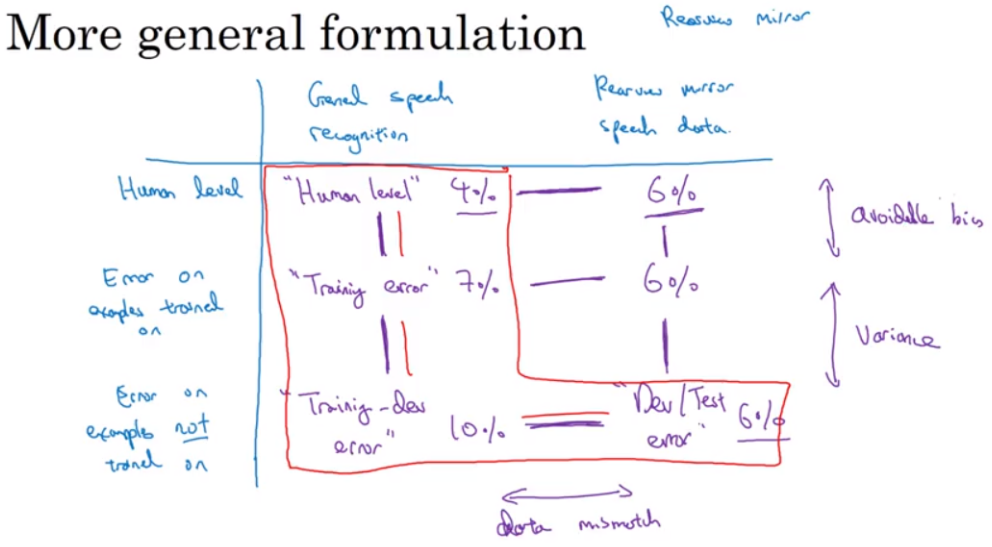

Figure 17: data mismatch and bias/variance analysis

Addressing data mismatch :

→ If your training set comes from a different distribution than your dev/test sets, and if error anal-

ysis shows you that you have a data mismatch, what can you do ?

- Carry out manual error analysis to try to understand difference between training and dev/test sets.
- Make training data more similar; or collect more data similar to dev/test sets.
- Artificial data synthesis :

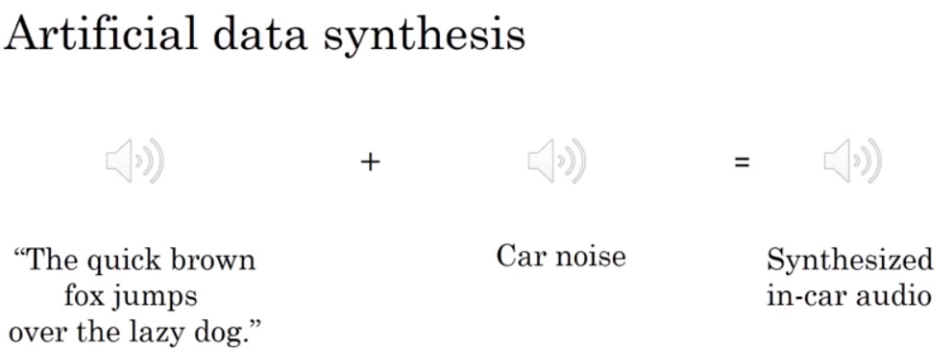

Figure 18: Artificial data synthesis

The picture shows a reasonable process to synthesize data to be a in-car data.

Caution : the algorithm may overfit the synthesized data because it represents a small subset

1.2.3 Learning from multiple tasks

Transfer learning :
→ One of the most powerful ideas in deep learning is that sometimes you can take knowledge the neural network has learned from one task and apply that knowledge to a seperate task.

→ Example 1 : having a neural network that can recognize objects like cats and then use that knowledge or part of it to help you do better job reading x-ray scans.

(a) Having a cat classifier (X,y) :

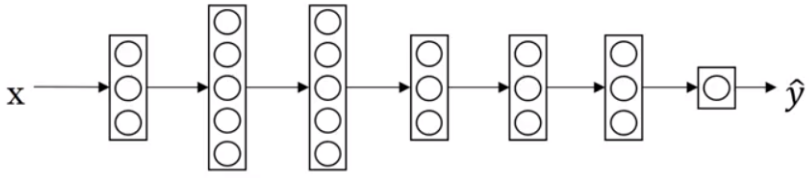

Figure 19: Neural Network for cat classification

(b) We can transfer to radiology diagnosis :

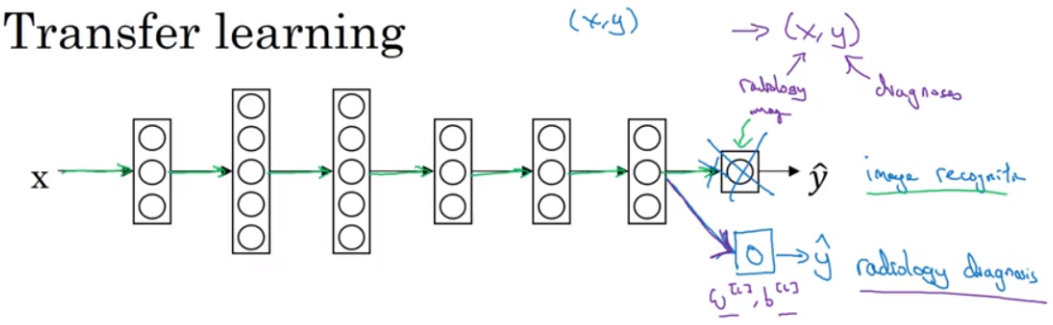

Figure 20: transfer learning

(c) There are two option for learning weights :

- If we have enough data ⇒ retrain all weights
- Otherwise : retrain only $W^{[L]}$ and $b^{[L]}$ weights

(d) if you are retraining all layers :

- The initial model is called a pre-trained model (because all weights were trained on another data)
- Otherwise : the final model is called fine tunning model

→ Example 2 : Transfer speech recognition ⇒ wakeword/trigger word

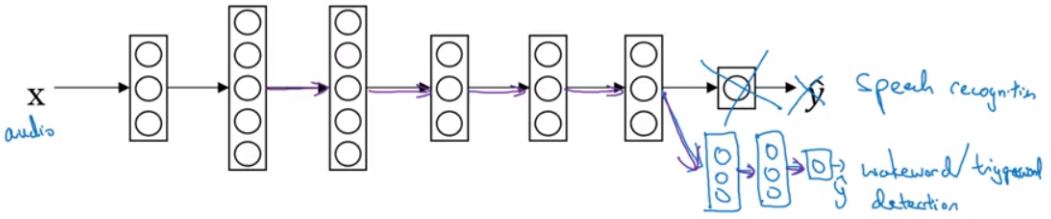

Figure 21: transfer learning example 2

Note that after deleting the last layer, you can add more layers in the end of your Neural network.

→ When transfer learning makes sense :

- Task A and B have the same input X
- you have a lot more data for Task A than Task B
- Low level features from A could be helpful for learning B.

Multi-task learning :
→ Simplified autonomous driving example :
Suppose that your self-driving car should detect :
- Pedestrians
- Cars
- stop signs
- traffic light

So for the example if we have an input $x^{(i)}$:

Figure 22: self driving system input example

So $y^{(i)}$ willbe a vector of 4 values (4,1) :
- Pedestrians : 0
- Cars : 1
- stop signs : 1
- traffic light : 0

$\Rightarrow Y = [y^{(1)}, y^{(2)}, ..., y^{(m)}](4,m)$
$So the neural network architecture is$:

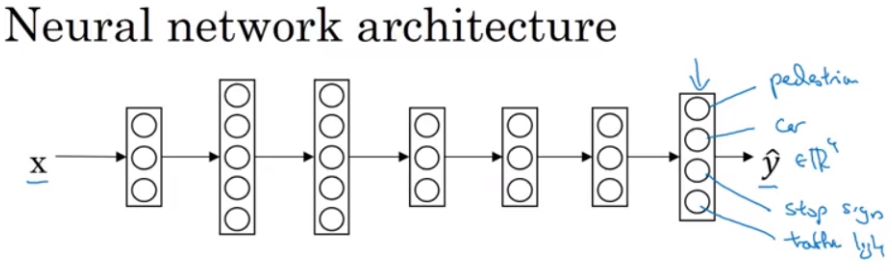

Figure 23: NN architecture

$$Cost = \frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{4} L(\hat{y}_j^{(i)}, y_j^{(i)})$$

- Multitask-learning is different from softmax regression where one image can have multiple labels.

- One may train four separate Neural Network, each one for a task. But of some of the earlier

→ When multitask learning make sense :

- 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
- Transfer learning is used more than multitask learning

1.2.4 End-to-end deep learning

What is end-to-end deep learning? :

Some systems have multiple stages to implement. An end-to-end deep learning system implements all stages with a single NN.

→ Example 1 : Speech recognition
(a) Not end-to-end DL system : Audio → features → phonemes → words → transcript

(b) End-to end DL system : audio → transcript

- End-to-end DL works well if have a lot of data (¿10 000 hours of audio)
- Not end-to-end : can work with small amount of data (3000 hours of audio)

→ Face recognition system :
(a) Not end-to-end DL system (Best approach for now) : Image → Face detection → Face recognition

(b) End-to end DL system : Images → Face recognition

- In practice the best approach is the second one for now

- In the second implementation, it is a two step approach where both parts are implemented using deep learning.

- It is working well because it is harder to get a lot of pictures with peaple in front of the cameras than getting faces of peaple and compare them.
- In the seconde implementation at the last step, the NN takes as input two faces and outputs if the

two faces are for the same person or not.

→ Machine translation system :

(a) End-to-end DL system (Best approach for now) : English → French

(b) Non End-to end DL system : English → text analysis → french

Here end-to-end DL system works better because we have a lot of data to train on.

→ Estimating child's age from the x-ray picture of a hand

(a) Not end-to-end DL system (Best approach for now) : Image → Bones → Age

(b) End-to end DL system : Images → Age

Here Non end-to-end DL system works better, because we do not have a lot of data to train on.

Whether to use end-to-end deep learning :

- Pros of end-to-end deep learning:
  - Let the data speak. By having a pure machine learning approach, your NN learning input from X to Y may be more able to capture whatever statistics are in the data, rather than being forced to reflect human preconceptions.
  - Less hand-designing of components needed.
- Cons of end-to-end deep learning:
  - May need a large amount of data.
  - Excludes potentially useful hand-design components (it helps more on the smaller dataset).
- Applying end-to-end deep learning:
  - 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.

2 Course 4 Convolutional Neural Networks

2.1 Week 1

2.2 Week 2

2.3 Week 3

2.4 Week 4

3 Course 5 Sequence Models

3.1 Week 1

3.2 Week 2

3.3 Week 3

3.4 Week 4

4 Bibliography

Coursera plateforme

Mahmoud Badry notes

Beautifully drawn notes by Tess Ferrandez