

# [2주차] ML Strategy(2)

## 알고리즘이 범하는 실수 점검법

1. 첫번째로, 대략 100개의 미스레이블된 dev set 샘플을 가지고와서 수동으로 검사
- 2-1. 그 중 5개만 강아지라면 오류가 10% → 9.5%로 내려갈 수 있다. (상대적으로 5%가 줄어든 것)  
⇒ 이것을 **ceiling on performance**
- 2-2. 100개 중 50장이 강아지라면 강아지 문제에 시간을 투자하는 것이 조금 더 긍정적! → 오류가 10퍼센트에서 5퍼센트로 줄 수 있기 때문

머신러닝에서는 사람들이 가끔씩 hand engineering에 대해 편하하거나 너무 많은 value insight를 사용한다고 생각하는데요, 하지만 적용시스템을 만드는 경우, 이렇게 **간단히 숫자를 세는 과정이**, 오류 분석이 많은 시간을 절약할 수 있게 해줍니다.

## 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	✓			✓	Pitbull
2			✓	✓	
3		✓	✓		Rainy day at zoo
:	:	:	:	:	
% of total	8%	43%	61%	12%	

Andrew Ng

기억하셔야할 부분은, 오류분석을 진행한다는 것은, 알고리즘이 잘못 카테고리화한 것들에 대해서 dev set example을 보는 것이라는 것 말이죠.

1. 잘못 인식된 이미지가 강아지 사진이였을 경우에, 여기에다 체크로 표시, 이것은 핏불 사진 코멘트 덧붙임.
2. 두번째 사진이 흐릿하다고하면 여기 노트
3. 세번째 사진이 비오는 날 동물원에서 찍은 사자의 이미지면, 잘못 인식된 것입니다. 이것은 great cat이고, 또 이것은 흐릿한 데이터겠죠. 노트를 이렇게 적을 수 있겠죠. 비오는 날 동물원

마지막으로, 이미지 세트를 리뷰하셨다면, 알고리즘의 몇 퍼센트가, 알고리즘의 오류 카테고리 중 몇 퍼센트가 강아지에 관련한 상황인지, 또는 great cat 인지 또는 흐릿한 이미지인지 개수를 셀 것입니다. 8퍼센트에 해당하는 수치는 강아지 사진일수도 있는데요, 43퍼센트 수치는 great cats, 61퍼센트는 흐릿한 이미지이구요. 이 것은, 세로줄을 단계별로 내려가시면서 몇 퍼센트의 이미지가 체크마크가 되어있는지 개수를 세면서 확인하는 것입니다. 이 절차를 어느정도 진행하셨으면, 또 다른 카테고리의 실수를 발견할 수 있는데요. 예를 들어, Instagram style filter와 같은 화려한 필터들이 여러분의 분류기를 망가트리는 경우가 있을 것입니다. 그런 경우, 괜찮습니다만, 진행 과정에서 세로줄을 이렇게 추가하셔도 됩니다. 멀티칼라 필터인 인스타그램 필터들과 스냅필터를 말이죠. 그런 뒤에 개수를 다시 세고, 새로운 오류 카테고리에서 몇 퍼센트의 오류가 발생하는지 확인을 합니다.

이런 절차를 모두 마치면, 이렇게 다른 종류의 오류 카테고리에 대해서 수정작업을 하는 것이 얼마나 값어치가 있는지 추정할 수 있습니다. 예를 들어, 이 예제에서보면 분명히 알 수 있는 것은 흐릿한 이미지에서 많은 오류가 있었고, great cat에 대한 오분류도 꽤 많았습니다. 그렇게해서, 분석의 결과는 흐릿한 이미지를 작업해야한다는 직접적인 결과가 아닙니다. 이런 결과는 단순한 수학 공식처럼 무엇을 어떻게 하라고 알려주는 것이 아닙니다. 대신에 전반적으로 가장 선택하기 좋은 옵션에 대한 감각적인 부분을 제시해주는 것이죠. 또한 예를 들어 이런 내용도 말해줍니다. 아무리 강아지에 대한 인식을 잘해도, 도는 인스타그램에서 분류를 잘해도 8퍼센트까지 밖에 성능개선을 못한다, 또는 12퍼센트 개선 가능하다 이런식으로 말이죠. 아니면 great cat 이미지 또는 흐릿한 이미지는 잠정적인 개선 부분이 있지만 이 부분은 이정도 밖에 개선이 안된다라는 부분을 말해줄 수 있습니다. 성능을 개선하고 증가시키는 부분에 있어서 특정 한계치가 있는데요, great cat에 대해 성능개선을 하는 idea가 얼마나 있는지에 따라 흐릿한 이미지의 idea가 얼마나 있는지에 따라서 말이죠 둘 중 한가지를 고르는 방법도 있습니다. 또는 팀원이 충분하다고 하면, 2개의 다른 팀으로 구성하는 방법도 있습니다. 한팀은 great cat에 대한 오류를 개선하고, 다른 한팀은 흐릿한 이미지를 개선하는 것입니다.

요약하자면, 오류 분석을 실행하는데에는 먼저 dev set나 development set에서 잘못 레이블된 example을 찾고, 이러한 잘못 레이블된 example들을 찾아서 false positives와 false negatives를 찾습니다. 그리하여 다양한 카테고리에 대한 오류의 총 개수를 각각 카테고리별로 찾아냅니다. 이런 진행과정에서 새로운 오류 관련 카테고리를 보신것과 같이 생성하기 원할 수 있는데요. example들을 보면서 "우와, 인스타그램 필터가 많네, 또는 스냅챗 필터가 많네," 라고 하며 분류기를 망친다는 생각이 드시면, 진행 와중에 새로운 카테고리를 생성하실 수 있습니다. 다양한 사유로 잘못 레이블된 example들을 직접 세면서 그 비율을 계산하면 우선순위를 정하는데 도움이 될 것입니다.

## 잘못 라벨링된 데이터 처리

만약 오류가 비교적 random 하다고 하면, 아마도 이 오류들을 그대로 놔둬도 괜찮을 것입니다. 고치는데 너무 많은 시간을 쓴을 필요 없이 말이죠. 물론 트레이닝 세트로 가서 레이블을 직접 검사하고 수정해서 나쁠 것은 없습니다. 가끔씩은 그럴 값어치가 있을 수 있습니다. 하지만 수정안해도 괜찮을 수 있습니다. 토탈 데이터 세트 사이즈가 충분이 크고 실제 오류의 퍼센트가 너무 크지 않으면 말이죠.

딥러닝 알고리즘이 random error에 강하다는 조건이죠. 딥러닝 알고리즘은 systemic error에는 덜 강합니다.

**잘못 레이블 된 example들이 dev set나 test set에서는 어떨까요?** 만약 여러분이 dev set나 테스트세트에서 잘못 레이블된 example에 대해 걱정이 되신다고 하면, 권장드리는 것은, 오류 분석 진행 중, 추가적으로 세로줄을 넣어서 example들의 숫자를 세는 것입니다. 레이블 Y가 틀린부분의 총 개수를 말이죠.

### 문제는, 여기 6퍼센트 틀리게 레이블된 샘플을 고치는 것이 과연 값어치가 있는 작업일까

→제가 조언 드리는 것은 만약 여러분의 알고리즘이 dev set에서 그 평가능력이 현저히 개선되어 바뀔 수 있다면 시간을 써서 레이블이 틀린 것들을 고치라고 말씀 드릴 수 있습니다.

ex1)

저는 여러분이 잘못 레이블된 example들을 줄일지의 여부를 결정하는데 3가지의 수치를 보고 결정하길 추천드립니다.

#### 1. 전체적인 dev set 오류를 보시길 권장

이전 비디오에서 보여줬던 예제를 살펴보면, 여기서는 시스템이 90퍼센트의 정확도를 가지고 있다고 했는데요, 그러면 10퍼센트 오류겠죠. 그러면 여러분은 오류의 개수를 봐야할 것입니다.

### 2. 또는 잘못 레이블된 것으로 인한 오류의 퍼센트입니다.

이번 케이스 같은 경우엔, 6퍼센트의 오류가 잘못된 레이블로 인한 오류입니다. 그러면 10퍼센트의 6퍼센트는 0.6퍼센트입니다.

### 3. 다른 나머지 사유로 인한 오류도 봐야합니다.

만약 10퍼센트 에러가 dev set에서 발생했고, 이것의 0.6퍼센트가 틀린 레이블로 인한 오류면 그러면 나머지 9.4퍼센트는 강아지를 고양이로 잘못 오인식한 경우와 같은 사유일 것입니다. 또는 great cats 와 같은 이미지를 말이죠. 그러므로 이런 케이스에서는, 9.4퍼센트의 오류만큼 수정하는데 집중할 수 있습니다.

ex2)

## 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%	

Overall dev set error ..... 10%

Errors due incorrect labels ..... 0.6% ←

Errors due to other causes ..... 9.4% ←

$\frac{2}{10} = 2\%$   
 $\frac{0.6}{10} = 0.6\%$   
 $\frac{9.4}{10} = 9.4\%$   
 $\frac{0.6}{9.4} = 0.064\% \approx 0.6\%$   
 $\frac{2}{9.4} = 0.21\% \approx 2.1\%$   
 $\frac{9.4}{9.4} = 1.0 \approx 1.0\%$

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

Andrew Ng

다른 example을 살펴보겠습니다. 만약 여러분이 러닝 문제에서 상당부분 진전이 있었다고 해봅시다. 그래서 10퍼센트 오류대신 오류를 줄여서 2퍼센트로 만들었다고 해보겠습니다. 하지만 아직 전체 오류의 0.6퍼센트는 잘못된 레이블로 인한 레이블입니다. 그러면 여러분이 잘못 레이블된 dev set 이미지를 검사하고 싶으면, dev set 데이터의 잘못 레이블링된 2퍼센

트입니다. 그러면 이것의 상당한 비중을 차지하는 비율인데요, 0.6 나누기 6퍼센트입니다. 그러면 이것은 여러분 레이블의 6퍼센트가 아니라 30퍼센트가 되는 것이죠. 틀린 example은 사실 틀린 레이블 example로 인한 것입니다. 그러면 다른 사유로 인한 오류는 이제 1.4퍼센트가 됩니다. 그러면 dev set에서 틀리게 레이블된 오류가, 그 비중이 매우 높기 때문에 이제는 dev set에서 틀린 레이블을 수정하는 작업이 값어치가 있겠습니다. dev set의 목표를 기억하실지 모르겠지만, dev set의 주된 목적은 classifier A와 B 사이에서 어떤 것을 고를지 선택할 수 있도록 도움을 주는 것입니다. 그러므로 여러분은 classifier A와 B를 2개 모두 시도해 보는 것이구요. 하나는 2.1퍼센트 오류이고, 다른 하나는 1.9퍼센트 오류입니다. dev set에서 말이죠. 하지만 이제 더 이상, 이 classifier가 더 이상 여기 실수의 0.6퍼센트가 틀린 레이블로 인한 오류이기 때문에 더 낫다고 이야기하는 것을 신뢰할 수 없게 되는 것입니다. 더 나은 classifier인지 모르는 것이죠. 그렇다면, 이러한 사실을 여러분이 직접 개입해서 dev set에서 틀린 레이블에 대한 부분을 고치는 것에 대한 충분한 이유일 것입니다. 여기 오른쪽 예제를 보시면, 알고리즘에서 발생하는 오류를 전반적으로 평가하는데서, 이것이 큰 영향을 끼치고 있습니다. 반면에 왼쪽 예제를 보면, 알고리즘에 끼치는 퍼센트 영향이 더 작습니다.

### [추가 가이드라인]

## 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.

Andrew Ng

1. 첫째로, 권장드리겠습니다. 여러분이 어떤 방식의 프로세스를 적용하던 dev set과 test set에 동시에 적용시키라고

- 둘째로, 여러분의 알고리즘이 잘 맞춘 것들을 포함해서 틀린 것들의 example을 잘 살펴보도록 하시기 바랍니다. 여러분의 알고리즘이 틀린 example들을 보고 그것들이 고쳐져야 한다고 판단하는 것은 쉽습니다. 그러나 또 가능한 것은, 여러분이 잘 맞추지 못한 것도 수정되어야 하는 것들이 있을 수 있다는 것입니다. 여러분의 알고리즘이 틀린 것만 고치게 되면, 알고리즘에서 더 많은 bias estimates 와 오류가 남게됩니다. 알고리즘에 불공평한 이점을 부여하는 셈이죠. 단순히 틀린것만 다시 확인하는 것입니다. 알고리즘이 맞춘 것은 다시 확인을 안하고 말이죠. 알고리즘은 분명 단순한 운으로 맞췄을 수도 있습니다. 그러면 레이블이 고쳐지지 않은 상태에서는 올바른 것에서 틀린 것으로 갈수도 있습니다. 2번째 부분은 쉬운 부분이 아닙니다. 그래서 항상 실행되지는 않죠. 실행되지 않는 이유는, 만약 여러분의 classifier가 굉장히 정확하면, 맞추는 것 대비 더 적게 틀리게 됩니다. 그렇게해서 만약 classifier가 98퍼센트 정확도를 갖으면, 이것은 2퍼센트 틀리고, 98퍼센트 맞추는 것입니다. 그러므로 2퍼센트의 데이터 레이블을 검사하고 입증하는 것이 훨씬 더 쉽고, 98퍼센트의 데이터를 입증하는 것은 훨씬 더 오래 걸립니다. 그렇기 때문에 잘 실행되지 않는 것이죠. 이것은 여러분이 고려할 사항입니다.
- 마지막으로, 저기있는 레이블들을 dev와 테스트세트에서 고치게하면 이러한 동일한 절차를 트레이닝세트에서도 적용시킬 수도 있고 그렇지 않을 수도 있습니다. 다른 비디오에서도 이야기했지만, 트레이닝세트에서 레이블을 수정하는 것은 덜 중요한 편입니다.

## 첫 시스템을 빠르게 빌드하고, 그 다음 반복하라

- 첫째로 가장 dev/test set와 그 매트릭을 셋업
  - = 이건 결국에 여러분이 어디에 목표를 둘지를 결정라는 과정입니다. 만약 여러분이 틀린 경우, 언제든지 나중시점에 바꿀 수 있습니다. 일단 먼저 그 목표를 어디든 설정을 하십시오.
- 그 다음 처음 머신 러닝 시스템을 만드는 것
  - = 트레이닝 세트를 찾으시고, 트레이닝시키고 결과를 지켜보십시오. 여러분의 dev/test set 그리고 값과 매트릭을 상대로 얼마나 잘 작동하는지 지켜보고 이해

초기 이니셜 시스템의 가치는 학습이 완료된 시스템의 구축, 그리하여 학습 완료된 시스템을 통해 편향/편차를 조절, 그리고 우선순위를 결정하는 것, 오류 분석을 가능케하고, 실수를 보고, 이에 맞는 수많은 접근 방식과 방향성을 알아내서, 어떤 부분이 가장 값어치가 있는지 알아내는 것

→ 그냥 일단 지저분하더라고 빨리 시스템을 만들고 이 것을 바탕으로 시스템의 우선순위를 정해 개선해 나가는 방식으로 접근하십시오.

## 다른 분포의 트레이닝과 테스팅

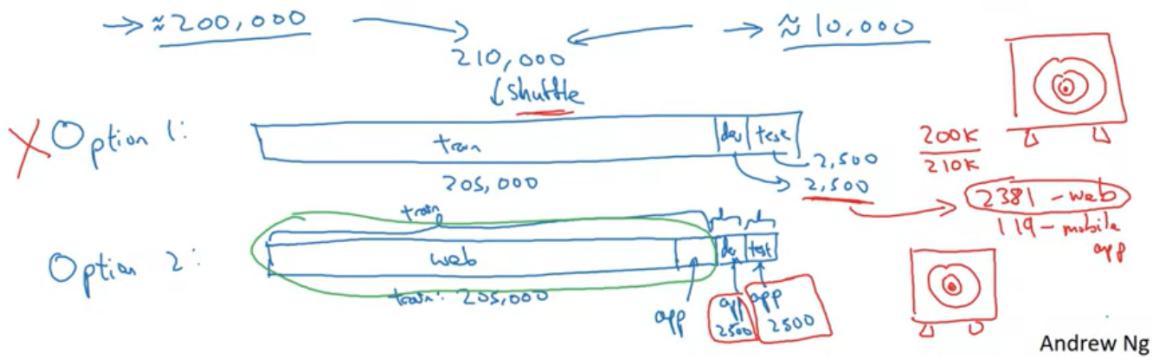
개발 시스템이 당신의 팀이 어디에 목표를 두는지 기억하십시오. 그리고 당신이 목표를 정하는 방식, 그리고 당신은 웹페이지 이미지 분포에 최적화된 시간을 사용하는 것

[추천하는 방식!]

Cat app example  
Data from webpages



*care about this*  
Data from mobile app



당신의 개발과 테스트 세트는 모두 모바일 앱 이미지일 것입니다.

그래서 훈련 세트는 웹에서의 200,000개의 이미지를 포함시킬 것입니다. 그리고 5,000개의 모바일 앱에서 포함시킬 것입니다. 개발 세트는 모바일 앱에서 2,500개, 테스트 세트는 모바일 앱에서 2,500개일 것입니다.

→ 당연히 단점은 당신의 훈련 분포가 당신의 개발과 테스트 세트 분포와 다르다는 것입니다. 그러나 이 데이터가 훈련 개발, 테스트로 나누어게 된 결과는 장기적으로 당신에게 훨씬 더 좋은 성과를 가져다 줄 것입니다.

## Speech recognition example

Speech activated rearview mirror



### Training

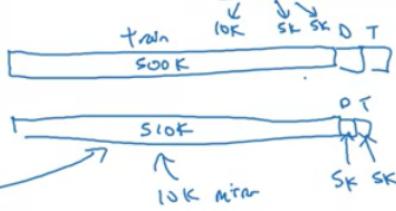
- { Purchased data       $\downarrow \downarrow$   
          X, y
- Smart speaker control
- Voice keyboard
- ...

500,000 utterances

### Dev/test

Speech activated  
rearview mirror

$\Rightarrow 20,000$



Andrew Ng

이 예에서는 당신의 훈련 세트는 왼쪽 500,000개의 발언과 개발 및 테스트들을 저는 축약해서 D와 T라고 하겠습니다. 이것들은 아마도 한 사람당 10,000 마디일 것입니다. 그것은 실제 음성인식 백미러에서 따온 것 입니다. 아니면, 당신이 20,000개의 모든 예를 음성인식 백미러의 개발과 테스트세트에서 넣을 필요가 없다고 생각하면, 그것의 반만 훈련 세트에 넣어도 됩니다.

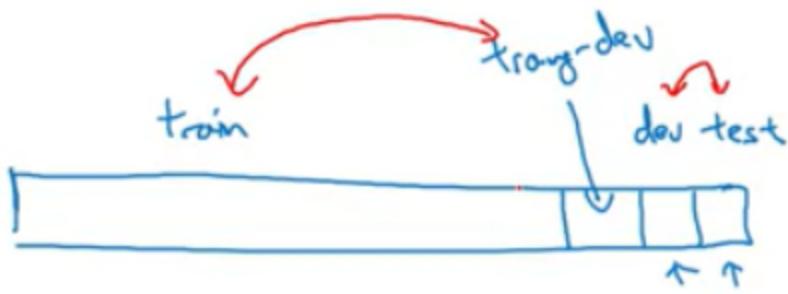
그럼 훈련 세트는 거기서 온 500명을 포함하여 백미러의 10,000건과 510,000개의 의 발언이 가능하고 그런 다음 개발 및 테스트 세트에서 각각 5,000건 정도 될 것 입니다 그러니까 20,000번의 발언 중에서 아마 10k는 훈련 세트에 들어가서 개발 세트에 5k 시험 세트에 5천 개가 될 것 입니다. 그리하여 이것은 또다른 합리적인 데이터를 훈련, 개발, 테스트로 나누는 방법입니다.

## 다른 분포에서 오는 편향과 편차 분석법

9 퍼센트로 증가한 오류 중, 얼마만큼이 알고리즘이 dev set에 있는 데이터를 못봐서 생기는 오류인지, 이것은 편차에서 오는 문제겠죠. 아니면 얼마만큼의 오류가 단순히 dev set 데이터가 다르기 때문에 생기는 오류인지 구분하기가 쉽지 않습니다.



이전에는 일정 트레이닝세트와 dev set, 그리고 데스트세트들을 이렇게 세팅했습니다. 그리고 dev 와 테스트세트들은 똑같은 분포도를 가졌었는데요. 트레이닝세트는 다른 분포도를 가질 것입니다.



저희는 트레이닝세트들을 무작위로 섞어서 일부 트레이닝세트를 추출해서 training-dev set라고 할 것입니다. 그러면 dev와 테스트세트가 동일한 분포도로 되어있듯이, 트레이닝세트와 training-dev set도 마찬가지로 동일한 분포도로 되어 있을 것입니다.

하지만 **다른점**은, 이제 여러분이 트레이닝 세트에서만 제대로 신경망을 트레이닝 시킨다는 점입니다. 신경망을 training-dev 부분에서는 따로 트레이닝 시키지 않을 것입니다. 오류분석을 하기위해서는, 트레이닝세트에서 인식기의 오류를 보고, 또 training-dev set에서 보고, 또 dev set에서도 볼 것입니다.

### 가정 1.

- training오류 : 1%
- traing-dev set 오류: 9%
- dev set 오류 : 10%

⇒처음 보는 익숙하지 않은, 같은 분포를 갖는 데이터에 일반화가 잘 되지 않음 확인

⇒ 편차문제가 있음!

## 가정 2.

- training오류 : 1%
- traing-dev set 오류: 1.5%
- dev set 오류 : 10%

⇒ 편차는 낮음

⇒ 전형적인 데이터 미스매치 문제

## 가정 3.

- Bayes error에 대한 인간레벨 프록시 값 : 대략 0%
- training오류 : 10%
- traing-dev set 오류: 11%
- dev set 오류 : 12%

⇒ avoidable bias problem

⇒ high bias setting

## 가정 4.

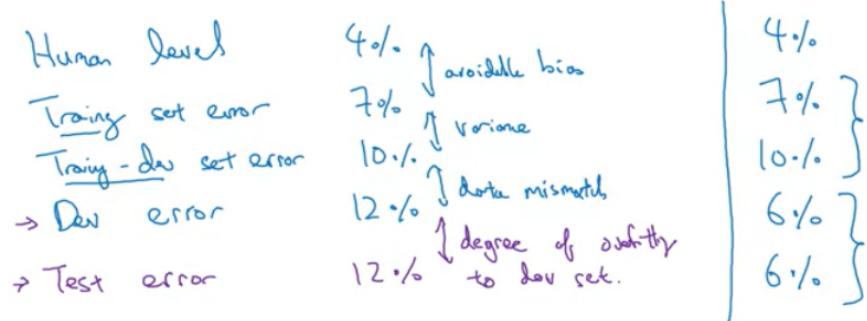
- training오류 : 10%
- traing-dev set 오류: 11%
- dev set 오류 : 20%

⇒ avoidable bias problem 큼

⇒ 데이터 미스매치 문제 매우 큼!

| 가장 핵심적으로 볼 부분은 인간레벨 오류, 여러분의 트레이닝 세트 오류,  
| 그리고 여러분의 trainin-dev set 오류입니다.

## Bias/variance on mismatched training and dev/test sets



Andrew Ng

얼마나 트레이닝세트에서 training-dev set로 일반화가 될까요?

dev set를 overfit했을 경우, 고려하실 수 있는 부분은 다시 돌아가서 dev set 데이터를 더 수집하는 것입니다.

만약...

인간레벨성능이 4퍼센트, 트레이닝 오류가 7퍼센트, training-dev 오류가 10퍼센트인경우, dev set로 간다고 해봅시다. 놀랍게도, dev set에서 더 작 작동하는 점을 발견했다고 해봅시다.

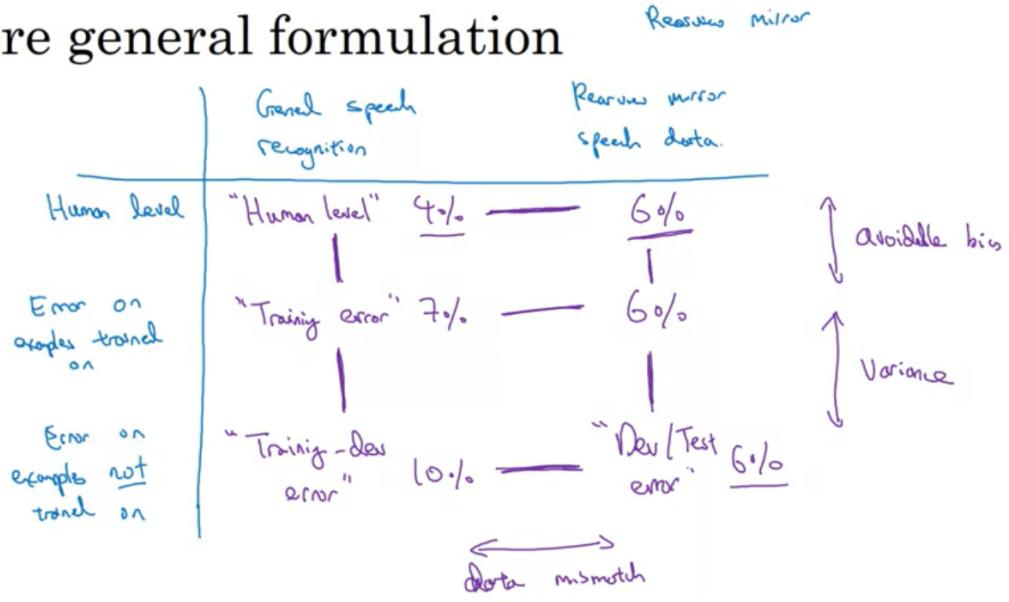
⇒ 트레이닝 데이터가 dev set과 테스트세트보다 훨씬 더 어려워진 경우!

Ex)

여러가지 음성인식 문제에서 수집한 뭉치의 데이터

vs 백미러와 관련된 특화 스피치 데이터

## More general formulation



Andrew Ng

## Data Mismatch를 다루는 법

- 개발 세트 오류의 특성을 파악할 수 있다면 아니면 개발세트가 교육 세트와 어떻게 다른지와 얼마나 더 어려운지에 대한 특성을 파악할 수 있다면 당신은 후에 교육 데이터를 보다 유사하게 만들 수 있는 방법을 찾습니다.
- 또는 개발 및 테스트 세트와 유사한 데이터를 더 수집해 보십시오. 예를 들어, 뒤에서 들리는 자동차 소음이 가장 큰 오류 원인이라는 것을 알게 되면 당신이 취할 수 있는 하나의 방법은, 차량 내에서 잡음이 심한 데이터를 시뮬레이션할 수 있습니다.

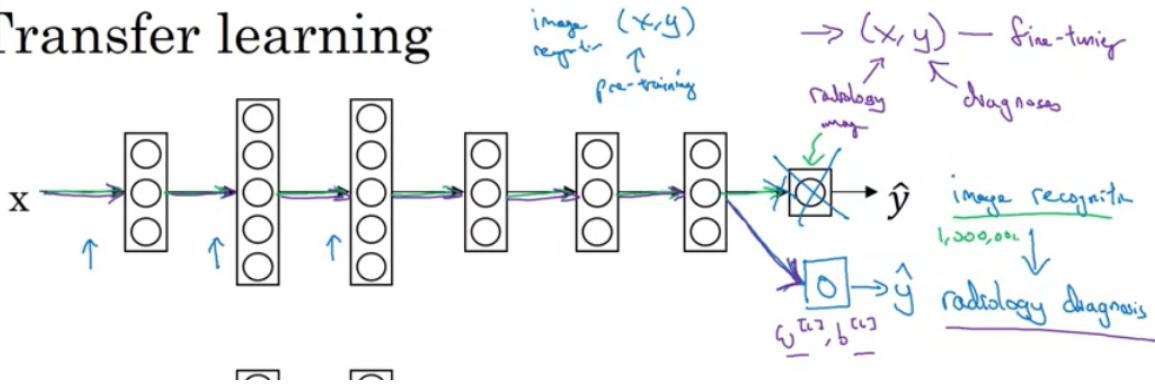
요약하자면, 데이터 불일치 문제가 있다고 생각하시면, 오류 분석을 할 것을 권합니다. 아니면 교육 세트를 보는 것을 권하고, 이 수치를 시험해 보려면 개발 세트를 보세요 이 두 데이터 분포가 어떻게 다를 수 있는지에 대한 통찰력을 얻으려면 그럼 다른 방법을 찾아서 개발 세트와 약간 유사한 교육 데이터를 수집할 방법을 찾으세요. 우리가 얘기했던 방법 중 하나는 인공 데이터 합성입니다.

가능한 모든 예제 공간의 작은 하위 집합에서만 데이터를 시뮬레이션하십시오. 바로 이것이 데이터 불일치를 해결하는 방법입니다.

## Transfer Learning

신경망 네트워크가 고양이와 같은 것을 인식할 수 있도록 러니시킬 수 있고, 그 지식을 이용해서 부분적으로 x-ray 스캔을 읽는데 도움이 될 수 있도록 만들 수 있습니다. ⇒ "트랜스퍼 러닝"

### Transfer learning



신경망 네트워크의 마지막 결과값 층인 이 부분을 가지고, 이것은 삭제하구요, 이 마지막 결과값 층에 feeding하는 이 weight들도 삭제합니다. 그런 뒤에, 마지막 층을 위한 무작위로 초기화된 weight 세트를 생성합니다. 이렇게 생성한 것들이 방사선학 진단을 결과값으로 표출할 수 있게 하는 것이죠. 구체적으로 말씀드리자면, 트레이닝 첫번째 단계에서 이미지 인식 업무 관련한 내용을 트레이닝 시키는 경우, **익숙한 신경망 parameter들을 트레이닝** 시키는데요. 모든 weight와 모든 층을 트레이닝 시키면 이렇게 이미지를 인식하고 예측하는 프로그램과 같은 것도 생기는 것입니다. 이런 신경망을 트레이닝한 경우, 이제 트랜스퍼 러닝을 도입하기 위해서 데이터세트 **X와 Y에서 스왑을 진행합니다**. 이것은 이제 그러면 방사선 이미지인데이요. Y는 예측하고 싶은 진단에 대한 부분인데요. 여기서 해야하는 부분은 **바로 마지막 층의 weight를 초기화시키는 것입니다**. 이것을 **W.L**이라고 하겠습니다. 그리고 **P.L**이라고 랜덤으로 이야기 하겠습니다. 이제 새로운 데이터세트에서 신경망을 다시 트레이닝 시키는데요, 새로운 방사선학 데이터세트에서 말이죠. 신경망을 방사선학 데이터로 재차 트레이닝 시키는 방법이 몇 가지 있는데요.

- 만약에 작은 양의 방사선학 데이터세트가 있는 경우, 마지막 층의 weight만 다시 트레이닝 시킬 수 있습니다. W.L.P.L만 말이죠. 그리고 나머지 매개 변수는 고정시키는 것입니다.

→ 결과값 층에서 마지막 층만 retrain 시킵니다.

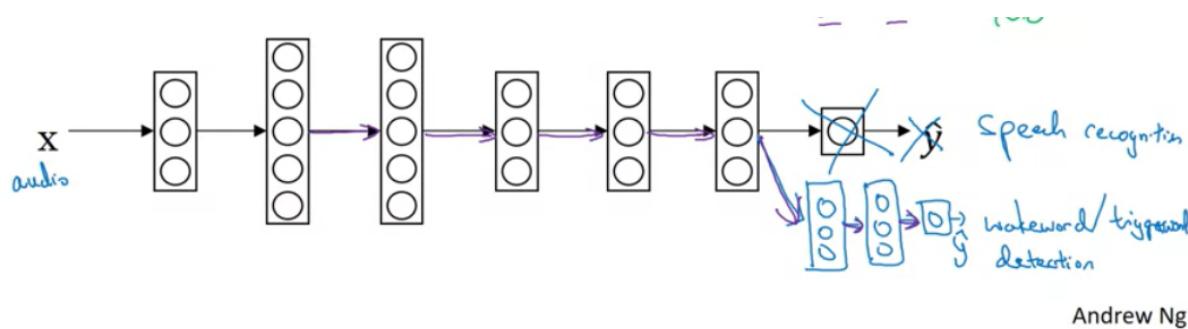
2. 데이터가 충분히 있는 경우, 나머지 신경망에 대한 부분도 모든 층을 다시 트레이닝 시킬 수 있습니다.

→ 네트워크에서 있는 모든 매개 변수를 다시 트레이닝

→ 첫번째 단계의 트레이닝을 pre-training

→ 그리고 weight를 나중에 업데이트 하는 경우, 방사선학 데이터에 트레이닝 시키는 것은 가끔 fine tuning이라고도 합니다.

것이 도움을 줄 수 있는 이유는, edges를 감지하는 특성, 커브 감지, positive object를 감지하는 특성들과 같이 수 많은 low level 특성 때문입니다.



또 다른 예제를 보겠습니다. 또 하나의 예제를 보도록 합시다. 여러분이 음성인식 시스템을 트레이닝 했다고 해보겠습니다. 그러면 이제  $X$ 는 오디오의 입력값 또는 오디오 단편입니다.  $Y$ 는 어떤 ink 기록이구요 그러면 여러분은 음성인식 시스템을 통해 transcript를 결과값으로 만듭니다. 이제 여러분은 "wake word" 또는 "trigger word" 를 감지하는 시스템을 만들고 싶다고 해보겠습니다. 이전에 말씀드렸지만, wake word 또는 trigger word는 음성인식기기를 깨우기 위해 또는 작동시키기 위해 말하는 단어입니다. 예를 들어, Amazon Echo를 작동시키기 위한 "Alexa"라는 단어, 또는 구글을 깨우기 위한 "OK Google"이라는 단어, 또는 애플 기기를 깨우기 위한 "hey siri" 또는 바이두 기기를 깨우기 위한 "ni hao Baidu" 와 같은 단어가 이에 해당합니다.

이렇게 만들기 위해서는, 신경망 네트워크의 마지막 층을 꺼내서 새로운 결과값 노드를 만들어야 할 것입니다. 가끔씩 또 할 수 있는 방법은, 1가지 새로운 결과값을 생성하는 것이 아니라, 여러개의 새로운 층을 신경망에서 생성시키는 것입니다.  $Y$  레이블을 wake word 감지 문제에 적용시키기 위해서 말이죠. 여러분이 얼마나 많은 데이터를 보유하고 있는지에 따라, 네트워크의 새로운 층을 다시 트레이닝 시키거나, 또는 신경망의 층들을 더 많이 트레이닝 시키는 방법도 있습니다. 그러면 언제 transfer learning이 말이 되는 것일까요?

transfer learning이 말이 되는 경우는, 여러분이 전송하려고 하는 곳의 데이터가 많고, 전송하는 곳의 문제 관련 데이터가 적은 경우 좋습니다.

많은 데이터를 보유하고 있는 문제에서 적은 데이터를 가지고 있는 문제로 transfer 시키는 것입니다.

1. A업무와 B업무가 같은 입력값을 가지는 경우
2. A업무의 데이터가 훨씬 더 많아야 합니다.
3. 여러분이 A업무의 low level 틀성들이 B업무를 배우는데 도움이 될 수 있다 판단되는 경우입니다.

## Multi-task Learning

- transfer learning : 순차적인 절차가 있었는 반면에, A업무를 배우고 B업무로 넘어가는 절차 말이죠
- multi-task 러닝 : 동시에 시작

## Simplified autonomous driving example



$$\begin{aligned} & \text{Pedestrians} & y^{(i)}_1 \\ & \text{Cars} & y^{(i)}_2 \\ & \text{Stop signs} & y^{(i)}_3 \\ & \text{Traffic lights} & y^{(i)}_4 \\ & \vdots & \vdots \\ & Y = \left[ \begin{array}{c|c|c|c} y^{(1)}_1 & y^{(1)}_2 & y^{(1)}_3 & \dots & y^{(1)}_4 \\ \hline y^{(2)}_1 & y^{(2)}_2 & y^{(2)}_3 & \dots & y^{(2)}_4 \\ \hline \vdots & \vdots & \vdots & \ddots & \vdots \\ \hline y^{(m)}_1 & y^{(m)}_2 & y^{(m)}_3 & \dots & y^{(m)}_4 \end{array} \right] & (4, m) \end{aligned}$$

Andrew Ng

$y(i)$ 의 dimension : 보행자유무, 차량유무, 정지sign유무, 신호등 유무

전체 트레이닝세트에서의 평균 loss는 1 나누기  $m$  의 합,  $i$ 가 1에서  $m$ 까지와, 그리고 개인별 예측의 loss에 대해서  $j$ 가 1에서 4까지의 합입니다.

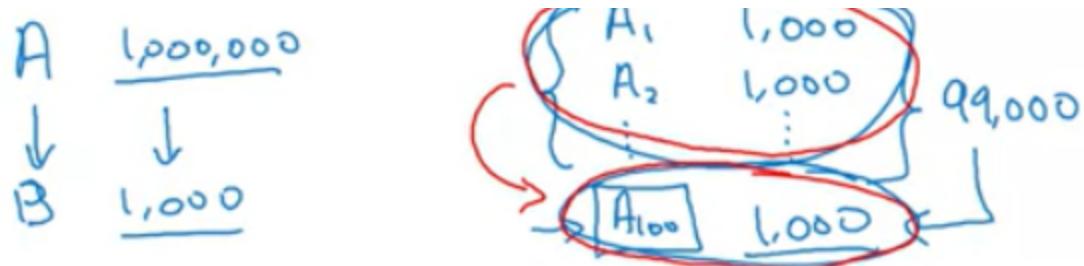
| 4가지의 요소에서, 보행자, 자동차, 장지 sign, 신호등에 대해서 그 값을 더하는 것

이것과 softmax regression의 주요 차이점은, softmax regression은 single example에 대해서 single label을 부여했는데요, 이런 것과는 달리, 여기 이 이미지는 복수의 레이블을 가질 수 있습니다.

### 또 다른 방법은 4개의 개별 신경망을 트레이닝 시키는 것

| Multi tasking이 말이 되는 경우

1. shared low-level features로부터 이득을 볼 수 있는 업무들을 트레이닝 시킬 때  
→ 자율주행차 같은 경우엔, 신호등, 보행자, 차량을 인식하는 것이 stop sign과 같은 것을 인식하는 것에 도움을 줄 수 있는 특성과 비슷할 것입니다. 이것들은 모두 도로의 특성을 가지고 있기 때문입니다.
2. (필수 조건은 아님) 각각의 업무에 있는 데이터 량이 서로 비슷할 때 다른 업무의 총 데이터 합산 양이 다른 업무에 비해 훨씬 더 많아야 한다는 것



→ 시스템적으로 또 다른 99개의 다른 업무들이 어떠한 데이터를 제공하거나 지식을 제공하여 100개의 업무에게 도움을 줄 수 있습니다.

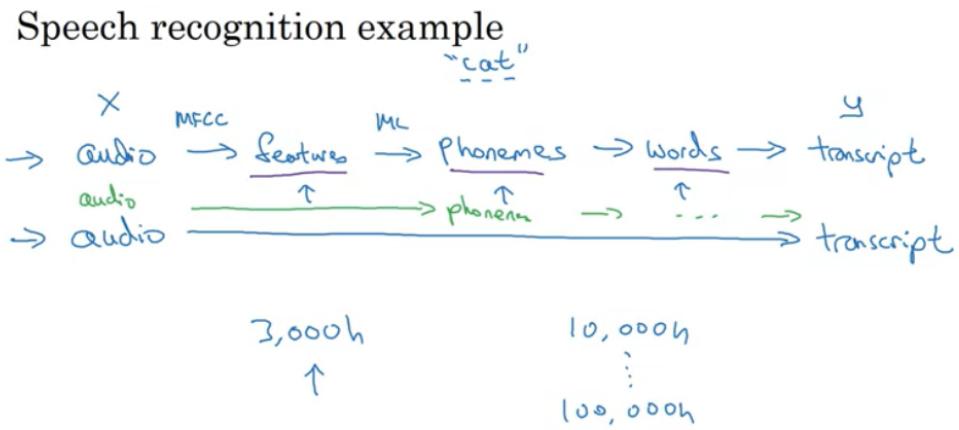
## [요약]

multi-task learning 은 transfer learning보다는 훨씬 덜 쓰이긴 합니다. 예외적인 분야는 computer vision object detection인데요, 여기서는 많은 수의 다른 물체들을 감지하는데 신경망을 트레이닝 시키는 경우입니다. 이런 경우에는, 신경망을 개별적으로 트레이닝 시키는 것보다 더 잘 작동합니다.

## end-to-end 딥러닝

엔드 투 엔드 딥 러닝이 하는 일은 이런 여러 단계를 거치는 것들을 수행하고 **하나의 신경 네트워크로 변환** 합니다.

## What is end-to-end learning?



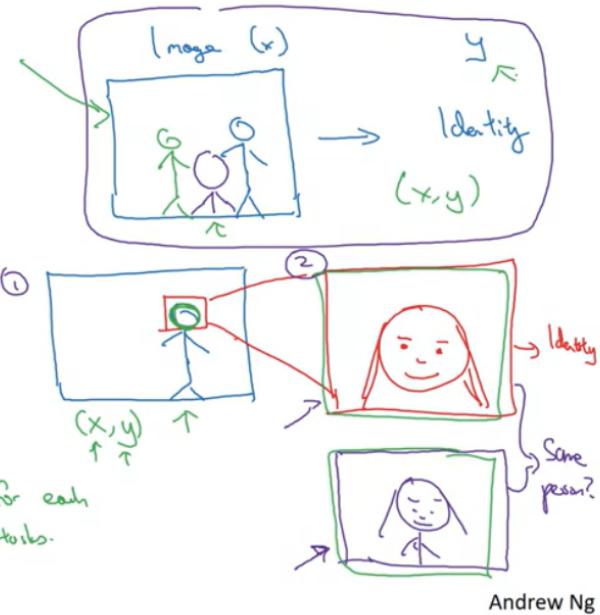
Andrew Ng

# Face recognition



[Image courtesy of Baidu]

Have data for each  
of 2 subjects.



Andrew Ng

여기서  $X$ 는 회전식 회전식 출입문에서 찍힌 것과 같은 이미지이고,  $Y$ 는 그 사람의 정체입니다.

1. 첫번째는 얼굴이 어디에 있는지 알아내는 것입니다.
2. 두번째로, 얼굴을 보고 이것이 누구인지 알아 내는 것입니다.

→ 빨간색 사진이 만명의 직원 중 한명이라는 알아내고 당신 사무실 건물 안으로 들어갈 수 있을 겁니다.

## 왜 두 단계 접근 방식이 효과가 있을까요?

1. 여러분이 해결하고 있는 두가지 문제가 실제로는 훨씬 더 단순하다
2. 두개의 하위 작업에 대한 데이터가 많다

엔드 투 엔드 티핑은 기계 번역에 매우 효과적입니다. 그것은 오늘날  $X-Y$  쌍의 큰 데이터 세트를 모으는 것이 가능하기 때문입니다 그것이 영어 문장이고 그것이 프랑스어 번역이기 때문이죠.

## [단점]

1. 첫째, 데이터가 많이 필요할 수 있습니다. 이  $X-Y$  매핑을 직접 확인하려면 많은  $X, Y$  데이터가 필요합니다.

## 2. 유용하게 설계된 수동 구성품을 배제한다는 점

→ 데이터가 많지 않을 때는 조심스럽게 손으로 설계 한 시스템을 사용하면 실제로 인간이 문제에 대한 많은 지식을 알고리즘 데크에 투입 할 수 있고, 매우 도움이 될 것