ML Strategy (2)

Error Analysis

mislabeld dataset examples

머신러닝에서 가끔씩 hand engineering에 대해 폄하하는 것이 있는데 적용시스템을 만드는 경우 이렇게 간단히 숫자를 세는 과정이 오류 분석에서 많은 시간을 절약해 준다. 어느 것이 가장 중요한지를 결정하는데 또는 가장 적합한 방향은 어디인지, 어느 방향으로 집중해야 하는지에 있어서.

Evaluate multiple ideas in parallel

Idea for cat detection:

1. Fix pictures of dog being recognized as cats
2. Fix great cats(lions, panthers, etc) being misrecognized.
3. Improve performance on blurry images.

error alaysis using table

Image	Dogs	Great Cats	Blurry	Camera Filter	Comments
1	o			o	pitbull
2			o	o	
3		o	o		rainy day at zoo
⋮	⋮	⋮	⋮	⋮	
% of total	8 %	43 %	61 %	12 %	

분석의 결과: 흐릿한 이미지를 작업해야 한다는 직접적 결과나 단순한 수학 공식처럼 무엇을 어떻게 하라고 알려주는 것이 아니라 전반적으로 가장 선택하기 좋은 옵션에 대한 감각적인 부분을 제시해주는 것이다.

Cleaning up Incorrectly labeled data

Q. supervised learning에서 y(labels) 자체가 틀렸다면 어떻게 해야 할까? 하나씩 고치는 것이 합리적일까?

⇒ DL algorithms are quite robust to random errors in the training set.

error가 비교적 랜덤하다면 그대로 놔둬도 괜찮을 것이다. 고치는데 너무 많은 시간을 쏟을 필요 없이. 물론 training set으로 가서 직접 검사하고 수정해서 나쁠 것은 없다. 가끔씩은 그럴 가치가 있을 수 있지만 수정 안해도 괜찮을 수 있다. 전체 데이터셋 사이즈가 충분히 크고 실제 error의 비중이 너무 크지 않다면.

하지만 딥러닝 알고리즘은 systemic error에는 덜 robust하다. 예로 들어 대부분의 흰색 강아지를 고양이라고 인식할 수 있다.

dev/test set에 대해 걱정된다면 권장하는 것은 error alaysis에서 잘못 라벨링된 데이터를 세는 열을 추가하는 것이다.

error alaysis using table

Image	Dogs	Great Cats	Blurry	Incorrectly labeled	Comments
⋮	⋮	⋮	⋮	⋮	
98				o	labeler missed cat in background
99		o			
100				o	drawing of a cat; not a real cat
% of total	8 %	43 %	61 %	6 %	

Q. 여기서 잘못 라벨링된 6%를 고쳐야 할까?

⇒ ng 교수의 advise: 만약 알고리즘이 dev set에서 그 평가 능력이 현저히 개선되어 바뀔 수 있다면 시간을 써서 레이블이 틀린 것을 고치는 것 괜찮다. 하지만 dev set으로 그 성능이 크게 바뀌지 않는다면 라벨링을 고치는 것은 비추천이다.

ex)

	example 1	example 2
Overall dev set error	10 %	2 %
Errors due incorrect labels	0.6 %	0.6 %
Errors due to other causes	9.4 %	1.4 %

ex 1의 경우 에러 해결의 중요한 부분은 9.4%이지 0.6%가 아니다.

ex 2의 경우 2% 중 0.6%로 전체 오류 중 30%에 해당하므로 틀린 레이블 수정 작업이 가치가 있다.

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

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.

Guideline: Build your first system quikly, then iterate

1. Set up dev/test set and metric.(목표)
2. Build initial system quikly.
3. Use Bias/Variance analysis & Error analysis to prioritize next steps.

ML 모델에 접근할 수 있는 다양한 방법들이 있는데 모두 합리적이고 시스템 개선에 도움을 준다. 문제는 정확히 어떤 방법을 골라서 집중하느냐이다. 완전히 새로운 ML 모델을 만든다면 가장 먼저 시스템을 빠르게 만들고 반복 테스트를 진행하는 것을 추천한다.

Mismatched training and dev/test data

Training and testing on different distribution

기존 데이터와 다른 distribution을 가진 새로운 데이터가 들어왔을 때 data set 비율

기존 데이터 from web page ＞ # 새로운 데이터 from user mobile app

option 1: 모두 다 합친 후 randomly suffle

train set	dev set	test set

장점: train 과 dev/test이 모두 같은 분포로 되어 관리가 쉽다.
단점. 매우 큰 단점: 진짜로 원하는 대상인 data from mobile app에 대한 dev set의 비율이 매우 작다. 서비스를 시작할 때는 mobile app에서의 사진으로 작동돼야 하는데 dev에 많은 부분이 web pages이기 때문에 정확한 측정이 안 된다.
따라서 option 1을 추천하지 않는다.

option 2:

train set		dev set	test set
기존 데이터	새로운 데이터		

option 2를 추천한다.
장점: 대상에 대한 조준을 확실히 할 수 있다.
단점: train과 dev/test data set의 distribution이 다르다. 그러나 train과 dev/test data set으로 나뉜 결과는 장기적으로 훨씬 더 좋은 성과를 가져다준다.
train과 dev/test data set의 distribution이 다를 때 다루는 기술이 있다.

Q. 가지고 있는 모든 데이터를 항상 다 쓸 것인가?
⇒ 대답은 미묘하다. 항상 모든 데이터를 쓰는 것은 아니다.

Bias and Variance with mismatched data distributions

bias와 variance를 예측하는 것은 다음에 어떤 업무를 해야 할지 우선 순위를 정하는 데 도움을 준다. 하지만 train과 dev/test set이 다른 distribution을 가진다면 bias와 variance를 분석하는 방법이 달라진다.

ex)

Training error	1%
Dev error	10%

training set과 dev set이 같은 distribution을 가졌다면 큰 편차는 큰 문제다. 제대로 일반화하지 못했다는 것이기 때문이다.
하지만 다른 distribution를 가졌다면 쉽게 판단하기 어렵다. train set에는 잘 적용됐으나 다른 distribution를 가진 dev set에는 잘 작동하지 않을 수 있기 때문이다.

training error에서 dev error로 넘어갈 때 2가지가 동시에 변했다.
1. 알고리즘이 training set은 보았지만 dev set은 보지 못했다.
2. dev set의 distribution은 training set의 것과 다르다.

따라서 9%의 차이에서 얼마가 알고리즘이 dev set에 있는 데이터를 못 봐서 생기는 error이기 때문인지 아니면 단순히 dev set이 다르기 때문인지 구분하기 어렵다.

Training-dev set:

Same distribution as training set, but not used for training.

train set	training-dev set	dev set	test set

dev와 test set이 같은 분포를 갖고 training과 training-dev set도 같은 분포를 가진다.
※ training set에서만 학습하고 training-dev에서는 따로 학습시키지 않는다.

ex)

Human error	-	-	0 %	0 %
Training error	1 %	1 %	10 %	10 %
Training-dev error	9 %	1.5 %	11 %	11 %
Dev error	10 %	10 %	12 %	20 %
	Variance	Data Mismatch	Bias	Bias & Data Mismatch

Bias/Variance on mismatched training and dev/test sets

Human level ≈ Bayes error
↕ Avoid Bias
Training set error
↕ Variance
Training–dev set error
↕ Data mismatch
Dev error
↕ degree of Overfitting to dev set
Test error

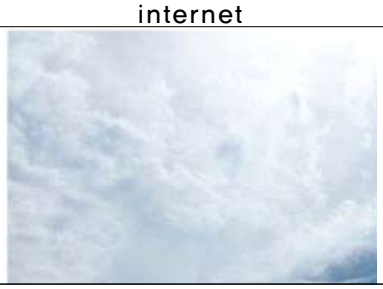
More general formation

	examples trained on		new examples not trained on
Human level	Human level		
	↕ Avoidable Bias		
Error on examples trained on	Training error		
	↕ Variance		
Error on examples not trained on	Training–dev error	⟷ Data mismatch	Dev/Test error

Addressing data mismatch

- 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

image from front–facing camera		foggy image from the internet		sythesied foggy image
	+		=	

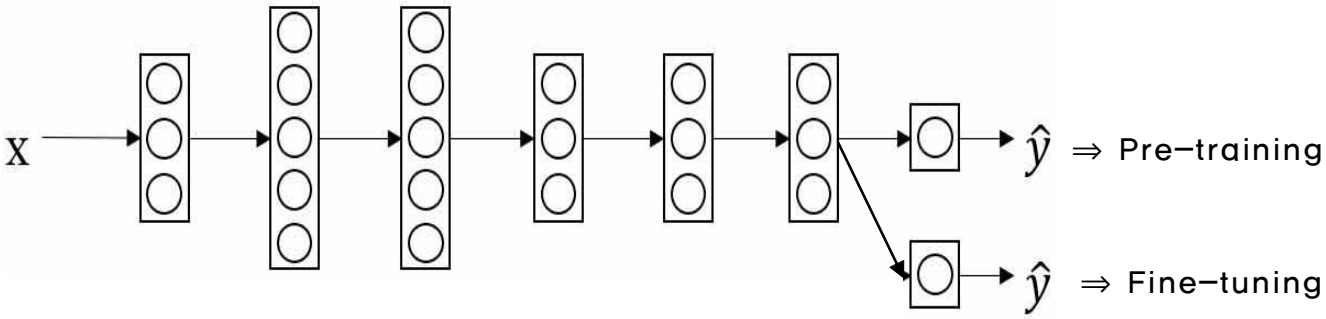
data mismatch 문제가 있다면 error analysis를 하거나 training set과 dev set의 두 error를 비교해 보는 것을 권한다. 두 데이터 분포가 어떻게 다를 수 있는 보려면 dev set과 약간 유사한 training data를 수집할 다른 방법을 찾아봐야 한다. 그중에 하나가 artificial data synthesis이다. 성능을 향상시키는 효과가 있다. artificial data synthesis를 사용할 때는 실수가 있는지를 조심해야 한다. 가능한 모든 예제 공간의 작은 하위 집합에서만 데이터를 시뮬레이션하기를 권한다.

Learning from multople tasks

Transfer learning

: 이미 학습되어있는 모델을 다른 분야에 적용

데이터가 충분하지 않으면 나머지 layer의 prameters는 고정한 채 마지막 layer 혹은 마지막 2개의 layer의 parameters만 학습시킨다. 데이터가 충분하다면 나머지 부분도, 즉 모든 layers의 parameters을 다시 트레이닝시킬 수 있다.



Pre-training(Pre-initializing): 첫 번째 단계의 training

Fine-tuning: 다른 유형의 데이터를 트레이닝시켜 weights를 다시 업데이트하는 경우

ex) Pre-training: image recognition / Fine-tuning: radiology diagnosis

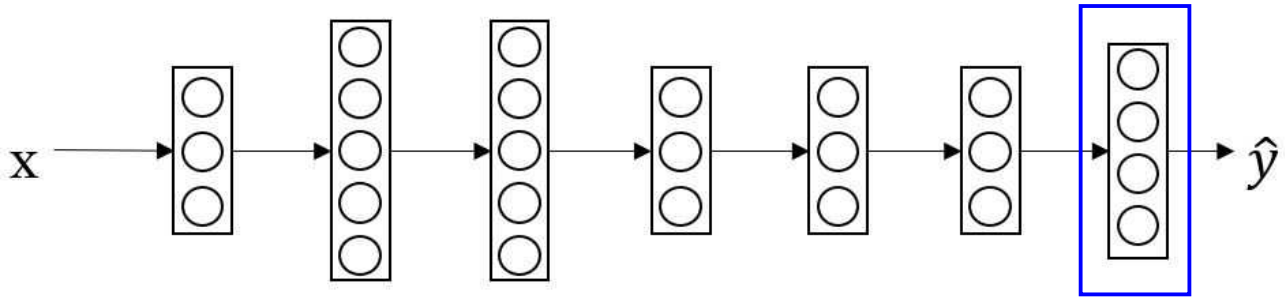
image recognition로 1,000,000장의 데이터를 가지고 있고 radiology diagnosis로 100장의 데이터를 가지고 있다면 radiology diagnosis를 위한 것에 비해 데이터가 너무 적다. 반대로 image recognition이 100장 radiology diagnosis가 1000이면 굳이 pre-train과 transfer learning을 할 필요가 없을 것이다.

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

transfer learnig은 A 이후에 B를 하듯 순차적으로 진행했다면 multi-task learning은 동시에 진행한다. 그래서 각 업무가 다른 업무들을 도와주길 기대한다.



Multi-task learning

unlike softmax regression:

one image can have multiple labels. softmax는 single example에 대해 single label을 부여했지만 여기서는 single image에 multiple labels을 가질 수 있다.

이 이미지가 어떤 레이블인지 맞추는 것이 아니라 이 이미지에 어떤 레이블이 있는지를 본다.

장점: 4개의 서로 다른 신경망을 만들어서 각각 따로 하는 것 보다 한꺼번에 학습시키는 것이 더 나은 성능을 보여준다.

ex) $Y = \begin{bmatrix} 1\,1\cdots 0\cdots ?\cdots \\ 0\,1\cdots 1\cdots 1\cdots \\ ?\,?\cdots 1\cdots ?\cdots \\ ?\,?\cdots 0\cdots ?\cdots \end{bmatrix}$

데이터에 따라 라벨링을 하지 않아 ?인 데이터가 있을 수 있다. 따라서 cost를 구할 때도 0 혹은 1로 라벨링된 것만 센다. 만약 첫 번째 데이터처럼 라벨링할 때 세 번째와 네 번째에 관심 없어 아무런 라벨링도 하지 않았다면 레이블이 있는 것만 2개에 대해서만 평균을 구한다.

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

When multi-task learning makes sense

- Training on a set of tasks that could benefit from having shared lower-level feartures.
- Usually: Amount of data you have for each task is quite similar.
- Can train a big enough neural network to do well on all the task.

실제로 보면 transfer learning이 muti-task learning보다 많이 사용되는 것을 볼 수 있다. multi-task learning을 하려면 많은 데이터가 필요하기 때문이다. computer

vision과 같이 예외가 있긴 하다.
data set의 크기가 비교적 작은 경우 transfer learning이 큰 도움을 줄 것이다. data set 크기가 훨씬 더 큰 문제를 찾으면 신경망을 트레이닝시키고, 그다음 데이터의 양이 적은 문제로 transfer 시킬 수 있다.

End-to-End deep learning

여러 단계의 처리를 신경 네트워크 하나로 변환한 데이터 프로세싱 시스템이나 학습 시스템

전통적인 방법								
audio	→	feature	→	phonemes	→	words	→	transcript

end-to-end		
audio	→	transcript

end-to-end가 잘 작동하려면 전통적인 방법에 비해 더 많은 데이터가 필요하다.
end-to-end를 위한 데이터의 양은 적다. 두 세 단계로 나누면 각 단계를 위한 데이터가 많아 end-to-end 보다 두 세 단계로 나누어 처리하면 성능이 더 좋을 수 있다.
ex) 얼굴로 누구인지 확인: 얼굴이 어디에 있는지 포커스 → db에 저장된 얼굴과 비교

왜 2 스텝이 1 스텝보다 성능이 좋은가?
1. 실제로 2 스텝의 문제가 1 스텝의 문제보다 단순하다.
2. 두 개의 하위 작업에 대한 데이터가 풍부하다.

장점
1. let the data speak
2. Less hand-designing of components needed.

단점
1. May need large amount of data
2. Excludes potentially useful handed-designed components.

Applying end-to-end deep learning
Key Q: Do you have sufficient data to learn a function of the complexity need to map x to y?
- you want to use machine learning or use deep learning to learn some individual components and 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.