

Coursera

Deep learning 흐름도

Andrew Ng

작성자: Github @cutthebutton

### \* 안내사항

- Coursera의 Andrew Ng 교수님의 Deep learning 강의 흐름도입니다.
- 트위터 @TessFerrandez의 노트를 기본으로 작성했습니다. 오류가 있을 수 있으므로 이해가 안된다면 참고하시길 바랍니다.  
(slideshare: <https://url.kr/r7fu2z>)
- 악필...입니다
- 딥러닝 영역에 어떤 내용이 있는지 훑어보기에는 괜찮을 겁니다.



## C1. Neural networks & Deep Learning

### Intro to Deep Learning

#### Supervised Learning (지도 학습)

: 학습 데이터에 정답 label이 붙어 있음

↔ Unsupervised Learning (비지도 학습)

• 종류 (NN Type) Input X Output Y

- Standard NN: 짐이 대한 특성 → 짐 가격

고객 특성 → 광고를 클릭할지 안 할지

- Conv NN : Image → Object

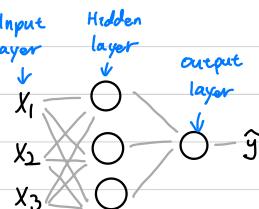
- Recurrent NN : Audio → text transcript

English → Chinese

- Custom / Hybrid

#### Network Architectures

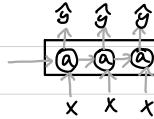
- Standard NN



- Convolution NN (CNN)



- Recurrent NN (RNN)



NN은 Structured Data와 Unstructured Data 모두 다룰 수 있음

(정형 데이터) (비정형 데이터)

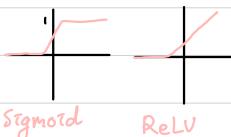
왜 지금 시점에서 Large NN이 대우될까?

· 多 데이터 + 多 기기 + optimized 알고리즘

Sigmoid 함수에서 ReLU 함수의 사용 이동은

더 빠른 Gradient Descent를 위함이며

NN 기술 발전의 Brg 돌파구 中 하나



#### Binary Classification



Logistic Regression

$$y = \hat{y} = \sigma(wx+b)$$

Linear Regression

$$\hat{y} = w_0 + w_1 x$$

T sigmoid

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

we b를 구해야 함 → But How?

A: Guess ( $\hat{y}$ , 추측)과 Truth ( $y$ , 실제값) 사이의 차이를 최소화  
함으로써 추측이 얼마나 좋은지 optimize 하는 것

$$\text{Loss} = f(\hat{y}, y) \rightarrow \text{Cost} = \text{mean}(\text{Loss})$$

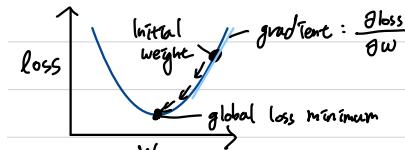
$$\text{Cost} = J(w, b) = \frac{1}{m} \sum_{i=1}^m f(\hat{y}_i, y_i)$$

Cost = Loss for the 전체 데이터 세트

#### Logistic Regression

Find the Minimum with Gradient Descent

→ Gradient Descent : 경사하강법



: Cost 함수인 J는 두개의 param을 가지는데 우리의 목표는 Cost func. 을 minimize 하는걸 ⇒ 두개의 param (w, b)을 줄여가며 J함수를 줄여갈걸

① Derivatives (미분)을 이용해 하강방향을 찾는다

② global minimum이 converge(수렴)할 때까지

$\alpha$ 의 learning rate로  $w$ 와  $b$ 를 업데이트 해나간다 (반복)

#### Putting it all together

$$X \rightarrow (z(x), \sigma(z)) \rightarrow \hat{y} : \text{mini neural net}$$

$$z(x) = wx + b \quad \hat{y} = \sigma(z) = \text{T sigmoid}(z)$$

• Forward Propagation :  $\hat{y}$ 를 계산해나가는걸

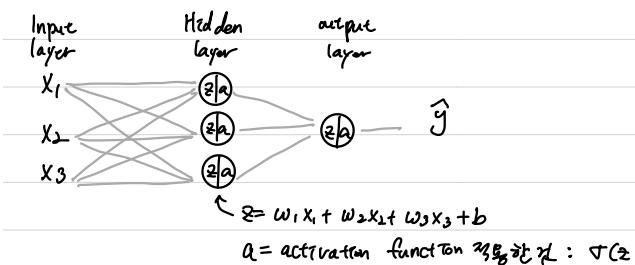
• Backward Propagation : 경사하강방법을 통해서  $w$ 와  $b$ 를 update  
Converge 되도록 가자



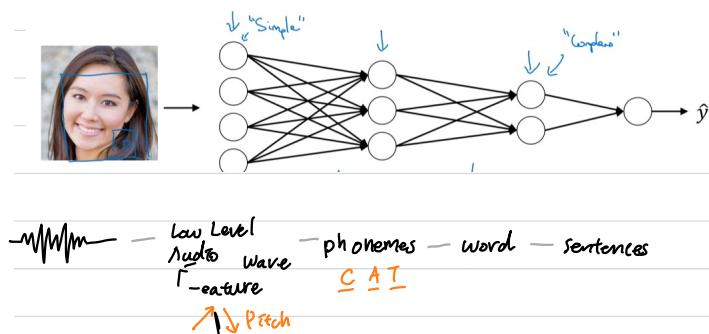
빠른 Computation은 process의 반복을 가속화하는데 중요

⇒ 빠른 가설 → 빠른 시도 → 빠른 update

## 2 layer Neural net



## Deep Neural Net



## Activation Function

- Sigmoid - Binary classification (이진 분류기)

- Only output layer에서 사용됨  
- Large/small values 정수가 되어 grad. descent가 느립

- Tanh

- Normalized version  
- grad. descent가 빠름

- ReLU

- activation 은 2줄짜리 가장 default  
- slope: 1 or 0

- Leaky ReLU

- Avoids zero slope at 0  
- 하지만 별로 안쓰임

## Shallow Neural Nets

## 왜 activation function을 사용하나?

ex) activation 이 없을 때 :  $a = z$

$$\text{layer 1} \quad | \quad a^{[1]} = z^{[1]} = w^{[1]}X + b^{[1]}$$

$$\text{layer 2} \quad | \quad a^{[2]} = z^{[2]} = w^{[2]} a^{[1]} + b^{[2]}$$

$$= w^{[2]} (w^{[1]} X + b^{[1]}) + b^{[2]}$$

$$= w^{[2]} w^{[1]} X + w^{[2]} b^{[1]} + b^{[2]}$$

$$= w' X + b' : \text{Linear function}$$

→ 굳이 NN을 쓰는 의미가 없음

## 와 b를 initializing 하는 것은?

만약 초기를 0으로 설정  $\Rightarrow$  모든 unit이 같아질 것이다

제일 같은 feature는 차이가 없겠지

$\Rightarrow$  SOLUTION : Random 한 Init

하지만 init이 작기를 바라니 Random(0.01) 해서 작게 만듬

Hyperparameter

## C2. Improving Deep NN

## Setting up your ML App.

## 데이터 세팅

<기존의 ML> : 1000 1000개 규모의 sample data

↓  $\rightarrow$  Train : 60% / Dev : 20% / Test : 20%

<최근 Deep Learning> : 1M 이상의 sample data

$\rightarrow$  Train : 98% / Dev & Test : 1% 각각

[!] Dev & Test set이라고 하면 여전히 많은 양을 얻을 수

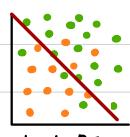
• Data는 같은 distribution 이어야 함

ex) Train data: 전문 사진 작가가 찍은 고양이

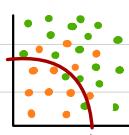
Dev/Test : 일반인의 폴카드 혹은 Blur羁기 20% 등

\* Dev & Test는 무조건 같은 distribution 이어야 함

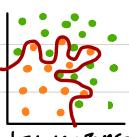
## Bias / Variance



high Bias  
"Underfit"



적당함



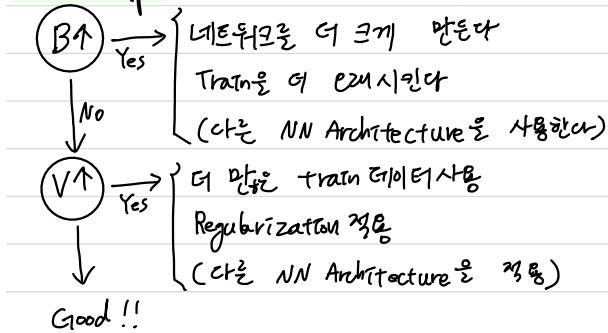
high variance  
"Overfit"

Error	Train	1%	15%	15%	0.5%
Test	11%	16%	30%	1%	V↑ B↑ V↑, B↑ Good!
	V↑	B↑	V↑, B↑		

$\therefore$  Bias : 적당에 균형하나 아니나 : 정확도와 연관

Variance : 데이터들이 퍼져있거나稠密 : 정밀도와 연관

## ML Recipe



## Other regularization

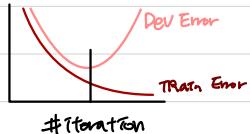
- Data Augmentation : 기존의 사진으로부터 조금 조작함



4 → 4 4

## Early stopping

Problem : Bias & Variance ↑



## Regularization : Preventing Overfitting

## L2 Regularization (정규화)

$$\text{Cost} : J(w, b) = \frac{1}{m} \sum_{i=1}^m f(y_i, \hat{y}_i) + \frac{\lambda}{2m} \|w\|_2^2$$

$$\star \|w\|_2^2 = \sum_{j=1}^n w_j^2 = w^T w$$

\* 2M개의 가중치는 미분함수에 차례로

\*  $\lambda$  : regulation parameter : python lambd

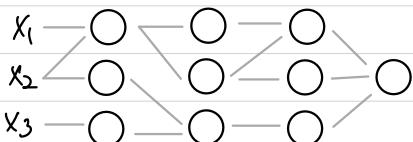
## L1 Regularization

$$\text{Cost} : J(w, b) = \frac{1}{m} \sum_{i=1}^m f(y_i, \hat{y}_i) + \frac{\lambda}{m} \|w\|_1$$

- L2가 L1보다 많이 사용됨

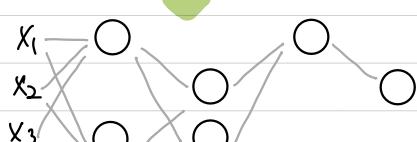
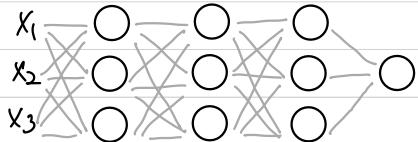
- 둘다 큰  $w$  (weight)을 처벌함 (penalize)

→ 어떤  $w$ 가 초기 가중치임  $\Rightarrow$  simple한 네트워크로 됨



## Dropout

Learning의 각 iteration에서 몇몇 node를 random하게 drop함

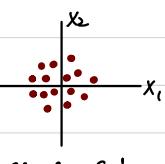
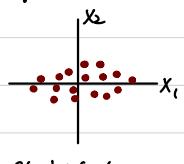
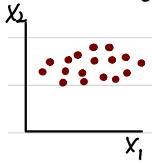


→ 더 simple한 network

style feature에 의존할 확률이 적어짐

## Optimizing Training

## Normalizing Inputs



Step 1 : Center  
around  $\mu, \sigma$

Step 2 : Scale  
 $\Rightarrow$  variance is same  
ex)  $-1 \rightarrow 1$

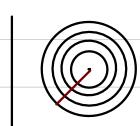
\* Dev/test를 normalize 할 때 같은 avg var 사용해야함

## What normalizing을 하나?

<normalize 안됨>



<normalize 됨>



: 예전에의 연습문제에서  
등면 모양으로 그림  
각 번의 고개

만약이 normalize가 되어있다면 커다란 learning rate  
 $\alpha$ 를 가능케 함

\* learning rate  $\alpha$

: gradient descent 할 때 경사각을 얼마나 내려갈지  
정하는 hyper param

## Vanishing / Exploding gradient

ex) Deep NN (L Layers),  $\hat{y} = w^{L-1} w^{L-2} \dots w^1 x + b$

$$\text{If } w = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \Rightarrow 0.5^{L-1} \Rightarrow \text{vanishing}$$

→ Vanishing gradient: 예전과 과정에서 일어나는 오류를  
가능성이 점점 감소

$$\text{If } w = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \Rightarrow 1.5^{L-1} \Rightarrow \text{exploding}$$

→ Exploding gradient: “ 가능성이 점점 증가 ”

• 두 케이스 모두 gradient descent 하는데 시간이 오래 걸림

⇒ Partial Solution

Initial value를 매우 신중히 선택

$$\text{- ReLU: } w^{[e]} = \text{rand} * \sqrt{\frac{2}{n^{e-1}}}$$

$$\text{- Tanh: } w^{[e]} = \text{rand} * \sqrt{\frac{1}{n^{e-1}}}$$

## Gradient checking

• 만약 cost 가 각 iteration에서 줄어들지 않는다면

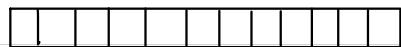
“ Back prop. bug ” 일 수도

• Gradient checking 으로 gradient 를 추적해본다.  
계산을 정확할 수 있음

• \* 매우 느리기 때문에 debugging 이지만 사용할 것

## Optimization Algorithm

## MINI - Batch Grad. Descent



데이터를 여러 Batch 로 나누는 것

→ 각 Batch 후에 Grad. Descent 함



## Mini-Batch 크기 선택

Size  $m$ : Batch grad. descent

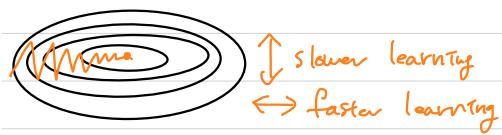
| 1 : stochastic(확률적) grad. descent

Batch(m)	middle	Stochastic(1)
각 step에서	빠름	비교적에서
사진증소모임	복잡사용방	모든 주제를 읽음

\* 만약 data가 2000개보다 적으면 Batch size = 2000

만약 그 이상이면 64, 128, 256, ... : CPU와 GPU에 맞춰

## Gradient Descent : W. Momentum



더빨리 목표점에 도착하기 위해 세로방향  $\uparrow$  의 진동을 줄여야 함

⇒ Solution : curve를 매끄럽게 만드는데 derivatives의

exponentially weighted average 를 취함

$$\text{+ exponentially weighted average } v_t = \beta_1 v_{t-1} + (1-\beta_1) \partial e$$

## RMSProp - Root Mean Squared

moving average 를 사용해서 gradient 를 normalize 함

$$Sdw = \beta_2 Sdw + (1-\beta_2) dw^2 \quad w = w - \alpha \frac{dw}{\sqrt{Sdw + \epsilon}}$$

$$Sdb = \beta_2 Sdb + (1-\beta_2) db^2 \quad b = b - \alpha \frac{db}{\sqrt{Sdb + \epsilon}}$$

## Adam optimization

momentum + RMSProp

## Learning rate Decay

: 매우 큰 learning rate  $\alpha$  를 사용할 시  
목표에 균형까지 수록 속도가 감소하는 것

< 해결책 >

$$\textcircled{1} \quad \alpha = \frac{1}{1 + \text{decay rate} \cdot \text{epoch}} \times \alpha_0$$

$$\textcircled{2} \quad \alpha = 0.95^{\text{epoch}} \times \alpha_0 \quad (\text{exponential})$$

$$\textcircled{3} \quad \alpha = \frac{h}{\text{epoch}} \times \alpha_0$$

$$\textcircled{4} \quad \alpha = \frac{h}{\text{E}} \alpha_0$$

$$\textcircled{5} \quad \alpha = \frac{h}{\text{epoch}} \quad (\text{Discrete stair case})$$

\textcircled{6} manual

\* epoch: 데이터를 한 번씩 예측 모델에 들어온 듯  
weigte 값을 갱신하는 주기

## Hyper Param. Tuning

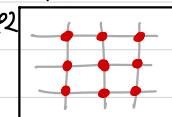
## 어떤 Hyper Param. 이 가장 중요할까?

- ↑  $\alpha$  learning rate
- # Hidden layer
- Minibatch size
- $\beta$  momentum ( $\text{tun} = 0.9$ )
- # layers
- learning rate decay
- $\beta_1 = 0.9 \quad \beta_2 = 0.999 \quad \epsilon = 10^{-8}$  (ADAM)

## Testing values

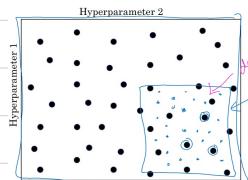
## &lt;Classic ML&gt;

HP1



- Grid Search
- 단계: one iter. after evaluation

⇒ Solution



random search + coarse dense  
= random search + 구조화된  
data sampling

## Multiclass classification

각각 구조화된 class가 2개 이상이면? ex) 4개면?

→ Soft max activation

$$t = e^{z^{[i]}} \quad a^{[i]} = \frac{t}{\sum t_i}$$

⇒

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \begin{matrix} P(Y_1|X) \\ P(Y_2|X) \\ P(Y_3|X) \\ P(Y_4|X) \end{matrix}$$

$$ex) z^{[i]} = \begin{bmatrix} 5 \\ 2 \\ -1 \\ 3 \end{bmatrix} \quad t = \begin{bmatrix} e^5 \\ e^2 \\ e^{-1} \\ e^3 \end{bmatrix} = \begin{bmatrix} 148.4 \\ 7.3 \\ 0.4 \\ 20.1 \end{bmatrix}$$

116.3

$$\Rightarrow a^{[i]} = \frac{t}{116.3} = \begin{bmatrix} 0.842 \\ 0.042 \\ 0.02 \\ 0.114 \end{bmatrix} = 11.4\%$$

## C3. Structuring ML Projects

## Structuring your ML projects

## Setting your goal

Goal은 핵심의 수준으로 설정되야 한다

	Precision	Recall
A	95%	90%
B	98%	85%

→ A와 B가 어떤 차이?

	Precision	Recall	F1
A	95%	90%	92.4%
B	98%	85%	91%

\* F1 score: Recall과 Precision 합의 조화평균 (Harmonic mean)

Optimizing 알고리즘? &amp; metric을 만족시킬지

	Accuracy	Runtime	Notes
A	90%	80ms	Accuracy는 좋지만 Runtime이 높음
B	92%	95ms	Accuracy는 좋음
C	95%	130ms	Runtime은 좋지만 Accuracy는 낮음

## Selecting your Dev/test set

ex) Dev set: 미3, 영3 데이터

Test set: 한3, 일3 데이터

만약 Dev와 Test의 데이터가 다르면

Dev를 optimize할 때 Test 데이터로는 목표를 target 설정

기타...

## Batch Normalization

: Layer output normalization

- training은 빠르다

- regularization 한 것 같은 효과

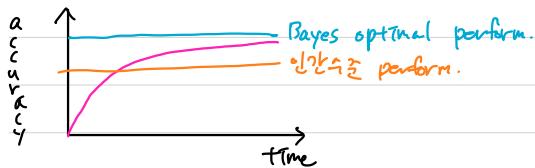
- weights은 더 초기다

→ 미니배치의 평균과 분산을 이용해 정규화, 이전에 신경망에서 흐름

→ 정규화 된 값을 정규화 계수의 일정으로 사용하고

최종 출력 값을 다음 계층의 일정으로 사용하는 것

## Human level Performance



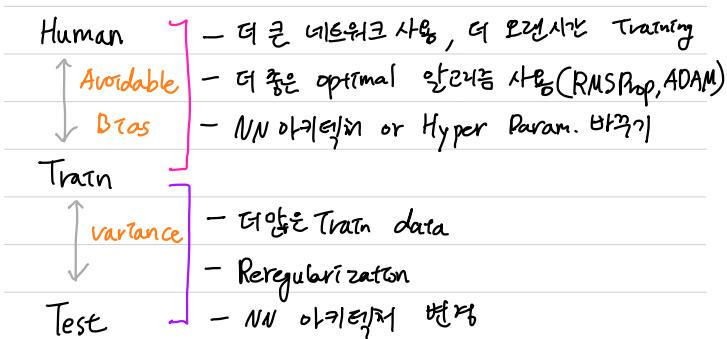
< even human level은 높아서면 accuracy가 줄어드나? >

- ① Bayes et 가장기 때문 (Bayes: Best optimal error)
- ② 인간이 더 이상 모델이 발전되는데 도움을 주지 않아
- ③ bias/variance를 봄 못하는 게 이유거나

## Avoidable Bias & variance

Ex) Cat classification

	A	B	*A: 선명한 전문적 사진, B: Blurry 사진
human Err.	1%	7.5%	Avoidable Bias
Train Err.	8%	8%	Variance
Dev Err.	10%	10%	



## Error Analysis

어떤 데이터가 강아지인데 고양이로 잘못 뿐만 아니라

10%의 Error가 있다면 더 많은 강아지 사진으로 train해야하나?

1. 100개의 m.s. Labeled 된 데이터를 고름
2. 각 error의 유가 원인 counting 함

	Dog	Blurry	Inseafilter	Big cat	...
1	1			1	
2					1
3		1			
...					
100			1		
5	...				

↑ 5% of all errors

데이터가 내가 Dev set 이 잘못 끌어온 데에 따라 맞는지  
잘못인가?

DL Algorithms이 random errors이 많음  
그리고 robust 함 (영향을 잘 받지 않음)

\*따라서 Dev set은 수동적으로 Test set과 수동적으로  
(Dev set이 원래 같은 데이터로 있음)

## Project 하는 순서

- ① 빠르게 시작 : Dev/Test Metrics
- ② get train set
- ③ train
- ④ Bias / variance analysis
- ⑤ Error analysis
- ⑥ 그 다음 단계의 우선순위를 정함

## Train vs. Dev/Test Mismatch

### Available Data

- 200k : Pro cat pics from Internet
- 10k : blurry cat pics from apps : 우리가 신경 쓰는 것

어떻게 train/Dev/Test set으로 나누까?

- option 1 : 모두 섞음
  - Train : 205k, Dev & Train : 각각 2.5k 개
  - 문제 : Dev/Test 이 거의 web pro pics
  - End scenario에서 blurry 사진 쓸

- Solution : Dev/Test는 모두 app의 G1이 E1  
Train은 기존의 Pro pics or app에서  
섞음

→ Train : Web + App : 205k

Dev/Test : App : 각각 2.5k 개

## Bias & Variance with Mismatched Train/Dev

human error : ~0%

Train error : 1% → 이 정도는 괜찮은 generalizing

Dev error : 10% → 5%의 훨씬 더 Dev data가

더 Harder than (?)

⇒ training set과 동일하지만 train이 사용되지 않는  
train-Dev set을 만들고

### Train / Train-Dev / Dev / Test

	A	B	C	D
Train	1%	1%	10%	10%
Train-Dev	9%	15%	11%	11%
Dev	10%	10%	12%	20%
Variance	Train/Dev mismatch	Bias	Bias + Data mismatch	

## Data mismatch을 다루는 법

ex) 차에서 음성으로 주소를 불러 GPS 찾는 데이터

① 차내를 이해하기 위해 속도 error analysis를 수행

(ex - noise, 주소음성)

② Train Data는 Dev Data의 일부로 만들거나

Dev set과 다른 Train Data로 만든다



\* 만약 딥러닝이 차량 소리 car noise가 알고

10시간까지 speech에 해당 데이터를 적용한다면

Car noise의 overfit을 조심해야

## Extended learning

## Transfer learning

문제: 당신이 어떤 medical img를 분류하고 싶은데

Cat을 분류하는 NN만 갖고 있음

- option 1: 만약 당신이 medical img를 소량만 가지고 있으면  
⇒ Cat NN에서 weight은 초기화하고 마지막 layer만 medical img에 맞게 retrain함
- option 2: 만약 당신이 medical img를 많이 갖고 있으면  
⇒ Cat NN에서 weight은 초기화하고 모든 layer를 retrain 함

\* Microsoft custom vision이 이런 방식 이용함

## Multi-task Learning

Multi tasks를 한 번에 training



⇒ softmax 다쓰기 한 번에 맞는 것들이 detection 됨으로써

$$\text{Cost} : J(w, b) = \frac{1}{m} \sum_{j=1}^m L(g_j^{(i)}, y_j)$$

↑ all output option을 다쓰기

4개의 NN으로 train 시킬 수 있지만 다른 condition으로  
multi task learning이 가능하다.① 가지고 있는 다른 tasks의 common learning Data의  
양과 같이 비슷할 때 ex) 1k cars, 1k stop sign.

② 데이터가 충분히 많을 때

## End-to-End Learning

ex) 아이의 손 X-ray 사진으로 아이의 나이 알아내기

&lt;기존 solution&gt;

① 뼈 X-ray에서 ML을 이용해 뼈 나이를 알아냄

② ①에서 얻어낸 뼈 나이를 통해 아이의 나이를 예측함

"End-to-End": X-ray Img → child age (413)

## C4. Convolutional NN

## Convolution Fundamental

## Computer Vision

• Image classification: 사진 보고 뭔지 짜릿이니?

• Object detection: Where is the car?

• Neural style transfer: 피카소처럼 그림그리기

→ 문제: 이미지가 엄청 크다

$$1000 \times 1000 \times 3 (\text{RGB}) = 3M$$

또 여기에 1000 Hidden unit 까지 있다면

$$3M \times 1000 = 3B \text{ 개의 parameter 필요함.}$$

⇒ solution: Convolution 사용하기

— Magnifying filter로 이미지를 스케일하는 것 같음

⊕ 이미지 상 항상 같은 위치에 맞지 않은 문제를 Detection하기 힘들

## Convolution

$$\begin{array}{c}
 \begin{matrix}
 3 & 0 & 1 & 2 & 7 & 9 & 9 \\
 1 & 5 & 8 & 9 & 3 & 1 \\
 2 & 7 & 2 & 5 & 1 & 3 \\
 0 & 1 & 3 & 1 & 9 & 8 \\
 4 & 2 & 1 & 6 & 2 & 6 \\
 2 & 4 & 5 & 2 & 3 & P
 \end{matrix} \\
 \times \quad \begin{matrix}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{matrix} \\
 = \quad \begin{matrix}
 -5 & -4 & 0 & 8 \\
 -16 & -2 & 2 & 3 \\
 0 & -2 & -4 & -1 \\
 -3 & -2 & -3 & -16
 \end{matrix} \\
 \text{Input } 6 \times 6 \text{ Image} \qquad \qquad \qquad \text{output } 4 \times 4 \text{ Image}
 \end{array}$$

$$\begin{matrix}
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0 \\
 10 & 10 & 10 & 0 & 0 & 0
 \end{matrix}$$

$$\begin{matrix}
 1 & 0 & -1 \\
 1 & 0 & -1 \\
 1 & 0 & -1
 \end{matrix}$$

$$\begin{matrix}
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0 \\
 0 & 30 & 30 & 0
 \end{matrix}$$

## Padding

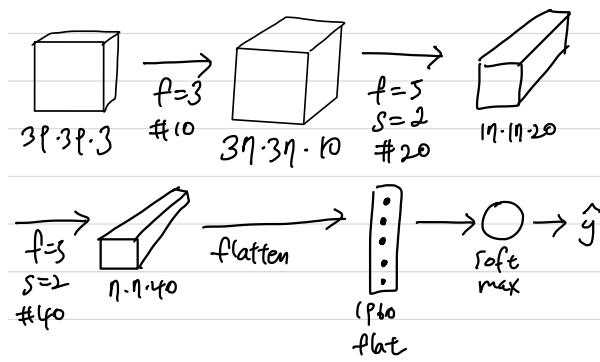
Problem: Image의 shrink 됨:  $6 \times 6 \rightarrow 3 \times 3 \rightarrow 4 \times 4$   
edges는 그냥 사용됨

⇒ solution: Convolving 초기값에 Ø의 border을 더함

$$\begin{matrix}
 0 & 0 & 0 \\
 0 & \text{Input} & 0 \\
 0 & 0 & 0
 \end{matrix}$$

- 2nd padding option : filter pad은 허용되지
- valid  $\Rightarrow P=0$  : no padding
- same  $\Rightarrow P = \frac{f-1}{2}$  : output size = input size  
 $f$ : filter size

### A Deep CNN

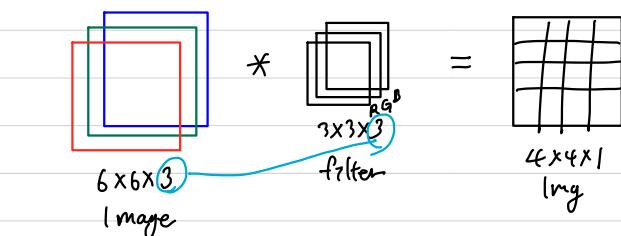


\* figuring out hyperparam 하는게 쉽지 않음

- #filters, stride, padding etc...

\* 좋은 트렌드: size-down / #filters-up

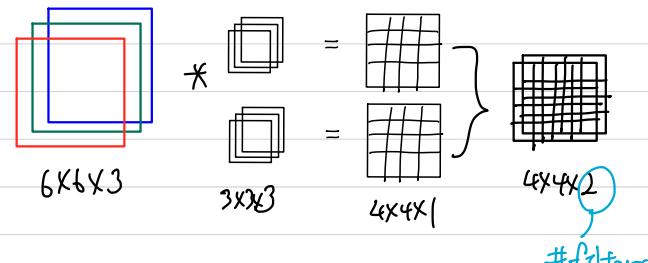
### Convolutions over volumes



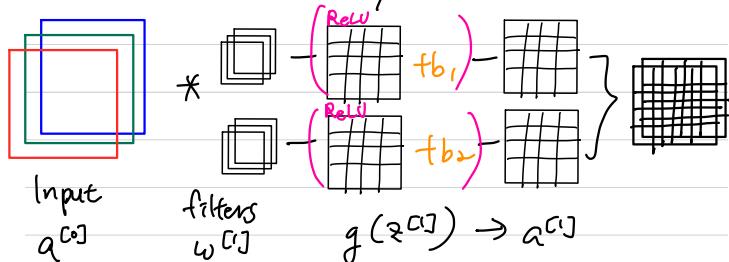
\* each filter가 image channel에서 작동함

### Multiple filters

: 한 번에 여러 feature를 detect할 때



### One conv. net layers



\* 인풋 사이즈가 얼마나 크든 문제가 되지 않음

learnable param  $w & b$ 는 각 filter에 대해서 그대로呗  
영향을 받음

▷ param ↑

$$w = 3 \times 3 \times 3 \times 2 = 54 \quad \text{36 param}$$

$$b=2$$

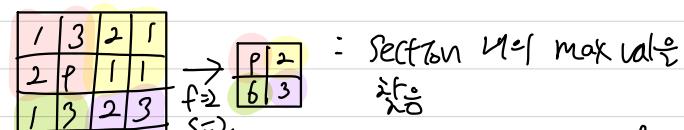
### Typical conv. net layers

- Convolution (CONV)

- Pooling (POOL)

- fully connected (FC)

### Pooling (Max)



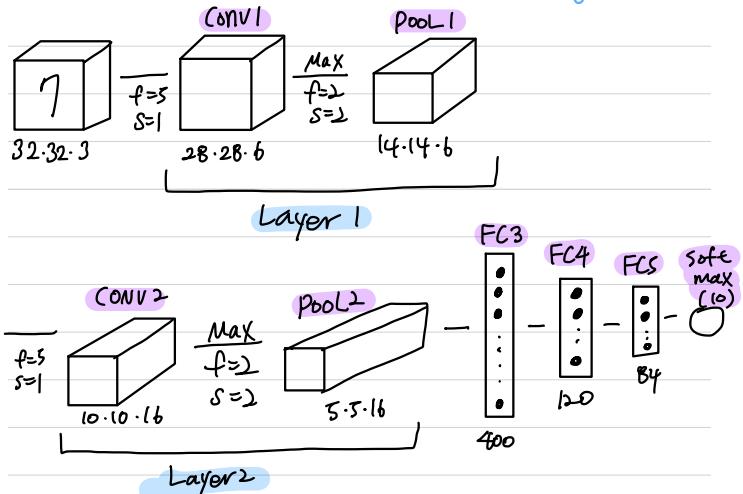
\* Pooling:  $\frac{n-f}{s}$

- Computation을 줄여주기 때문에 좋음

- 여러 detected feature를 더 robust하게 함

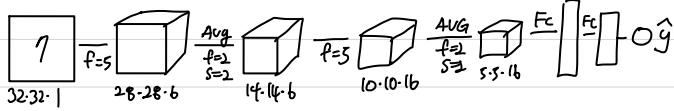
### Conv net Example

\* LeNet-5를 기반으로 함, 손으로 쓴 숫자를 detecting하는 문제



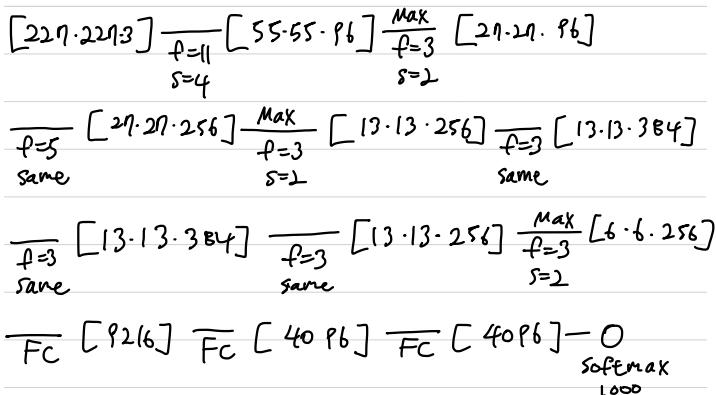
## classic Conv. Nets

## LeNet 5

: Document classification  $\approx$  60k Parameters

- Trends: 각 layer마다 줄어들고 계층의 수는 많아짐
- 공통적인 Pattern: 몇몇 개의 Convnet Pool Layer가 있고 몇몇 개의 FC layer가 따라옴
- Old stuff:
  - max pooling and avg pooling을 씀
  - Padding을 자주 쓰지 않음
  - ReLU와 sigmoid 및 Tanh를 씀

## Alex Net

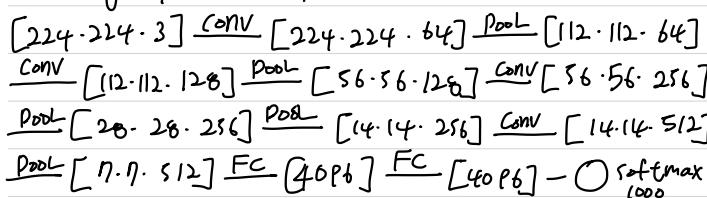
: Image classification  $\approx$  60M Parameters

- LeNet이랑 비슷하지만 더 크다

- ReLU를 사용한다.

## VGG-16

- 모든 conv layer가 같은 param을 가지고 있음

:  $f=3, s=1, p=\text{same}$ - Pooling layer가  $f=2, s=2$  $\approx 138\text{M}$  Params

- 매우 깊다

- 아키텍처가 수준다

- # filter 가 2단계가 되어간다 - 64, 128, 256, 512

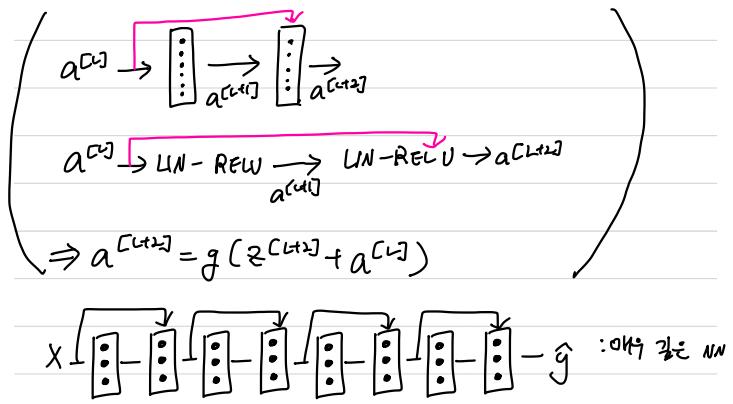
## Special Networks

## ResNets

• 문제: 같은 NN은 종종 Gradient vanishing or gradient exploding의 문제를 겪는다.

- Solution : Residual Nets (ResNets)

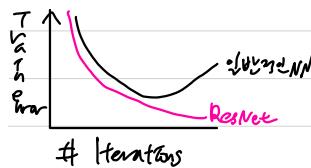
\* Residual Block



$$a^{[L+2]} = g(z^{[L+2]} + a^{[L+1]}) = w^{[L+2]} a^{[L+1]} + b^{[L+2]} + a^{[L+1]}$$

• worst case : weight bias가 0이면 vanished gradient

$$a^{[L+2]} = g(a^{[L+1]}) = a^{[L+1]}$$



## Networks in Networks (IxI Convolution)

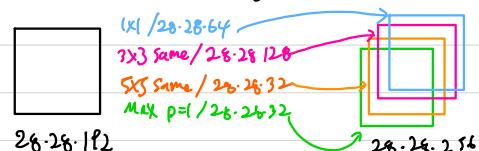
$$\begin{bmatrix} 6 & 5 & 3 & 2 \\ 4 & 1 & 9 & 5 \\ 5 & 8 & 2 & 4 \\ 0 & 3 & 6 & 1 \end{bmatrix} * \boxed{2} = \begin{bmatrix} 12 & 10 & 6 & 4 \\ 8 & 2 & 18 & 10 \\ 10 & 16 & 9 & 8 \\ 0 & 6 & 12 & 2 \end{bmatrix}$$

: 같은 filter로 비슷한 2개의 특징이 있음

: 연산량 감소, 계산 속도 증가, 비용 절약

## Inception Networks

IxI, 3x3, 5x5 or pooling layer을 끼는 대신 모듈 적용



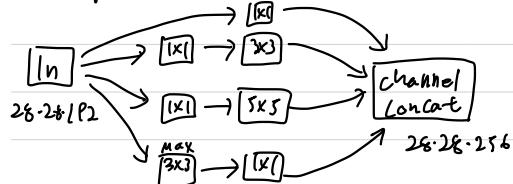
- 문제: 계산이 너무 많이 필요하다

- Solution : 모든 filter를 적용하기 전에  $1 \times 1$  conv를 사용해 계산량 줄임

계산량 줄임

## MobileNet v2 Full Architecture

## \* Inception module

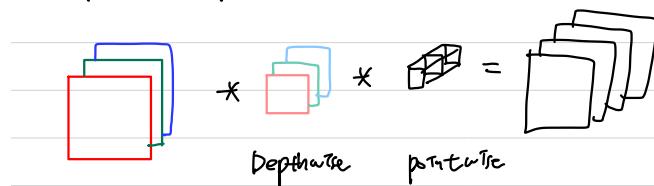


- Inception 네트워크를 만들기 위해선 양자 Inception module 을 활용하는가.

## Mobile Net

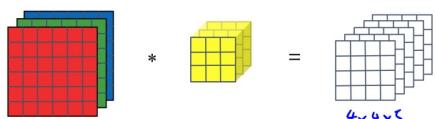
- Mobile 기기에서 소모전력과 계산비용을 줄이기 위해서

→ Depthwise Separable Convolution

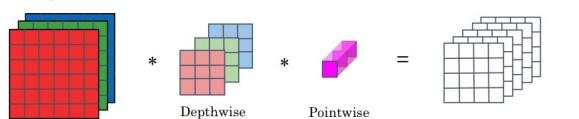


## Depthwise Separable Convolution

Normal Convolution



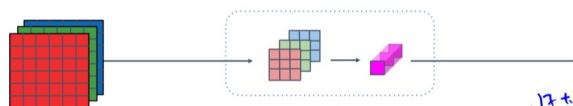
Depthwise Separable Convolution



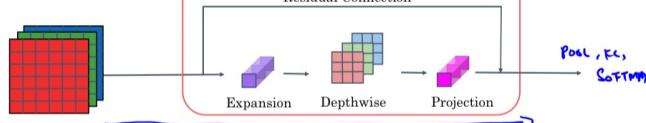
- param of normal convolution 많다.

## MobileNet 2

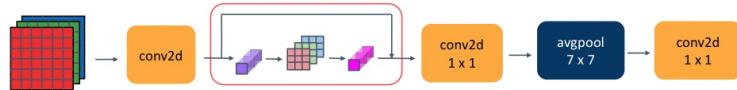
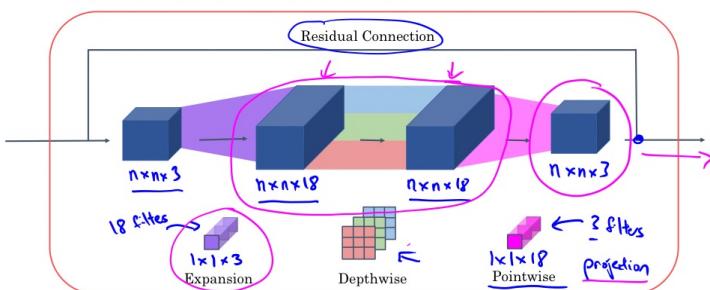
MobileNet v1



MobileNet v2



## MobileNet v2 Bottleneck



## Efficient Net

Compound Scaling은 사용해 모델의 크기(G), 층수(w)  
입력 이미지의 크기(r)를 조절함

\* Compound scaling: w, d, r은 하나 이상 조절하는 경우  
det r은 함께 조절하여 차례의 흐름을 맞춘다

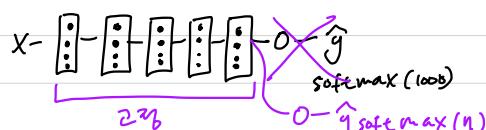
## Practical Advice

## Use open source Implementation

여러 papers는 코드를 설명하기 어려우므로 다른 open source의  
것을 사용하는게 좋다

## Transfer learning

- 데이터는 원본 학습에 충분한 데이터가 필요하니  
→ 다른 사람의 이미지 train된 net & weight를 다운받음



- param은 2개 사용

softmax layer만 내가 학습하는 카테고리에 맞게 수정하고  
train 가능

## Data Augmentation

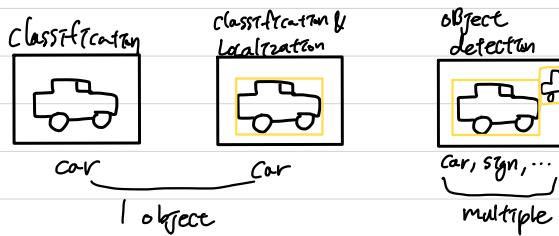
- 데이터 증강은 반드시 필요

- mirroring, random crop, color shifting 등

## Tip for doing well on Benchmarks / competitions

- Ensembling: Avg output from multi NN
- Multi-Crop at test time: Avg outputs from multiple crop at the image

## Detection Algorithm



Yolo : You Only Look Once

- ① 이미지를  $X(g)$  grid cell로 쪼개
- ② 각 cell에 대해서 car + boundary 박스가 포함되는 있는지 찾기

이렇게 종종 찾을 수 있다?



: red square은 얼마나 찾나?

$$IOU = \frac{\text{size of } \cap}{\text{size of } \cup}$$

: Union of intersection (IOU)

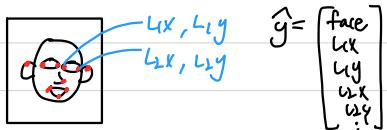
- 보통적으로  $IOU > 0.5$  면 correct라고 봄

만약 여러 square가 같은 차를 detect 했으면?

→ Non-Max Suppression

: 만약 2개 이상의 bounding box가 높은 IOU를 갖고 있다면 그 중 가장 높은 IOU를 가진 것은 뒤로 차운다. 예제는 제거함

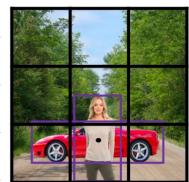
## Landmark Detection



- 얼굴에서 랜드마크(눈꺼풀, 눈썹, 코 등)를 detect하기 위해 랜드마크의 x, y 좌표를 label 사용함

## Anchor Boxes

Anchor Boxes는 같은 square에 여러 object를 detect하게 함



## Region proposals: R-CNN

- 전처한 Region은 CNN의 feature(영역별)으로 활용하여 object detection을 수행하는 신경망
- 각각은自身的 Bounding Box를 주게 됨
- 이미지 하나의 영역의 영역별로 함께 잘려 merge해 wrapping한 후, CNN의 input으로 통과해

Feature vector를 추출하고, 적용된 label들을 이용해 SVM을 통해 Bounding Box Regression을 수행하여 영역 위치까지 예측을 하게 되는 모델

## Semantic Segmentation with U-Net

- 일반적으로 semantic segmentation 모델은 출연을 확장하는 형태

↓ 출연: Down sampling: Encoder

↑ 확장: Up sampling: Decoder

- U-Net: U자 모양으로 설계, 유니온이 segmentation이 Good

→ 주도는 확장하고 contexted localization의 trade off에

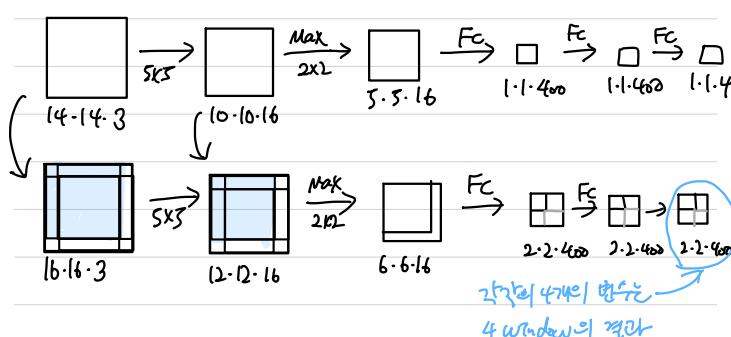
적용

• sliding window가 아닌 patch 방식 사용

• Contracting Path에서 이미지의 context를 확장,

Expanding Path에서 feature map을 upsampling 한 후

context는 확장해 localization의 정확도를 높임



- 학습이 지나갈수록 동일한 이미지가 가능해
- 기존에는 전처리단계에서 자동차를 안내까지 반복적으로
- Convolution은 이용해 모든 예측값의 계산이 가능해
- 합성곱은 신경망에 한 번만 통과시켜도 자동차의 위치를 찾을 수 있음

## Face Recognition

### Face Verification



이 사람과 같은가?

### Face Recognition



이 사람과 같은가?

## Content Cost function

- pretrained의 convnet 사용함 (ex VGG)

- 중간에 있는 hidden layer를 고른다.

-  $a^{(c)}$  et  $a^{(G)}$  를 activation으로 사용함

- 만약  $a^{(c)}$  et  $a^{(G)}$  가 비슷하면 그들은 content임

\* 둘이 같은 hidden unit을 trigger하기 때문

• 어떻게 둘이 비슷하게 암수 있을까?

$$J_{content}(C, G) = \frac{1}{2} \|a^{(c)} - a^{(G)}\|^2$$

## Style matrix

각각의 pos(x,y) & channel pair(k,k')에 대해서 activation가 얼마나 잘 맞는지에 대한 거의 matrix를 만드는 것

$$G_{kk'} = \sum_{i=1}^{n_h} \sum_{j=1}^{n_w} a_{ijk} \cdot a_{ijk'}^*$$

## Style cost function

$$J(S, G) = \|G^{(S)} - G^{(G)}\|_F^2$$

Frobenius norm

## C5. Sequence models

### Recurrent neural networks

- speech recognition, music generation, sentiment classification, DNA sequence analysis, machine translation, video activity recognition, name entity recognition

### Name entity recognition

X: Harry Potter and Hermione Granger invented a new spell

$x^{(1)} \quad x^{(2)} \quad x^{(3)} \quad x^{(4)} \quad x^{(5)} \quad x^{(6)} \quad x^{(7)}$

$T_x$  (단어수) = 9

$y = 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0$

$y^{(1)} \quad y^{(2)} \quad y^{(3)} \quad y^{(4)} \quad y^{(5)} \quad y^{(6)} \quad y^{(7)}$

### 어떻게 단어를 표현하나?

단어집을 만든다. 각 단어는 one-hot vector로 표시됨

$$\begin{bmatrix} a \\ aaron \\ and \\ Harry \\ \vdots \\ zulu \end{bmatrix} \stackrel{1}{\textcolor{pink}{a}} \stackrel{2}{\textcolor{pink}{aaron}} \stackrel{3}{\textcolor{pink}{and}} \stackrel{4}{\textcolor{pink}{Harry}} \stackrel{5}{\textcolor{pink}{zulu}} \qquad \qquad \qquad \text{Harry} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \rightarrow \text{tokens}$$

→ Standard network에서 사용할 수 있음 하지만

① example끼리 input과 output의 길이가 다름

② 다른 위치에 있는 단어끼리 learned feature는

공유하지 못함

→ 각 사람에 대한 softmax distribution에 접근하고 각 사람을  
인식하는 사람을 recognition하고 한다면  
각 사람에 대한 softmax - CNN으로 브레이크 할 수 있음  
- 흐릿한 sample이 있어서  
- 새롭고 사람의 추가되었을 때 Network를 다시  
train할 수 있기 때문

⇒ solution: similarity function을 써보자  
 $d(\text{img}_1, \text{img}_2) = \text{2}^{-\text{loss}}$   
그리고 이를 더해보자

## Siamese net work: deepface

$\left[ \begin{array}{l} \text{img } x' - \text{CNN} - f(x') \\ \text{img } x^2 - \text{CNN} - f(x^2) \end{array} \right] \rightarrow d(x', x^2) = \|f(x') - f(x^2)\|^2$   
- 만약  $x'$ 와  $x^2$ 가 같은 사람  $\rightarrow d(x', x^2)$ : 짧음  
- 만약  $x'$ 와  $x^2$ 가 다른 사람  $\rightarrow d(x', x^2)$ : 길음

### Triplet loss : Face Net

: 양쪽 다른 사람과 같은 사람으로 similarity function을  
제공하여 이를 triplet을 만들고 similarity function을  
제공하는 것을 학습

### Neural Style Transfer

#### 〈어떻게 다른 style의 이미지를 만들어 볼 수 있나?〉

→ idea

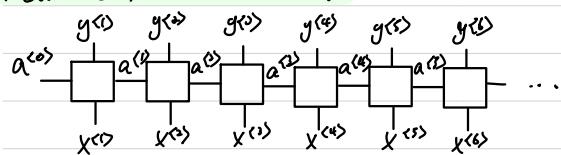
① random img를 만듬

② cost function은 최적화함

$$J(G) = \alpha J_{content}(C, G) + \beta J_{style}(S, G)$$

③ update each pixel

## Recurrent Neural Net (RNN)



이전의 결과가 아직 (input으로 들어온가지) \$\Rightarrow\$ context를 갖음

$$a^{(t)} = g_1(W_{aa}a^{(t)} + W_{ax}x^{(t)} + b_a) \text{ Tanh/ReLU}$$

$$\hat{y}^{(t)} = g_2(W_{ya}a^{(t)} + b_y) \text{ Sigmoid}$$

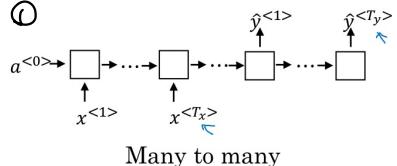
같은 weight가 모든 step에서 사용됨

$$\cdot L^{(t)}(\hat{y}^{(t)}, y^{(t)}) = -y^{(t)} \log \hat{y}^{(t)} - (1-y^{(t)}) \log (1-\hat{y}^{(t)})$$

$$f(\hat{y}, y) = \sum_{t=1}^T f^{(t)}(\hat{y}^{(t)}, y^{(t)})$$

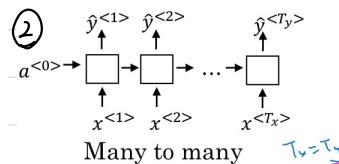
## Different Types of RNN

①



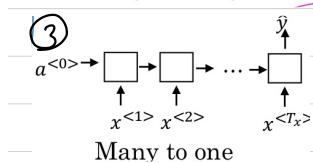
Many to many

②

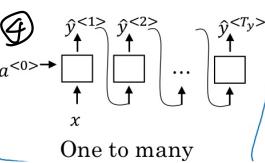


Many to many

③



Many to one



One to many

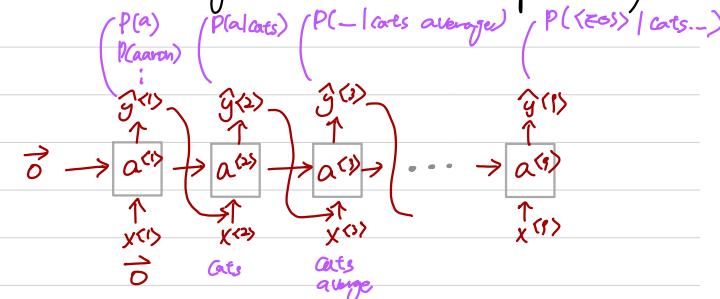
## More on RNNs

## Language Modeling

언제 어떤 사람이 • Apple and pair / Apple and pear  
이라도 말했을 때 어떻게 알 수 있나?

\$\Rightarrow\$ Language Model은 확률을 계산하는 것임

ex. Cats average 15 hours of sleep a day (EOS)



## Sampling Sentences

① train on all Harry Potter Books

② Randomly select a word

③ Pass this into the next timestamp  
& sample a new word

④ Repeat until X words or you reached (EOS)

\$\rightarrow\$ J.-k Rowling의 작품 스타일을 가질수 있음

## Vanishing Gradients

The Cat, who already ate apples and orange --, was  
긴 히스토리가 있는 편

\$\rightarrow\$ Standard NN일 때 vanishing gradient 문제 있음  
\$\rightarrow\$ Deep NN 사용

\* exploding gradient는 gradient clipping으로 해결 가능

\* gradient clipping

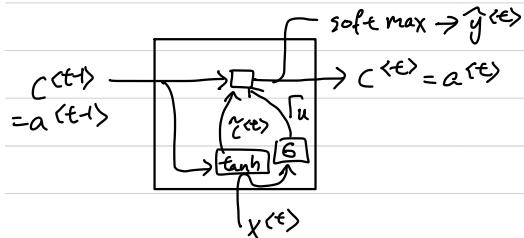
: gradient가 일정 threshold를 넘어가면 clipping을  
해주는 것, clipping은 gradient의 L2 norm으로  
나눠주는 것

## Gated Recurrent Unit : GRU

다른가지 디플레이션이나 부수증이 있는가 recall 하는걸 드러냄

: GRU는 "기억"이라는 특징

GRU Gate \$f\_u\$가 \$C\_t\$에 영향을 미친다. 말거나 기억



$$\tilde{C}^{(t)} = \tanh(W_C[C^{(t-1)}, x^{(t)}] + b_C)$$

$$f_u = \sigma(W_u[C^{(t-1)}, x^{(t)}] + b_u)$$

$$C^{(t)} = f_u * \tilde{C}^{(t)} + (1-f_u) * C^{(t-1)}$$

## Long Shortterm Memory : LSTM

: LSTM은 GRU의 Variation — 더 많거나 (Forget gate)

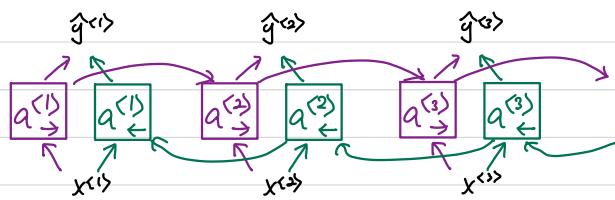
$$f_t = \sigma(W_f[a^{(t-1)}, x^{(t)}] + b_f)$$

## Bi-Directional RNNs (BRNN)

[Teddy bears are on sale]

[Teddy Roosevelt was a great president]

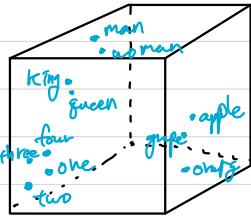
- Problem: 풀마지 보고 알수있을 'Teddy'가 장난감인지 이름인지  
모름



• exg: full sentence 시작하기 전에 알수있음

→ live speech에는 적용하지 못함

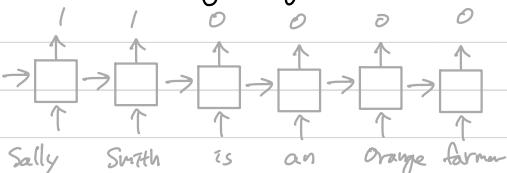
→ The feature are learned



→ t-SNE: visual representation of 200D word Embeddings

## Using Word Embeddings

## ex1) Name / Entity Recognition



word Embedding을 이용해 orange farmer 을 사용해  
Sally Smith은 이름인 것은 알 수 있음

## ex2) Man is to woman , King is to ?

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	0.93	0.95
Age	0.03	0.02	0.7	0.69	0.70	0.69
Food	0.09	0.01	0.02	0.02	0.01	0.95

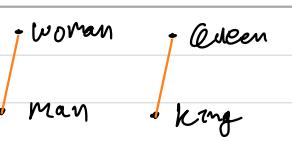
↑                   ↑                   ↑                   ↑

man               woman           king               orange

• man - woman

• king - ?

$$\begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ 0 \end{bmatrix}$$



• man → woman

• king → ?

Find(w):

Arg.MAX Sim(Ear, Eary - Emant Ewoman)

$$* \text{Sim}(U, V) = \frac{U^T V}{\|U\|_2 \|V\|_2}$$

↑ U : cosine  
↓ V Similarity

## Word Embeddings

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	0.93	0.95
Age	0.03	0.02	0.7	0.69	0.70	0.69
Food	0.09	0.01	0.02	0.02	0.01	0.95
size						
cost						
alt						
verb						

↑                   ↑                   ↑                   ↑

man               woman           king               orange

I want a glass of orange juice.  
I want a glass of apple juice.

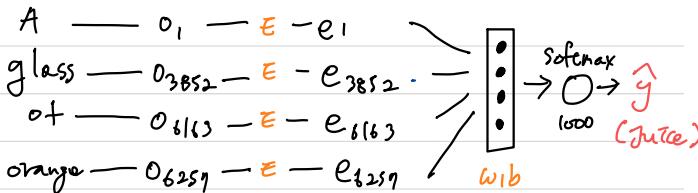
Andrew Ng

## Learning word Embeddings

How do we learn the embedding matrix  $E$ ?

① Idea 1: Neural Language Model 사용하는 것

I want a glass of orange  $\hat{g}$



- Context: Last 4 words를 풀어보기
  - 4 words on left & right
  - Last 1 word
  - Near by 1 word

## ② Idea 2: Skip grams : Word2Vec

I want a glass of orange juice to go along with my cereal

→ random context/target pairs 사용

context | target

orange	juice
orange	glass
orange	my

$O_c \rightarrow E \rightarrow e_c \rightarrow \text{softmax} \rightarrow \hat{y}(\text{Or})$

: Simple NN이 Or을 예측하는 동안  
같은 goal은 E를 학습하는 것

• 계산비용이 매우 많이 들지만

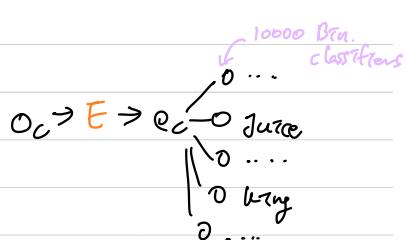
Hierarchical softmax classifier 사용해 optimize 가능

## ③ Idea 3: Negative sample

1. Positive Sample로 Context/Target pairs 사용

2. Negative examples Context는 random하게 뽑기

context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0



• If 512bit Context, word는 train  
→ training efficient

## Glove word Vectors

$X_{ij} = \text{word } i(\text{target } c) \text{ in context } j(\text{context } c)$  이다  
내부에 있는 것

(어디서 예상되는 것인가)

$$\text{Minimize } I_{i=1}^{10k} I_{j=1}^{10k} f(x_{ij}) (O_i^T e_j + b_i + b_j - \log x_{ij})^2$$

## Sentiment classification

$x \longrightarrow y$

The dessert is excellent.



Service was quite slow.



Good for a quick meal, but nothing special.



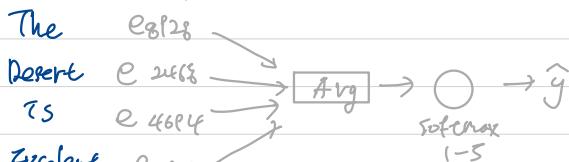
Completely lacking in good taste, good service, and good ambience.



문제: 데이터셋이 알고 암금

→ pre-trained된 Embedding matrix  $E$ 를 사용하는 암금

## ① Idea ①: simple classification



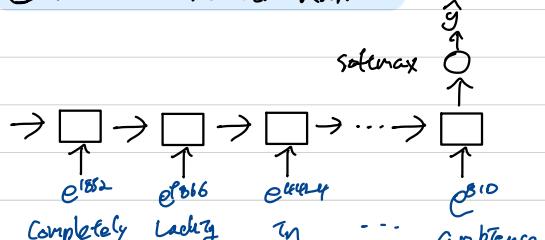
→ 디자인 문제에 대해서는 working well

하지만 문제의 문제는 context를 고려하지 못함

ex) Completely lacking in good taste  
good service, good ambience

→ 안 좋은 내용이지만 좋게 평가 되었대요

## ② Idea 2: Use an RNN



⇒ 순서와 context는 고려함으로 암금

## Eliminating Bias in word embeddings

man is to computer programmers

woman is to house maker

: 중증 예언이 성별, 인종, 나이 등에 따른 Bias를 가짐으로  
우리는 우리의 예언이 이러한 Bias를 가지는 것을  
알려야 한다.

## Addressing Bias

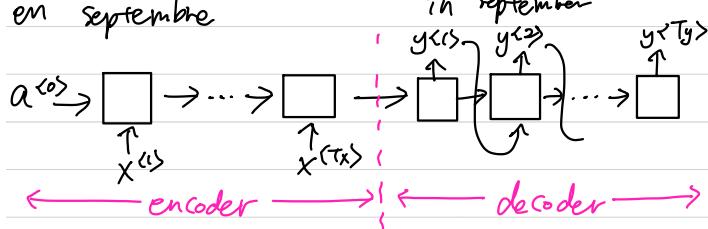
	• doctor	① Bias direction을 헤아리자
• baby sister		② Neutralize : girl, boy, her, she 등 짐작하기 하기가 어렵거나 예상
girl	• boy	③ Equalize pairs : girl / boy 같은 단어는 gender에 의해 diff로 나누어 올라온다
sister	• he	

→ 그걸 어떻게 하면 Neutralize할 수 있나?  
→ class definition의 일정한 패턴에 맞아내면서  
class fier을 Training함

## Sequence to Sequence

### Basic model

Jane visite l'Afrique → Jane is visiting Africa  
en septembre



### (Image captioning)



→ This is a cat on a chair

CNN → RNN

## How do you pick the most likely sentence?

English French

$$\rightarrow \text{Arg Max } P(y^{(1)}, \dots, y^{(T_y)} | x)$$

Idea: Use greedy search

1. 가장 높은 확률을 가진 단어를 고름
2. <eos>에 도달한 때까지 반복

문제: 시계 등이 틀려질 수 있음

→ solution: 현재 문장의 확률을 optimize

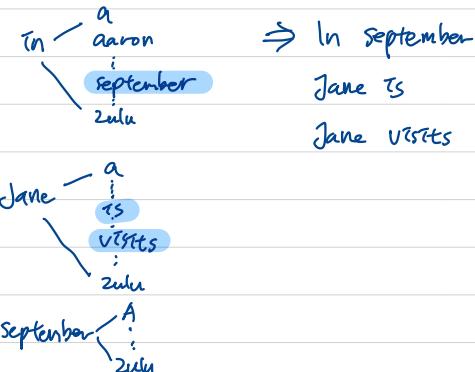
## Beam search

① Pick the first word

- pick the "B"(예) 3) : Best alternatives  
(in, Jane, september)

② 각각의 B 단어가 다음 단어를 고름

그리고 B pair가 되는 pairs를 평가함



### ③ 끝날 때까지 반복

$$\arg \max \prod_{t=1}^{T_y} P(y^{(t)} | x, y^{(1)}, \dots, y^{(t-1)})$$

문제: ① 확률은 끊임없이 줄어들 수가 있기 때문  
② mult.를 optimize하면서 확률은 증가하는 경향을 선호하게됨  
(즉 확률은 냉혹하기 때문)

→ optimize with this

$$\frac{1}{T_y} \sum_{t=1}^{T_y} \log P(y^{(t)} | x, y^{(1)}, \dots, y^{(t-1)})$$

## 어떻게 B를 고른다?

[ B가 좋은: Better Result, slower

    B가 나쁜: worse Result, faster

## Error Analysis in Beam Search

- human result:  $y^*$
- Algo. result:  $\hat{y}$
- $\rightarrow$  RNN이  $P(y^*|x)$ ,  $P(\hat{y}|x)$  를 출력
- if:  $P(y^*) > P(\hat{y})$ : 다음은 B를 선택해야함
- else: RNN이 정답

## Bleu score

Idea: 현실 언어에서 'the'가 얼마나 나온는지 체크  
: 마지막 단어의 결과가 사람이 직접 번역한 결과와  
얼마나 비슷한지 비교하여 번역에 대한 평점을  
제작함

- 흐름성을 n-gram이 기본함

- ① 단어 개수 카운트로 흐름하기
- ② 중복을 제거하여 보류하기
- ③ 보류된 유니그램 정밀도 구현하기
- ④ 순서를 고려하기 위해 n-gram으로 확장하기

$$\text{BLEU} = \exp\left(\frac{1}{n} \sum w_n \log p_n\right)$$

## Attention Model

RNN이 기반한 seq2seq 모델

<문제> ① 고정된 크기의 벡터에 모든 정보 압축하려는데  
정보丢失이 발생함

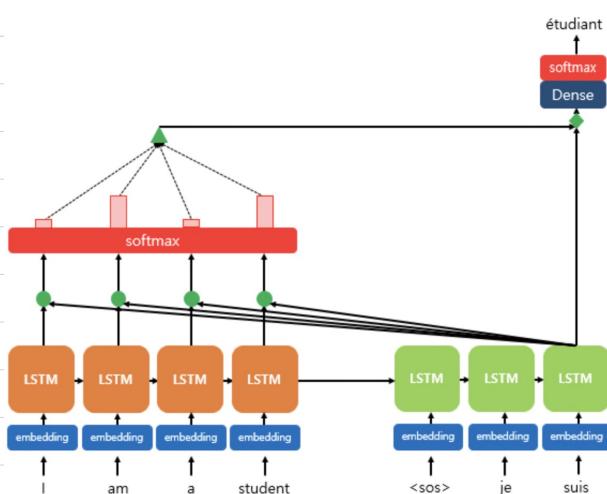
② RNN이 고정된 문제인 가로기 소설 문제

## ⇒ Solution

다중언어에서 훈련데이터를 예측하는 대사형마다 인코더에서

전체문장을 한번 더 담고

만, 전체문장을 다동일한 바탕으로 처리하는 것이 아니라  
대화여정에서 예측해야할 단어로 연관이 있는 경우 다른  
부분을 좀더 강조해서 (attention) 봄



## Speech Recognition



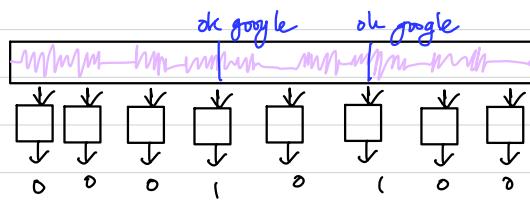
: The quick Brown Fox ...

- 기존 RNN 문제: 100Hz의 10초 clip은 1000 input이  
들어온다. But 20% input만  
나오면 됨

⇒ solution : CTC cost 사용 (Connection temporal  
classification)

ttt-h-goo... --> --> --> -->

## Trigger word detection



## Sequence to Sequence models

### Transformers

- 기존의 seq2seq 구조는 인코더-디코더를 따로 둔다  
는데 이를 통해 Attention으로는 구현할 문제  
RNN을 사용하지 않고 인코더-디코더 구조를 설계  
But RNN이 더 푸