

핵심: CNN 기본 용어 + 2012-2015 근본 논문 개요 초간단 소개

Fast-Forward to today: ConvNets are everywhere.

① classification

② Retrieval

③ Detection

④ Segmentation



⑤ Image Captioning

⑥ Style transfer

(이제는 옛날 논문이지만 그래도 오류인
안되는 상황제)

(논문: CS231n 2010)

Fully-Connected Layer
vs Convolution Layer

기능: filter

Convolution Layer 와 기존의 FC 레이어의 주된 차이점이 있다면,
C.L.은 기존의 가로를 서로 사용하는 것.

(반복 32x32x3 Input 이미지 FC는 3072x1 을 병행하여 처리하는
C.L.은 10x12x3이다.)



Convolution Layer에서는 이미지의 필터 사용 (7x1)

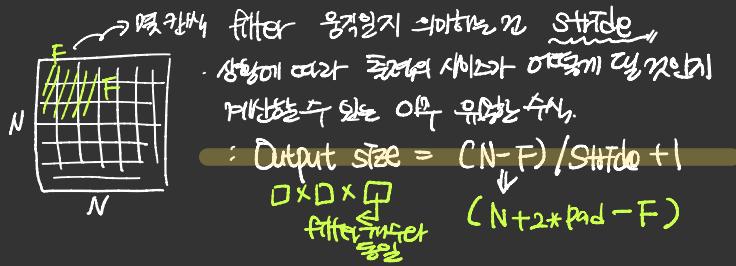
→ 이미지를 퍼트리마다 다른 특성을 추출하고 여러 다른

필터가 아니라 동일한 수만큼 feature map 모아놓고 학습에 사용

Input → CONV → ReLU → CONV → ReLU → POOL → ... → FC

activation maps
사이즈를 줄이는 연산

Spatial Dimensions



- 크기의 차이를 최대화로 만들어 주기 위함.
- 아마지의 가장자리를 0으로 처리
- 사용 이유: 데이터를 재구조화해 영역의 차이를 극복해/유연화.
(필터 사용하면서 흡수되는 차이를 예상하고)

용도정리

- (6x6 filter가 있다고 가정)
- 한 뉴런의 "Receptive Field"가 6x6이다.
(한 뉴런이 한 번에 수용할 수 있는 뉴런)

Pooling Layer

- makes the representations smaller. (전체 입력 수 ↓)
- downsampling
- Depth에는 영향을 주지 않으며, 일반적으로 Max pooling 사용
- \oplus pooling 시엔 stride를 조절하지 않기 때문에 일반적.

Batch Normalization

+ whitener

- 표준, 분산을 계산하는 과정
- Pixel 간 차이를 대비하기
- 아마, 신경망 안에 포함되어
학습 시 광고와 분산을 계산하는
과정

- 학습 과정에서 각 뉴런 단위로
데이터가 달라져 분산을
계산하는데 꽤 비효율적
- 분산과 분산을 이용해
규모화하는 것

Activation Functions

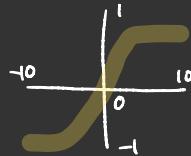
non-linear simi 바꿔줌

① Sigmoid : $\sigma(x) = 1 / (1 + e^{-x})$



- 문제 1) 이걸 flat한 구간에서 gradient들이 다 죽음.
- 문제 2) zero-centered 가 $x \rightarrow \infty$ 비례로
- (입력 x 가 양수일 때 모두 가지고 있는데)
전부 같은 방향으로 움직이는 일은 별다른 x
- 문제 3) $\exp(x)$ 때문에 계산비용이 크다.

② $\tanh(x)$



zero centered :)

문제 1) still kills gradients when saturated
: C

③ ReLU : $f(x) = \max(0, x)$

Does not saturate (In + region) :)

Computationally efficient :)

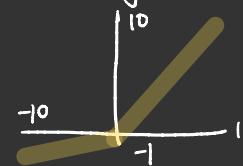
문제 1) Not zero-centered output.

용의 경유면 saturation

→ Dead ReLU 발생할 수도 있음.



④ Leaky ReLU : $f(x) = \max(0.01x, x)$



Dead ReLU까지 해결!

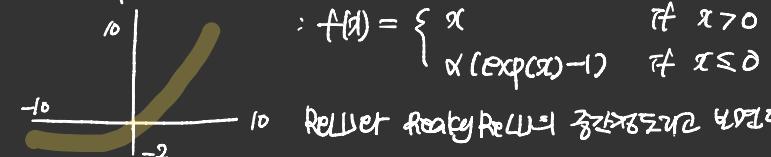
⑤

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

→ backprop - 음수 계수 사용 가능
인정됨

⑥ Exponential Linear Units (ELU)



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

ReLU와 leaky ReLU의 중간地带하고 보면됨

⑦ Maxout "Neuron"

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

(generalizes ReLU and Leaky ReLU.
(Linear regime! Does not saturate! Does not die!)
문제 1) parameter共享 가능

Optimizer

최적화 알고리즘 SGD의 문제점



어려운 미세한 loss 최적화

gradient를 빼고, 흐름으로
움직일 수 있을까?

① 손동작의 기준은 loss가 넓은데 진짜로 같아.

다시 말해, Loss는 원점 방향의 기울기 변화에 훨씬 더 민감

→ 경계선이 멀면 더 넓어지니 nasty(거친)



거친!

② local minima & saddle point

local stop!

이 경계선 기울기 ↓
→ 가로로 퍼터버트팅
saddle point에서 stop!
(별로 멈춰있나?)

③ loss를 계산할 때 마다 그걸 다 계산할 수는 없기 때문에
최소화된 양의 노드 진행 → 불정학 !!



①-④의 문제점을 해결해보자

한 번 더 방법

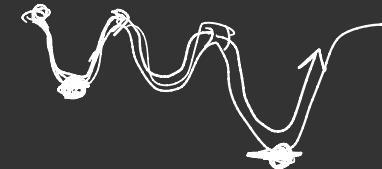
→ SGD + Momentum

CoverShoot
한 번에 뛰어
움직이는 경향

gradient를 계산할 때, velocity를 유지하는 것!

(속도가 높으니까 그 이후 부분을 잘 극복하고 계속 나아갈 것.)

→ Momentum을 추가해서 Velocity가 성가롭고
적은 noise가 표현된다.



(설명 생략. 너무 수식이라..)

Adagrad (~~Neg ×~~) → RMSprop. → Adam. ~~※※※~~
(Seems like
RMSprop + momentum)

마침까지 차이점을 이용한
최적화 방법 알아보기
최적화 방법 알아보기
위로 미니마이즈! ⇒ second
-order solver

아직, 학습률 높지 못한 데이터에 대한

성능을 높이려면?? ① Model ensemble

① - ③

② 양방향이 아닌 단일 모델의 성능을 향상시키기 위해서는? Regularization
(L1, L2, Elastic) (L1+L2)



①

NN에서 가장 많이 사용하는 regularization은 dropout!!

: forward pass에서 원의 노드를 랜덤으로 0으로 만들고 활성화(activation)를 끄는 것

* dropout은 fc layer에서 흔히 사용.

전체 conv net에서 전체 feature map에서 dropout 10%

: dropout 계층 A,

$$y = f_w(x, z)$$

output Input \ random mask

: test할 때 원래는 풀지 X
→ at test time, multiply
by dropout probability

②

: BN은 dropout에 비슷한 역할

→ Train 시에는 stochasticity (noise) 가 추가되지만

Test 시에는 그걸 제거하여 더 청楚하게

→ BN 사용 시, dropout 사용 X

: data augmentation 도 Regularization의 일종

{ (color jittering 포함)

translation,

rotation, stretching, ...

* 참고할 만한 노트
내용입니다 *



CNN Architecture

Case study

AlexNet → VGG → GoogleNet → ResNet

AlexNet (2012)

- 3 layer of conv net, ConvNet 영역의 확장을 일으킴
- conv - pool - normalization 단계가 두 번 반복
- Input : $227 \times 227 \times 3$
 - First layer (conv1) : 96 11x11 filters, stride 4.
→ output : $(227-11)/4+1 = 55 \Rightarrow 55 \times 55 \times 96$ ✓
parameters : $(11 \times 11 \times 3) \times 96 = 25k$
 - Second layer (pool1) : 3x3 filters, stride 2.
→ output : $(55-3)/2+1 = 27 \Rightarrow 27 \times 27 \times 96$
parameters : \times (pooling layer에는 parameter x)

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

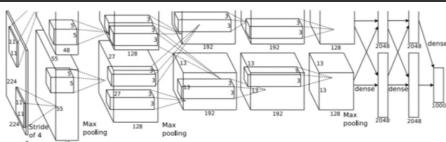
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)



ReLU 사용, data augmentation, dropout, batch size = 128,
SGD momentum, learning rate = 1e-2, weight decay,
Ensemble 3 개 티타

ZFNet (2013)

AlexNet의 하이퍼파라미터를 개선한 모델

↓ 이제 정밀한 경계지정 시작.

VGGNet (2014)

- 층별 필터링하고, 더 작은 필터 사용
- 왜 더 작은 필터 사용?

→ 필터 크기가 작으면 필터링의 수가 더 적어서 가중 계이터를 갖기
쉽다.

- Stack of three 3×3 conv(stride=1) layers has same effective receptive field as one 7×7 conv layer.
- 3×3 filter(stride=1) 시그널 receptive field는?

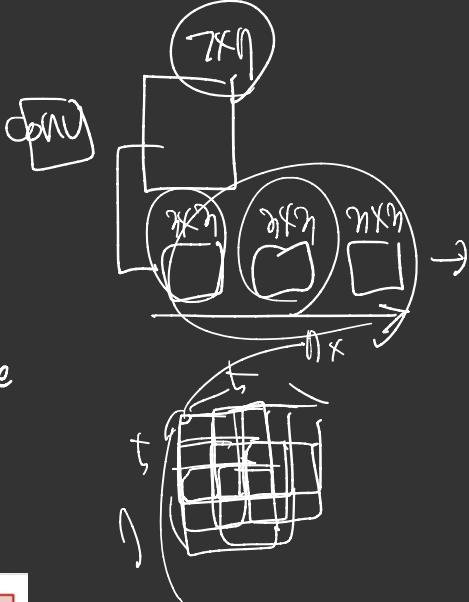


```

INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

```

VGG16



GoogleNet (2014)

- 2개의 layers
- Inception module 사용
- FC layer 사용 (파악률은 좋아지기 위해서.. AlexNet이 60%인데
이제 50% 정도)

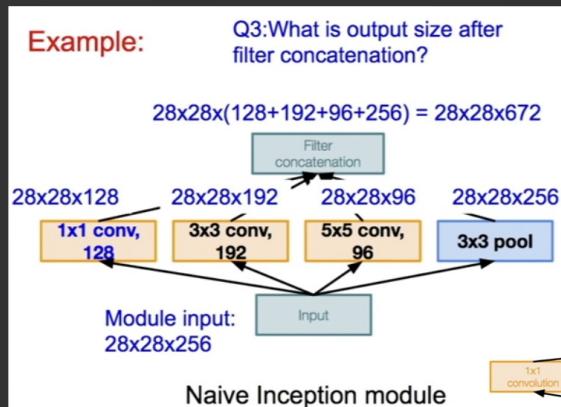
NATIVE

Inception Module

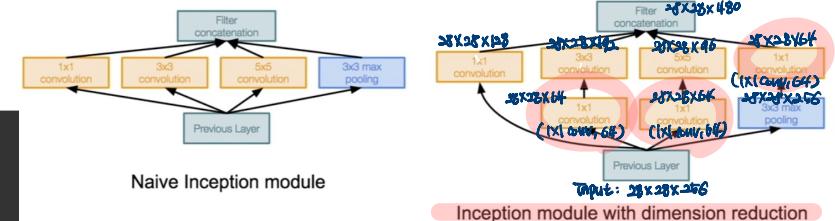
Inception Module은 동일한 입력을 받는 Keras 다른 다양한 블록들이

'병렬'로 처리

(1x1 / 3x3 / 5x5 convolution pooling)로 일정. → 각 레이어에서 나온 동작들을
모두 Depth 방향으로 합침 → 전체의 tensor를 축소되고 이를 다음 레이어로 전달



원하시는
계산량이
너무 많다는거.
↓
따라서 | 이유
bottleneck
layer 사용.

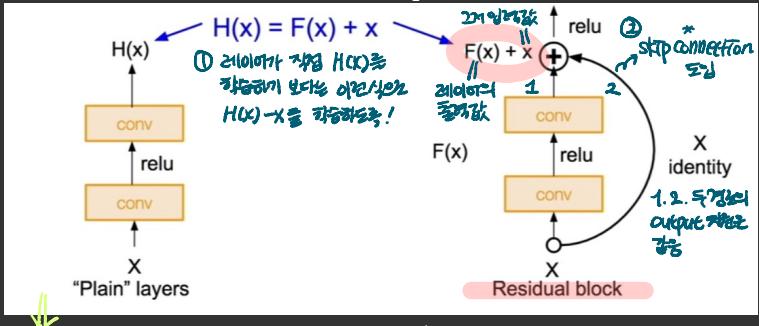


주요 아이디어는, 서로 입력의 depth를 줄이는 것
→ 각 레이어의 계산량은 1×1 conv를 통해 줄여온다

- GoogleNet은 Inception module을 네트워크 만들었음

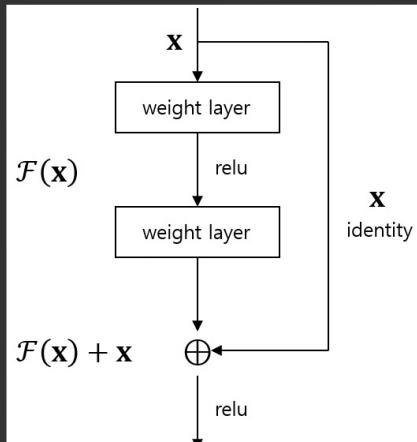
ResNet (2015)

(152 layers)
very deep networks using residual connections



$H(x)$ 는 계산하기 어려움. $H(x) = F(x) + x$ 이니
 $F(x)$ 를 계산하기보다. $H(x)$ 를 직접 배우는 대신에 x 에
 훨씬 더 쉽게 대하고 빼야하는 경우, 나중에 더 만들기 생각
 예전 $F(x)$ 가 residual!!

* skip connection
 : 가중치가 없으며, 입력을 identity mapping으로 그대로 출력단으로 내보냄
 그동안 신경레이어에는 변화량 (δ)만
 학습하였음. (=입력에 대한 정차)
 ∵ 학습해야하는 Input x + Residual



위 그림이 ResNet을 가장 잘 나타내는 그림입니다. 그림을 잘보면 신경망의 입력값은 x 인데 이 입력값 x 가 가중치 레이어를 거쳐서 만드는 값이 $F(x)$ 입니다. 그리고 오른쪽을 보면 그냥 입력값 x 자체가 레이어를 거치지 않고 그대로 건너 뛰는데 이를 identity mapping이라고 합니다. identity mapping은 입력값과 출력값이 같은 맵핑을 의미합니다. identity mapping은 skip connection 또는 identity shortcut connection이라고도 부릅니다. 따라서 x 라는 입력값이 신경망을 거치고 난 결과는 가중치를 거친 값 $F(x)$ 와 identity mapping을 거쳐 만들어진 x 더한 $F(x) + x$ 가 됩니다. 따라서 최종 출력값이 y 라고 하면 y 는 다음과 같이 쓸 수 있습니다.

$$y = F(x) + x$$

이것이 가장 간단한 형태의 ResNet입니다.