

딥러닝 - Part1_v1

2020-12-08 이동환



목차

Part1 - 개념

1. 인공지능

1. 약 인공지능과 강 인공지능

2. 머신러닝

1. 순서도
2. 지도 학습 - 분류와 회귀
 1. 선형 회귀
 2. 경사 하강법
 1. 고급 경사 하강법(일차미분 최적화)
 2. 이차미분 최적화
 3. 다중 선형 회귀
 4. 로지스틱 회귀

1. 딥러닝

1. 딥러닝 프레임워크(with GPU)
2. 퍼셉트론
3. 퍼셉트론의 한계(XOR 문제)
4. 오차 역전파
5. 기울기 소실 문제
 1. Activation Function
 2. Batch Normalization
6. CNN(Convolutional Neural Network)
 1. Convolution 연산
 2. CNN 구조 – Convolution, Pooling, FC layer
 3. 유명한 CNN Model – ImageNet Challenge

인공지능

Artificial Intelligence

인공지능

사고나 학습 등 인간이 가진
지적 능력을 컴퓨터를 통해
구현하는 기술



Machine Learning

머신러닝

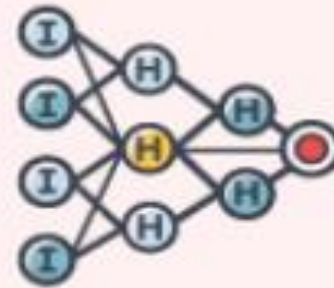
컴퓨터가 스스로 학습하여
인공지능의 성능을
항상 시키는 기술 방법



Deep Learning

딥러닝

인간의 뉴런과 비슷한
인공신경망 방식으로
정보를 처리



약 인공지능과 강 인공지능

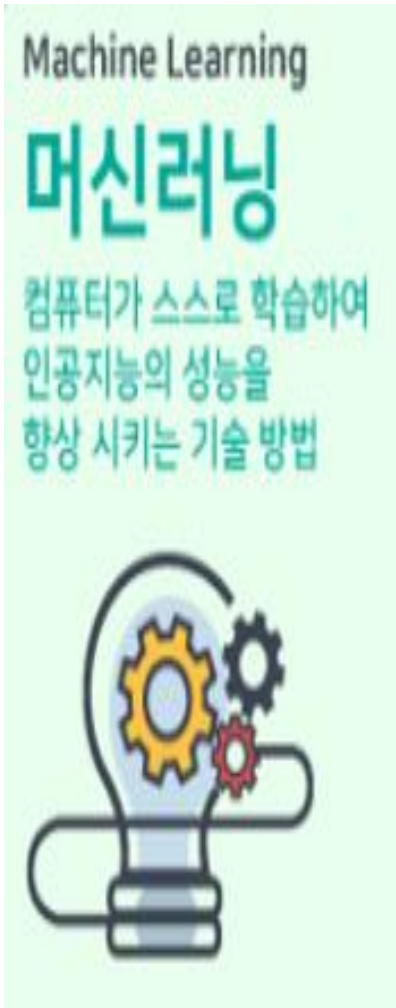


약 인공지능 - 알파고



강 인공지능 - 터미네이터

머신러닝



지도학습

- 훈련 데이터와 출력 사이의 연관성을 매핑해 학습하고, 학습하지 않은 데이터에 적용
- 분류, 회귀는 지도 학습 알고리즘의 두 가지 주요 유형

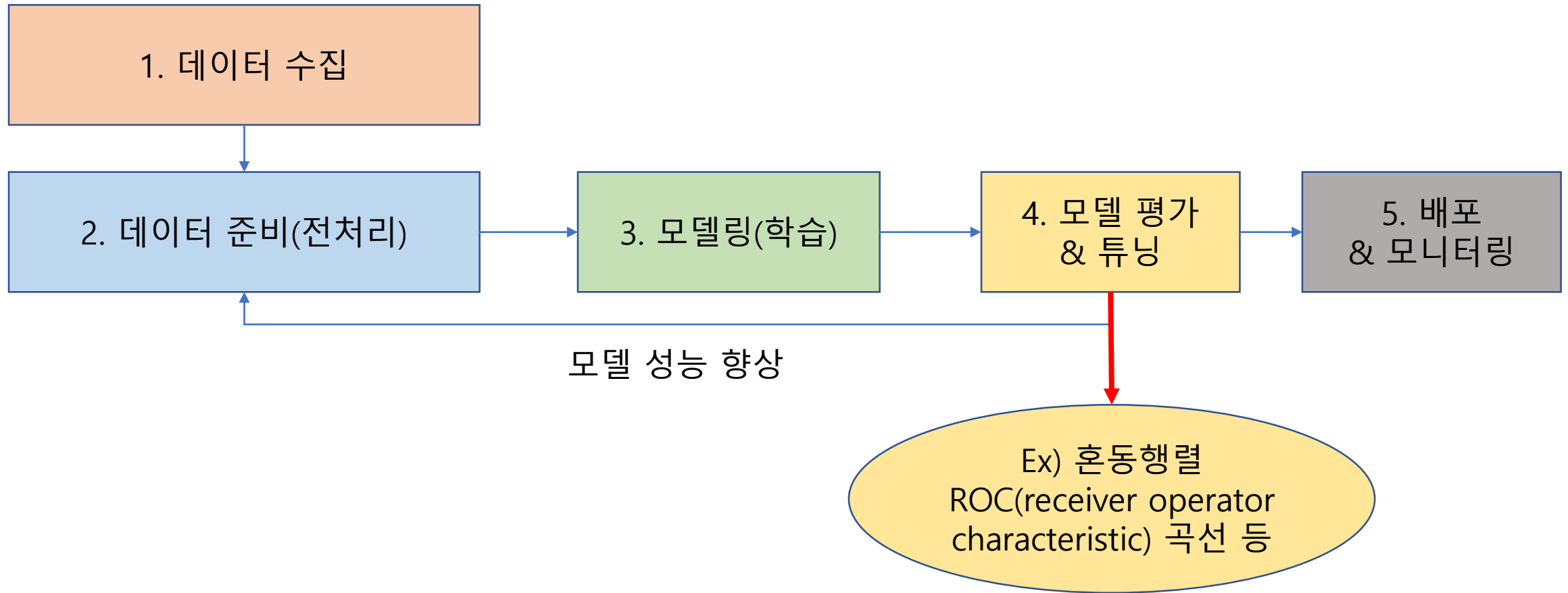
비지도학습

- 입력과 출력 레이블을 연관시키지 않고, 입력 데이터에 내재된 잠재적 패턴, 관계 등 학습
- 군집화, 차원 축소 등

강화학습

- 중심 주체인 에이전트를 일정 기간 환경과 상호 작용하면서 보상/벌칙에 기초해 전략/정책을 반복적으로 배우고 바꾸며, 보상을 극대화하도록 훈련함

순서도



(참고) 모델 평가 방법

		예측	
		FALSE	TRUE
실제	FALSE	1(TN)	2(FP)
	TRUE	1(FN)	6(TP)

$$\begin{aligned}
 (\text{Precision}) &= \frac{TP}{TP + FP} & (\text{Recall}) &= \frac{TP}{TP + FN} \\
 (\text{Accuracy}) &= \frac{TP + TN}{TP + FN + FP + TN} & \text{Fall-out(FPR)} &= \frac{FP}{TN + FP} \\
 (\text{F1-score}) &= 2 \times \frac{1}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
 \end{aligned}$$

코로나 검사자 10명 중

Accuracy : $[(1+6)/(1+2+1+6)]$ - "검사 키트"가 전체 사람 중 양성(6명)은 양성, 음성(4명)은 음성으로 분류했다.

-> 실제 걸리지 않았는데, 걸렸다고 판단 / 걸렸는데 걸리지 않았다고 판단하는 경우

Recall : $[6/(6+1)]$ - 실제로 양성(6명), 음성(1명)인 사람 중 검사 키트가 양성(6명)을 찾아냈다.

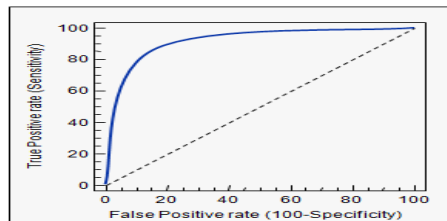
Precision : $[6/(6+2)]$ - 검사 키트가 8명을 확진자로 판정했고, 그 중 실제 양성(6명)이다.

F1-score : 특정 데이터만 잘 맞추는 경우, precision과 recall의 조화평균을 지표로 사용

ROC 곡선

가로 : FPR

세로 : Recall



세로 : 실제 걸렸는데, 걸렸다고 판단

가로 : 실제 걸리지 않았는데, 걸렸다고 판단

지도학습 - 분류



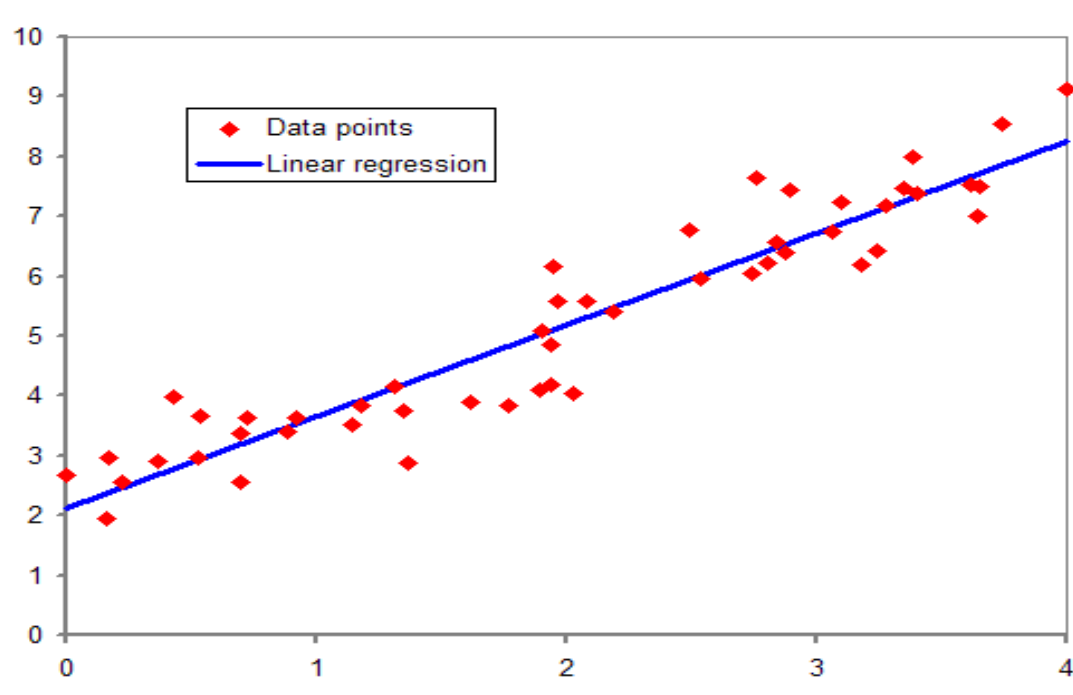
개 vs 고양이
이진분류문제



MNIST
다중분류문제

알고리즘 : 로지스틱 회귀 분석, 서포트 벡터 머신(SVM), 신경망, 랜덤 포레스트, k-NN, 의사결정 트리 등

지도학습 - 회귀



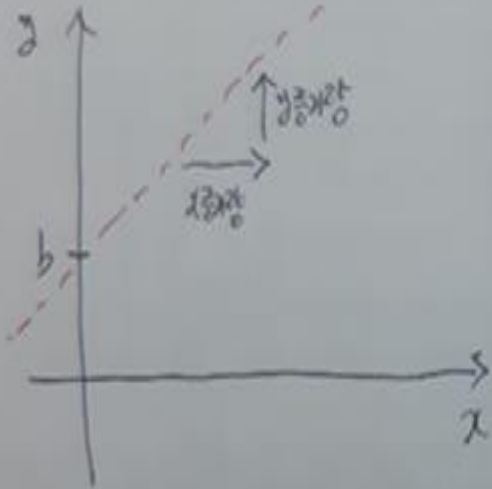
알고리즘

단순 선형 회귀 - 하나의 종속변수(출력)과 하나의 독립변수(입력) 사이의 관계 분석

다중 선형 회귀 - 하나의 종속변수(출력)과 여러 독립변수(입력) 사이의 관계 분석

선형 회귀

공부한 시간 (x)	2시간	4시간	6시간	8시간
성적 (y)	81	93	91	97



$$y = ax + b$$

기원지 정해진

"정확한 a와 b를 찾아야 한다."

[최소제곱법 (method of least squares)]

$$a = \frac{\sum_{i=1}^n (x - \text{mean}(x)) (y - \text{mean}(y))}{\sum_{i=1}^n (x - \text{mean}(x))^2}$$

$$b = \text{mean}(y) - (\text{mean}(x) * a)$$

[평균 제곱근 오차]

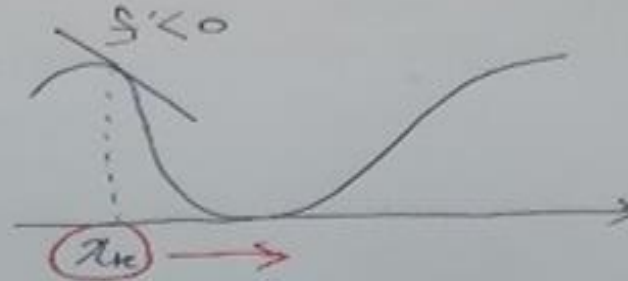
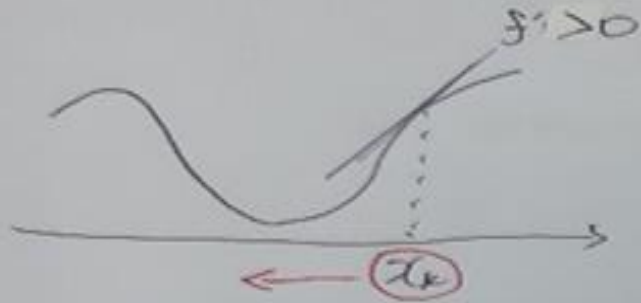
실제값과 예측값 간의 차이. (오차) \rightarrow 음수 때문에 오차² 사용

$$\text{평균 제곱 오차 (MSE)} = \frac{1}{n} \sum_{i=1}^n (p_i - y_i)^2$$

경사 하강법(오차 최소화)

[일차미분을 이용한 최적화]

$f(x) \Rightarrow$ 최소화 하고자 하는 함수 (목적 함수).



$$x_{k+1} = x_k - \lambda f'(x_k)$$

λ learning rate

\rightarrow 너무 작으면 오래 걸림.

\rightarrow 너무 크면 벗어나버림.

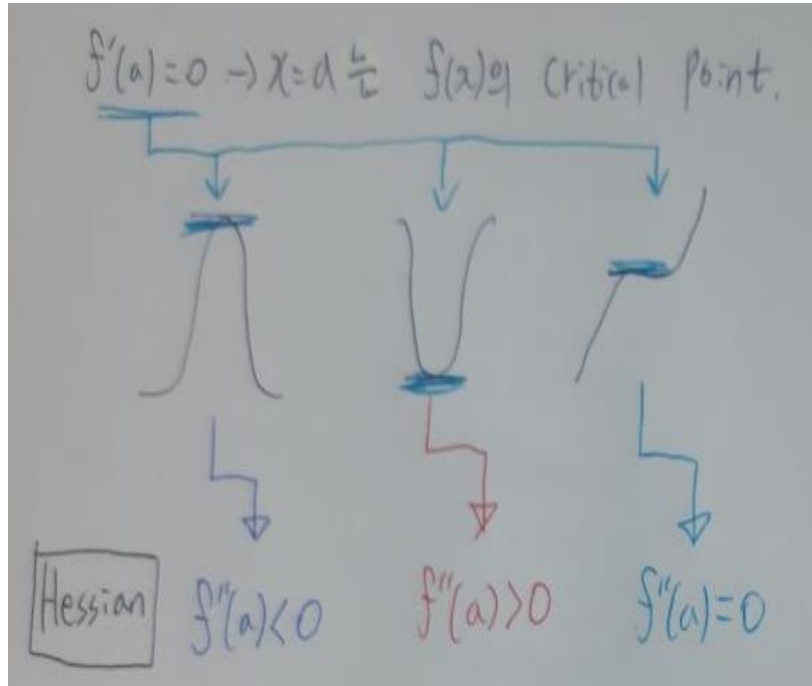


방향은 올바르게지만, 수렴 속도와 learning rate를 결정하기 어렵다.

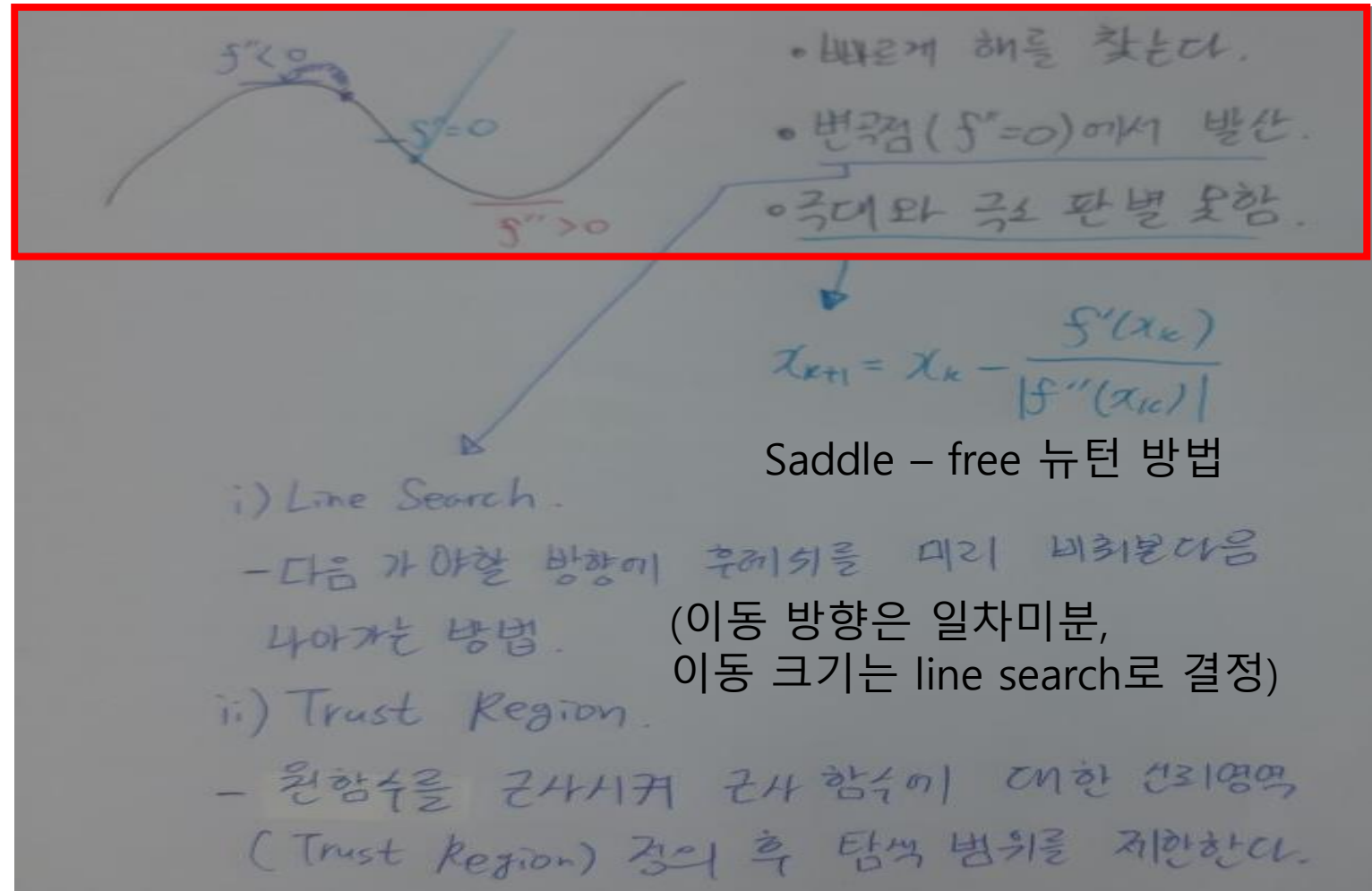
고급 경사 하강법(1차 미분)

고급 경사 하강법	설명
확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터만 사용
모멘텀 (Momentum)	운동량을 적용하여 정확도를 개선시킨다. (사발 가장자리에서 구슬을 굴렸을 때, local minima(구덩이)에 빠지면 구슬이 조금 올라갔다가 계속 내려감)
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 이동하여, 기울기를 계산한 다음 이동한다. (구슬이 사발 밑바닥으로 접근 하면, 구슬에 브레이크를 걸어줌)
아다그라드 (Adagrad)	변수가 자주 변하면 학습률 낮춰주고, 변수가 자주 변하지 않으면 학습률 높여 속도를 조절한다.
알엠에스프롭 (RMSProp)	아다그라드의 속도 조절 민감도를 보완한다.
아담 (Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법

이차미분을 통한 최적화



$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \quad (\text{Newton 방법})$$



다중 선형 회귀

공부한 시간 (x_1)	2시간	4시간	6시간	8시간
과외 (x_2)	0	4	2	3
성적 (y)	81	93	91	97

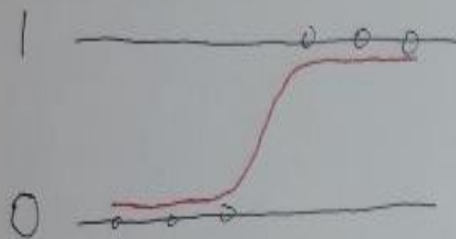
$$y = a_1x_1 + a_2x_2 + b$$

- 실제로 공부한 시간 외 다른 요인(과외) 또한 성적에 영향을 미칠 수 있다.
- 하나의 종속 변수(출력)에 독립 변수(입력)이 2개인 다중 선형 회귀 문제
- 선형 회귀 문제와 같이 해결 가능

로지스틱 회귀

공부한 시간 2 4 6 8 10 12

합격 여부 0 X X X 0 0 0 (1)



" $y=ax+b$ " 직선으로 그리기 어렵다.

↓
"시그모이드 함수"

$$y = \frac{1}{1 + e^{-(ax+b)}}$$

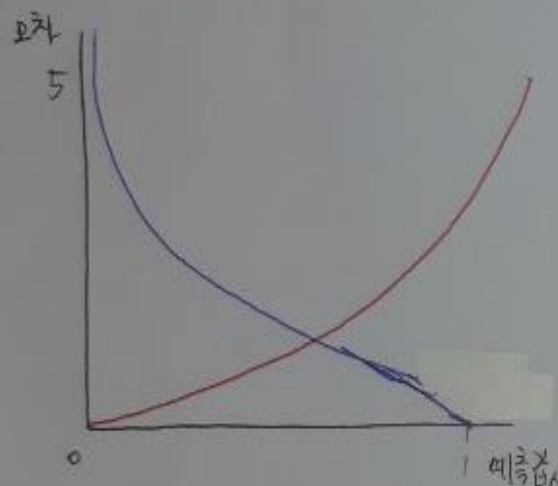
a: 그래프의 경사도 (↑: 좌우)

b: 그래프의 좌우이동 (←: →)

[오차 공식]

$$L = -\frac{1}{N} \sum_{i=1}^N t_i \log(y_i) + (1 - t_i) \log(1 - y_i)$$

$$\text{오차} = -\text{평균} \left(y \cdot \log y + (1-y) \cdot \log(1-y) \right)$$



파란색: 실제 값이 1인 경우.

(예측값이 1에 가까울수록 오차 0에 근접)

빨간색: 실제 값이 0인 경우.

(예측값이 0에 가까울수록 오차 0에 근접)

딥러닝



모형의 복잡도를 주어진 훈련 자료의 양이나 계산 능력에 맞게 손쉽게 조정 가능
- 신경망 구조에 뉴런들을 더 추가하거나 제거

자료 수집 기술이 발전하며 '빅데이터' 시대 도래
- 데이터 쉽게 수집하고 저장 가능

강력한 GPU의 발전
- 계산 속도 증가, 실행 시간 감소를 통한 효율적인 실험과 검사 가능

CNN : 데이터의 특징을 추출하여 특징들의 패턴 파악(영상)
RNN : 반복적이고 순차적인 데이터 학습에 특화(음성, 텍스트)

딥러닝 프레임워크(with GPU)

텐서플로(TensorFlow)

- 가장 많이 쓰이는 프레임워크, Static Graph(모델을 구축하고, 같은 그래프를 학습 시 반복적 사용)
- Define-and-Run

케라스(Keras)

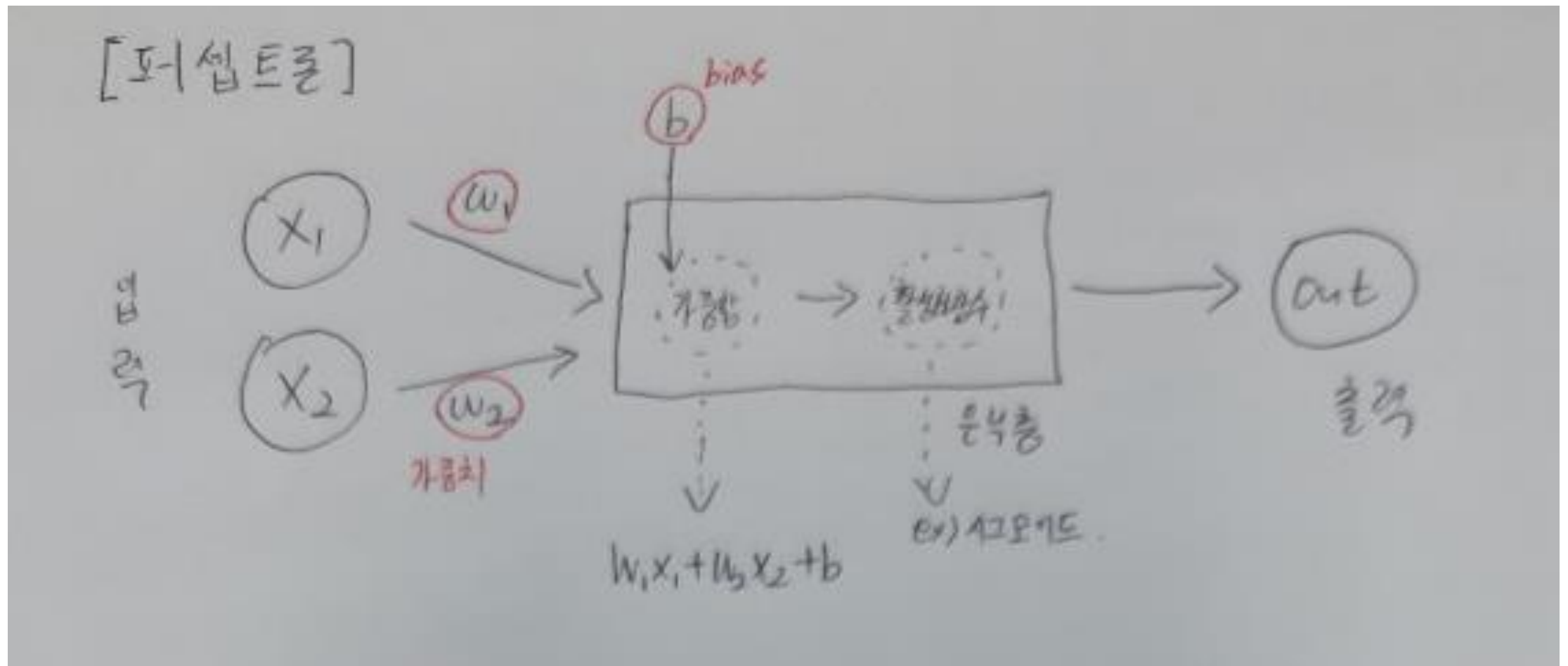
- 최소한의 코드로 효율적인 딥러닝 모델을 구축할 수 있는 고수준 API 제공

파이토치(PyTorch)

- 텐서플로보다 성능이 좋음, 코드 간결함, 많은 연구 진행 중, Dynamic Graph(반복마다 새로운 graph 구성 가능)
- Define-by-Run

하드웨어(GPU)의 발전 : 딥러닝 계산은 주로 행렬 연산인데, GPU를 통해 병렬로 빠르게 처리 가능

퍼셉트론



(참고) 편향(bias) 분산(variance) tradeoff

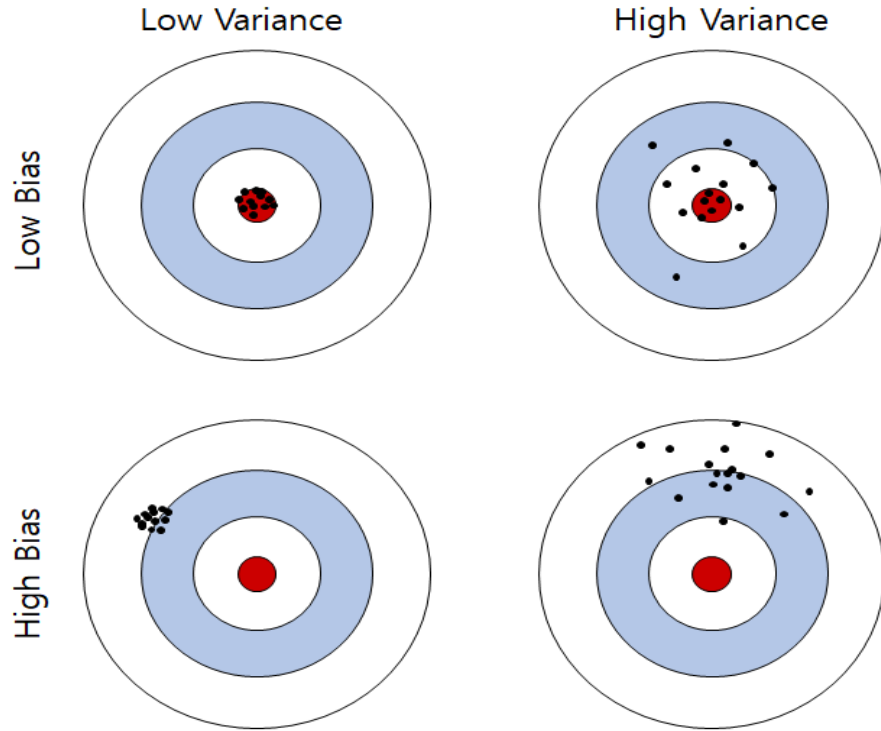


Fig 1.

빨간 점 : True, 검은 점 : False
Bias : (검은 점이)빨간 점과 떨어진 정도
Variance : 검은 점이 퍼져 있는 정도

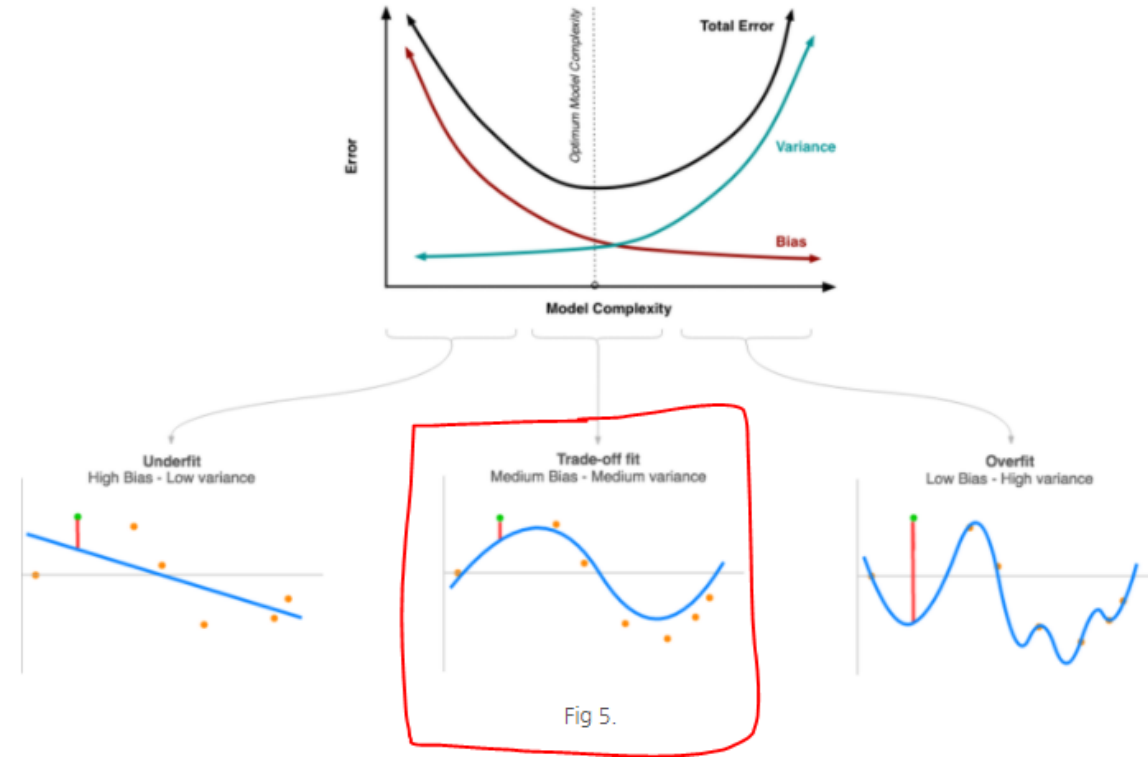


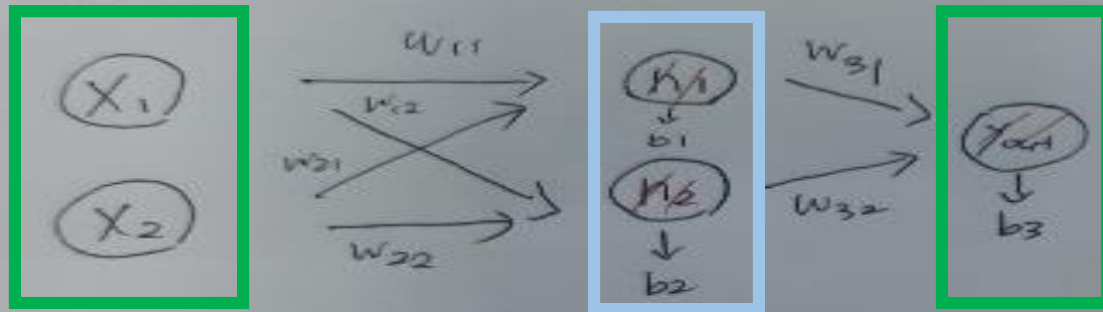
Fig 5.

[Underfitting]
모델이 매우 단순하며, 큰 편차는 없으나(Low Variance), Bias는 높다.
[Overfitting]
모델이 매우 복잡하며, 편차가 크지만(High Variance), Bias는 낮다.

퍼셉트론의 한계(XOR 문제)

[다층 퍼셉트론]

"XOR" 문제 해결을 위하여. (입력-출력 \rightarrow 입력-은닉-출력)



$$\underline{n_1} = \underline{\sigma} (x_1 w_{11} + x_2 w_{21} + b_1)$$

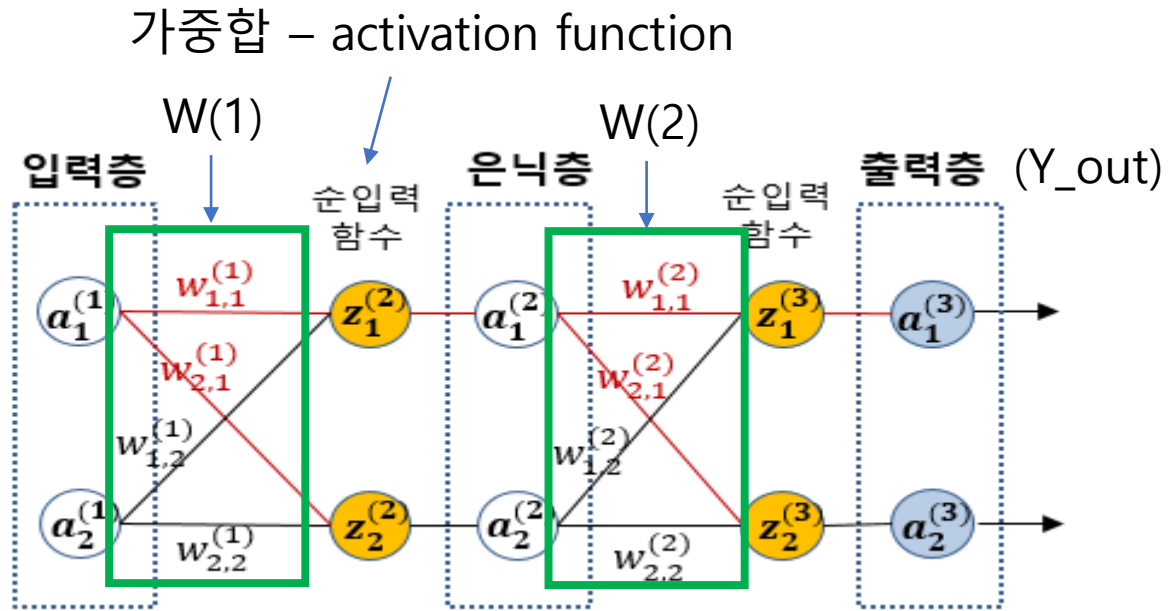
시그모이드

$$\underline{n_2} = \underline{\sigma} (x_1 w_{12} + x_2 w_{22} + b_2)$$

$$\underline{y_{out}} = \underline{\sigma} (n_1 w_{31} + n_2 w_{32} + b_3)$$

어떤 w 와 어떤 bias를 사용하든
 \Rightarrow 오차역전파(back propagation)

오차역전파(Back Propagation)



Y_out 값에서 거꾸로 거슬러 올라가며 가중치 W(2)와 W(1)이 더 이상 업데이트 되지 않을 때 까지 반복하여 계산하는 것

편미분 (Partial derivative) 상수 취급 (Variable)

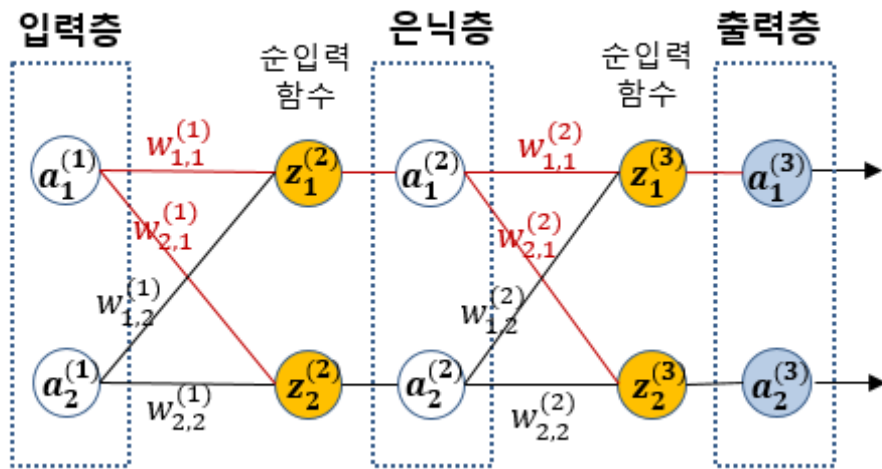
$$f(x, y) = x y \quad \frac{\partial y}{\partial x} = y$$

상수 취급 (매개변수)

$$f(x, y) = x y \quad \frac{\partial x}{\partial y} = x$$

상수 취급 (매개변수)

오차역전파(Back Propagation)



1. 순전파 방향으로 진행
2. 역전파 방향으로 진행
3. 가중치 업데이트

$$z_1^{(2)} = w_{1,1}^{(1)} a_1^{(1)} + w_{1,2}^{(1)} a_2^{(1)}$$

$$z_2^{(2)} = w_{2,1}^{(1)} a_1^{(1)} + w_{2,2}^{(1)} a_2^{(1)}$$

$$a_1^{(2)} = \phi(z_1^{(2)})$$

$$a_2^{(2)} = \phi(z_2^{(2)})$$

(ϕ = activation function)

입력층 -> 은닉층

$$z_1^{(3)} = w_{1,1}^{(2)} a_1^{(2)} + w_{1,2}^{(2)} a_2^{(2)}$$

$$z_2^{(3)} = w_{2,1}^{(2)} a_1^{(2)} + w_{2,2}^{(2)} a_2^{(2)}$$

$$a_1^{(3)} = \phi(z_1^{(3)})$$

$$a_2^{(3)} = \phi(z_2^{(3)})$$

Cost function

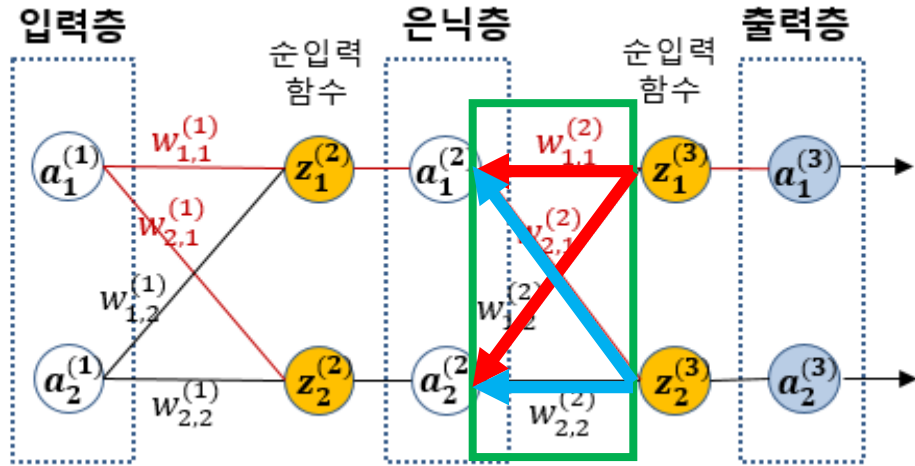
$$J_1 = \frac{1}{2} (a_1^{(3)} - y_1)^2$$

$$J_2 = \frac{1}{2} (a_2^{(3)} - y_2)^2$$

$$J = J_1 + J_2$$

은닉층 -> 출력층

오차역전파(Back Propagation)



$$\frac{\partial J_{out}}{\partial w_{1,1}^{(2)}} = \boxed{\frac{\partial J_1}{\partial a_1^{(3)}}} \times \boxed{\frac{\partial a_1^{(3)}}{\partial z_1^{(3)}}} \times \boxed{\frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}}}$$

$$\begin{aligned} \frac{\partial J_1}{\partial a_1^{(3)}} &= \frac{1}{2} \frac{\partial}{\partial a_1^{(3)}} (a_1^{(3)} - y_1)^2 = (a_1^{(3)} - y_1) \\ \frac{\partial a_1^{(3)}}{\partial z_1^{(3)}} &= \phi(z_1^{(3)}) (1 - \phi(z_1^{(3)})) = a_1^{(3)} (1 - a_1^{(3)}) \\ \frac{\partial z_1^{(3)}}{\partial w_{1,1}^{(2)}} &= a_1^{(2)} \end{aligned}$$

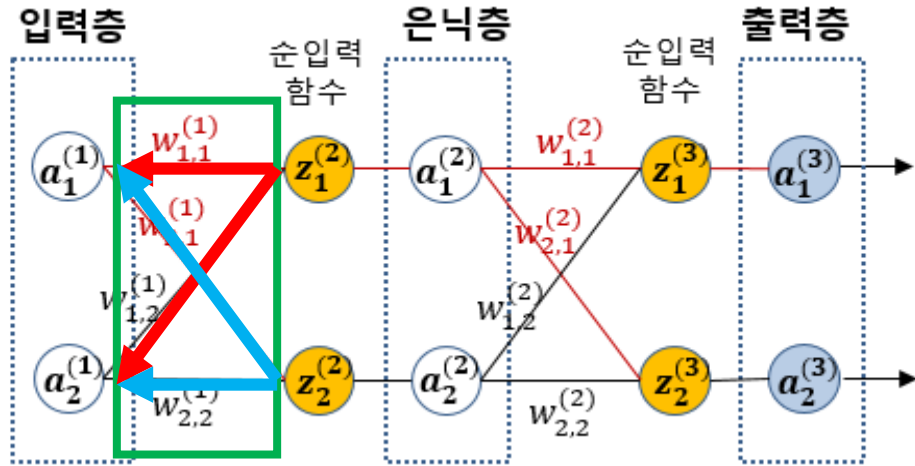
→ 시그모이드 미분

$$\frac{\partial \phi(x)}{\partial x} = \phi(x) \cdot (1 - \phi(x))$$

$$\begin{aligned} \frac{\partial J_{out}}{\partial w_{1,1}^{(2)}} &= (a_1^{(3)} - y_1) \times a_1^{(3)} \times (1 - a_1^{(3)}) \times a_1^{(2)} \\ \delta_1^{(3)} &= \frac{\partial J_1}{\partial z_1^{(3)}} \quad \text{가중치 } w_{1,1}^{(2)} = w_{1,1}^{(2)} - \delta_1^{(3)} \cdot a_1^{(2)} \\ \text{가중치 } w_{1,2}^{(2)} &= w_{1,2}^{(2)} - \delta_1^{(3)} \cdot a_2^{(2)} \\ \delta_2^{(3)} &= \frac{\partial J_2}{\partial z_2^{(3)}} \quad \text{가중치 } w_{2,1}^{(2)} = w_{2,1}^{(2)} - \delta_2^{(3)} \cdot a_1^{(2)} \\ \text{가중치 } w_{2,2}^{(2)} &= w_{2,2}^{(2)} - \delta_2^{(3)} \cdot a_2^{(2)} \\ &\downarrow \\ &\underline{(a_2^{(3)} - y_2) \times a_2^{(3)} \times (1 - a_2^{(3)})} \end{aligned}$$

은닉층 <- 출력층

오차역전파(Back Propagation)



$$\frac{\partial J_{out}}{\partial w_{1,1}^{(1)}} = \frac{\partial J_{out}}{\partial a_1^{(1)}} \times \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial w_{1,1}^{(1)}}$$

$$\frac{\partial J_{out}}{\partial a_1^{(1)}} = \frac{\partial J_1}{\partial a_1^{(1)}} + \frac{\partial J_2}{\partial a_1^{(1)}} = \frac{\partial J_1}{\partial z_1^{(2)}} \times \frac{\partial z_1^{(2)}}{\partial a_1^{(1)}} + \frac{\partial J_2}{\partial z_2^{(2)}} \times \frac{\partial z_2^{(2)}}{\partial a_1^{(1)}}$$

$$= \delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}$$

$$\frac{\partial J_{out}}{\partial w_{1,1}^{(1)}} = (\delta_1^{(3)} w_{1,1}^{(2)} + \delta_2^{(3)} w_{2,1}^{(2)}) \times a_1^{(2)} \times (1 - a_1^{(2)}) \times a_1^{(1)}$$

$$\delta_1^{(2)} \text{ 가중치 } w_{1,1}^{(1)} = w_{1,1}^{(1)} - \delta_1^{(2)} \cdot a_1^{(1)}$$

$$\text{가중치 } w_{1,2}^{(1)} = w_{1,2}^{(1)} - \delta_1^{(2)} \cdot a_2^{(1)}$$

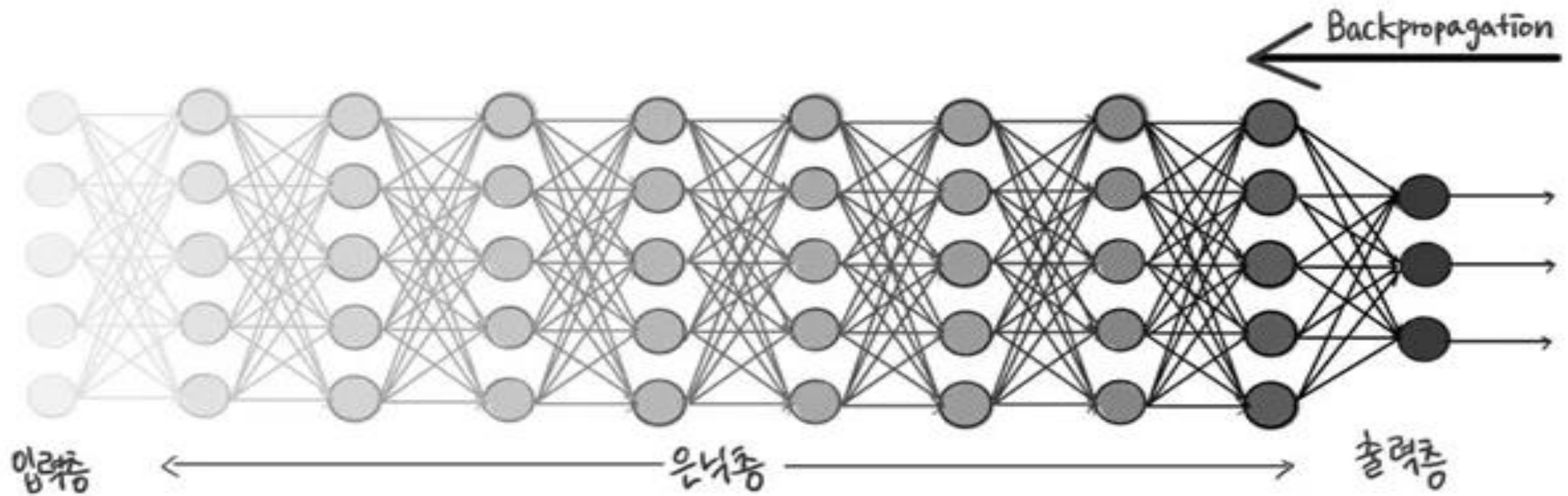
$$\delta_2^{(2)} \text{ 가중치 } w_{2,1}^{(1)} = w_{2,1}^{(1)} - \delta_2^{(2)} \cdot a_1^{(1)}$$

$$\text{가중치 } w_{2,2}^{(1)} = w_{2,2}^{(1)} - \delta_2^{(2)} \cdot a_2^{(1)}$$

$$(\delta_1^{(3)} w_{1,2}^{(2)} + \delta_2^{(3)} w_{2,2}^{(2)}) \times a_2^{(2)} \times (1 - a_2^{(2)})$$

입력층 <- 은닉층

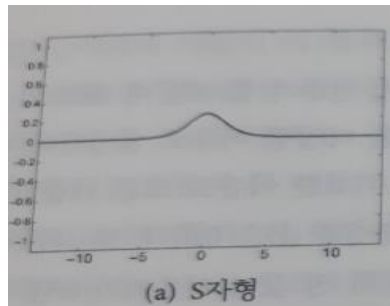
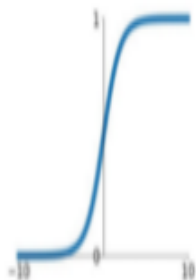
기울기 소실 문제



Activation Function1

Sigmoid

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

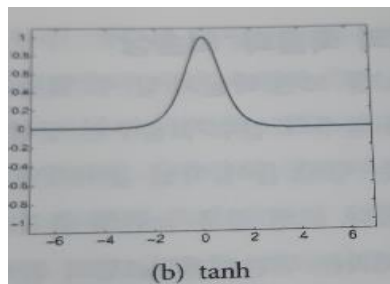
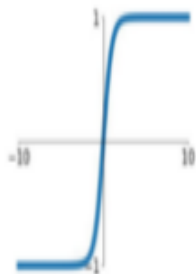


(a) S자형

Sigmoid : 미분 시 기울기 소실 문제 발생

tanh

$$\tanh(x)$$

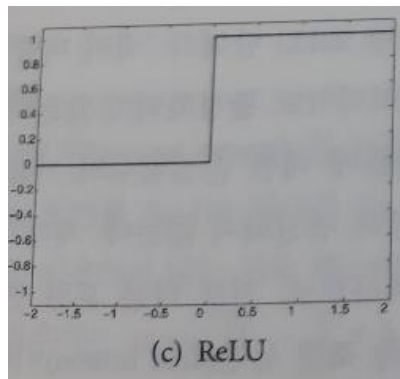


(b) tanh

tanh : 미분 시 기울기 소실 문제 발생
Sigmoid 범위를 -1에서 1로 확장

ReLU

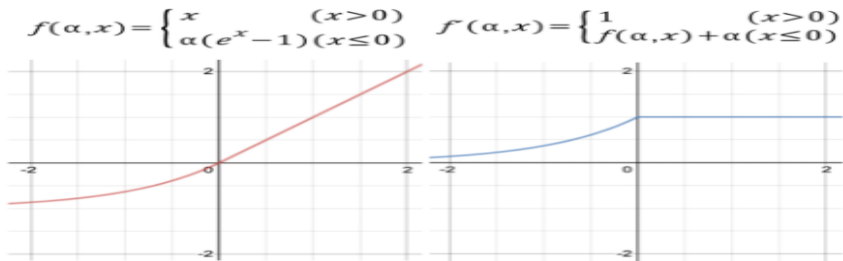
$$\max(0, x)$$



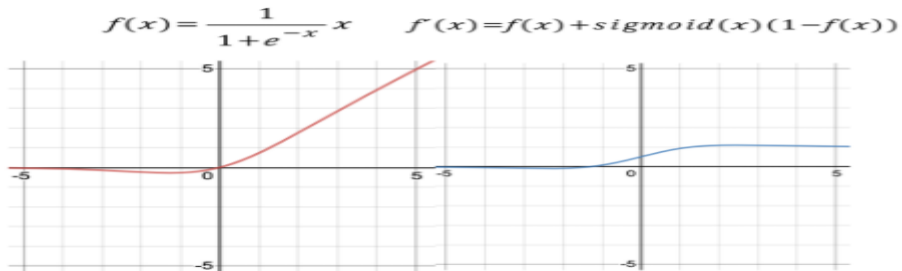
(c) ReLU

ReLU : 0보다 큰 구간에서 기울기 유지(2010)
dead neuron : 학습 속도가 빠르거나, 가중치가 0으로 설정된 경우 해당 뉴런은 어떤 자료가 입력되어도 갱신되지 않는다.

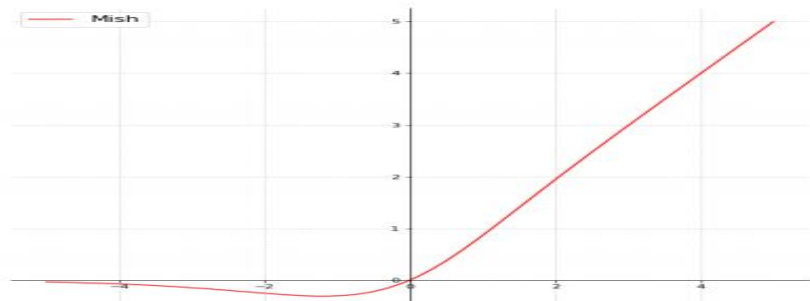
Activation Function2



ELU, SeLU : ReLU의 dead neuron 문제 해결
알파 값이 1이 아닌 경우 SeLU라고 부름(2016)



swish : 깊은 레이어 학습 시 ReLU보다 높은 성능
Mobilenet 학습 시 사용(2017)



mish : swish와 거의 비슷하다.
(2019)

$$f(x) = x * \tanh(\text{softplus}(x)) \quad \text{softplus}(x) = \ln(1 + e^x).$$

Batch Normalization

목표 : Training 과정 자체를 전체적으로 안정화 하여 학습 속도 가속

불안정화의 원인 : Internal Covariance Shift

- Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상
- 이 현상을 막기 위해 distribution을 평균 0, 표준편차 1인 input으로 normalize 시키는 방법 적용 시 해결 가능(whitening)
- 하지만, covariance matrix의 계산과 inverse의 계산이 필요하기 때문에 계산량이 많고, whitening을 하면 일부 parameter 들의 영향이 무시됨.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

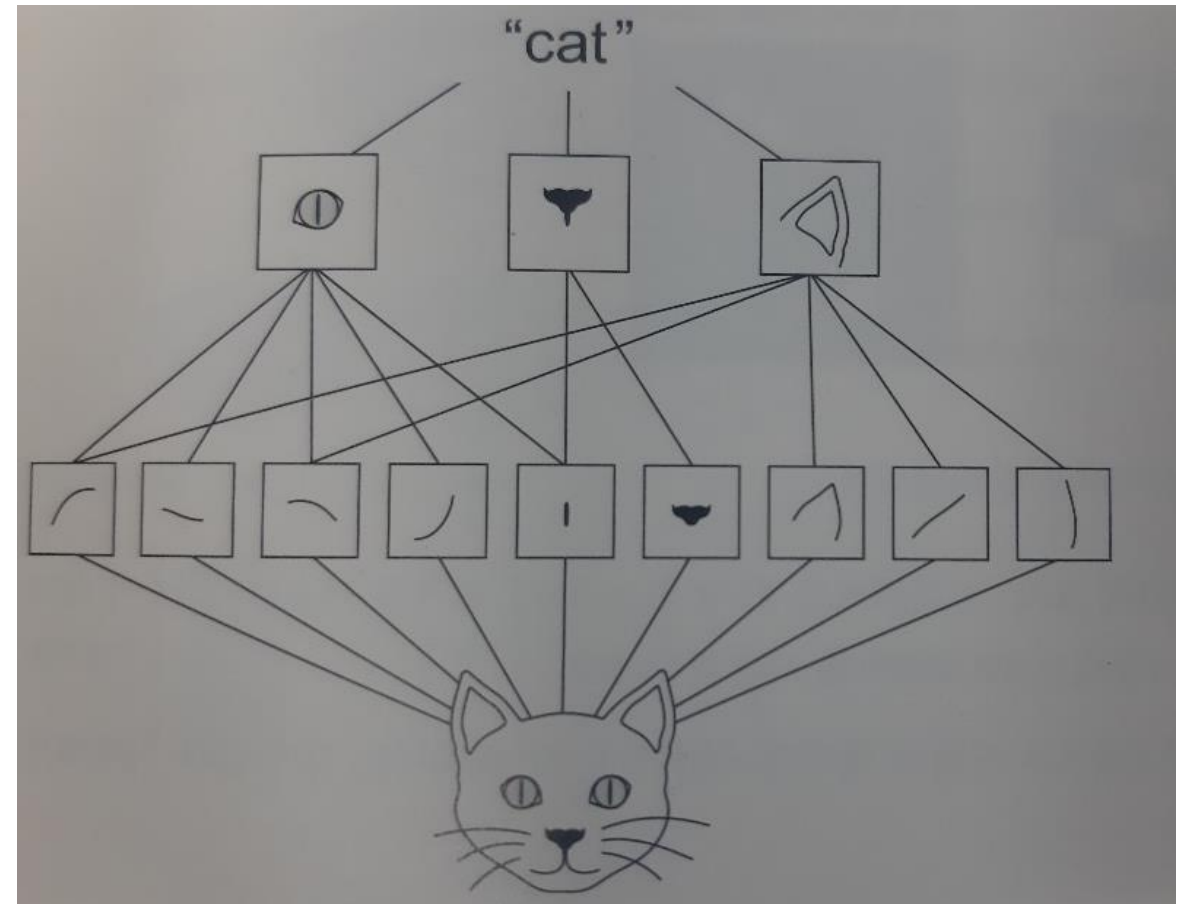
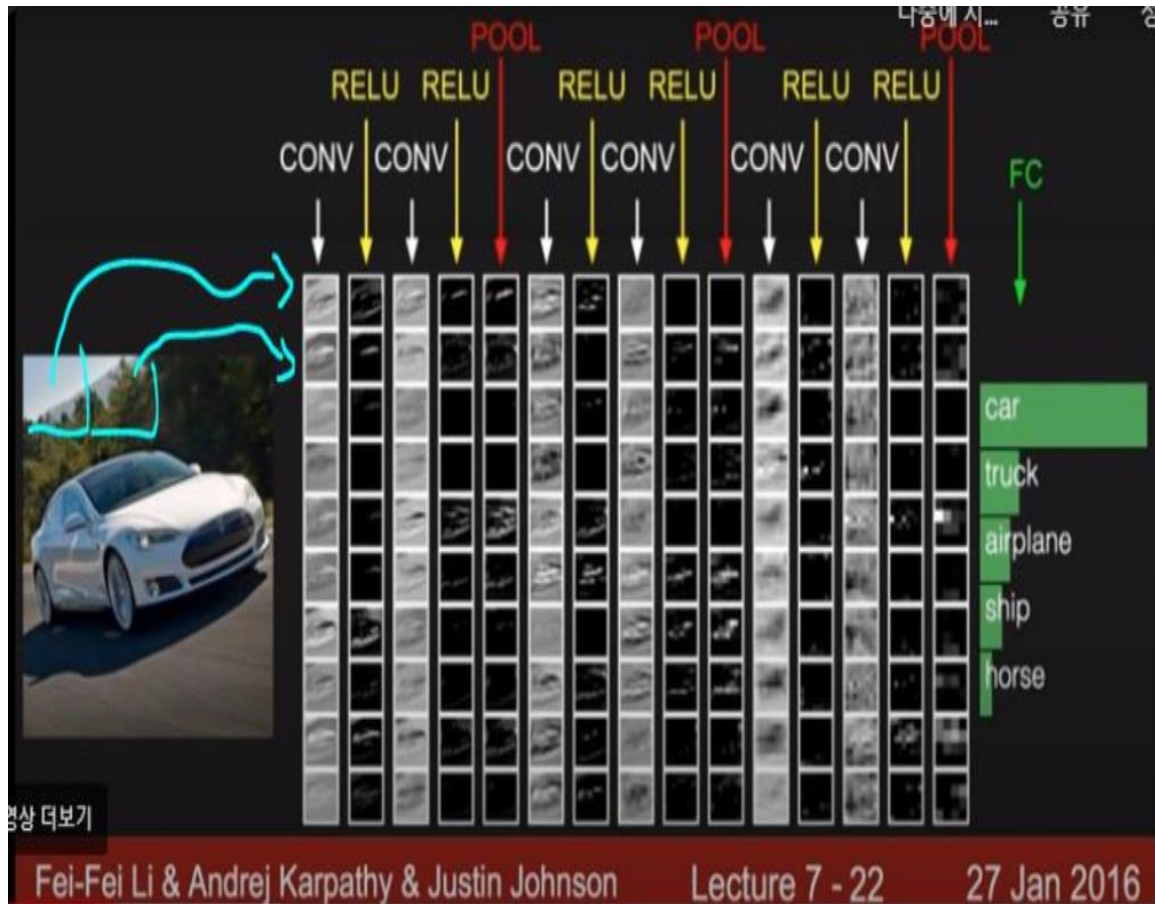
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

1. 가정 : 각각의 feature들이 이미 uncorrelated 되어있다.
feature 각각에 대해서만 scalar 형태로 mean과 variance를 구하고 각각 normalize 한다.

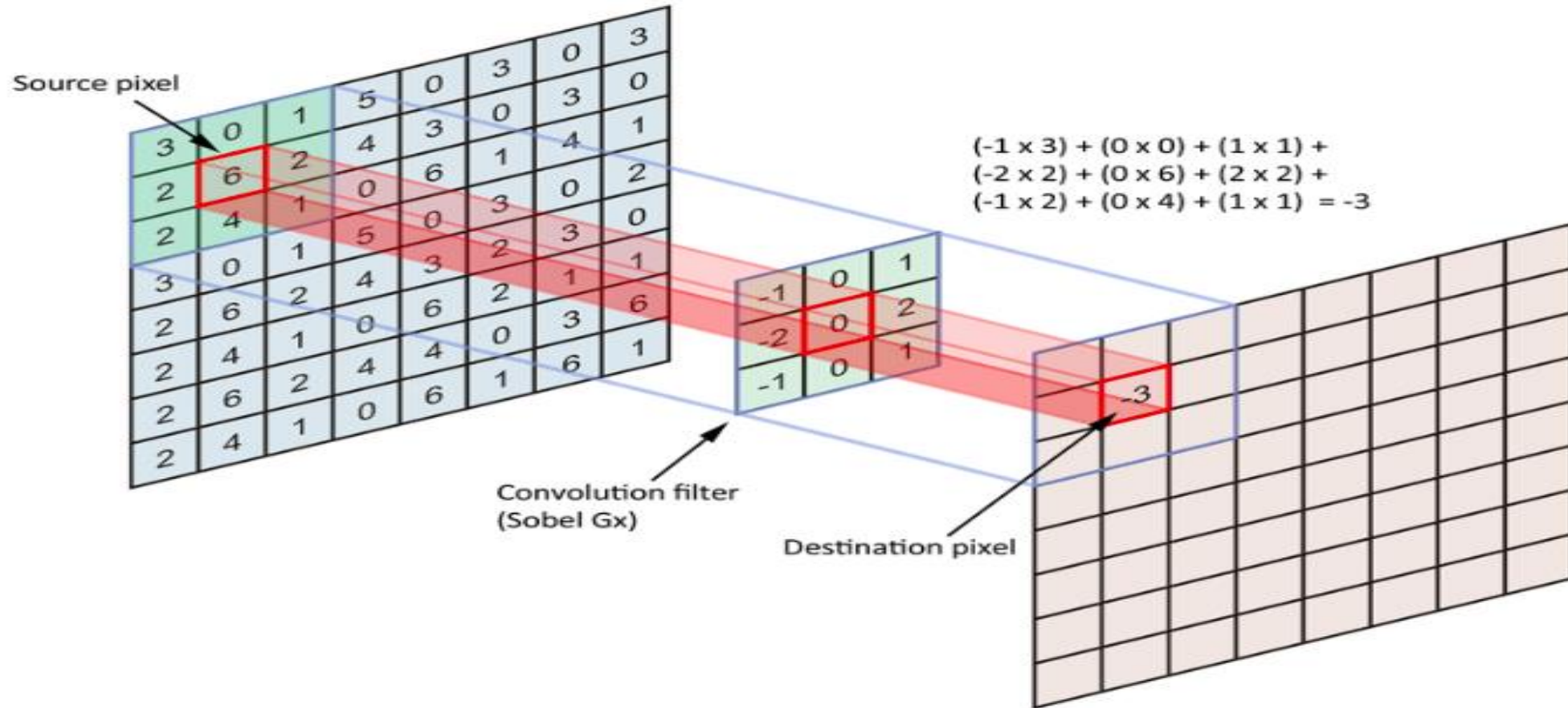
2. 단순히 mean과 variance를 0, 1로 고정시키는 것은 오히려 Activation function의 nonlinearity를 없앨 수 있다. 또한, feature가 uncorrelated 되어있다는 가정에 의해 네트워크가 표현할 수 있는 것이 제한될 수 있다. 이 점들을 보완하기 위해, normalize된 값들에 scale factor (gamma)와 shift factor (beta)를 더해주고 이 변수들을 back-prop 과정에서 같이 train 시켜준다.

CNN(Convolutional Neural Network)

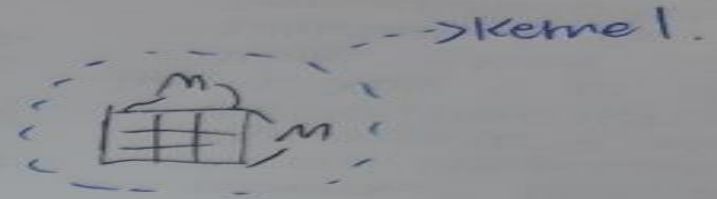
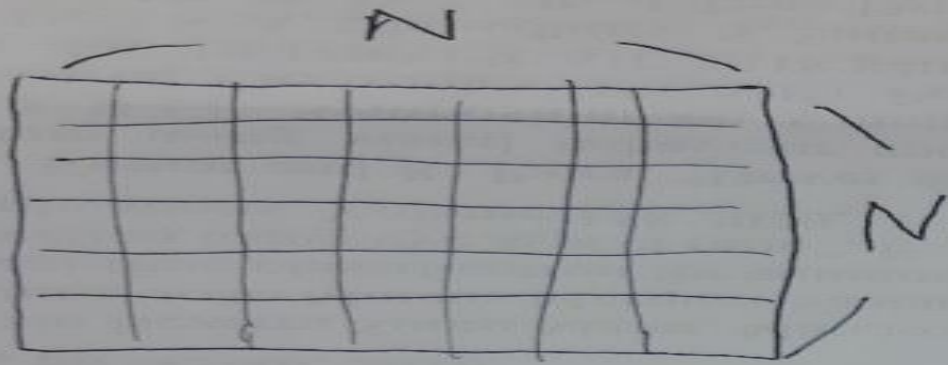


추천 구조 : Convolution -> (Batch Normalization) -> Activation -> (Dropout) -> Pooling

Convolution 연산

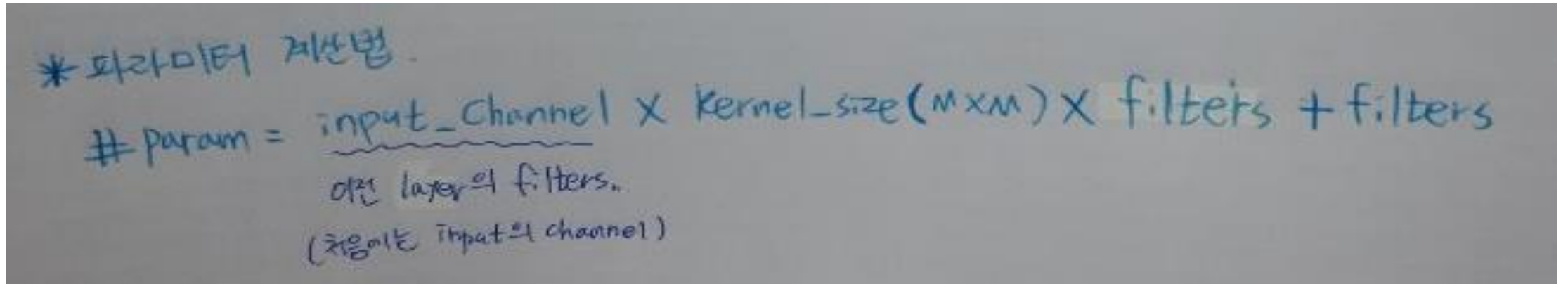


합성곱층(Convolution layer)



- ① $\text{Input_Shape} = (\text{height}, \text{width}, \text{channel})$
→ Input Image의 크기를 의미
- ② $\text{filters} = _$
→ filter의 수 (= Activation Map 개수)
- ③ $\text{kernel_Size} = (_, _)$
→ Convolution 연산을 위한 kernel size.
- ④ $\text{padding} = \text{"same"}$
→ 입력과 동일한 높이와 너비를 가진 특징 맵을 얻기 위한 방법. (zero)
- ⑤ $\text{activation} = \text{"relu"}$

(참고) parameter 계산법



* 파라미터 계산법.

$$\# \text{param} = \underbrace{\text{input_channel}}_{\substack{\text{이전 layer의 filters,} \\ \text{(처음에는 input의 channel)}}} \times \text{kernel_size}(M \times M) \times \text{filters} + \text{filters}$$

동일한 성능을 가진 모델이라면, 파라미터가 적을수록 좋다. – "SENet"(42page)

- More efficient distributed training : 병렬 학습 시 큰 효율
- Less overhead when exporting newmodels to clients : 실시간 서버 소통 시 과부하 감소
- Feasible FPGA and embedded deployment : 모델을 제한된 메모리에 직접 배치 가능

풀링층(Pooling layer)

1	1	2	4
5	6	7	<u>8</u>
<u>3</u>	2	1	0
1	2	3	<u>4</u>



6	<u>8</u>
<u>3</u>	<u>4</u>

[Max pooling]

① pool_size = (2,2)

→ (앞의 kernel과 같다고 생각)

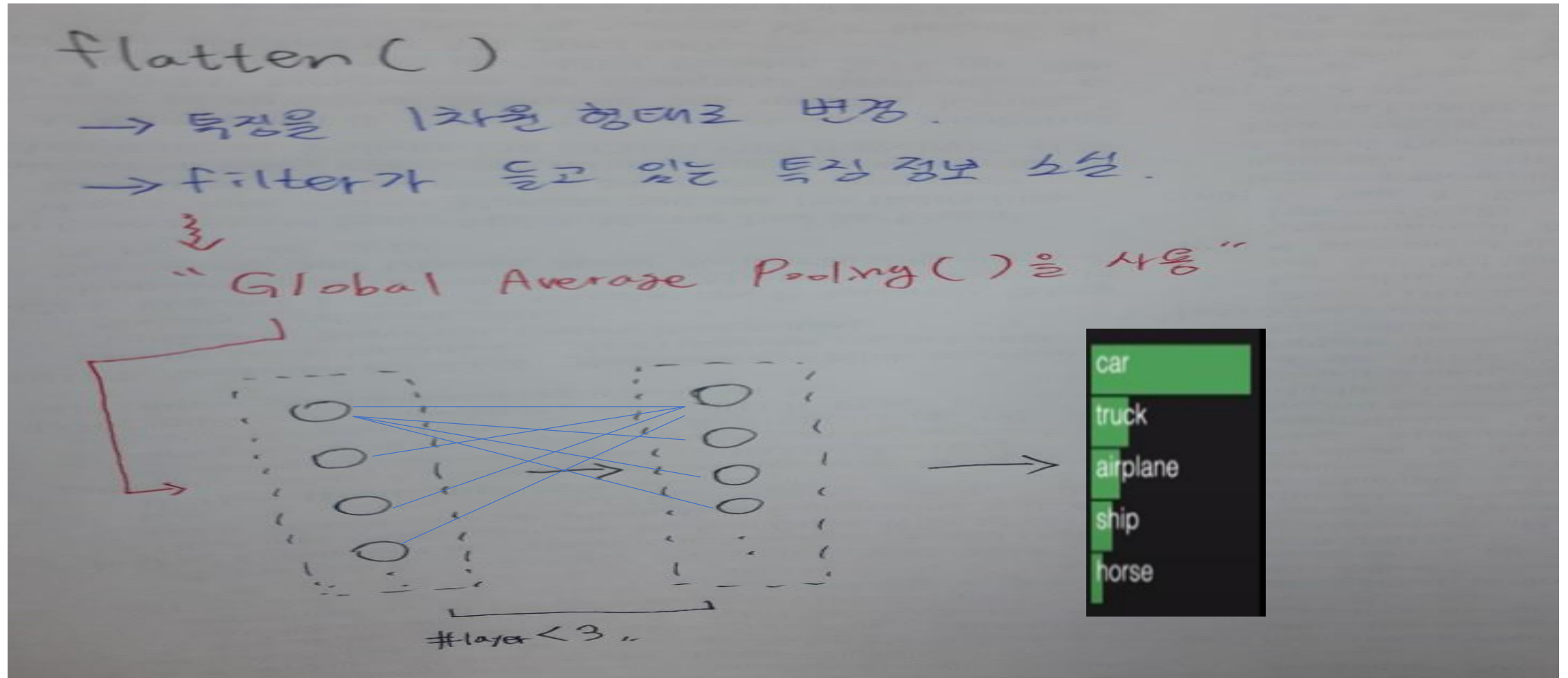
② stride s = 2

→ 얼마만큼씩 이동할지 결정

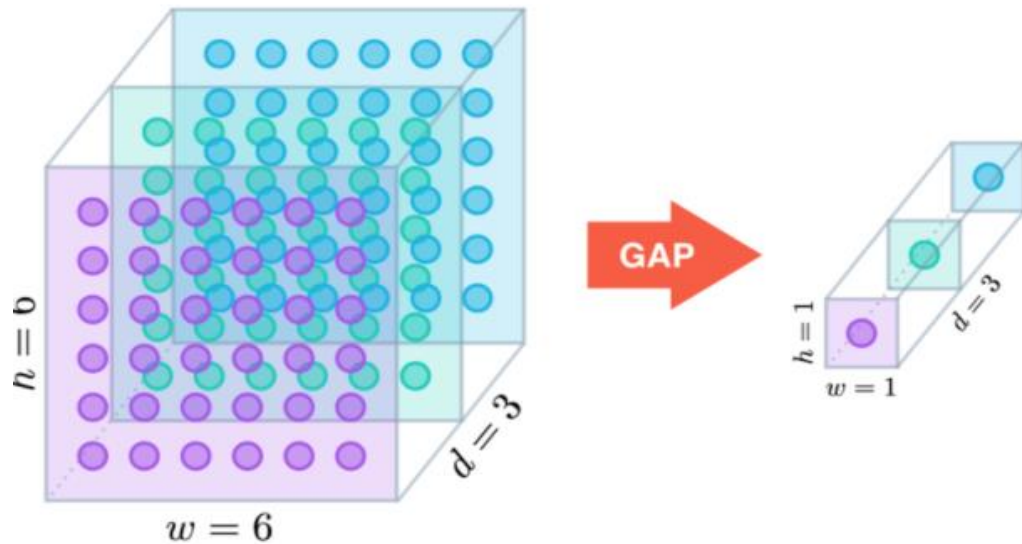
$$* \text{output_size} = 1 + (\text{Input_size} - \text{pool_size}) / \text{strides}$$

$$* \text{output_size} = 1 + (\text{Input_size} - \text{pool_size} + \text{padding} * 2) / \text{strides} \quad (\text{padding 있는 경우})$$

FC(Fully Connected layer)



(참고) Global Average Pooling

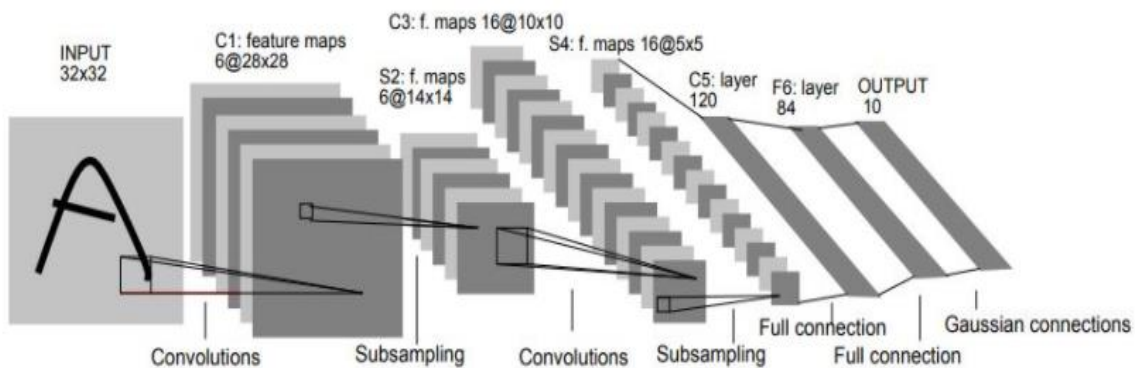


max_pooling2d_1 (MaxPooling2	(None, 7, 7, 128)	0
global_average_pooling2d (Gl	(None, 128)	0
dense (Dense)	(None, 500)	64500
activation_3 (Activation)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010
activation_4 (Activation)	(None, 10)	0
=====		
Total params: 326,534		
Trainable params: 326,534		
Non-trainable params: 0		

차원 변경 : (H,W,D) -> (1,1,D)

LeNet-5(1998)

LeNet - 5 Architecture:



초창기 합성곱 신경망(입력 : 32x32x1)

- 주로 문자 인식에 사용
- 얇은 합성곱 신경망
- 입력 : gray scale image

[LeNet-5 (1998)]

1 Conv2D(filters=6, kernel_size=(5,5), padding='valid', activation='tanh',
input_shape=(32,32,1))

⇒ output: $1 + (32 - 5) / 1 \Rightarrow (28, 28)$, #param: $(1 \times 5 \times 5 \times 6) + 6 = 156$.

2 Average Pooling2D(strides=2) ⇒ output: (14,14)

3 Conv2D(filters=16, kernel_size=(5,5), padding='valid', activation='tanh')

⇒ output: $1 + (14 - 5) / 1 \Rightarrow (10, 10)$, #param: $(6 \times 5 \times 5 \times 16) + 16 = 2,416$.

4 Average Pooling2D(strides=2) ⇒ output: (5,5)

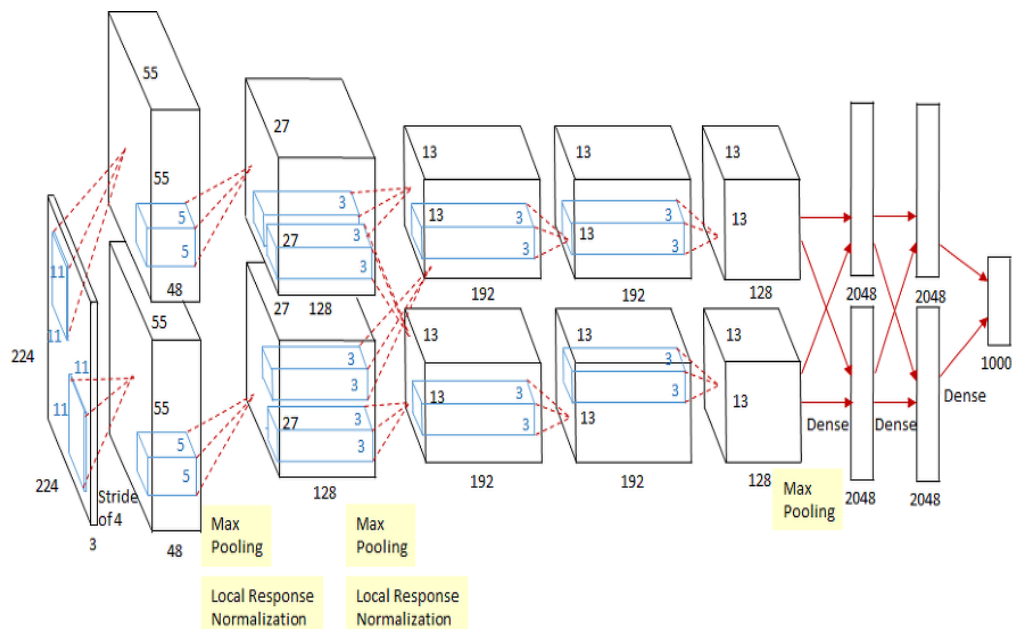
5 Flatten() ⇒ output: $5 \times 5 \times 16 = 400$.

6 Dense(120, activation='tanh') ⇒ #param: $400 \times 120 + 120 = 48,120$.

7 Dense(84, activation='tanh') ⇒ #param: $120 \times 84 + 84 = 10,164$.

8 Dense(10, activation='Softmax') ⇒ #param: $84 \times 10 + 10 = 850$.

AlexNet(2012)



3GB GTX 580 GPU 한 대로 훈련하려 했으나,
메모리 부족으로 두 대의 GPU로 분할함

- Local Response Normalization : ReLU 출력은 양수일 때 입력에 비례한다. 매우 높은 한 픽셀이 다른 픽셀에 미치는 영향을 방지하기 위해 사용
- 입력 : (227x227x3)

1th: Convolutional Layer: 96 kernels of size $11 \times 11 \times 3$
(stride: 4, pad: 0)

$$55 \times 55 \times 96 \text{ feature maps} \quad 3 \times 11 \times 11 \times 96 + 96 = 34,944$$

Then 3×3 Overlapping Max Pooling (stride: 2)

$$27 \times 27 \times 96 \text{ feature maps}$$

Then Local Response Normalization

$$27 \times 27 \times 96 \text{ feature maps}$$

2nd: Convolutional Layer: 256 kernels of size $5 \times 5 \times 48$
(stride: 1, pad: 2)

$$27 \times 27 \times 256 \text{ feature maps} \quad 96 \times 5 \times 5 \times 256 + 256 = 614,656$$

Then 3×3 Overlapping Max Pooling (stride: 2)

$$13 \times 13 \times 256 \text{ feature maps}$$

Then Local Response Normalization

$$13 \times 13 \times 256 \text{ feature maps}$$

3rd: Convolutional Layer: 384 kernels of size $3 \times 3 \times 256$
(stride: 1, pad: 1)

$$13 \times 13 \times 384 \text{ feature maps} \quad 256 \times 3 \times 3 \times 384 + 384 = 885,120$$

4th: Convolutional Layer: 384 kernels of size $3 \times 3 \times 192$
(stride: 1, pad: 1)

$$13 \times 13 \times 384 \text{ feature maps} \quad 384 \times 3 \times 3 \times 384 + 384 = 1,327,488$$

5th: Convolutional Layer: 256 kernels of size $3 \times 3 \times 192$
(stride: 1, pad: 1)

$$13 \times 13 \times 256 \text{ feature maps} \quad 384 \times 3 \times 3 \times 256 + 256 = 884,992$$

Then 3×3 Overlapping Max Pooling (stride: 2)

$$6 \times 6 \times 256 \text{ feature maps} \quad (6 \times 6 \times 256) \times 4096 + 4096 = 37,752,832$$

6th: Fully Connected (Dense) Layer of 4096 neurons

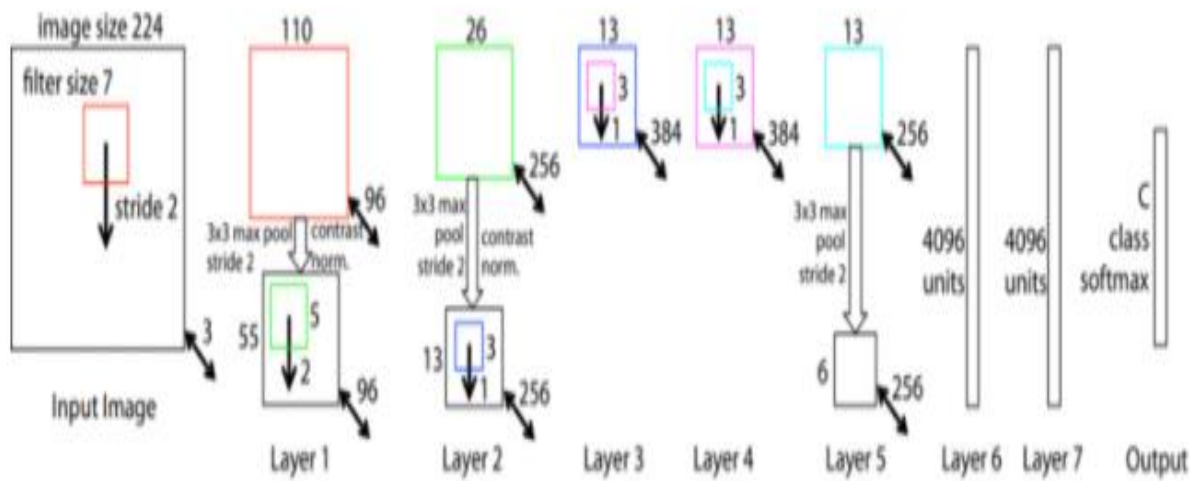
$$4096 \times 4096 + 4096 = 16,781,312$$

7th: Fully Connected (Dense) Layer of 4096 neurons

$$4096 \times 1000 + 1000 = 4,097,000$$

$$8th: Fully Connected (Dense) Layer of 1000 neurons (since there are 1000 classes)
Softmax is used for calculating the loss$$

ZFNet(2013)

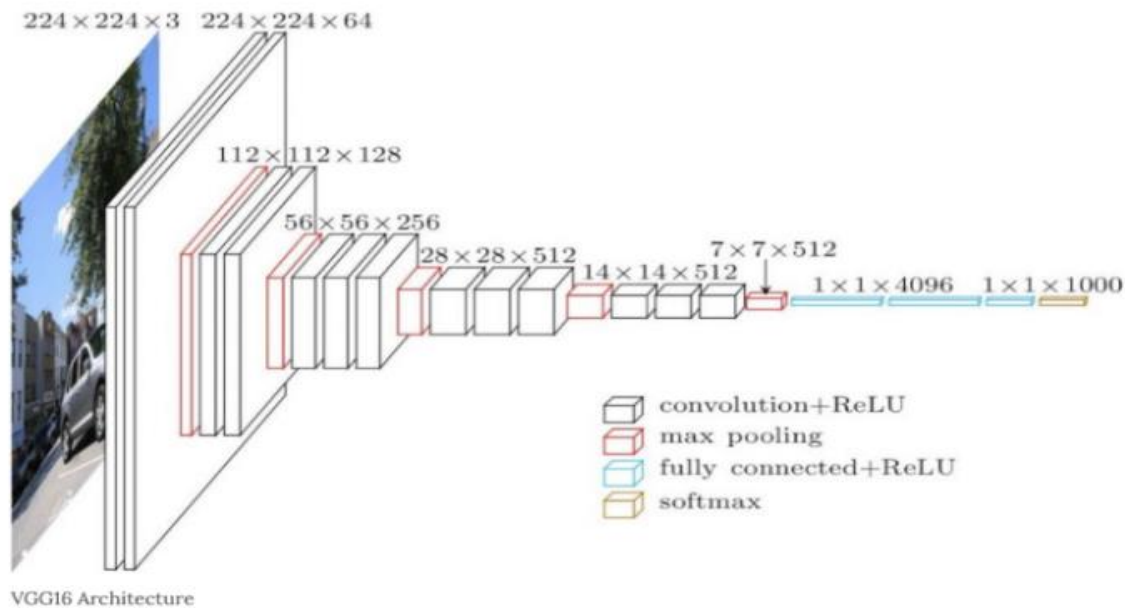


AlexNet과 매우 비슷함, 정확도 개선
입력 : (227x227x3)

표 8.1: AlexNet과 ZFNet의 비교

	AlexNet	ZFNet
볼륨: 연산:	224×224×3 합성곱 11×11(보폭 4)	224×224×3 합성곱 7×7(보폭 2), 최댓값 풀링
볼륨: 연산:	55×55×96 합성곱 5×5, MP	55×55×96 합성곱 5×5(보폭 2), 최댓값 풀링
볼륨: 연산:	27×27×256 합성곱 3×3, MP	13×13×256 합성곱 3×3
볼륨: 연산:	13×13×384 합성곱 3×3	13×13×512 합성곱 3×3
볼륨: 연산:	13×13×384 합성곱 3×3	13×13×1024 합성곱 3×3
볼륨: 연산:	13×13×256 최댓값 풀링, 완전 연결	13×13×512 최댓값 풀링, 완전 연결
FC6: 연산:	4096 완전 연결	4096 완전 연결
FC7: 연산:	4096 완전 연결	4096 완전 연결
FC8: 연산:	1000 소프트맥스	1000 소프트맥스

VGGNet(2014)



3x3 filter를 여러 개 쌓는 구조

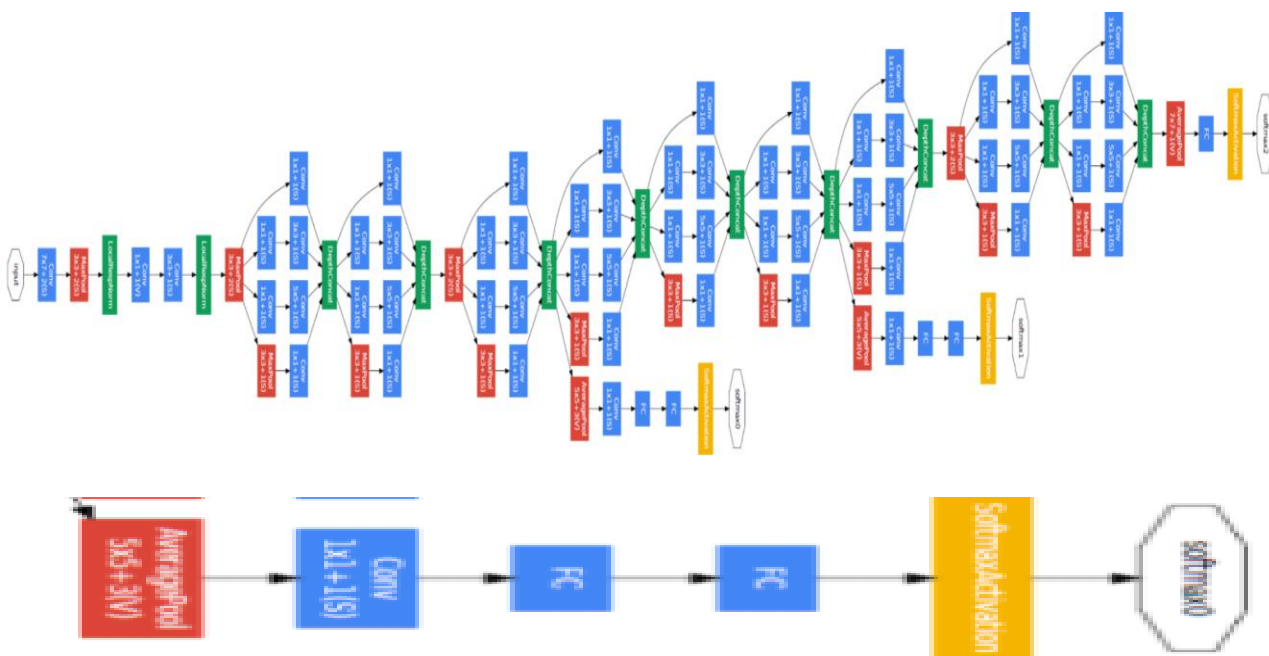
- 3x3 2개 = 5x5 1개, 3x3 3개 = 7x7 1개
- 파라미터 수 감소, 학습 속도 향상
- Non-linearity 증가로 인해 유용한 feature 추출 가능

#weight : (no bias)

INPUT: [224x224x3]	memory: 224*224*3=150K	weights: 0
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	weights: (3*3*3)+64 = 1,728
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	weights: (3*3*64)+64 = 36,864
POOL2: [112x112x64]	memory: 112*112*64=800K	weights: 0
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	weights: (3*3*64)+128 = 73,728
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	weights: (3*3*128)+128 = 147,456
POOL2: [56x56x128]	memory: 56*56*128=400K	weights: 0
CONV3-256: [56x56x256]	memory: 56*56*256=800K	weights: (3*3*128)+256 = 294,912
CONV3-256: [56x56x256]	memory: 56*56*256=800K	weights: (3*3*256)+256 = 589,824
CONV3-256: [56x56x256]	memory: 56*56*256=800K	weights: (3*3*256)+256 = 589,824
POOL2: [28x28x256]	memory: 28*28*256=200K	weights: 0
CONV3-512: [28x28x512]	memory: 28*28*512=400K	weights: (3*3*256)+512 = 1,179,648
CONV3-512: [28x28x512]	memory: 28*28*512=400K	weights: (3*3*512)+512 = 2,359,296
CONV3-512: [28x28x512]	memory: 28*28*512=400K	weights: (3*3*512)+512 = 2,359,296
POOL2: [14x14x512]	memory: 14*14*512=100K	weights: 0
CONV3-512: [14x14x512]	memory: 14*14*512=100K	weights: (3*3*512)+512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	weights: (3*3*512)+512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	weights: (3*3*512)+512 = 2,359,296
POOL2: [7x7x512]	memory: 7*7*512=25K	weights: 0
FC: [1x1x4096]	memory: 4096	weights: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]	memory: 4096	weights: 4096*4096 = 16,777,216
FC: [1x1x1000]	memory: 1000	weights: 4096*1000 = 4,096,000
TOTAL memory: 24M * 4 bytes ~ 93MB / image (only forward! ~*2 for bwd)		
TOTAL params: 138M parameters		

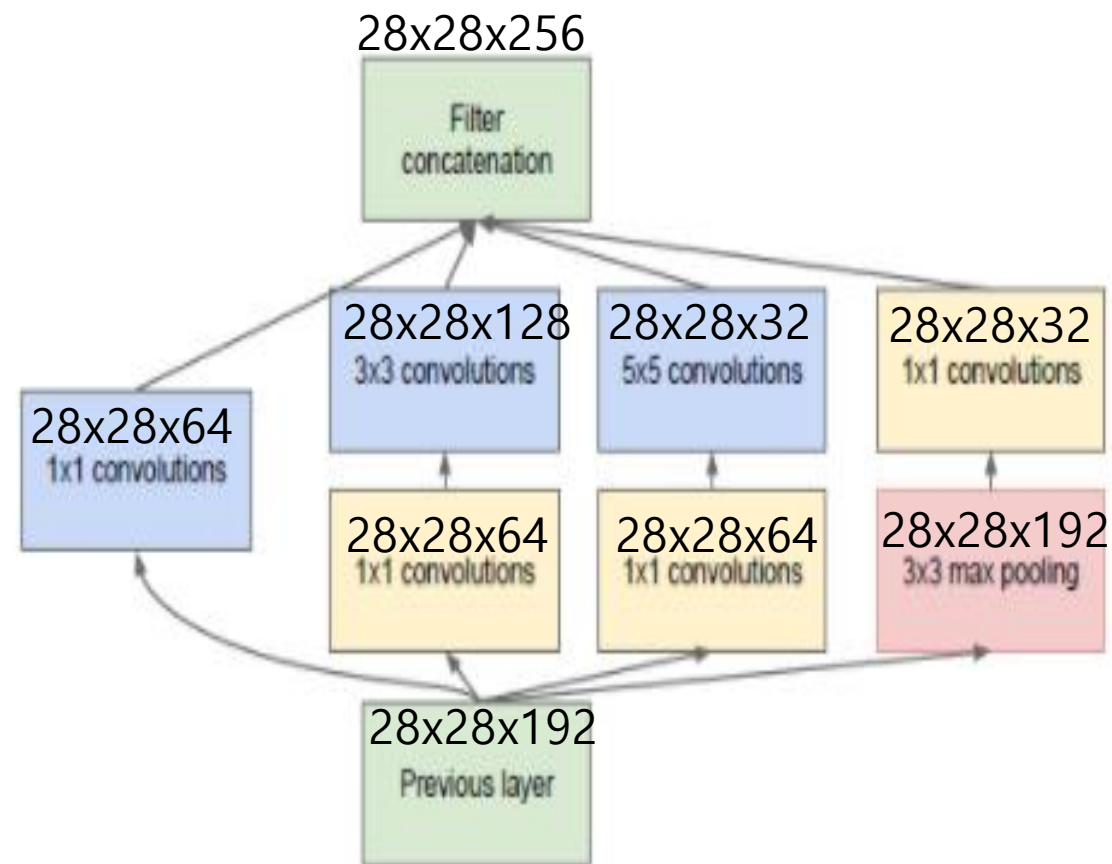
Memory 및 weight(VGG16)

GoogLeNet(2014)



총 22개의 layer, 입력: (224x224x3)

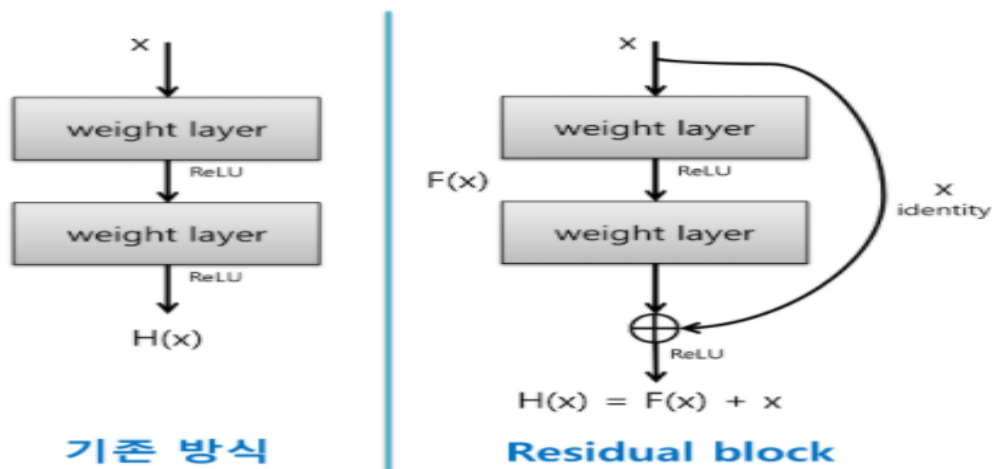
- Inception module 사용
 - 1x1 convolution layer를 통해 연산량 감소 효과
- Auxiliary classifier 사용
 - 기울기 소실 문제 해결을 위해 훈련 시에만 사용
 - Pooling->Convolution->Fc->Fc->Softmax 구성



inception (3a)		28x28x256	2	64	96	128	16	32	32	159K	128M
				1x1		3x3		5x5	pool		

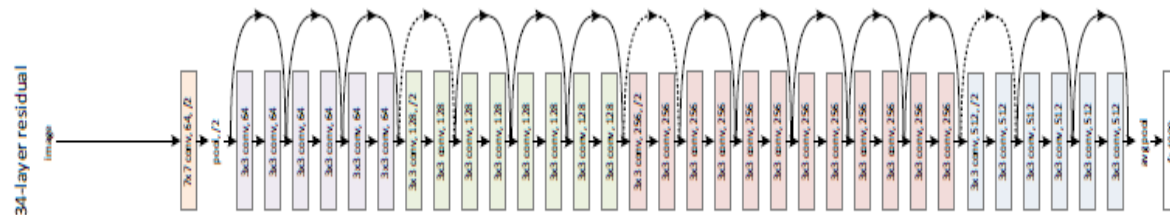
첫번째 Inception Module 연산

ResNet(2015)



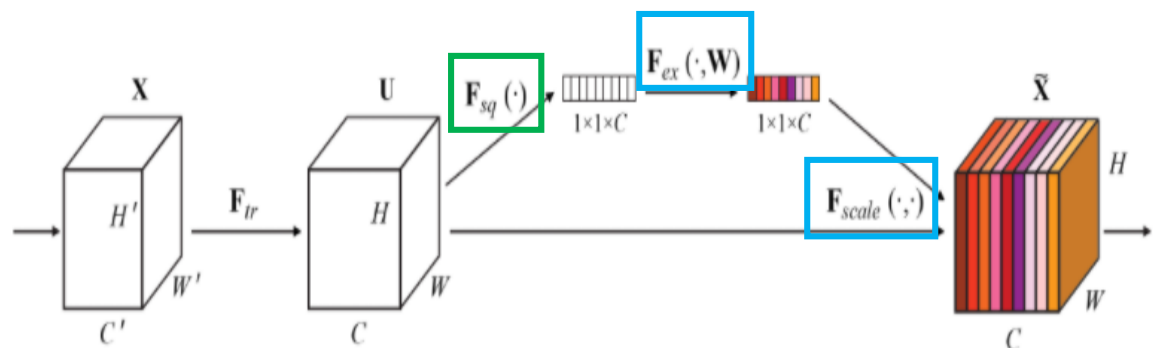
최대 152개 layer, 입력: (224x224x3)

- Residual Block
- "F(x)"를 최소화(0) 하는 것을 목적으로 한다.
- 즉, "H(x)-x"(=residual)를 최소화 하는 것이다.
- H(x) : layer를 통과한 출력, x : 입력
- 인접한 층(i와 i+1)만 연결하는 것이 아니라, i와 (i+r) 층 사이의 연결이 허용된다.

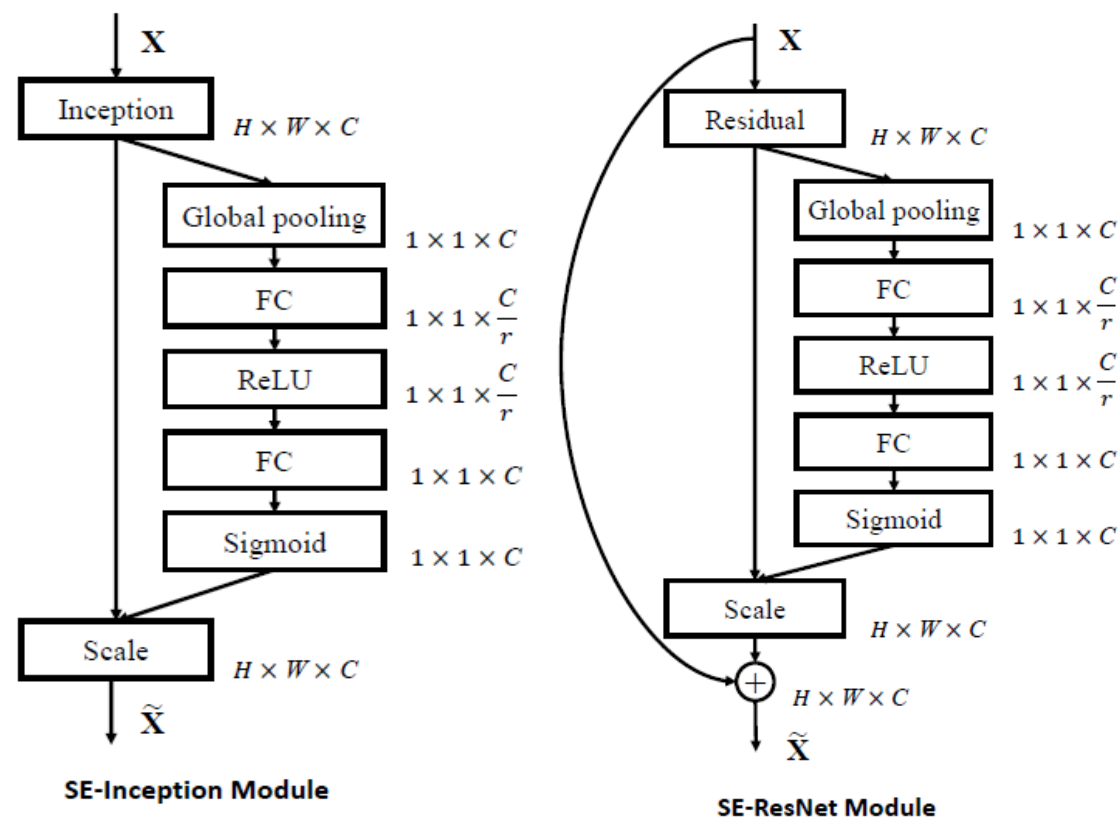


- 제 1 구간 (conv1) : 7*7 filter size, 64 filter num, 2 stride (output size : 112*112)
- 제 2 구간 (conv2_x) : 3*3 max pooling, 2 stride & 3개의 Residual Block (output size : 56*56)
 - > conv 2_1 : 3*3 filter size, 64 filter num
 - > conv 2_2 : 3*3 filter size, 64 filter num
- 제 3 구간 (conv3_x) : 4개의 Residual Block (output size : 28*28)
 - > conv 3_1 : 3*3 filter size, 128 filter num
 - > conv 3_2 : 3*3 filter size, 128 filter num
- 제 4 구간 (conv4_x) : 6개의 Residual Block (output size : 14*14)
 - > conv 4_1 : 3*3 filter size, 256 filter num
 - > conv 4_2 : 3*3 filter size, 256 filter num
- 제 5 구간 (conv5_x) : 3개의 Residual Block (output size : 7*7)
 - > conv 5_1 : 3*3 filter size, 512 filter num
 - > conv 5_2 : 3*3 filter size, 512 filter num
- 제 6 구간 (avg_pool) : average pooling, 1000-d Fully-Connectde, Softmax (output size : 1*1)

Squeeze-and-Excitation Net, SENet(2017)



- SE Block(fire module) : 컨볼루션을 통해 생성된 특성을 채널당 중요도를 고려하여 재보정(re-calibration)하는 것
- Squeeze(Fsq)
 - GlobalAveragePooling를 통해 특성맵을 1x1 사이즈로 변환한다.
 - 각 특성맵에 대한 전체 정보 요약
- Excitation(Fex, Fscale)
 - 각 채널의 상대적 중요도를 알아냄
 - FC -> Activation -> FC -> Activation을 통해 조절 (bottle-neck구조 : 은닉 층의 뉴런 수는 입력 층보다 작게, 출력 층의 뉴런 수는 입력 층과 동일)
 - 연산량 감소 및 일반화 도움을 주기 위함



모델 유연성 - 다른 모델과 결합