Image/Video BM 이해
-딥러닝-

김 승 환

swkim4610@inha.ac.kr

3. 딥러닝

3.0 기계학습 이해

3.1 DNN 관련 기술 발전사

3.1 CNN 이해

3.2 Fashion MNIST 실습

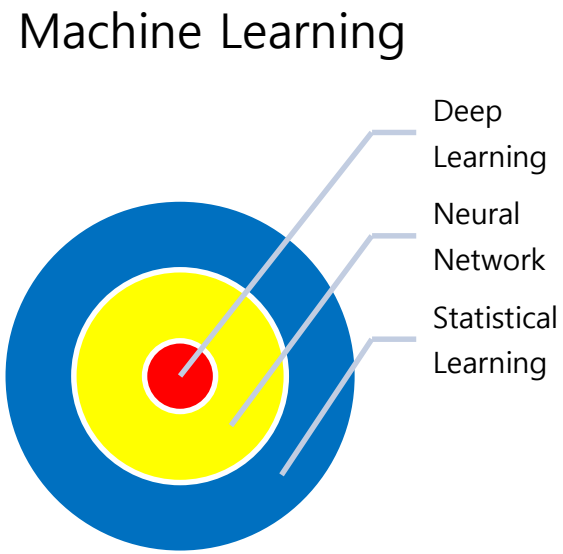
3.3 CNN 고급 기술

3.4 ImageNet Challenge

3.5 BM Ideation

3.0 기계학습 이해

- 2006년 제프리 힌튼은 손글씨 숫자를 인식할 수 있는 딥러닝 모형에 관한 논문을 발표
- 인간보다 더 정확한 결과(98%)를 기록하여 인간보다 컴퓨터가 더 잘할 수 있다는 가능성을 보임
- 이후, 고사양 컴퓨터의 성능과 빅데이터가 결합되어 다른 머신러닝 기법 보다 좋은 결과를 나타내는 여러 사례가 나타남
- 지금 딥러닝은 전세계에서 가장 핫(Hot)한 분야임
- 휴대폰 지문인식, 음성인식, 인공지능 비서에서 자율주행 자동차까지 머신러닝의 분야는 엄청난 발전을 하고 있음
- Machine Learning: 명시적 프로그래밍 없이 컴퓨터 스스로 학습할 수 있는 능력을 갖추게 하는 분야



3.0 기계학습 이해

- Supervised Learning:

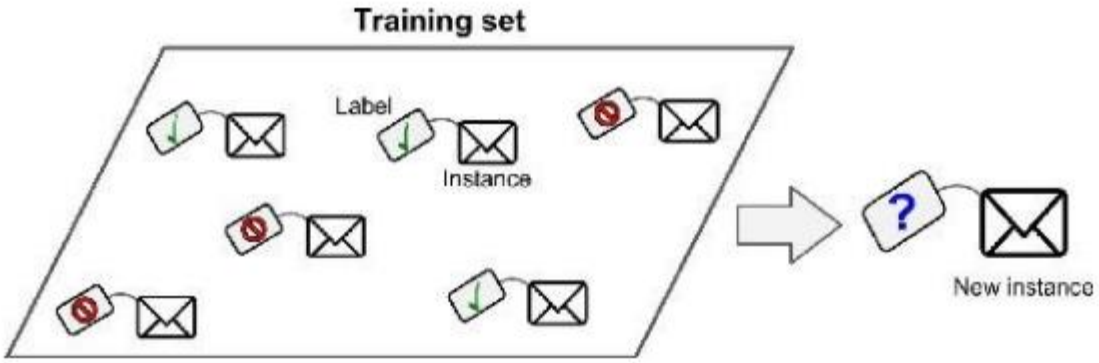


Figure 1-5. A labeled training set for supervised learning (e.g., spam classification)

A typical supervised learning task is *classification*. The spam filter is a good example of this: it is trained with many example emails along with their *class* (spam or ham), and it must learn how to classify new emails.

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks[2]

3.0 기계학습 이해

- Un-supervised Learning:

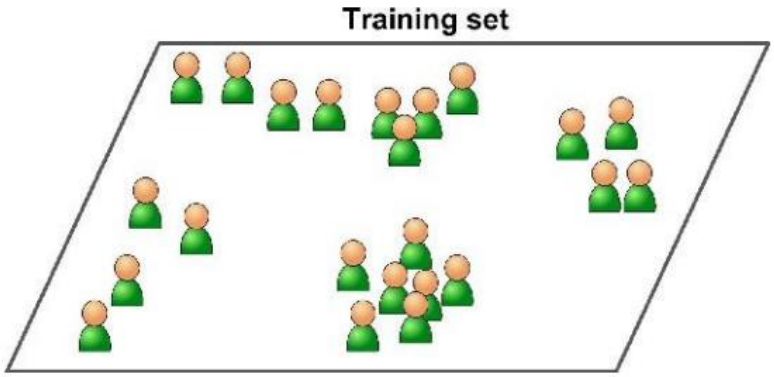


Figure 1-7. An unlabeled training set for unsupervised learning

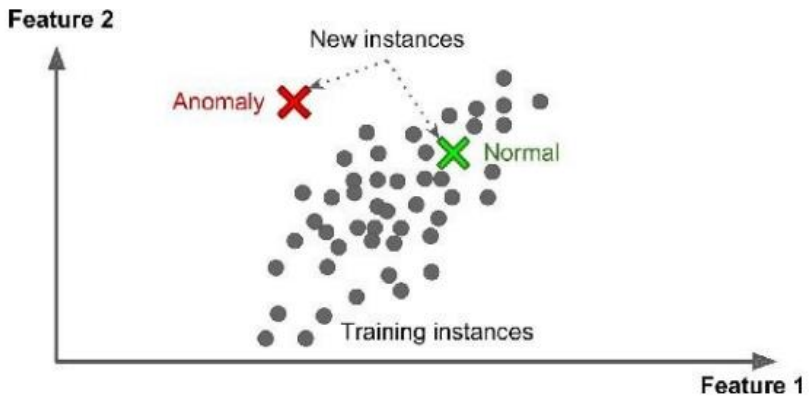


Figure 1-10. Anomaly detection

- Clustering
  - k-Means
- Visualization and dimensionality reduction
  - Principal Component Analysis (PCA)
- Association rule learning
  - Apriori

3.0 기계학습 이해

- Reinforcement Learning:

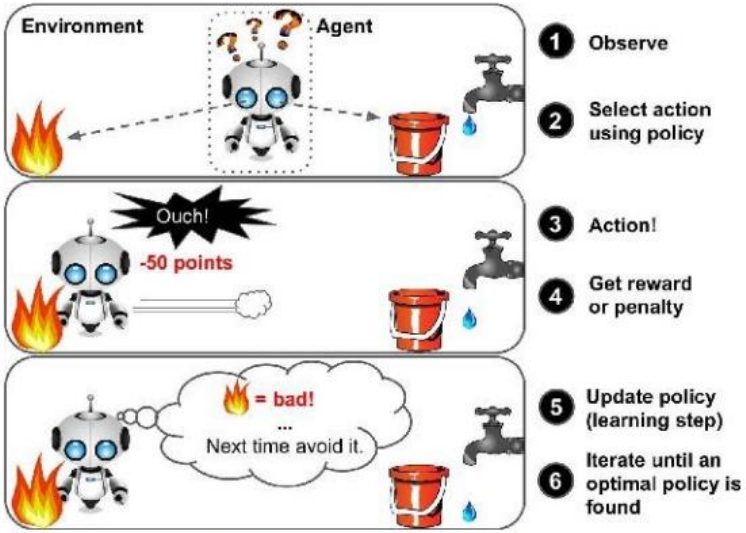


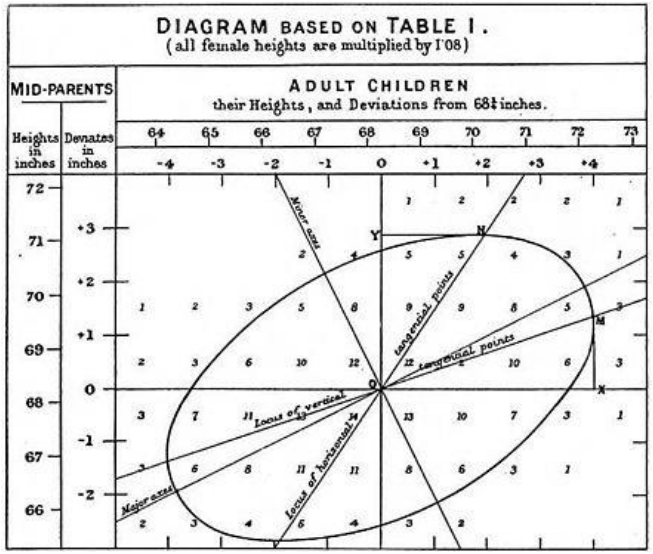
Figure 1-12. Reinforcement Learning

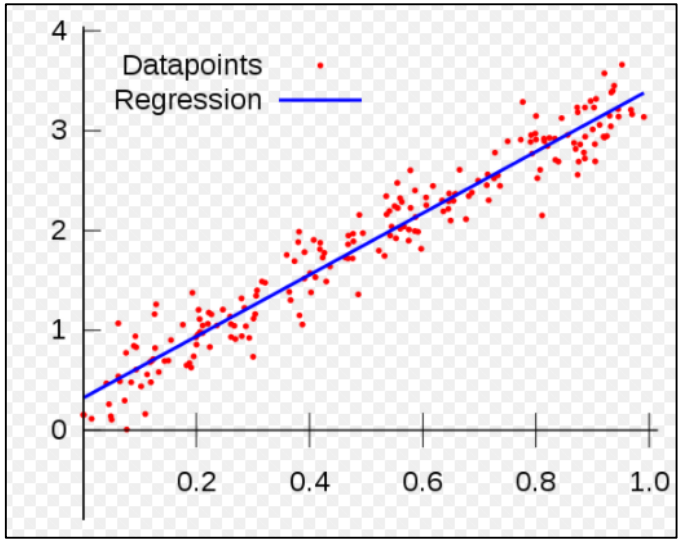
현재 policy에 의해 의사결정 하고 실행한다.
실행 결과에 따라 벌점을 받을 수 있다.
벌점을 최소화하는 방향으로 의사결정 policy를 수정한다.
이 과정을 반복한다.

https://youtu.be/Aut32pR5PQA

3.1 DNN 관련 기술 발전사

Galton의 Regression





[Francis Galton's](#) 1875

"[regression toward the mean](#)"

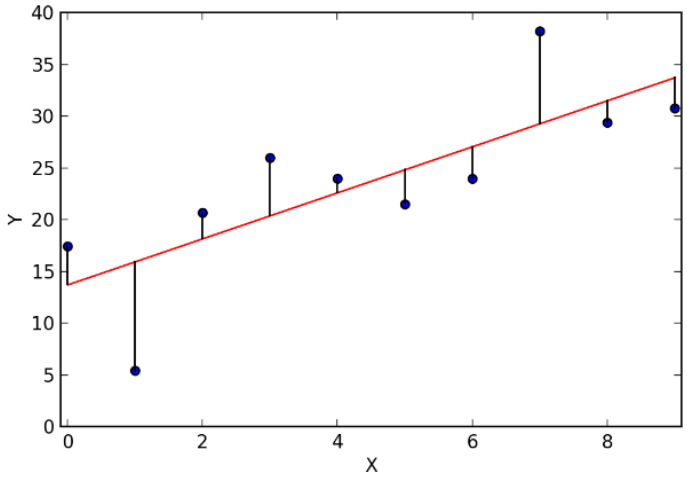
$$Y = \alpha + \beta x + \epsilon$$

$$\hat{\beta} = \frac{\sum x_i Y_i - n\bar{x}\bar{Y}}{\sum {x_i}^2 - n\bar{x}^2} = \frac{\sum (x_i - \bar{x})(Y_i - \bar{Y})}{\sum (x_i - \bar{x})^2}, \qquad \hat{\alpha} = \bar{Y} - \hat{\beta}\bar{x}$$

3.1 DNN 관련 기술 발전사

Least Square Method

- 데이터를 가장 잘 설명하는 직선의 방정식은 아래와 같이 SSE(Error sums of square)를 최소화하는 α, β 를 구하는 것임
- SSE는 α, β 에 대한 2차 방정식으로 기울기가 0이 되는 α, β를 구함(최소제곱법)
- 이 경우, 운 좋게 정확한 해를 구할 수 있음



$$SSE = \sum(\varepsilon_i{}^2) = \sum(Y_i - \alpha - \beta x_i)^2$$

- Least Square Estimation for α, β

$$\frac{\partial SSE}{\partial \alpha} = 2\sum(Y_i - \alpha - \beta x_i)(-1) = 0 \cdots\cdots\cdots ①$$

$$\frac{\partial SSE}{\partial \beta} = 2\sum(Y_i - \alpha - \beta x_i)(-x_i) = 0 \cdots\cdots\cdots ②$$

①× $\bar{x}$ 를 하면,

$$\alpha\sum x_i + \beta\bar{x}\sum x_i = n\bar{x}\bar{Y} \cdots\cdots\cdots\cdots ③$$

$$\alpha\sum x_i + \beta\sum x_i{}^2 = \sum x_i Y_i \cdots\cdots\cdots ④$$

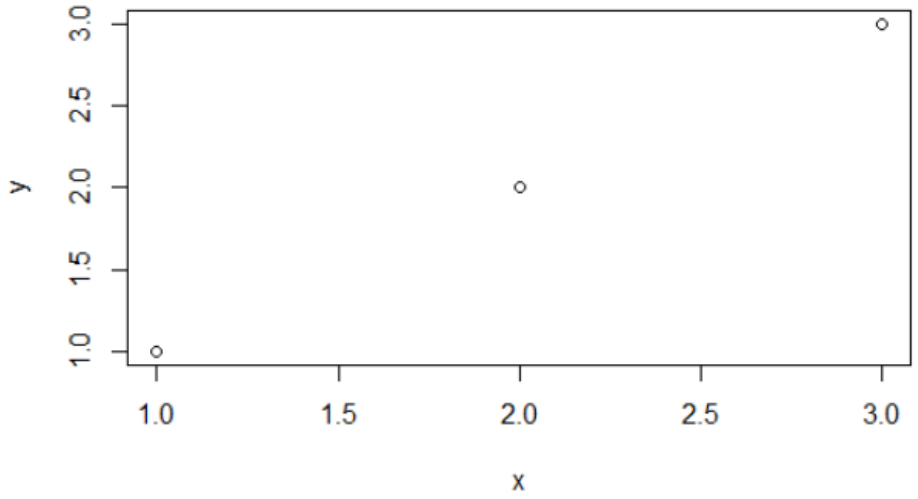
④-③을 하면,

$$\hat{\beta} = \frac{\sum x_i Y_i - n\bar{x}\bar{Y}}{\sum x_i{}^2 - n\bar{x}^2} = \frac{\sum(x_i - \bar{x})(Y_i - \bar{Y})}{\sum(x_i - \bar{x})^2}, \qquad \hat{\alpha} = \bar{Y} - \hat{\beta}\bar{x}$$

3.1 DNN 관련 기술 발전사

Steepest Descent Method

- 아래와 같이 x=[1,2,3], y = [1,2,3] 의 경우, 세 점을 잘 지나가는 직선의 방정식을 구해보자.
- 최소제곱법(Least Square method), 최대경사법(Steepest descend method)으로 각각 실습해 보세요~

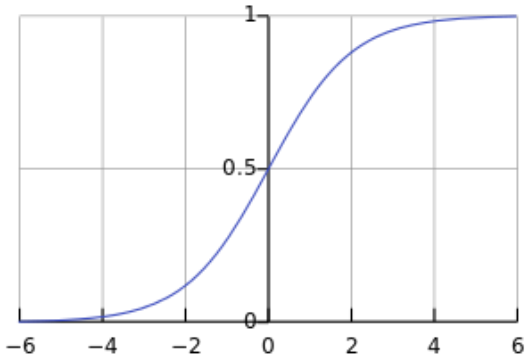


최대경사법 공식: $\theta = \theta - \lambda \frac{\partial Cost(\theta)}{\partial \theta}$

3.1 DNN 관련 기술 발전사

Cox의 Logistic Regression

- 회귀모형에서 종속변수 Y=0,1 과 같이 Binary Factor 변수일 경우 문제가 발생
- 일반적인 회귀모형 $Y = \alpha + \beta X + \epsilon$ 으로 표현할 경우, 좌변의 범위는 (0, 1) 이고 우변은 $(-\infty, \infty)$ 인 문제가 발생. 이 문제를 해결하기 위해 로지스틱 함수를 사용함
- 로지스틱 함수는 0~1 사이 값을 가지는 함수로 Sigma 모양과 비슷하여 Sigmoid라고 함
- Logistic 함수를 이용하여 회귀분석을 수행하는 것을 Logistic Regression(Cox, 1958)이라 함
- 불행하게도 이 문제에서 $\hat{\beta}$를 수학적으로 계산할 수 없어 최대경사법으로 계산하여야 함



$$Y = \frac{1}{1 + e^{-x}}$$

$$Y = \frac{1}{1 + \exp(-(\beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k + \epsilon))} \quad \hat{Y} = H(X) = \frac{1}{1 + \exp\left(-\left(\widehat{\beta_0} + \widehat{\beta_1} x_1 + \cdots + \widehat{\beta_k} x_k\right)\right)}$$

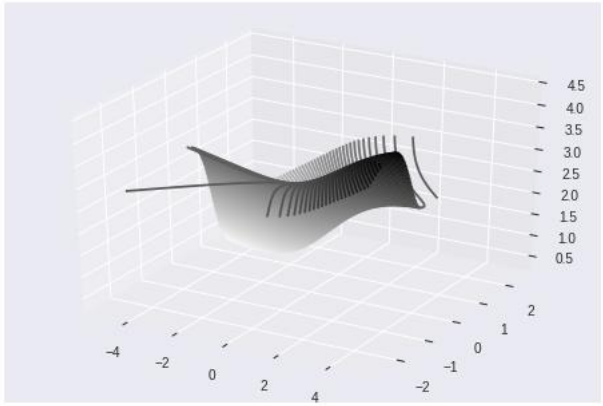
3.1 DNN 관련 기술 발전사

Cost 함수의 Convexity

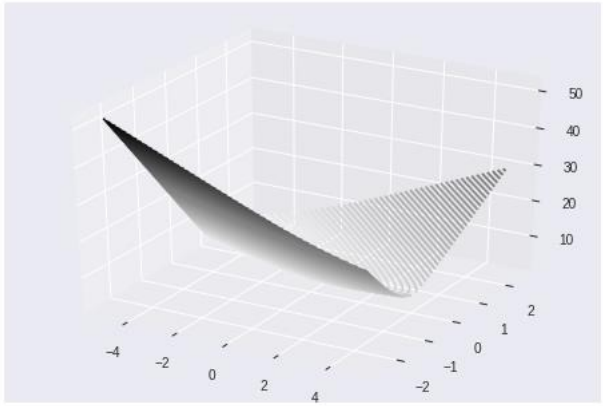
- 최대경사법으로 해를 구하기 위해서는 Cost 함수가 convexity(볼록 함수 성질)를 만족해야 함
- 로지스틱 회귀모형의 경우, SSE가 Convex 하지 않아 대안으로 아래의 Cross Entropy 함수를 사용함
- 이 함수는 y = 1 일 때, H(X) 가 커지면 함수 값이 작아지고, H(X)가 작아지면 무한대가 됨
  또한, y = 0 일 때, H(X) 가 작아지면 함수 값이 작아지고, H(X)가 커지면 무한대가 됨

$$Cost(W) = \begin{cases} \sum -log(H(X)), & y = 1 \\ \sum -log(1 - H(X)), & y = 0 \end{cases} \Rightarrow -\sum (y_i log H(X)_i + (1 - y_i)(1 - \log H(X)_i))$$

convexity.py



SSE 함수



Cross Entropy 함수

3.1 DNN 관련 기술 발전사

Cross Entropy 함수

Cross Entropy 함수는 Logistic Regression의 Maximum Likelihood 함수다.

최대가능도 함수(Maximum Likelihood function)에 대해 알아보자.

$Y_i = \frac{1}{1+e^{(-b-wx_i+\epsilon)}}$ 의 식에서 Y_i 는 0 혹은 1의 값을 갖는 확률변수다.

그러므로 $Y_i|X_i \sim Bernoulli(p_i)$ 이다. 여기서, $p_i = \Pr(Y_i = 1)$ 이고 확률질량함수(P.M.F.)는 아래와 같다.

$$f(y_i) = P(Y = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

만약, $Y = [y_1, y_2, \cdots, y_n]$ 이 관측되었다면 이렇게 관측될 확률은 아래와 같다.

$$(p_1^{y_1}(1-p_1)^{1-y_1}) \times (p_2^{y_2}(1-p_2)^{1-y_2}) \times \cdots \times (p_n^{y_n}(1-p_n)^{1-y_n}) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$$

위 수식에서 y_i는 주어진 값이고 미지수는 오직 p_i 이고, p_i는 b, w 의 함수다.

이 함수를 최대화 하는 p_i 가 가장 관측 값을 잘 설명하는 추정 값이라고 생각하는 것이 MLE 추정(Maximum Likelihood Estimation)이다. 이 식에 "-1"을 곱한 식이 Cross Entropy 함수다.

$$\ln L(p_i) = \sum_{i=1}^{n} y_i \ln p_i + (1-y_i)\ln(1-p_i), \quad p_i = \frac{1}{1+e^{-\hat{b}-\hat{w}x_i}}$$

3.1 DNN 관련 기술 발전사

Cross Entropy 함수 최소 해 구하기

Logistic Regression Gradient Descent Algorithm

$$Cost(w,b) = -\sum \left[y_i log(S(wx_i + b)) + (1 - y_i) log(1 - S(wx_i + b)) \right]$$

$$\frac{\partial}{\partial b} y_i log(S(wx_i + b)) = y_i (1 - S(wx_i + b))$$
$$\frac{\partial}{\partial b} (1 - y_i) log(1 - S(wx_i + b)) = -(1 - y_i) S(wx_i + b))$$
$$\frac{\partial}{\partial w} y_i log(S(wx_i + b)) = y_i (1 - S(wx_i + b)) x_i$$
$$\frac{\partial}{\partial w} (1 - y_i) log(1 - S(wx_i + b)) = -(1 - y_i) S(wx_i + b)) x_i$$

$$\frac{\partial}{\partial b} Cost(w,b) = -\sum \left[y_i - S(wx_i + b) \right]$$
$$\frac{\partial}{\partial w} Cost(w,b) = -\sum \left[y_i - S(wx_i + b) \right] x_i$$

$$b = b - \lambda \frac{\partial Cost(b,w)}{\partial b}, w = w - \lambda \frac{\partial Cost(b,w)}{\partial w}$$

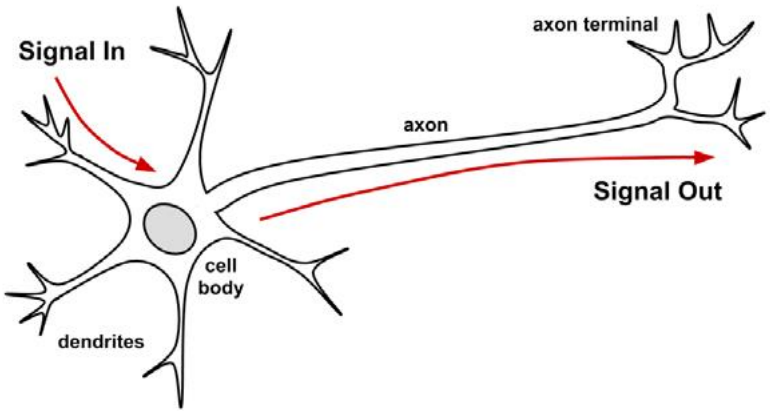
logistic_gradient.py

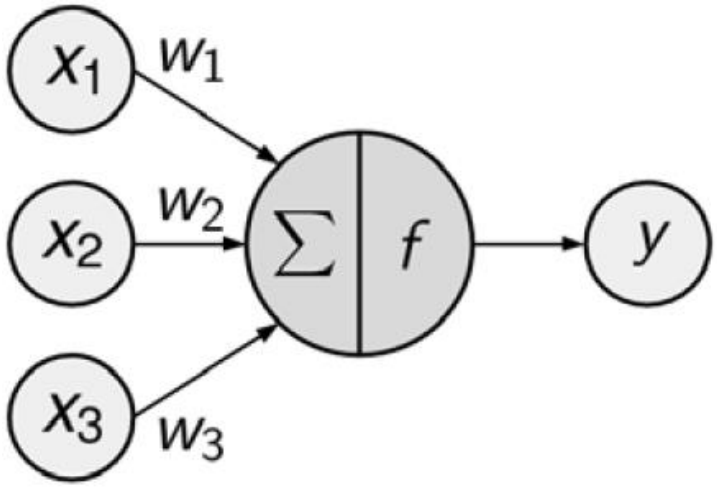
3.1 DNN 관련 기술 발전사

Artificial Neural Network

신경망 모형(Warren McCulloch and Walter Pitts, 1943)은 인간의 뇌를 수학적 모형으로 표현하여 인간처럼 판단을 수행하고자 하는 아이디어로 출발하였다.

이 아이디어는 $f(\sum w_i x_i + b)$ 의 값이 y 값이 되도록 미지수 w, b 값을 구하고자 하는 것이다.



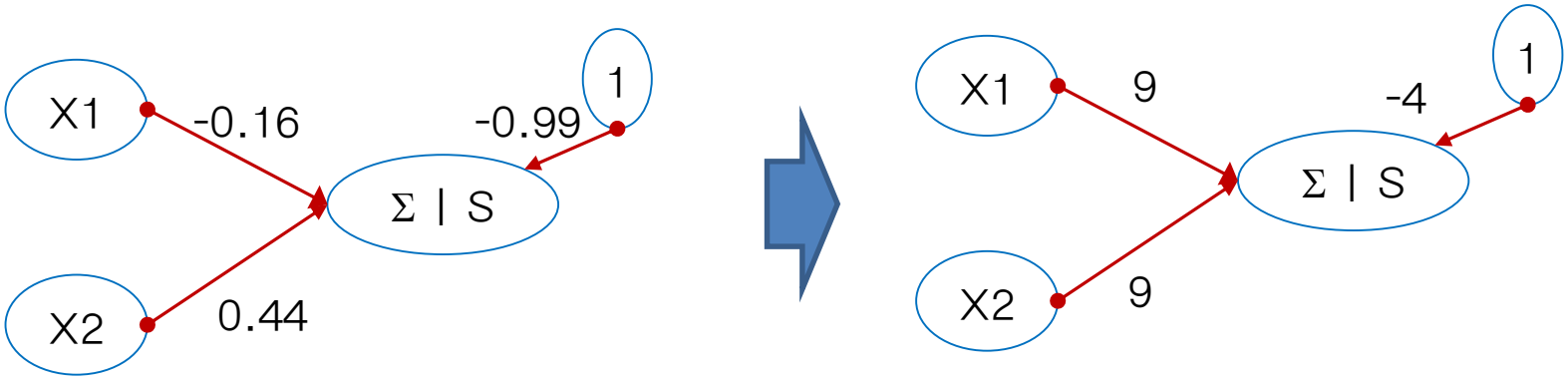


$$Y = S(w_1x_1 + w_2x_2 + w_3x_3 + b)$$

$$= \frac{1}{1+exp(-(w_1x_1+w_2x_2+w_3x_3+b))}$$

3.1 DNN 관련 기술 발전사

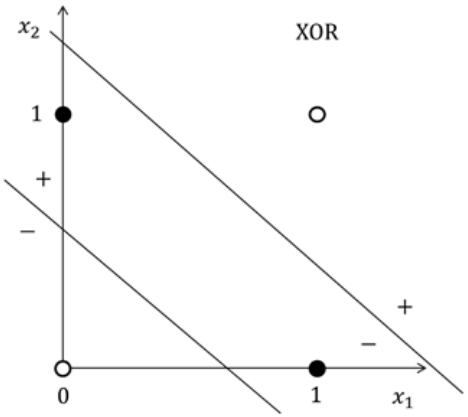
신경망 모형을 이용한 OR 연산 예



x1	x2	sum	Y	T	error
0	0	0*-0.16+0*0.44-0.99=-0.99	1/(1+exp(0.99)) = 0.27	0	0.27
0	1	0*-0.16+1*0.44-0.99=-0.55	1/(1+exp(0.55)) = 0.36	1	-0.64
1	0	1*-0.16+0*0.44-0.99=-1.15	1/(1+exp(1.15)) = 0.24	1	-0.76
1	1	1*-0.16+1*0.44-0.99=-0.71	1/(1+exp(0.71)) = 0.33	1	-0.67

3.1 DNN 관련 기술 발전사

xor 문제는 쉽게 풀리지 않는다. 이 문제는 아래의 그림처럼 h1, h2 두개 선을 사용하여 해결할 수 있다.
이렇게 히든 레이어가 등장하게 되었는데 문제는 w, b 값을 구하는 것임

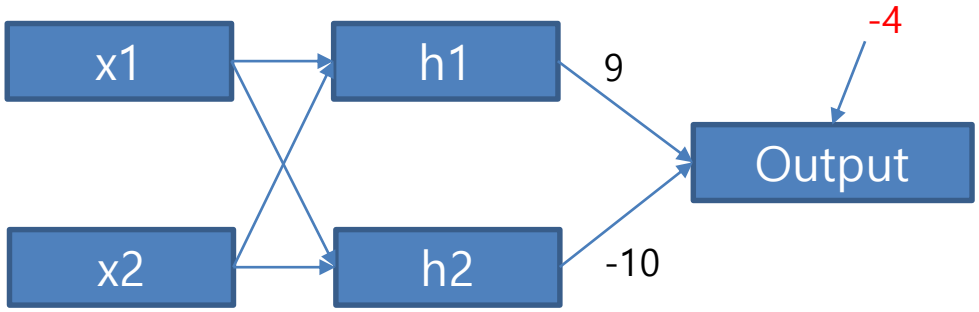


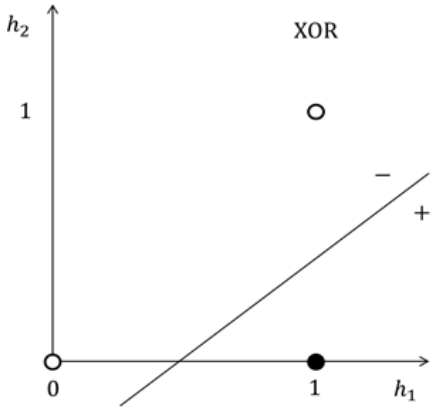
x_1	x_2	h_1	h_2	y
0	0	0(-)	0(-)	0
0	1	1(+)	0(-)	1
1	0	1(+)	0(-)	1
1	1	1(+)	1(+)	0

$$Y = S(w_1h_1 + w_2h_2 + b) > 0.5$$

$$(w_1h_1 + w_2h_2 + b) > 0$$

or and

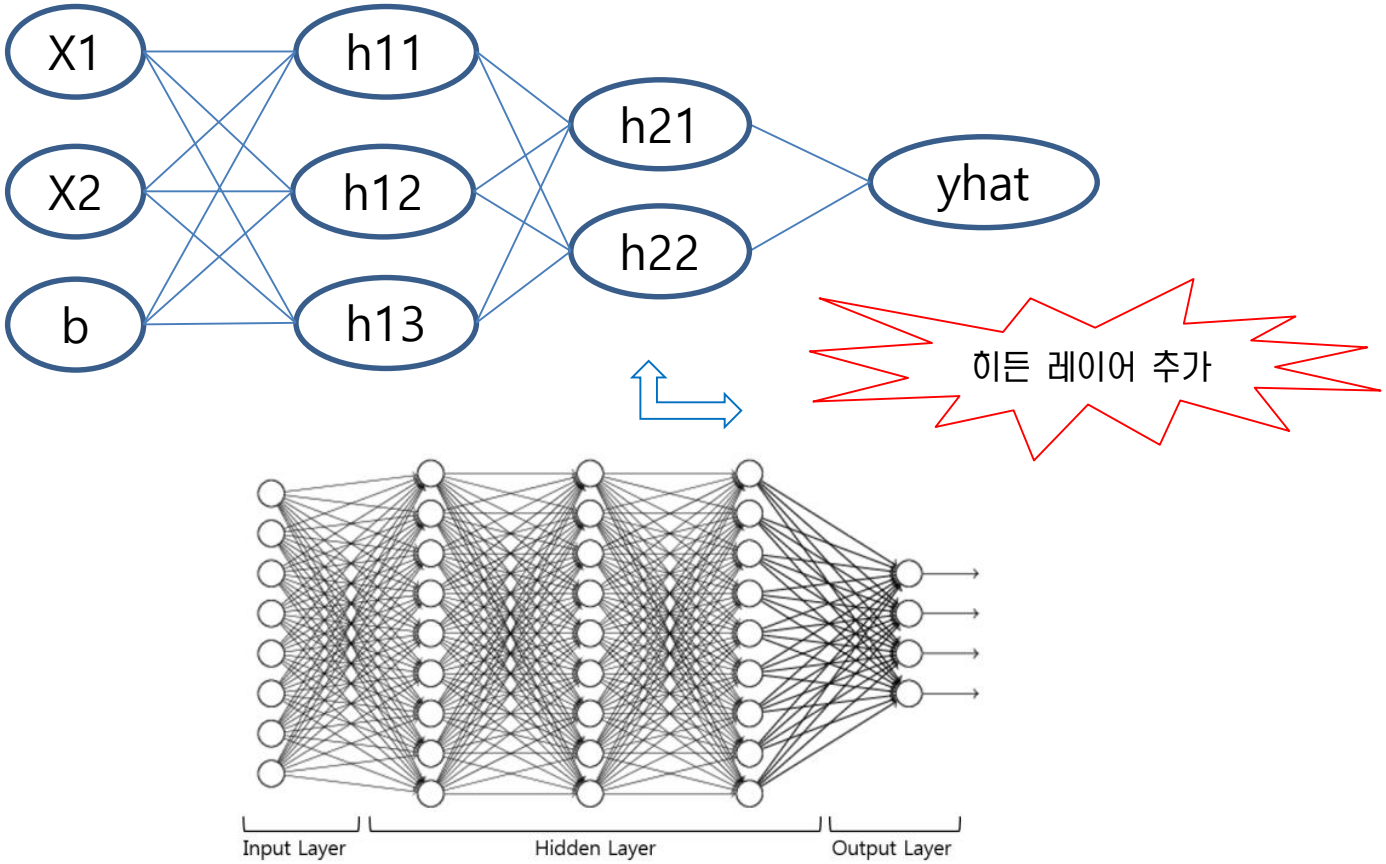




3.1 DNN 관련 기술 발전사

Deep Neural Network

Deep Learning 모형은 Neural Network 모형에서 Hidden Layer를 2개 이상으로 확장한 모형이므로 간단하게 신경망 모형에서 딥러닝 모형으로 확장 가능함



엘리웁 숏: https://drive.google.com/open?id=1JBJvHBBtRr7Cz5qCjBebT35TB9x0rOtB

3.1 DNN 관련 기술 발전사

Backpropagation 원리

$h = g(w_1 x), y = g(w_2 h), t : \text{Target Value}$ 이고 $g(x) = \frac{1}{1+exp(-x)}$ 이라고 가정하자.

$$x \xrightarrow{w_1} h \xrightarrow{w_2} y, Cost = \frac{1}{2}(y-t)^2$$

우리의 목표는 Cost 함수를 최소화하는 가중값을 구하는 것이다. 가중값은 아래와 같이 Gradient Descent Method를 반복적으로 사용하여 구한다.

$$w_{k+1} = w_k - \lambda \frac{\partial Cost(w)}{\partial w}$$

먼저 출력 값을 결정하는 w2 부터 풀어보자.

$$w_2 = w_2 - \lambda \frac{\partial Cost(w_2)}{\partial w_2}$$

$$\begin{aligned} \frac{\partial}{\partial w_2} Cost(w_2) &= (y-t)\frac{\partial}{\partial w_2} y = (y-t)\frac{\partial}{\partial w_2} g(h \cdot \mathrm{w}_2) \\ &= (y-t)g(h \cdot \mathrm{w}_2)\big(1 - g(h \cdot \mathrm{w}_2)\big)h = (y-t)y(1-y) \cdot h \\ &= \delta_y h \end{aligned}$$

여기서, $\delta_y = (y-t)y(1-y)$ 로 간단히 줄여쓰자. 최종적으로 가중값은 아래와 같이 업데이트할 수 있다.

$$w_2 = w_2 - \lambda \delta_y h$$

3.1 DNN 관련 기술 발전사

다음은 w1 차례이다.

$$w_1 = w_1 - \lambda \frac{\partial Cost(w_1)}{\partial w_1}$$

$$\frac{\partial}{\partial w_1} Cost(w_1) = (y-t)\frac{\partial}{\partial w_1} y = (y-t)\frac{\partial}{\partial w_1} g(h \cdot \mathrm{w}_2)$$

$$= (y-t)g(h \cdot \mathrm{w}_2)\big(1 - g(h \cdot \mathrm{w}_2)\big) w_2{}' \frac{\partial}{\partial w_1} h$$

$$= (y-t)y(1-y)w_2{}' \frac{\partial}{\partial w_1} g(x \cdot \mathrm{w}_1)$$

$$= (y-t)y(1-y)w_2{}'\, g(x \cdot \mathrm{w}_1)\big(1 - g(x \cdot \mathrm{w}_1)\big) x$$

$$= (y-t)y(1-y)w_2{}'\, h(1-h)x$$

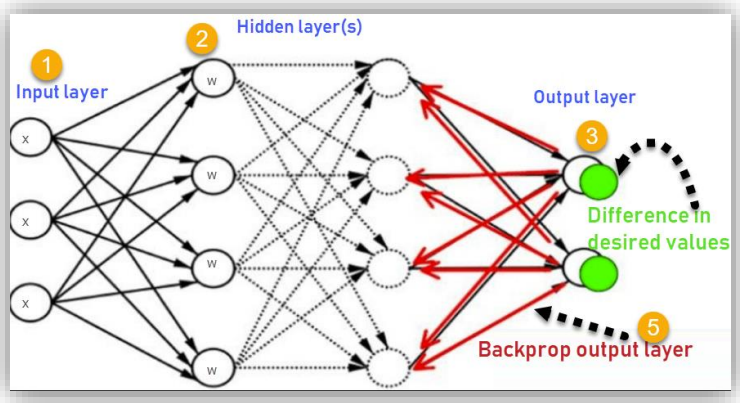
$$= \delta_h x$$

여기서, $\delta_h = (y-t)y(1-y)w_2{}'\, h(1-h)$ 로 간단히 줄여 쓰자.

$\delta_h = \delta_y w_2{}'\, h(1-h)$ 로 앞에서 구한 결과를 이용하여 재 표현이 가능하다.

3.1 DNN 관련 기술 발전사

Backpropagation



$$x \xrightarrow{w_1} h_1 \xrightarrow{w_2} h_2 \xrightarrow{w_3} h_3 \xrightarrow{w_4} y$$

$$\delta_1 \leftarrow \delta_2 \leftarrow \delta_3 \leftarrow \delta_y$$

Backpropagation

$$\delta_y = (y-t)y(1-y)$$

$$\delta_3 = \delta_y \cdot t(w_4) \cdot h_3(1-h_3)$$

$$\delta_2 = \delta_3 \cdot t(w_3) \cdot h_2(1-h_2)$$

$$\delta_1 = \delta_2 \cdot t(w_2) \cdot h_1(1-h_1)$$

$$w_4 = w_4 - t(h_3) \cdot \lambda \cdot \delta_y$$

$$w_3 = w_3 - t(h_2) \cdot \lambda \cdot \delta_3$$

$$w_2 = w_2 - t(h_1) \cdot \lambda \cdot \delta_2$$

$$w_1 = w_1 - t(x) \cdot \lambda \cdot \delta_1$$

3.1 DNN 관련 기술 발전사

Backpropagation 구현 XOR_Backpropagation.py

```
import numpy as np

def Sigmoid(x):
    return 1./(1. + np.exp(-x))

lamda = 1
x = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])                    # x: 4*2
t = np.array([[0],[1],[1],[0]])                                   # t: 4*1
w1 = np.array([[-0.5, 0.5, 0.5], [-0.5, 0.5, -0.5]])              # w1: 2*3
w2 = np.array([[-0.5], [0.5], [-0.5]])                            # w2: 3*1
b1 = np.array([0, 0, 0])

for i in range(100):
    h = Sigmoid(np.dot(x, w1) + b1)                               # h: 4*3
    y = Sigmoid(np.dot(h, w2))                                    # y: 4*1
    delta_y = np.multiply(y - t, np.multiply(y, (1 - y)))         # delta_y: 4*1
    delta_h = delta_y * np.multiply(w2.T, np.multiply(h, (1 - h))) # delta_h: 4*3
    w2 = w2 - np.dot(h.T, lamda * delta_y)                        #
    w1 = w1 - np.dot(x.T, lamda * delta_h)
    b1 = b1 - lamda * delta_h
print(y)
```

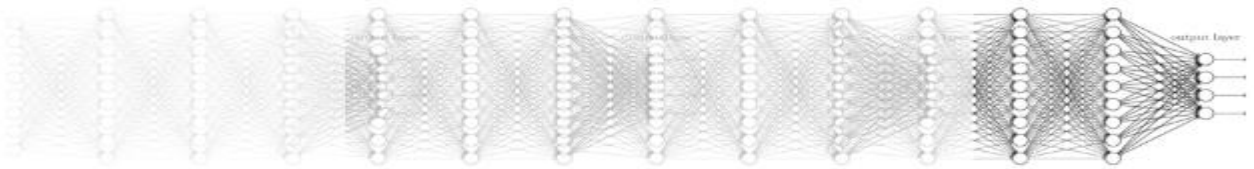
3.1 DNN 관련 기술 발전사

Vanishing Gradient와 ReLU 함수

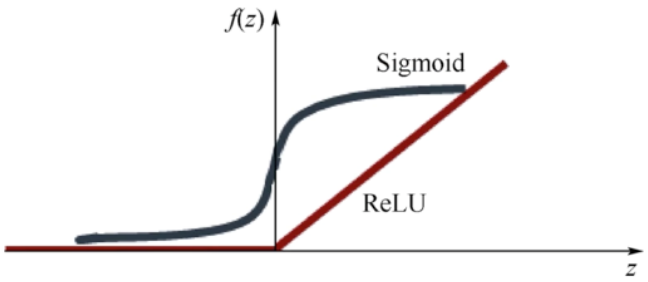
Deep Learning 모형은 Vanishing gradient problem과 Overfitting의 문제에 직면함

Vanishing gradient (NN winter2: 1986-2006)

$$W_i = W_i + t(h_{i-1})\lambda\delta_i, \delta_i = \delta_{i+1}t(W_{i+1})h_i(1-h_i)$$



Sigmoid!



3.1 DNN 관련 기술 발전사

Sigmoid 함수

- 활성함수는 자극 x에 대한 반응의 크기를 나타내는 함수로 자극이 커지면 단조 증가하는 형태를 가진다.
- 또한, 입력을 출력으로 전환하는 과정에서 비선형적 연결을 만들어 함수관계를 정교하게 튜닝한다.
- 단점으로 Saturation 현상이 발생하고 완만한 구간에서 수렴속도가 느려 진다.
- exp 계산이 포함되어 계산속도가 느리다.

$$f(x) = \frac{1}{1+exp(-x)}$$

$$f(x)' = -\frac{(1+exp(-x))'}{(1+exp(-x))^2} = \frac{exp(-x)}{(1+exp(-x))^2} = \frac{1}{1+exp(-x)} \cdot \frac{exp(-x)}{1+exp(-x)}$$

$$= f(x)\frac{1+exp(-x)-1}{1+exp(-x)} = f(x)\left(1 - \frac{1}{1+exp(-x)}\right) = f(x)(1-f(x))$$

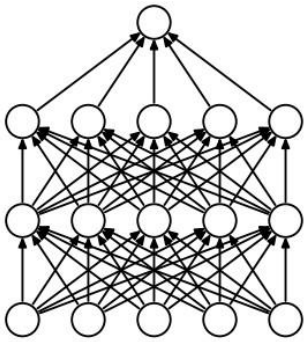
ReLU 함수

- 비선형적 특성을 가지지만 미분 값이 0, 1로 간단하게 계산된다.
- Gradient Vanishing의 가능성이 작다.
- 히든 레이어 구간에서 시그모이드 함수 대신 사용할 수 있다.
- 단, 최종 레이어의 활성함수는 반드시 시그모이드 함수를 사용한다.
- Leakly ReLU, eLU 등 변종 함수들이 많이 제안되었다.

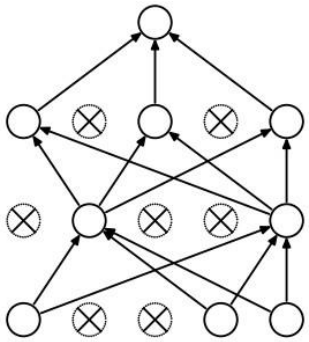
$$RELU(x) = \begin{cases} 0 \; if \; x < 0 \\ x \; if \; x >= 0 \end{cases}$$

3.1 DNN 관련 기술 발전사

2006년 이후, 오버피팅을 방지, 해 수렴속도를 개선 분야에서 드롭아웃, 배치정규화, ADAM Optimizer 등 많은 연구 성과에 의해 비약적인 발전을 하게 됨



(a) Standard Neural Net



(b) After applying dropout.

$$BN(x_i) = \gamma \cdot \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

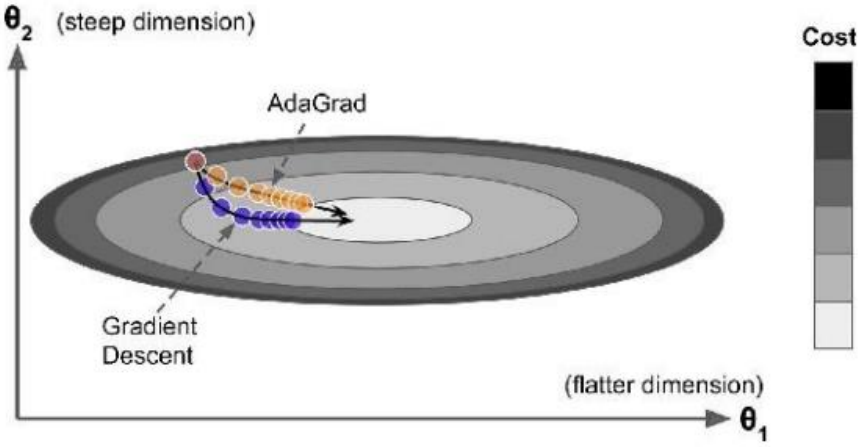
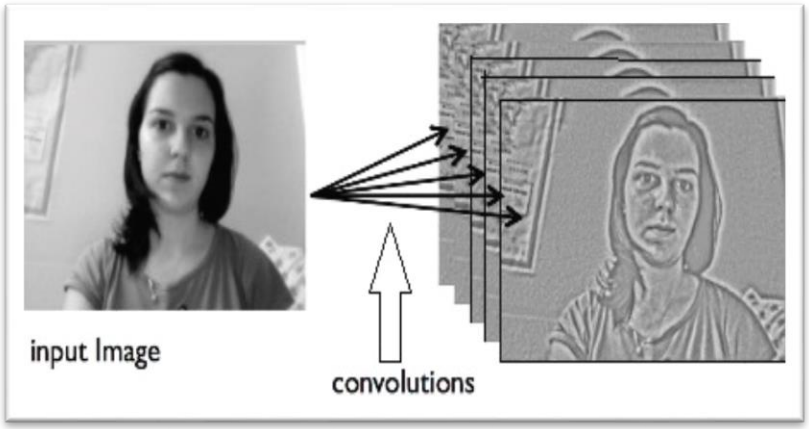
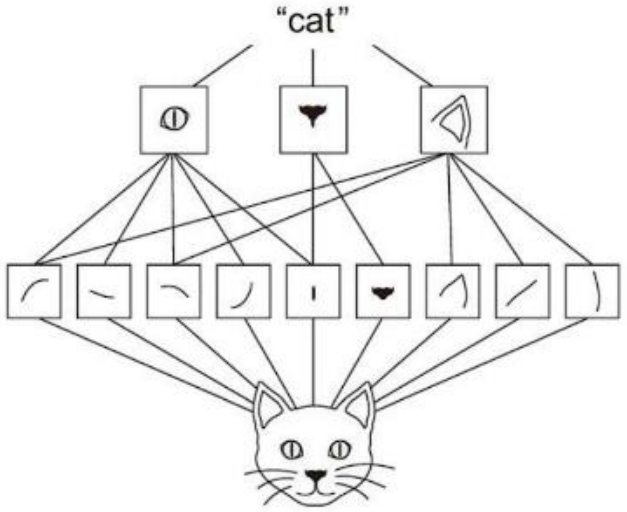


Figure 11-7. AdaGrad versus Gradient Descent

3.2 CNN 이해

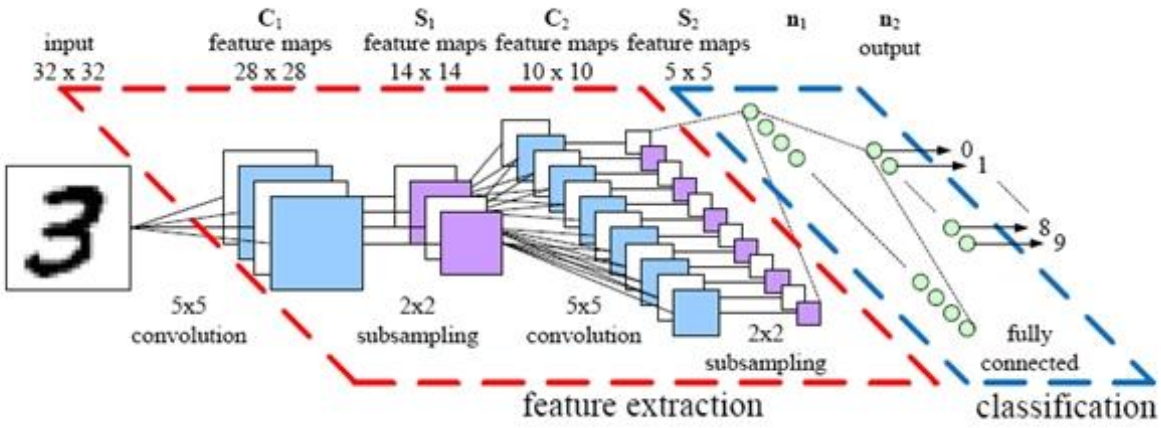
- CNN은 자동 특징 추출기(Feature Extractor) 임
- CNN이 나오기 전에는 사람의 얼굴에서 특징을 만들어내는 과정을 직접 코딩해야 했는데 이제는 CNN이 이러한 작업을 대신하기도 하고 많은 특징 정보를 만들어 낼 수 있음
- 사람이 물체를 볼 때, 세밀하게 다 보는 것이 아니고 전반적인 Shape, 특징을 봄. CNN의 원리임
- CNN은 2차원으로 데이터를 처리하기 때문에 일반적인 신경망 모형에 비해 유리
- 고차원의 이미지를 윤곽선(좌측 Layer)부터 세세한 특징(우측 Layer)까지 따로 분리할 수 있음





3.2 CNN 이해

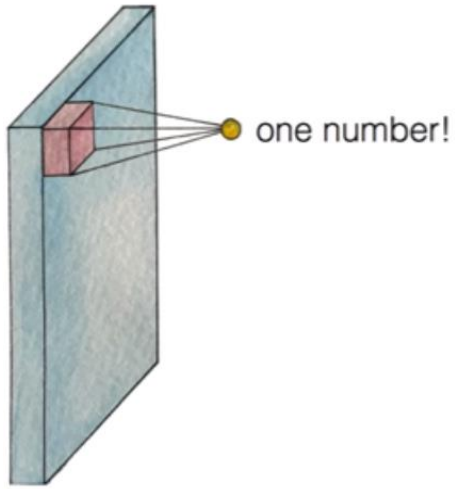
- CNN 처리 과정은 크게 두 과정을 구분됨
  첫번째 과정은 이미지로부터 Feature를 Extraction 하는 과정이고 다음은 Feature를 이용하여 딥러닝을 수행하는 과정임
- Feature Extraction은 Convolution layer를 이용하여 이미지의 특징을 추출하고 Pooling Layer에서 대표값을 만드는 과정을 반복하여 특징을 단순화 함
- 단순화된 특징은 이미지를 구별할 수 있는 간결한 특징만 남고 판별에 도움이 되지 않는 자세한 이미지 정보는 삭제하는 원리
- CNN은 사람이 이미지를 인식하는 과정과 유사함
  사람이 이미지를 볼 때, 이미지 전체를 학습하는 것이 아니라 몇 개의 특징을 학습한다고 함



3.2 CNN 이해

- 우리가 학습할 그림은 아래와 같이 가로*세로*3(R, G, B) 형식의 3차원 행렬정보임
- 이 이미지에 우측의 작은 필터를 끼워보자. 이제, 우리는 필터를 통해 그림을 볼 것임
  필터는 5*5*3개의 정보를 하나의 값으로 변환하여 정보를 축약함
- 이 필터의 정보가 하나의 값으로 변환되는 과정은
- 각 채널에 5*5 개의 이미지 정보를 임의의 5*5 W에 elementwise 곱을 하여 채널 별로 모두 합하여 하나의 값을 생성함
- 여기서, 5*5 는 커널 사이즈임

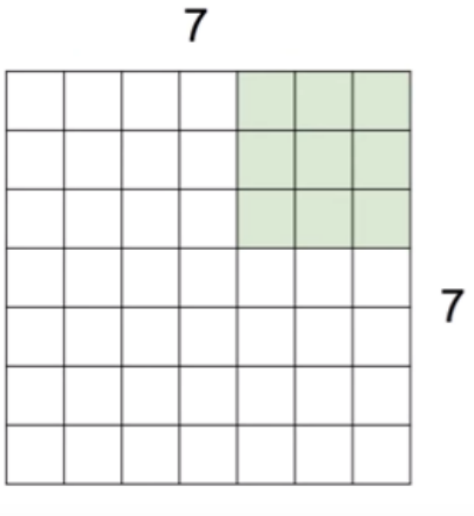
32x32x3 image

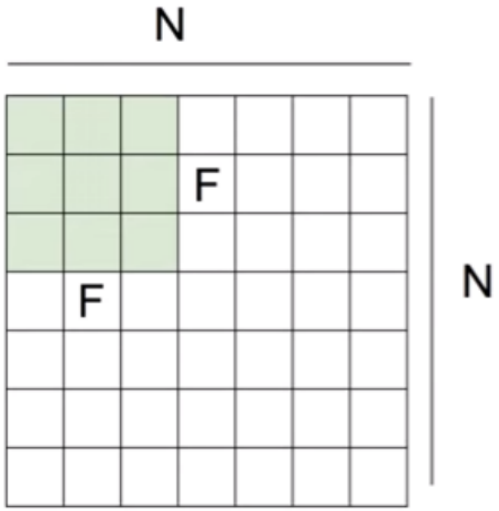


32x32x3 image

3.2 CNN 이해

- 편의 상, 아래와 같이 7*7 이미지가 있고 필터는 3*3 라고 하자.
- 이 필터를 좌에서 우로 한 칸씩 이동(Stride=1)하여 만들 수 있는 이동 회수는 5*5개임
  모든 이동에 대해 필터의 출력 값은 같은 가중 값으로 계산하여 좌→우, 위→아래로 내려가면서 출력 생성
- 만약 두 칸 씩 이동 (Stride=2) 하면 이동 회수는 3*3개임
- N*N 이미지에 대해 F*F 필터를 통과 시키면, 이동회수는 (N-F)/Stride + 1 이 됨
- 원본 이미지의 차원은 필터를 통과하면서 점차 줄어들어 작은 이미지로 변환됨
- 이를 방지하기 위해 Zero Padding을 사용하기도 함

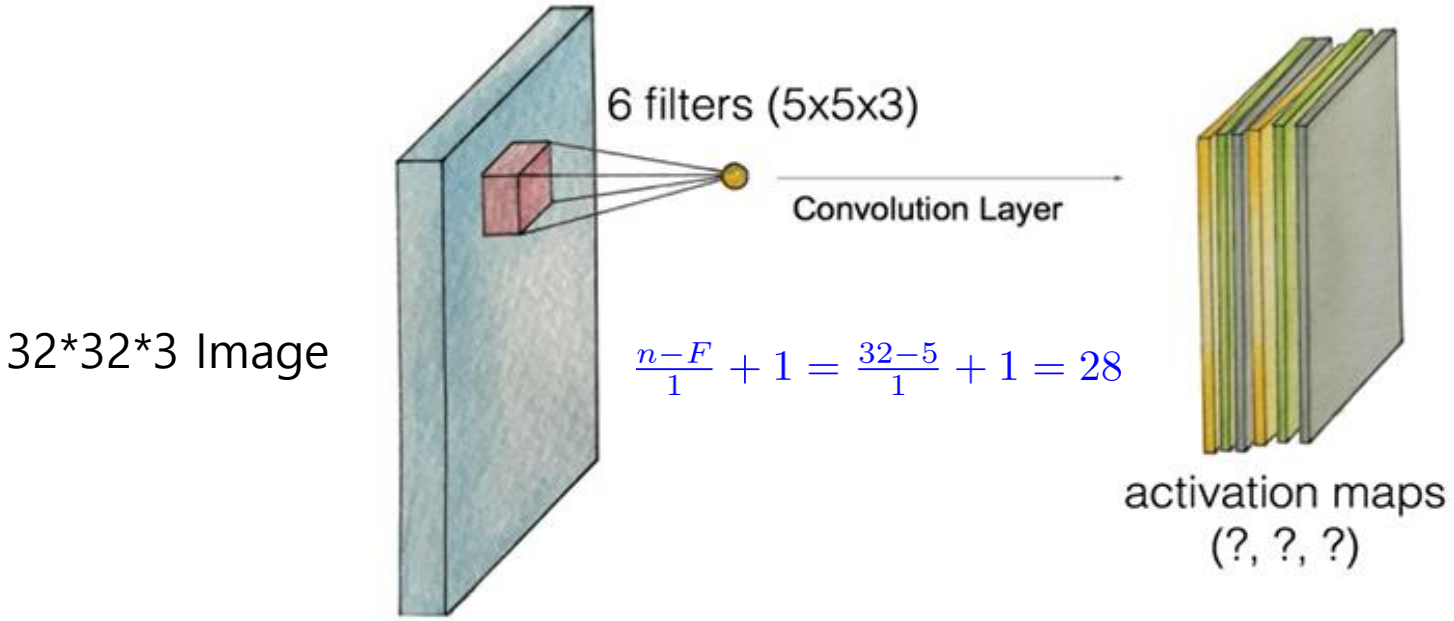




0	0	0	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	1	0	0
0	0	0	1	1	1	0
0	0	1	1	1	0	0
0	0	1	1	0	0	0
0	0	0	0	0	0	0

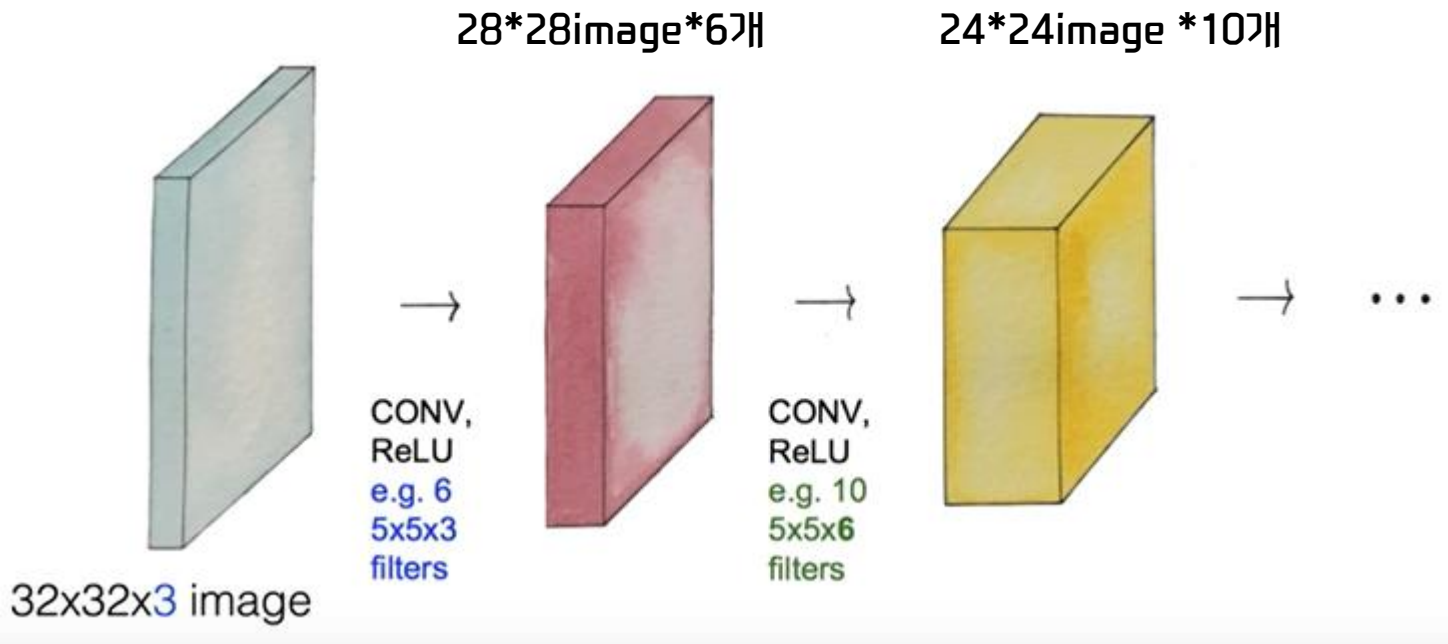
3.2 CNN 이해

- 우리가 학습할 그림은 아래와 같이 가로*세로*3(R, G, B) 형식의 3차원 행렬정보임
- 이 이미지에 작은 필터를 끼워보자. 이제, 우리는 필터를 통해 그림을 볼 것임
  필터는 5*5*3개의 정보를 하나의 값으로 변환하여 정보를 축약함
- 필터를 통과한 값은 28*28개의 값이 생성됨
- 가중치를 바꿔가면서 6번 반복하면 28*28*6개의 그림이 생성됨
  여기서, 몇 번 반복할 것인가를 적당하게 정하여야 함



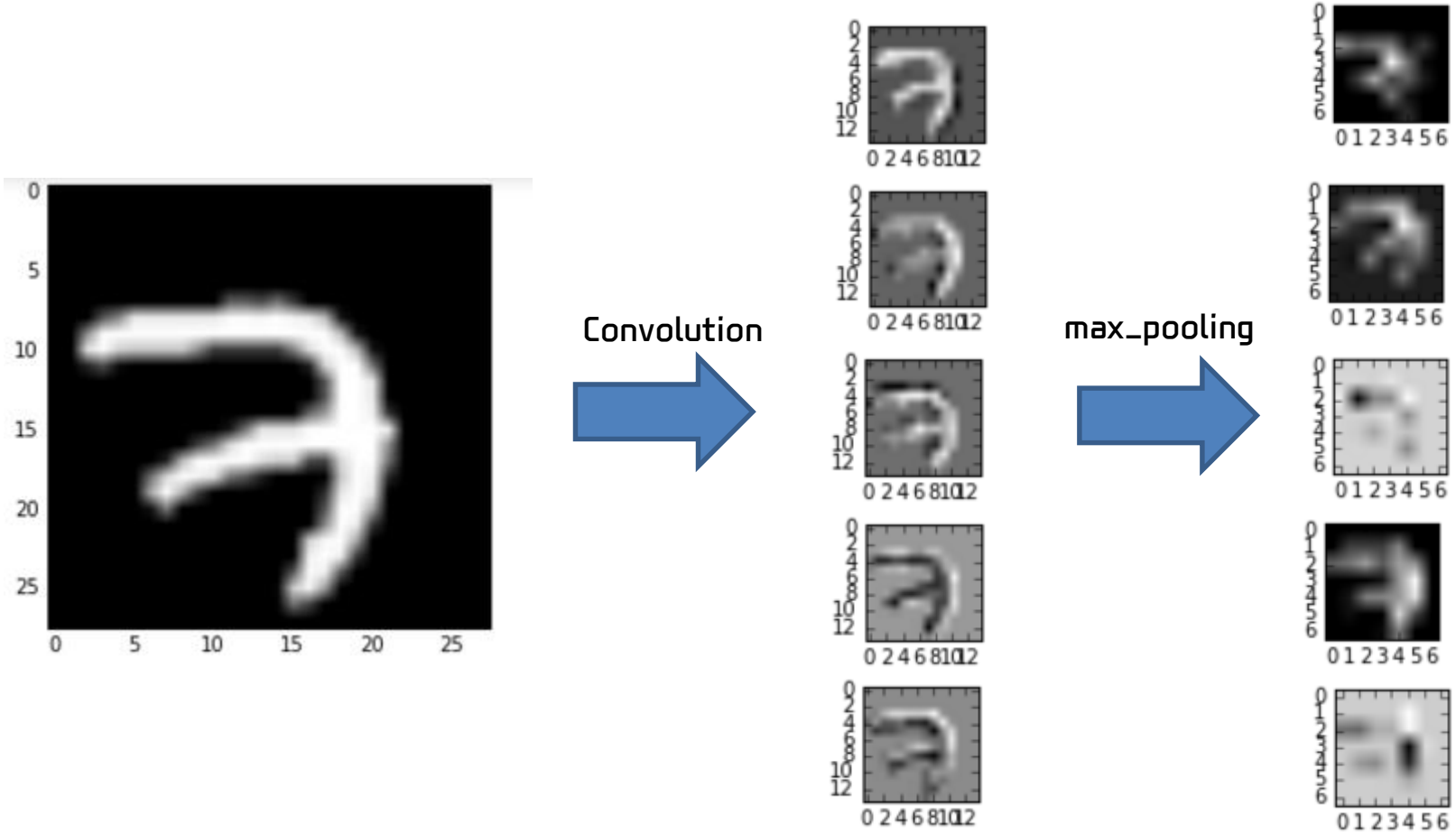
3.2 CNN 이해

- 이 과정을 반복하면서 원본 이미지는 Feature Map으로 바뀌는 것임



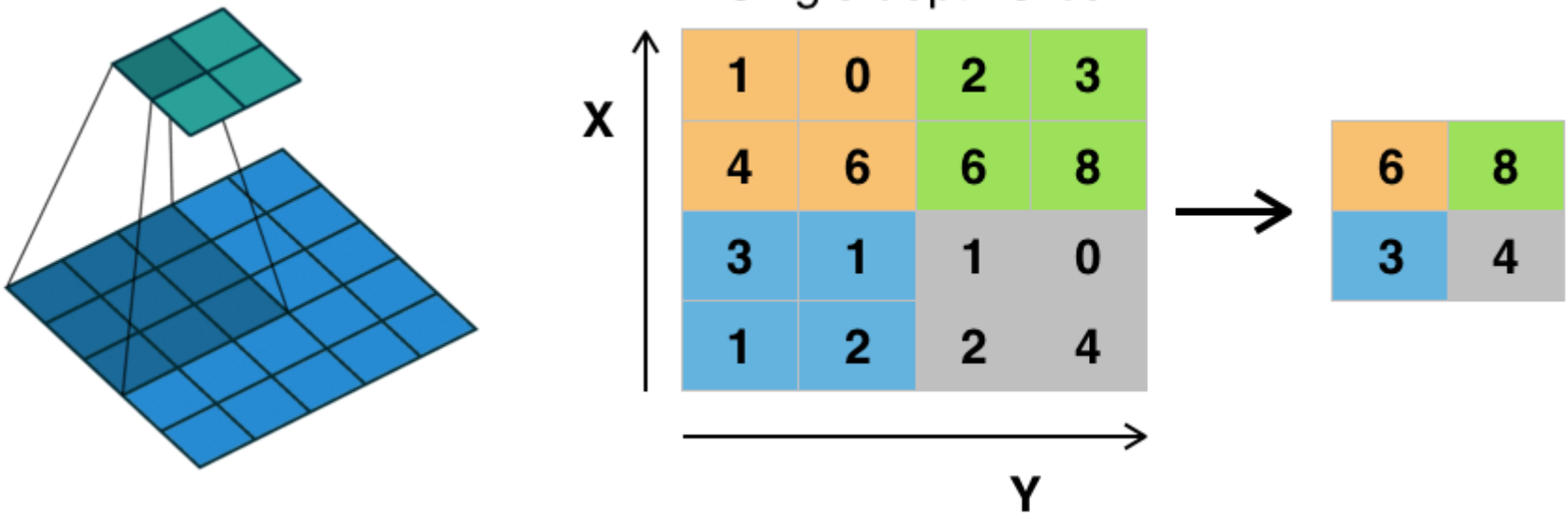
3.2 CNN 이해

Conv. Layer는 특징을 추출하고 Max-Pooling Layer는 이미지의 잡음을 제거 하면서 동시에 차원을 축소하여 고차원 이미지의 계산량을 줄여 줌



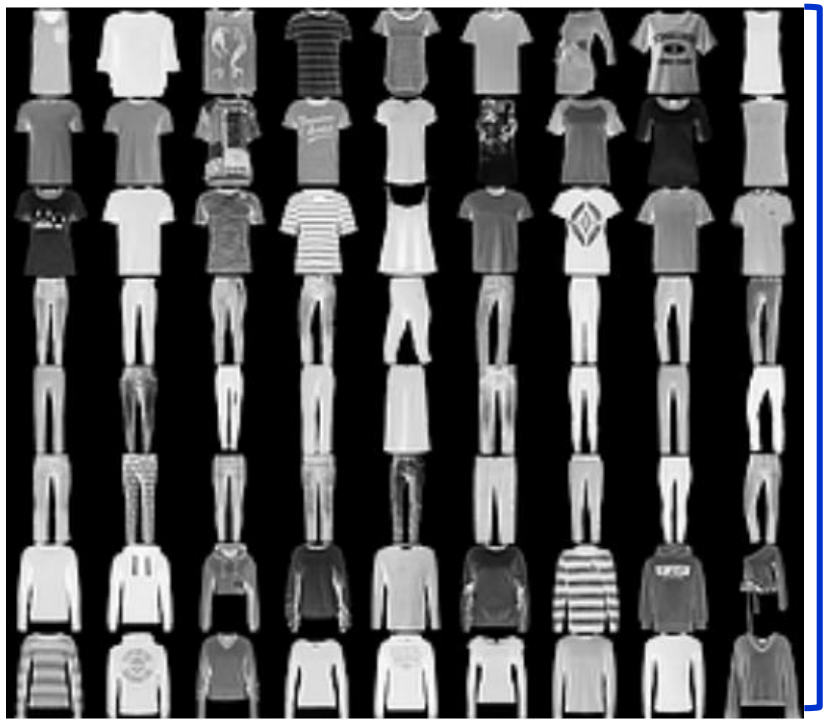
3.2 CNN 이해

- 아래는 max pooling layer를 예임
- max pooling layer는 이미지에서 뚜렷한 부분만 남기고 나머지는 제거하는 레이어로 잡음을 제거하는 효과와 동시에 이미지 크기를 줄여주는 역할을 수행함



3.3 Fashion Mnist 실습

Fashion Mnist는 CNN의 Hello World 프로그램임
Mnist Data와 똑 같이 28*28*1로 Training 6만장, Test 만장으로 이루어짐

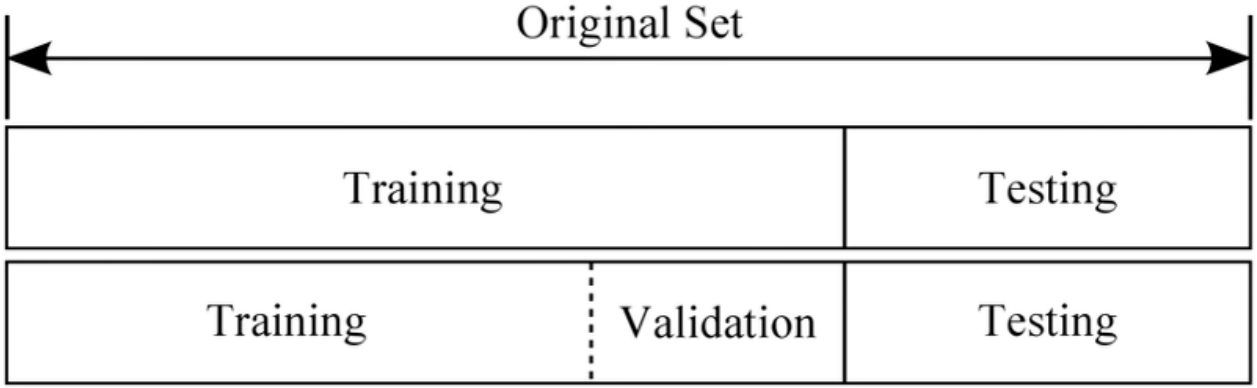
f  =

```
["T-shirt/top",  # index 0
 "Trouser",      # index 1
 "Pullover",     # index 2
 "Dress",        # index 3
 "Coat",         # index 4
 "Sandal",       # index 5
 "Shirt",        # index 6
 "Sneaker",      # index 7
 "Bag",          # index 8
 "Ankle boot"] # index 9
```

3.3 Fashion Mnist 실습

Training / Validation / Test

- 데이터가 충분히 많다면 보유한 모든 데이터를 학습하지 않고 일부만 학습하고 일부는 학습이 잘되었는지를 판단하는데 사용함
- Training Dataset은 학습에 사용하고 Validation Dataset은 학습이 잘되는지 체크하는 용도임
- Test Dataset은 학습이 실제 상황에서 어느 정도 정확도를 줄지를 예상하는 용도임
- Training의 정확도에 비해 Validation 정확도가 낮다면 오버피팅과 같은 상황을 의심해 볼 수 있음
- 3개 데이터 셋에서 정확도가 비슷하게 나오는 것이 목표임



3.3 Fashion Mnist 실습

Batch, Epoch, SGD(Stochastic Gradient Descending)

- 6만장 이미지를 학습하는데 파라미터를 한번 갱신하기 위해 6만장의 이미지를 사용하여 Cost 함수를 계산하는 것은 비효율적임
- 예를 들어, 100장의 이미지만 이용해서 Cost함수를 계산하고 파라미터를 갱신한다면 6만장의 데이터로 600번 갱신이 가능함. 여기서, 100장이 Batch Size임.
- 에포크는 더 많은 갱신을 위해 전체 데이터 6만 건을 몇 회 반복할 것인지를 결정하는 것임
- 5 Epoch라면 600 * 5회 갱신이 일어 남
- 배치 크기를 크게 하면 파라미터 갱신이 적어지는 대신 갱신된 값의 정확도가 올라감
- 배치 크기가 작으면 파라미터 갱신이 많아지는 대신 갱신된 값이 부정확하여 지그재그로 갱신됨
- SGD는 배치 데이터로 파라미터를 갱신하는 것을 의미함. 즉, 배치 데이터가 확률적으로 선택된다는 의미
- 배치크기가 커지면 많은 GPU 메모리를 요구됨

표준화 변환의 필요성

$$Z_i = \frac{X_i - \bar{X}}{S_X}$$

$$Z_i = \frac{X_i - X_{Min}}{X_{Max} - X_{Min}}$$

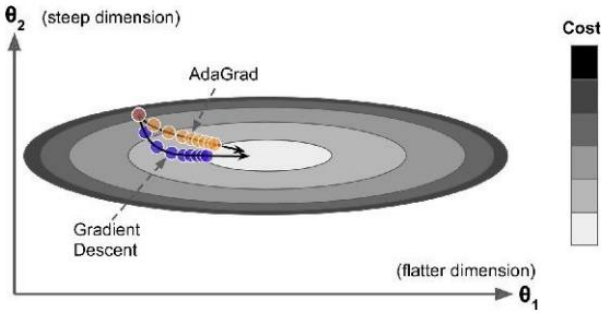


Figure 11-7. AdaGrad versus Gradient Descent

3.3 Fashion Mnist 실습

One Hot Encoding

Mnist Data의 Y=0,1, … , 9 이지만, 이 값이 크기를 가지는 것은 아니다.
이 경우, 통계학의 Dummy Variable 개념을 이용하여 아래와 같이 0,1로 구성할 수 있다.
이렇게 데이터를 변환하는 것을 one hot encoding 이라고 한다.

$$Y = \begin{pmatrix} 0 \\ 1 \\ 4 \\ 9 \end{pmatrix} \xrightarrow{\text{One hot encoding}} Y = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Y가 one hot encoding 되면 이에 해당하는 Hypothesis 값도 같은 형식으로 아래와 같이 행렬로 표시된다.

$$H(X) = \begin{pmatrix} 0.7 & 0.2 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.5 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0.99 \end{pmatrix}$$

3.3 Fashion Mnist 실습

Softmax 함수

- Mnist 문제는 Y Label이 10개 임. 이 문제는 Simple Logistic Regression을 10회 수행하는 것임
- 즉, 특정 이미지가 0인지, 1인지, ... , 9 인지를 각 판별하기 위해 10개 모형을 만들어 각 모형에서 0일 확률, 1일 확률, ... , 9일 확률을 계산함
- 이 때, 0일 확률과 0이 아닐 확률의 합은 1이므로, 각 레이블 확률의 합 = 1로 변환하는 과정이 필요함
- 이렇게 각각의 확률을 전체 레이블의 확률로 변환하는 함수를 Softmax 라고 함
- 케라스에서는 아래와 같이 Softmax 함수를 사용함

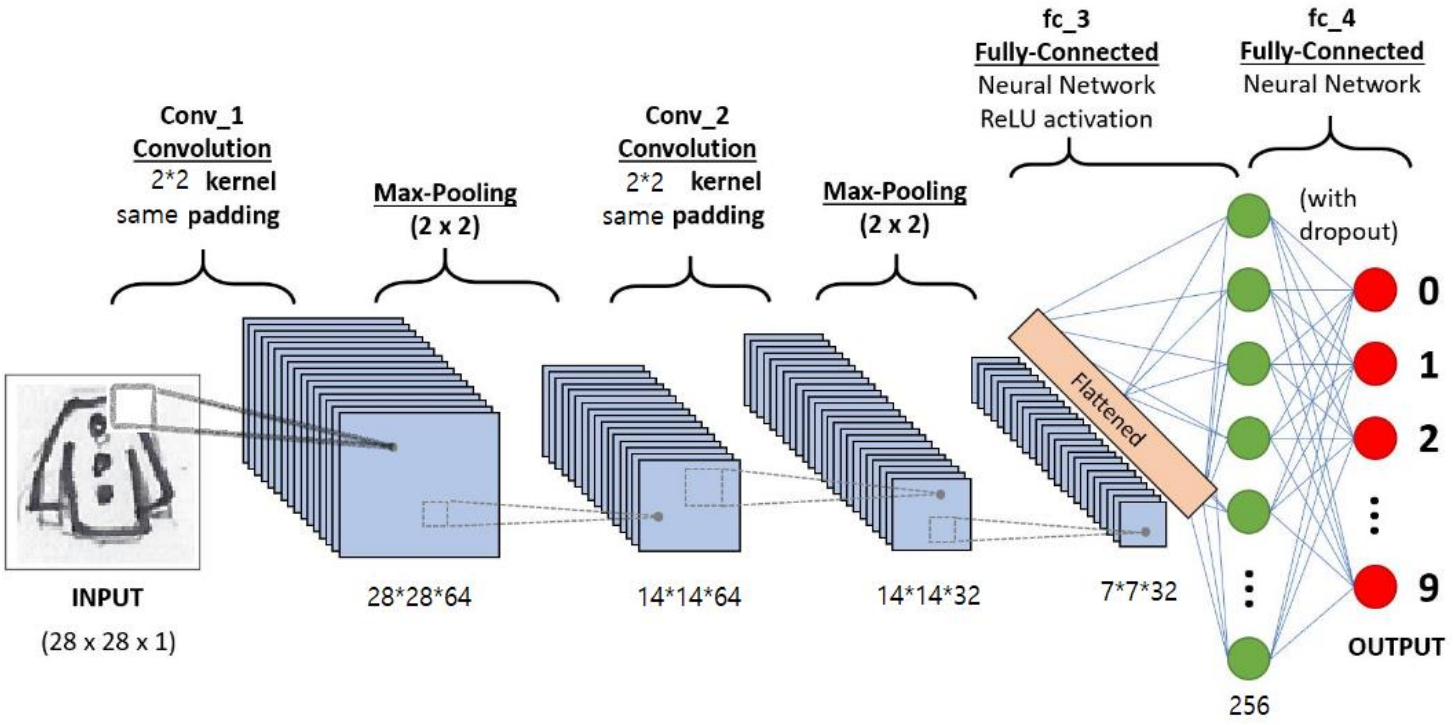
```
model.add(tf.keras.layers.Dense(10,activation='softmax' ))
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

$$H(X) = \begin{pmatrix} 0.7 & 0.2 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 \\ 0 & 0 & 0.2 & 0 & 0.5 & 0 & 0 & 0.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.01 & 0.99 \end{pmatrix}$$

3.3 Fashion Mnist 실습

28*28*1 이미지를 아래와 같은 CNN 모형으로 학습 시켜보자.

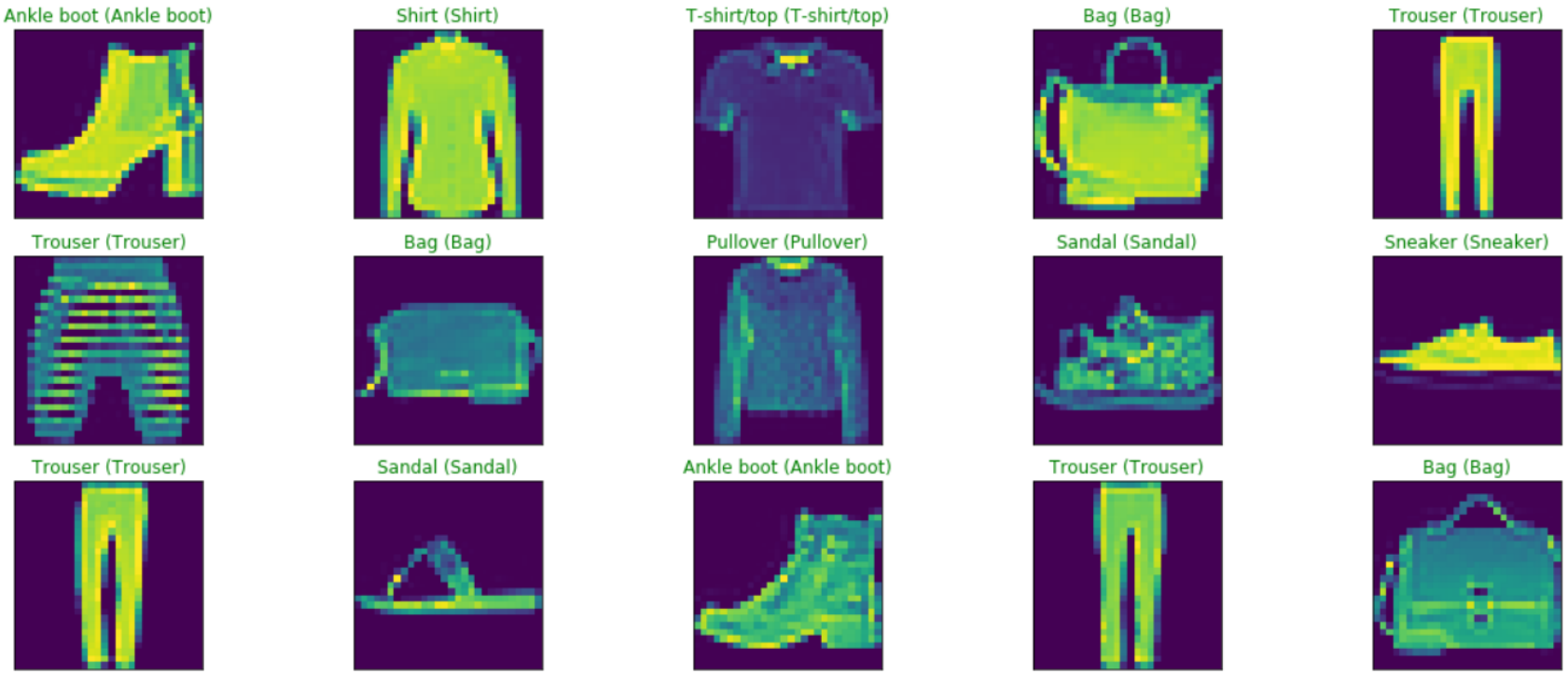
CNN을 사용하지 않고 딥러닝으로 이 문제를 학습하면 Flattened Data 즉, 784개가 순차적으로 입력되는 구조를 가진다. 이는 이미지의 특성인 2차원 정보를 반영하지 못하므로 CNN과 같이 2차원 구조의 입력이 효율적 임



3.3 Fashion Mnist 실습

모형에서 사용되는 총 parameter는 412,778개이고 정확도는 92%를 기록

FashionMnist.py

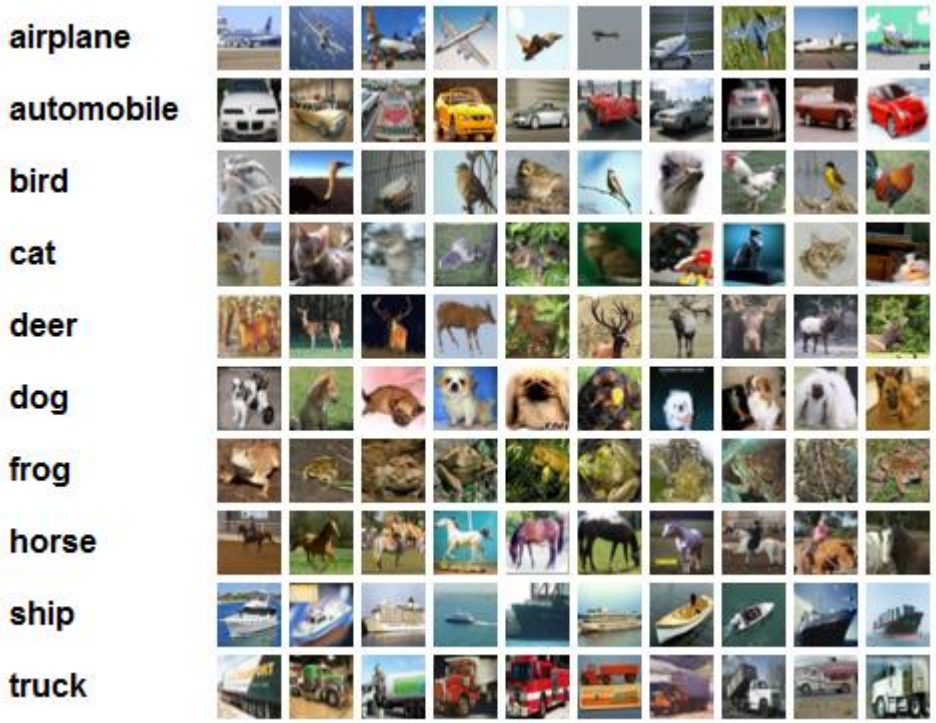


3.4 CNN 발전

좀 더 복잡한 문제를 생각해보자.

Cifar10 자료는 Canadian Institute For Advanced Research에서 만든 airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, trucks 이미지임

Fashion Mnist와 다르게 배경이 존재하고 이미지 크기는 32*32*3 임



3.4 CNN 발전

이미지 파일을 읽어 수치행렬로 바꾸어야 딥러닝이 가능하므로 이 과정도 직접 수행해보자.
이미지 파일 한 장을 수치로 바꾸는 것은 아래와 같이 간단하게 처리할 수 있음

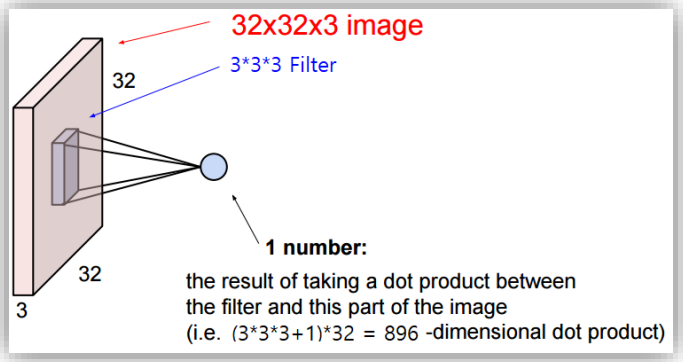
(312, 240, 3)

```
[[[208 208 206]
  [ 12  12  10]
  [  2   4   1]
  ...
  [ 62  66  75]
  [ 60  64  73]
  [ 68  72  81]]
 ...
```

image2pixel.py

3.4 CNN 발전

CIFAR10 자료를 CNN으로 학습해보자. Cifar10_CNN_74_Train.py




```
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 32, 32, 3)         0
conv1 (Conv2D)               (None, 30, 30, 32)        896
dropout_1 (Dropout)          (None, 30, 30, 32)        0
conv2 (Conv2D)               (None, 30, 30, 32)        9248
pool1 (MaxPooling2D)         (None, 15, 15, 32)        0
conv3 (Conv2D)               (None, 15, 15, 64)        18496
pool2 (MaxPooling2D)         (None, 7, 7, 64)          0
flatten_1 (Flatten)          (None, 3136)              0
dropout_2 (Dropout)          (None, 3136)              0
poolhidden (Dense)           (None, 512)               1606144
dropout_3 (Dropout)          (None, 512)               0
output (Dense)               (None, 10)                5130
=================================================================  Total params: 1,639,914
```

(3*3*3+1) * 32장 = 896

(3*3*32+1) * 32장 = 9248

(3*3*32+1) * 64장 = 18496

7*7*64 = 3136

3136*512 + 512 = 1606144

512*10+10= 5130

3.4 CNN 발전

아래는 이작업을 수행하는 keras 코드임

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
from keras import datasets
from keras import layers, models
from keras.utils import np_utils
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()
Y_train = np_utils.to_categorical(y_train)
Y_test = np_utils.to_categorical(y_test)
img_rows, img_cols, _ = x_train.shape[1:]
X_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 3)
X_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 3)
input_shape = (img_rows, img_cols, 3)

X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
num_classes = 10
batch_size = 32
```

3.4 CNN 발전

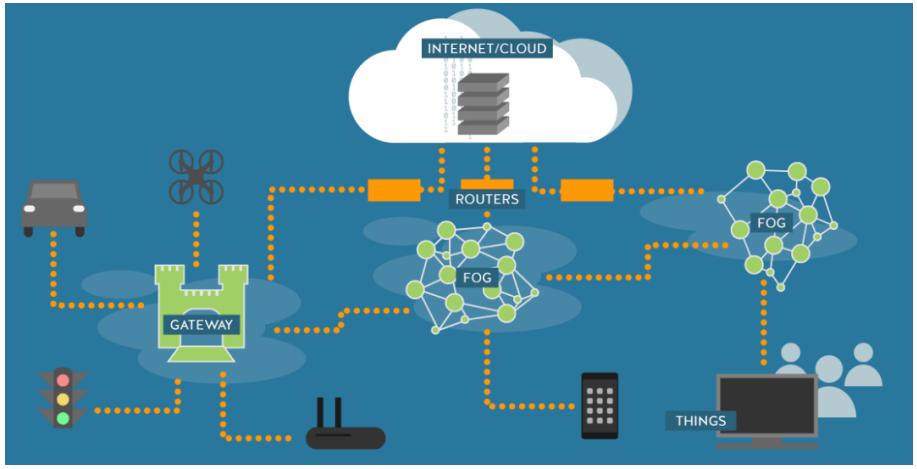
```
x = layers.Input(shape=input_shape,  name='input')
h = layers.Conv2D(32, kernel_size=(3, 3), activation='relu',  name='conv1')(x)
h = layers.Dropout(0.2)(h)
h = layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same', name='conv2')(h)
h = layers.MaxPooling2D(pool_size=(2, 2), name='pool1')(h)
h = layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same', name='conv3')(h)
h = layers.MaxPooling2D(pool_size=(2, 2), name='pool2')(h)
h = layers.Flatten()(h)
h = layers.Dropout(0.2)(h)
h = layers.Dense(512, activation='relu', name='poolhidden')(h)
h = layers.Dropout(0.2)(h)
y = layers.Dense(num_classes, activation='softmax', name='output')(h)

model = models.Model(x, y)
print(model.summary())
epochs = 25
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
history = model.fit(X_train, Y_train, batch_size=batch_size,
          epochs=epochs, validation_split=0.1, verbose=2)
score = model.evaluate(X_test, Y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

3.4 CNN 발전

- 학습은 시간과 메모리가 많이 소요되는 작업임
- 일단, 학습이 되면 한동안 학습할 필요가 없으므로 학습된 미지수를 저장해두는 것이 필요함
- 학습된 이미지를 저장했다면 이를 읽어 forward 연산만 수행하면 됨

forward ⇔ backward
(Backpropagation)

Edge Computer: forward Only



Cifar10_CNN_74_Test.py

3.4 CNN 발전

- **결과는 74% 정도다.** 비교적 높은 수치다. 하지만, 인간보다 더 잘한 것은 아님
- CNN은 대부분 GPU를 사용하여 계산하는데 계산효율을 위해 Mini-batch로 학습 함
- 2015년 arXiv에 Mini-batch 학습의 문제점을 찾아내고 이를 해결하는 Batch Normalization 논문이 발표됨 (Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariance Shift)

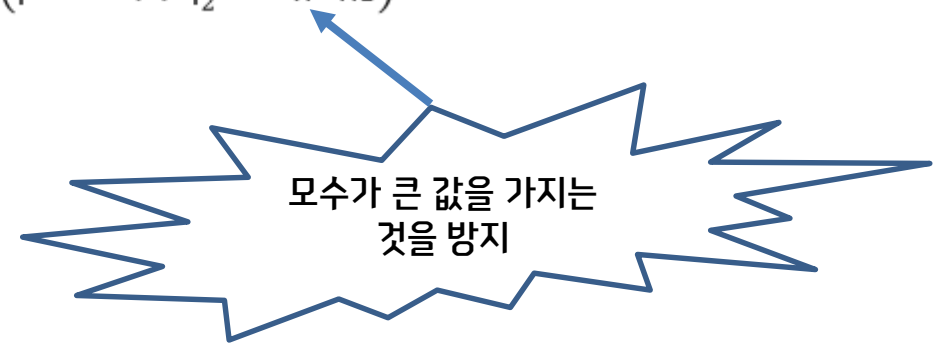
  한마디로 설명하면 미니배치 처리에서 배치마다 입력 데이터의 분포가 달라지면 달라진 분포를 학습하기 위해 방향을 조금씩 이탈하는 것을 방지하는 아이디어임(일종의 미니배치 정규화)

```
x = layers.Input(shape=input_shape,  name='input')
h = layers.BatchNormalization()(x)
h = layers.Conv2D(32, kernel_size=(3, 3), activation='relu',  name='conv1')(h)
h = layers.Dropout(0.2)(h)
h = layers.BatchNormalization()(h)
h = layers.Conv2D(32, kernel_size=(3, 3), activation='relu', padding='same',name='conv2')(h)
h = layers.MaxPooling2D(pool_size=(2, 2), name='pool1')(h)
h = layers.BatchNormalization()(h)
h = layers.Conv2D(64, kernel_size=(3, 3), activation='relu', padding='same', name='conv3')(h)
h = layers.MaxPooling2D(pool_size=(2, 2), name='pool2')(h)
h = layers.Flatten()(h)
        .
        .                                  Cifar10_CNN_79_Train.py    Cifar10_CNN_79_Test.py
        .
```

3.4 CNN 발전

- 결과는 79% 로 Batch Normalization 적용으로 5% 정도 상승
- 층을 더 깊게 하여 정확도를 증가시키고 싶은데 오버피팅 위험이 걱정됨
- 가중 값이 커서 하나의 노드에 결과가 영향을 받는 것을 지양할 수 없을까?
- 아래의 식은 통계학의 Ridge Regression에서 가져온 것임
  회귀 방정식의 모수를 θ 라고 할 때, Cost 함수에 θ 의 제곱합을 더해서 θ 의 값이 커지는 것을 방지하는 방법임

Kernel Regularization $\min\left(\left|\left|Y - X(\theta)\right|\right|_2^2 + \lambda||\theta||_2^2\right)$

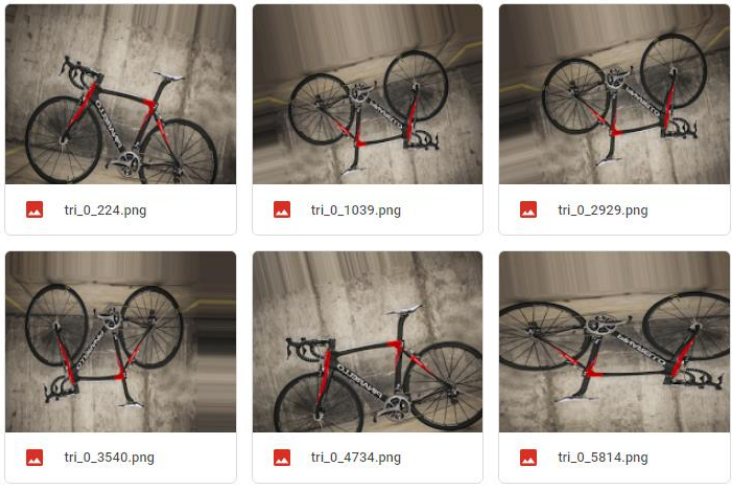


79% CNN 모형에서 32 conv → 64 conv → 128 conv 형식으로 Convolution Layer를 추가 확장하고 Kernel Regularization을 추가한 결과, 84%의 정확도를 확보

Cifar10_CNN_84_Train.py Cifar10_CNN_84_Test.py

3.4 CNN 발전

다음 아이디어는 Image Generator다. 단순한 아이디어지만, 결과는 아주 훌륭함
아래는 좌측 이미지를 Image Generator를 통해 여러가지 이미지로 변형한 것임



https://drive.google.com/open?id=1IW8gkBsKUEjjW4XCM558aIPLHrphgRTf

Image Generator를 추가하고 Epoch를 대폭 늘려 수행한 결과, 88%로 4% 증가
답이 안나온다. GPU 업글하고 싶다~~~
학습시간이 너무 오래 걸려 학습결과를 Save하기로 하자.

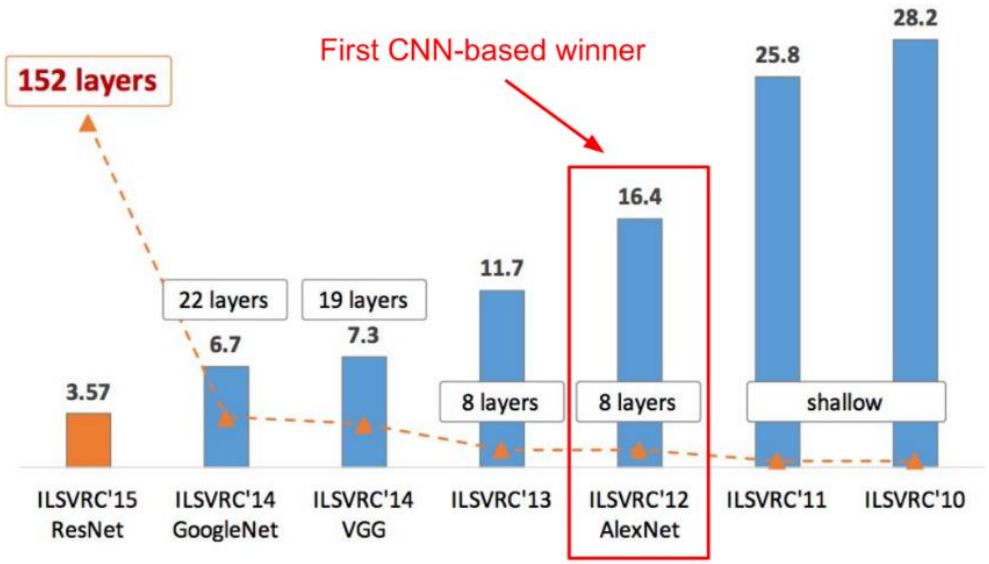
Cifar10_CNN_88_Train.py Cifar10_CNN_88_Test.py

3.5 ImageNet Challenge

- ImageNet은 2007년에 스탠포드 대학의 페이페이 리와 프리스턴 대학의 카이리교수가 시작
- 집단 지성을 이용해 10억 여장 이미지를 다운로드하고 이를 소셜 클라우딩으로 Labeling
- 이미지넷에는 1,000여개의 다양한 클래스가 존재
- 2012년 토론토 대학의 Alex가 개발한 AlexNet이 이 이미지넷 챌린지에서 우승을 하면서 지금까지 많은 CNN 개발에 기초가 됨
- 2013년 Oxford 대학에서 AlexNet을 좀 더 깊게 만든 VGGNet16, VGGNet19(´14) 발표
- 2014년 GoogleNet은 CNN을 좌우 뿐 아니라 상하로도 Wide하게 확장한 CNN 모형을 제안함
- 2015년 MS에서 ResNet을 발표. 이는 이전에 발표한 CNN보다 더 깊게 만든 것인데 깊게 만들면서 발생할 수 있는 오버피팅을 피하기 위해 Residual Shortcut 방법론 제안

3.5 ImageNet Challenge

- 이처럼 성능 좋은 CNN 모형이 만들어지면서 초기 난수로 만든 Conv. Layer가 학습을 끝냈을 때에는 이미지의 각종 특징을 주성분으로 분해한 Feature Set로 변한다는 사실을 알게 됨
- 이는 CNN 모형의 Conv. Layer를 재사용하게 되는 획기적인 전환을 또 한번 이루게 됨
- 현재, VGGNet, GoogleNet, ResNet은 상용 프로젝트에 많이 쓰이는 네트워크임
- VGGNet, GoogleNet, ResNet은 Transfer Learning에 주로 사용되는 Pre-trained CNN임



감사합니다