딥러닝 (Deep Learning)

- **Neural Network (인공 신경망)**

  : 기계 학습 역사에서 가장 오래된 기계 학습 모델

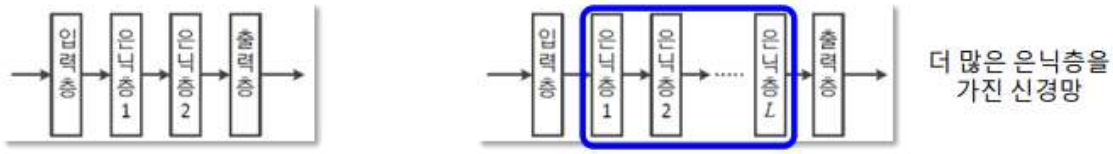
  현재 가장 다양한 형태를 가짐

  퍼셉트론 -> 다층 퍼셉트론 -> 딥러닝 (기계 학습의 주류 기술)

- 신경망 모델

  1) 전방 신경망 (Feed-forward)과 순환 (Recurrent) 신경망

  2) 얕은 신경망과 깊은 신경망



  3) 결정론(Deterministic) 신경망과 확률적(Stochastic) 신경망

- **Perceptron (퍼셉트론)**

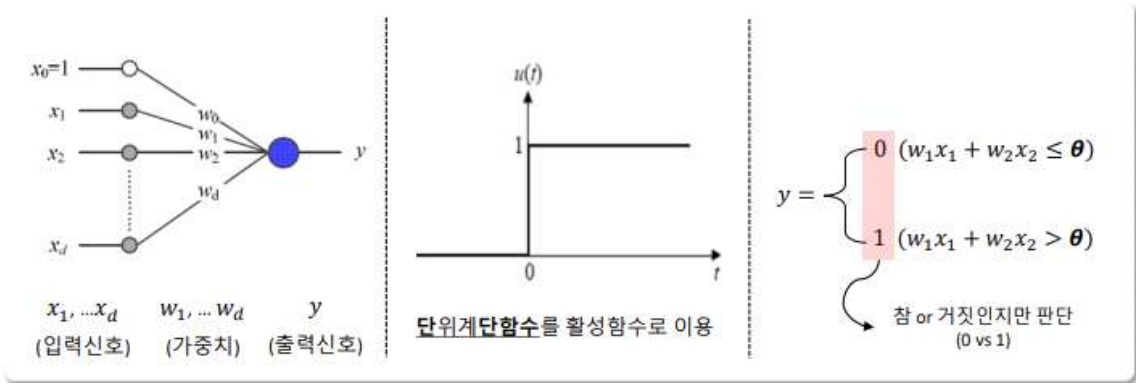
  : 1세대 딥러닝,다수의 신호(n개)를 입력받아 특정한 연산을 거쳐 하나의 값을 출력하는 방식

  단일층 구조

  출력층은 한 개의 노드

  i번 째 입력층 노드와 출력층을 연결하는 엣지는 가중치 wi를 가짐

- 단점 : 간단한 xor 연산도 학습하지 못하는 문제 발생

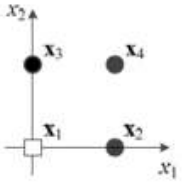


- Perceptron 동작

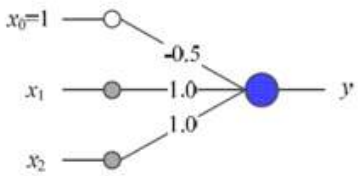
□ 2차원 특징 벡터로 표현되는 샘플 4개를 가진 훈련집합 X, Y

- $X = \{X_1, X_2, X_3, X_4\}$
- $Y = \{Y_1, Y_2, Y_3, Y_4\}$

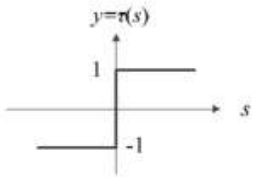
$$\mathbf{x}_1 = \begin{pmatrix}0\\0\end{pmatrix},\ y_1 = -1,\ \mathbf{x}_2 = \begin{pmatrix}1\\0\end{pmatrix},\ y_2 = 1,\ \mathbf{x}_3 = \begin{pmatrix}0\\1\end{pmatrix},\ y_3 = 1,\ \mathbf{x}_4 = \begin{pmatrix}1\\1\end{pmatrix},\ y_4 = 1$$



Training set



Perceptron



Activation function

✓ **OR논리 게이트를 이용한 퍼셉트론 동작**

$\boldsymbol{X}_1$: $\boldsymbol{S} = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5$ $\quad y = -1$

$\boldsymbol{X}_2$: $\boldsymbol{S} = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5$ $\quad y = 1$

$\boldsymbol{X}_3$: $\boldsymbol{S} = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5$ $\quad y = 1$

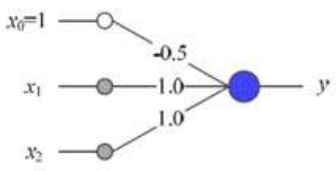
$\boldsymbol{X}_4$: $\boldsymbol{S} = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5$ $\quad y = 1$

□ 퍼셉트론에 해당하는 결정 경계

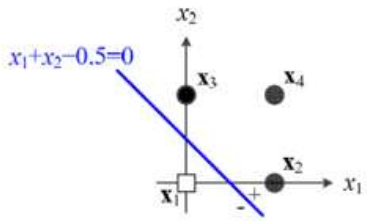
□ 결정 경계 $d(X) = d(x_1, x_2) = w_1x_1 + w_2x_2 + w_0 = 0\} \rightarrow\ x_1 + x_2 - 0.5 = 0$

- w_1과 w_2는 직선의 방향, w_0은 절편을 결정
- 결정 경계는 전체 공간을 +1과 -1의 두 부분공간으로 분할하는 분류기 역할

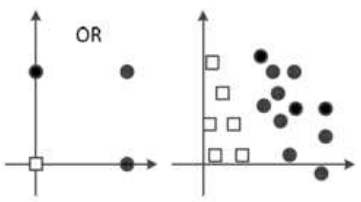
- 2차원은 결정 경계
- 3차원은 결정 평면
- 4차원 이상은 결정 초평면



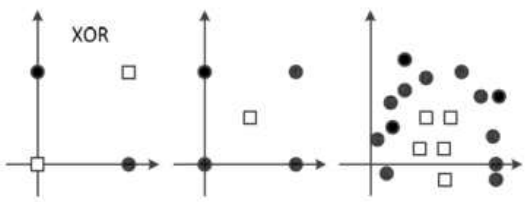
Perceptron



- Perceptron의 한계 : 선형분리가 불가능한 상황 존재



(a) 선형분리 가능



(b) 선형분리 불가능

- **Multi-layered Perceptron (다층 퍼셉트론)**

  : 2세대 딥러닝

  입력과 출력 사이에 하나 이상의 은닉층(hidden layer)을 추가해 학습

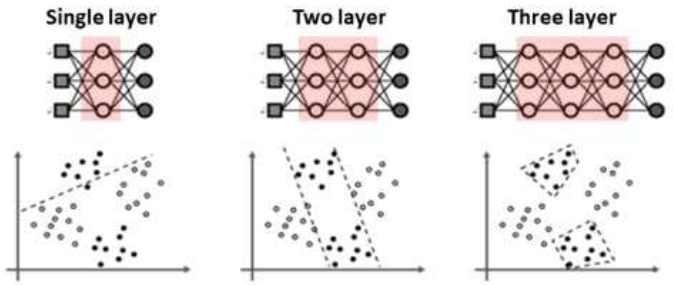
- 핵심 : 1) 은닉층의 존재 - 특징공간을 분류하는데 유리한 새로운 특징공간으로 변환시키는 역할

  2) 시그모이드 활성함수 - 융통성 있는 의사결정 가능 (Soft decision-making)

  3) 오류역전파 알고리즘 사용 : 다층 퍼셉트론이 순차적으로 이어진 구조

  은닉층의 개수가 증가할수록 가중치의 개수도 증가 -> 학습이 어려움

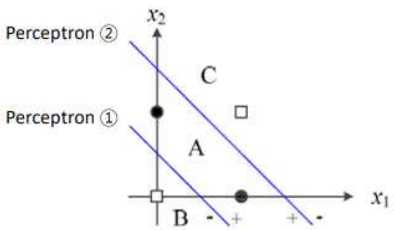
  역방향으로 진행하면서 한 번에 한 층씩 gradient를 계산하고 가중치를 갱신하는 방식 사용



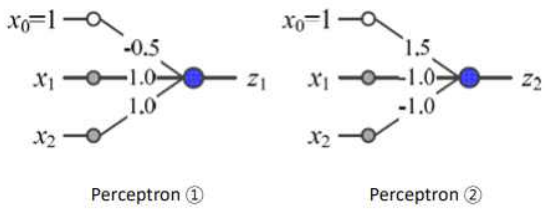
- 특징 공간 변환

  1) Perceptron 2개를 사용한 XOR 문제 해결

  • Perceptron ①과 Perceptron ②가 모두 +1이면 ● 부류이고 그렇지 않으면 □ 부류임



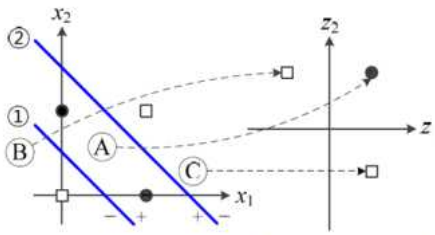
Perceptron 2개를 이용한 공간 분할



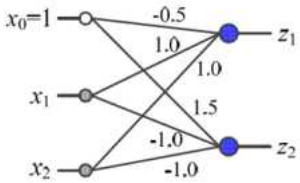
  2) Perceptron 2개를 병렬로 결합

  • 원래 공간 $X = (x_1, x_2)^T$를 새로운 특징 공간 $Z = (z_1, z_2)^T$로 변환

  • 새로운 특징 공간 z에서는 선형 분리 가능함



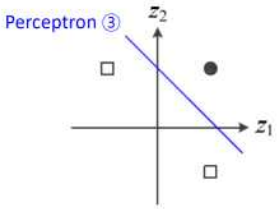
원래 특징 공간 x를 새로운 특징 공간 z로 변환



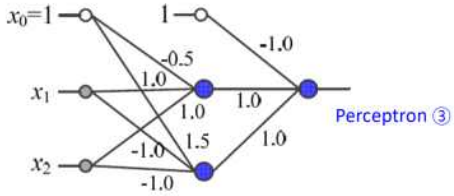
2개의 Perceptron을 병렬로 결합

3) Perceptron 1개를 순차로 결합

- 새로운 특징 공간 $Z = (z_1, z_2)$ 에서 선형 분리를 수행하는 퍼셉트론③ 을 순차 결합
- Multi-layered neural network 이 생성됨

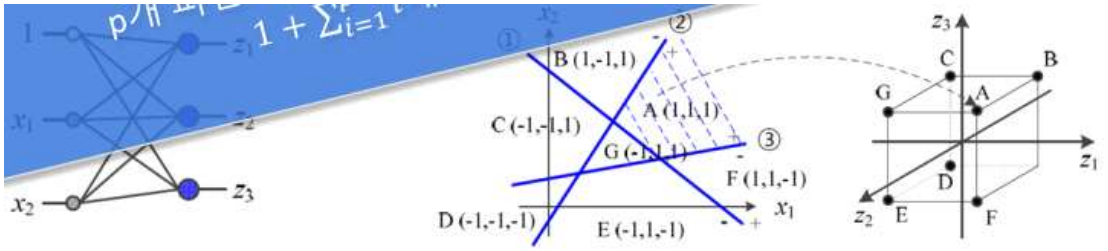


새로운 특징 공간에서 분할



3개의 Perceptron을 결합한 다층 Perceptron

3개의 퍼셉트론 결합 -> 2차원 공간을 7개의 영역으로 나누고 각 영역을 3차원 점으로 변환
활성함수(계단함수)를 사용하여 영역을 점으로 변환

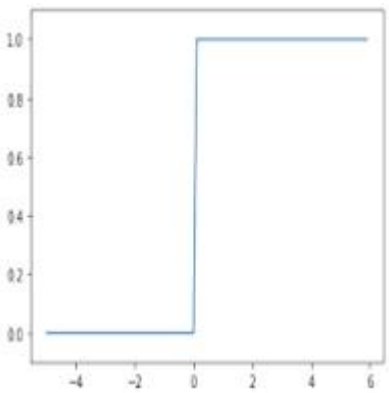


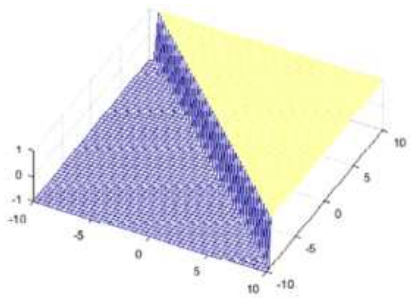
Perceptron 3개 결합　　7개 부분 공간으로 나눔　　3차원 공간의 점으로 매핑

=> p개의 퍼셉트론을 결합하면 p차원 공간으로 변환
　　개의 영역으로 분할

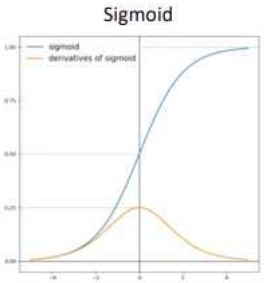
- **Activation function (활성함수)**

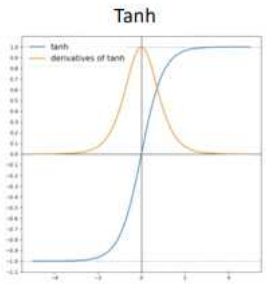
- Hard & Soft 공간 분할
　　딱딱한 의사결정 - 계단함수 (영역을 점으로 변환)

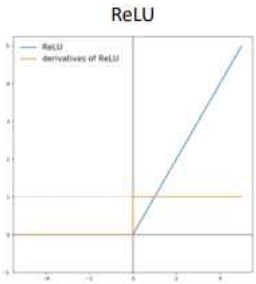


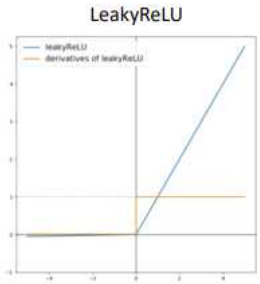


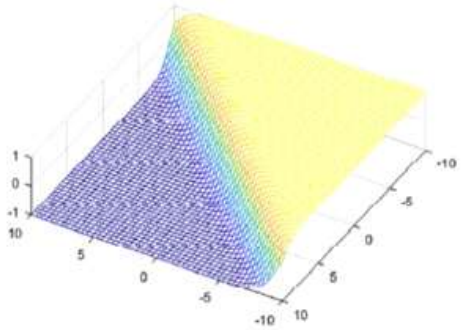
부드러운 의사결정 - 그 외 활성함수 (영역을 영역(면)으로 변환)











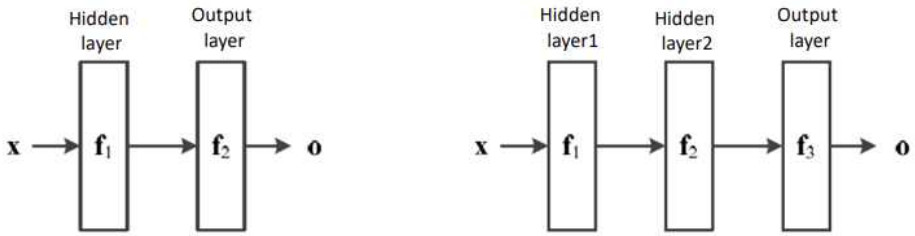
- 신경망이 사용하는 다양한 활성함수
  퍼셉트론 : 계단함수
  다층 퍼셉트론 : 시그모이드 / 하이퍼볼릭 탄젠트
  딥러닝 : ReLU

Name	Function	1th a derived function	Range
Step	$r(s)=\begin{cases}1 & s\geq 0\\ -1 & s<0\end{cases}$	$r'(s)=\begin{cases}0 & s\neq 0\\ \text{불가} & s=0\end{cases}$	-1과 1
Sigmoid	$r(s)=\frac{1}{1+e^{-as}}$	$r'(s)=ar(s)(1-r(s))$	(0~1)
Tanh	$r(s)=\frac{2}{1+e^{-as}}-1$	$r'(s)=\frac{a}{2}(1-r(s)^2)$	(-1~1)
ReLU	$r(s)=max(0,s)$	$r'(s)=\begin{cases}0 & s<0\\ 1 & s>0\\ \text{불가} & s=0\end{cases}$	(0~∞)

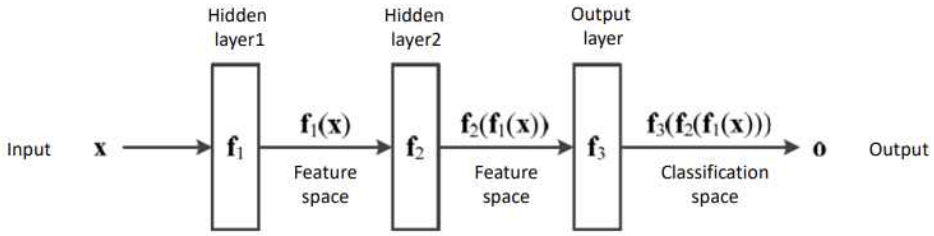
- Architecture

□ 특징 벡터 x를 출력 벡터 o로 매핑하는 함수로 간주할 수 있음

- 2개 Perceptron: $\mathbf{o} = f(x) = f_2(f_1(x))$
- 3개 Perceptron: $\mathbf{o} = f(x) = f_3(f_2(f_1(x)))$



-은닉층은 특징 벡터를 분류에 더 유리한 새로운 특징 공간으로 변환시켜 줌
딥러닝은 더 많은 Hidden layer를 거쳐 특징학습 (feature learning)을 함



특징 추출기로써의 은닉층

- Goals of Machine Learning

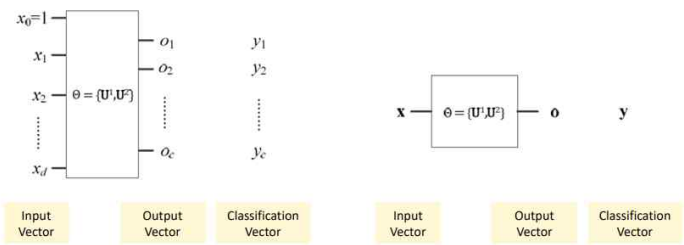
: 모든 샘플을 옳게 분류하는 함수 f(x)를 찾는 것을 목표로 함

- $Y = f(X)$
- $Y = f(X_i), i = 1,2,\dots,n$

- Loss function MSE (Mean Squared error, 평균제곱오차)로 정의

$$J(\Theta) = \frac{1}{n}\sum_{i=1}^{n}\|y - o(\Theta)\|^2$$

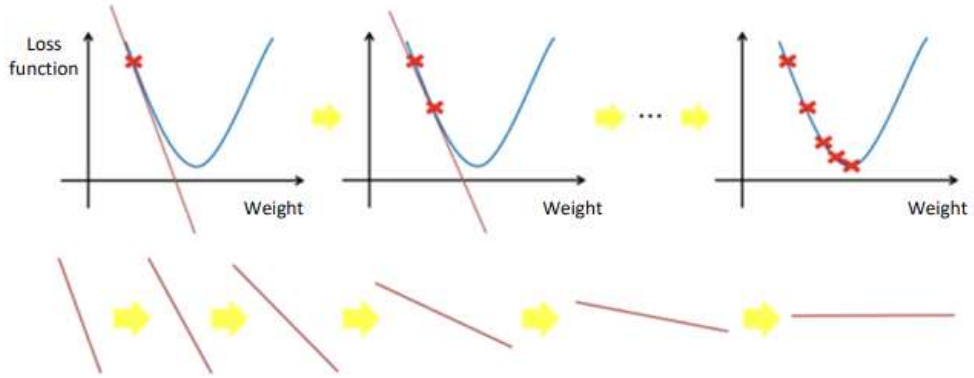
□ 2개 Perceptron에 대한 가중치 행렬: $\Theta = \{U^1, U^2\}$



- **경사하강법 -> 평균 제곱 오차의 최저점을 찾아줌**

  : 함수의 기울기를 구하여(<- 미분) 기울기가 낮은 쪽으로 계속 이동시켜 극값(최적값)에 이를 때까지 반복하는 과정 / 해당 함수의 최소값 위치를 찾기 위해 비용함수(Cost function)의 경사 반대 방향으로 정의한 Step Size(학습률)를 가지고 조금씩 움직여 가면서 최적의 파라미터를 찾으려는 방법

- 경사는 파라미터에 대해 편미분한 벡터를 의미



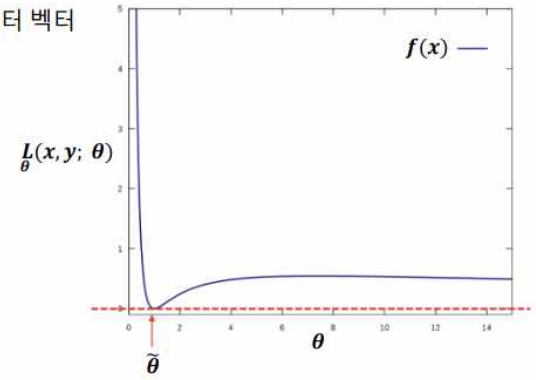
- **Deep Learning Optimization**
- Loss function (= Cost function = Objective function = Energy function , 손실함수)
  : 지도학습 시 알고리즘이 예측한 값(출력값)과 실제 정답의 차이를 비교하기 위한 함수
  = 학습 중에 알고리즘이 얼마나 잘못 예측하는지 정도를 확인하기 위한 함수

  - 특징: 출력값과 실제 정답이 일치할수록 손실함수의 값은 작아짐
    성능 척도와는 다른개념(모델의 성능을 평가하는 지표가 아님)

- Loss function 수식

- L : 손실 함수
- $argmin$: arguments of minimum, 목적 함수를 최소화하는 입력값을 찾는 역할
- x : 학습 데이터의 입력값, x 로 얻어낸 예측값 ($\hat{y}$)은 정답 (y)과 비교하게 됨
- y : 학습 데이터의 정답
- θ : 알고리즘 학습 시 사용되는 모든 파라미터 벡터
- $\tilde{\theta}$: 추정된 최적의 파라미터

$$\tilde{\theta} = \underset{\theta}{argmin}\, L(x, y;\, \theta)$$



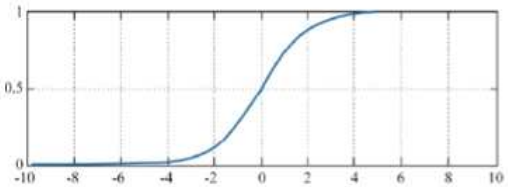
최적화 : 매개변수 (가중치, bias 등)를 조절하여 손실함수 값을 최저로 만드는 과정
(=> 대표적인 최적화 방법 : 경사하강법)

- 평균제곱오차 (Mean Square Error, MSE)

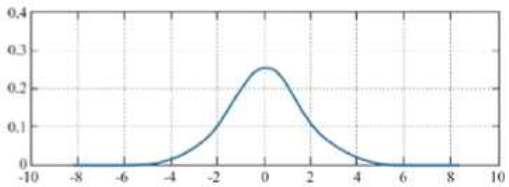
$$e = \frac{1}{2}\|\mathbf{y} - \mathbf{o}\|^2 = \frac{1}{2}(y - \sigma(wx + b))^2$$

=> 오차가 클수록 e값이 커짐 (=좋지 않은 모델)

- $wx + b$
  - 아래 그래프의 가로축의 값이 커지면 Gradient 가 작아짐



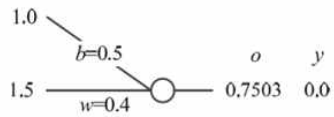
Sigmoid function



도함수

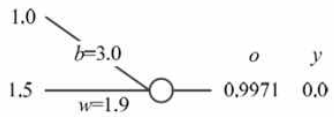
MSE가 목적함수로 부적절한 예시)

- (좌) e=0.2815



$$= \frac{1}{2}(0.0 - 0.7503)^2 = 0.281475045$$

(우) e=0.4971



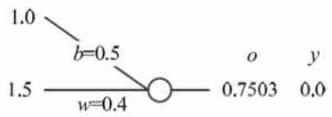
$$= \frac{1}{2}(0.0 - 0.9971)^2 = \mathbf{0.497104205}$$

오차 값이 더 크므로 좋지 않은 모델

$$e = \frac{1}{2}\|\mathbf{y} - \mathbf{o}\|^2 = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{\sigma}(\boldsymbol{wx} + \boldsymbol{b}))^2$$

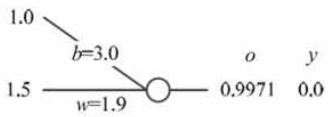
$$\frac{\partial e}{\partial w} = -(\boldsymbol{y} - \boldsymbol{o})\boldsymbol{x}\boldsymbol{\sigma}'(\boldsymbol{wx} + \boldsymbol{b})$$

$$\frac{\partial e}{\partial b} = -(\boldsymbol{y} - \boldsymbol{o})\boldsymbol{\sigma}'(\boldsymbol{wx} + \boldsymbol{b})$$



$\frac{\partial e}{\partial w} = 0.2109$

$\frac{\partial e}{\partial b} = 0.1406$

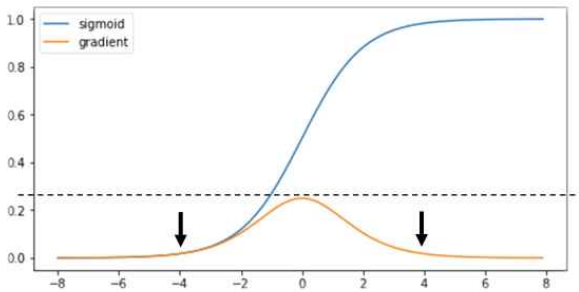


$\frac{\partial e}{\partial w} = 0.0043$

$\frac{\partial e}{\partial b} = 0.0029$

✓ Gradient를 계산해보면 왼쪽의 Gradient가 더 큼
✓ 더 많은 오류를 범한 상황이지만 MSE로 측정할 경우, 더 낮은 벌점을 받게 되는 상황

-> 시그모이드 함수를 사용할 경우
: 시그모이드 함수의 미분 최대치는 0.3 (< 1)
은닉층이 많아질수록 기울기는 점점 0에 가까워짐



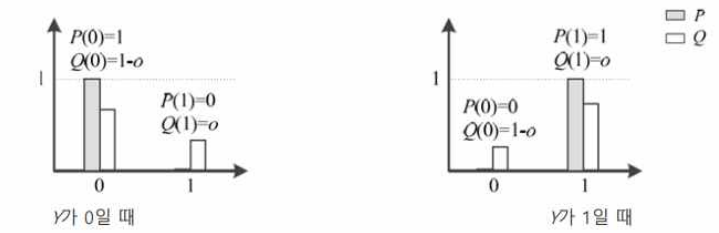
시그모이드 함수를 미분
한 값 최대치 0.3

활성화(activation)들이 [-4,4] 범위에 들어가지 않게 되면 전체 곱의 기울기는 소멸
대체 함수 => ReLU 함수 (시그모이드 함수와 모양이 비슷하고 경사가 소실되지 않음)

- 교차 엔트로피 (Cross Entropy)

교차 엔트로피 수식 $$H(P,Q) = -\sum_{y\in\{0,1\}} P(y)log_2Q(y)$$

=> Y가 0과 1일 때의 P와 Q의 확률분포
(P는 정답 레이블의 확률분포 , Q는 신경망 출력의 확률분포)



- $P(0) = 1 - y$ $Q(0) = 1 - o$
- $P(1) = y$ $Q(1) = o$

- 교차 엔트로피 목적함수

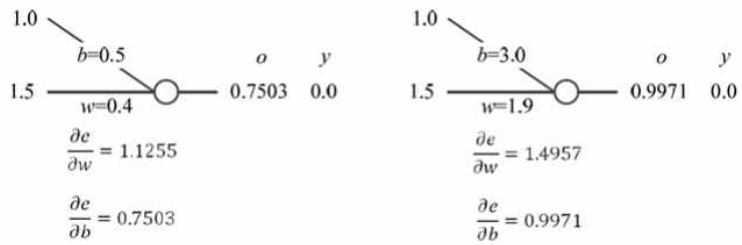
$$e = -(ylog_2o + (1-y)log_2(1-o)), \quad \text{이때,} \quad e = \sigma(Z)\text{이고,}\ Z = wx + b$$

- y가 1, o가 0.98일 때 (예측이 잘된 경우)
  - **오류** $e = -(1\ \log_2 0.98 + (1-1)\log_2(1-0.98)) = 0.0291$로서 낮은 값 (**오차가 작은 값**)
- y가 1, o가 0.0001일 때 (예측이 엉터리인 경우)
  - **오류** $e = -(1\ \log_2 0.0001 + (1-1)\log_2(1-0.0001)) = 13.2877$로서 높은 값 (**오차가 큰 값**)

도함수 측정

$$\begin{aligned}\frac{\partial e}{\partial w} &= -\left(\frac{y}{o} - \frac{1-y}{1-o}\right)\frac{\partial o}{\partial w}\\ &= -\left(\frac{y}{o} - \frac{1-y}{1-o}\right)x\sigma'(z)\\ &= -x\left(\frac{y}{o} - \frac{1-y}{1-o}\right)o(1-o)\\ &= x(o-y)\end{aligned} \longrightarrow \begin{aligned}\frac{\partial e}{\partial w} &= x(o-y)\\ \frac{\partial e}{\partial b} &= (o-y)\end{aligned}$$

gradient(기울기)를 계산해보면, 오류가 더 큰 오른쪽에 더 큰 벌점(gradient) 부과

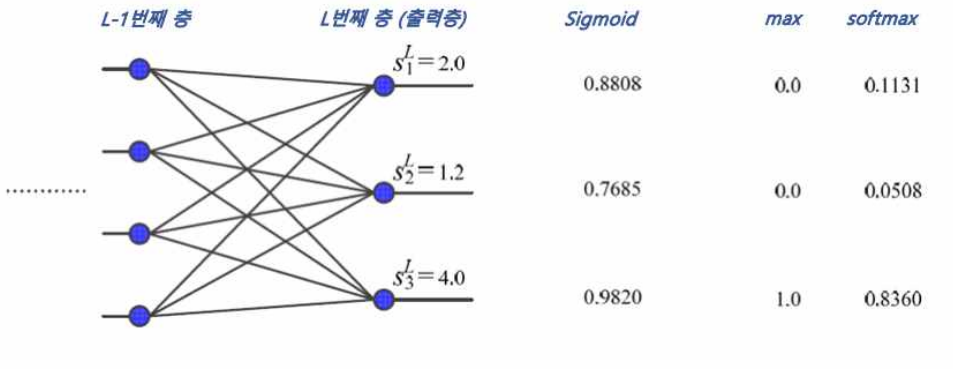


=> 교차 엔트로피를 목적함수로 사용하여 느린 학습문제를 해결

- Softmax

: 여러 개의 연산 결과를 정규화하여 모든 클래스의 확률값의 합이 1 (출력의 총합이 1)
max를 모방 (출력노드의 중간 계산 결과 에서 최댓값은 더욱 활성화하고 작은 값은 억제)

$$o_j = \frac{e^{S_j}}{\sum_{i=1,c} e^{S_j}}$$



- 로그우도 목적함수 (Log Likelihood)

: 출력층의 활성화함수로 softmax가 사용됐을 경우, 목적함수로 사용
Multi-class classification
모든 노드를 고려하는 평균 제곱오차나 교차 엔트로피와는 달리, 정답에 해당하는 노드만 확인

$$e = -log_2 o_y$$

- Softmax와 로그우도

: Softmax는 최댓값이 아닌 값을 억제하여 0에 가깝게 만든다는 의도 내포 -> 학습 샘플이 알려주는 부류에 해당하는 노드만 보겠다는 로그우도와 잘 어울림 => 둘을 결합하여 사용

- **Performance improvement**
- 딥러닝의 최적화 기법

(1) 데이터 전처리

데이터 셋 내 각 feature의 규모 문제

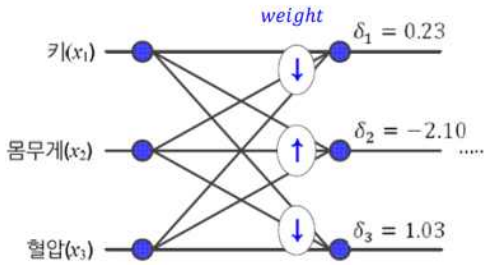
=> ex) 건강 관련 데이터

키 : 1.885m와 1.525m 차이는 33cm -> 미터로 기준값 설정하면 차이는 0.33

몸무게 : 65.5kg 과 45.0kg 차이는 20.5kg

=> 키와 몸무게 feature는 대략 62배 규모차이

- $-\delta_j z_i$가 gradient 이기 때문에 첫 번째 특징에 연결된 가중치는 두 번째 특징에 연결된 가중치에 비해 대략 62 배 느리게 학습됨



=> 1) 정규화 (Normalization)

: 데이터 값들을 공통된 규정/규범의 간격에 맞게 변경

- Min-Max 정규화 : 데이터들을 0~1사이의 공통간격으로 재배치 (최대값=1, 최소값=0)

$$\text{MinMax} = \frac{\text{data} - \text{data.min}}{\text{data.max} - \text{data.min}}$$

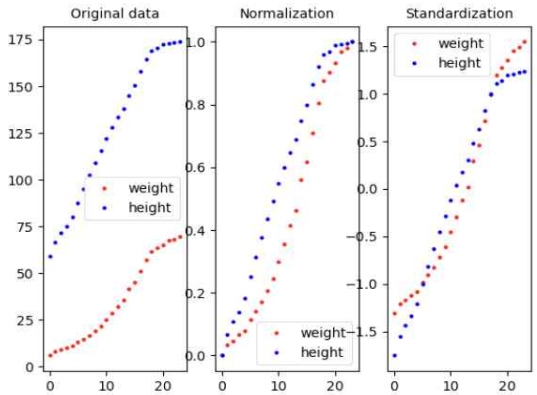
2) 표준화 (Standardization)

: 데이터 값들을 공통된 근거나 기준인 척도를 변경

- Z-score : 데이터에서 데이터의 평균을 뺀 값을 표준편차로 나눔

$$\text{Zscore} = \frac{\text{data} - \text{data.mean}}{\text{data.std}}$$

(데이터 간격의 크기는 달라질 수 있어도, 데이터 간격의 의미는 달라지지 않음)



3) 명칭값(Norminal value) one-hot 코드로 변환

: Norminal value 는 거리 개념이 없음

one-hot 코드는 값의 개수만큼 비트를 부여

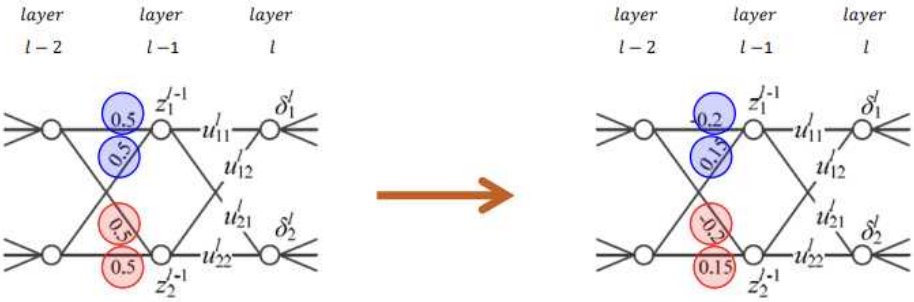
- Ex) 성별의 남(1)과 여(2), 체질의 태양인(1), 태음인(2), 소양인(3), 소음인(4)
- 성별은 2비트, 체질은 4비트 부여

$$(1.755, 65.5, 122, 1, 3) \rightarrow (1.755, 65.5, 122, \underbrace{1,0}_{\text{성별}}, \underbrace{0,0,1,0}_{\text{체질}})$$

(2) 가중치 초기화

가중치의 초기값을 무엇으로 설정하느냐가 신경망 학습에 영향을 끼치게 됨 (각 뉴런의 가중치를 기반으로 error를 결정하기 때문)

-> 정확한 모델을 얻기 위해 가장 작은 error가 도출되도록 해야함



- 대칭적 가중치에서는 z_1^{l-1} 과 z_2^{l-1}가 같은 값이 됨
- $-\delta_j z_i$ 가 gradient 이기 때문에 u_{11}^l과 u_{12}^l이 같은 값으로 갱신됨
- 두 노드가 같은 일을 하는 중복성 발생하게 됨

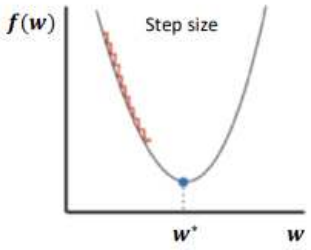
→ 난수로 초기화함으로써 대칭 파괴

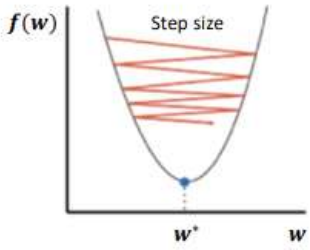
(3) **Gradient Descent** & Momentum (탄력, 가속도)

- Gradient Descent (경사하강법)
  : 미분값(기울기)이 최소가 되는 지점에 알맞은 가중치 매개변수를 찾아냄
  비용함수의 경사 반대방향으로 정의된 Step size에 따라 움직이면서 최적의 파라미터를 찾으려 함 (경사하강법에서 step size 중요)

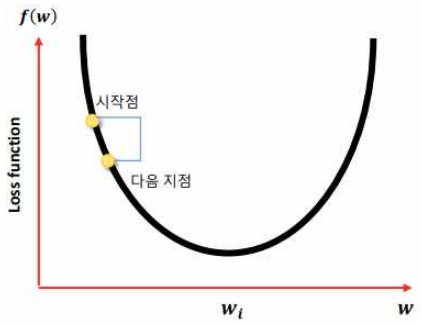
- 학습률 (step size)이 너무 작을 경우
  : 알고리즘이 수렴하기 위해 반복해야 하는 값이 많음 -> 학습시간이 오래 걸림
  지역 최소값(local minimum)에 수렴할 수 있음

- 학습률 (step size)이 너무 클 경우
  : 학습시간은 적게 걸리지면 전역 최소값(global minimum)을 가로질러 반대편으로 건너뛰어 최소값에서 멀어질 수 있음





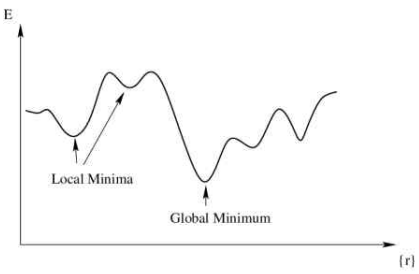
□ Gradient Descent



$$x_o = x_o - \eta \frac{\partial f}{\partial x_0}$$

η: Step size (=learning rate) $\frac{\partial f}{\partial x_0}$: 기울기

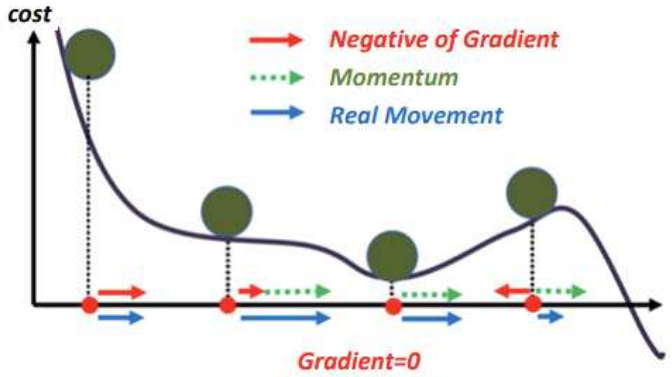
- 경사하강법은 현재 위치에서 기울기를 사용하기 때문에 local minimum에 빠질 수 있음
- 무작위 초기화(random initialization)로 인해 알고리즘이 global minimum이 아닌 local minimum에 수렴할 수 있음



(3) Gradient Descent & **Momentum (탄력, 가속도)**

- Momentum : 학습 방향이 바로 바뀌지 않고, 일정한 방향을 유지하려는 성질
  같은 방향의 학습이 진행된다면 가속을 가지며 더 빠른 학습을 기대할 수 있음

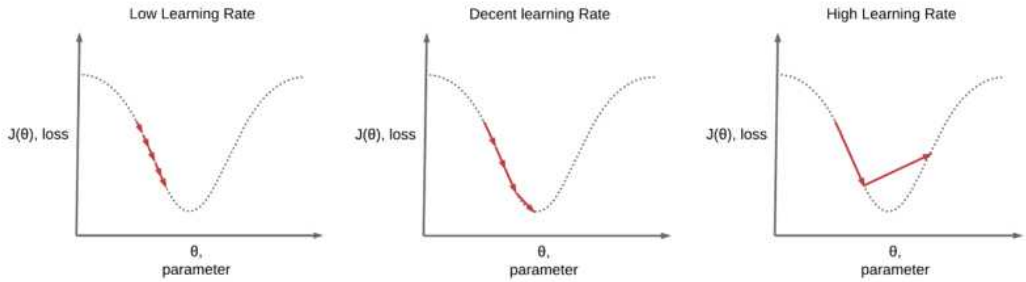
- Movement = Negative of Gradient + Momentum



기울기에 관성을 부과하여 작은 기울기는 쉽게 넘어갈 수 있도록 만들어 줌
공의 관성을 이용하여 쉽게 넘어갈 수 있게 하여 지역 최소값을 탈출
(모멘텀을 사용하지 않으면 기울기가 매우 작은 구간을 빠져나오는데 아주 오랜 시간이 걸림)

(4) 적응적 학습률 (Adaptive Methods)

- 학습률(step size)(Momentum)의 중요성
  학습률이 너무 작으면 학습에 많은 시간 소요
  학습률이 너무 크면 진동할 가능성이 높음 -> 오버슈팅에 따른 진자현상

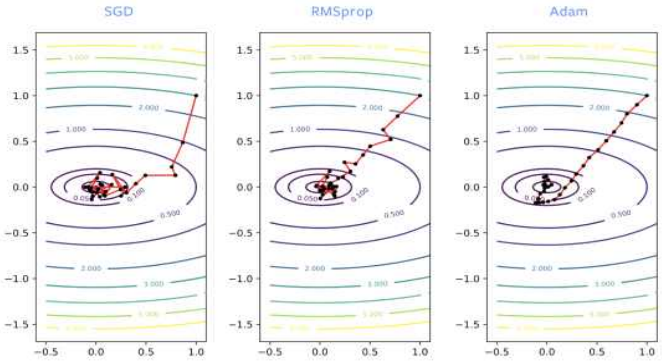


- 적응적 학습률
  Momentum (운동량) : 질량 * 속도
  신경망에서는 질량을 1로 가정 -> 속도만 나타냄
  적응적 학습률은 매개변수마다 자신의 상황에 따라 학습률을 조절해 사용

1) SGD : 네트워크 상 모든 개별 가중치는 동일한 학습률 사용
2) Adagrad : 이전 gradient를 누적한 정보 벡터 r을 이용,
   이전 gradient 누적값이 클 때 -> 다음에 파라미터가 조금만 이동
   작을 때 -> 많이 이동

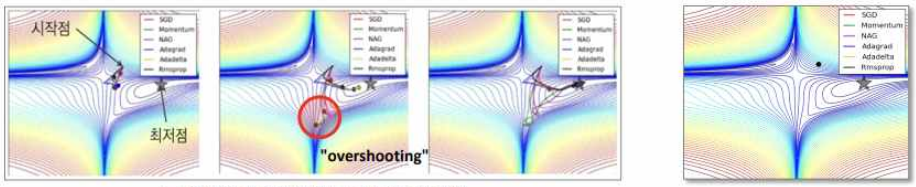
3) RMSprop : 이전 gradient를 누적할 때 오래된 것의 영향을 지수적으로 줄이기 위해 가중 이동평균 기법을 사용하여 Adgrad를 개선한 기법

4) Adam : RMSProp에 모멘텀을 적용하여 RMSProp를 개선한 기법 (가장 일반적으로 사용)

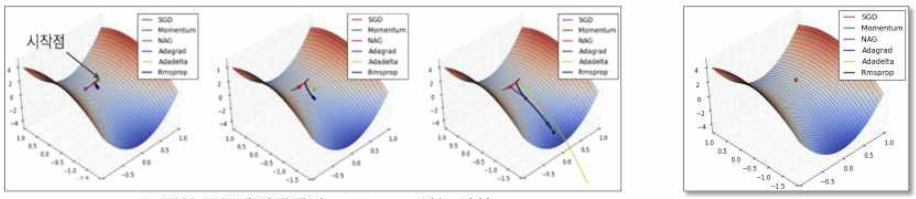


SGD : 대칭을 깨는데 매우 어려움을 겪고 상단에 고정

RMSprop : 안장 방향에서 매우 낮은 기울기, RMSprop 업데이트의 분모 항으로 인해 이 방향을 따라 효과적인 학습 속도가 증가



(a) 중앙으로 급강하하는 절벽 (협곡) 지형

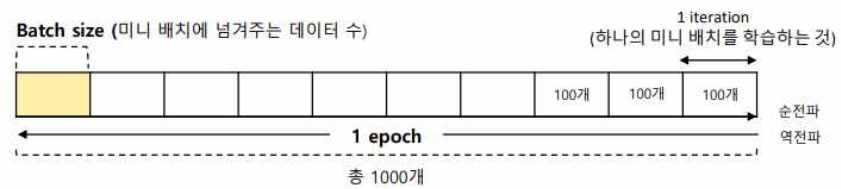


(b) 중앙 부근에 안장점이 saddle point 있는 지형

(5) Epoch (에포크)

: 훈련 데이터 셋에 포함된 모든 데이터들이 한 번씩 모델을 통과한 횟수
학습 데이터 전체를 한번 학습하는 것

1 epoch : 전체 학습 데이터 셋이 한 신경망에 적용되어 순전파와 역전파를 통해 신경망을 한 번 통과했다는 의미 (10 epoch = 학습 데이터 셋을 10회 모델에 학습)
epoch 값이 높음 -> 다양한 무작위 가중치로 학습하였다는 의미
지나치게 epoch이 올라감 -> 학습 데이터 셋에 대한 과적합 발생

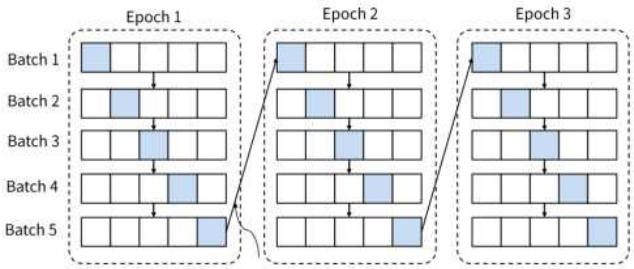


- Batch size

: 연산 한번에 들어가는 데이터 크기

Batch : 단일 반복에서 기울기를 계산하는데 사용하는 data의 총 개수 (gradient를 구하는 단위)

mini Batch : 1 Batch size에 해당하는 데이터 셋



- 배치 사이즈가 너무 큰 경우 : 한번에 처리해야 할 데이터의 양이 많음
  -> 학습 속도가 느려지고, 메모리 부족 문제 발생 가능
- 배치 사이즈가 너무 작은 경우 : 적은 데이터를 대상으로 가중치를 업데이트하고 이 업데이트가 자주 발생하므로 훈련이 불안정

(6) Batch Normalization

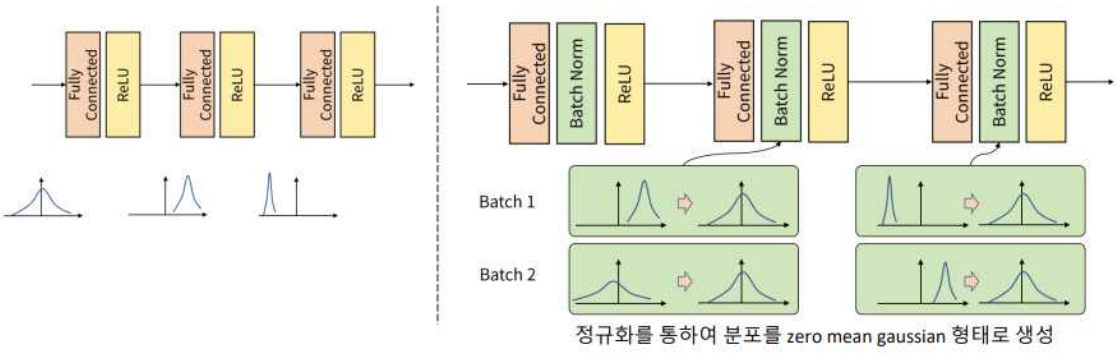
- Internal Covariate Shift 현상 (내부 공변량 이동)

: 학습 과정에서 계층 별로 입력의 데이터 분포가 달라지는 현상

Batch 단위로 학습을 하게 되면 발생하는 문제점

한 레이어마다 입력과 출력을 가지고 있는데 이 레이어들끼리 covariate shift(공변량 이동) 발생

레이어가 깊어질수록 분포 형태가 더 심하게 변함 -> Batch Normalization을 이용



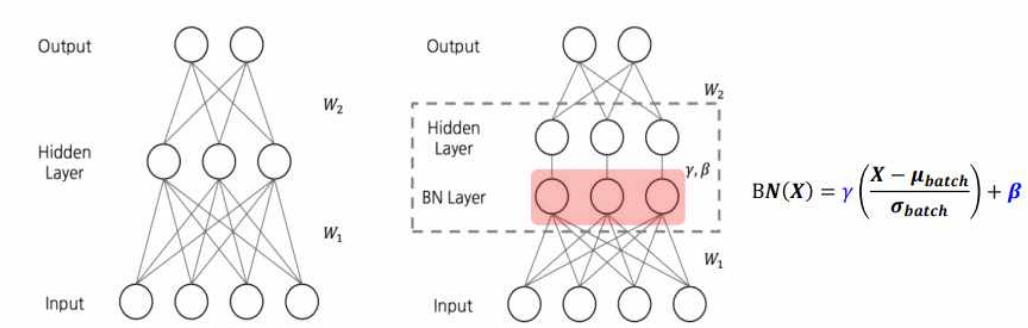
- Batch Normalization

: 학습 과정에서 각 배치 단위 별로 데이터가 다양한 분포를 가지더라도 각 배치별로 평균과 분산을 이용해 정규화 하는 과정 => Batch마다 입력값의 범위를 스케일링 하는 과정

- 장점 :

1) 데이터 Scale 통일 : 모든 레이어의 feature가 동일한 scale이 되도록 함
2) 활성화 함수 맞춤형 분포 변화 : 추가적인 스케일링( )과 편향( )을 학습함으로써

활성화 함수 종류에 맞게 적합한 분포로 변환 가능



3) 학습률을 높게 설정할 수 있어서 학습속도가 개선됨

4) 과적합(overfitting)억제 / 기울기 소실(Gradient vanishing) 방지

- 배치 정규화 알고리즘

Mini-batch 평균 $\mu_B \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$

- 미니배치 $B = x_1, x_2, \dots, x_n$ 을 평균이 0, 분산이 1인 표준정규분포를 따르는 $\hat{B} = \widehat{x_1}, \widehat{x_2}, \dots, \widehat{x_n}$로 정규화

Mini-batch 분산 $\sigma^2{}_B \leftarrow \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_B)^2$

- 정규화에서 $x_i - \mu_B$ 가 평균을 0으로 설정

Normalization $\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2{}_B + \epsilon}}$

- $\sqrt{\sigma^2{}_B + \epsilon}$ 가 분산을 1로 설정
- 표준편차로 나누면 분산이 1이 됨
- ϵ은 0으로 나뉘어지는 것을 방지하기 위한 상수

Scale and shift $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta(x_i)}$

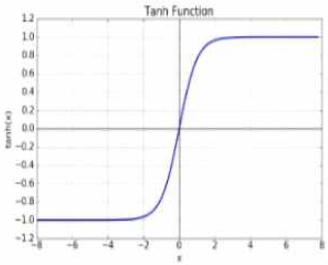
- 배치 정규화 계층마다 데이터에 대한 고유한 스케일링(Scaling, γ), 쉬프팅(Shifting, β) 수행

(7) ReLU (Rectified Linear Unit) 활성함수

- tanh(x) 함수를 사용하면 미분값이 범위가 확장
  고차원 데이터를 다룰 경우에는 값이 커질 수 있어 다시 경사가 소실될 수 있음
  복잡한 데이터일수록 고차원일 경우가 많은데 이를 회피할수 있는 활성화 함수
  -> ReLU 함수

$$\varphi(x) = \begin{cases} x, x > 0 \\ 0, x \le 0 \end{cases}$$

$$\varphi(x) = \max(0, x)$$



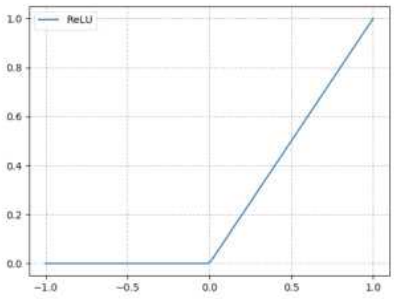
$$f'(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$$

ReLU 함수의 도함수 (미분한 식)

- x (입력값)가 0보다 작을 때 (음수)는 모든 값을 0으로 출력
- x (입력값)가 0보다 클 때 (양수)는 입력값을 그대로 출력
- x가 0보다 크기만 하면 미분값이 1이 되기 때문에 여러 은닉층을 거쳐도 맨 처음 층까지 사라지지 않고 남아있음
- 딥러닝의 발전에 기여!!!

$$\varphi(x) = \begin{cases} x, x > 0 \\ 0, x \leq 0 \end{cases}$$

$$\varphi(x) = \max(0, x)$$



$$f'(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases}$$

ReLU 함수의 도함수
(미분한 식)

- ReLU 함수의 도함수는 x가 아무리 커져도 1을 반환하므로 경사가 소실되지 않음
- 시그모이드 함수나 쌍곡탄젠트 함수에 비교해 학습 속도가 빠름
- ReLU와 ReLU의 도함수는 단순한 식으로 표현되기 때문에 빠르게 계산이 가능
- 단점으로는 x≦0일 때는 함수값도 경사도 0이기 때문에 ReLU를 활성화 함수로 사용한 신경망 모델의 뉴런 중 활성화되지 못한 뉴런은 학습 동안 활성화가 되지 않는 문제가 있음

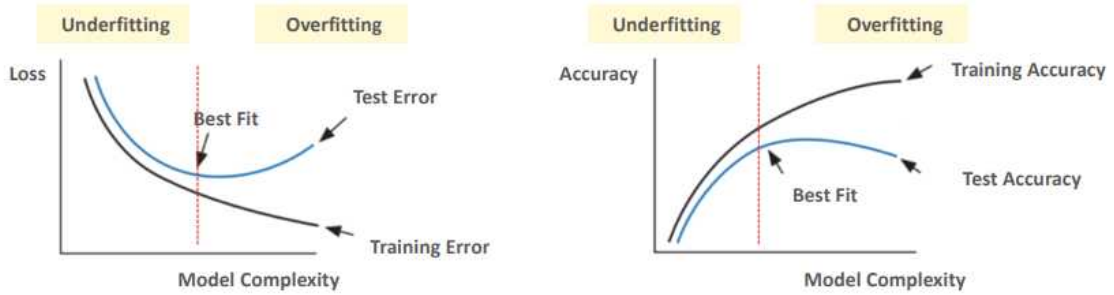
(8) Stochastic Pooling

- **Regulation principle (규제 원칙)**

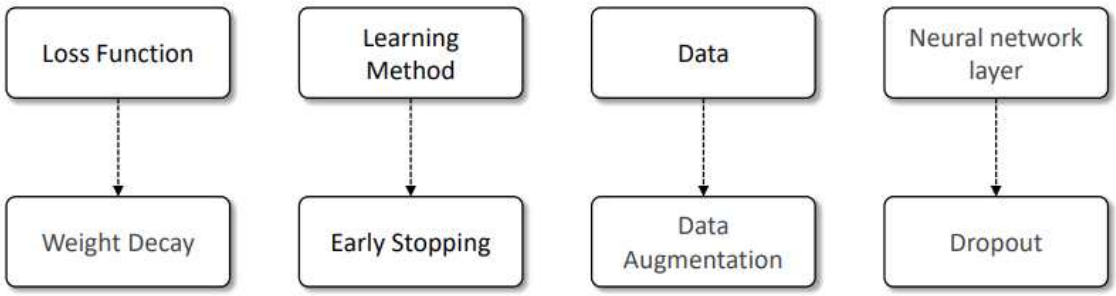
- 규제의 필요성

  과대적합에 빠지는 이유 : 학습 모델의 용량에 따른 일반화 능력

  Training Dataset을 단순히 암기하는 과대적합에 주의해야 함



- 과대적합을 피하는 전략 : 학습 과정에서 여러 규제 기법 적용



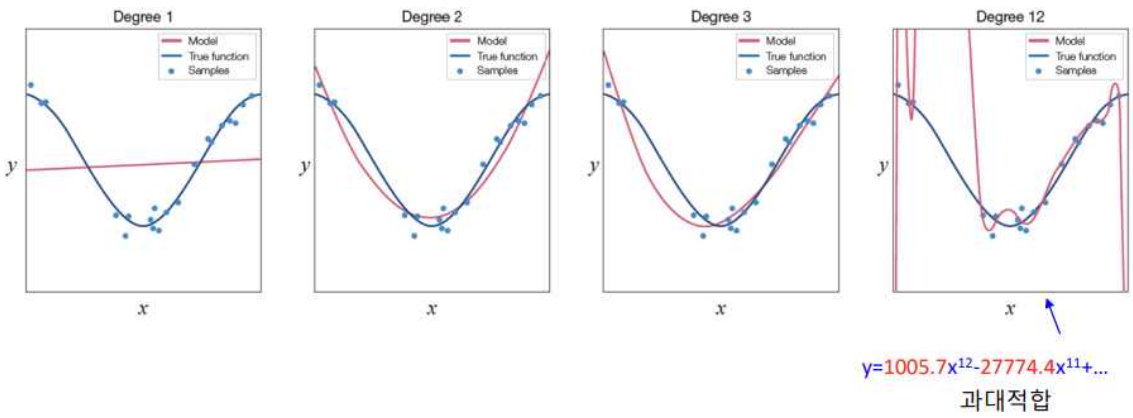
(1) 가중치 감수 (Weight Decay)

- 과대적합에서는 가중치 값 ($\boldsymbol{\theta}$) 이 아주 큰 현상이 나타남

  가중치 감소 -> 성능을 유지한 채로 가중치 크기 ($\boldsymbol{\theta}$)를 낮추는 규제 기법

  모델의 weight의 제곱합 (L2 Norm)을 패널티 텀으로 주어 (= 제약을 걸어) loss 최소화

  = L2 penalty



- 가중치 감소

$$\underbrace{\boldsymbol{Loss}(\boldsymbol{\theta};\mathbb{X},\mathbb{Y})}_{\text{규제를 적용한 목적함수}} = \underbrace{Loss\ (\boldsymbol{\theta};\mathbb{X},\mathbb{Y})}_{\text{목적함수}} + \underbrace{\frac{1}{2}\gamma\|\boldsymbol{\theta}\|^2}_{\text{규제 항}}$$

- 오차함수에 가중치의 제곱합 (Norm의 제곱)을 더한 뒤, 이를 최소화
- 감마( )는 이 규제화의 강도를 제어하는 파라미터
- 감마를 추가하여 가중치는 자신의 크기에 비례하는 속도로 항상 감쇠하도록 업데이트
- 가중치 $\boldsymbol{\theta}$가 커지게 되면 R항(규제항)이 커지게 되고 결과적으로 손실함수 J가 증가
- 학습 알고리즘은 손실 함수가 작아지도록 학습하므로 R항은 가중치의 크기에 제약을 가하는 역할을 해야함
- 규제항 R은 가중치를 작은 값으로 유지하므로 모델의 용량을 제한 하는 역할
- 규제항은 훈련집합과 무관하며, 데이터 생성과정에 내재한 사전 지식에 해당
- 규제항 R 로 L2놈이나 L1놈을 사용 (큰 가중치에 패널티, 작은 가중치 유지)

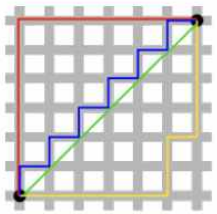
- 규제항 R로 가장 널리 쓰이는 것은 L2놈 (Norm, 차수)이며 이를 가중치 감쇠기법이라 함
  (norm은 크기의 일반화로 벡터의 크기를 측정하는 방법)

$$\begin{aligned} x &= [1,2,3,4,5] \\ \|x\|_1 &= (|1|+|2|+|3|+|4|+|5|) \\ &= 15 \end{aligned}$$

L1 Norm은 벡터의 요소에 대한 절댓값의 합

$$\begin{aligned} x &= [1,2,3,4,5] \\ \|x\|_2 &= \sqrt{(|1|^2+|2|^2+|3|^2+|4|^2+|5|^2)} \\ &= \sqrt{1+4+9+16+25} \\ &= \sqrt{55} \\ &= 7.4161 \end{aligned}$$

L2 Norm은 유클리드 공간에서 벡터 크기 계산



- **L1 Norm**
  - 빨간색, 파란색, 노란색 선으로 표현
  - 여러 가지 path
- **L2 Norm**
  - 오직 초록색 선으로만 표현
  - Unique shortest path

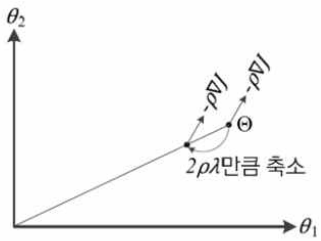
- 매개변수 갱신하는 수식

$$\boldsymbol{Loss}(\boldsymbol{\theta};\mathbb{X},\mathbb{Y}) = Loss\ (\boldsymbol{\theta};\mathbb{X},\mathbb{Y}) + \frac{1}{2}\gamma\|\boldsymbol{\theta}\|^2$$

Loss에 대한 미분

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\left(\frac{\partial DataLoss}{\partial \boldsymbol{\theta}} + \lambda\boldsymbol{\theta}\right)$$

$$= \boldsymbol{\theta}(\mathbf{1} - \eta\lambda) - \eta\left(\frac{\partial DataLoss}{\partial \boldsymbol{\theta}}\right)$$



최종 해를 원점 가까이 당기는 효과
(즉, 가중치를 작게 유지함)

-> 손실되는 데이터 없게, 불필요한 데이터ㄲ지 학습할 필요 없게

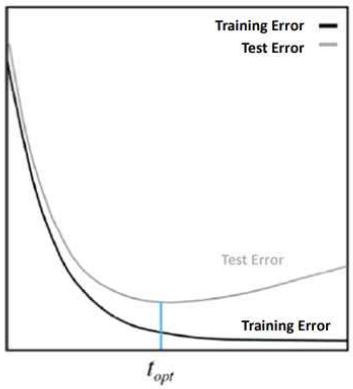
미분을 했을 때, 기본 Dataloss에 θ의 람다배 만큼을 더하게 되므로 가중치 값이 그만큼 보정

θ(1-학습률*람다)가 되기 때문에 weight가 아주 작은 factor에 비례하여 감소함

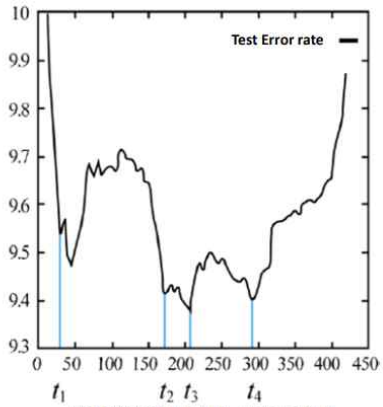
(2) 조기 종료 (Early Stopping)

- 일정 시간이 지나면 과대적합(overfitting)현상이 나타남 -> 일반화 능력 저하(Train data 에 대해서는 잘 맞추고 test data에 대해서는 맞추지 못함)

=> 훈련 데이터를 단순히 암기하기 시작



검증집합의 오류가 최저인 점 t_{opt} 에서 학습을 멈춤



실제 데이터에 나타나는 지그재그 현상

- 조기 종료(early stopping) : 모델이 과적합되기 전에 훈련을 멈추는 정규화 기법

훈련 중 주기적으로 성능검증 수행 -> 성능이 더 좋아지지 않으면 과적합이라 판단 -> 훈련 멈춤

Epoch 단위로 성능 검증 수행, Epoch 보다 자주 검증해야 할 때는 Batch 실행단위로 검증하기도 함

-조기 종료 기준 : 모델의 성능이 바로 향상하지 않는다고 종료해버리면 학습이 제대로 되지 않을 수 있기 때문에 일시적 변동이 아닌 지속적인 정체 또는 하락에 의한 판단이 들었을 때 종료

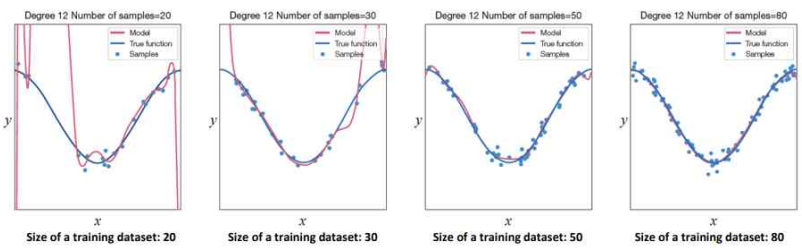
(3) 데이터 증대 (Data Augmentation)

: 과대 적합을 방지하는 가장 확실한 방법 = 큰 훈련 집합 사용

하지만 대부분의 상황에서 데이터를 늘리는 일은 많은 비용이 소모

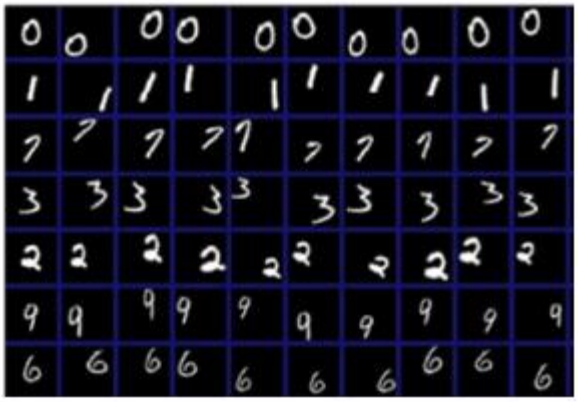
딥러닝에서는 주어진 데이터를 인위적으로 늘리는 데이터 증대를 적

(ex > 영상을 이동, 회전, 좌우반전 / 명암 조정 등



데이터가 커지면 과대적합이 자연스럽게 사라지는 현상 (예시)

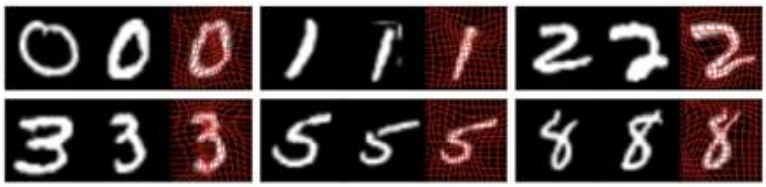
1) MNIST에 Affine 변환 (이동-Translation, 회전- rotation, 확대-zoom, 반전 -invert, 전단 - shearing을 적용)



2) 모핑 (Morphing)을 이용한 변형

: 비선형 변환으로서 어파인 변환에 비해 훨씬 다양한 형태의 확대

학습기반 - 데이터에 맞는 비선형 변환 규칙을 학습

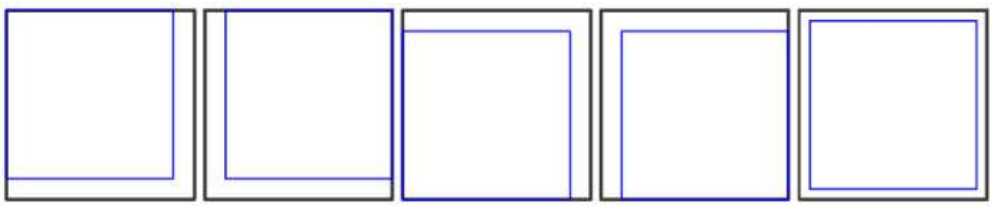


3) 자연영상 확대

✓ 256*256 영상에서 224*224 영상을 1024장 잘라내어 이동 효과. 좌우 반전까지 시도하여 2048배로 확대

✓ PCA를 이용한 색상 변환으로 추가 확대

✓ 예측 단계에서 5장 잘라내고 좌우 반전하여 10장을 만든 다음 앙상블 적용



예측 단계에서 영상 잘라내기

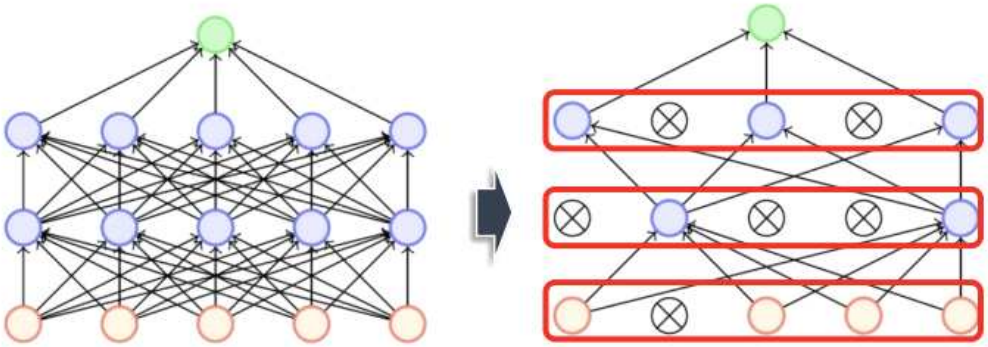
4) 잡음을 섞어 확대

: 입력 데이터에 잡음을 섞는 기법

은닉 노드에 잡음을 섞는 기법(고급 특징 수준에서 데이터를 확대하는 셈)

(4) Dropout

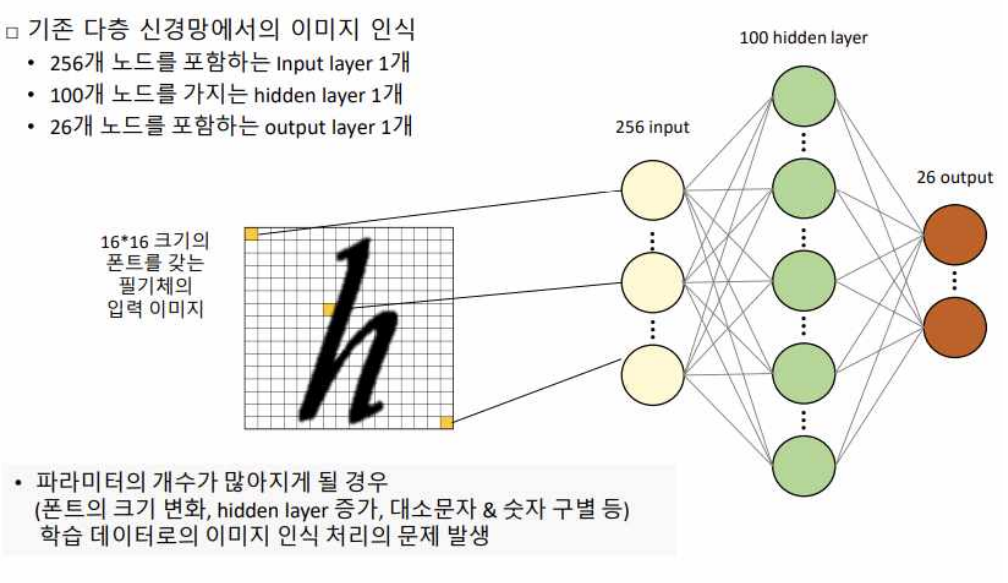
: 신경망 전체를 다 학습시키지 않고 일부 노드만 무작위로 골라 학습시키는 기법
일정 비율의 가중치를 임의로 선택하여 불능으로 만들고 학습하는 규제 기법
학습하는 중간에 일정 비율로 노드들의 출력을 0으로 만들어 신경망의 출력을 계산함
(특정 뉴런의 확률 p를 0으로 바꾸는 것)



□ Dropout 적용 순서: ReLU등의 Activation 함수 적용 후, Pooling 이전일때가 가장 적절

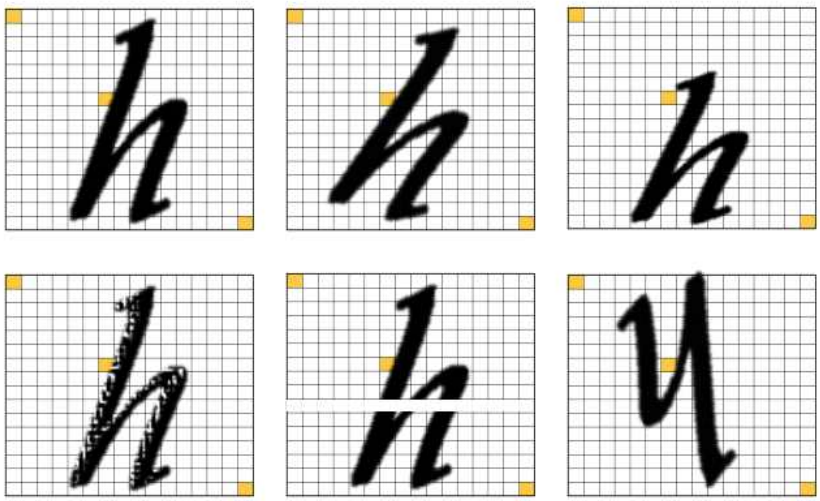
- Convolution -> Batch Normalization -> Activation -> Dropout -> Pooling

- **CNN (Convolutional Neural Network, 합성곱 신경망)**
- 기존 다층 신경망 : 영상(이미지) 데이터 기반의 인식 알고리즘 적용의 한계 존재

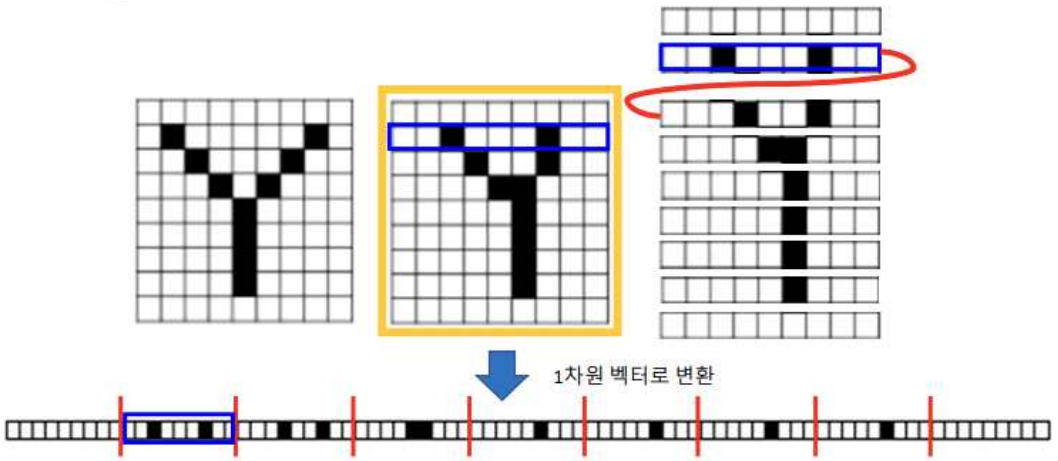


- Image Topology (size, spin & Angle, shape, color 등)을 고려하지 않음

-> 이미지 변형이 조금이라도 생기게 될 경우 해당 데이터로의 학습 없이는 결과가 좋지 않음



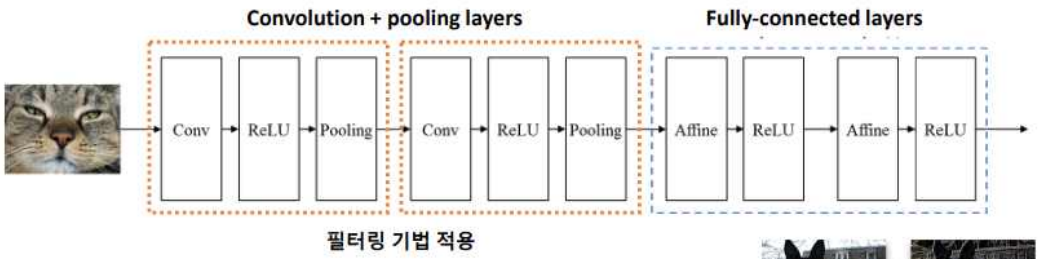
ex) Y라는 글자 이미지를 다층 퍼셉트론으로 분류
이미지를 1차원 텐서인 벡터로 변환하고 다층 퍼셉트론의 입력층으로 사용



- 1차원 벡터로 변환 전 가지고 있던 공간적인 구조 정보가 유실됨
- 공간적인 구조 정보
  - 거리가 가까운 픽셀들 간의 값의 연관성
  - 어떤 픽셀의 값은 비슷하거나 다른 경우들이 존재함
- 공간적인 구조 정보를 보존하면서 이미지 데이터를 학습할 방법이 요구됨

- CNN
: 필터링 기법을 인공신경망에 적용하여 이미지를 더욱 효과적으로 처리하기 위한 알고리즘
기존의 필터링 기법은 고정된 필터를 이용하여 이미지 처리 -> CNN은 행렬로 표현된 필터의 각 요소가 데이터 처리에 적합하도록 자동으로 학습되도록 함

합성곱 계층과 풀링 계층이라고 하는 새로운 층을 fully-connected 계층 이전에 추가
원본 이미지에 필터링 기법을 적용한 뒤 필터링 된 이미지에 대해 분류 연산이 수행되도록 구성

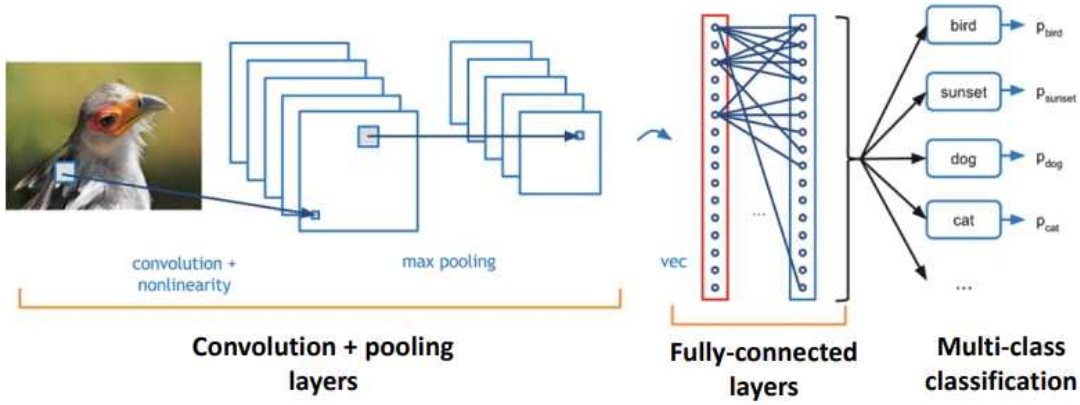


* 합성곱 계층 : 이미지를 분류하기 위해 filter를 통해 중요한 특징 정보를 추출하는 계층
* 풀링 계층 :이미지의 국소적인 부분들을 하나의 대표적인 스칼라 값으로 변환(Abstract feature 추출)

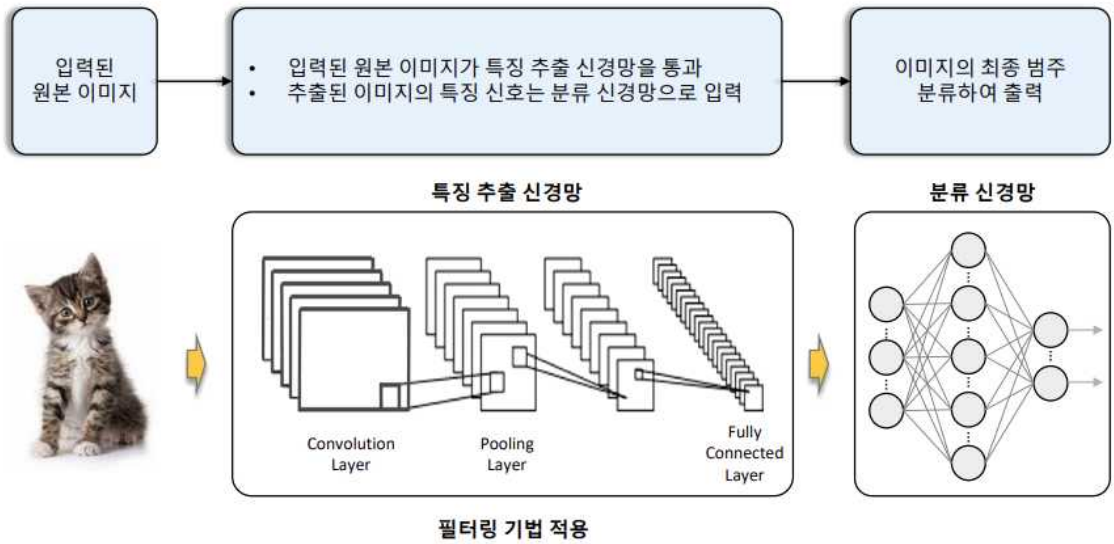
- 컨브넷 (ConvNet)

: 뇌의 시각피질의 이미지를 처리하고 인식하는 원리를 차용한 신경망
영상 또는 사진에서 객체가 어느 범주로 분류되는지 인식
영상인식에서 사용되는 컨브넷의 출력층은 다범주 분류(Multi-class Classification) 신경망

- 어떤 영상인식 기술이든 원본 이미지를 바로 사용하지 않음
- 원본 이미지의 고유한 특징이 잘 드러나도록 가공된 이미지를 사용



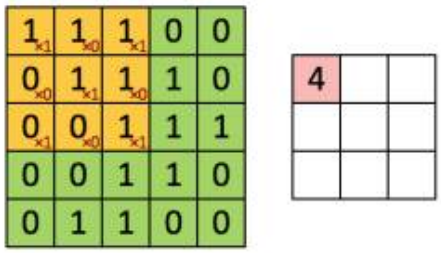
- 컨브넷은 특징 추출 신경망이 깊을수록 영상 인식 성능도 좋아짐
신경망의 계층 구조가 깊어지면 학습을 시키는데 어려움이 발생



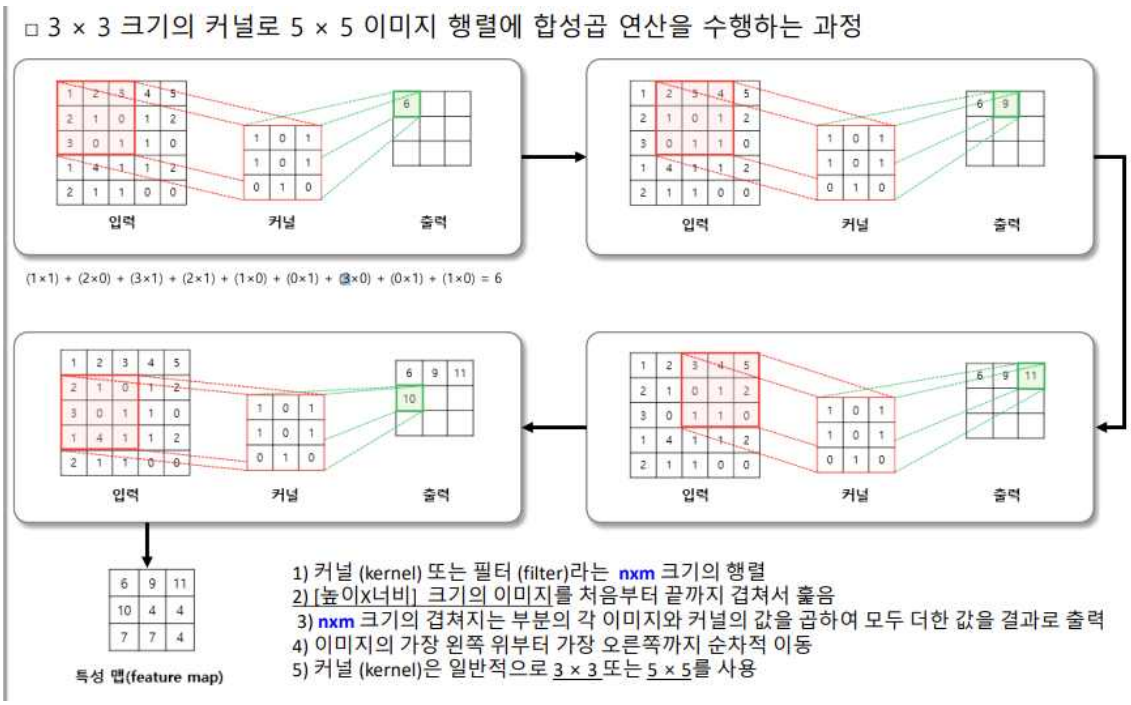
- **CNN Architecture**
- Convolution Layer (합성곱 계층)
 ; 입력 이미지(데이터)로부터 합성곱 연산을 통해 중요한 정보를 추출해 내는 필터 역할 수행
  새로운 이미지, 특징맵(feature map)를 만들어 내는 역할

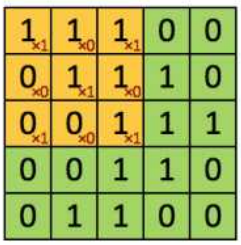
-> 어떤 합성곱 필터를 사용하느냐에 따라 합성곱 계층에서 추출해 내는 특징이 결정됨

  합성곱 계층은 일반적인 신경망의 계층과는 다른 구조와 방식으로 작동
- 합성곱 계층의 노드
  : 연결 가중치와 가중합의 개념이 아님
   입력 이미지를 다른 이미지로 변환하는 필터 (커널, kernel)로 이미지 처리

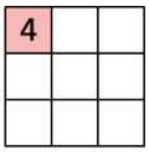


*Convolution filter or Kernel (합성곱 필터 or 커널)
  : 입력 이미지를 다른 이미지로 변환하는 필터
   합성곱 필터로 입력 이미지를 처리하면 특징 맵을 얻을 수 있음





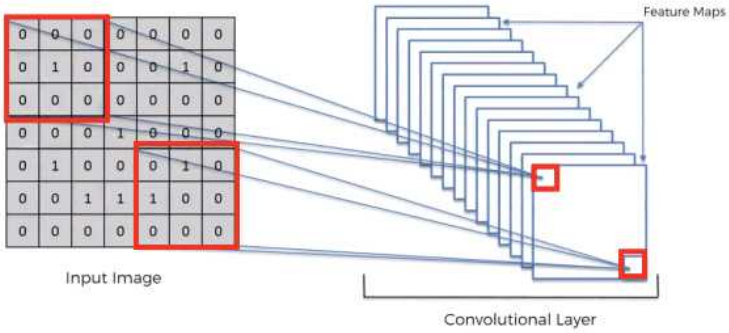
Image

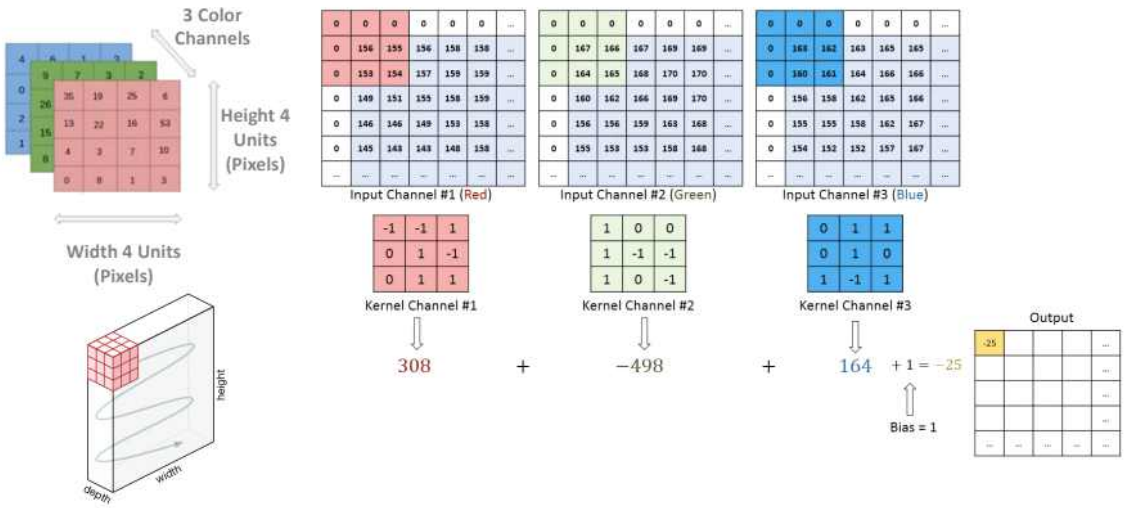


Convolved Feature

Image

Convolved Feature



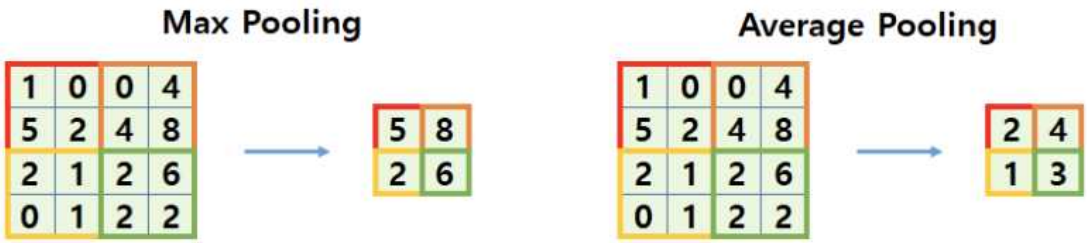


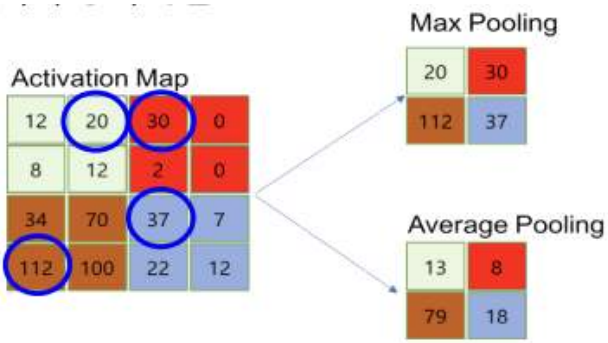
- Pooling Layer (풀링 계층)
  : 입력 이미지의 특정 영역에 있는 픽셀을 묶어서 하나의 대표 픽셀로 축소
  -> 즉, 이미지의 차원을 축소하여 이미지의 크기를 줄이는 역할을 함
  CNN 내 앞 layer의 출력 feature map의 모든 데이터가 필요하지 않기 때문에 pooling layer 만 사용(추론을 위한 적당량의 데이터만 있어도 충분함 -> overfitting speed

- sub-sampling을 이용하여 feature map의 크기를 줄이고, 위치나 이동에 강한 특징을 추출하기 위한 방법

*Max pooling : pooling window 내의 가장 큰 값을 선택하는 방법

*Average pooling : 평균 연산으로 인해 학습 결과가 좋지 않음





*Stochastic pooling : 최대값 또는 평균값 대신 확률에 따라 적절한 activation을 선택함
확률값은 특정 activation에 대해 전체의 activation의 합을 나누는 방식으로 계산됨

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

Dropout과 같이 다양한 네트워크를 학습하는 듯한 model average 효과를 얻을 수 있음

- Channel (채널)

: 이미지는 높이(height),너비(width), 채널(color)이라는 3Dimensional Tensor 구성

* 채널 : 색 성분

->각 픽셀을 RGB 3개의 실수로 표현한 3차원 데이터

컬러 이미지는 3개의 채널로 구성

흑백 사진은 2차원 데이터로 1개 채널로 구성

각 픽셀은 0부터 255 사이의 값으로 이루어짐

28*28 픽셀의 이미지
(28*28*1)의 3차원 텐서

□ 통상적으로 접하게 되는 컬러 이미지

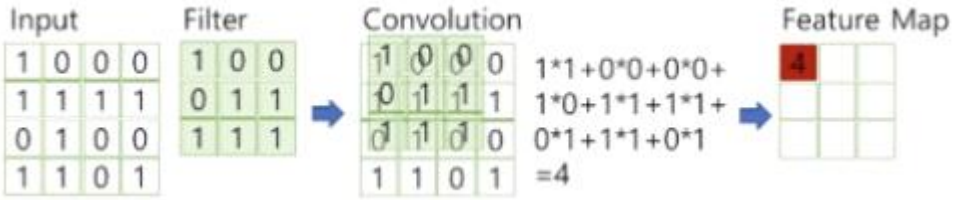
사막 이미지는 (28 × 28 × 3)의 크기를 가지는 **3차원 텐서**

- Filter (= Kernel)

: 이미지의 특징을 찾아내기 위한 공용 파라미터

일반적으로 (4,4)이나 (3,3)과 같은 정사각 행렬로 정의

CNN에서 학습의 대상은 필터 파라미터



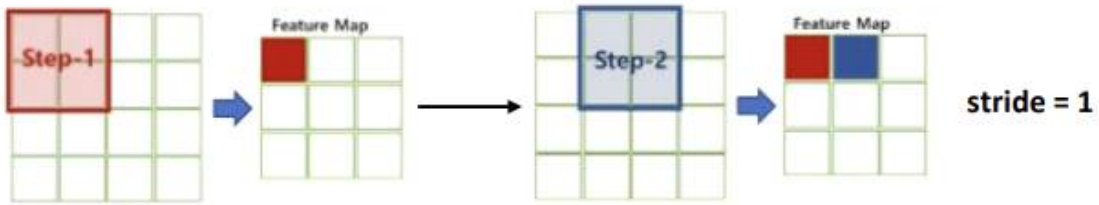
->입력 데이터를 지정된 간격으로 순회

채널별로 합성곱을 하고 모든 채널(컬러 :3개)의 합성곱의 합을 Feature Map으로 만들어 냄

- Stride

  : 지정된 간격으로 필터를 순회하는 간격(필터의 이동량)

  필터는 입력 데이터를 지정한 간격으로 순회하면서 합성곱을 계산함

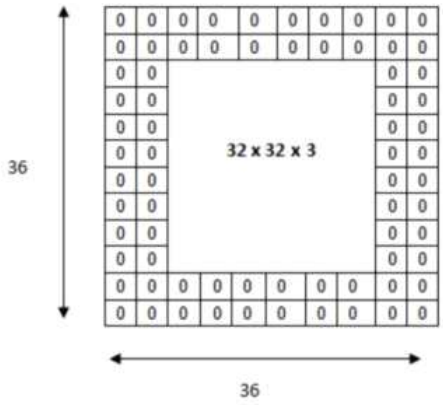


- Padding

  : 입력 데이터의 외각에 지정된 픽셀만큼 특정 값으로 채워 넣는 것

  Convolution(합성곱) 레이어의 출력 데이터가 줄어드는 것을 방지 -> CNN과정에서 여러 번의 계산을 하게 될 때, 초반에 이미지가 너무 작아져 버려 더 깊게 학습할 데이터가 부족해지는 현상 발생 => 보통 패딩 값으로 0을 채움

  Convolution 레이어에서 Filter와 Stride에 작용으로 Feature Map 크기는 입력데이터보다 작음



(32, 32, 3) 데이터를 외각에 2 pixel 추가 -> (36, 36, 3) 행렬

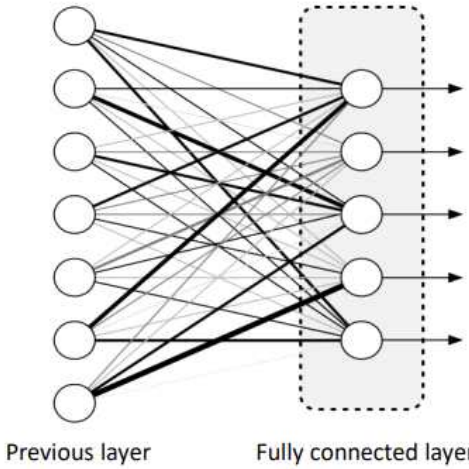
-Fully Connected Layer(FCN) : 평탄화

-> Convolution Layer과 Pooling Layer으로부터 얻어진 특징 벡터들은 FCN의 입력으로 사용

이전 레이어의 출력을 평탄화(Flatten)하여 다음 스테이지의 입력이 될 수 있는 단일 벡터로 변환

비선형 공간에서의 분류를 수행하게 됨

Fully Connected -> 모든 뉴런들이 전부 연결되어 있는 형태를 가지고 있어 붙여짐



>> 1~3 과정을 Fully Connected Layers라고 정의함

① 2차원 배열 형태의 이미지를 1차원 배열로 평탄화
② 활성화 함수 (Relu, Tanh 등)뉴런을 활성화
③ 분류기 (Softmax) 함수로 분류

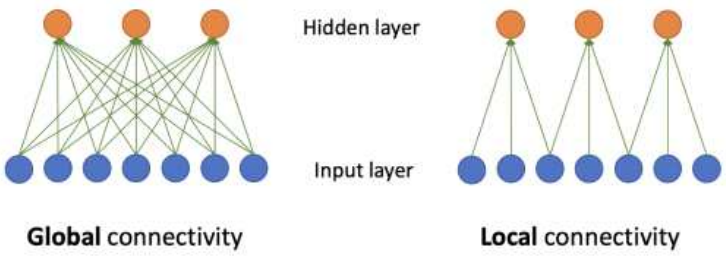
- CNN 특징

: CNN은 기존의 다층 신경망에 비해 중요한 두 가지 특징이 존재

1) Locality (Local Connectivity)

- 모든 데이터에 대해 input 값으로 받아들이지 않고, 인접한 데이터에 대해서만 input 값으로 받아들임
- 각 output 값은 모든 input 값을 고려하지 않고 적은 수의 input 값에 의해서만 결정될 수 있음

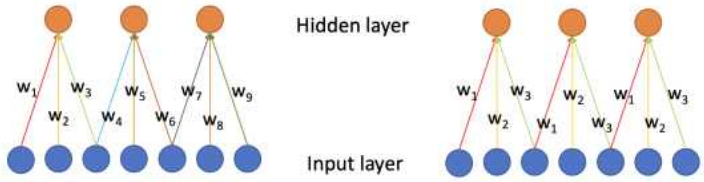
- Global connectivity로 구현할 경우 hidden layer의 각 unit의 차원수 (dimensionality) -> 7
  - 총 3 * 7개의 파라미터 (weight)이 필요
- Local connectivity로 구현할 경우 hidden layer의 각 unit의 차원수 (dimensionality) -> 3
- 총 3 * 3개의 파라미터(weight)만 필요



2) Shared Weights

- 동일한 계수를 갖는 filter를 전체 영상에 반복적으로 적용함으로 변수의 수를 획기적으로 줄임
- Topology 변화에 무관하게 항상성(invariance)을 얻을 수 있음

- Weight sharing을 사용하지 않는 모델은 모든 unit이 다른 파라미터를 사용, 3 * 3개의 파라미터가 필요 (왼쪽 그림)
- Weight sharing을 사용하는 모델은 각 unit에서 파라미터를 공유, 3 * 1개의 파라미터만 필요 (오른쪽 그림)



Without weight sharing **With** weight sharing