AI 특강

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

5. 군집화

- Scikit-learn의 machine-learning 방법 분류

- 군집화 (clustering)

- 군집화는 매우 다양한 응용 분야를 가지고 있다

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: *SVM*, *nearest neighbors*, *random forest*, ... — *Examples*

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: *SVR*, *ridge regression*, *Lasso*, ... — *Examples*

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: *k-Means*, *spectral clustering*, *mean-shift*, ... — *Examples*

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: *PCA*, *feature selection*, *non-negative matrix factorization*. — *Examples*

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: *grid search*, *cross validation*, *metrics*. — *Examples*

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: *preprocessing*, *feature extraction*. — *Examples*

- 군집화란?
- Clustering is the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar to each other than to those in other groups
- 대표적인 군집화 알고리즘: K-means 알고리즘

- 군집화 사용 예시

- 어떤 사람이 몇년간 쇼핑몰을 성공적으로 운영해 왔다 (수백만명의 고객을 가지고 있다). 그 동안 한 종류의 홍보 판플렛을 만들어 발송했는데 이제부터는 고객의 취향을 분석하여 4-6종류의 판플렛을 만들어 맞춤 홍보를 하고자 한다. 일종의 개인화 (personalization) 홍보 전략이다. 고객에 대한 각종 정보는 DB에 저장되어 있어 이것을 기초 자료로 활용 가능하다

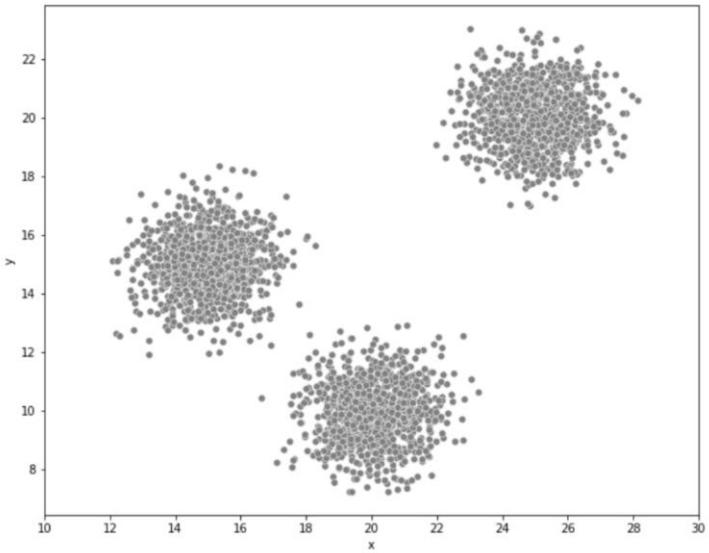
- 고객 정보는 월평균 구매액, 선호하는 물품의 종류와 수준, 결제 방법, 반품 성향, 직업, 성별, 나이, 거주 지역등 다양하다.

- Q) 수백만명의 고객이 있을때 이를 어떻게 4-6개의 그룹으로 나눌수 있을까?

- 해결 방법: 유사한 샘플(데이터)들끼리 모아서 4-5개의 그룹으로 만든다
- 여기서 유사하다는 것은 데이터와의 거리가 가깝다는 것을 의미 한다
- 이 각각의 그룹을 군집 (cluster)라고 하고 이 경우에는 4-5개의 군집이 만들어 진다
- 여태까지 배운 Supervised learning과 다르게 오늘 배울 군집화 (clustering)은 unsupervised learning으로써 데이터 정보만 있을 뿐 이 데이터의 분류는 주어져 있지 않다

K-means 알고리즘

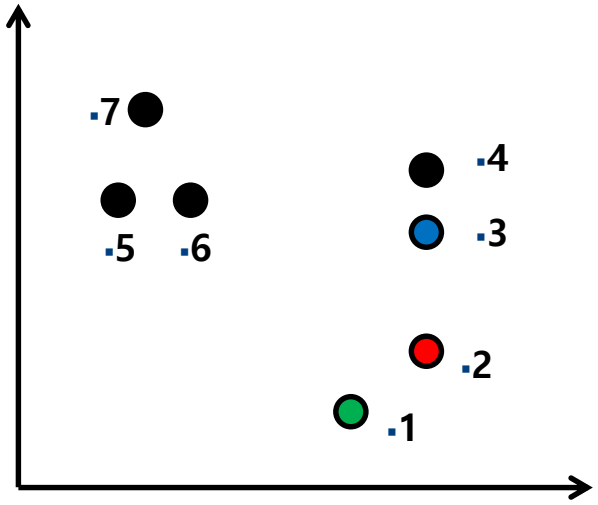
- K-means 군집화 알고리즘은 주어진 데이터를 K개의 클러스터 (군집)으로 묶는 알고리즘이다
- 아래와 같은 데이터의 갱우 K=3이라고 생각할 수 있다
- K-means 군집화 알고리즘의 경우 기본적으로 입력으로 사용자가 K를 정하지만 뒤에서 배울 elbow 방법과 같은 방법으로 K를 정할수도 있다



- 다음 예제는 총 7개의 데이터에 대해서 K-means 군집화 알고리즘을 실행한 예제이다

- 이 경우에 K=3으로 사용자가 정했다

- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



1. x1, x2, x3를 초기 군집 중심으로 삼고

- 이를 z1, z2, z3라고 놓는다

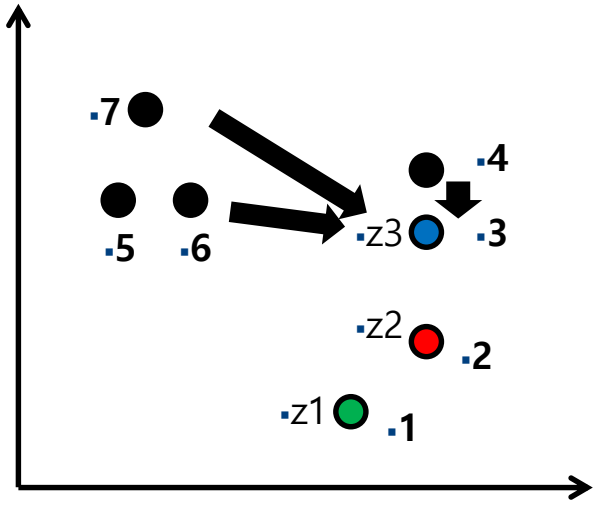
- z1=(18,5)

- z2=(20,9)

- z3=(20,14)

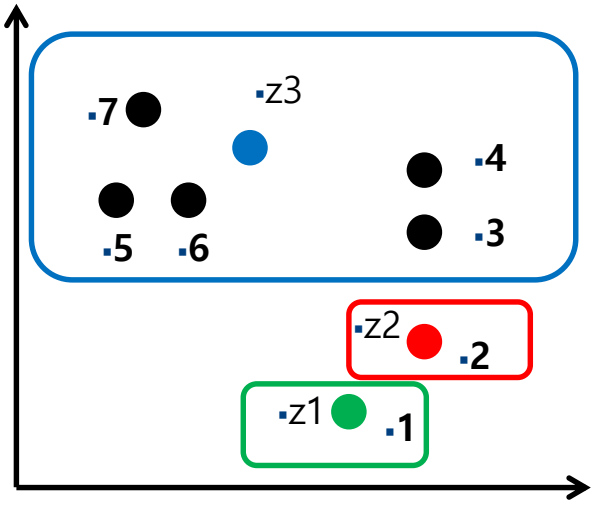
- x1=(18,5), x2=(20,9), x3=(20,14), x4=(20,17), x5=(5,15), x6=(9,15), x7=(6,20

- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



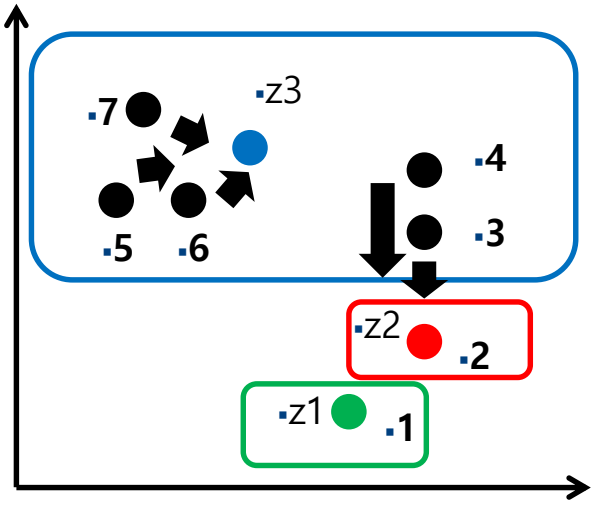
- 2. 각 데이터들은 z1,z2,z3중
- 가장 가까운 군집 중심에 각각 배정된다
- {x1}은 z1
- {x2}은 z2
- {x3,x4,x5,x6,x7}은 z3 에 배정된다

- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 2. {x1}은 z1
- {x2}은 z2
- {x3,x4,x5,x6,x7}은 z3 에 배정된다
- 3. 2에서 배정된 데이터들을 이용해 새로운
- 군집 중심을 찾고 (위치 평균)
- 이를 새로운 z1,z2,z3라 놓는다
- z1=(18,5)
- z2=(20,9)
- z3=(x3+x4+x5+x6+x7)/5=(12,16.2)

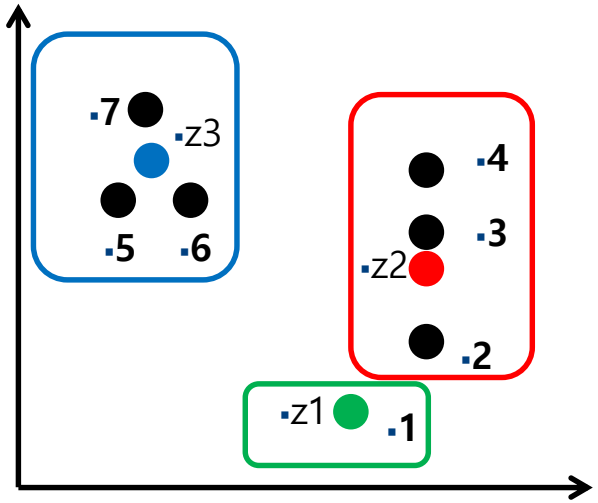
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 앞의 step 2를 반복한다. 즉
- 각 데이터들은 가장 가까운 군집 중심에
- 각각 배정된다

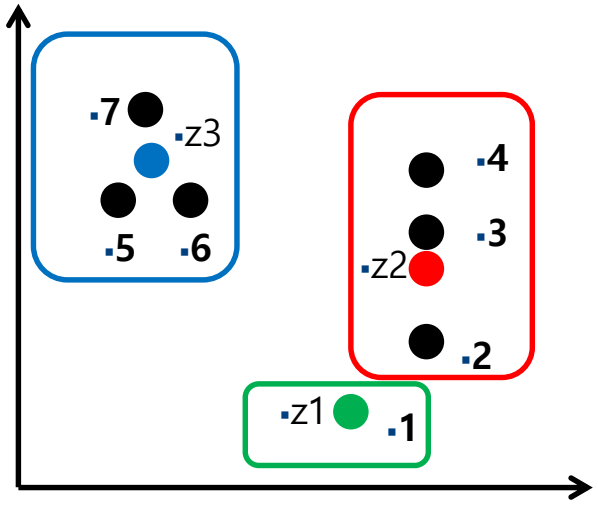
- {x1}은 z1
- {x2,x3,x4}는 z2
- {x5,x6,x7}은 z3 에 배정된다

- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- {x1}은 z1
- {x2,x3,x4}는 z2
- {x5,x6,x7}은 z3 에 배정된다
- 앞의 step 3를 반복한다. 즉
- 배정된 데이터들의 군집 중심을 찾고
- 이를 새로운 z1, z2, z3라 놓는다
- z1=(18,5), z2=(x2+x3+x4)/3=(20,13.333)
- z3=(x5+x6+x7)/3=(6.667,16.667)

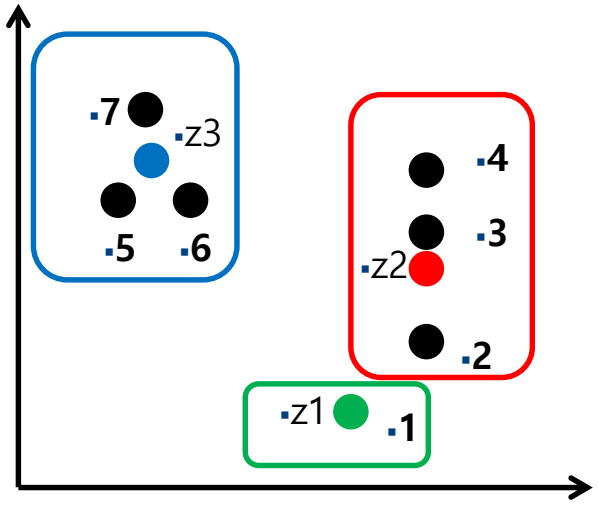
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 그 다음에 step 2와 3를 반복한다
- Step 2: 각 데이터들은 가장 가까운
- 군집 중심에 각각 배정된다
- 변화가 있는가? 없음
- Step 3: 배정된 데이터들의 각각의
- 군집 중심을 찾고 (위치 평균)
- 이를 새로운 z1,z2,z3라 놓는다

- 따라서 step3의 변화도 없음

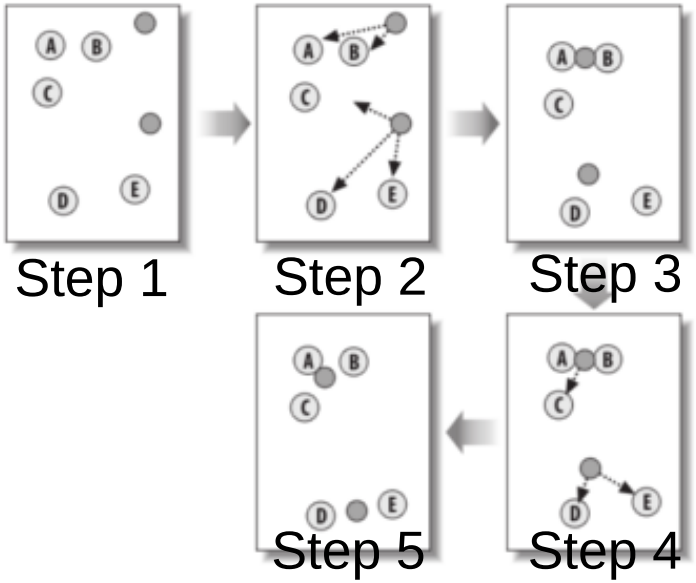
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 즉 최종적으로 왼쪽의 데이터들을
- K=3 (군집개수=3)으로 군집화 한
- 경우
- {x1}, {x2,x3,x4}, {x5,x6,x7}과 같이
- 세 개의 군집으로 군집화 할 수 있다

K-means 알고리즘

- K=2이고 5개의 데이터 (A, B, C, D, E)가 있는 경우
- 1. 임의의 위치에 군집 중심 2개가 위치
- 2. 각각의 데이터가 가장 가까운 군집 중심을 찾음. 이 경우에는 A, B는 위쪽 군집 중심과 가깝고, C, D, E는 아래쪽 군집 중심과 가까움
- 3. A, B의 평균 위치가 새로운 군집 중심의 위치, C,D, E의 평균 위치가 새로운 군집 중심의 위치
- 4. 다시 2 반복. 이 경우 A, B, C는 위쪽 군집 중심과 가까움. D, E는 아래쪽 군집 중심과 가까움
- 5. 다시 3 반복.
- 언제까지 2,3 반복?

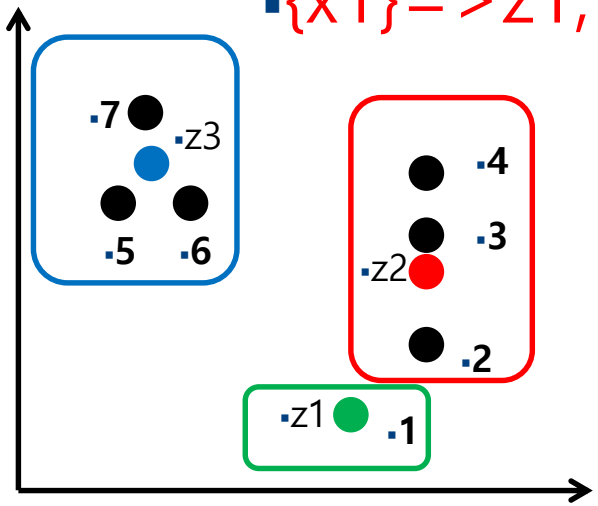


- SSE와 Elbow method

- 군집화가 잘 되었다라는것은 어떻게 판단하나?
- K-means 에서 군집화가 잘 되어 있다 라는건 데이터들이 군집 중심 근처에 모여 있는 것이다. 데이터 들이 군집내에서 퍼져 있는건 군집화가 잘되지 않았다라고 생각할 수 있다
- 군집화가 잘되었나 판단하는 척도 (metric) 중 하나는
- 각각의 데이터가 그것에 가장 가까운 군집 중심 사이의 거리 (혹은 거리의 제곱)을 모두 더해 보는 것이다
- 이를 Sum of squared error (SSE)라고도 한다

- 예를 들어 마지막 최종 군집화 결과의 경우

- {x1}=>z1, {x2,x3,x4}=>z2, {x5,x6,x7}=>z3



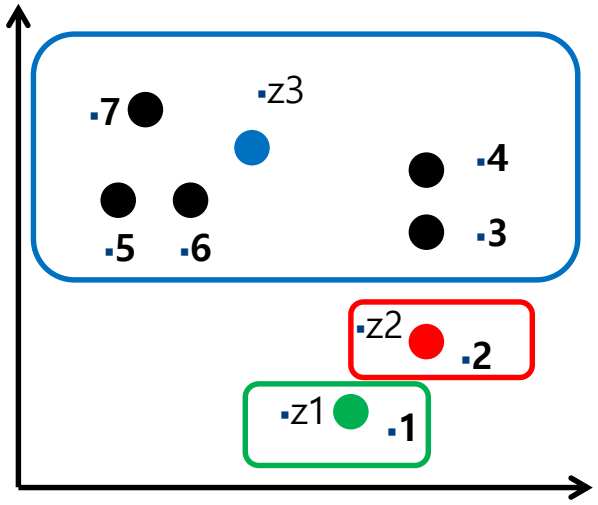
- 예를 들어 마지막 최종 군집화 결과의 경우

- {x1}=>z1, {x2,x3,x4}=>z2, {x5,x6,x7}=>z3

- SSE: 제곱 오류

- SSE=58

- 최종 군집화 바로 이전의 경우



- {x1}=>z1, {x2}=>z2, {x3,x4,x5,x6,x7}=> z3

- SSE=244.8 => 즉 최종 군집화된 결과가 제곱 오류가 더 적다!

- Iris dataset을 이용한 군집화 예제

- Iris dataset

총 150개의 데이터

데이터설명 : 아이리스(붓꽃) 데이터에 대한 데이터이다. 꽃잎의 각 부분의 너비와 길이등을 측정한 데이터이며 150개의 레코드로 구성되어 있다. 아이리스 꽃은 아래의 그림과 같다. 프랑스의 국화라고 한다.

필드의 이해 :

데이터의 이해를 돕기 위해 포함된 6개의 변수에 대하여 간략하게 설명한다.
총 6개의 필드로 구성되어있다. caseno는 단지 순서를 표시하므로 분석에서 당연히 제외한다.
2번째부터 5번째의 4개의 필드는 입력 변수로 사용되고, 맨 아래의 Species 속성이 목표(종속) 변수로 사용된다.

caseno	일련번호이다. (1부터 150까지 입력된다.)
Sepal Length	꽃받침의 길이 정보이다.
Sepal Width	꽃받침의 너비 정보이다.
Petal Length	꽃잎의 길이 정보이다.
Petal Width	꽃잎의 너비 정보이다.
Species	꽃의 종류 정보이다. setosa / versicolor / virginica 의 3종류로 구분된다.

```
from sklearn.datasets import load_iris

iris = load_iris()
iris
```

This code gives:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],...
'target': array([0, 0, 0, ... 1, 1, 1, ... 2, 2, 2, ...
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
...}
```

실습

- 퍼셉트론 (Perceptron)

- 신경망은 machine learning (기계 학습) 역사에서 가장 오래된 기계 학습 모델

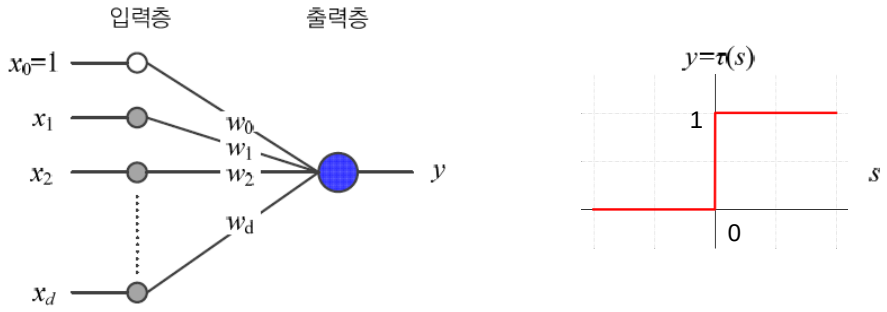
- 1950년대 퍼셉트론 -> 1980년대 다층 퍼셉트론
- 최근 딥러닝의 가장 기초

- 2000년대 딥러닝 등장

- 퍼셉트론 (perceptron)은 노드, 가중치, 층과 같은 새로운 개념을 도입하고 학습 알고리즘을 창안함

- 퍼셉트론은 원시적 신경망이지만, 딥러닝을 포함한 현대 신경망은 퍼셉트론을 병렬과 순차 구조로 결합하여 만듬 → 현대 신경망의 중요한 구성 요소

- 퍼셉트론이란 가장 간단한 형태의 인공 신경망으로 입력과 출력은 모두 숫자이며 입력은 가중치 (weight)라는 것과 연결되어 있다
- 여기서 젤 위에 있는 것은 바이어스 노드라 하고 출력은 한 개의 노드 이다
- 퍼셉트론에서는 다음 2가지 과정을 순차적으로 수행한다
- 1. 입력과 각각의 가중치를 모두 곱해서 더한 가중치합을 계산한다
- $s = w_0 + w_1 x_1 + \cdots w_d x_d = w_0 + \sum_{i=1}^{d} w_i x_i$
- 2. 이 가중치 합에 활성함수라고 하는 계단 함수를 통과시키킨다.
- 1에서 계산한 s가 0보다 크거나 같으면 1을 출력, s가 0보다 작으면 0을 출력



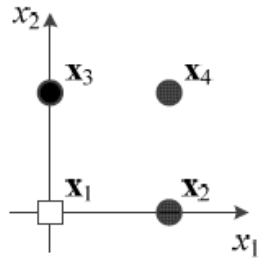
(a) 퍼셉트론의 구조 (b) 계단함수를 활성함수 $\tau(s)$로 이용함

그림 3-3 퍼셉트론의 구조와 동작

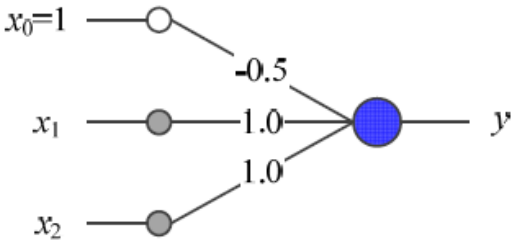
예제 3-1 퍼셉트론의 동작

2차원 특징 벡터로 표현되는 샘플을 4개 가진 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, $\mathbb{Y} = \{y_1, y_2, y_3, y_4\}$를 생각하자. [그림 3-4(a)]는 이 데이터를 보여준다.

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



(a) 훈련집합



(b) 퍼셉트론

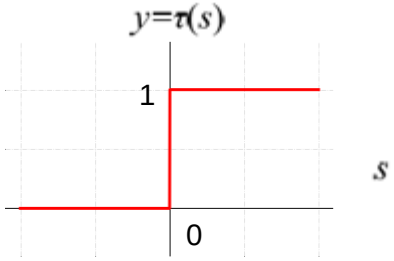
그림 3-4 OR 논리 게이트를 이용한 퍼셉트론의 동작 예시

▪OR Gate

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

$$\begin{aligned}
&\mathbf{x}_1\text{: } s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, && \tau(-0.5) = 0\\
&\mathbf{x}_2\text{: } s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_3\text{: } s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_4\text{: } s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, && \tau(1.5) = 1
\end{aligned}$$



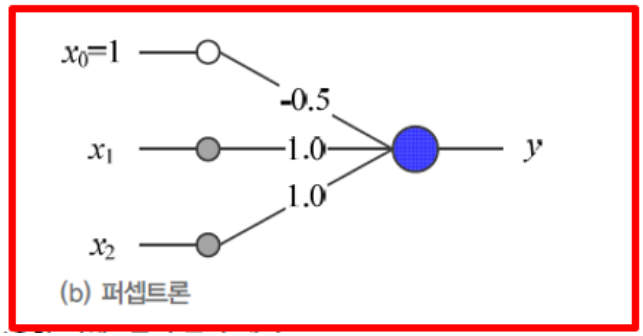
결국 [그림 3-4(b)]의 퍼셉트론은 샘플 4개를 모두 맞추었다. 이 퍼셉트론은 훈련집합을 100% 성능으로 분류한다고 말할 수 있다.

- Python으로 구현한 ‘OR’ 퍼셉트론 예

```
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5])
    b = -0.4
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else :
        return 1

import numpy as np
print(OR(0,0))
print(OR(0,1))
print(OR(1,0))
print(OR(1,1))
```

- 이 퍼셉트론을 기하학적으로 설명하면
  - 결정 직선 $d(\mathbf{x}) = d(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0 = 0$ → $x_1 + x_2 - 0.5 = 0$
    - w_1과 w_2는 직선의 방향, w_0은 절편을 결정
    - 결정 직선은 전체 공간을 1과 0의 두 부분공간으로 분할하는 분류기 역할
- 즉, 퍼셉트론은 선형 분류기라고 생각할 수 있다



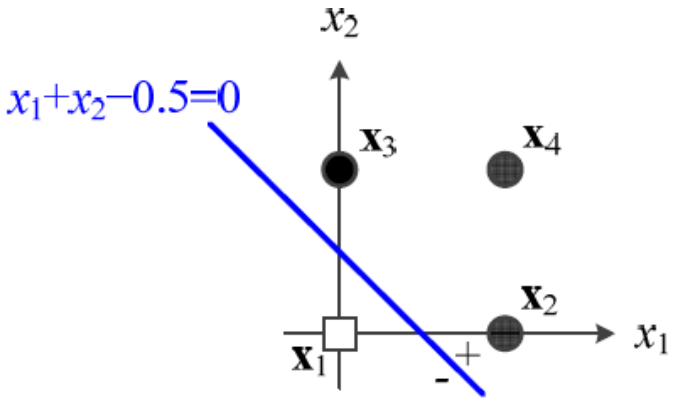
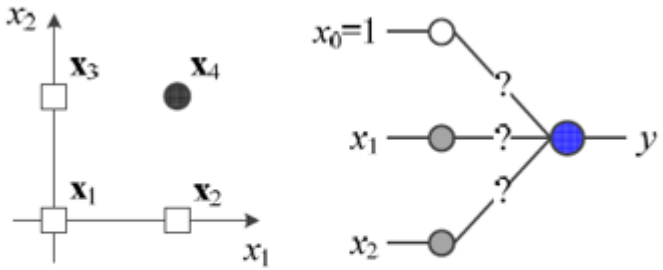


그림 3-5 [그림 3-4(b)]의 퍼셉트론에 해당하는 결정 직선

- 실습: 다음과 같은 AND Gate를 선형 분류할 수 있는 퍼셉트론을 직접 구현해보자

▪AND Gate

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



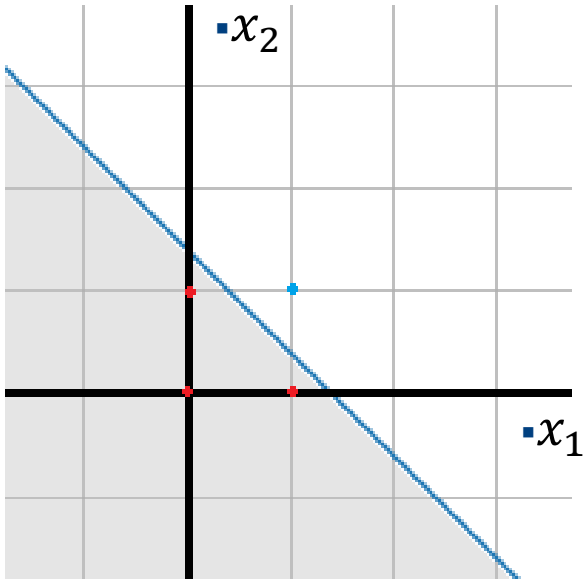
(a) AND 분류 문제

- **퍼셉트론으로 나타낸 AND Gate** $(x_1, x_2, \theta) = (0.5, 0.5, 0.7)$

- $$y = \begin{cases} 0 & (0.5x_1 + 0.5x_2 \leq 0.7) \\ 1 & (0.5x_1 + 0.5x_2 > 0.7) \end{cases}$$

▪AND Gate

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



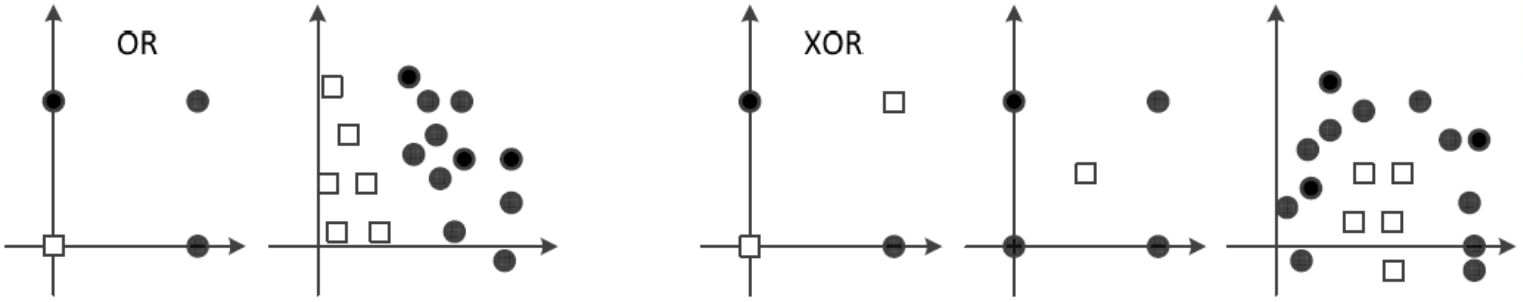
```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else :
        return 1

import numpy as np
print(AND(0,0))
print(AND(0,1))
print(AND(1,0))
print(AND(1,1))
```

- **단층 퍼셉트론의 한계**
- **지금까지 배운 내용을 단층 퍼셉트론이라고 한다**
- **단층 퍼셉트론은 단순한 선형 분류기 이기 때문에 XOR에 대한 선형 분류가 불가능하다**

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



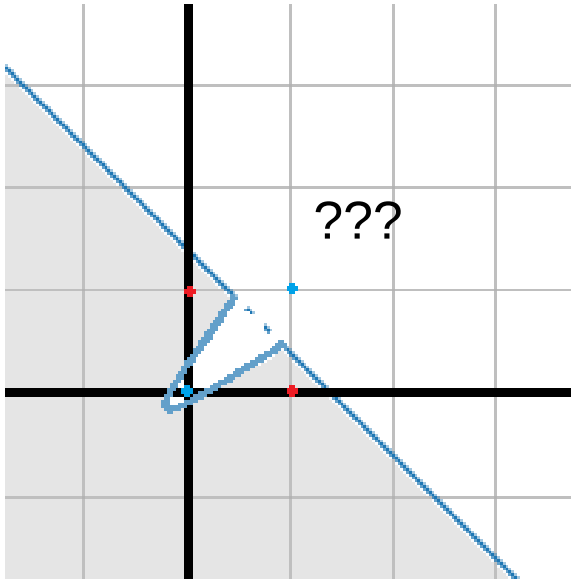
(a) 선형분리 가능 (b) 선형분리 불가능

그림 3-7 선형분리가 가능한 상황과 불가능한 상황

- **다층 퍼셉트론 (MLP)**

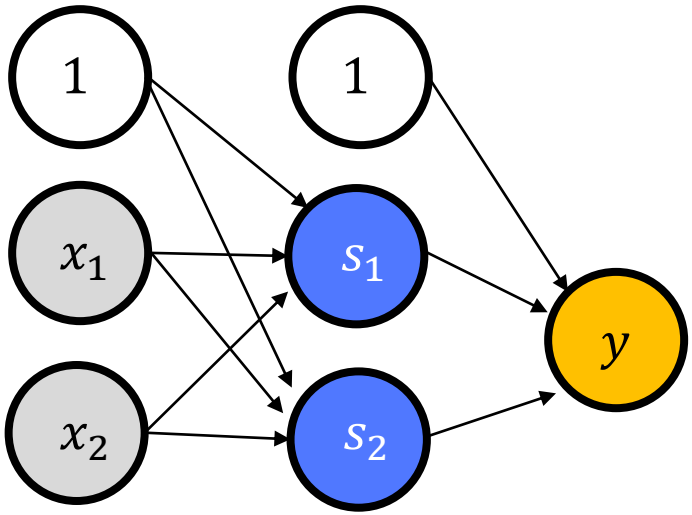
- 다층 퍼셉트론 (Multi-Layer Perceptron, MLP) 개요
- XOR과 같이 직선 하나로는 표현 불가능한 영역도 존재한다.

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

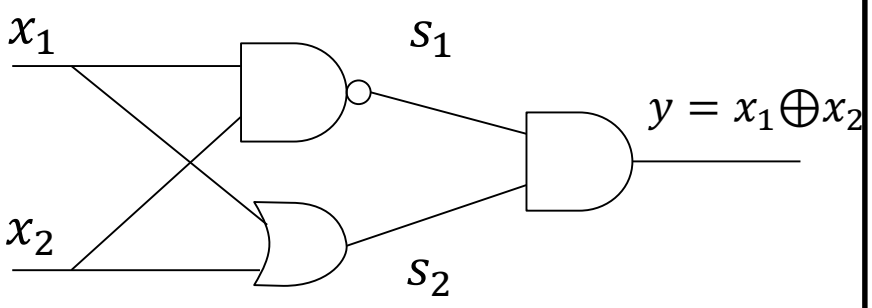


- 다층 퍼셉트론은 말 그대로, 퍼셉트론을 여러층 쌓는 것이다.
- i층의 퍼셉트론의 출력들이 i+1층의 퍼셉트론 입력이 된다.

- 입력데이터의 층을 입력층,
- 출력 신호가 나오는 퍼셉트론층을 출력층,
- 그 외 층들은 보이지 않는다 하여 은닉층이라 한다.



- 다층 퍼셉트론으로 XOR을 나타내보자.
- XOR은 NAND (Not-AND), OR, AND Gate로 나타낼 수 있다.

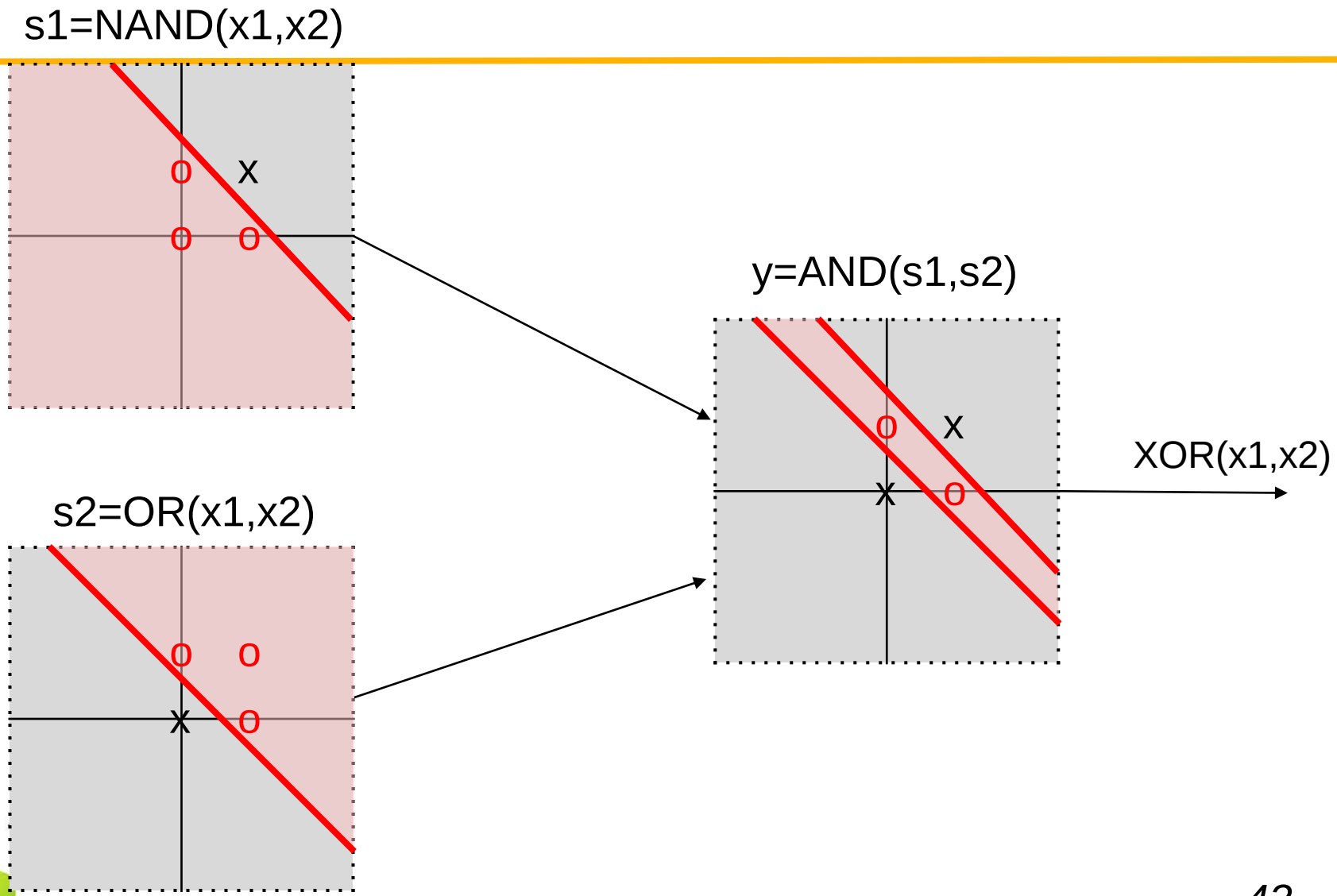



```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

x1	x2	s1	s2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



- **실습**

- **퍼셉트론 (신경망)의 학습**

- 신경망을 학습시키려면 다음 두 가지가 필요하다

1. 정답(label)을 가지고 있는 데이터 (훈련데이터)
2. 신경망이 얼마나 정확한지/부정확한지 나타내는 평가지표

- 신경망을 학습시킨다는 것은, 주어진 데이터에 대하여 평가 지표를 최적으로 하는 신경망의 가중치를 찾아내는 과정이다.

- 손실함수 (Loss function)
- 신경망이 얼마나 나쁜지 나타내는 지표. 개선할 수 있는 여지의 정도를 나타낸다.
- 손실함수가 0이다 = 완벽한 신경망

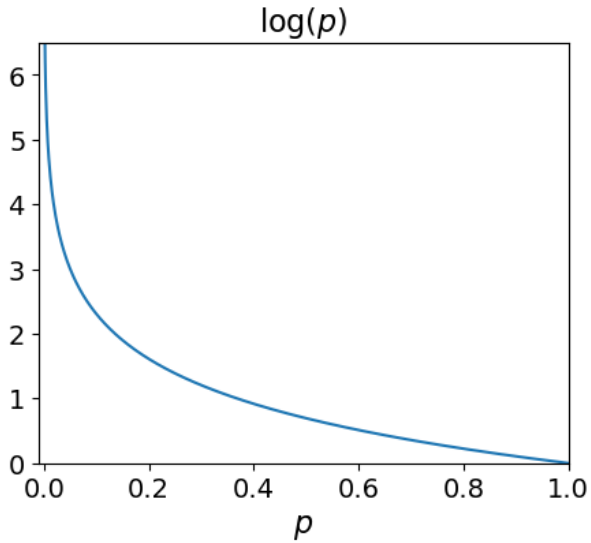
- 이진 분류기(Binary Classifier)의 손실함수
- 이진 분류기란 입력된 샘플 데이터에 대하여 0, 또는 1로 구분하는 것이 목적이다. (지금동안 보았던 퍼셉트론도 이진 분류기이다)
- 출력은 주로 입력 샘플이 1에 속할 확률을 출력한다.

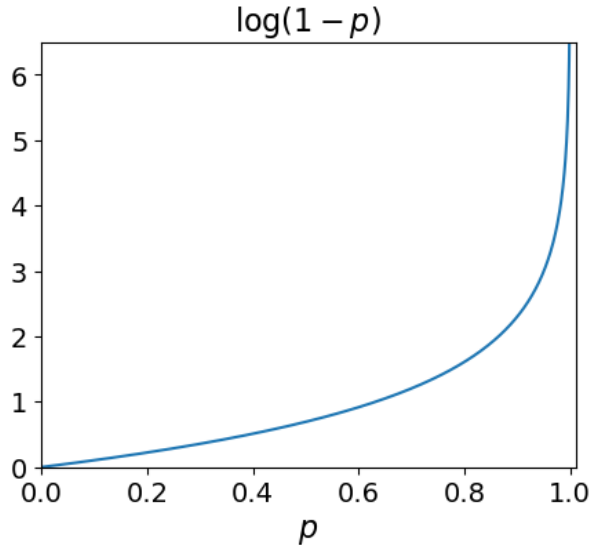
$$f_\theta(\mathbf{x}) = p(y = 1|\mathbf{x}) \in [\mathbf{0}, \mathbf{1}]$$

- 훈련 데이터의 정답이 0이라면, $f_\theta(x)$도 0에 가까워져야 하며, 정답이 1이라면 $f_\theta(x)$도 1에 가까워져야 한다.

- **크로스 엔트로피 오차 손실함수** ($t_i \in \{0, 1\}$는 훈련데이터 x_i의 정답)

- $$L(\theta) = \frac{1}{|X|} \sum_{x_i \in X} [-t_i \log(f_\theta(x_i)) - (1 - t_i) \log(1 - f_\theta(x_i))]$$





- 손실 함수를 최소로 만드는 가중치는 어떻게 찾을 것인가?
- 하나의 가중치 w_i에 대하여, 해당 가중치가 커져야 하는지, 작아져야 하는지 어떻게 알 수 있을까?
- 가중치 w_i 커지고 작아짐에 따라 손실함수가 어떻게 변할 지 알아낼 수 있다면…?

- **가중치 w_i 커지고 작아짐에 따라 손실함수가 어떻게 변할 지 알아낼 수 있다면…? = 기울기**

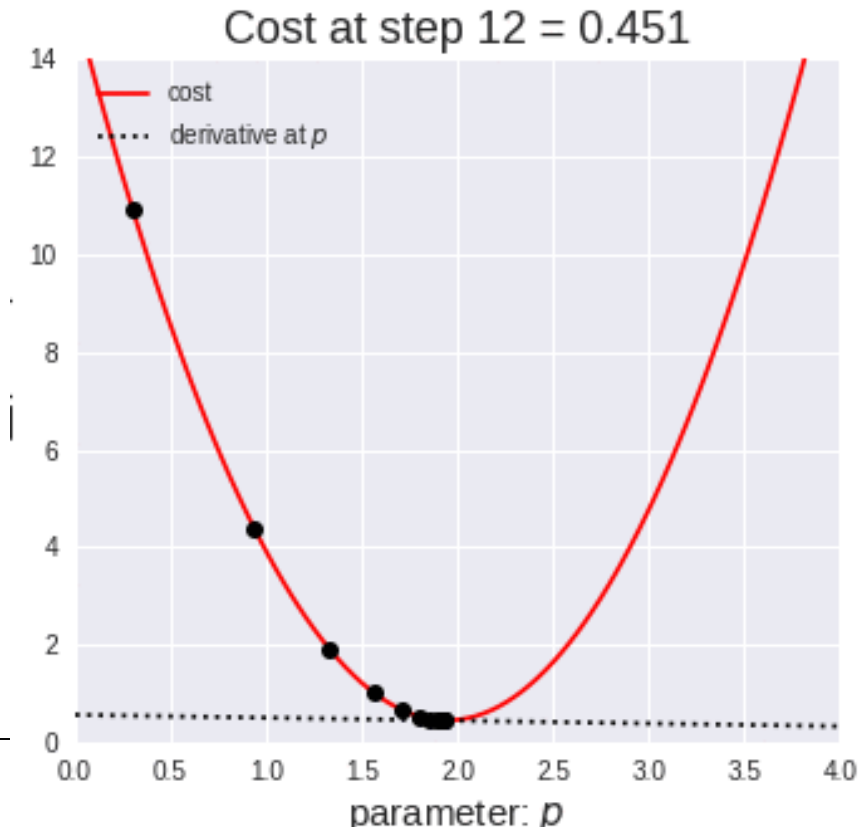
- **Gradient Descent (경사하강법, GD)는 일종의 학습 방법으로, 손실함수에 대한 가중치의 기울기를 구하고 (미분) 손실함수가 작아지는 방향으로 가중치를 변화시키는 방법이다.**

$$w' \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

- **α는 학습률로, 한번에 얼마나 이동할 지 결정한다.**

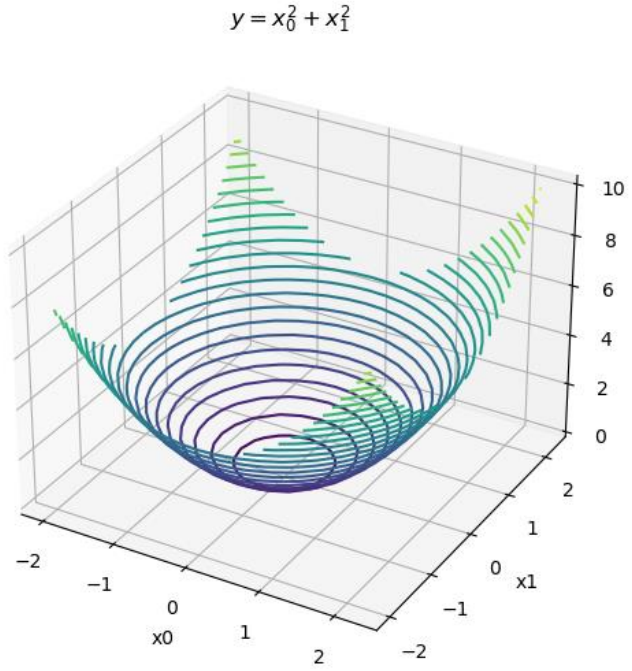
가중치 w에 대한
손실함수 L의 기울기

$$w' \leftarrow w - \alpha \frac{\partial L}{\partial w}$$



- **GD 코드**

```
def function_2(x):
    if x.ndim == 1:
        return np.sum(x**2)
    else:
        return np.sum(x**2, axis=1)
|
```

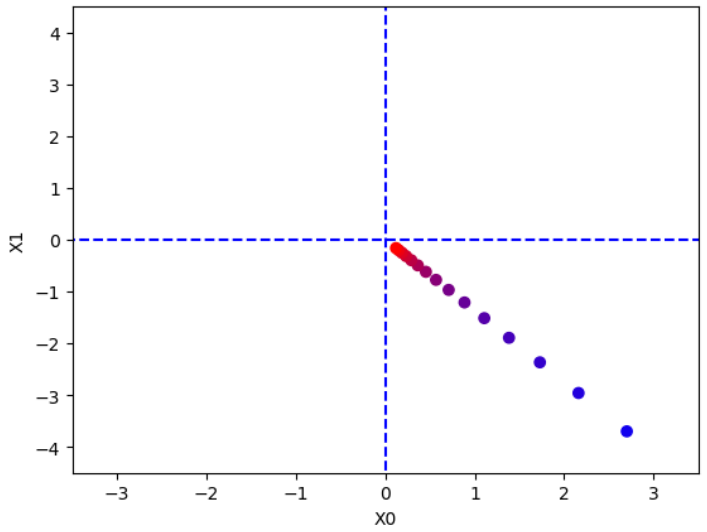



```
def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x
    x_history = []

    for i in range(step_num):
        x_history.append( x.copy() )

        grad = numerical_gradient(f, x)
        x -= lr * grad

    return x, np.array(x_history)
```



- **Stochastic Gradient Descent (확률적 경사하강법, SGD)**
- **원래 손실함수를 한번 구하기 위해서는 훈련데이터 전체를 보아야 한다.**

$$L(\theta) = \frac{1}{|X|} \sum_{x_i \in X} \{\text{loss of } x_i\}$$

- **훈련 데이터가 방대하면 방대할 수록, 이는 오래 걸리는 일이 되고 학습 속도를 저하시킨다.**

- **Stochastic Gradient Descent (확률적 경사하강법, SGD)**
- **따라서 우리는 훈련데이터 중 무작위로 k개를 추출하여 이에 대한 손실함수를 구하고 이 손실함수에 대하여 학습하고자 한다.**

$$L_{SGD}(\theta) = \frac{1}{k}\sum_{i=1}^{k}\{\text{loss of } x_i\}$$

- **이때 무작위로 추출한 k개를 미니배치(mini batch)라고 하며, k를 배치 크기(batch size)라고 한다.**
- **미니배치를 이용하여 훈련 데이터 전부를 한번씩 학습시키면, 이를 1 에폭(epoch) 라고 한다.**

- GD와 SGD의 차이점
- GD는 방향이 정확하나 모든 데이터셋을 봐야 하므로 한번 이동이 느리다
- SGD는 데이터의 일부만 보므로 한번 이동이 빨리 일어나나 방향이 어긋날 수 있다.

경사 하강법 SGD

- 그러나 SGD도 충분히 많이 반복하면 통계적으로 옳은 지점으로 향하게 된다.

- **신경망 학습 실습**