

0. K-평균 클러스터링 (K-Means clustering)의 문제점

- ✓ 초기 K 값을 사용자가 결정
- ✓ 센터들의 초기값을 정확하게 설정하기가 어려움 (random initialization)
- ✓ 클러스터 간 데이터의 밀도 차이가 있을 경우 클러스터링이 잘 되지 않음

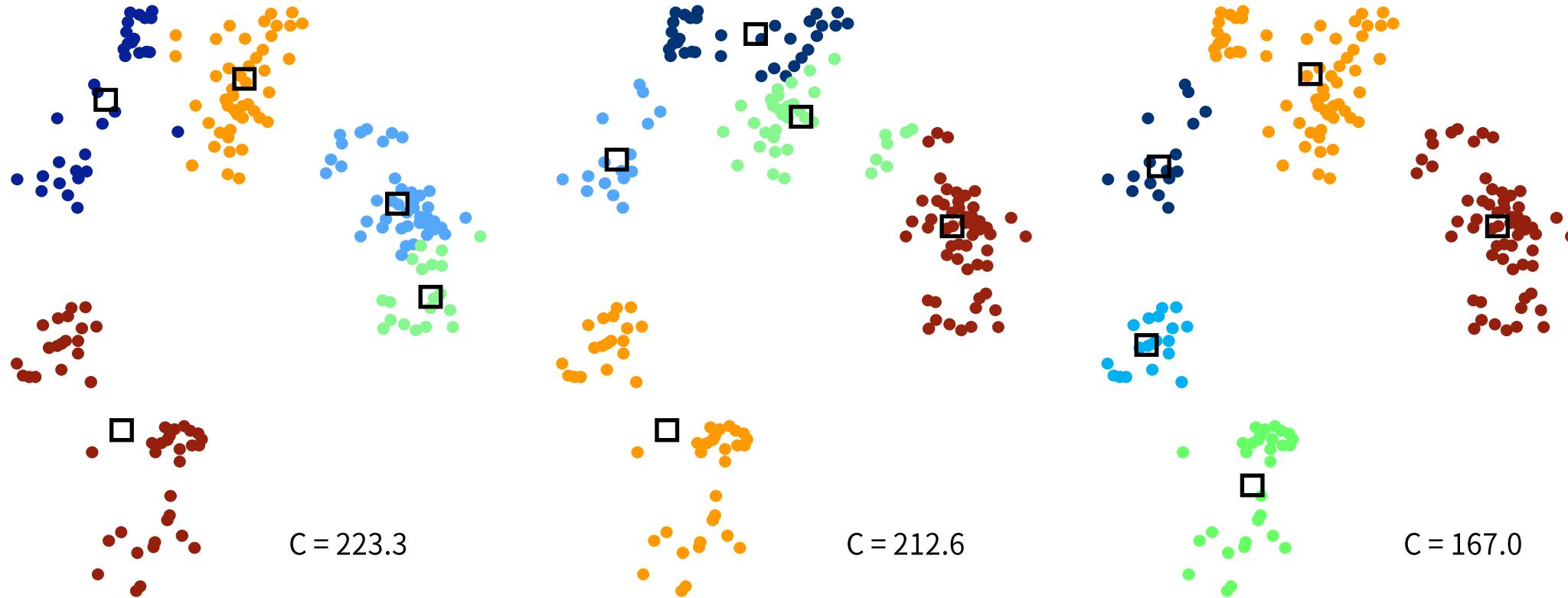


- ✓ 여러 번 반복해서, 최적의 결과를 선택
- ✓ 다른 초기화 전략을 사용
- ✓ 목적함수 C를 활용

1. 초기값 설정(center initialization)

▣ 여러 번 반복해서, 최적의 결과를 선택

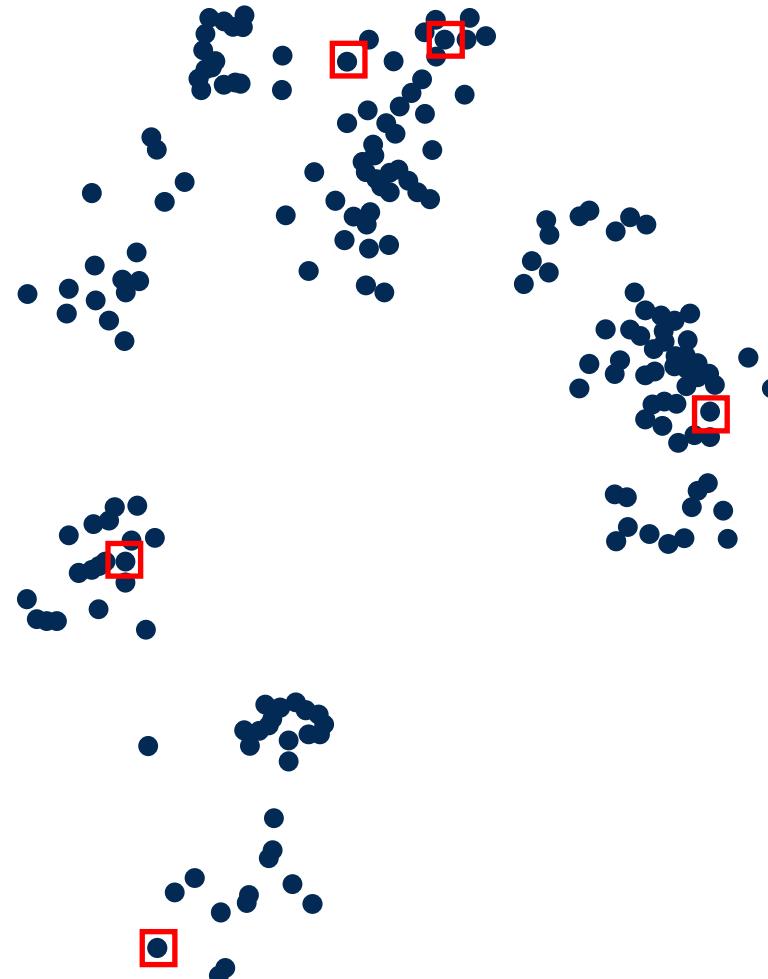
▣ 목적함수 C를 활용



1. 초기값 설정(center initialization)

Random initialization

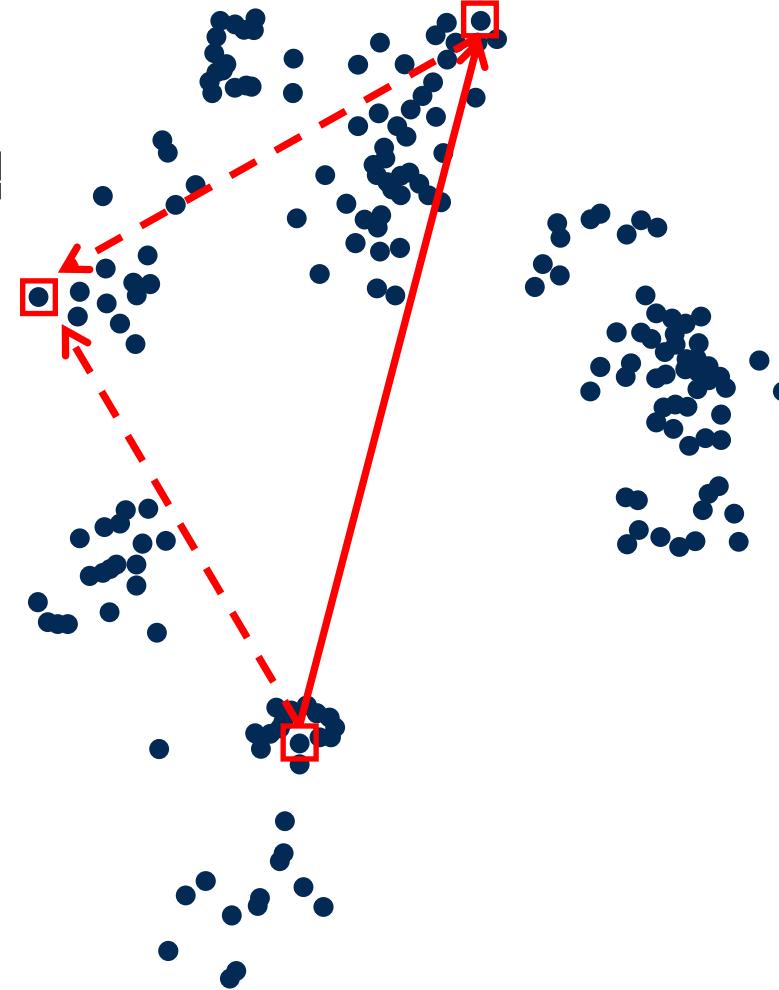
- ✓ 데이터 인덱스로부터 랜덤하게 선택
- ✓ 문제점: 서로 가까운 데이터들이 선택될 수 있음



1. 초기값 설정(center initialization)

Distance based initialization

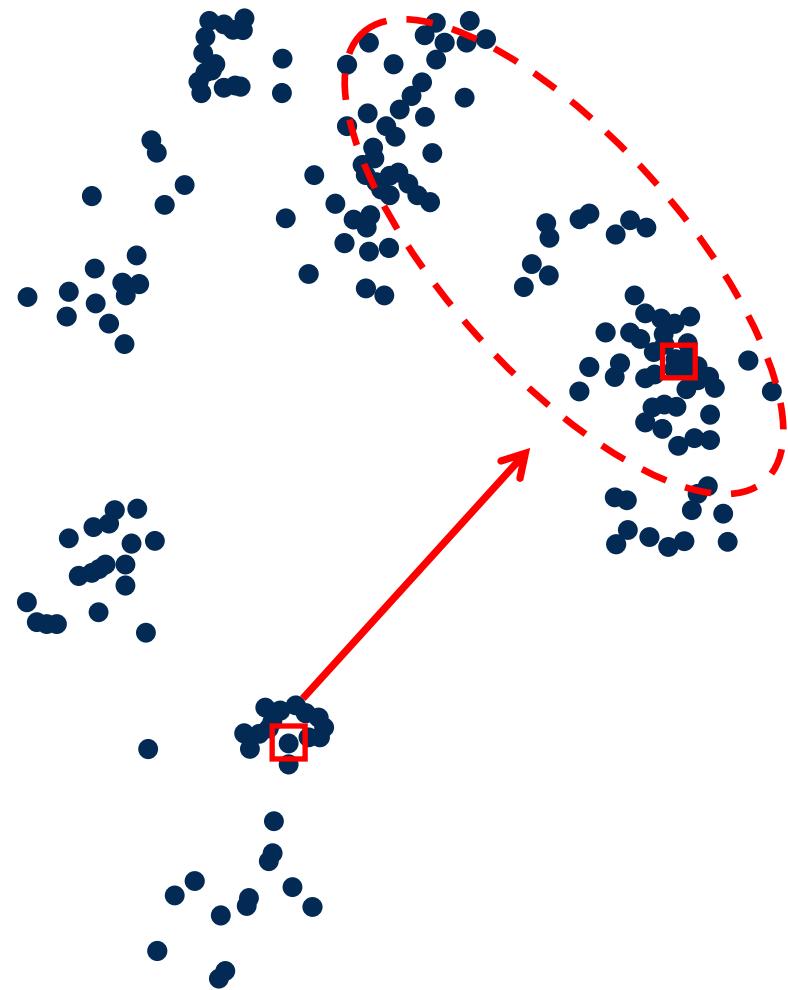
- ✓ 하나의 랜덤 데이터로부터 시작
- ✓ 지금까지의 클러스터 센터로부터 가장 먼 데이터를 선택
- ✓ 문제점: outliers 를 선택



1. 초기값 설정(center initialization)

Random + Distance (“kmeans++”)

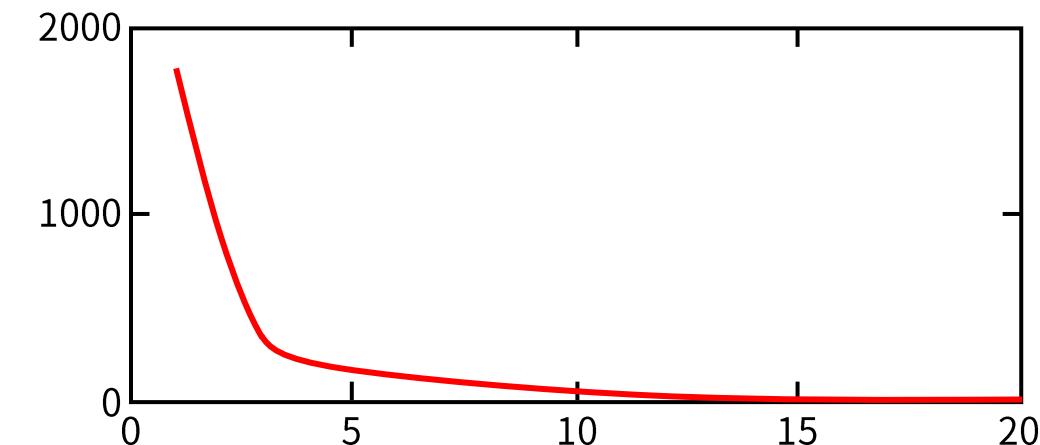
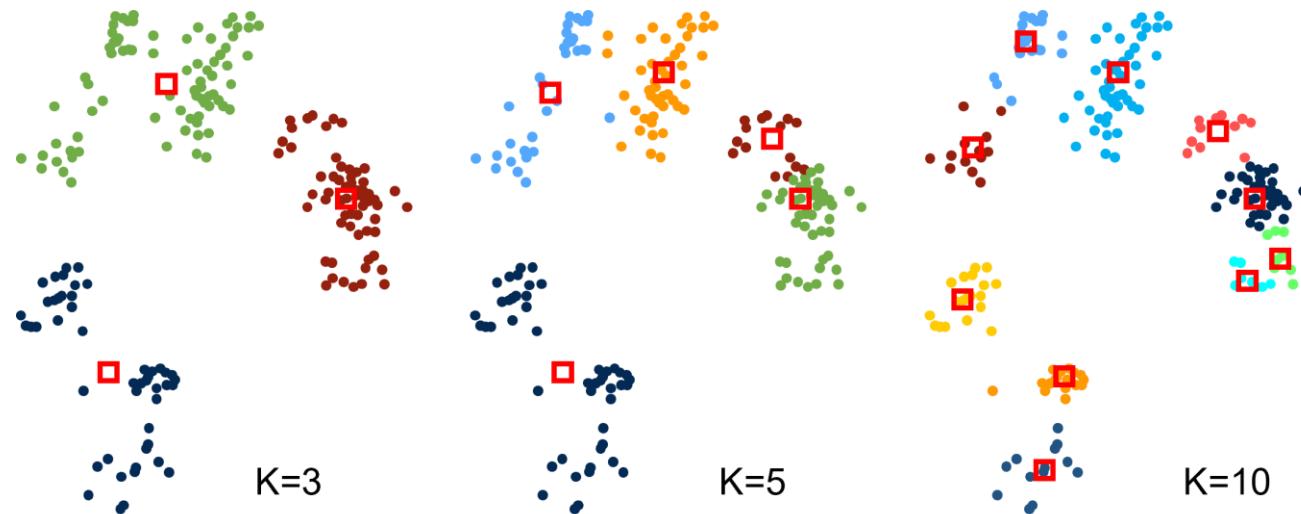
- ✓ 하나의 랜덤 데이터로부터 시작
- ✓ “far but randomly” 데이터를 선택
- ✓ 문제점: 많은 데이터를 가진 하나의 영역에서 편향된 클러스터 센터를 설정



2. 최적의 클러스터 개수 찾기

■ 비용함수 이용: $C(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$

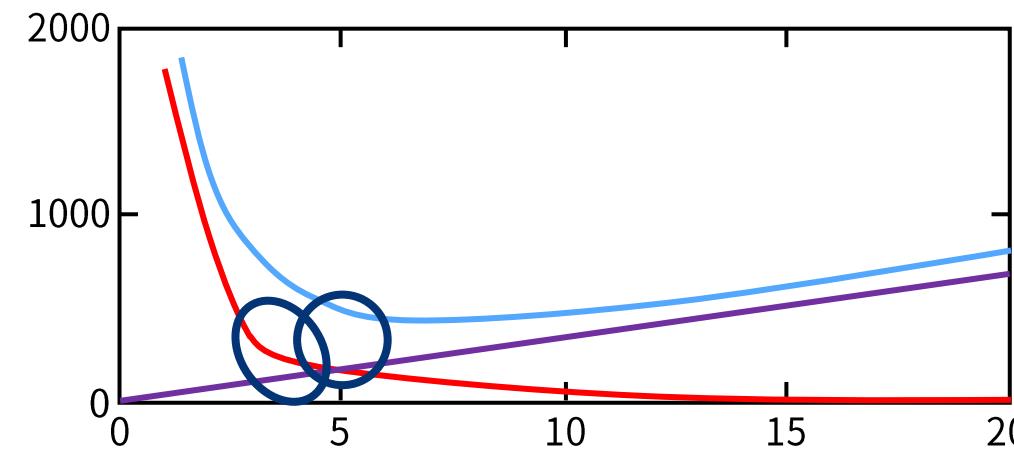
■ 일반적으로 K가 증가할 수록, C는 감소, but 비용 증가



2. 최적의 클러스터 개수 찾기

▣ 복잡도를 제어할 수 있는 penalty를 추가

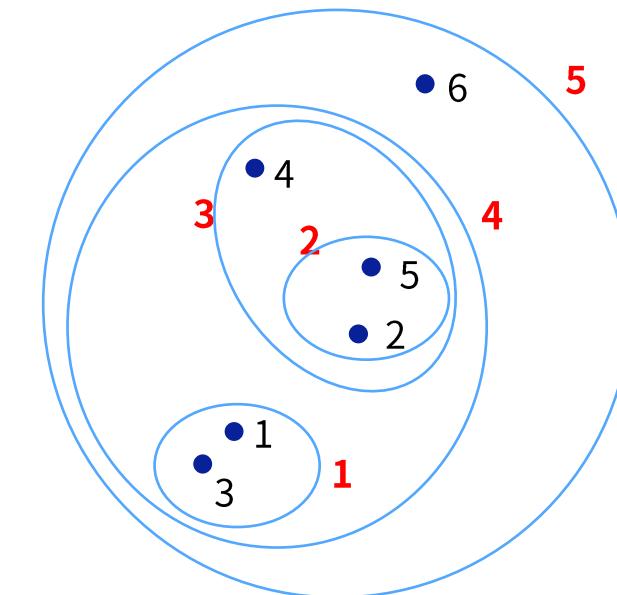
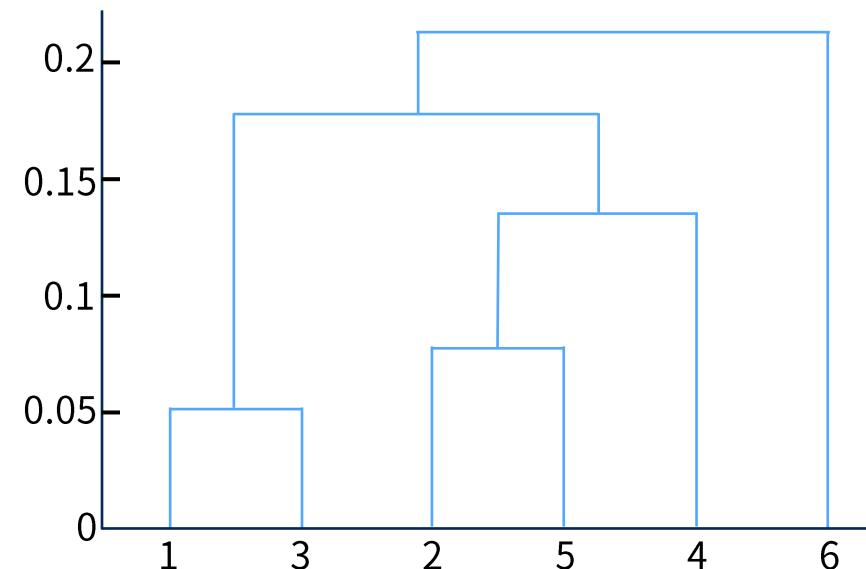
- ✓ Total = Error + Complexity



▣ 결과에 크게 개선하지 않는 경우, 더 적은 클러스터의 개수를 선택

3. 계층 클러스터링 (hierarchical clustering)

- ▣ 데이터 간 계층을 기반으로 데이터 간 병합 또는 분할을 통해 해당 데이터가 속할 그룹을 결정
- ▣ 중첩된 클러스터의 집합을 계층적 트리 형태로 조직화
- ▣ 덴드로그램 (dendrogram) 으로 시각화
 - ✓ 병합 또는 분할된 순서를 기록한 도표와 같은 트리





3. 계층 클러스터링 (hierarchical clustering)

▣ 클러스터의 개수를 미리 지정할 필요가 없음

- ✓ 각 레벨에서 데이터가 어떻게 분리되는지 시각적으로 확인이 가능
- ✓ 덴드로그램을 원하는 레벨에서 “cutting”함으로써 원하는 클러스터의 개수의 결과를 획득

▣ 각각은 의미 있는 분류(taxonomies)에 대응



3. 계층 클러스터링 (hierarchical clustering)

▣ 계층 클러스터링의 유형

- ✓ 병합 (agglomerative):

- 개별 데이터 포인트를 하나의 클러스터로 설정하고 시작
- 각 단계에서 유사도가 가장 높은 클러스터 쌍을 병합하여 최종적으로 데이터 셋 전체가 하나의 클러스터로 묶이는 방법

- ✓ 분할 (divisive):

- 전체 데이터 셋을 하나의 클러스터로 놓고 시작
- 클러스터 내부에서 가장 멀리 떨어진 데이터를 다른 클러스터로 분리하는 과정의 반복을 통해, 마지막에는 클러스터가 데이터 개수만큼 분리하는 방법

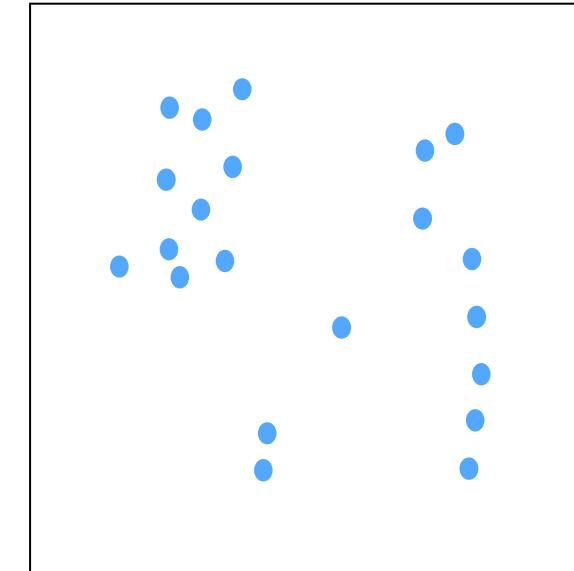
▣ 계층 클러스터링은 유사도 또는 거리 행렬을 사용

- ✓ 매번 하나의 클러스터를 병합 또는 분리

4. 병합 계층 클러스터링 (Agglomerative clustering)

- ✓ 많이 사용하는 대표적인 클러스터링 방법
- ✓ 클러스터 간의 거리/유사도 정의
- ✓ 초기: 각 데이터가 하나의 클러스터로 배정
- ✓ 반복:
 - 모든 클러스터 간에 거리/유사도를 계산
(클러스터 간의 관계를 proximity matrix로 저장)
 - 가장 가까운 두 개의 클러스터를 병합
- ✓ 클러스터 간의 병합 순서를 모두 저장
- ✓ “Dendrogram”

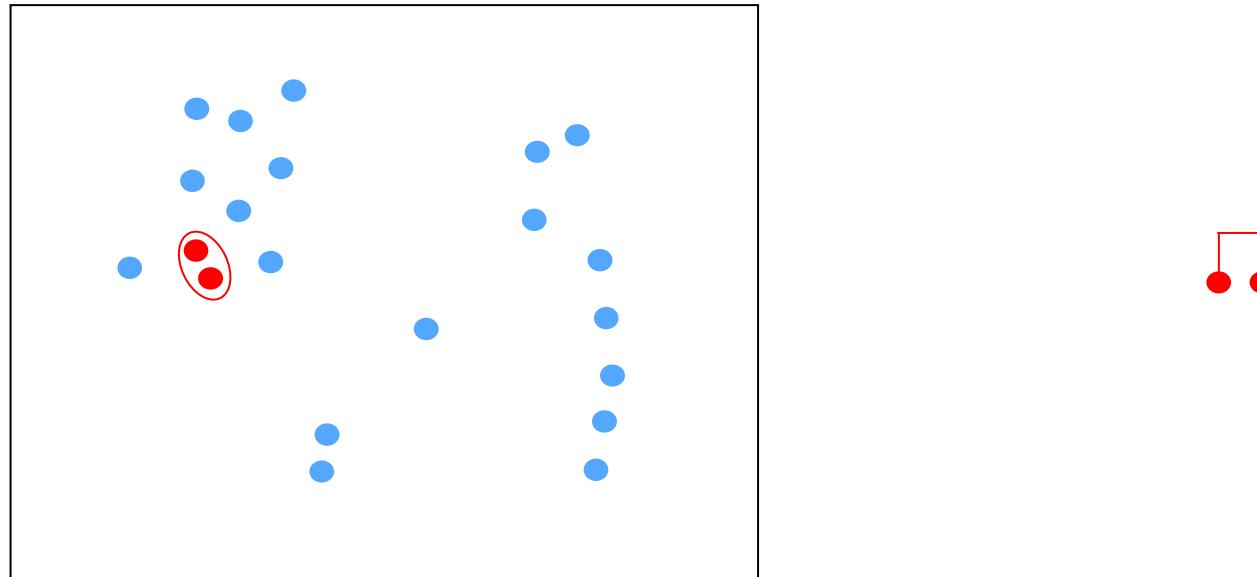
초기에는 각 데이터가
하나의 클러스터로 배정





4. 병합 계층 클러스터링 (Agglomerative clustering)

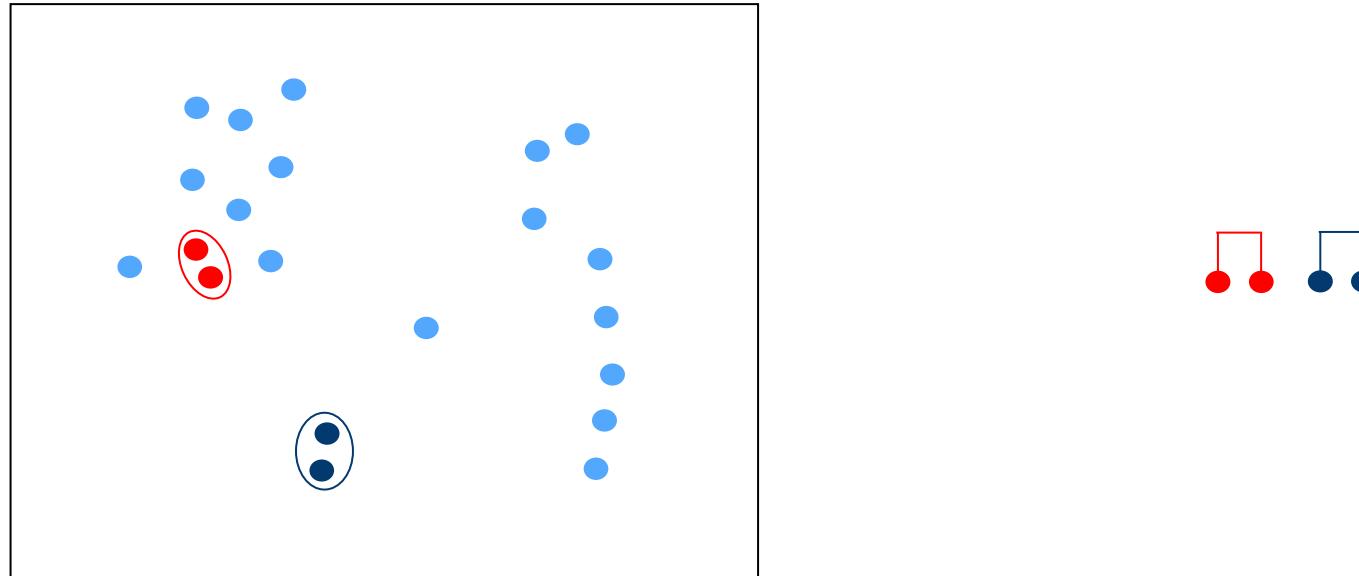
Iteration 1





4. 병합 계층 클러스터링 (Agglomerative clustering)

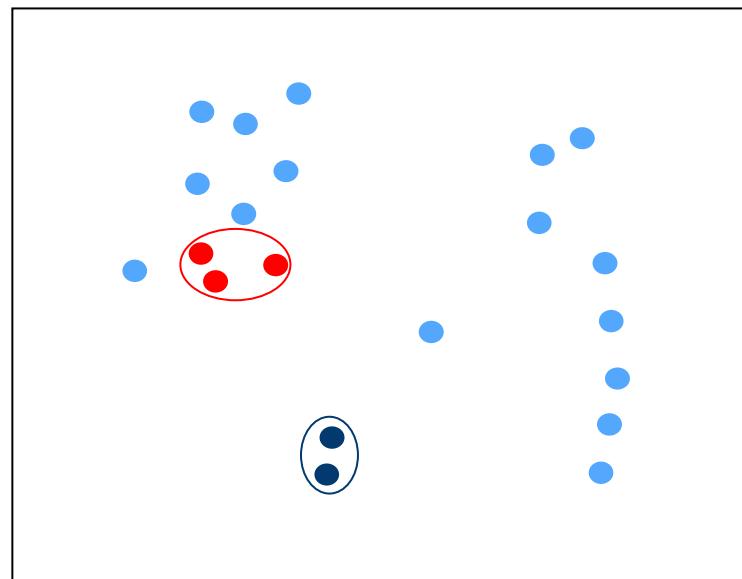
Iteration 2



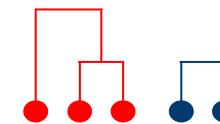


4. 병합 계층 클러스터링 (Agglomerative clustering)

Iteration 3



- Builds up a sequence of clusters (“hierarchical”)



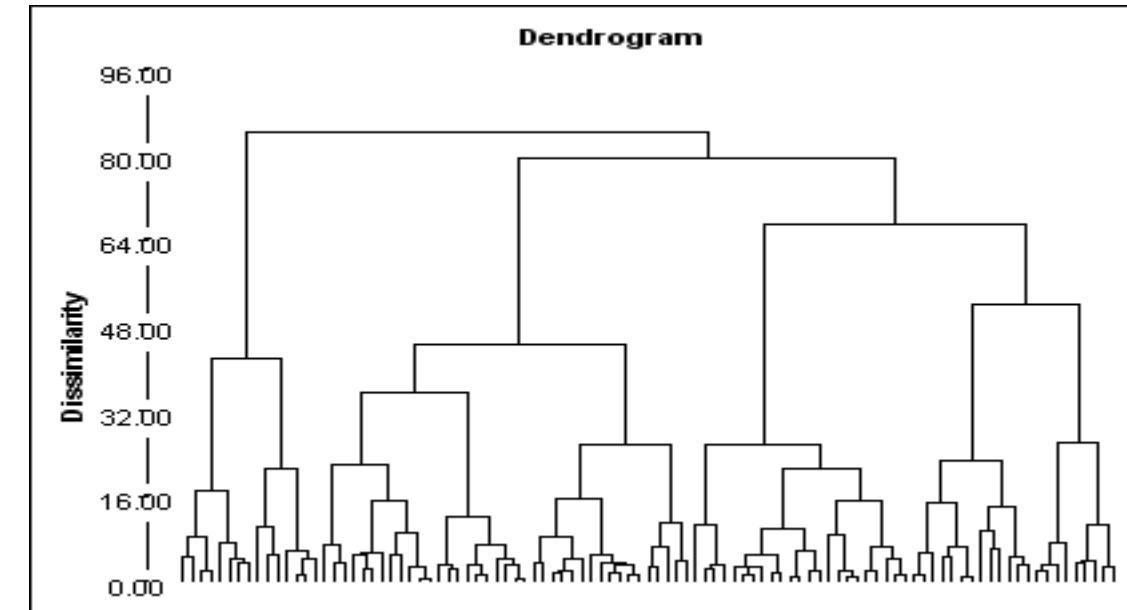
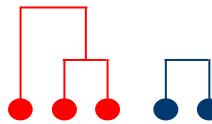
- Algorithm complexity $O(N^2)$ (Why?)

In matlab: “linkage” function (stats toolbox)



4. 병합 계층 클러스터링 (Agglomerative clustering)

Dendrogram





5. 병합 계층 클러스터링 알고리즘

▣ 가장 많이 사용되는 계층 클러스터링 기술

▣ Basic algorithm

1. 유사도행렬 (proximity matrix)를 계산
2. 각 데이터포인트가 하나의 클러스터로 초기화

3. Repeat

4. 가장 가까운 클러스터 쌍을 병합
5. Proximity matrix를 업데이트

6. Until 하나의 클러스터만 남을 때까지

▣ 가장 중요한 연산은 두 클러스터 간의 유사도를 계산하는 것

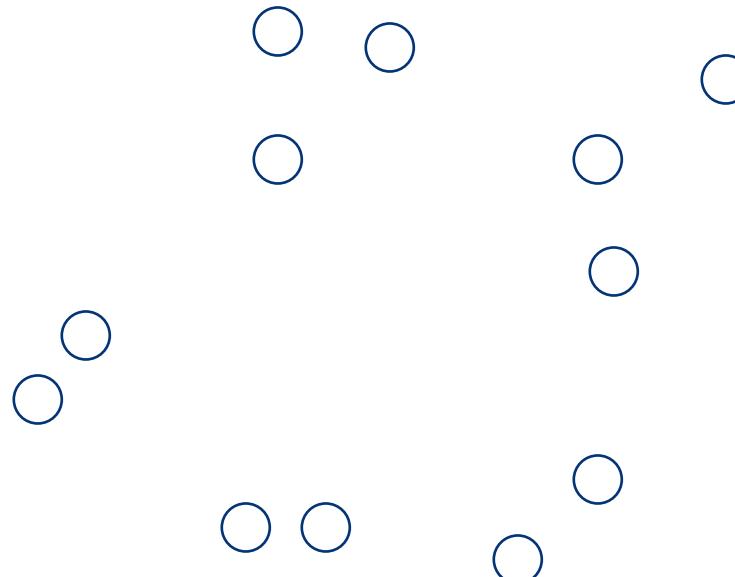
- 클러스터 간의 거리를 정의하는 방법에 따라 다양한 알고리즘이 구분됨



5. 병합 계층 클러스터링 알고리즘

▣ 초기 상태

- Start with clusters of individual points and a proximity matrix



	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						

Proximity Matrix

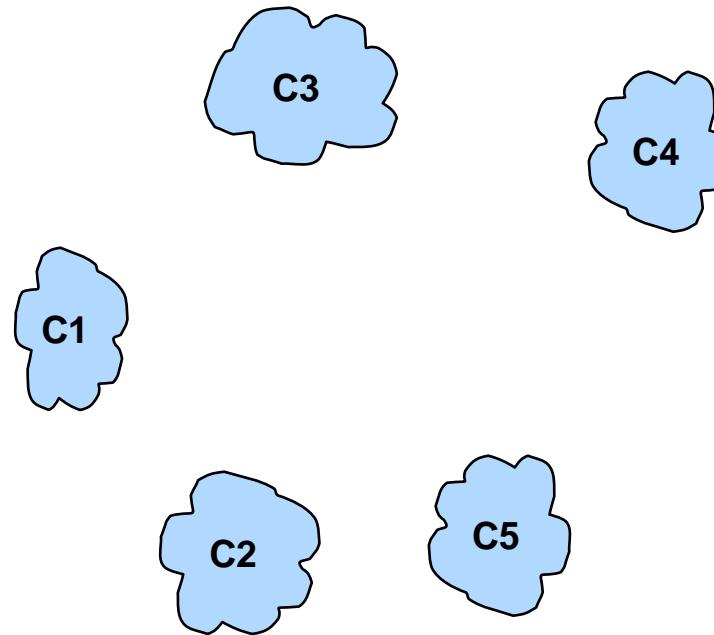
p1 p2 p3 p4 ... p9 p10 p11 p12



5. 병합 계층 클러스터링 알고리즘

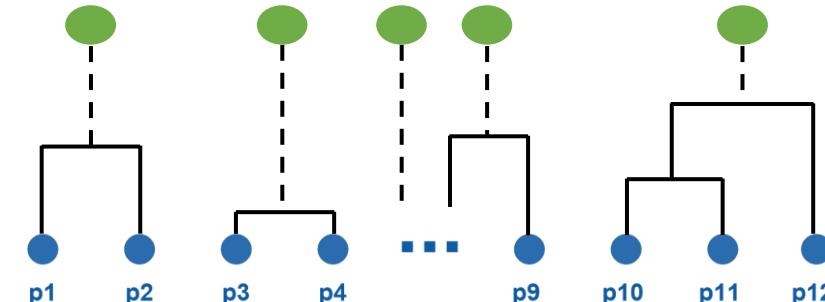
Intermediate 상태

- 병합 후에 클러스터들의 정보를 업데이트



	C1	C2	C3	C4	C5
C1					
C2					
C3					
C4					
C5					

Proximity Matrix

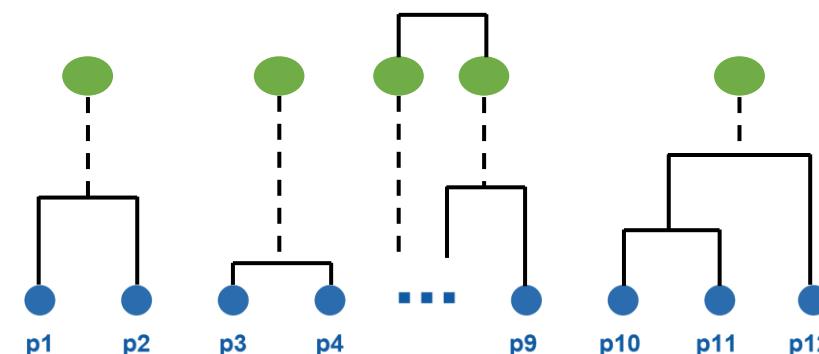
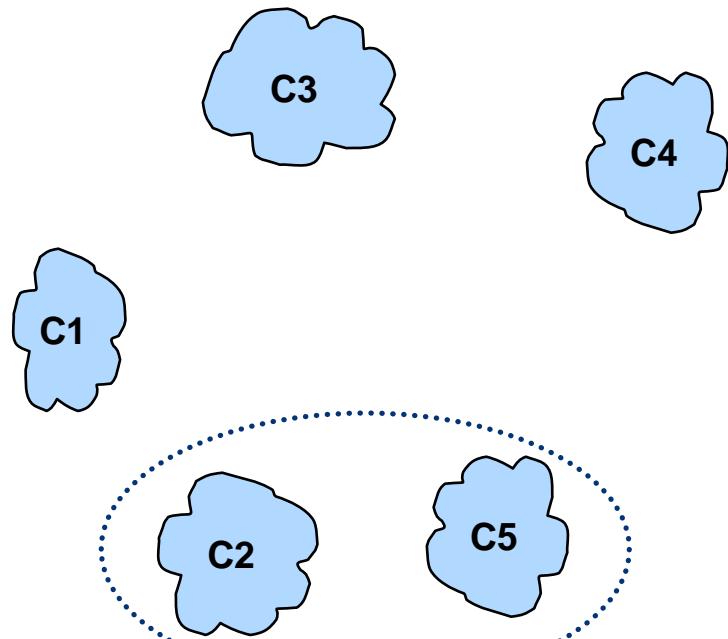




5. 병합 계층 클러스터링 알고리즘

■ Intermediate 상태

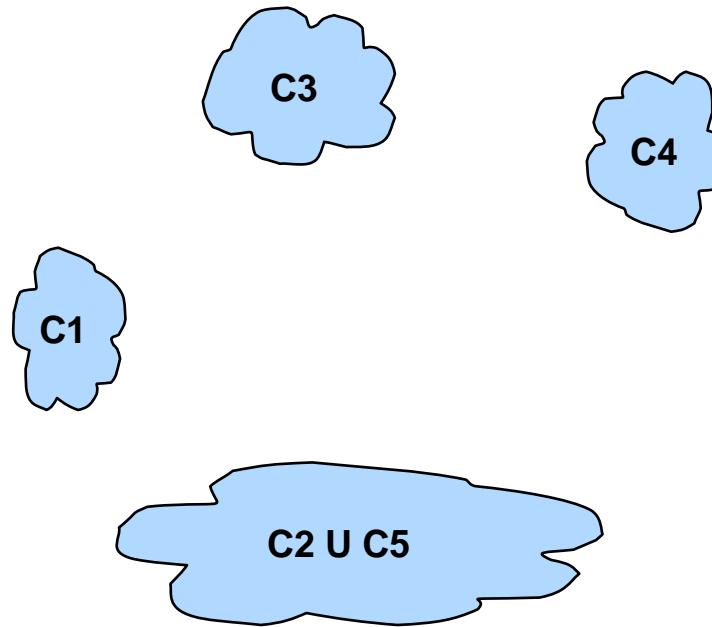
- 가장 가까운 클러스터링쌍 (C2 와 C5)를 병합하고 proximity matrix를 업데이트.





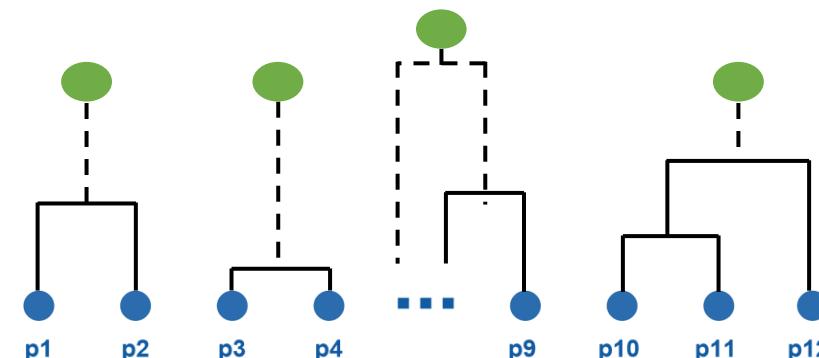
5. 병합 계층 클러스터링 알고리즘

- ✓ 어떻게 proximity matrix를 업데이트 할 것인가?



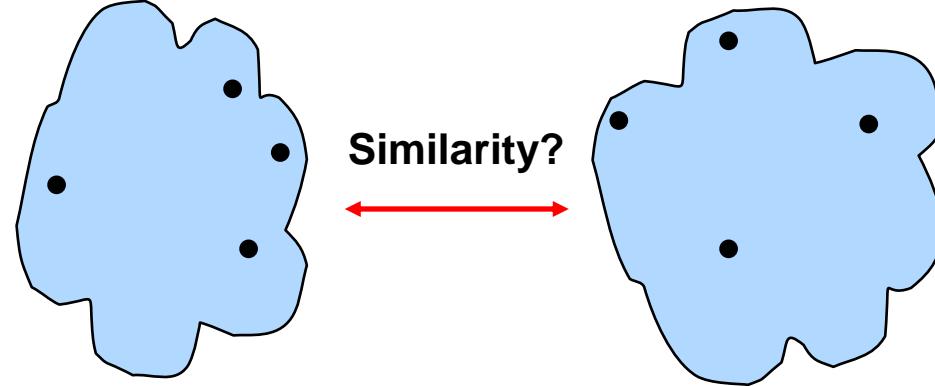
		C2 U			
		C1	C5	C3	C4
C2 U C5	C1				
	C2 U C5				
	C3				
C4					

Proximity Matrix





5. 클러스터 간 유사도 정의 (Inter-cluster similarity)



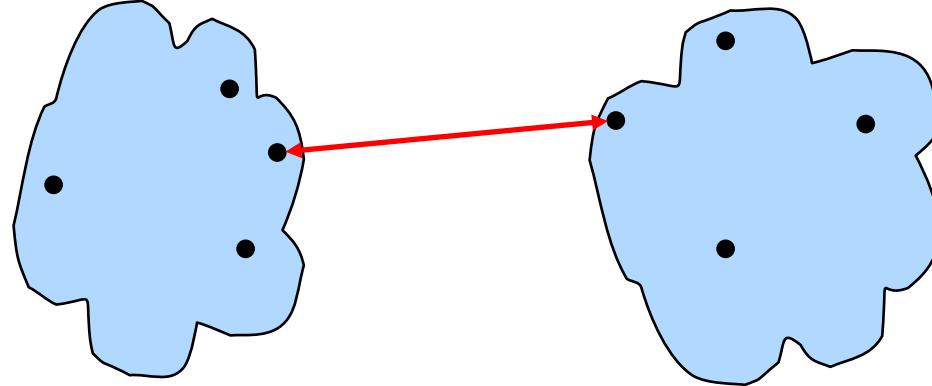
- ✓ MIN
- ✓ MAX
- ✓ Group Average
- ✓ Distance Between Centroids

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						

Proximity Matrix



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)



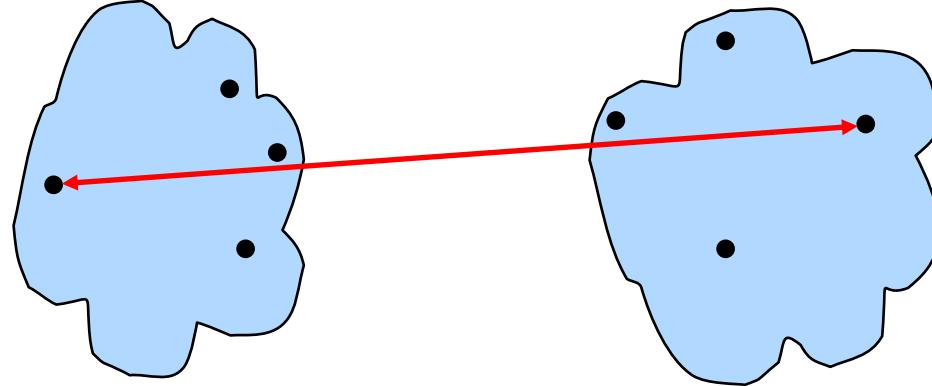
- ✓ MIN
- ✓ MAX
- ✓ Group Average
- ✓ Distance Between Centroids

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						

· Proximity Matrix



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)

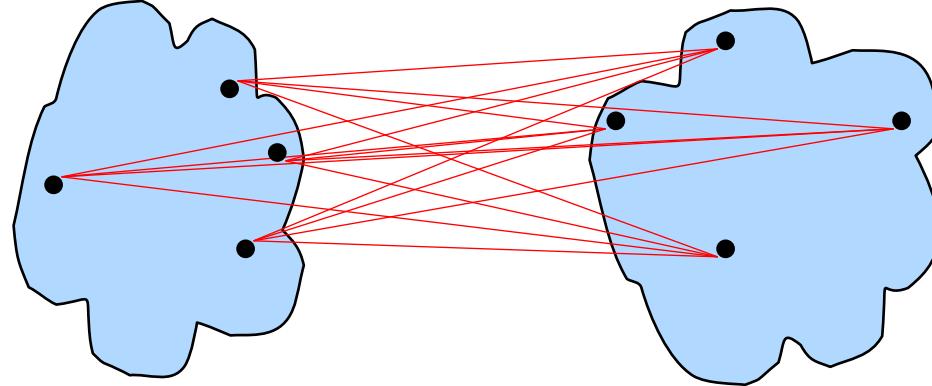


- ✓ MIN
- ✓ MAX
- ✓ Group Average
- ✓ Distance Between Centroids

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						
.						
Proximity Matrix						



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)



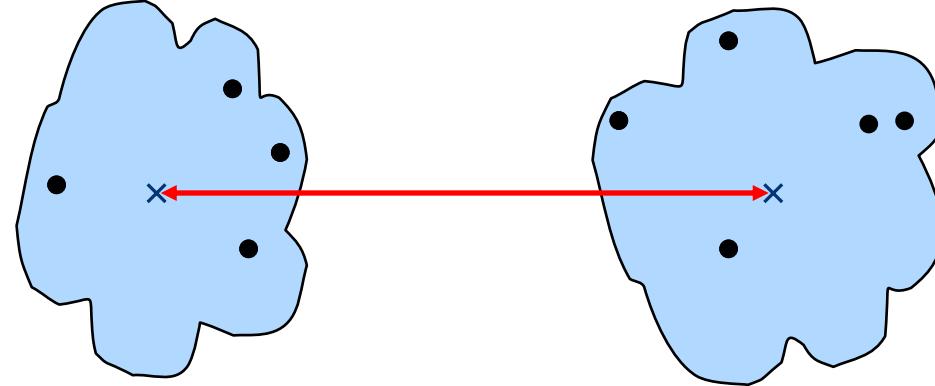
- ✓ MIN
- ✓ MAX
- ✓ Group Average
- ✓ Distance Between Centroids

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						

Proximity Matrix



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)



- ✓ MIN
- ✓ MAX
- ✓ Group Average
- ✓ Distance Between Centroids

	p1	p2	p3	p4	p5	...
p1						
p2						
p3						
p4						
p5						
.						
.						

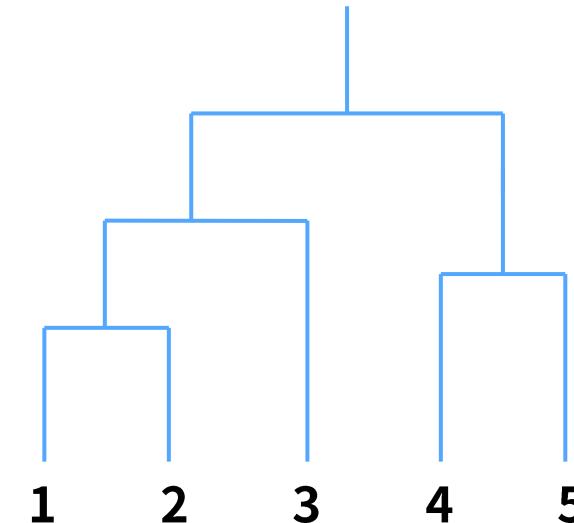
Proximity Matrix

6. 클러스터 간 유사도 정의 (Inter-cluster similarity)

■ Cluster Similarity: MIN (Single Link)

- ✓ 가장 가까운 거리에 있는 데이터를 연결하는 방법
 - 하나의 클러스터쌍을 결정.

	I1	I2	I3	I4	I5
I1	1.00	0.90	0.10	0.65	0.20
I2	0.90	1.00	0.70	0.60	0.50
I3	0.10	0.70	1.00	0.40	0.30
I4	0.65	0.60	0.40	1.00	0.80
I5	0.20	0.50	0.30	0.80	1.00

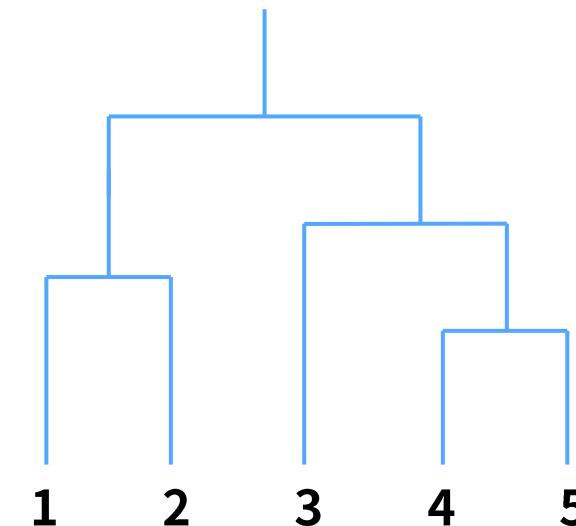


6. 클러스터 간 유사도 정의 (Inter-cluster similarity)

Cluster Similarity: MAX (Complete Linkage)

- 클러스터 간 가장 비슷하지 않은 샘플을 비교하여 병합

	I1	I2	I3	I4	I5
I1	1.00	0.90	0.10	0.65	0.20
I2	0.90	1.00	0.70	0.60	0.50
I3	0.10	0.70	1.00	0.40	0.30
I4	0.65	0.60	0.40	1.00	0.80
I5	0.20	0.50	0.30	0.80	1.00



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)

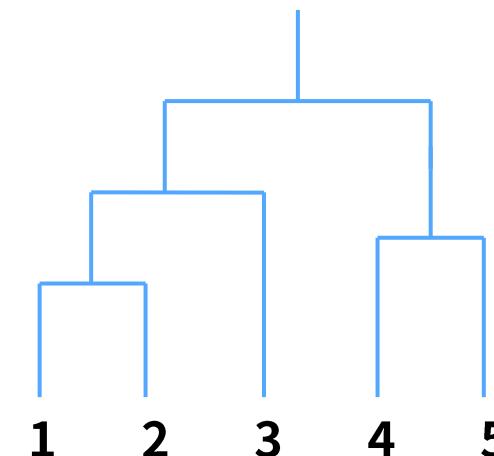
Cluster Similarity: Group Average

- 클러스터 간의 유사도는 두 개의 클러스터 내에 존재하는 모든 데이터 쌍간의 평균 유사도로 정의

$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| * |\text{Cluster}_j|}$$

- 전체 유사도가 큰 사이즈의 클러스터를 선호하기 때문에, 크기값으로 정규화

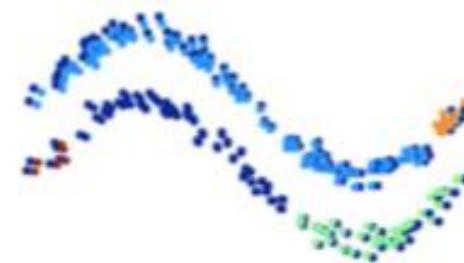
	I1	I2	I3	I4	I5
I1	1.00	0.90	0.10	0.65	0.20
I2	0.90	1.00	0.70	0.60	0.50
I3	0.10	0.70	1.00	0.40	0.30
I4	0.65	0.60	0.40	1.00	0.80
I5	0.20	0.50	0.30	0.80	1.00



6. 클러스터 간 유사도 정의 (Inter-cluster similarity)

▣ 다른 거리 정의에 따라 클러스터링 결과가 달라짐

Single linkage (min)



Complete linkage (max)



정리하기

▶ 병합 계층 클러스터링 (Agglomerative Clustering)

- ▶ 클러스터 간의 유사도/거리를 이용
- ▶ 지속적으로 가장 가까운 클러스터 쌍을 병합
- ▶ “Dendrogram”: 클러스터 간의 병합 순서 및 거리/유사도를 기록

정리하기

- ▶ 병합 클러스터링은 “proximity matrix”에 기반함
- ▶ 데이터 간의 관계를 이해할 수 있는 “Clustergram”

들어가기

| 학습목표 |

- 비지도학습의 알고리즘인 클러스터링의 기본 개념을 이해
- Partitional clustering과 hierarchical clustering 알고리즘을 학습하고, 이들의 장단점을 비교
- K-Means clustering을 구현

들어가기

| 학습목표 |

- Agglomerative hierarchical clustering의 프로세스를 이해
- Agglomerative clustering에서 proximity matrix의 계산법을 이해

들어가기

| 학습목차 |

1 | 클러스터링 (Clustering)

2 | K-평균 클러스터링 (K-Means Clustering)

3 | 병합 계층 클러스터링 (Agglomerative Clustering)



0. 비지도 학습 (unsupervised learning)

■ 지도 학습 (Supervised Learning)

- ✓ labeled data를 제공 : $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$

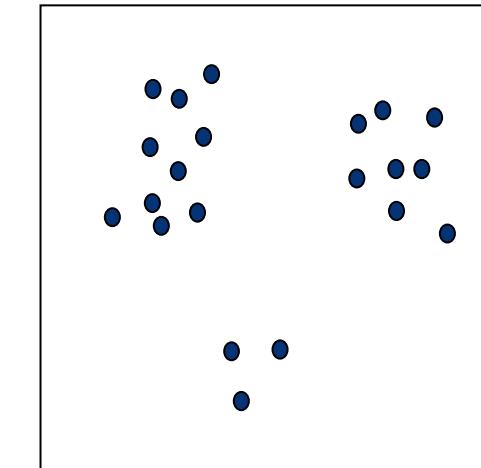
■ 비지도 학습 (unsupervised learning)

- ✓ 라벨링이 되어 있지 않은 데이터를 이용하는 학습 방법(just “x”)

• labeled data를 제공 : $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$

다양한 분야에서 활용

- 데이터 마이닝 (“explain”)
- Missing data values (“impute”)
- feature generation or selection



■ 대표적인 기술: *clustering*



0. 비지도 학습 (unsupervised learning)

▣ 지도 학습 (Supervised learning)

- ✓ labeled data를 제공 : $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$

▣ 비지도 학습 (unsupervised learning)

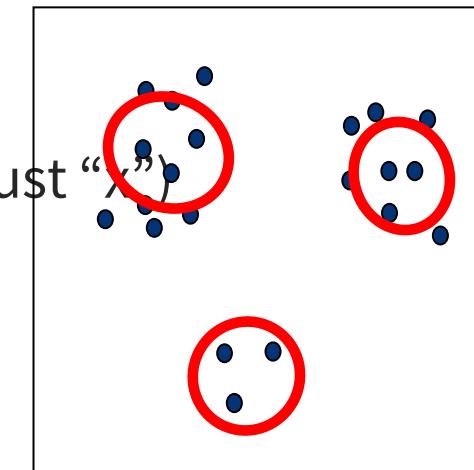
• labeled data를 제공 : $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$

라벨링이 되어 있지 않은 데이터를 이용하는 학습 방법(just “X”)

• labeled data를 제공 : $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$

다양한 분야에서 활용

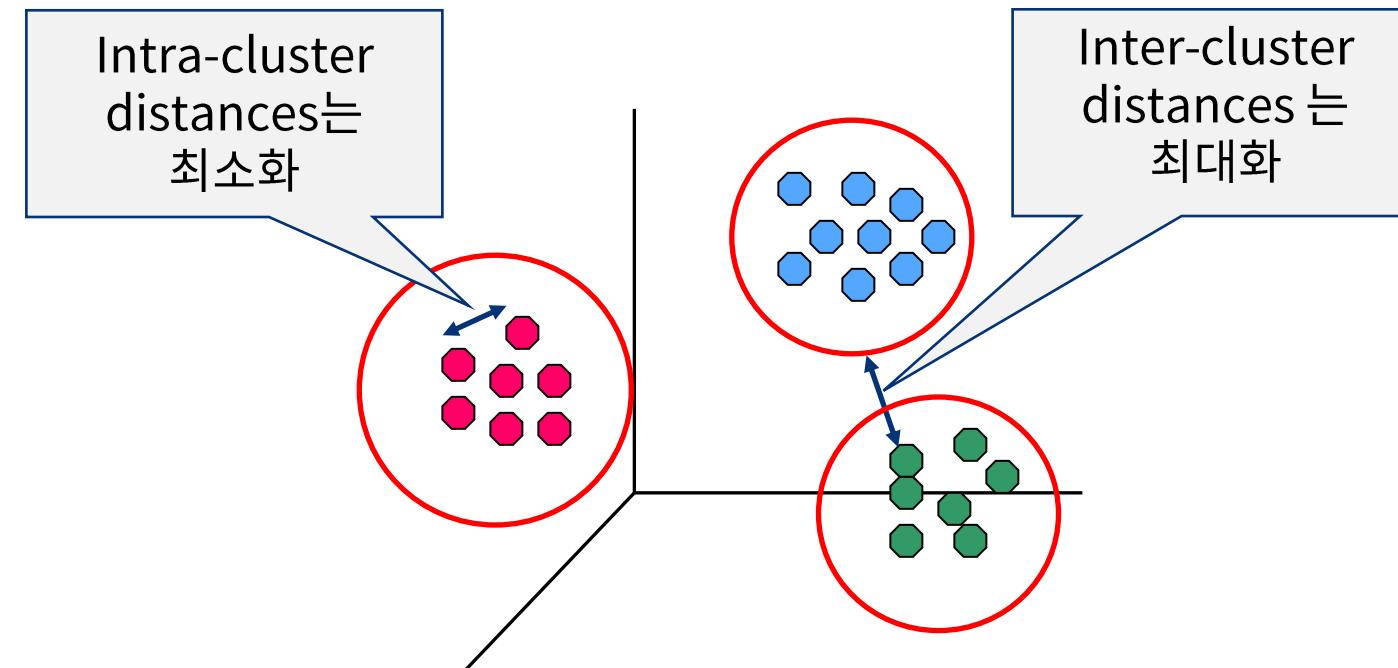
- 데이터 마이닝 (“explain”)
- Missing data values (“impute”)
- feature generation or selection



▣ 대표적인 기술: *clustering*

1. 클러스터링 (Clustering)

- 비슷한 샘플을 클러스터 (cluster)로 모음
- 서로 유사한 객체들은 하나의 클러스터로, 관련이 없거나 차이가 많은 객체들은 다른 클러스터로 구분하여, 객체들의 그룹을 발견하는 기술

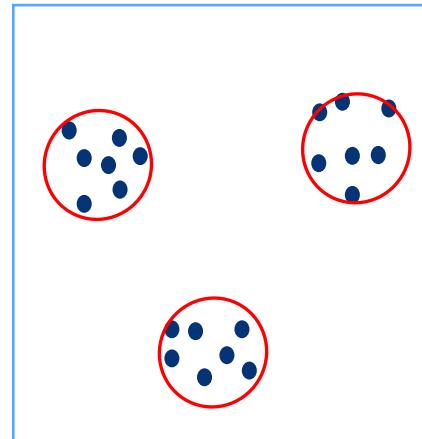




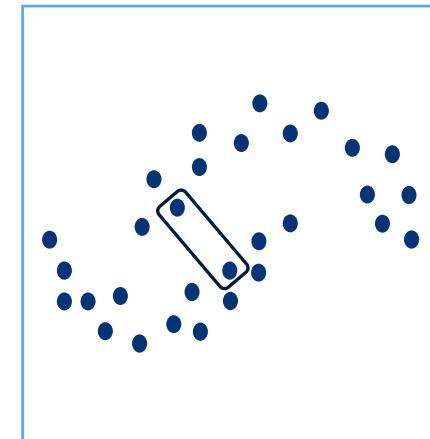
1. 클러스터링 (Clustering)

- 클러스터링은 데이터들을 “그룹 (group)” 단위로 표현
- 그룹의 의미는 데이터별로 다양해 질 수 있음

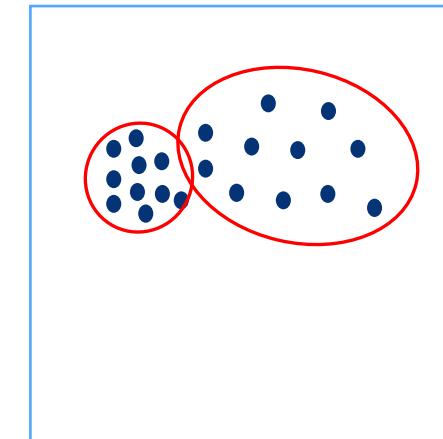
- ✓ 예제



Location



Shape



Density



1. 클러스터링 (Clustering)

▣ 클러스터링의 활용

✓ Understanding

- 유사한 기능을 가지는 유전자와 단백질을 그룹화
- 유사한 가격변동 흐름을 가지는 주식들을 그룹화
- 고객 분류

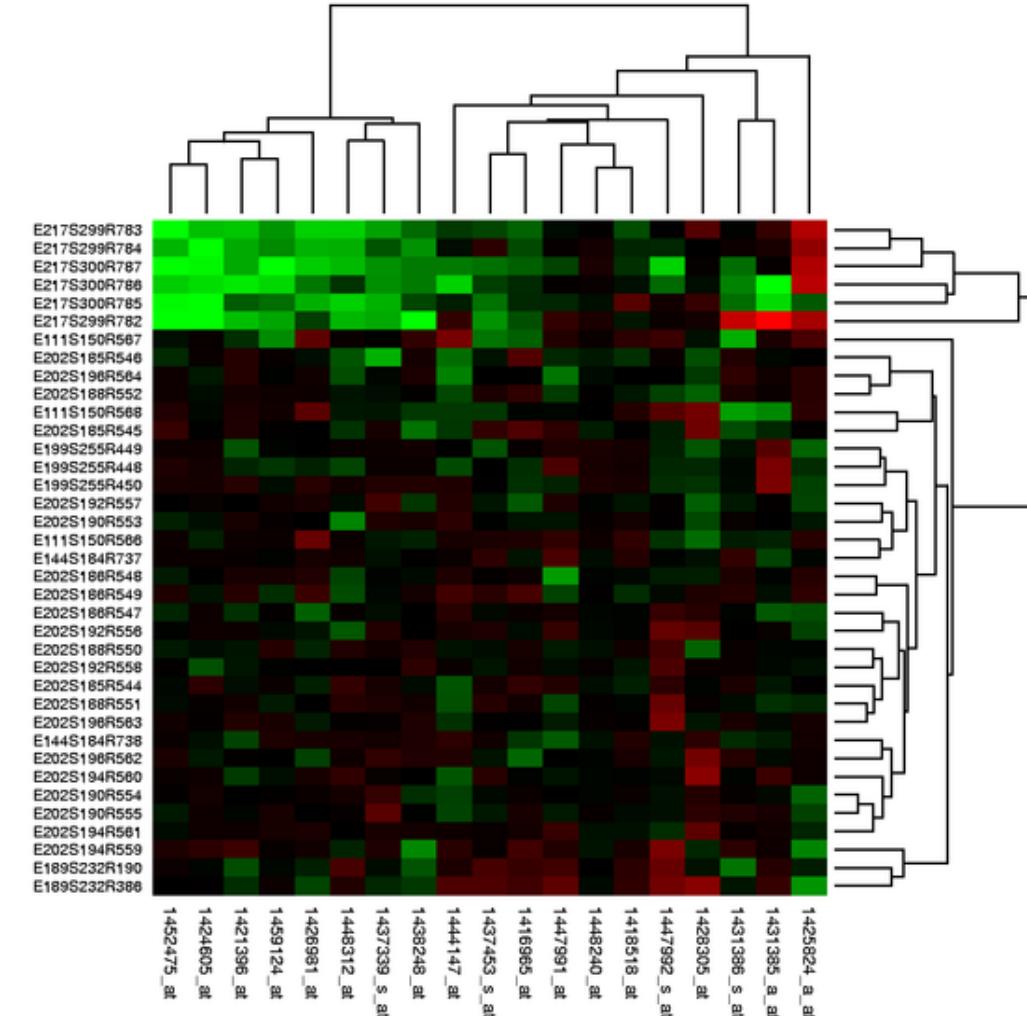
✓ Summarization

- 대용량 데이터셋의 크기를 축소



1. 클러스터링 (Clustering)

Understanding 의 예
: Gene clustering





1. 클러스터링 (Clustering)

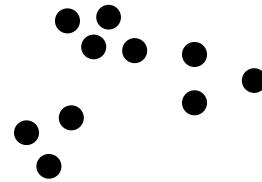
Summarization의 예: face clustering



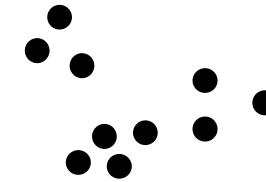


1. 클러스터링 (Clustering)

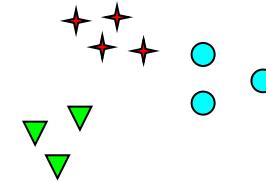
▣ 클러스터링의 개념은 다소 애매함 (ambiguous)



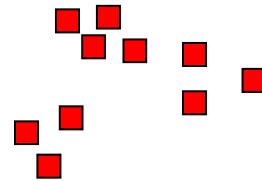
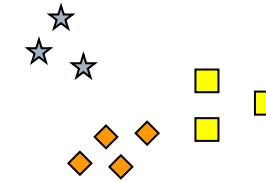
적합한 클러스터의 개수?



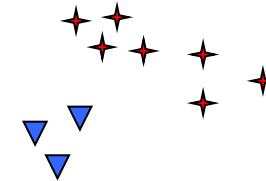
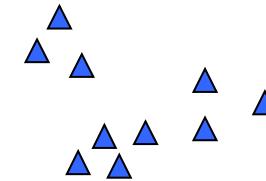
Six Clusters



Four Clusters



Two Clusters





1. 클러스터링 (Clustering)

▣ 클러스터링의 종류

- Partitional Clustering

- 데이터를 클러스터 (non-overlapping subsets)으로 나눔,
즉 하나의 데이터는 정확하게 하나의 클러스터에 할당

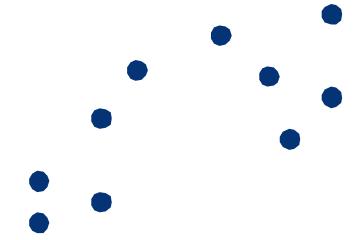
- Hierarchical clustering (계층 클러스터링)

- 클러스터들을 계층적인 tree 형태로 조직화

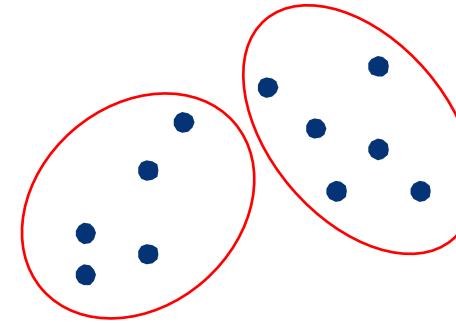


1. 클러스터링 (Clustering)

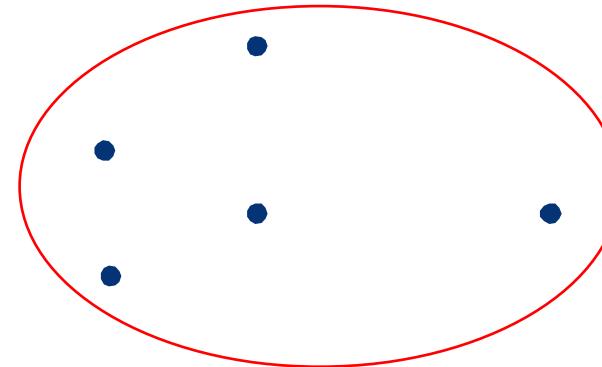
Partitional Clustering



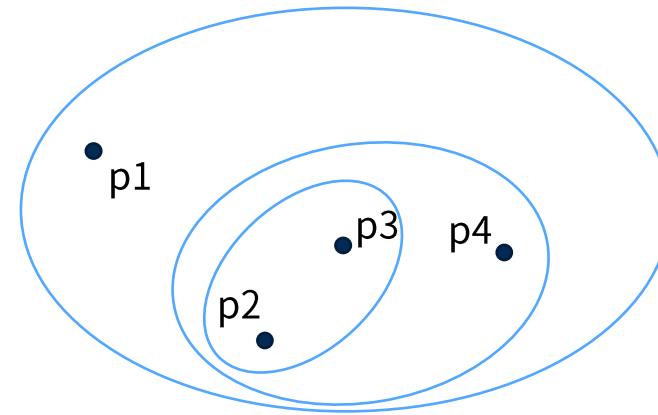
입력



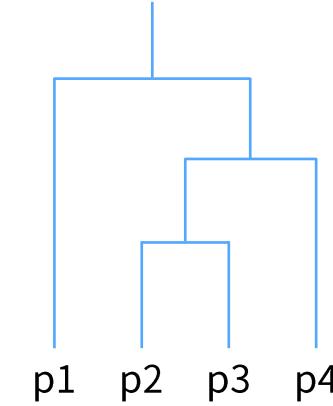
클러스터링 결과



Hierarchical Clustering



Hierarchical Clustering



Dendrogram



2. K-평균 클러스터링(K-Means Clustering)

■ K-means Clustering

- ✓ 가장 단순한 partitional clustering 기술
- ✓ 각 클러스터들은 하나의 중심 (a center point)에 의해 표현 (summarizing)
- ✓ 각 데이터 포인트와 각 클러스터 평균 간의 거리를 구한 후 가장 가까운 클러스터로 배정
- ✓ 클러스터의 개수 K 는 사전에 지정
- ✓ 기본 알고리즘
 - 그룹 할당: 각 데이터의 소속 클러스터 업데이트
 - 그룹 평균 업데이트: 각 클러스터의 센터를 업데이트

1: Select K points as the initial centroids.

2: **repeat**

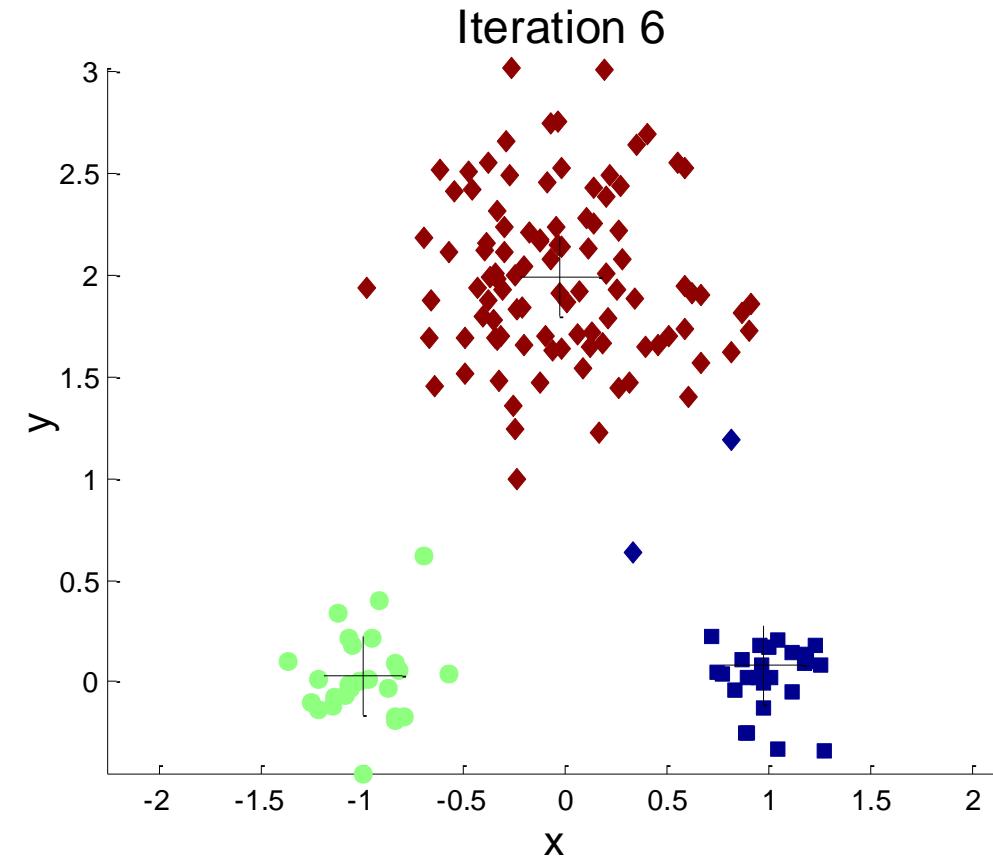
3: Form K clusters by assigning all points to the closest centroid.

4: Recompute the centroid of each cluster.

5: **until** The centroids don't change

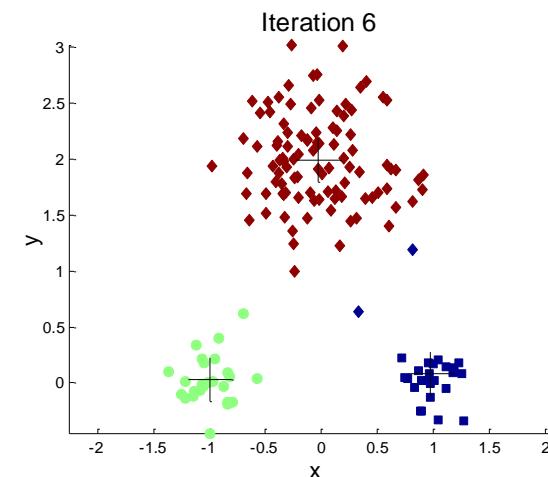
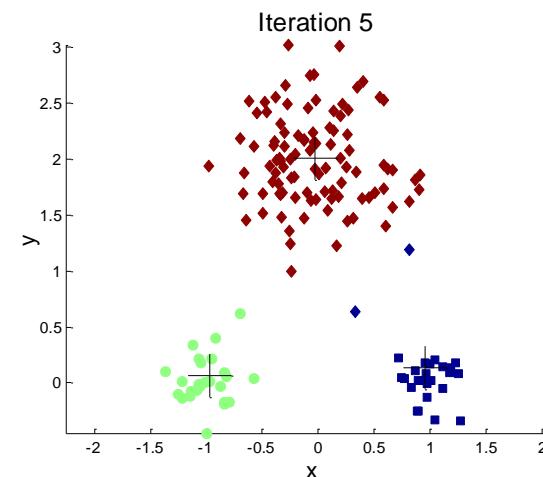
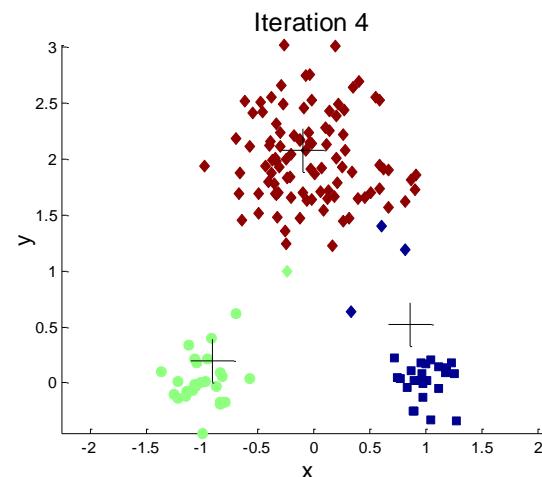
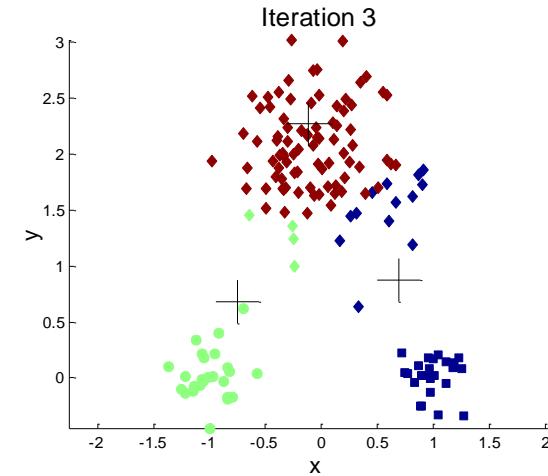
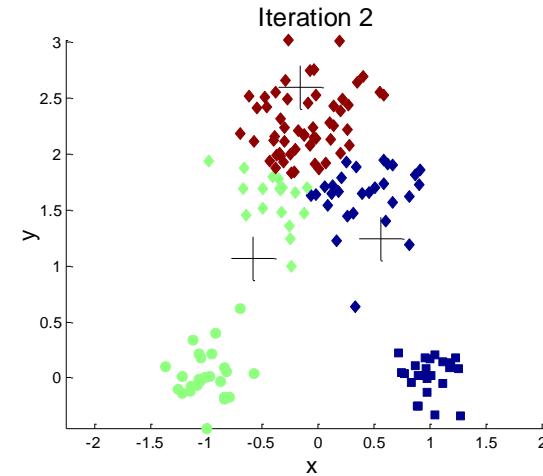
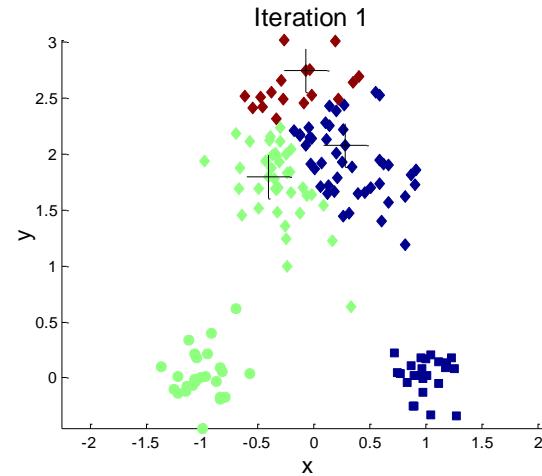


2. K-평균 클러스터링(K-Means Clustering)





2. K-평균 클러스터링(K-Means Clustering)





2. K-평균 클러스터링(K-Means Clustering)

▣ 각 클러스터의 센터를 초기화, 가장 기본적인 방법은 랜덤으로 평균을 초기화

- ✓ 매번 실행할 때마다, 클러스터가 다르게 초기화 됨

▣ 클러스터의 센터는 그 클러스터에 소속된 데이터 포인트들의 평균값으로 설정

▣ ‘Closeness/Distance’ 정의

- ✓ Euclidean distance, cosine similarity, correlation 등으로 정의

▣ 대부분의 경우, 초기 몇번의 반복을 통해 수렴에 도달

- ✓ 종료 조건: “기존의 클러스터와 새롭게 할당 받은 클러스터와의 차이가 거의 없을 때”로 설정



2. K-평균 클러스터링(K-Means Clustering)

▣ 알고리즘

- ✓ 수렴할 때까지 반복:

{

각 데이터 x_i 에 대해, 소속 클러스터 z_i 를 결정

$$z_i = \arg \min_c \|x_i - \mu_c\|, \forall i$$

각 클러스터 c 에 대해서, 센터 μ_c 를 계산

$$\mu_c = \frac{1}{m_c} \sum_{i \in S_c} x_i, \quad S_c = \{i : z_i = c\}, m_c = |S_c|$$

}



2. K-평균 클러스터링(K-Means Clustering)

▣ 목적 함수: $C(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$

▣ Coordinate descent 최적화 기법

▣ 클러스터 할당 (fixed μ_c)

- ✓ 목적함수에서 z_i 만이 변경
- ✓ 가장 가까운 μ_c 를 선택함으로써 함수 C 를 최소화

▣ 클러스터 센터 업데이트 (fixed z_i)

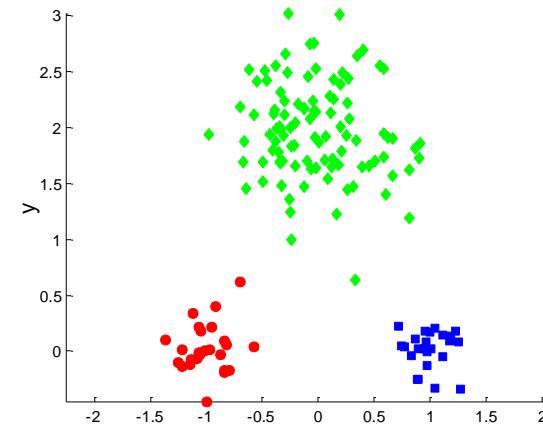
- ✓ 각 클러스터 c 에 소속된 x_i 에 대해서 계산을 수행
- ✓ 평균 업데이트를 통해 함수 C 를 최소화

몇 번의 반복 후, 수렴에 도달

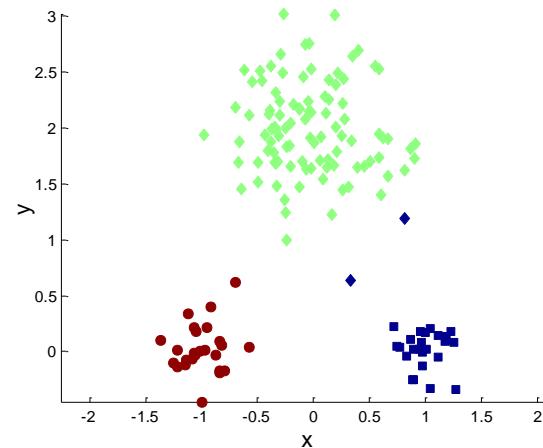


2. K-평균 클러스터링(K-Means Clustering)

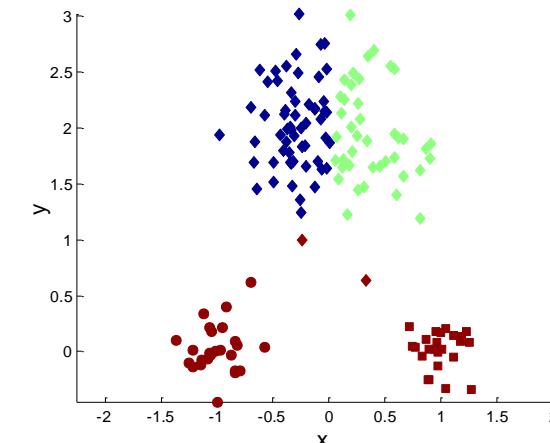
▣ 클러스터링의 서로 다른 예



입력 데이터



Optimal Clustering



Sub-optimal Clustering



3. K-평균 클러스터링 (K-Means clustering)의 문제점

- ✓ 초기 K 값을 사용자가 결정
- ✓ 센터들의 초기값을 정확하게 설정하기가 어려움 (random initialization)
- ✓ 클러스터 간 데이터의 밀도 차이가 있을 경우 클러스터링이 잘 되지 않음



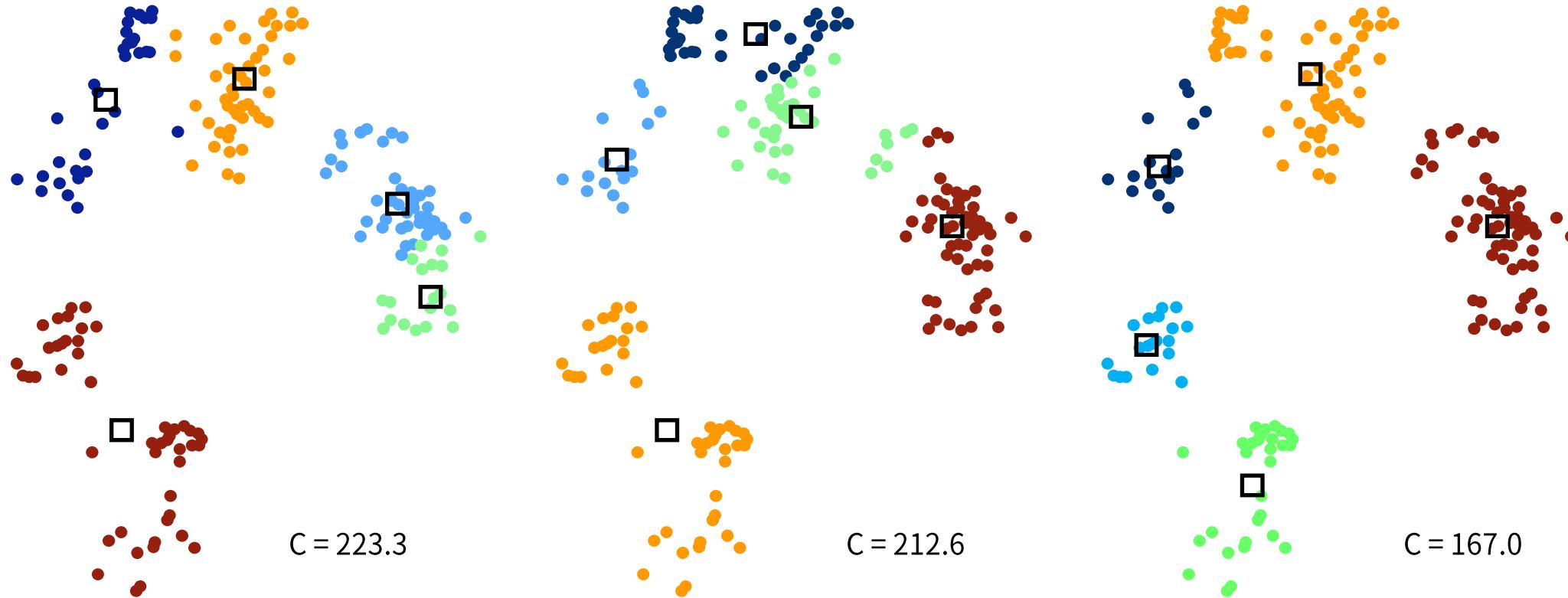
- ✓ 여러 번 반복해서, 최적의 결과를 선택
- ✓ 다른 초기화 전략을 사용
- ✓ 목적함수 C를 활용



4. 초기값 설정(center initialization)

■ 여러 번 반복해서, 최적의 결과를 선택

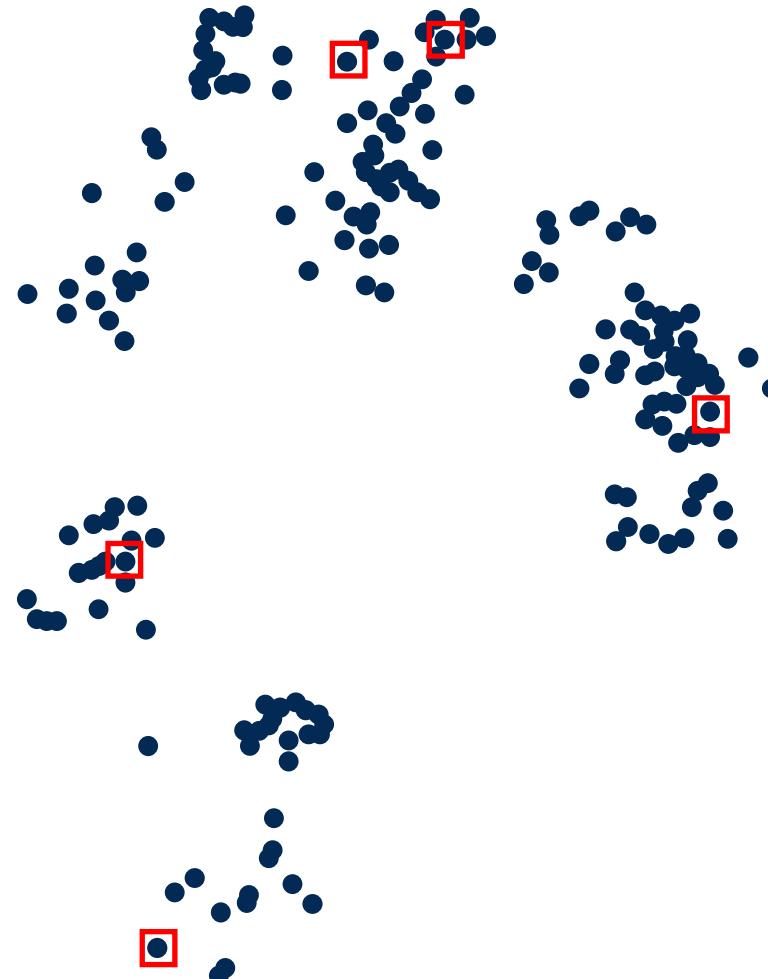
■ 목적함수 C를 활용



4. 초기값 설정(center initialization)

Random initialization

- ✓ 데이터 인덱스로부터 랜덤하게 선택
- ✓ 문제점: 서로 가까운 데이터들이 선택될 수 있음

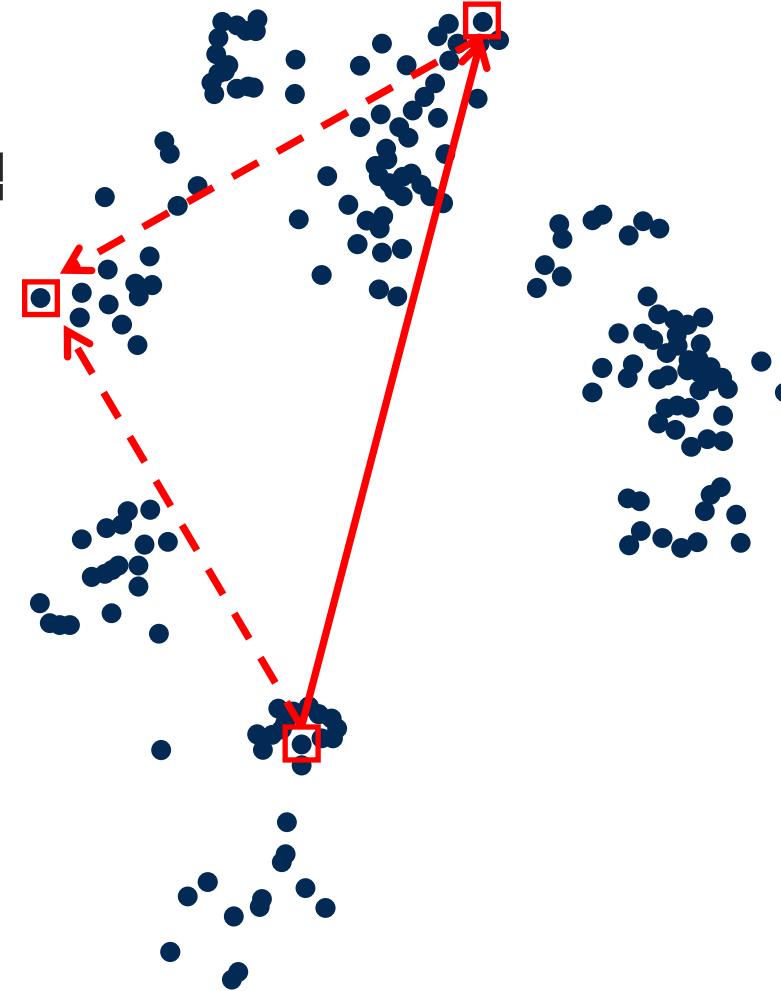




4. 초기값 설정(center initialization)

Distance based initialization

- ✓ 하나의 랜덤 데이터로부터 시작
- ✓ 지금까지의 클러스터 센터로부터 가장 먼 데이터를 선택
- ✓ 문제점: outliers 를 선택

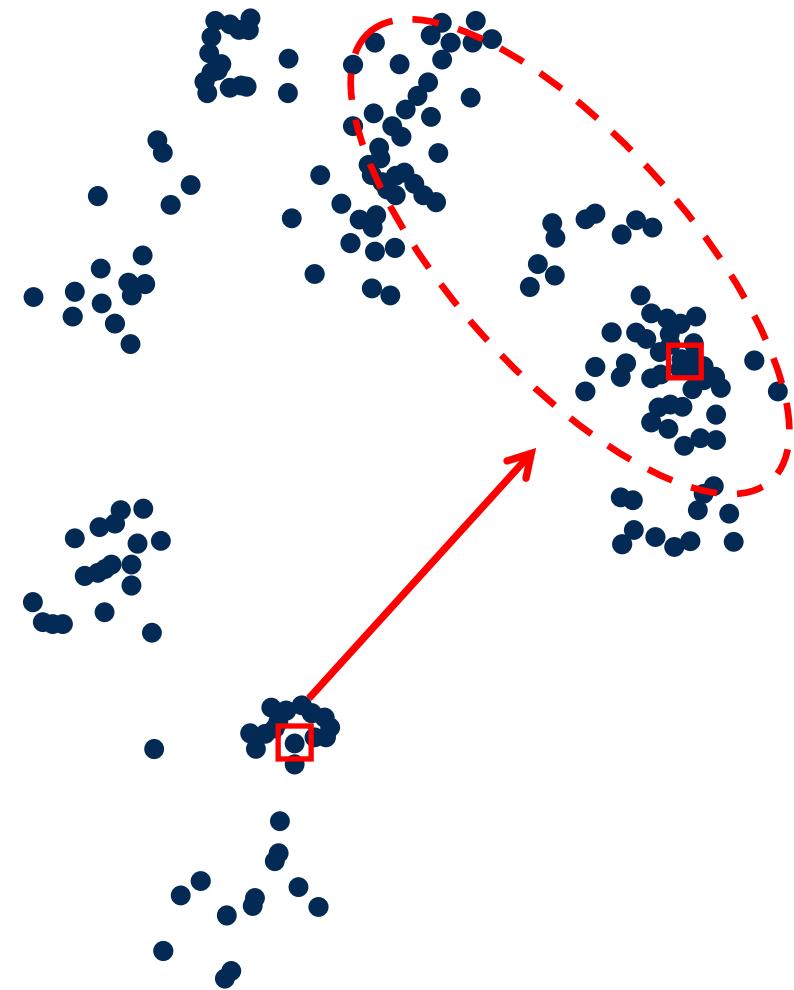




4. 초기값 설정(center initialization)

■ Random + Distance (“kmeans++”)

- ✓ 하나의 랜덤 데이터로부터 시작
- ✓ “far but randomly” 데이터를 선택
- ✓ 문제점: 많은 데이터를 가진 하나의 영역에서 편향된 클러스터 센터를 설정

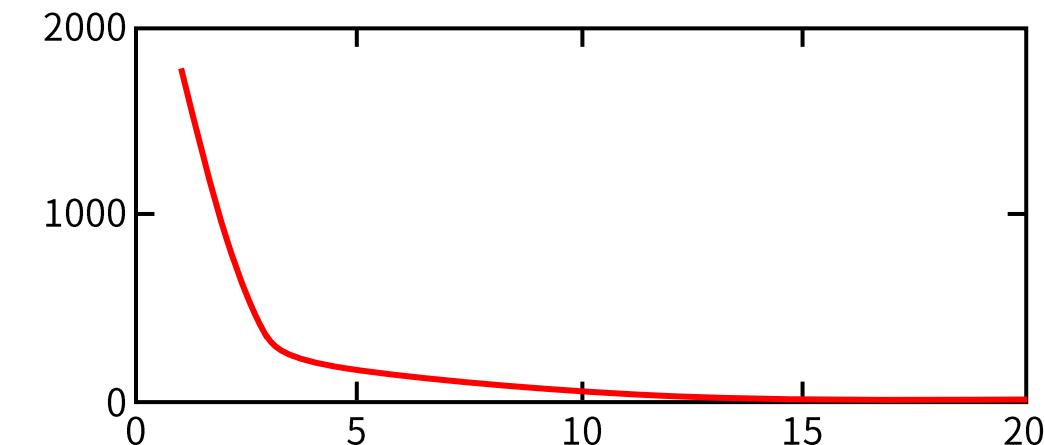
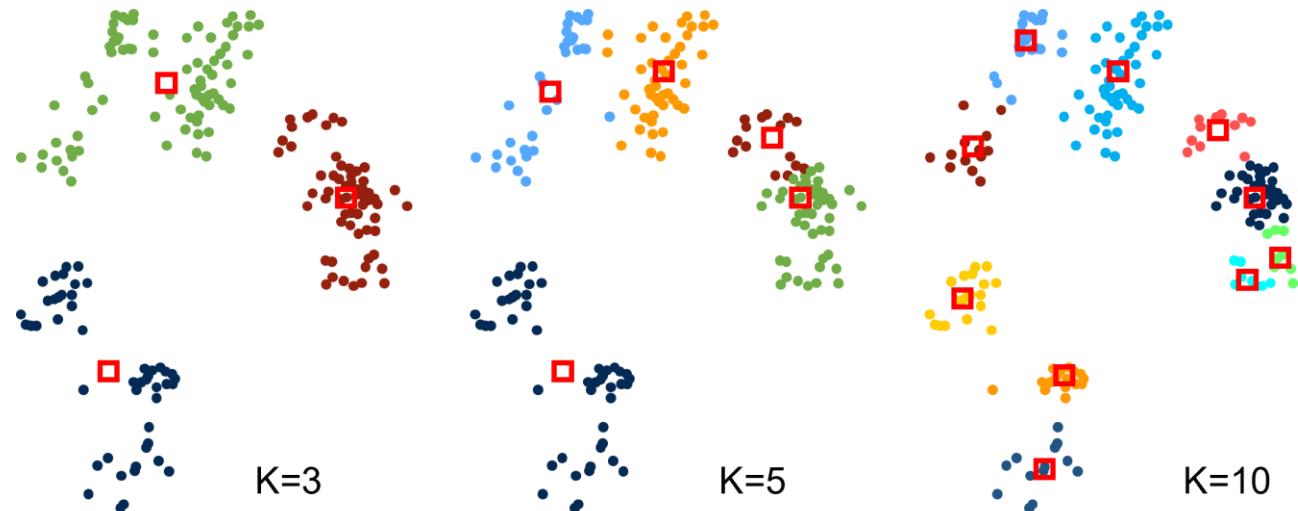




5. 최적의 클러스터 개수 찾기

■ 비용함수 이용: $C(z, \mu) = \sum_i \|x_i - \mu_{z_i}\|^2$

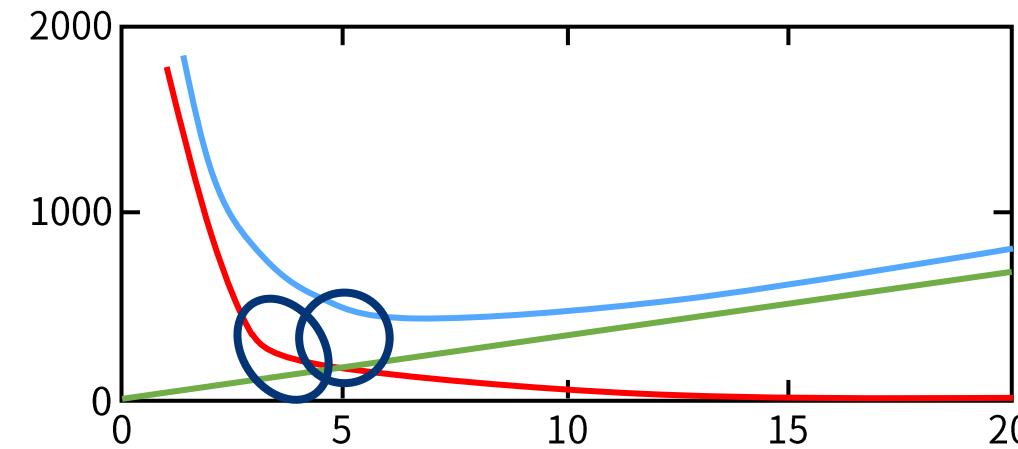
■ 일반적으로 K가 증가할 수록, C는 감소, but 비용 증가



5. 최적의 클러스터 개수 찾기

복잡도를 제어할 수 있는 penalty를 추가

- ✓ Total = Error + Complexity



결과에 크게 개선하지 않는 경우, 더 적은 클러스터의 개수를 선택

정리하기

▶ K-Means Clustering

- ▶ 특징 공간에서 위치(center)를 기반으로 클러스터를 정의

▶ 알고리즘

- ▶ 클러스터 센터 초기화
- ▶ 반복:
 1. 각 데이터 포인트에 가장 가까운 클러스터를 할당
 2. 오차 제곱을 최소화 하는 방향으로 클러스터 센터들을 업데이트

정리하기

▶ 속성

- ▶ Coordinate descent on MSE criterion
- ▶ Local optima에 빠지기 쉬움 → 초기화가 중요

▶ # of clusters, K의 선택

출처

[출처01] Data from Garber et al. PNAS (98), 2001

[출처02] <https://www.pyimagesearch.com/>

들어가기

| 학습내용 |

- Logistic regression에서의 가설 모델
- SVM에서의 hinge loss vs. logistic loss
- Large margin과 SVM decision boundary의 관계

들어가기

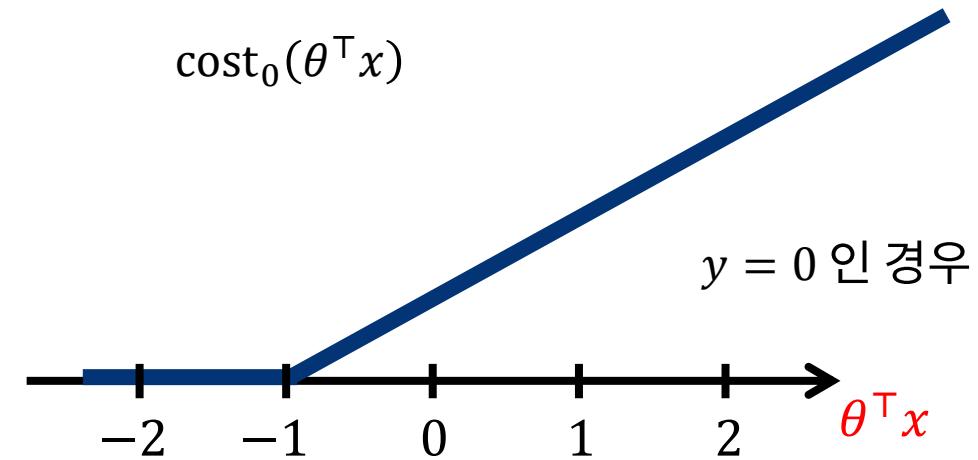
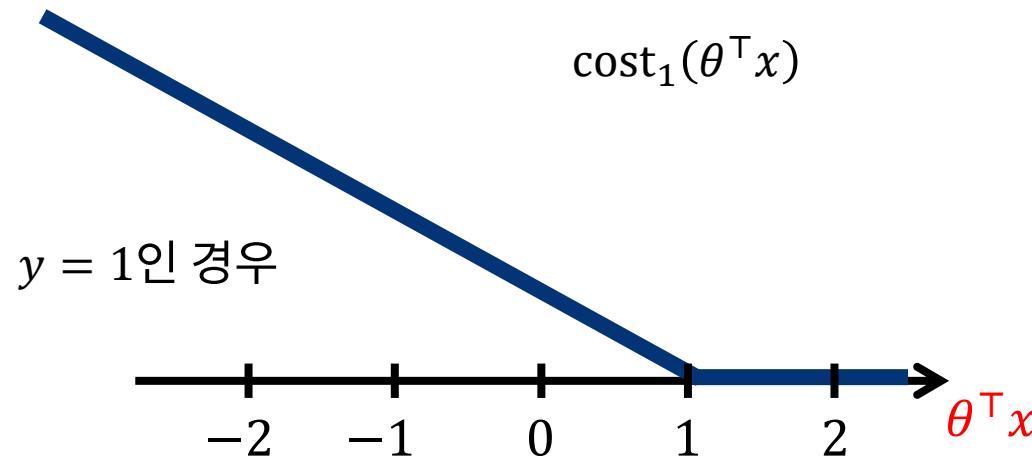
| 학습내용 |

- Kernel trick을 이용한 비선형 분류기로서의 SVM의 동작 원리
- SVM을 multi-class 분류 문제에 적용



0. SVM decision boundary(recap)

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



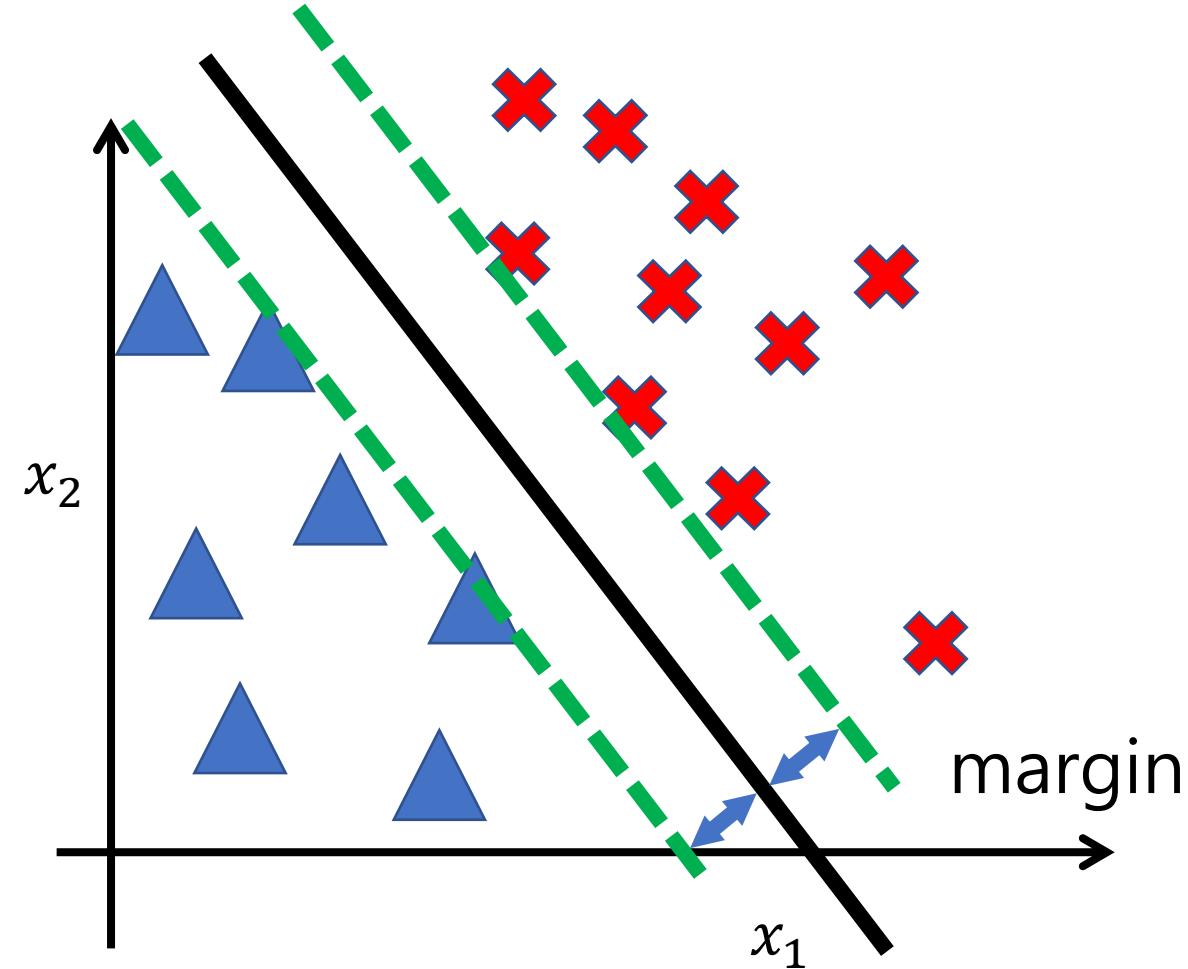
- “ $y = 1$ ” 일 때, $\theta^\top x \geq 1$ (not just ≥ 0)
- “ $y = 0$ ” 일 때, $\theta^\top x \leq -1$ (not just < 0)



0. SVM decision boundary(recap)

선형 분류 문제

출처 번호 수정

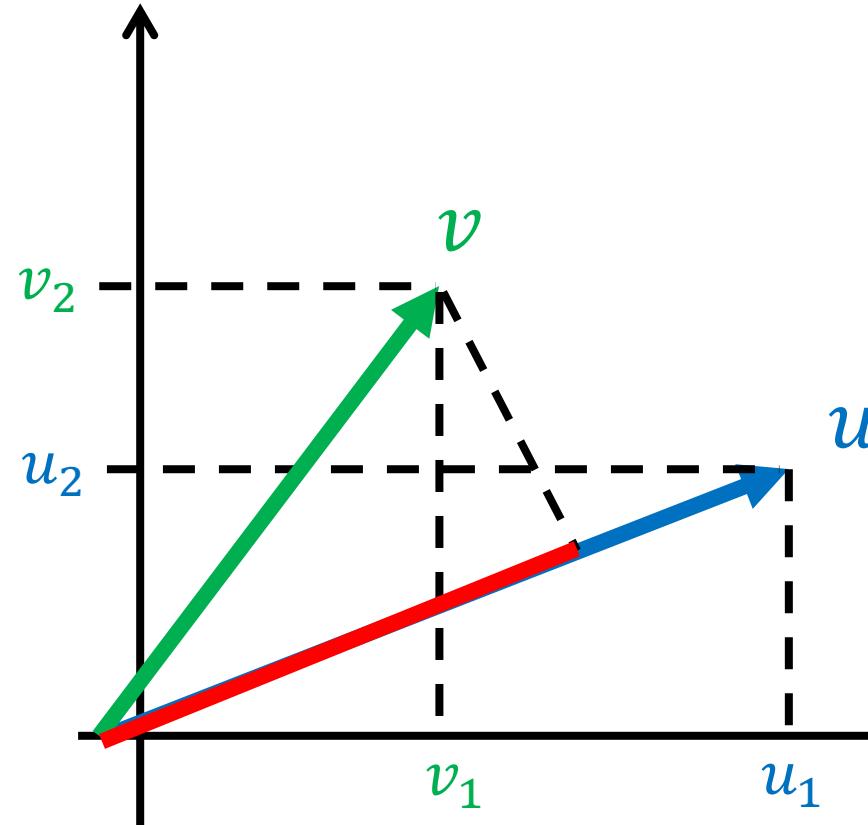




0. SVM decision boundary(recap)

출처 번호 수정

KU KONKUK UNIVERSITY



$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\begin{aligned}\|u\| &= \text{length of vector } u \\ &= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}\end{aligned}$$

p = length of projection of v onto u

$$\begin{aligned}u^\top v &= p \cdot \|u\| \\ &= u_1 v_1 + u_2 v_2\end{aligned}$$



0. SVM decision boundary(recap)

출처 번호 수정

KU KONKUK UNIVERSITY

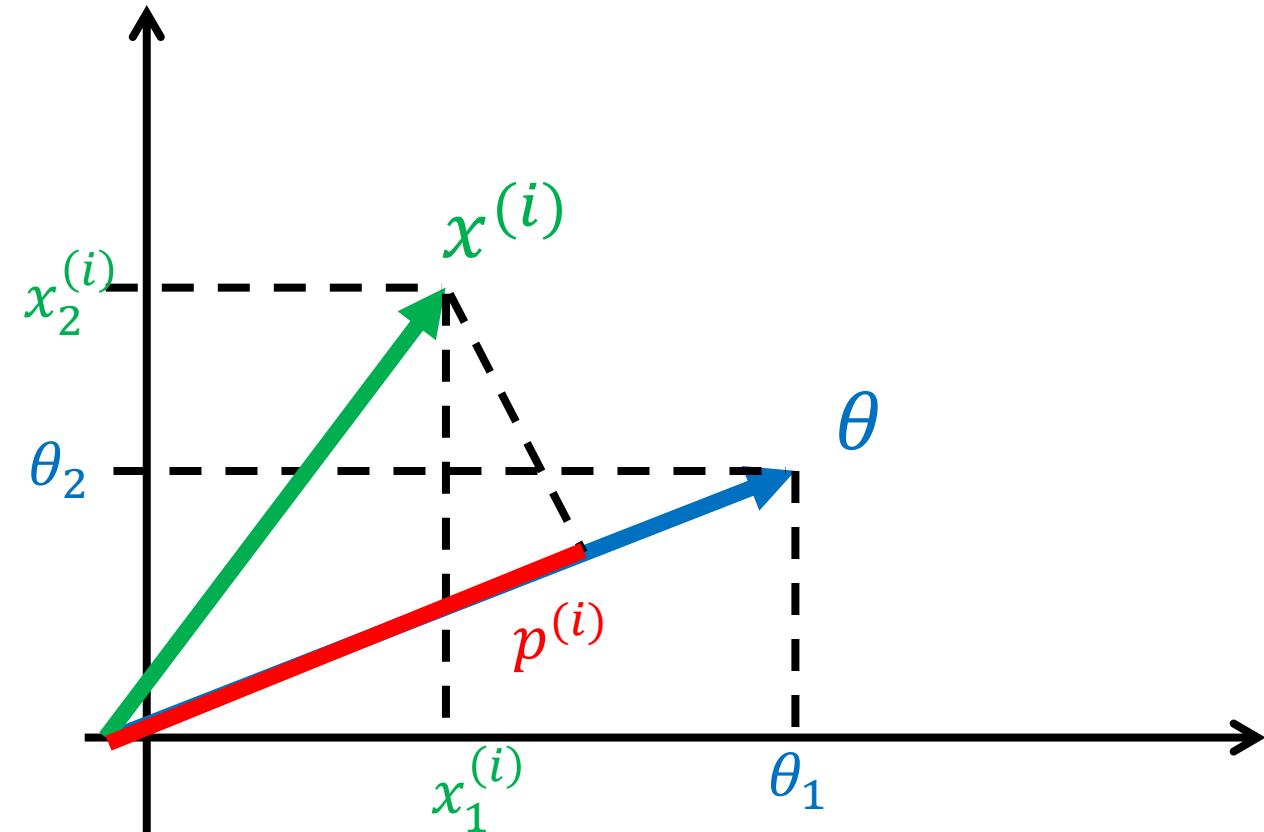
$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$$\text{s.t. } \begin{aligned} \theta^\top x^{(i)} &\geq 1 & \text{if } y^{(i)} = 1 \\ \theta^\top x^{(i)} &\leq -1 & \text{if } y^{(i)} = 0 \end{aligned}$$

Simplification: $\theta_0 = 0, n = 2$

What's $\theta^\top x^{(i)}$?

$$\theta^\top x^{(i)} = p^{(i)} \|\theta\|^2$$





0. SVM decision boundary(recap)

출처 번호 수정

KU KONKUK UNIVERSITY

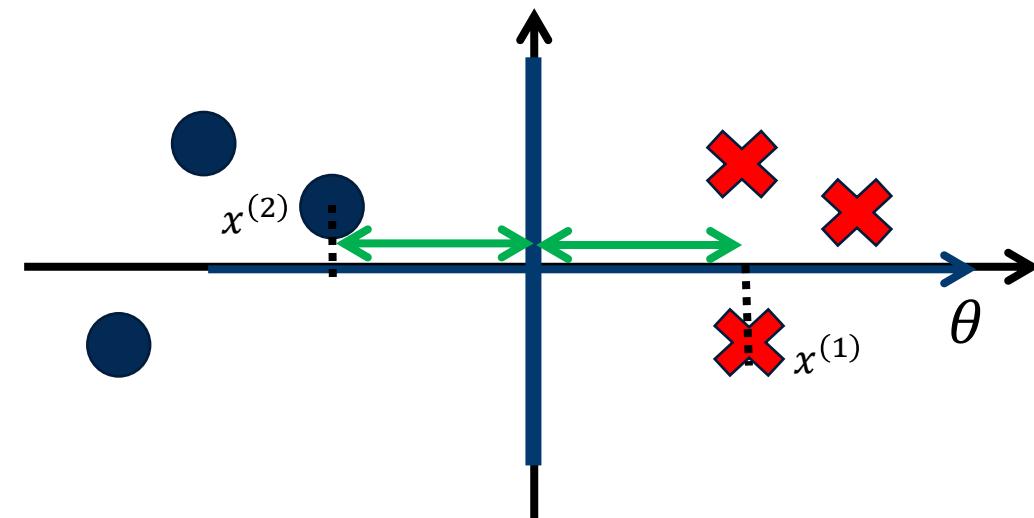
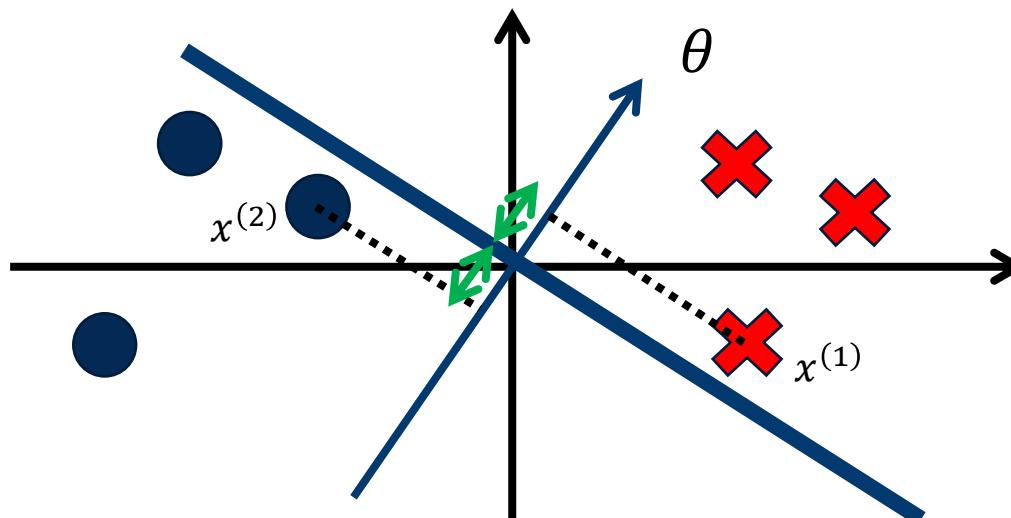
$$\min_{\theta} \frac{1}{2} \|\theta\|^2$$

$$\text{s. t. } p^{(i)} \|\theta\|^2 \geq 1 \quad \text{if } y^{(i)} = 1 \\ p^{(i)} \|\theta\|^2 \leq -1 \quad \text{if } y^{(i)} = 0$$

Simplification: $\theta_0 = 0, n = 2$

$p^{(1)}, p^{(2)}$ small $\rightarrow \|\theta\|^2$ large

$p^{(1)}, p^{(2)}$ large $\rightarrow \|\theta\|^2$ can be small

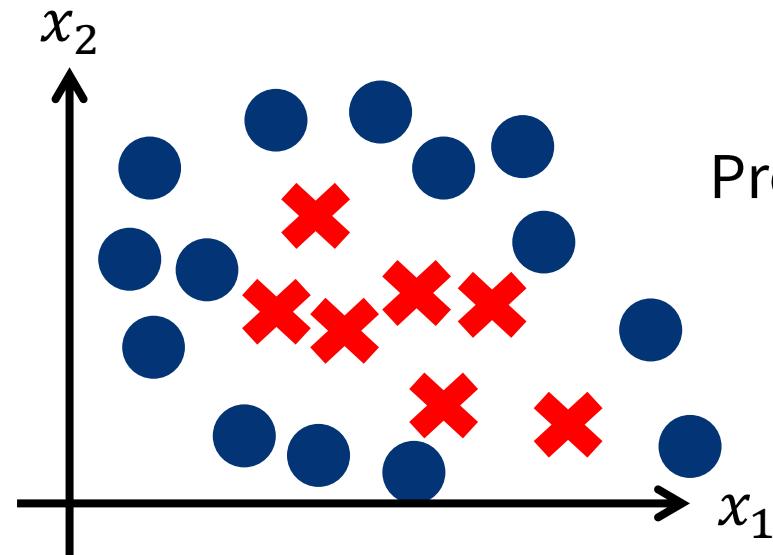




1. 비선형 분류 문제

출처 번호 수정

KU KONKUK UNIVERSITY



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots \geq 0$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, \dots$$

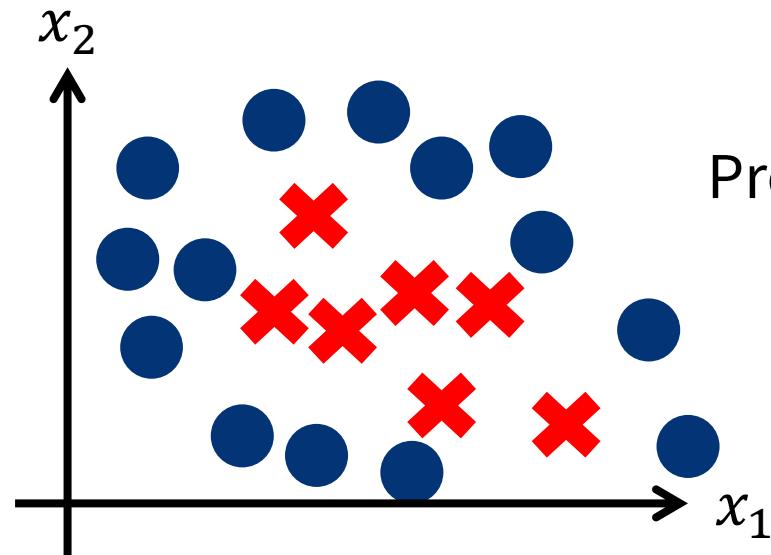
f_1, f_2, f_3, \dots 을 선택하기 위한 더 좋은 방법 ?



1. 비선형 분류 문제

출처 번호 수정

KU KONKUK UNIVERSITY



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots \geq 0$$

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, \dots$$

f_1, f_2, f_3, \dots 을 선택하기 위한 더 좋은 방법 ?



2. 커널 (Kernel)

출처 번호 수정

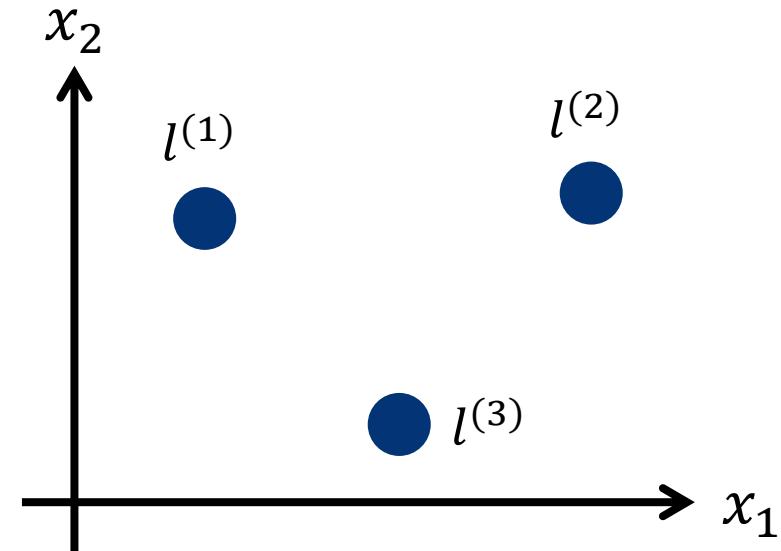
KU KONKUK UNIVERSITY

- 입력 x 에 대해, landmarks $l^{(1)}, l^{(2)}, l^{(3)}$ 와의 인접성 (proximity)을 기반으로 새로운 특징들을 계산

- $f_1 = \text{similarity}(x, l^{(1)})$
- $f_2 = \text{similarity}(x, l^{(2)})$
- $f_3 = \text{similarity}(x, l^{(3)})$

- Gaussian kernel

- $\text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$

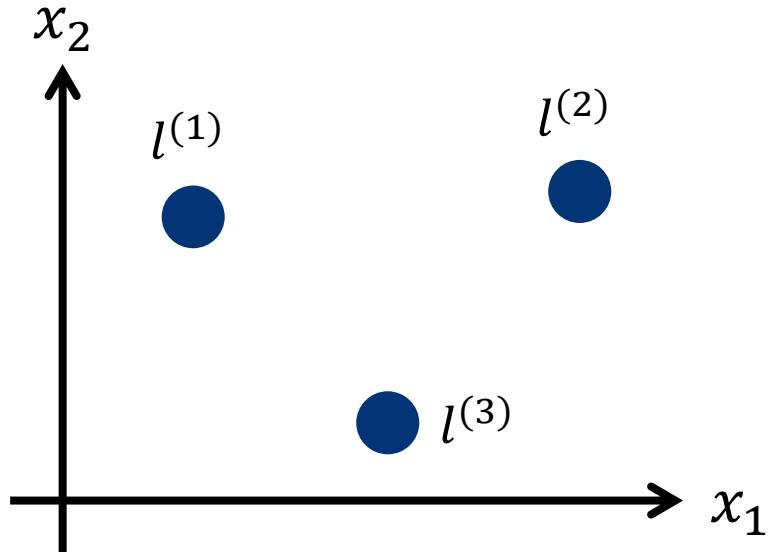




2. 커널 (Kernel)

출처 번호 수정

KU KONKUK UNIVERSITY



- $f_1 = \text{similarity}(x, l^{(1)})$
- $f_2 = \text{similarity}(x, l^{(2)})$
- $f_3 = \text{similarity}(x, l^{(3)})$

✓ 입력 x 에 대해, landmarks $l^{(1)}, l^{(2)}, l^{(3)}$ 와의 인접성 (proximity)을 기반으로 새로운 특징들을 계산

✓ Gaussian kernel

- $\text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$



3. 커널 (Kernel) 과 유사도 특성

출처 번호 수정

KU KONKUK UNIVERSITY

Kernel과 유사도 (similarity)

- 비선형 특성을 다루는 또 다른 기법은 각 샘플이 특정 랜드마크(landmark)와 얼마나 닮았는지 측정하는 유사도 함수(similarity function)로 계산한 특성을 추가하는 것

$$f_1 = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



3. 커널 (Kernel) 과 유사도 특성

출처 번호 수정

KU KONKUK UNIVERSITY

■ 예제

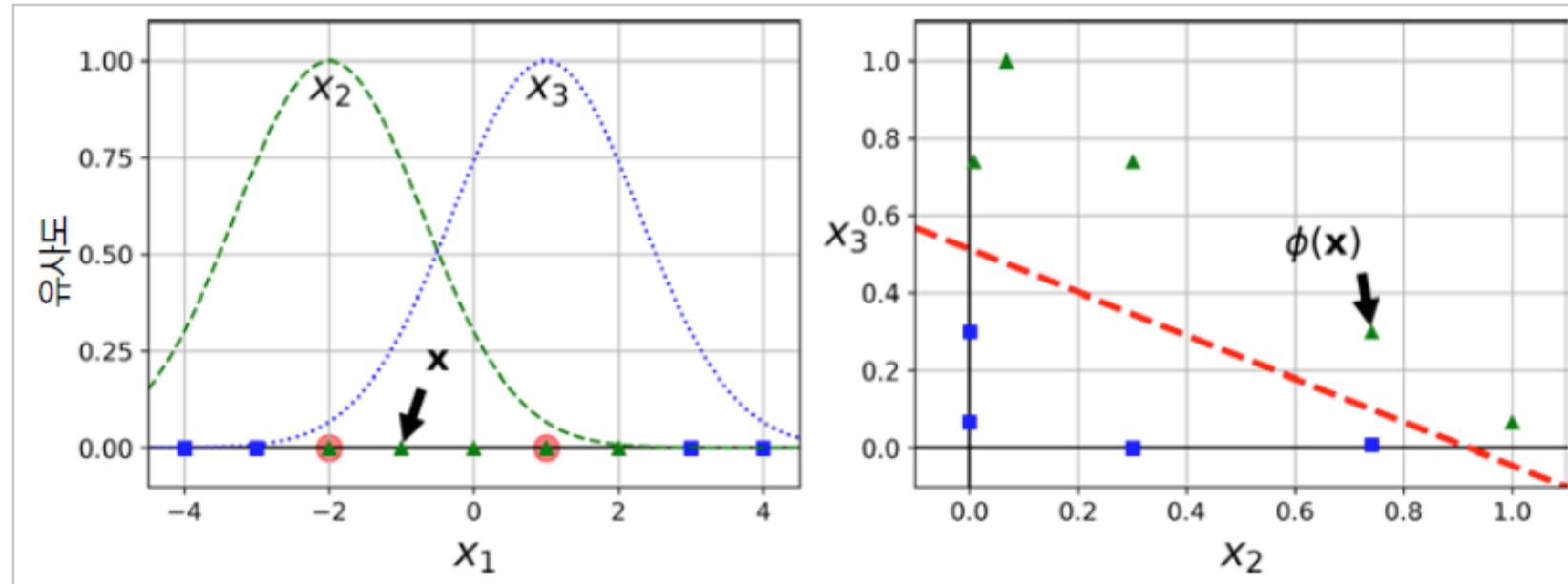
$$l^{(1)} = [3 \ 5]^T \quad f_1 = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$



3. 커널 (Kernel) 과 유사도 특성

출처 번호 수정

KU KONKUK UNIVERSITY



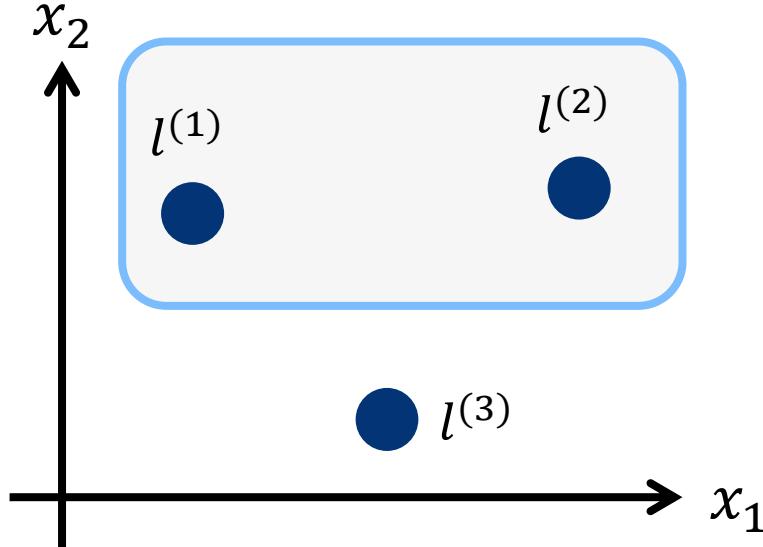
가우시안 RBF를 사용한 유사도 특성



3. 커널 (Kernel) 과 유사도 특성

출처 번호 수정

KU KONKUK UNIVERSITY



Predict $y = 1$ if

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

Ex: $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$

- $f_1 = \text{similarity}(x, l^{(1)})$
- $f_2 = \text{similarity}(x, l^{(2)})$
- $f_3 = \text{similarity}(x, l^{(3)})$



3. 커널 (Kernel) 과 유사도 특성

출처 번호 수정

KU KONKUK UNIVERSITY

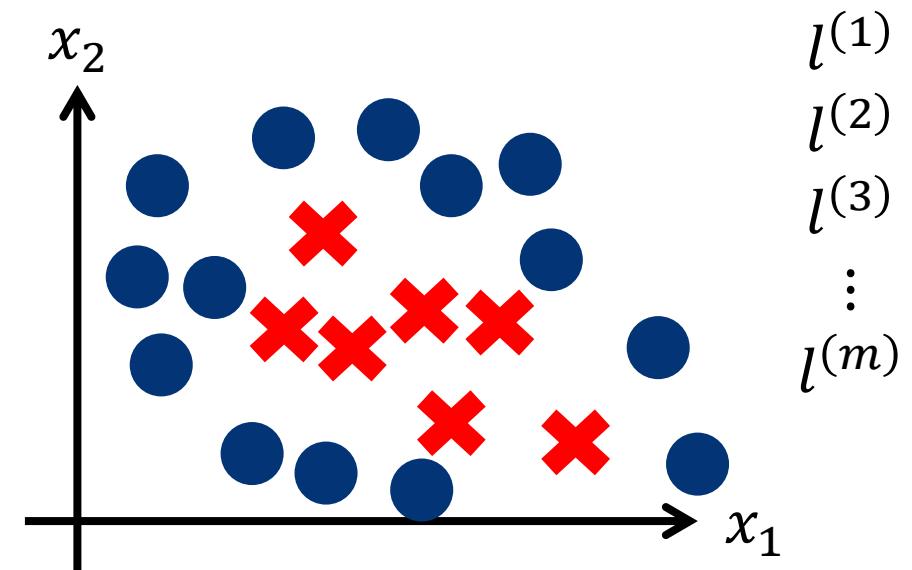
■ Landmarks의 선택

- 주어진 x 에 대해

$$f_1 = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right)$$

“ $y = 1$ ” if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \dots$?





4. 비선형 SVM 분류

출처 번호 수정

KU KONKUK UNIVERSITY

- ✓ 학습 데이터: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$
- ✓ Landmarks 선택: $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, l^{(3)} = x^{(3)}, \dots, l^{(m)} = x^{(m)}$
- ✓ 입력 x 에 대해:
 - $f_1 = \text{similarity}(x, l^{(1)})$
 - $f_2 = \text{similarity}(x, l^{(2)})$
 - ...
- ✓ 학습 데이터 샘플 $(x^{(i)}, y^{(i)})$ 에서:
 - $x^{(i)} \rightarrow f^{(i)}$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$$



4. 비선형 SVM 분류

출처 번호 수정

KU KONKUK UNIVERSITY

- 분류 모델: 입력 x 에 대해, 새로운 특징 $f \in \mathbb{R}^{m+1}$ 를 계산

- Predict $y = 1 \text{ if } \theta^\top f \geq 0$

- Training (original)

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

- Training (with kernel)

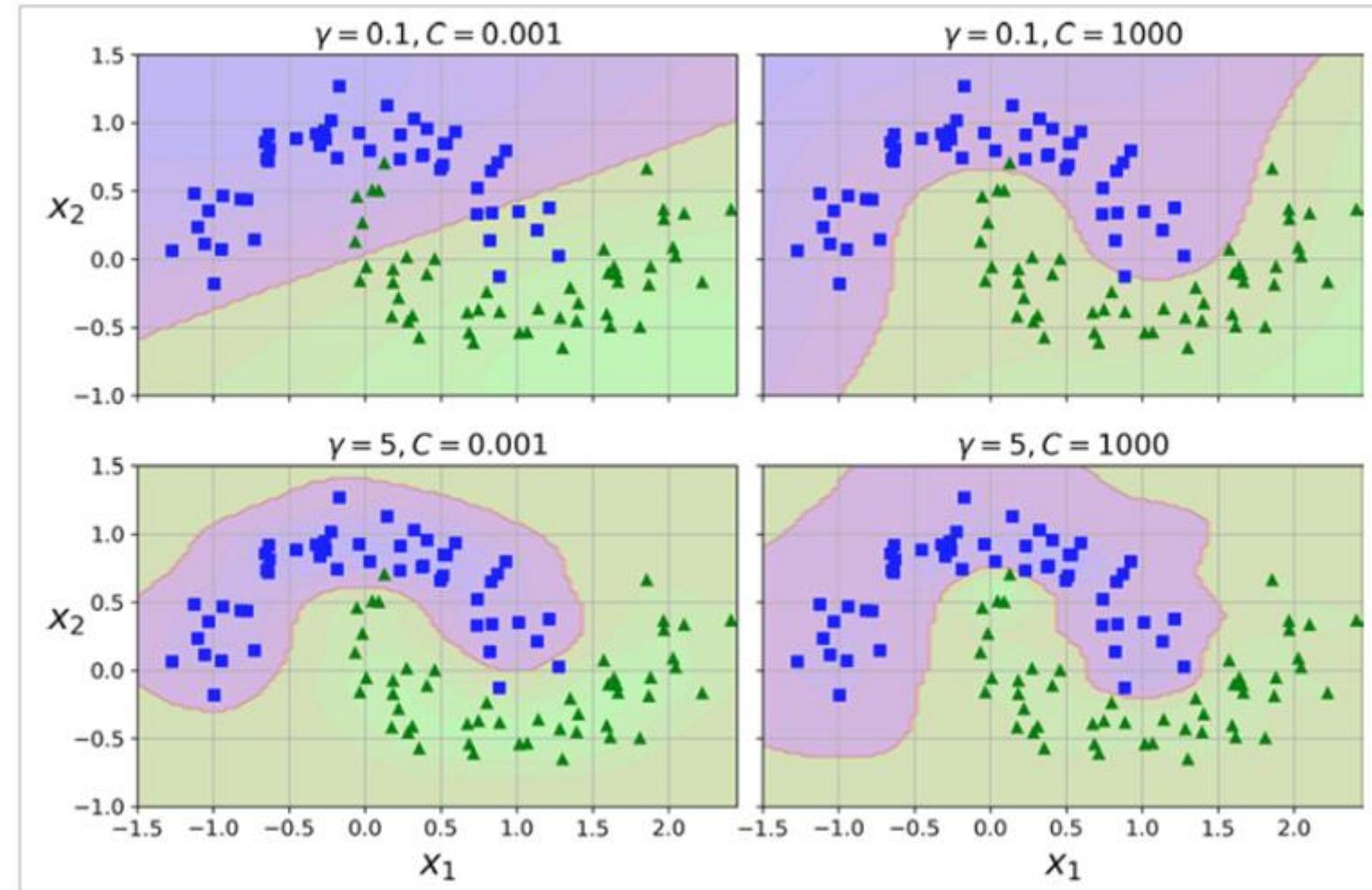
$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^m \theta_j^2$$



4. 비선형 SVM 분류

삭제 페이지

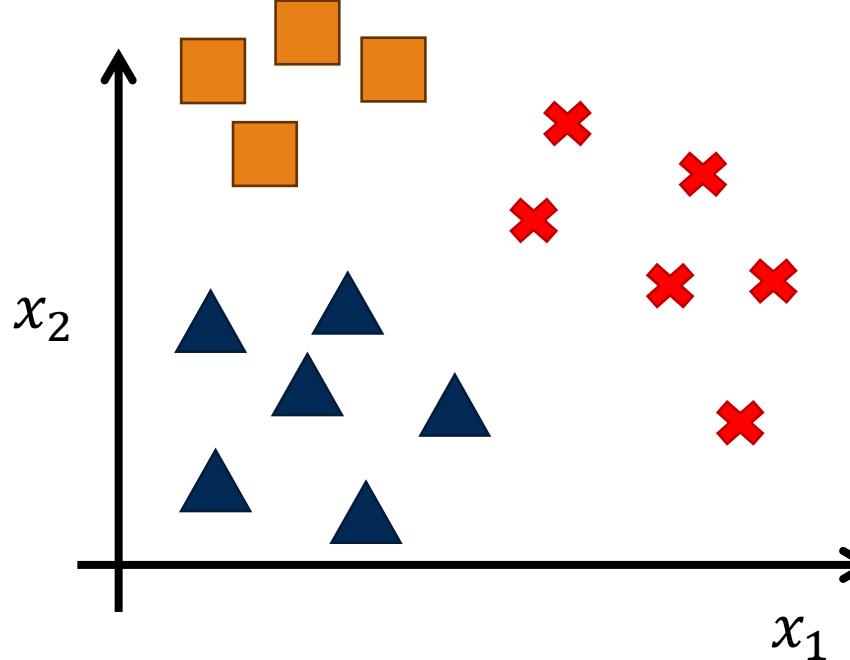
KU KONKUK UNIVERSITY



RBF 커널을 사용한 SVM 분류기



5. Multi-class 분류



출처 번호 수정

- ✓ 많은 SVM 패키지들에 multi-class classification functionality 이 탑재됨
- ✓ One-vs.-all 방법
 - 각 i 번째 클래스를 다른 클래스로부터 식별할 수 있는 K 개의 SVMs을 학습, $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(K)}$ 을 모델
 - 가장 큰 $\theta^{(i)^\top} x$ 의 값을 가진 클래스를 선택



6. Logistic Regression vs. SVMs

출처 번호 수정

KU KONKUK UNIVERSITY

■ $n = \#.$ features ($x \in \mathbb{R}^{n+1}$), $m = \#.$ 학습데이터

1. **$n \gg m$:** ($n = 10,000, m = 10 - 1000$)
→ logistic regression 또는 선형 SVM (“linear kernel”)을 사용
2. **Small n , intermediate m :** ($n = 1 - 1000, m = 10 - 10,000$)
→ 비선형 SVM (SVM with Gaussian kernel)
3. **Small n , large m :** ($n = 1 - 1000, m = 50,000+$)
→ 새로운 특징 생성 및 추가 후, logistic regression or linear SVM을 사용

Neural network이 모든 상황에서 잘 동작하나, 학습에 시간이 많이 소요

정리하기

- ▶ SVM에서 Kernel trick을 이용하여 비선형 분류 문제에 적용
- ▶ Kernel에서의 landmarks 생성과 이로부터의 유사도를 이용하여 새로운 특징을 생성
- ▶ SVM은 Linear separable / non-linear separable / multi-class 분류 등 매우 강력하고 다양한 문제에 사용할 수 있는 다목적 머신러닝 모델임

출처

[출처01] Standford University CS229: Machine Learning (by Andrew Ng.)

[출처02] 오렐리앙 제롬 지음, 박해선 옮김, 『핸즈온 머신러닝(2판)』, 한빛미디어, 2020.

들어가기

| 학습목표 |

- 서포트 벡터 머신 (SVM)의 원리를 이해한다.
- SVM과 logistic regression과의 차이를 설명할 수 있다.
- Large margin classification을 이해한다.

들어가기

| 학습목표 |

- Kernel trick을 설명할 수 있다.

- SVM을 multi-class 분류 문제에 적용

들어가기

| 학습내용 |

- Logistic regression에서의 가설 모델
- SVM에서의 hinge loss vs. logistic loss
- Large margin과 SVM decision boundary의 관계

들어가기

| 학습내용 |

- Kernel trick을 이용한 비선형 분류기로서의 SVM의 동작 원리
- SVM의 하이퍼파라미터 최적화를 통한 성능 향상

들어가기

| 학습목차 |

1 | Hypothesis model in SVM

2 | Cost function

3 | Large margin classification

4 | Kernels

5 | Using an SVM



0. Logistic Regression

▣ 결정 함수(decision boundary)

- ✓ $h_{\theta}(x) = g(\theta^T x)$

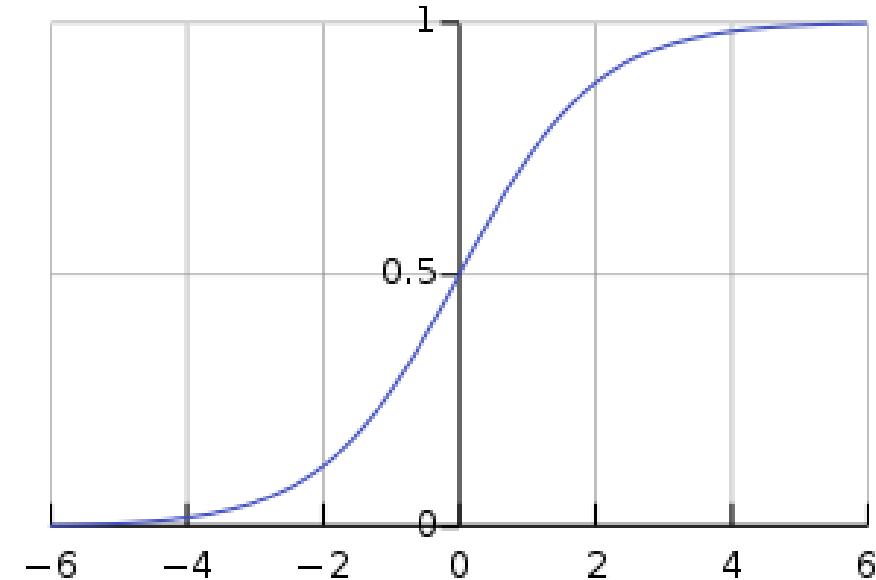
- ✓ $g(z) = \frac{1}{1+e^{-z}}$

- $h_{\theta}(x) \geq 0.5$ 인 경우, “y = 1”로 예측

$$z = \theta^T x \geq 0$$

- $h_{\theta}(x) < 0.5$ 인 경우, “y = 0”로 예측

$$z = \theta^T x < 0$$





0. Logistic Regression

출처 번호 수정

KU KONKUK UNIVERSITY

▣ 결정 함수(decision boundary)

- ✓ $h_{\theta}(x) = g(\theta^T x)$

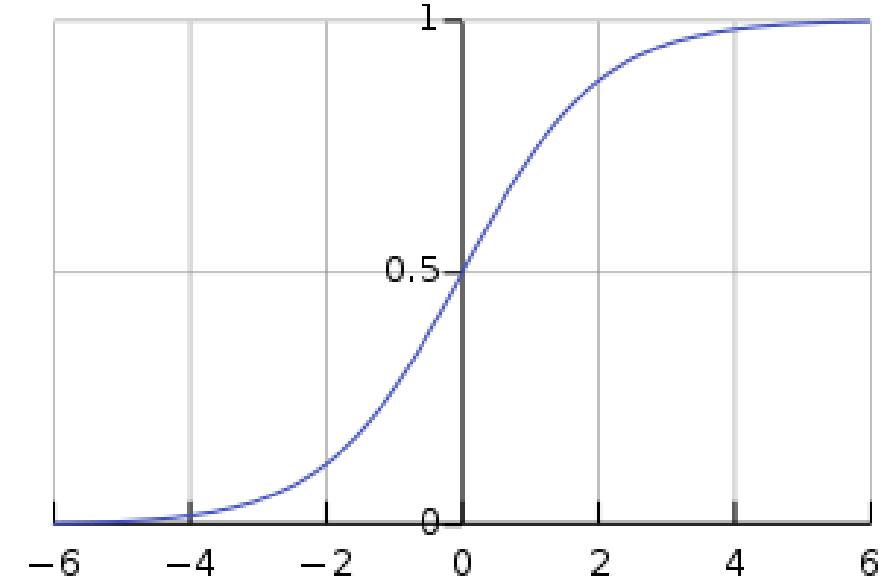
- ✓ $g(z) = \frac{1}{1+e^{-z}}$

- “ $y = 1$ ” 인 경우, $h_{\theta}(x) \approx 1$ 을 선호

$$z = \theta^T x \gg 0$$

- “ $y = 0$ ” 인 경우, $h_{\theta}(x) \approx 0$ 을 선호

$$z = \theta^T x \ll 0$$



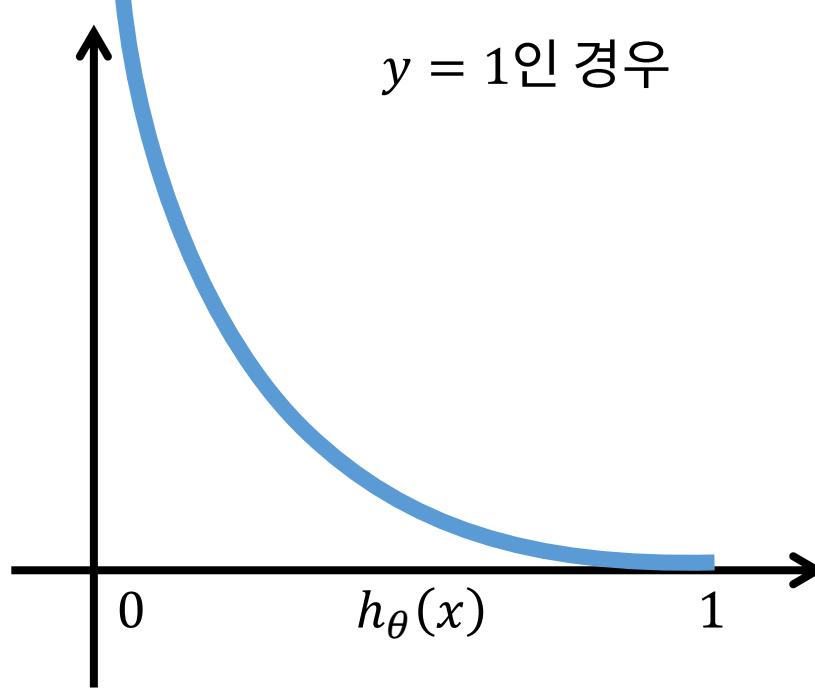


1. Logistic Regression에서의 비용 함수

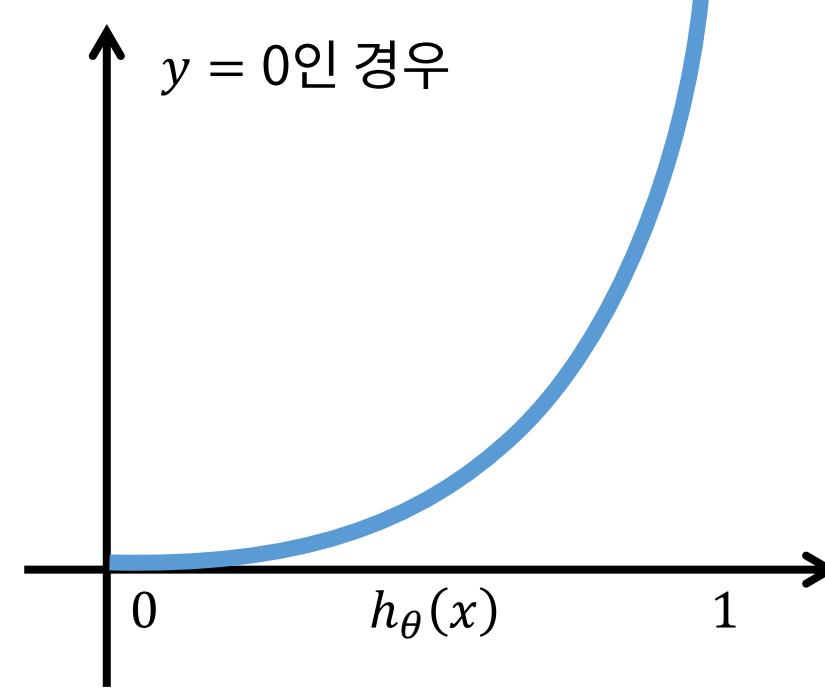
비용 함수(Cost function)

출처 번호 수정

- Cost($h_{\theta}(x), y$) =
$$\begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$



$y = 1$ 인 경우



$y = 0$ 인 경우

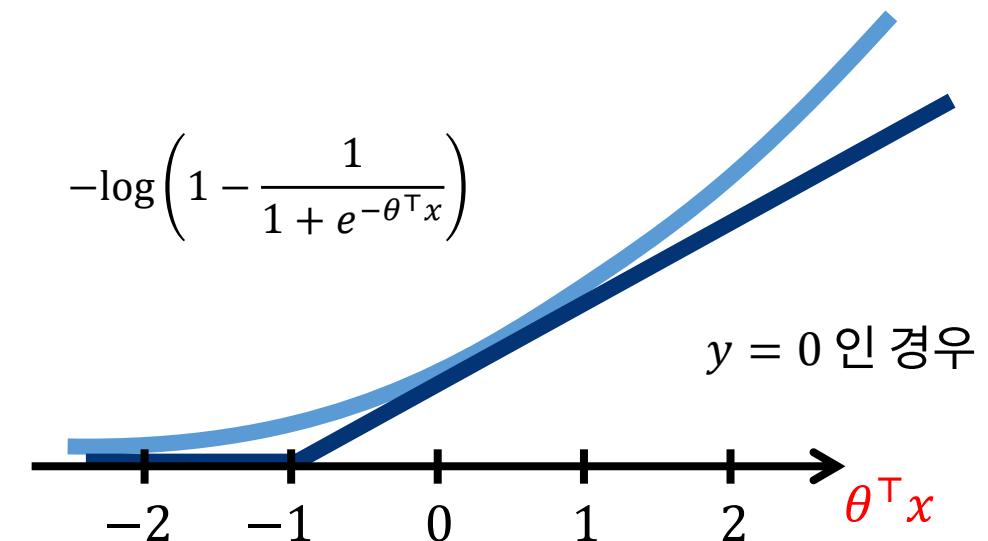
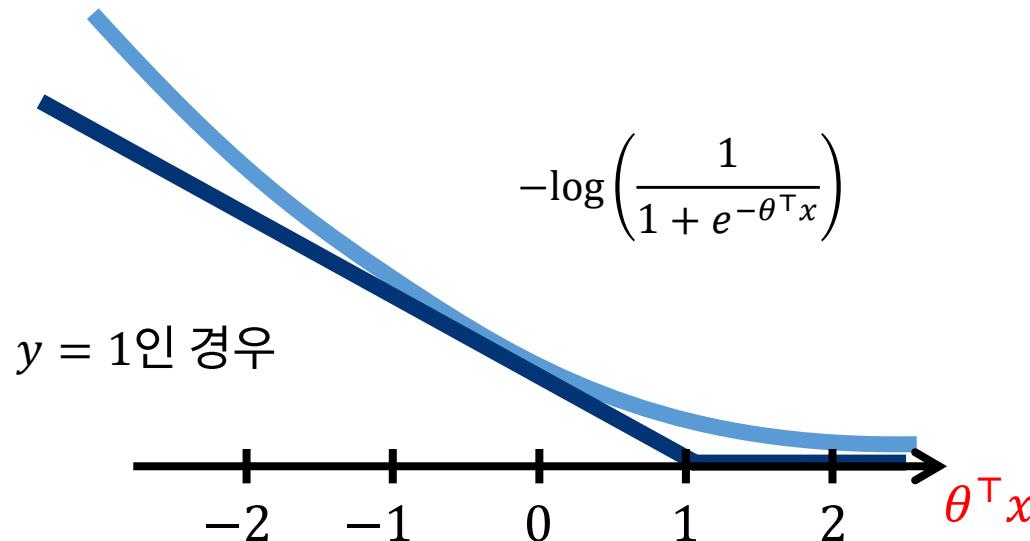


1. Logistic Regression에서의 비용 함수

비용 함수(Cost function)

출처 번호 수정

- Cost($h_{\theta}(x), y$) = $-y \log(h_{\theta}(x)) - (1 - y) \log(1 - h_{\theta}(x))$
 $= y \left(-\log \left(\frac{1}{1 + e^{-\theta^T x}} \right) \right) + (1 - y) \left(-\log \left(1 - \frac{1}{1 + e^{-\theta^T x}} \right) \right)$

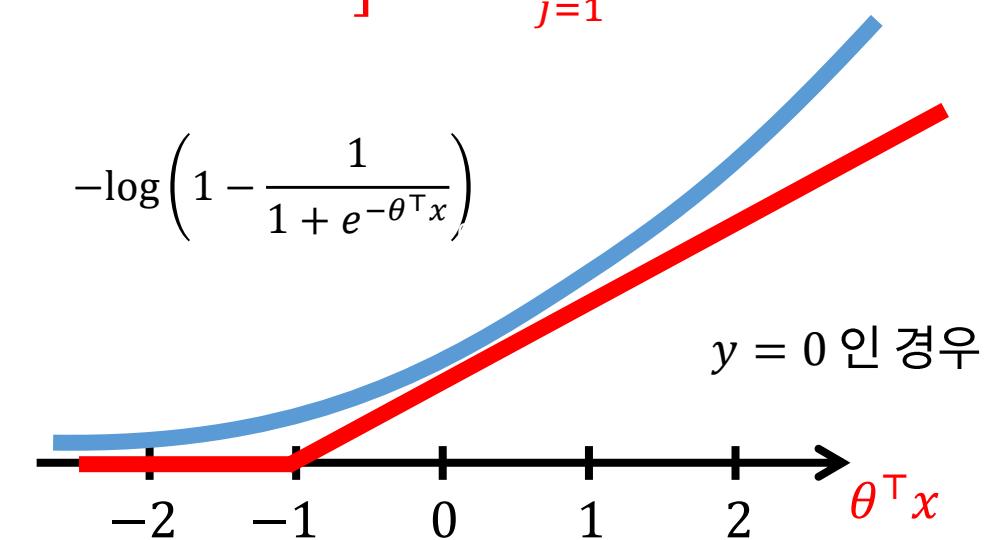
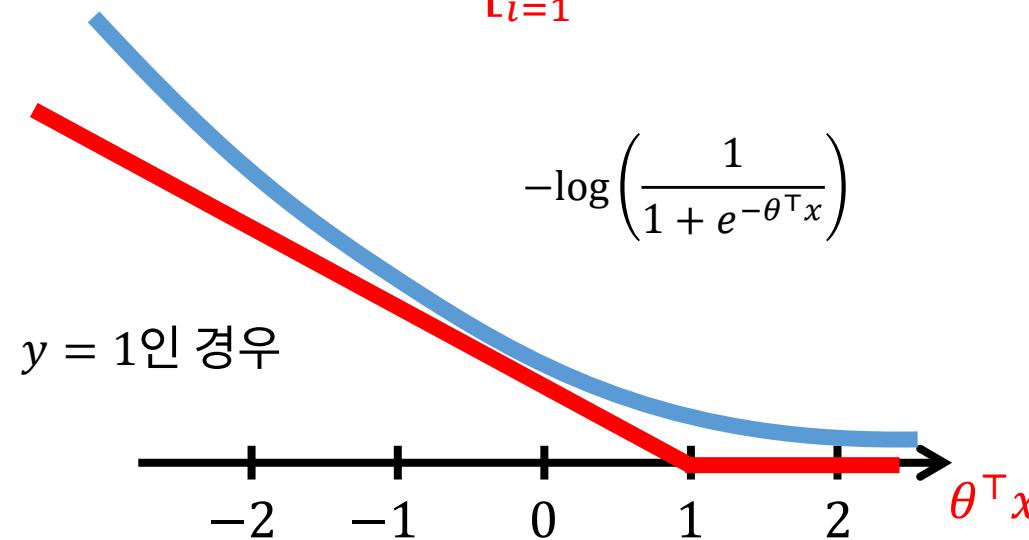


■ Logistic Regression (logistic loss)

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \left(-\log(h_{\theta}(x^{(i)})) \right) + (1 - y^{(i)}) \left(-\log(1 - h_{\theta}(x^{(i)})) \right) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

■ Support vector machine (hinge loss)

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$





2. SVM에서의 목적 함수

▣ 목표: 비용 함수(Cost function) 의 최적화

출처 번호 수정

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

1. Multiply m



2. Multiply $C = \frac{1}{\lambda}$

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



2. SVM에서의 목적 함수

▣ 목적 함수

출처 번호 수정

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

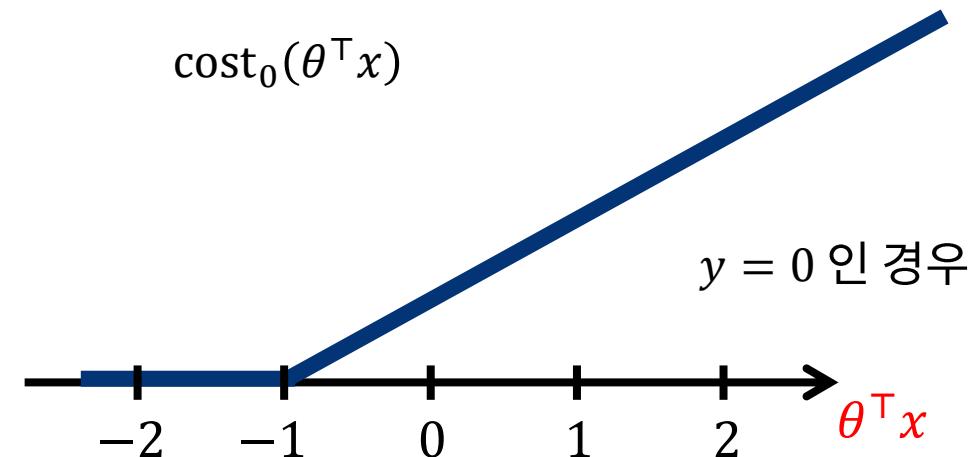
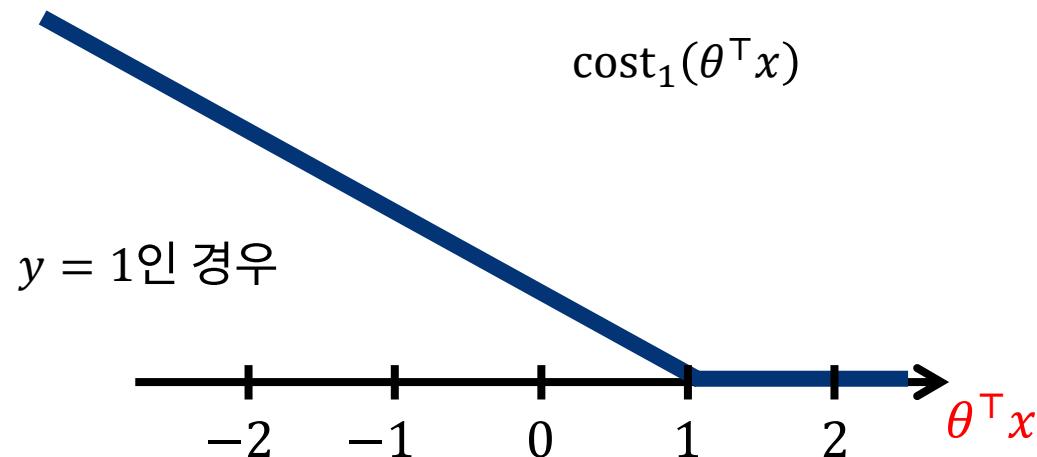
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^\top x \geq 0 \\ 0 & \text{if } \theta^\top x < 0 \end{cases}$$



3. 결정 경계 (Decision boundary)

출처 번호 수정

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



- “ $y = 1$ ” 일 때, $\theta^\top x \geq 1$ (not just ≥ 0)
- “ $y = 0$ ” 일 때, $\theta^\top x \leq -1$ (not just < 0)



3. 결정 경계 (Decision boundary)

출처 번호 수정

$$\min_{\theta} C \left[\sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^\top x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^\top x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

- 매우 큰 C 로 설정한 경우

- Whenever $y^{(i)} = 1$:

$$\theta^\top x^{(i)} \geq 1$$

- Whenever $y^{(i)} = 0$:

$$\theta^\top x^{(i)} \leq -1$$

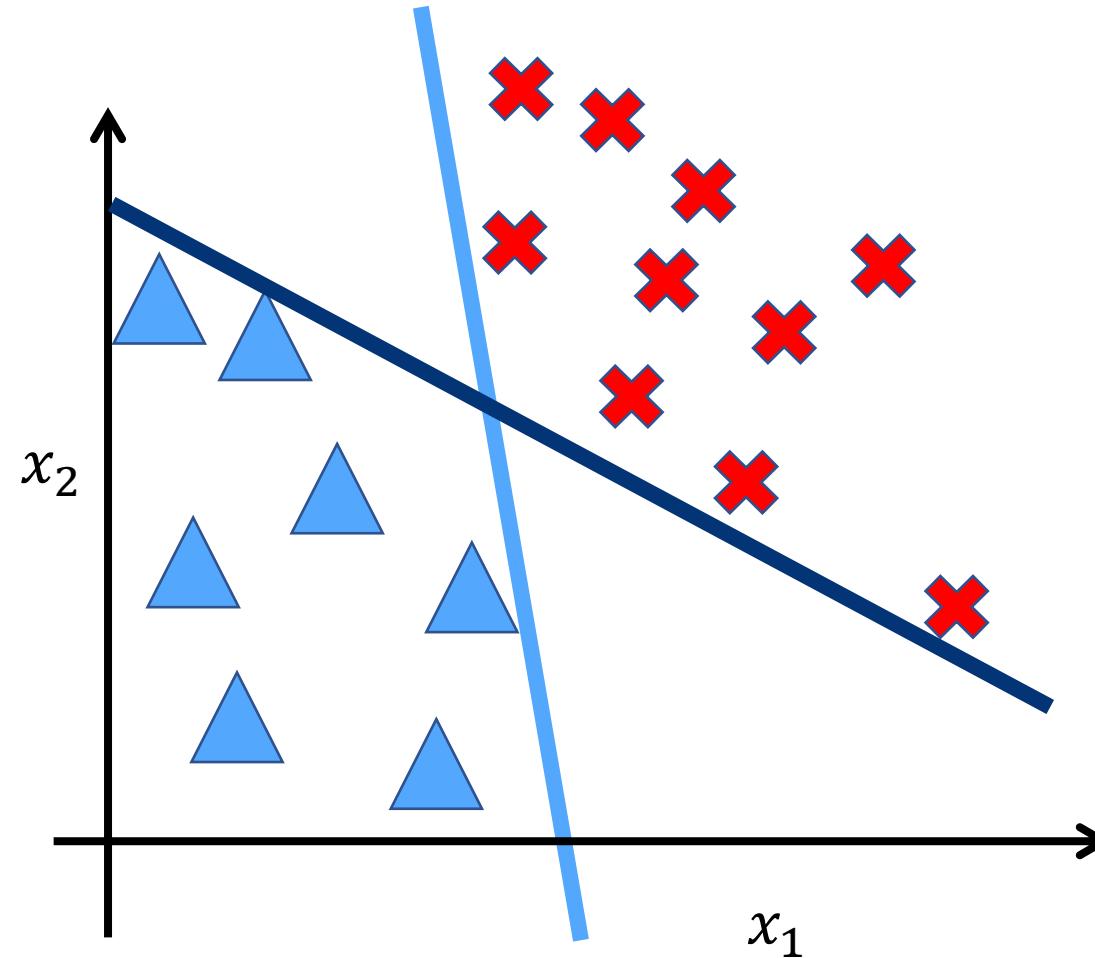
$$\begin{aligned} & \min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 \\ \text{s.t. } & \theta^\top x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1 \\ & \theta^\top x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0 \end{aligned}$$



3. 결정 경계 (Decision boundary)

선형 분류 문제

출처 번호 수정

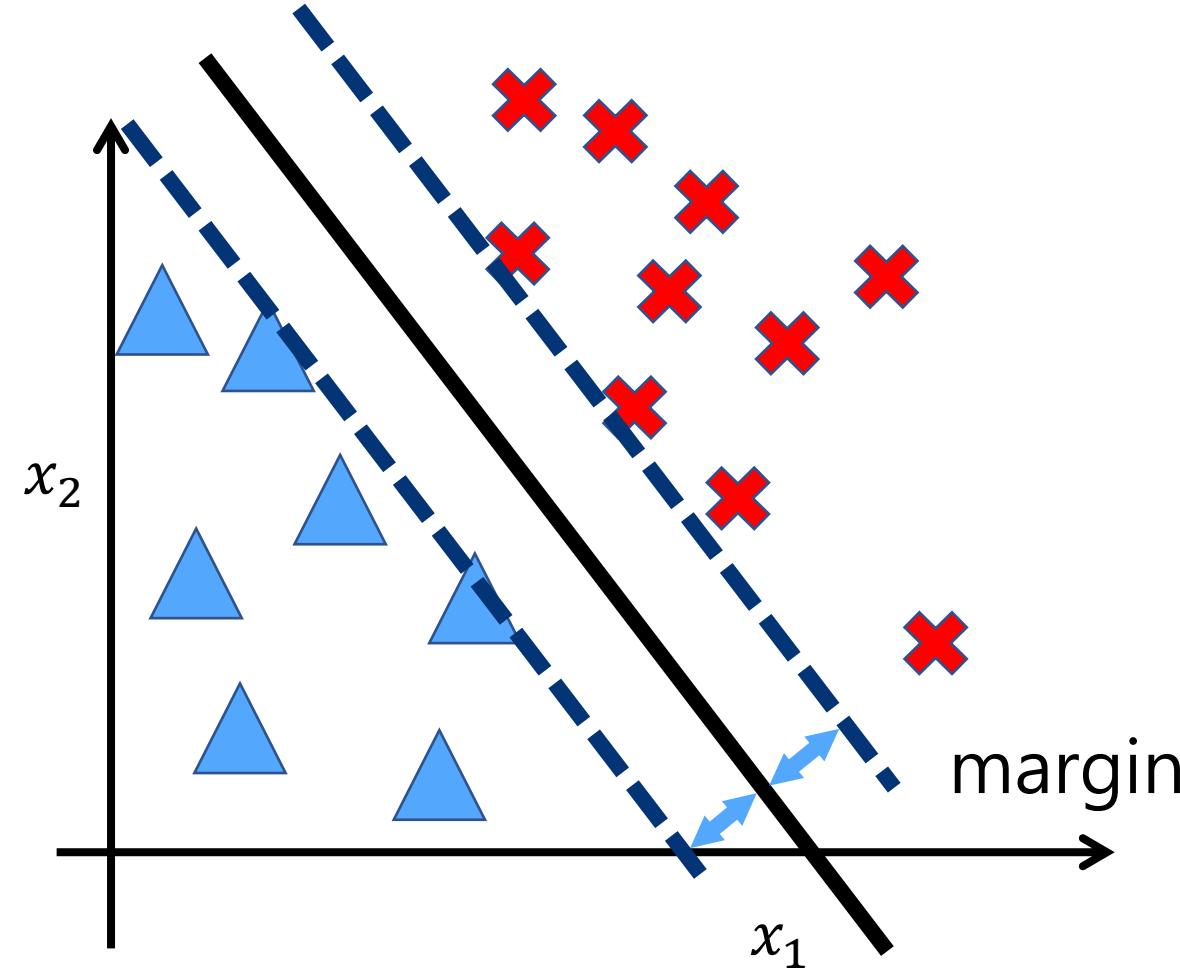




3. 결정 경계 (Decision boundary)

선형 분류 문제

출처 번호 수정

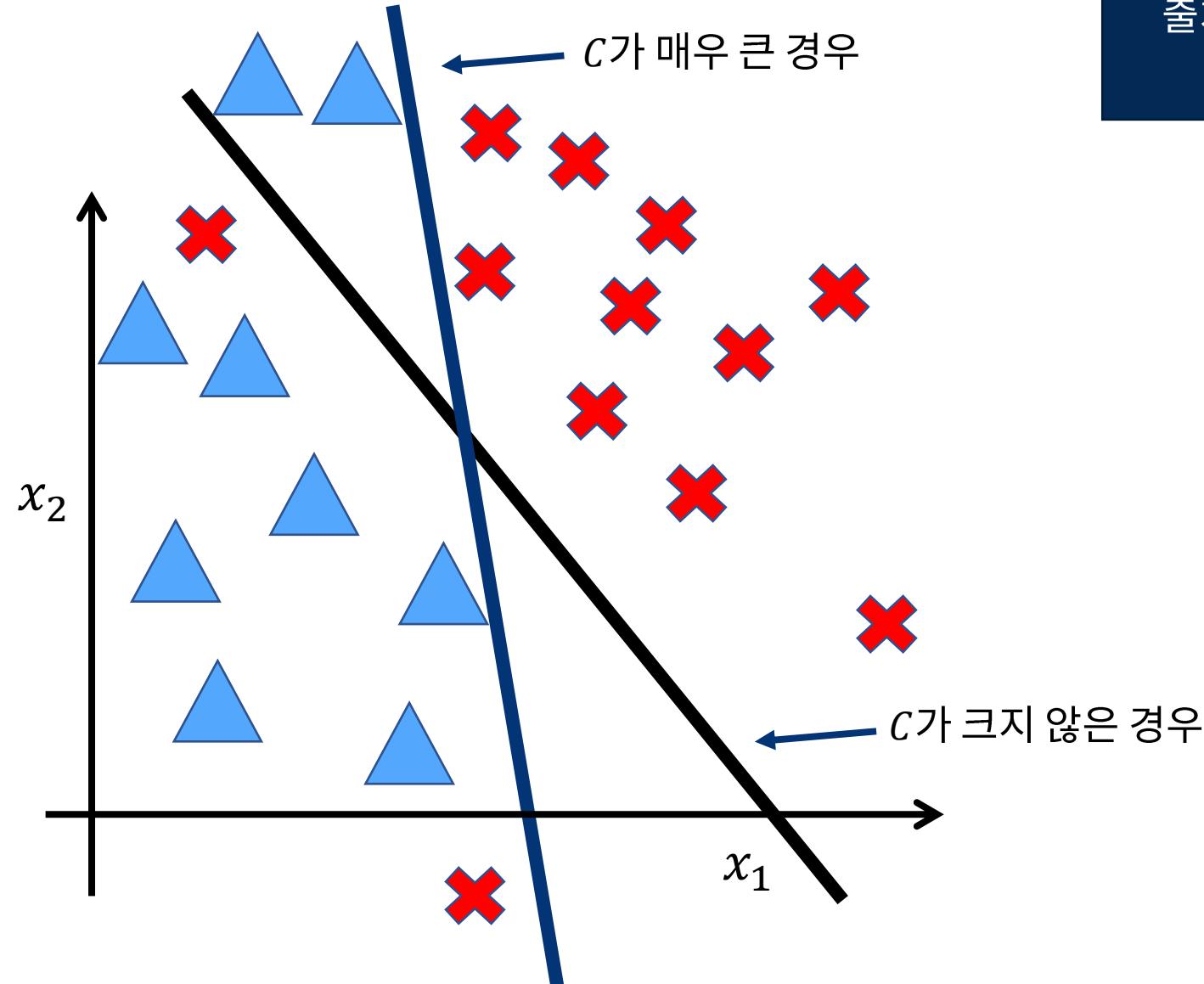




3. 결정 경계 (Decision boundary)

선형 분류 문제

출처 번호 수정



정리하기

- ▶ **서포트 벡터 머신(SVM)**
: 매우 강력하고 다양한 문제에 사용할 수 있는 다목적 머신러닝 모델이다

- ▶ **SVM에서의 hinge loss 최적화**

정리하기

- ▶ Large margin classification을 수행
- ▶ Support vector들에 의해 Decision boundary가 정의

들어가기

| 학습목표 |

- 이론으로 배운 신경망 모델을 실습을 통해 구현하는 방법을 학습하고, 현실 데이터를 사용하여 다층 퍼셉트론 모델을 만들어 본다.

들어가기

| 학습내용 |

- 파이토치 (pytorch) 설치

- 데이터셋

MNIST 데이터셋을 불러오고, 트레이닝 데이터와 테스트 데이터로 분할하는 법을 학습한다.

- 다층 퍼셉트론 모델 정의

다층 퍼셉트론 모델, 크로스 엔트로피 오차, 그리고 경사 하강법 함수를 정의한다.

- 다층 퍼셉트론 모델 학습

경사 하강법을 통해 다층 퍼셉트론 모델을 학습하고 테스트 한다.

들어가기

| 학습목차 |

1 | 파이토치 (pytorch) 설치

2 | 데이터셋

3 | 다층 퍼셉트론 모델 정의

4 | 다층 퍼셉트론 모델 학습



3. 다층 퍼셉트론 모델 정의

■ 다층 퍼셉트론 모델 정의

1

모델 이름을 Net으로 정의

4 | 5 | 6

3개 linear layer 정의, 입력 차원 784,
은닉 노드 차원 256, 64, 출력 차원 10

9

이미지를 784차원 벡터로 만들기

10 | 11 | 12

이미지 벡터를 세 개 linear layer 순차적으로 통과,
linear layer 사이에 relu 비선형 함수 추가.

13

10차원 벡터 출력

```
1 class Net(nn.Module):
2     def __init__(self):
3         super(Net, self).__init__()
4         self.fc1 = nn.Linear(784, 256)
5         self.fc2 = nn.Linear(256, 64)
6         self.fc3 = nn.Linear(64, 10)
7
8     def forward(self, x):
9         x = x.view(-1, 784)      # [batch, 784]
10    x = F.relu(self.fc1(x))   # [batch, 256]
11    x = F.relu(self.fc2(x))   # [batch, 64]
12    x = self.fc3(x)          # [batch, 10]
13    return x
```



3. 다층 퍼셉트론 모델 정의

▣ 경사 하강법, 오차 함수 정의

1

학습 할 장치 설정 (cpu / gpu)

2

정의한 Net()모델을 설정한 장치로 이동

4 15

학습율 0.01로 경사 하강법 함수 정의

6

크로스 엔트로피로 오차 함수 정의

```
1 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
2 model = Net().to(device)
3
4 lr = 0.01
5 optimizer = optim.SGD(model.parameters(), lr=lr)
6 loss_func = nn.CrossEntropyLoss()
```



4. 다층 퍼셉트론 모델 학습

■ 모델 트레이닝

```
1 epochs = 10
2
3 for epoch in range(1, epochs+1):
4     model.train()
5     train_loss = 0
6     correct = 0
7     for batch_idx, (data, target) in enumerate(train_loader):
8         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
9         optimizer.zero_grad()                                # 옵티마이저 초기화
10        output = model(data)                             # 트레이닝 데이터를 모델에 입력하여 출력 계산
11        loss = loss_func(output, target)                 # 오차값 계산
12        loss.backward()                                 # 모델 파라미터가 오차값에 대한 평균분값 계산
13        optimizer.step()                               # 경사하강법으로 파라미터 학습
14        train_loss += loss.item()                      # 오차값 누적
15        pred = output.argmax(dim=1)                  # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
16        correct += sum(pred==target).item()           # 예측값과 타깃값이 같은 개수 누적
17        train_loss /= len(train_loader.dataset)       # 평균 오차값 계산
18        print('Train Epoch {} \tLoss: {:.6f} \tAccuracy: {}/{} ({:.0f}%)'.format(epoch, train_loss,
19                                         correct, len(train_loader.dataset),
20                                         100 * correct / len(train_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

▣ 학습 결과 출력

Train Epoch 1 Loss: 0.026858 Accuracy: 31782/60000 (53%)
Test set: Average loss: 0.0008, Accuracy: 7851/10000 (79%)

Train Epoch 2 Loss: 0.009245 Accuracy: 50297/60000 (84%)
Test set: Average loss: 0.0004, Accuracy: 8804/10000 (88%)

Train Epoch 3 Loss: 0.006328 Accuracy: 53223/60000 (89%)
Test set: Average loss: 0.0004, Accuracy: 8983/10000 (90%)

Train Epoch 4 Loss: 0.005466 Accuracy: 53986/60000 (90%)
Test set: Average loss: 0.0003, Accuracy: 9091/10000 (91%)

Train Epoch 5 Loss: 0.005000 Accuracy: 54490/60000 (91%)
Test set: Average loss: 0.0003, Accuracy: 9132/10000 (91%)

Train Epoch 6 Loss: 0.004641 Accuracy: 54884/60000 (91%)
Test set: Average loss: 0.0003, Accuracy: 9198/10000 (92%)

Train Epoch 7 Loss: 0.004339 Accuracy: 55220/60000 (92%)
Test set: Average loss: 0.0003, Accuracy: 9255/10000 (93%)

Train Epoch 8 Loss: 0.004057 Accuracy: 55573/60000 (93%)
Test set: Average loss: 0.0002, Accuracy: 9308/10000 (93%)

Train Epoch 9 Loss: 0.003802 Accuracy: 55872/60000 (93%)
Test set: Average loss: 0.0002, Accuracy: 9347/10000 (93%)

Train Epoch 10 Loss: 0.003568 Accuracy: 56111/60000 (94%)
Test set: Average loss: 0.0002, Accuracy: 9370/10000 (94%)



4. 다층 퍼셉트론 모델 학습

▣ 모델 테스트

```
22 model.eval()
23 test_loss = 0
24 correct = 0
25 with torch.no_grad():
26     for data, target in test_loader:
27         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
28         output = model(data)                                # 테스트 데이터를 모델에 입력하여 출력 계산
29         test_loss += loss_func(output, target).item()       # 오차값 계산
30         pred = output.argmax(dim=1)                         # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
31         correct += pred.eq(target).sum().item()            # # 예측값과 타깃값이 같은 개수 누적
32
33 test_loss /= len(test_loader.dataset)                      # 평균 오차 값 계산
34
35 print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(test_loss,
36                                         correct, len(test_loader.dataset),
37                                         100. * correct / len(test_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

■ 모델 트레이닝

```
1 epochs = 10
2
3 for epoch in range(1, epochs+1):
4     model.train()
5     train_loss = 0
6     correct = 0
7     for batch_idx, (data, target) in enumerate(train_loader):
8         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
9         optimizer.zero_grad()                                # 옵티마이저 초기화
10        output = model(data)                             # 트레이닝 데이터를 모델에 입력하여 출력 계산
11        loss = loss_func(output, target)                # 오차값 계산
12        loss.backward()                                 # 모델 파라미터가 오차값에 대한 평균분값 계산
13        optimizer.step()                               # 경사하강법으로 파라미터 학습
14        train_loss += loss.item()                      # 오차값 누적
15        pred = output.argmax(dim=1)                  # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
16        correct += sum(pred==target).item()           # 예측값과 타깃값이 같은 개수 누적
17        train_loss /= len(train_loader.dataset)       # 평균 오차값 계산
18        print('Train Epoch {} \tLoss: {:.6f} \tAccuracy: {}/{} ({:.0f}%)'.format(epoch, train_loss,
19                                         correct, len(train_loader.dataset),
20                                         100 * correct / len(train_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

▣ 모델 테스트

```
22 model.eval()
23 test_loss = 0
24 correct = 0
25 with torch.no_grad():
26     for data, target in test_loader:
27         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
28         output = model(data)                                # 테스트 데이터를 모델에 입력하여 출력 계산
29         test_loss += loss_func(output, target).item()       # 오차값 계산
30         pred = output.argmax(dim=1)                         # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
31         correct += pred.eq(target).sum().item()            # # 예측값과 타깃값이 같은 개수 누적
32
33 test_loss /= len(test_loader.dataset)                      # 평균 오차 값 계산
34
35 print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(test_loss,
36                                         correct, len(test_loader.dataset),
37                                         100. * correct / len(test_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

■ 모델 트레이닝

```
1 epochs = 10
2
3 for epoch in range(1, epochs+1):
4     model.train()
5     train_loss = 0
6     correct = 0
7     for batch_idx, (data, target) in enumerate(train_loader):
8         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
9         optimizer.zero_grad()                                # 옵티마이저 초기화
10        output = model(data)                             # 트레이닝 데이터를 모델에 입력하여 출력 계산
11        loss = loss_func(output, target)                # 오차값 계산
12        loss.backward()                                 # 모델 파라미터가 오차값에 대한 평균분값 계산
13        optimizer.step()                               # 경사하강법으로 파라미터 학습
14        train_loss += loss.item()                      # 오차값 누적
15        pred = output.argmax(dim=1)                  # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
16        correct += sum(pred==target).item()           # 예측값과 타깃값이 같은 개수 누적
17        train_loss /= len(train_loader.dataset)       # 평균 오차값 계산
18        print('Train Epoch {} \tLoss: {:.6f} \tAccuracy: {}/{} ({:.0f}%)'.format(epoch, train_loss,
19                                         correct, len(train_loader.dataset),
20                                         100 * correct / len(train_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

▣ 모델 테스트

```
22 model.eval()
23 test_loss = 0
24 correct = 0
25 with torch.no_grad():
26     for data, target in test_loader:
27         data, target = data.to(device), target.to(device)      # data와 target, 설정한 장치로 이동
28         output = model(data)                                # 테스트 데이터를 모델에 입력하여 출력 계산
29         test_loss += loss_func(output, target).item()       # 오차값 계산
30         pred = output.argmax(dim=1)                         # 10차원 출력 값에서 제일 큰 수의 index를 예측값으로 지정
31         correct += pred.eq(target).sum().item()            # # 예측값과 타깃값이 같은 개수 누적
32
33 test_loss /= len(test_loader.dataset)                      # 평균 오차 값 계산
34
35 print('Test set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)'.format(test_loss,
36                                         correct, len(test_loader.dataset),
37                                         100. * correct / len(test_loader.dataset)))
```



4. 다층 퍼셉트론 모델 학습

▣ 학습 결과 출력

Train Epoch 1 Loss: 0.026858 Accuracy: 31782/60000 (53%)
Test set: Average loss: 0.0008, Accuracy: 7851/10000 (79%)

Train Epoch 2 Loss: 0.009245 Accuracy: 50297/60000 (84%)
Test set: Average loss: 0.0004, Accuracy: 8804/10000 (88%)

Train Epoch 3 Loss: 0.006328 Accuracy: 53223/60000 (89%)
Test set: Average loss: 0.0004, Accuracy: 8983/10000 (90%)

Train Epoch 4 Loss: 0.005466 Accuracy: 53986/60000 (90%)
Test set: Average loss: 0.0003, Accuracy: 9091/10000 (91%)

Train Epoch 5 Loss: 0.005000 Accuracy: 54490/60000 (91%)
Test set: Average loss: 0.0003, Accuracy: 9132/10000 (91%)

Train Epoch 6 Loss: 0.004641 Accuracy: 54884/60000 (91%)
Test set: Average loss: 0.0003, Accuracy: 9198/10000 (92%)

Train Epoch 7 Loss: 0.004339 Accuracy: 55220/60000 (92%)
Test set: Average loss: 0.0003, Accuracy: 9255/10000 (93%)

Train Epoch 8 Loss: 0.004057 Accuracy: 55573/60000 (93%)
Test set: Average loss: 0.0002, Accuracy: 9308/10000 (93%)

Train Epoch 9 Loss: 0.003802 Accuracy: 55872/60000 (93%)
Test set: Average loss: 0.0002, Accuracy: 9347/10000 (93%)

Train Epoch 10 Loss: 0.003568 Accuracy: 56111/60000 (94%)
Test set: Average loss: 0.0002, Accuracy: 9370/10000 (94%)



Homework#1

▣ FashionMNIST 데이터로 의류 분류 모델 만들기

✓ 데이터 불러오기

```
1 train_data = datasets.FashionMNIST('./data', train=True, download=True, transform=transforms.ToTensor())
2 test_data = datasets.FashionMNIST('./data', train=False, download=True, transform=transforms.ToTensor())
```



들어가기

| 학습목표 |

- 이론으로 배운 신경망 모델을 실습을 통해 구현하는 방법을 학습하고, 현실 데이터를 사용하여 다층 퍼셉트론 모델을 만들어 본다.

들어가기

| 학습내용 |

- 파이토치 (pytorch) 설치

- 데이터셋

MNIST 데이터셋을 불러오고, 트레이닝 데이터와 테스트 데이터로 분할하는 법을 학습한다.

- 다층 퍼셉트론 모델 정의

다층 퍼셉트론 모델, 크로스 엔트로피 오차, 그리고 경사 하강법 함수를 정의한다.

- 다층 퍼셉트론 모델 학습

경사 하강법을 통해 다층 퍼셉트론 모델을 학습하고 테스트 한다.

들어가기

| 학습목차 |

1 | 파이토치 (pytorch) 설치

2 | 데이터셋

3 | 다층 퍼셉트론 모델 정의

4 | 다층 퍼셉트론 모델 학습



1. 파이토치 (pytorch) 설치

□ <https://pytorch.org/get-started/locally/>

PyTorch Build	Stable (1.9.1)	Preview (Nightly)	LTS (1.8.2)
Your OS	Linux	Mac	Windows
Package	Conda	Pip	LibTorch Source
Language	Python		C++ / Java
Compute Platform	CUDA 10.2	CUDA 11.1	ROCM 4.2 (beta) CPU
Run this Command:	<pre>conda install pytorch torchvision torchaudio cudatoolkit=10.2 -c pytorch</pre>		



1. 파이토치 (pytorch) 설치

명령 프롬프트

Microsoft Windows [Version 10.0.19041.1237]
(c) Microsoft Corporation. All rights reserved.

```
C:\Users\jintongbin\lab>conda install pytorch torchvision cudatoolkit=10.2 -c pytorch
```



2. 데이터셋

필요한 모듈 불러오기

1 | 2 | 3 | 4

모델과 학습에 관련된 torch, nn, functional, optim 모듈 불러오기

5

데이터에 관련된 datasets, transforms 모듈 불러오기

6

이미지 시각화 하는 matplotlib 모듈 불러오기

```
1 import torch
2 import torch.nn as nn
3 import torch.nn.functional as F
4 import torch.optim as optim
5 from torchvision import datasets, transforms
6 import matplotlib.pyplot as plt
```



2. 데이터셋

▣ MNIST 데이터셋 정의

1

트레이닝 데이터 배치 사이즈 64로 지정

2

테스트 데이터 배치 사이즈 1000으로 지정

4

train_data에 트레이닝 데이터 저장 (텐서 형태로 변환)

5

test_data에 테스트 데이터 저장 (텐서 형태로 변환)

7

트레이닝 데이터 중 한번에 64개 데이터를 반환하는 DataLoader 만들기

8

테스트 데이터 중 한번에 1000개 데이터를 반환하는 DataLoader 만들기

```
1 batch_size = 64
2 test_batch_size = 1000
3
4 train_data = datasets.MNIST('./data', train=True, download=True, transform=transforms.ToTensor())
5 test_data = datasets.MNIST('./data', train=False, download=True, transform=transforms.ToTensor())
6
7 train_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size, shuffle=True)
8 test_loader = torch.utils.data.DataLoader(test_data, batch_size=test_batch_size)
```



2. 데이터셋

트레이닝/테스트 데이터 정보

1 | 2

트레이닝/테스트 데이터 개수 출력

```
1 print(len(train_loader.dataset))  
2 print(len(test_loader.dataset))
```

60000
10000

00000000000000000000
111111111111111111
222222222222222222
333333333333333333
444444444444444444
555555555555555555
666666666666666666
777777777777777777
888888888888888888
999999999999999999



2. 데이터셋

▣ 트레이닝/테스트 데이터 정보

1 | 2

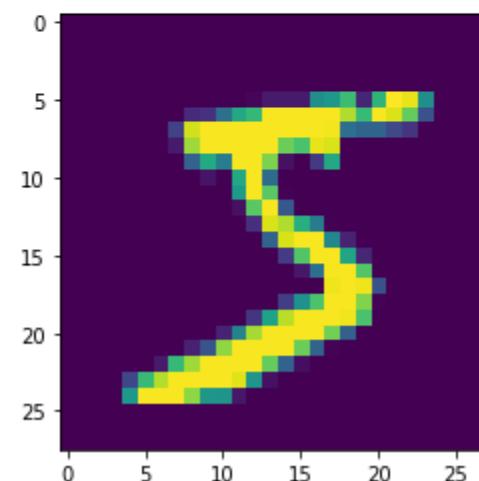
트레이닝 데이터 중 첫 번째 데이터 크기 및 타겟 출력

4 | 5

트레이닝 데이터 중 첫 번째 데이터 이미지 시각화

```
1 print(train_loader.dataset[0][0].shape)
2 print(train_loader.dataset[0][1])
3
4 plt.imshow(train_loader.dataset[0][0][0])
5 plt.show()
```

```
torch.Size([1, 28, 28])
5
```



들어가기

| 학습내용 |

- 다층 퍼셉트론의 구조와 원리
- 다층 퍼셉트론의 일반화된 표현 능력 (generalized expressive power)
- 오차 역전파 알고리즘 (error backpropagation)을 이용한
다층 퍼셉트론의 학습

들어가기

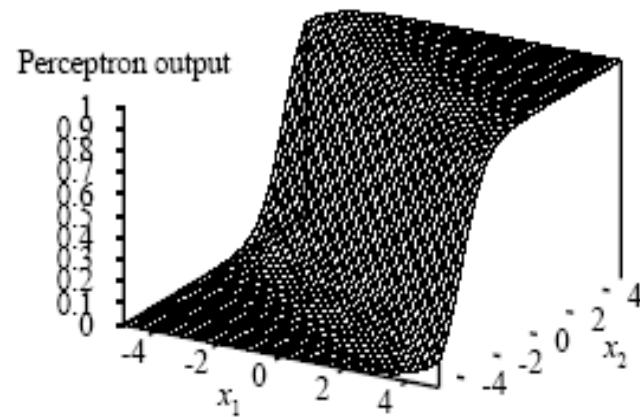
| 학습내용 |

- MLP의 중요한 하이퍼파라미터를 이해하고 최적의 신경망 구조를 설계
- Convolutional NN의 핵심 원리를 이해
- MLP의 다양한 응용분야를 이해

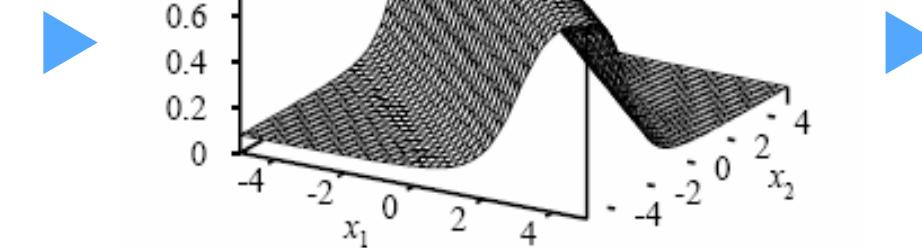


0. Expressiveness of MLP (recap)

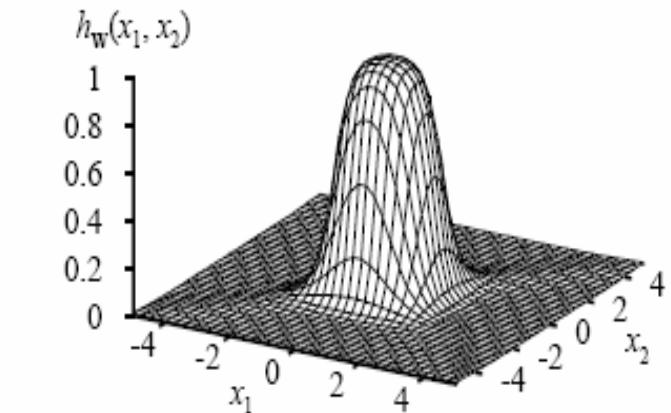
■ Soft threshold



two opposite-facing soft threshold functions을
결합하여 ridge function을 구현



two ridges 를 결합하여
bump function을 구현



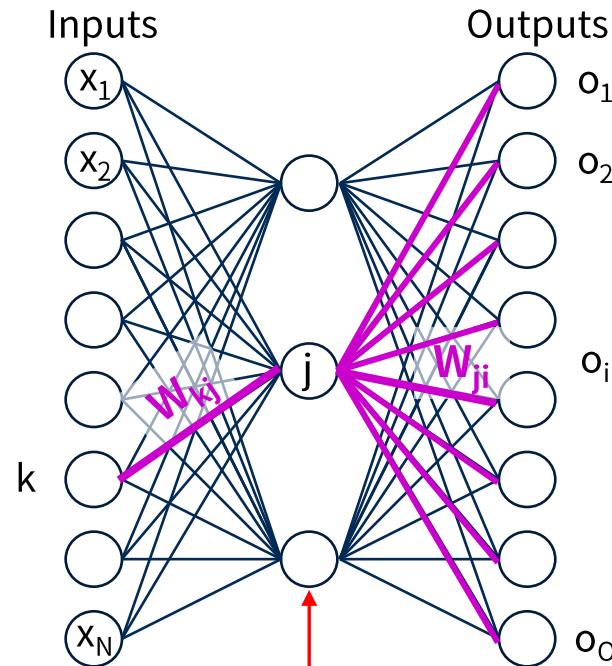
다양한 크기와 위치의
bump function을 결합하여
특정 surface를 표현

All continuous functions w/ 2 layers, all functions w/ 3



0. MLP의 학습 (recap)

Error Backpropagation



Hidden layer: **back-propagate** the error from the output layer

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

$Err_j \rightarrow$ “Error” for hidden node j

Perceptron update:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

$$Err_i = y_i - O_i$$

Output layer weight update (퍼셉트론과 유사)

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

$$\Delta_i = Err_i \times g'(in_i)$$

- ✓ Hidden node j는 자신과 연결된 output node i에서 발생한 에러에 일정비율의 책임이 있다.
 - 그 비율은 hidden node와 output node i 사이의 연결강도에 의해 결정된다



0. MLP의 학습 (recap)

Error Backpropagation

- ✓ Optimization problem: minimize $E = \frac{1}{2} \sum_i (y_i - a_i)^2$
- ✓ Variables: network weights w_{ij}
- ✓ Algorithm: local search via gradient descent

Randomly initialize weights.

Until performance is satisfactory, cycle through examples (epochs):

- Update each weight:

Output node:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$
$$\Delta_i = Err_i \times g'(in_i)$$

Hidden node:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$



■ Network architecture

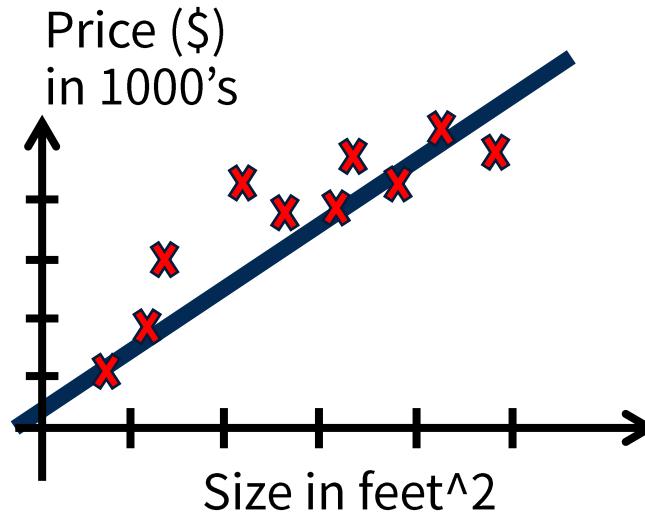
- ✓ #. Hidden layer & #. Hidden nodes per a layer
 - Hidden units의 개수가 많아질 경우, 신경망은 입력 패턴을 기억 → **overfitting**
 - Hidden units의 개수가 너무 작은 경우,
표현력이 부족하여 일반화 (generalization)에 실패 → **underfitting**
- ✓ 노드간의 연결 (fully? Partial?)



1. 신경망 설계

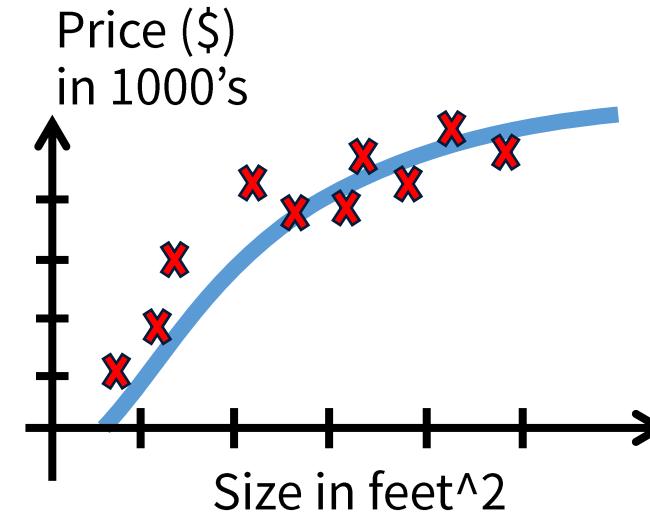
Overfitting vs. Underfitting

- 예제: linear regression



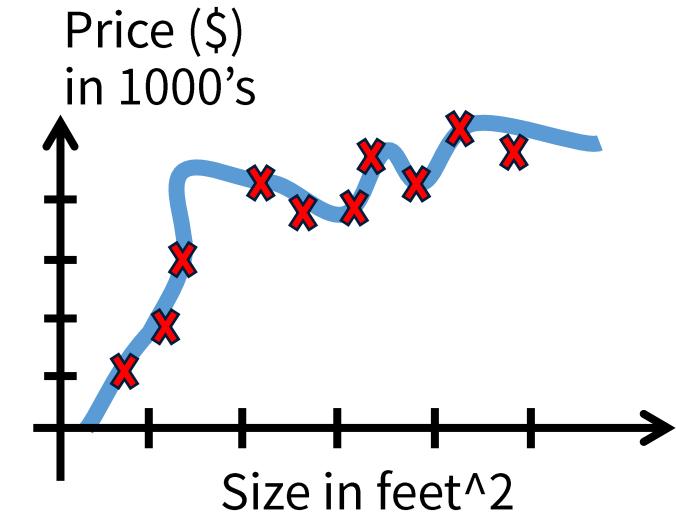
$$h_\theta(x) = \theta_0 + \theta_1 x$$

Underfitting



$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

Just right

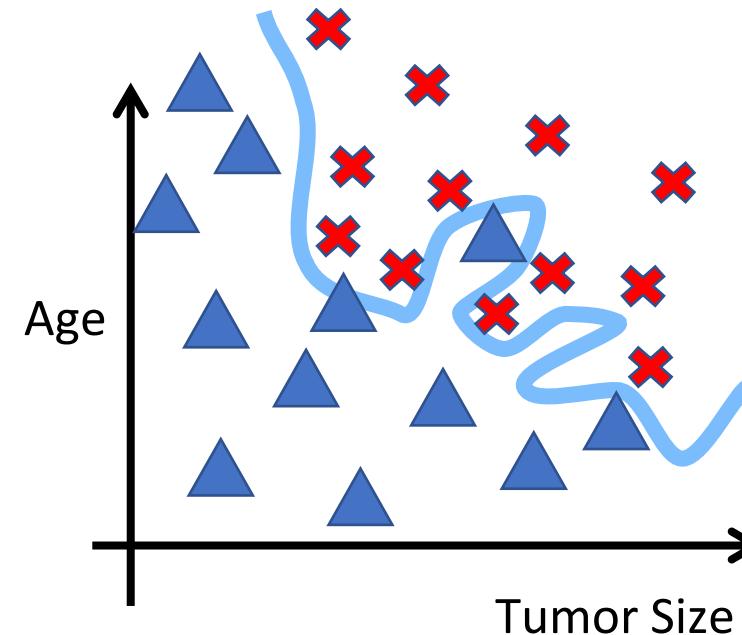


$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4 + \dots$$

Overfitting

Overfitting vs. Underfitting

- ✓ 예제: logistic regression



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$



▣ Overfitting vs. Underfitting

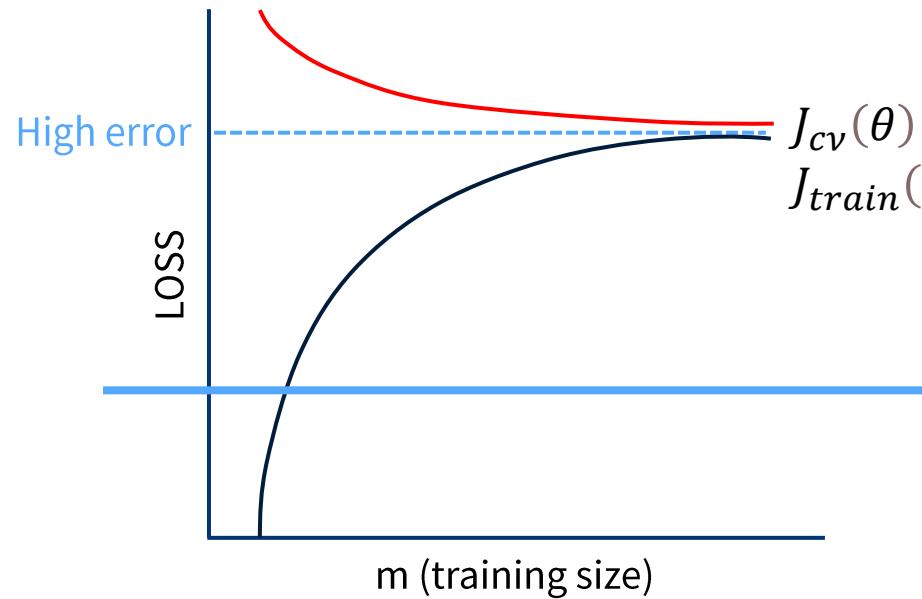
- ✓ Cross validation을 통해 하이퍼파라미터를 최적화
- ✓ 데이터를 *train/validation/test set*으로 구분
 - Training dataset에서 모델파라미터들을 학습
 - Validation dataset에서 하이퍼파라미터들을 결정
 - Test dataset에서의 error를 generalization error로 사용



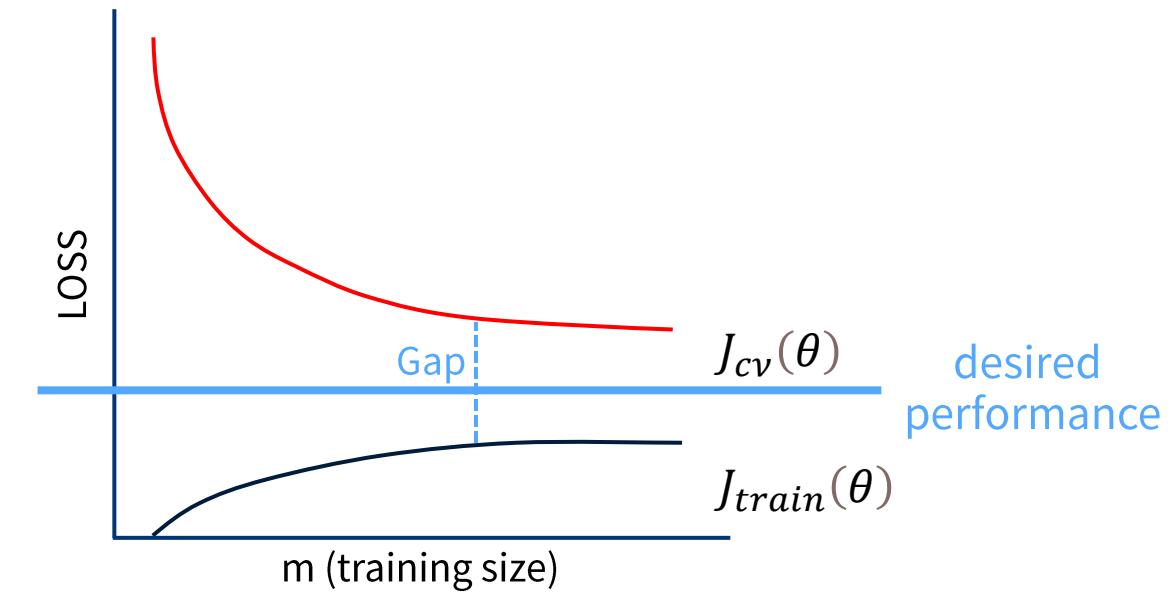
1. 신경망 설계

Overfitting vs. Underfitting

- ✓ Cross validation을 통해 하이퍼파라미터를 최적화
- ✓ 데이터를 *train/validation/test set*으로 구분



Underfit



Overfit



■ Network training time

- ✓ 신경망 학습의 목표는 학습 데이터의 known pattern에 대한 hit ratio와 새로 입력되는 unknown pattern에 대한 hit ratio간의 균형을 이루는 것
(Balance between memorization and generalization)
- ✓ 학습 시간이 너무 긴 경우, overfitting의 가능성이 높아짐
- ✓ Cross-validation을 통해 반복횟수 (#. epochs)를 결정

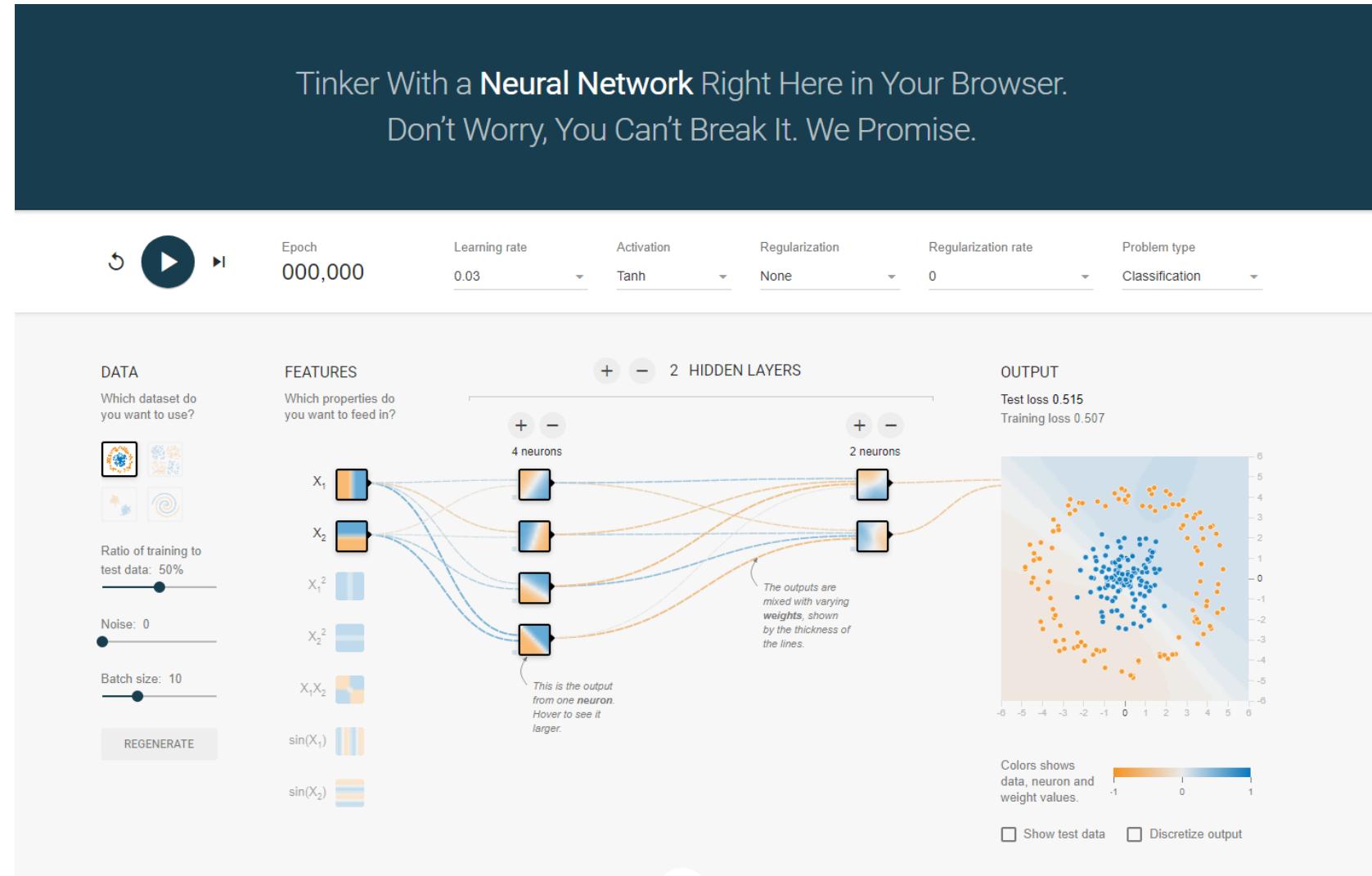


■ MLP expressiveness vs. computational complexity

- ✓ MLP는 모든 일반적인 non-linear function을 표현할 수 있음
- ✓ 하지만, local minima와 해공간(search space)의 높은 차원 때문에 학습하기가 어려움
- ✓ 학습을 통해 도출된 가설을 해석할 수 없음 (**difficult to interpret/be explainable**)



2. 신경망 시연





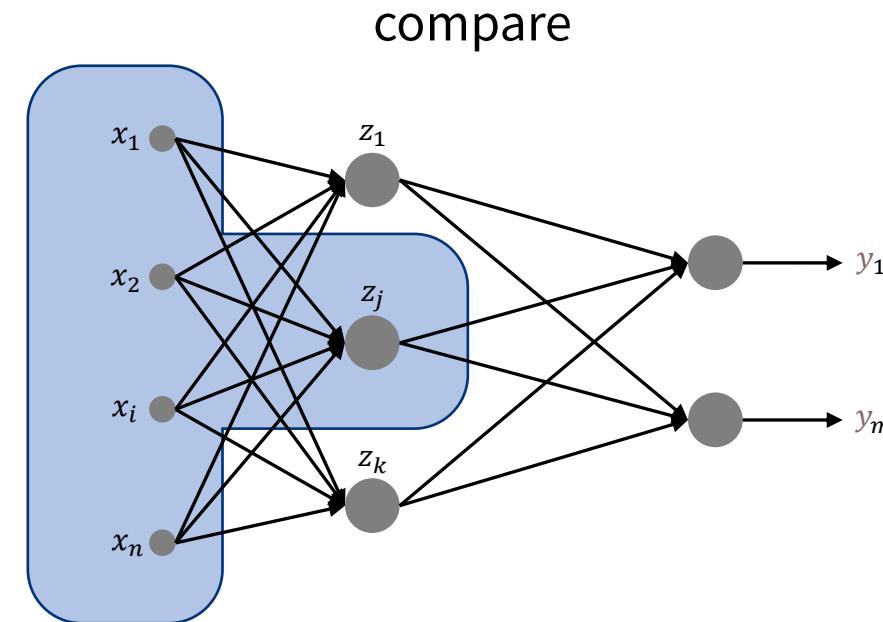
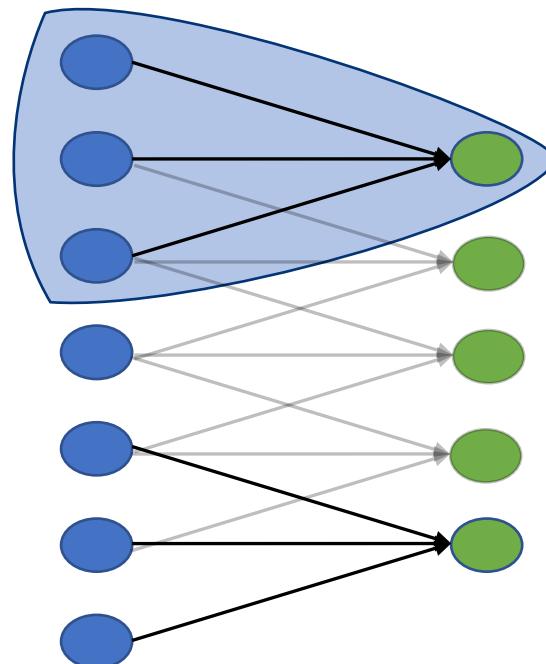
3. From NNs to Convolutional NNs

- Local connectivity
- Shared (“tied”) weights
- Multiple feature maps
- Pooling

3. From NNs to Convolutional NNs

Local connectivity

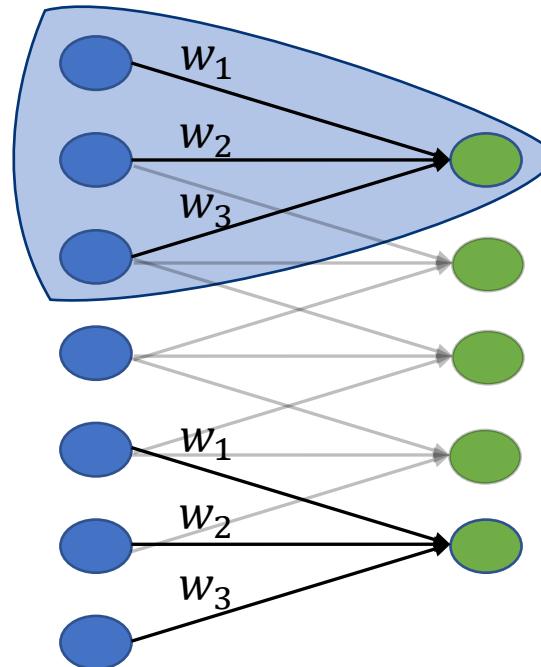
- 각각의 green unit은 3개의 인접한 blue units과 연결



3. From NNs to Convolutional NNs

▣ Shared (“tied”) weights

- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용

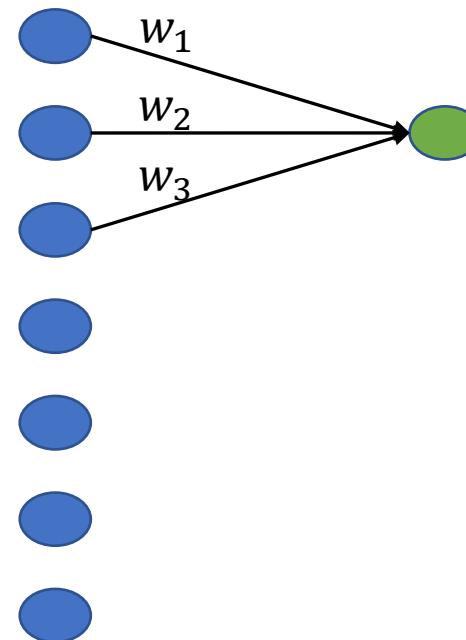




3. From NNs to Convolutional NNs

■ Convolution with 1-D filter:

- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용

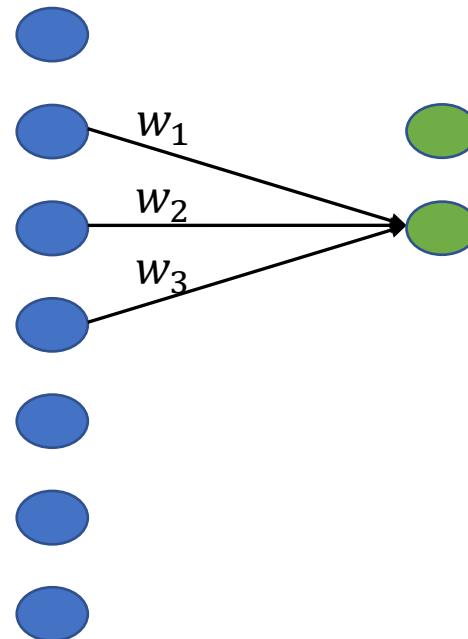




3. From NNs to Convolutional NNs

■ Convolution with 1-D filter:

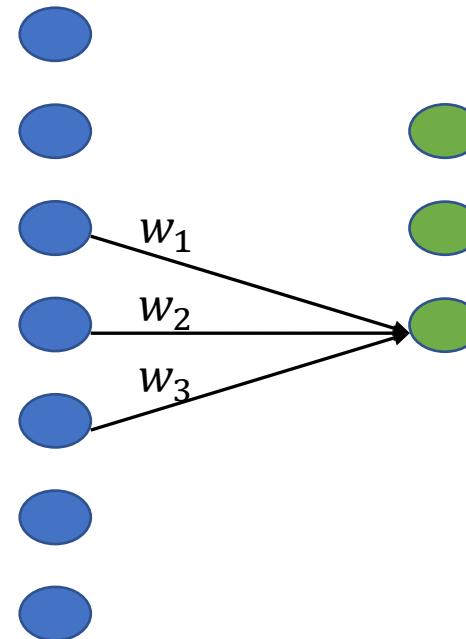
- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용



3. From NNs to Convolutional NNs

■ Convolution with 1-D filter:

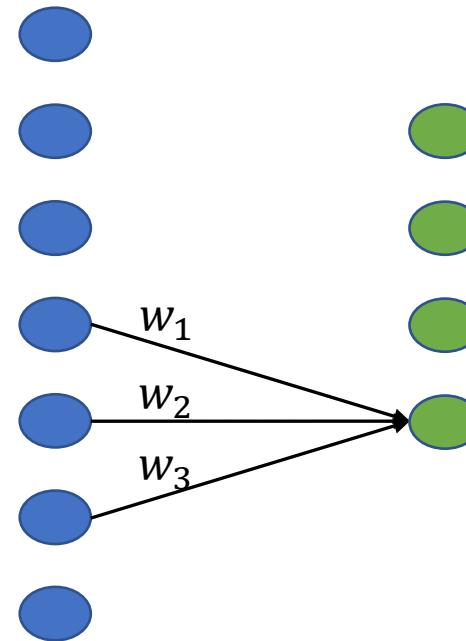
- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용



3. From NNs to Convolutional NNs

■ Convolution with 1-D filter:

- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용

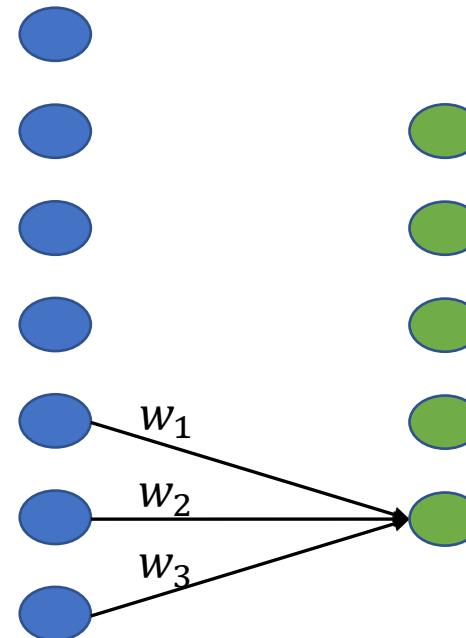




3. From NNs to Convolutional NNs

■ Convolution with 1-D filter:

- ✓ 모든 green units 은 같은 가중치 파라미터 w 를 공유
- ✓ 각 green unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용

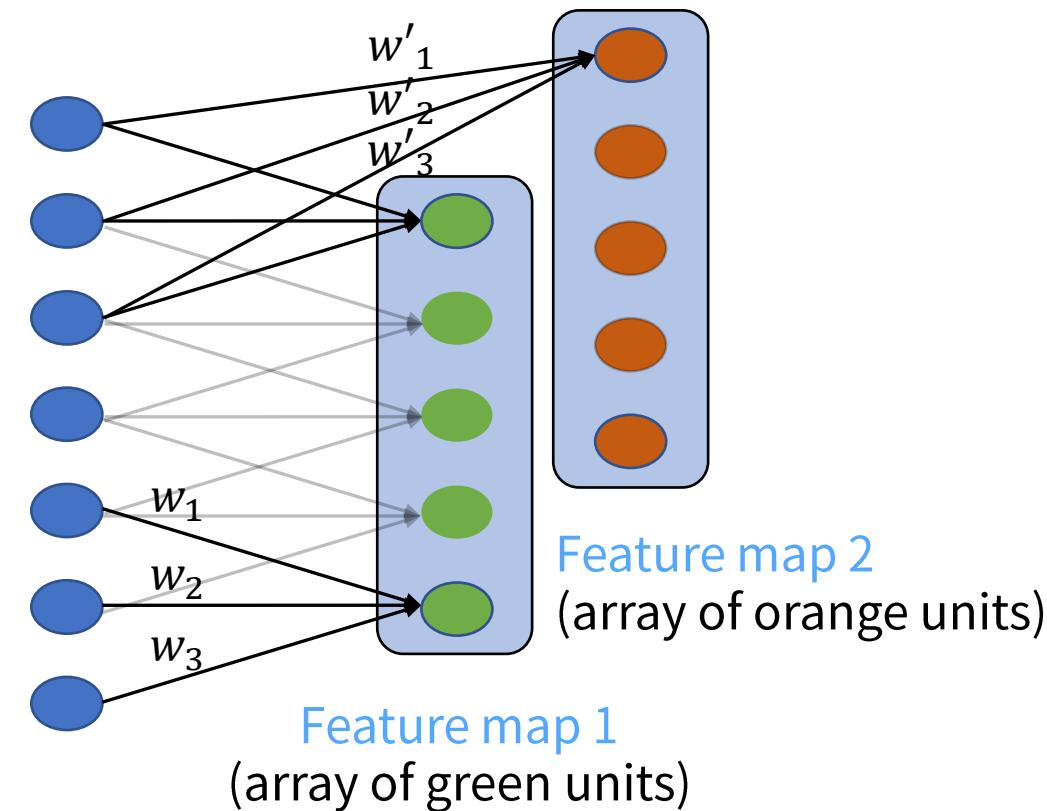




3. From NNs to Convolutional NNs

■ Multiple feature maps

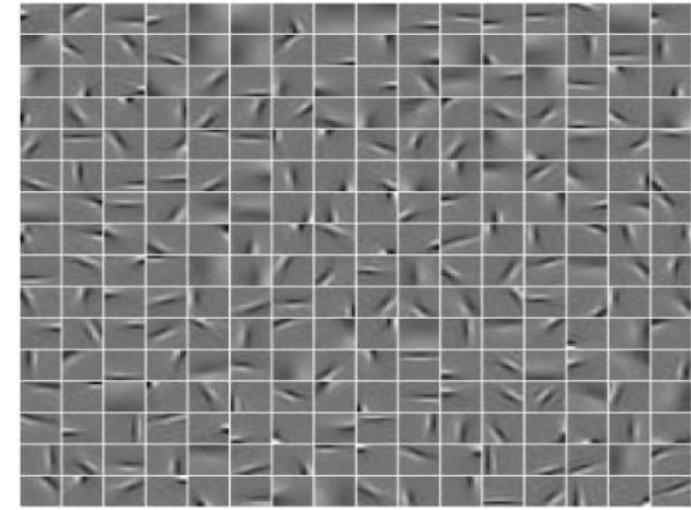
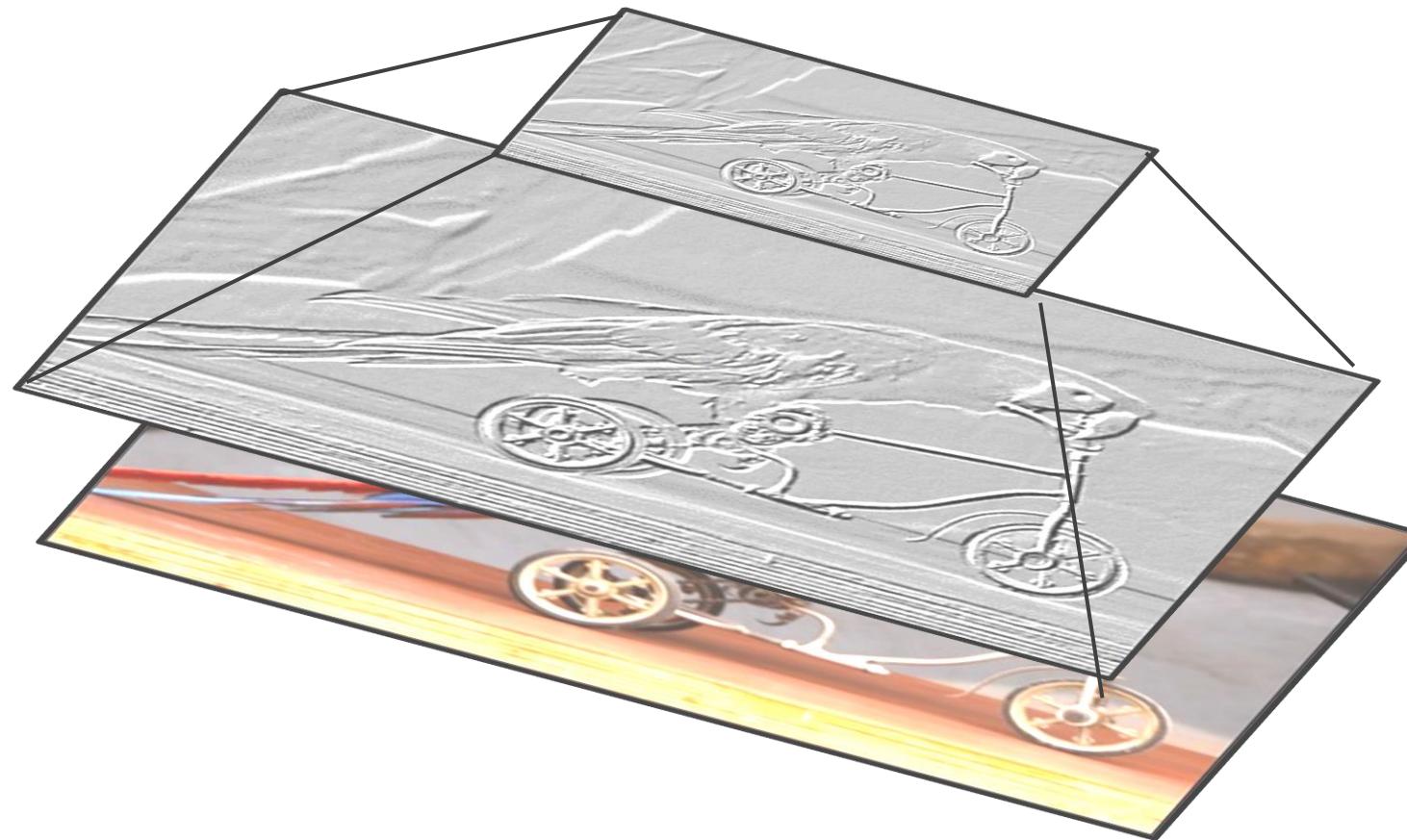
- 모든 orange unit 은 같은 함수를 이용하지만, 서로 다른 입력 (different input window)을 사용
- 모든 orange units과 green units 은 서로 다른 기능 (different kernel)을 수행





3. From NNs to Convolutional NNs

Convolution + Pooling



Pooling



Convolution



Image

정리하기

- ▶ 퍼셉트론 (One-layer network)는 제한된 표현력을 가진다.
즉 Linear decision boundary 만을 생성 → Only linear classifier

- ▶ 퍼셉트론의 학습은 단순하고 효율적임

정리하기

▶ MLP은 충분한 표현력을 가지고 있음

- ▶ MLP는 모든 일반적인 non-linear function을 표현할 수 있음
- ▶ Gradient descent, 즉 back-propagation에 의해 학습

▶ Balance between memorization and generalization

- ▶ 너무 많은 hidden units을 가진 경우, overfitting의 가능성이 높아짐
- ▶ “Prune the NN”

정리하기

- ▶ local minima와 해공간(search space)의 높은 차원 때문에 학습하기가 어려움
- ▶ 응용분야
: speech, vision, driving, handwriting, fraud detection 등

들어가기

| 학습목표 |

- 퍼셉트론 (perceptron)의 기본 구조와 원리를 이해한다.
- 퍼셉트론의 학습 과정을 이해 한다.
- 다층 퍼셉트론 (Multi-layer perceptron)의 구조와 원리를 이해한다.

들어가기

| 학습목표 |

- MLP의 학습 알고리즘인 오류역전파 (back-propagation)을 이해한다.
- 은닉층 (hidden layer)과 은닉노드 (hidden node)를 통한 MLP의 expressive power를 설명할 수 있다.
- 다층 퍼셉트론을 분류기로 활용할 수 있다.

들어가기

| 학습내용 |

- 다층 퍼셉트론의 구조와 원리
- 다층 퍼셉트론의 일반화된 표현 능력 (generalized expressive power)
- 오차 역전파 알고리즘 (error backpropagation)을 이용한
다층 퍼셉트론의 학습

들어가기

| 학습내용 |

- MLP의 중요한 하이퍼파라미터를 이해하고 최적의 신경망 구조를 설계
- Convolutional NN의 핵심 원리를 이해
- MLP의 다양한 응용분야를 이해

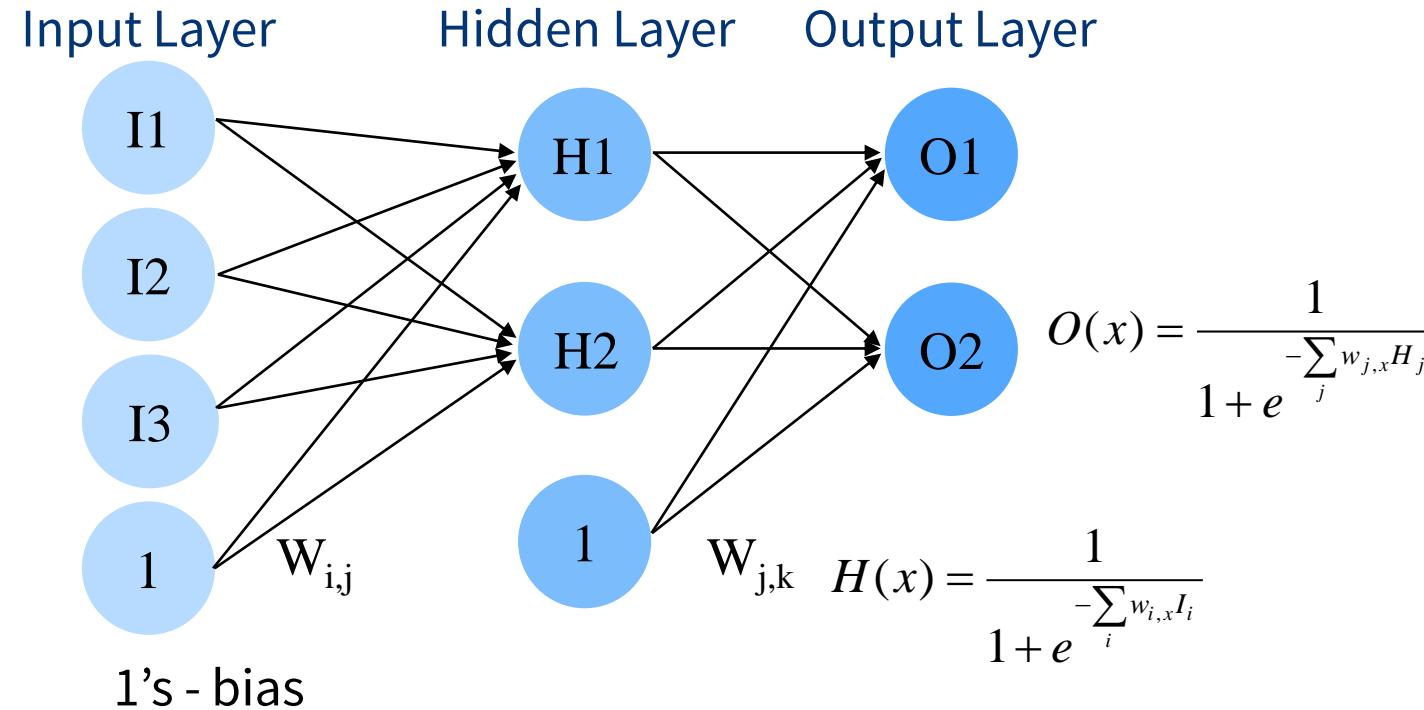
들어가기

| 학습목차 |

- 1 | 생물학적 신경망 vs. 인공신경망
- 2 | 신경망의 역사
- 3 | 퍼셉트론의 구조와 원리 및 학습
- 4 | 다층 퍼셉트론
- 5 | 오류역전파 학습알고리즘 (Backpropagation learning algorithm)
- 6 | 신경망 설계 (NN design)
- 7 | Convolutional NN의 핵심 원리

▣ 퍼셉트론

- ✓ 선형분류기로서의 한계를 극복하기 위해 여러개의 층을 가진 신경망을 고안
(Rumelhart and McClelland, late 70's)
- ✓ Fully connected, feedforward network





1. Hidden Units

■ Hidden units

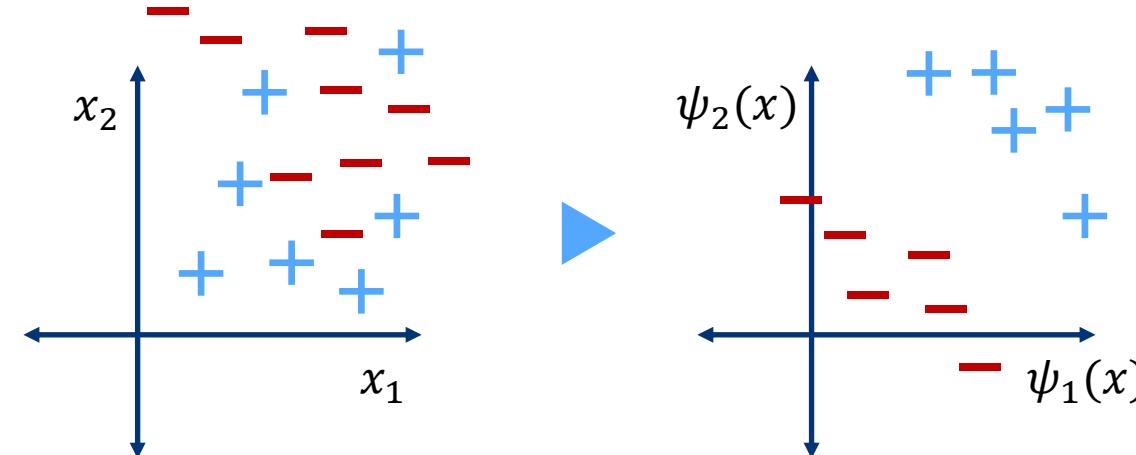
- ✓ 입력노드와 출력 노드 사이에 위치
- ✓ 비선형 함수 (non-linear function) 의 학습을 가능하게 함
- ✓ 입력 특징들의 조합을 표현



1. Hidden Units

■ Neural nets

- ✓ Nonlinear feature mapping을 학습하는 과정
- ✓ 마지막 은닉층을 특징맵 (feature map)으로 고려
- ✓ 마지막 층의 가중치는 생성된 특징맵을 이용한 linear model로 동작



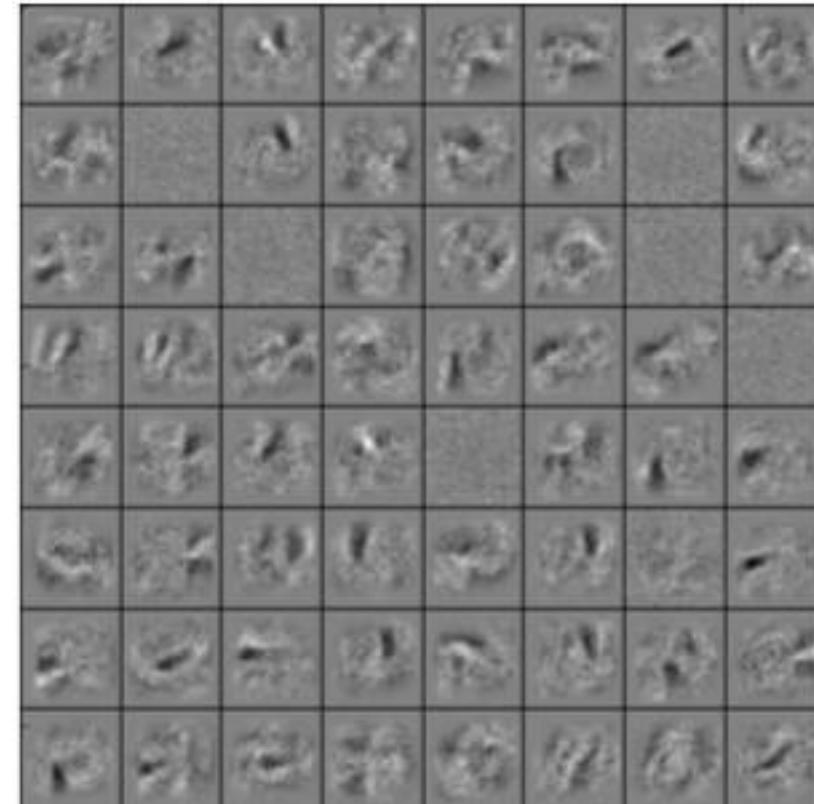


1. Hidden Units

Feature map



MNIST handwritten digit dataset



A subset of learned first layer features:
many of them pick up oriented image



2. Expressiveness of MLP

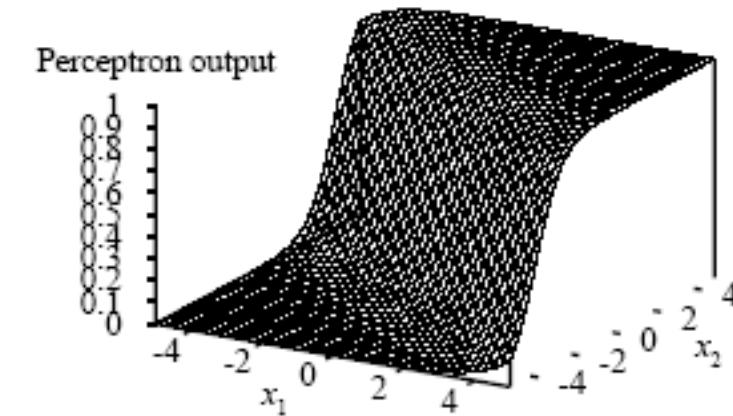
■ Soft threshold

- ✓ 은닉층 추가 → 신경망이 표현할 수 있는 가설의 범위를 확장

각 은닉노드가 입력 특징 공간에서 하나의 soft threshold function 으로서 동작



하나의 출력노드는 이러한 함수들의 soft-thresholded linear combination 으로 동작

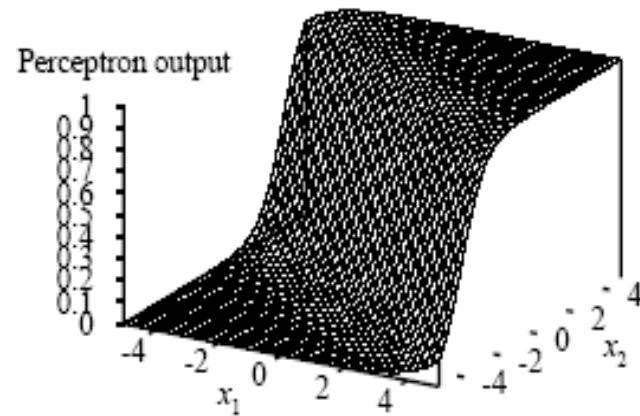


(b)
Soft threshold function

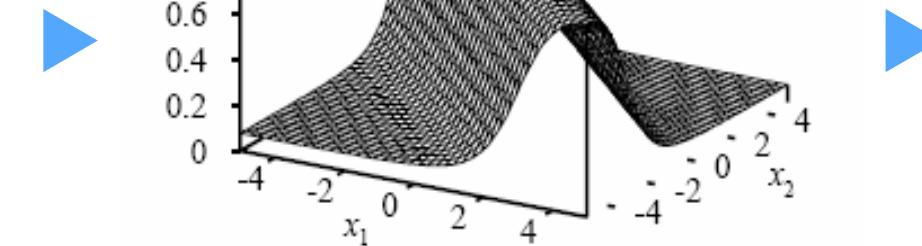


2. Expressiveness of MLP

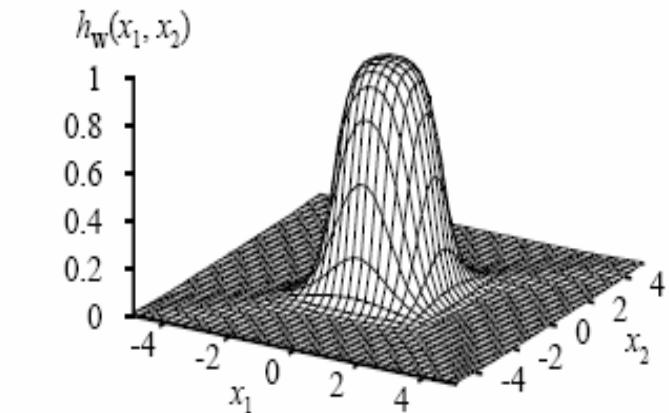
■ Soft threshold



two opposite-facing soft threshold functions을
결합하여 ridge function을 구현



two ridges 를 결합하여
bump function을 구현



다양한 크기와 위치의
bump function을 결합하여
특정 surface를 표현

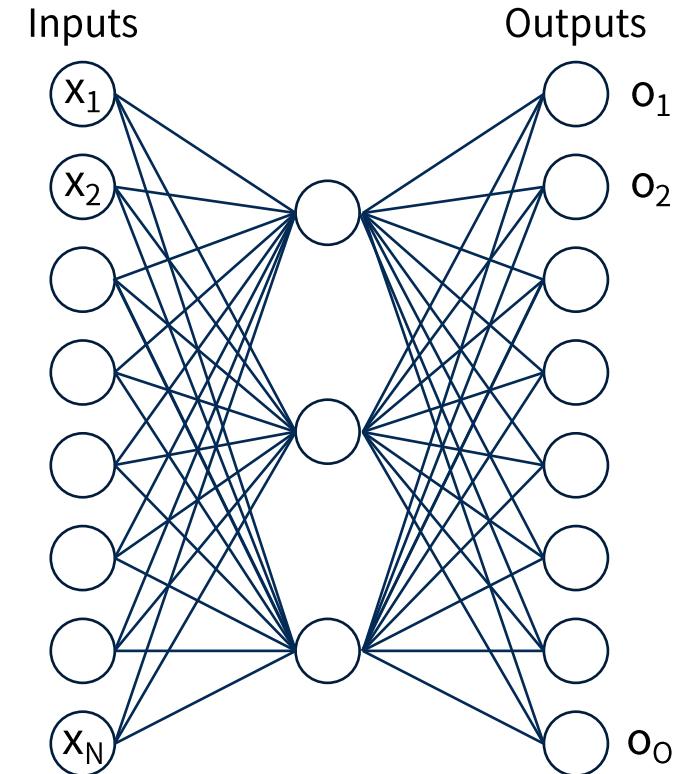
All continuous functions w/ 2 layers, all functions w/ 3



2. Expressiveness of MLP

Cybenko 1988

- ✓ 두개의 은닉층을 가진 신경망의 경우,
임의의 모든 함수 형태에 근사화 할 수 있음



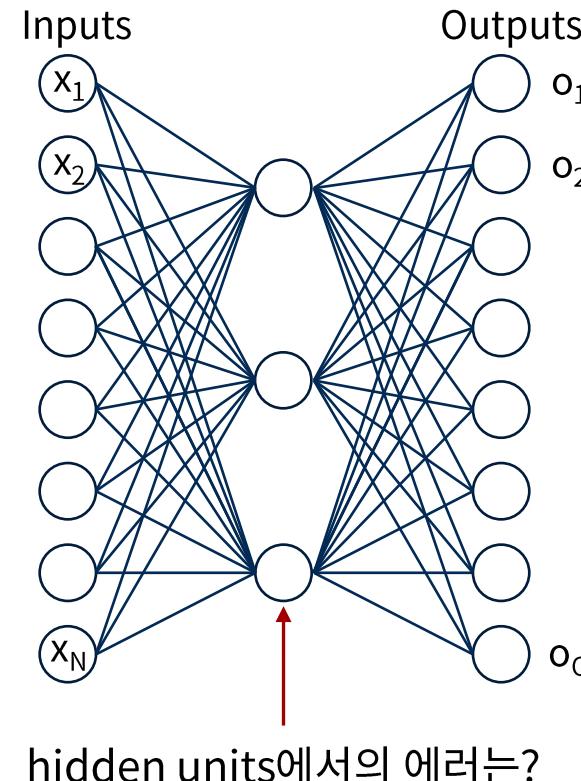
$$o_i = g \left(\sum_h w_{h,i} g \left(\sum_j w_{j,h} x_j \right) \right)$$



3. MLP의 학습

▣ 목표: minimize sum of squared errors

- ✓ 출력층에서 은닉층으로 오차값을 역전파 (error backpropagation)
- ✓ 역전파 과정은 overall error gradient의 미분으로부터 유도



$$\text{Err}_1 = y_1 - o_1$$

$$\text{Err}_2 = y_2 - o_2$$

$$\text{Err}_i = y_i - o_i$$

$$\text{Err}_o = y_o - o_o$$

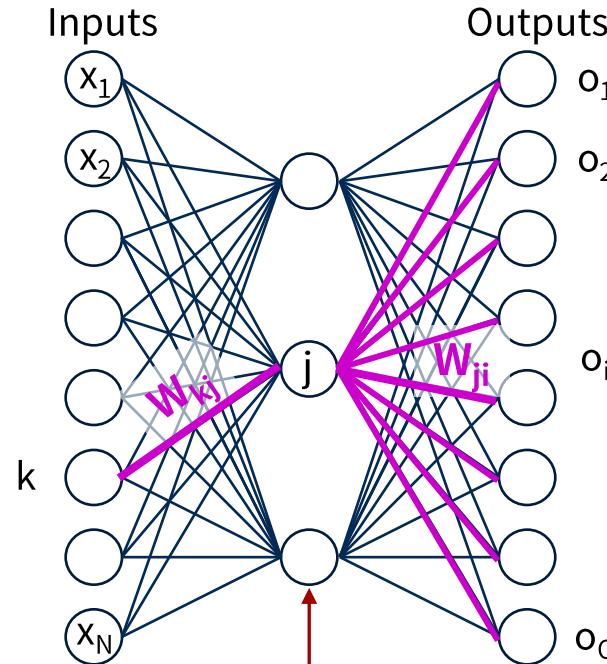
$$o_i = g \left(\sum_h w_{h,i} g \left(\sum_j w_{j,h} x_j \right) \right)$$

parameterized function of inputs:
출력함수의 모델 파라미터는 가중치 (weights).



3. MLP의 학습

Error Backpropagation



Hidden layer: **back-propagate** the error from the output layer

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

$Err_j \rightarrow$ “Error” for hidden node j

Perceptron update:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

$$Err_i = y_i - O_i$$

Output layer weight update (퍼셉트론과 유사)

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

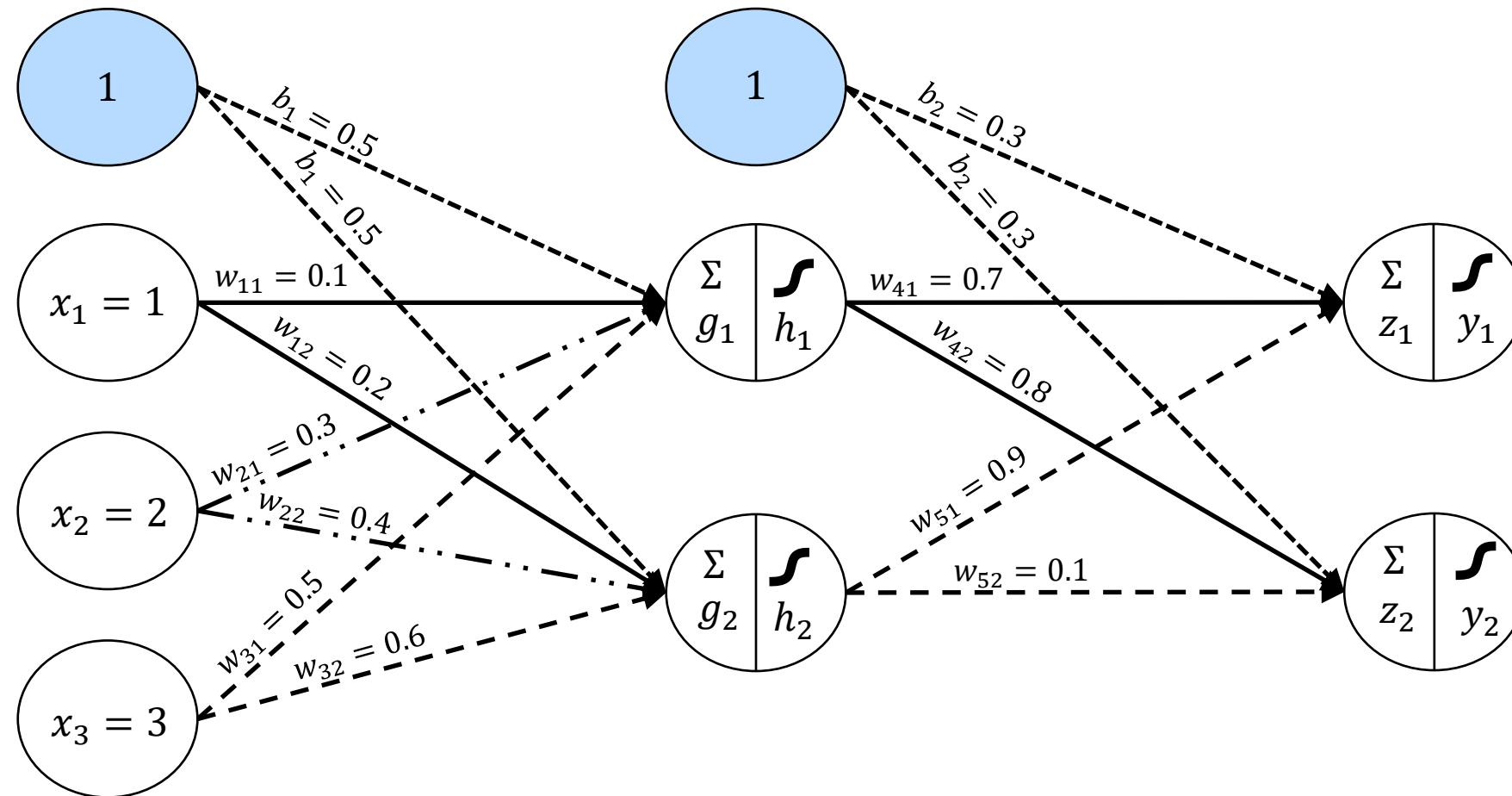
$$\Delta_i = Err_i \times g'(in_i)$$

- ✓ Hidden node j 는 자신과 연결된 output node i 에서 발생한 에러에 일정비율의 책임이 있다.
 - 그 비율은 hidden node와 output node i 사이의 연결강도에 의해 결정된다



3. MLP의 학습

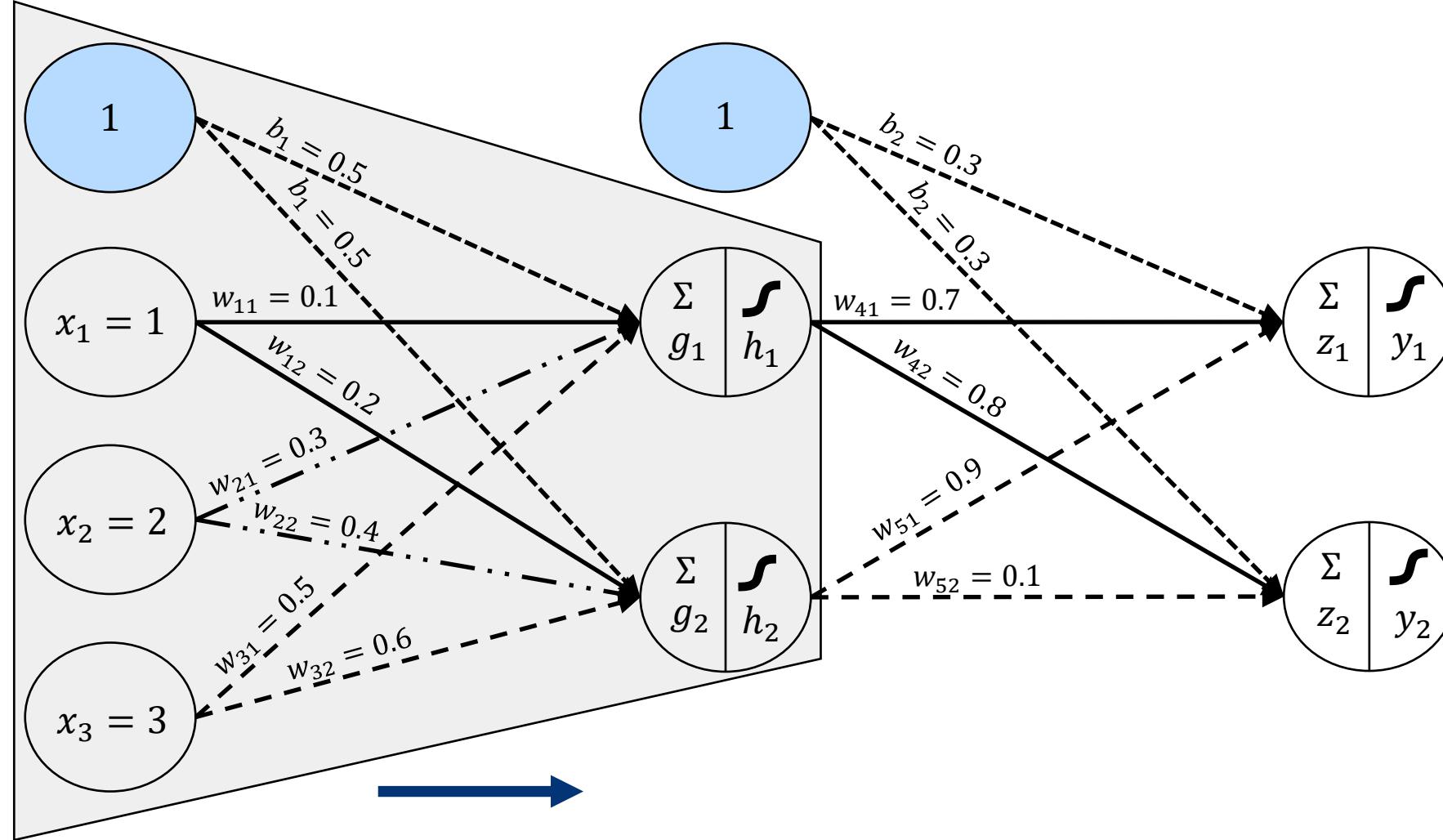
Error Backpropagation: forward step





3. MLP의 학습

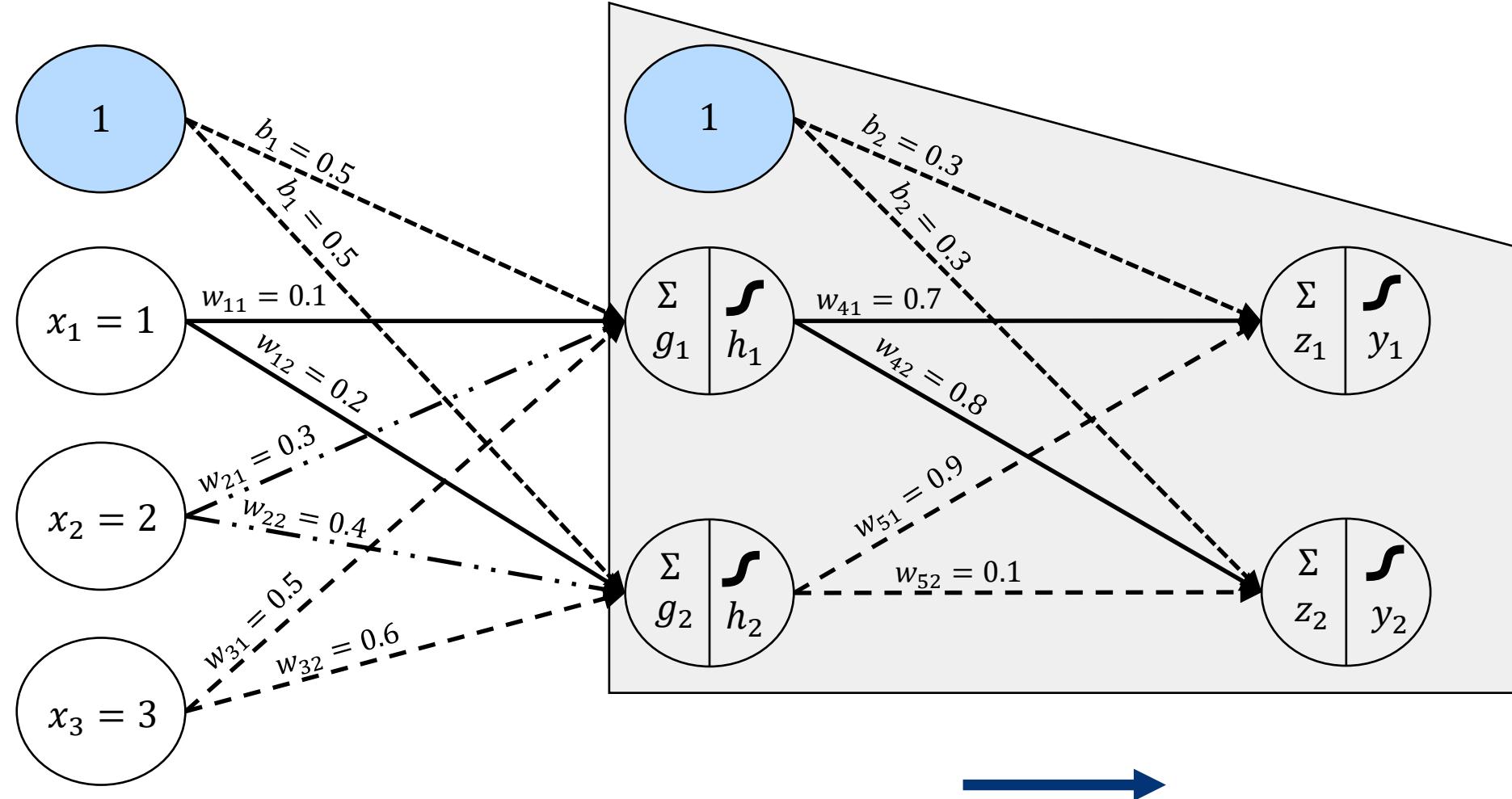
Error Backpropagation: forward step





3. MLP의 학습

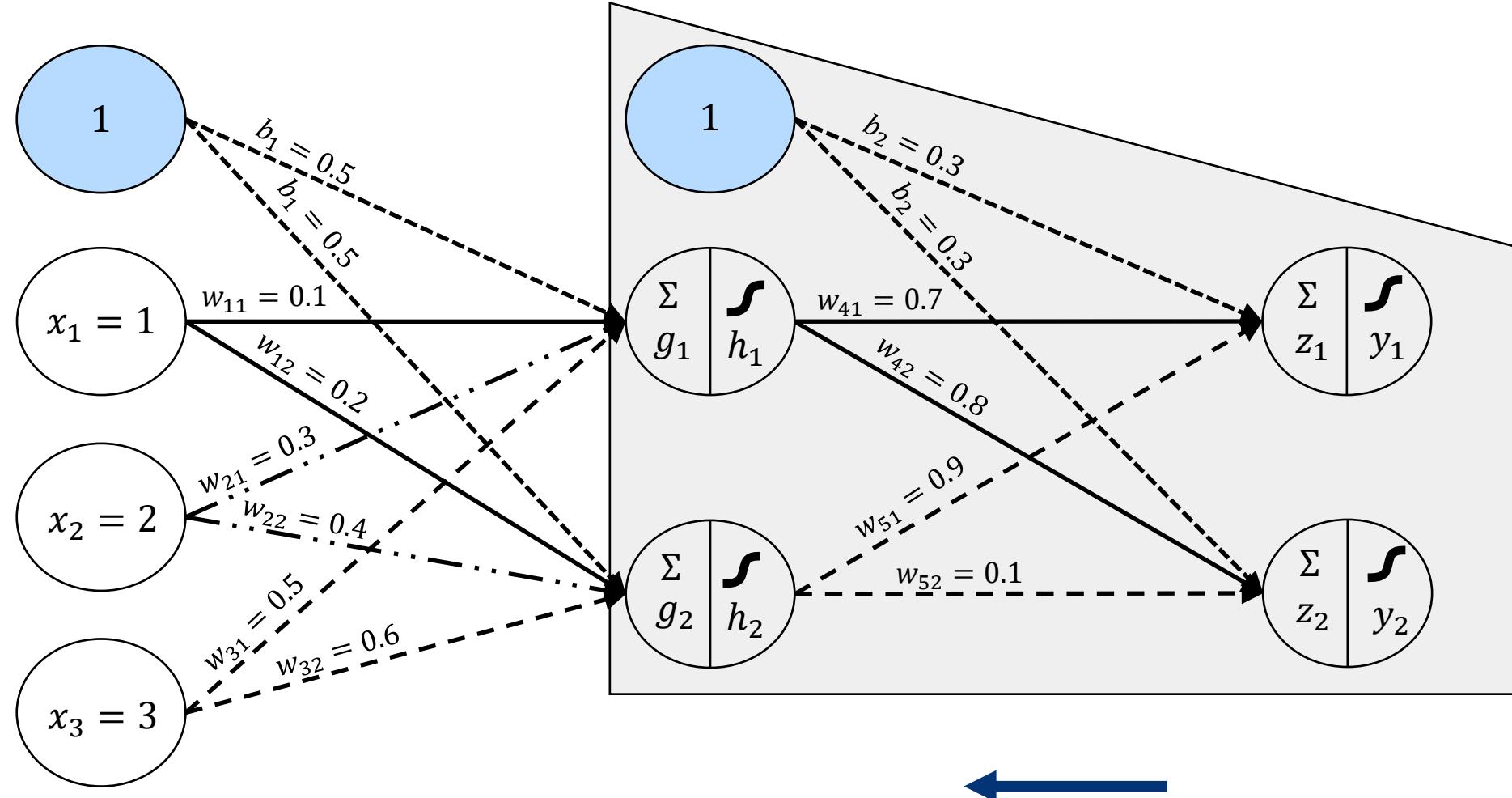
Error Backpropagation: forward step





3. MLP의 학습

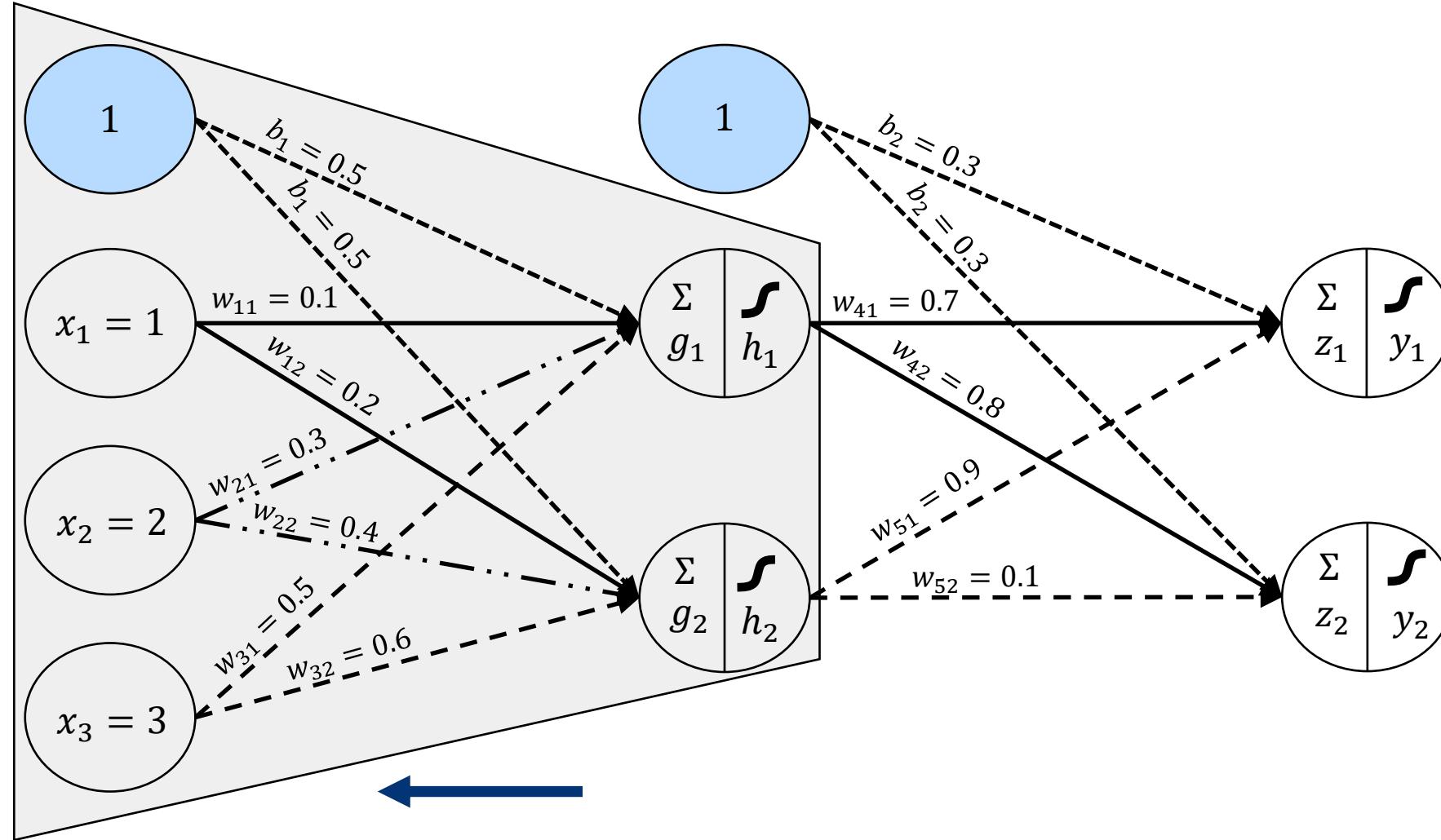
Error Backpropagation: backward step





3. MLP의 학습

Error Backpropagation: backward step





3. MLP의 학습

Error Backpropagation

- ✓ Optimization problem: minimize $E = \frac{1}{2} \sum_i (y_i - a_i)^2$
- ✓ Variables: network weights w_{ij}
- ✓ Algorithm: local search via gradient descent

Randomly initialize weights.

Until performance is satisfactory, cycle through examples (epochs):

- Update each weight:

Output node:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$
$$\Delta_i = Err_i \times g'(in_i)$$

Hidden node:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

학습율 α 의 선택

How many restarts (local optima) of search to find good optimum of objective function?



3. MLP의 학습

Error Backpropagation

- ✓ 퍼셉트론 학습 과정과 유사
 - ✓ Minor difference: 여러 개의 출력값을 가진다.
 - (output vector $h_w(x)$, output vector y)
 - ✓ Major difference:
 - 출력층에서의 에러 ($y - h_w$)는 명확하게 정의가 됨
 - 은닉층에서의 에러는 mysterious (\because 학습데이터에서 은닉노드의 값을 포함하고 있지 않음)
- ✓ 출력층에서 은닉층으로 오차값을 역전파 (error backpropagation)
 - ✓ 역전파 과정은 overall error gradient의 미분으로부터 유도



3. MLP의 학습

Error Backpropagation

Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

Perceptron update:

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

$Err_i \rightarrow i^{\text{th}}$ component of vector $y - h_W$

Hidden layer: **back-propagate** the error from the output layer

$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i$$

Update rule for weights in hidden layer

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j$$



3. MLP의 학습

Derivation in Error Backpropagation

- 하나의 샘플 데이터에 대한 제곱 오차는 출력층에서의 노드상에서 발생하는 오차의 합으로 정의

$$E = \frac{1}{2} \sum_i (y_i - a_i)^2$$

- Derivation of E

$$\begin{aligned}\frac{\partial E}{\partial W_{j,i}} &= -(y_i - a_i) \frac{\partial a_i}{\partial W_{j,i}} = -(y_i - a_i) \frac{\partial g(in_i)}{\partial W_{j,i}} \\ &= -(y_i - a_i)g'(in_i) \frac{\partial in_i}{\partial W_{j,i}} = -(y_i - a_i)g'(in_i) \frac{\partial}{\partial w_{j,i}} \left(\sum_j w_{j,i} a_j \right) \\ &= -(y_i - a_i)g'(in_i) a_j = -a_j \Delta_i\end{aligned}$$



3. MLP의 학습

Derivation in Error Backpropagation

$$\begin{aligned}\frac{\partial E}{\partial W_{k,j}} &= - \sum_i (y_i - a_i) \frac{\partial a_i}{\partial W_{k,j}} = - \sum_i (y_i - a_i) \frac{\partial g(in_i)}{\partial W_{k,j}} \\ &= - \sum_i (y_i - a_i) g'(in_i) \frac{\partial in_i}{\partial W_{k,j}} = - \sum_i \Delta_i \frac{\partial}{\partial W_{k,j}} \left(\sum_j W_{j,i} a_j \right) \\ &= - \sum_i \Delta_i W_{j,i} \frac{\partial a_j}{\partial W_{k,j}} = - \sum_i \Delta_i W_{j,i} \frac{\partial g(in_j)}{\partial W_{k,j}} \\ &= - \sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial in_j}{\partial W_{k,j}} \\ &= - \sum_i \Delta_i W_{j,i} g'(in_j) \frac{\partial}{\partial W_{k,j}} \left(\sum_k W_{k,j} a_k \right) \\ &= - \sum_i \Delta_i W_{j,i} g'(in_j) a_k = - a_k \Delta_j\end{aligned}$$



3. MLP의 학습

Backpropagation Learning algorithm

```
function BACK-PROP-LEARNING(examples, network) returns a neural network
    inputs: examples, a set of examples, each with input vector  $\mathbf{x}$  and output vector  $\mathbf{y}$ 
            network, a multilayer network with  $L$  layers, weights  $W_{j,i}$ , activation function  $g$ 

    repeat
        for each  $e$  in examples do
            for each node  $j$  in the input layer do  $a_j \leftarrow x_j[e]$ 
            for  $\ell = 2$  to  $M$  do
                 $in_i \leftarrow \sum_j W_{j,i} a_j$ 
                 $a_i \leftarrow g(in_i)$ 
                for each node  $i$  in the output layer do
                     $\Delta_i \leftarrow g'(in_i) \times (y_i[e] - a_i)$ 
                for  $\ell = M - 1$  to 1 do
                    for each node  $j$  in layer  $\ell$  do
                         $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$ 
                        for each node  $i$  in layer  $\ell + 1$  do
                             $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$ 
        until some stopping criterion is satisfied
    return NEURAL-NET-HYPOTHESIS(network)
```

정리하기

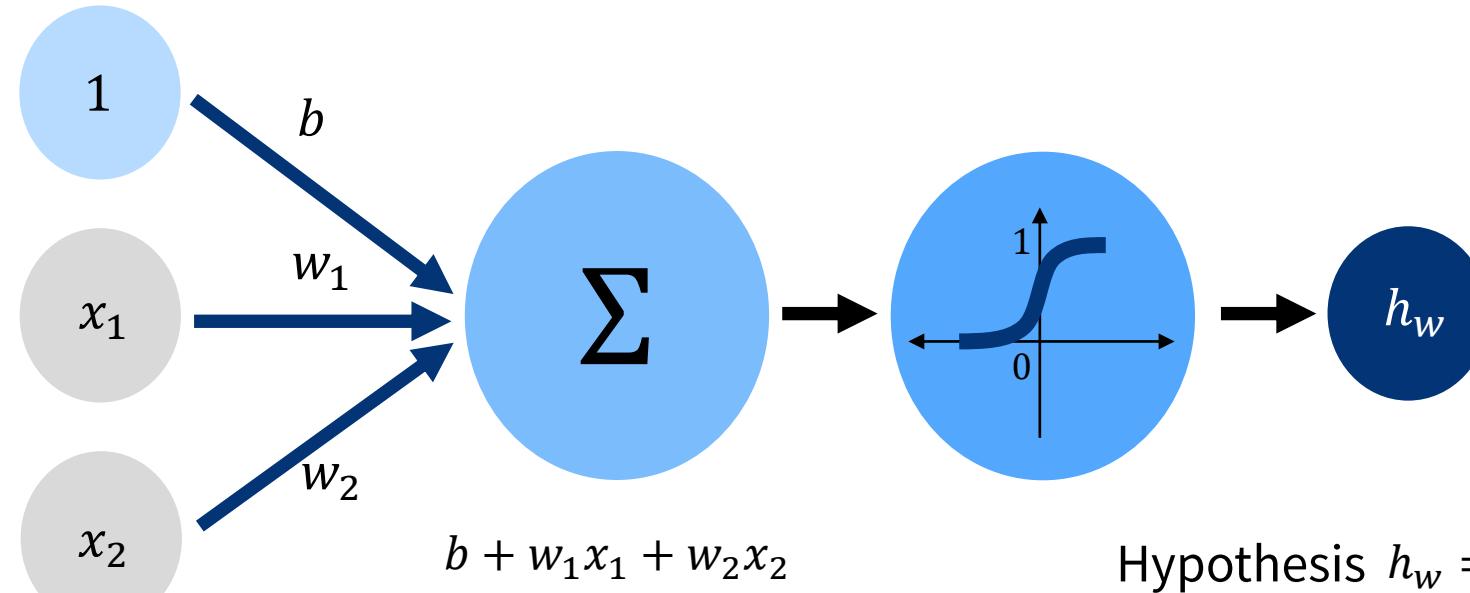
- ▶ Hidden units의 추가로 기존의 single layer NN, 퍼셉트론의 문제점을 극복
- ▶ MLP는 모든 일반적인 non-linear function을 표현할 수 있음
- ▶ Gradient descent, 즉 back-propagation에 의해 학습



0. 퍼셉트론 (recap)

▣ 구조와 원리

- ✓ 입력층: $d+1$ 개의 노드 (특징벡터 X)
- ✓ 출력층: 한 개의 노드 (따라서 2-부류 분류기)
- ✓ 에지와 가중치



$$\text{Hypothesis } h_w = \begin{cases} 1: \left(\sum_i w_i x_i + b \right) > 0 \\ 0 : \text{otherwise} \end{cases}$$

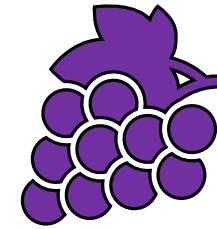


1. 퍼셉트론 (Perceptron)

▣ 예제: 입력 이미지 데이터로부터 레몬과 포도를 분류

1. 이미지 데이터 벡터 변환

실제 이미지



이미지 데이터 행렬

$$\begin{pmatrix} 2 & 3 \\ 5 & 1 \end{pmatrix}$$

이미지 데이터 벡터

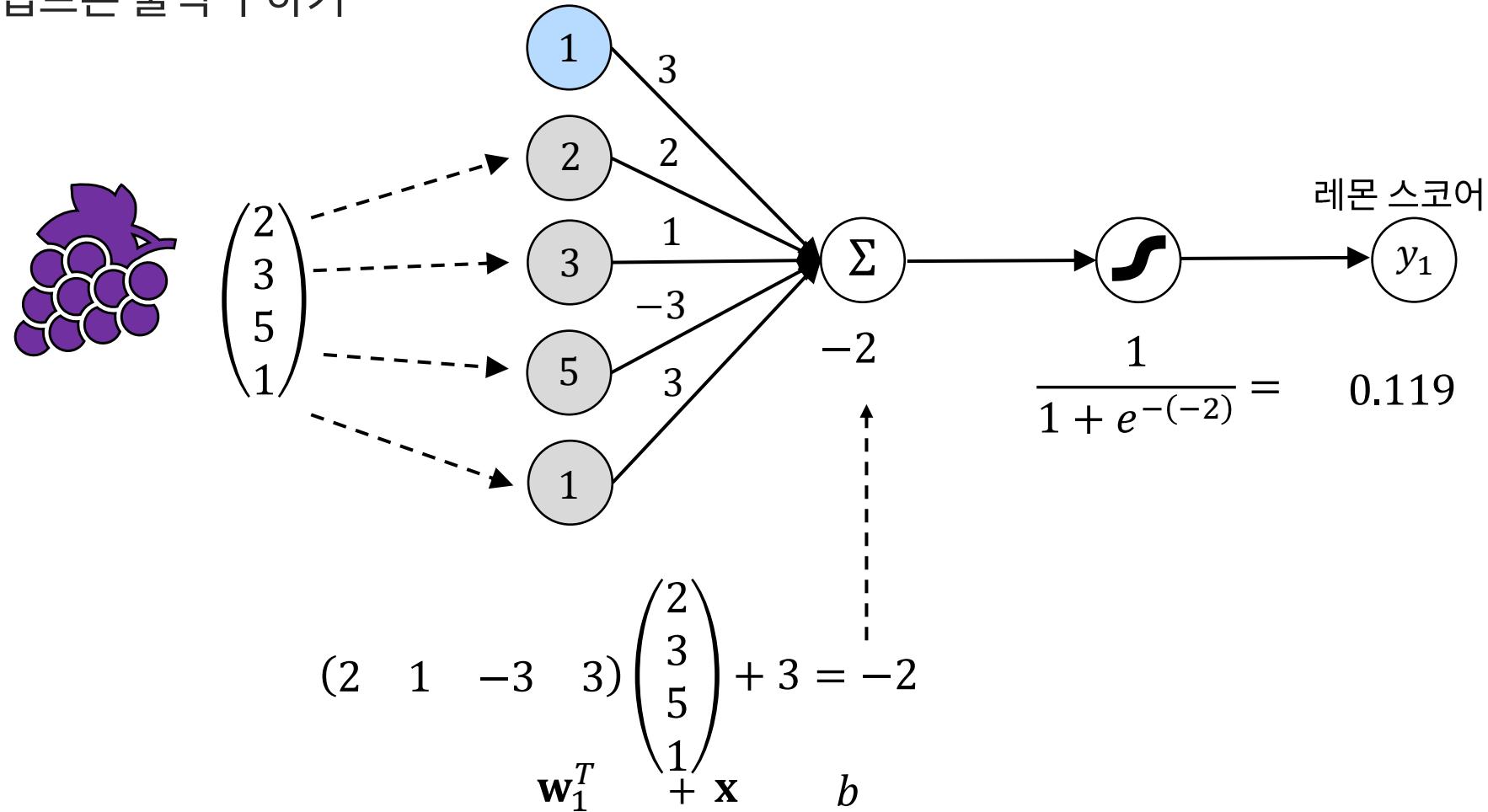
$$\begin{pmatrix} 2 \\ 3 \\ 5 \\ 1 \end{pmatrix}$$



1. 퍼셉트론 (Perceptron)

▣ 예제: 입력 이지 데이터로부터 레몬과 포도를 분류

2. 퍼셉트론 출력 구하기

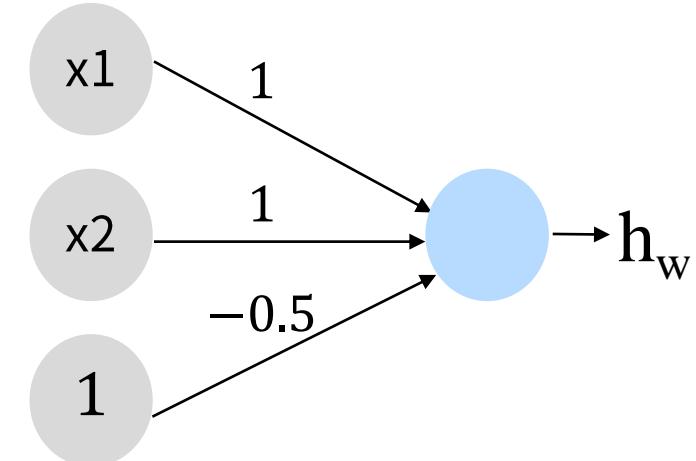
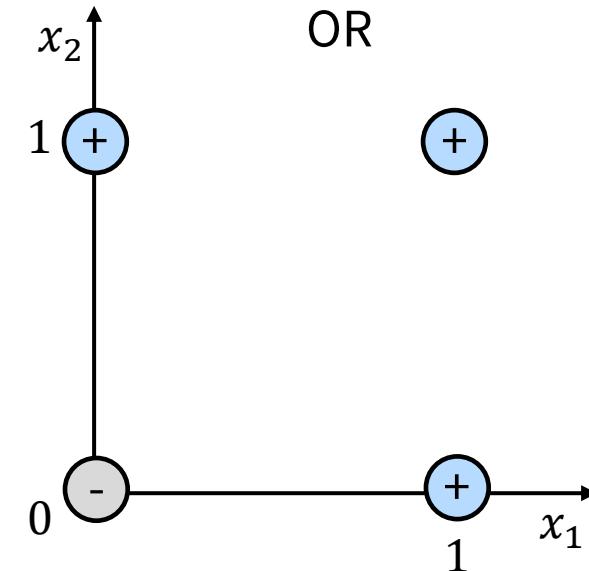




1. 퍼셉트론 (Perceptron)

예제

	x_1	x_2	y
a	0	0	0
b	0	1	1
c	1	0	1
d	1	1	1



- ✓ 초기 가중치 $w=(1,1)^T$, $b= -0.5 \rightarrow$ 결정 직선은 $d(x) = x_1 + x_2 - 0.5$
- ✓ 샘플 a를 인식해보자
- ✓ 나머지 b, c, d는?



2. 퍼셉트론 학습

■ Key idea

- ✓ 샘플 데이터 셋에서의 에러를 줄이는 방향으로 가중치를 조정함으로써 학습
- ✓ 각 예제에 대해 가중치의 수정을 반복
- ✓ 오차: Sum of squared errors



2. 퍼셉트론 학습

■ Training dataset:

m개의 학습데이터가 주어짐 $(x^{(1)}, y^{(1)}), \dots (x^{(m)}, y^{(m)})$

■ Hypothesis

$$h_w(x) = g(w^\top x),$$

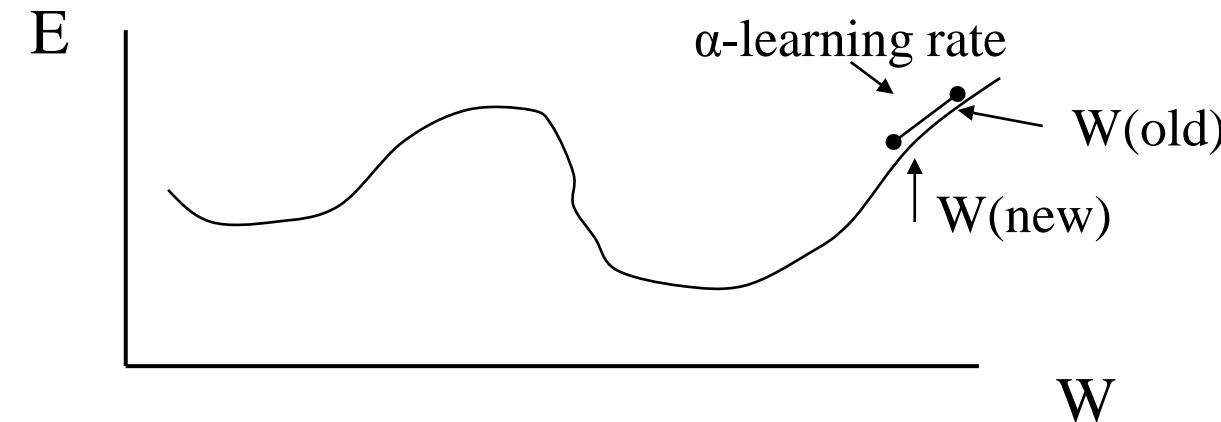
$$\text{where } g(z) = \frac{1}{1+e^{-z}}$$

$$h = \begin{cases} 1: g\left(\sum_i w_i x_i + b\right) > 0 \\ 0: \text{otherwise} \end{cases}$$

■ Cost function

$$E(\mathbf{x}) = \text{SquaredError}(\mathbf{x}) = \frac{1}{2} (y - h_w(\mathbf{x}))^2$$

■ Gradient descent를 이용하여 최적화를 수행





2. 퍼셉트론 학습

■ 입력 x 에 대한 출력 h_w 와 정답간의 오차제곱

$$h_w = g(\sum_j w_j I_j + Q) = \frac{1}{1 + e^{-\sum_j w_j I_j - Q}}$$
$$E = \frac{1}{2} Err^2 \equiv \frac{1}{2} (y - h_w(x))^2$$

■ Gradient descent를 이용하여 최적화를 수행

- 각 가중치(weight), w 에 대한 편미분을 계산

$$\frac{\partial E}{\partial w_j} = Err \times \frac{\partial Err}{\partial w_j} = Err \times \frac{\partial}{\partial w_j} \left(y - g \left(\sum_{j=0}^n w_j x_j \right) \right)$$
$$= -Err \times g'(in) \times x_j$$

- Note: $g'(in)$ 활성화 함수의 미분.
- For sigmoid $g' = g(1-g)$.
- For threshold perceptrons, $g'(n)$ is undefined



2. 퍼셉트론 학습

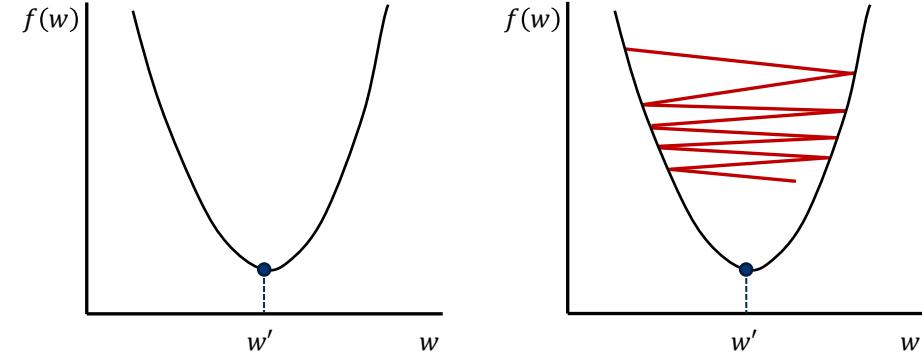
Gradient descent를 이용하여 최적화를 수행

$$\frac{\partial E}{\partial w_j} = -Err \times g'(in) \times x_j$$

- Gradient descent를 이용하여, 오차 E 를 최소화
- 기울기 방향으로 가중치를 업데이트

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

$$W_j \leftarrow W_j + \alpha \times I_j \times Err$$



- 직관적으로
 - $Err = y - h_w(x)$ positive
→ weights are increased for positive inputs and decreased for negative inputs.
 - $Err = y - h_w(x)$ negative
→ opposite



2. 퍼셉트론 학습

■ Perceptron learning rule: $W_j \leftarrow W_j + \alpha \times I_j \times Err$

1. Start with random weights, $\mathbf{w} = (w_1, w_2, \dots, w_n)$.
2. Select a training example $(x, y) \in S$.
3. Run the perceptron with input x and weights \mathbf{w} to obtain g
4. Let α be the training rate (a user-set parameter).

$$\forall w_i, w_i \leftarrow w_i + \Delta w_i,$$

where

$$\Delta w_i = \alpha(y - g(in))g'(in)x_i$$

5. Go to 2.

Epochs are repeated until some stopping criterion is reached—typically, that the weight changes have become very small.

The **stochastic gradient method** selects examples randomly

Epoch → cycle through the examples



Gradient Descent Learning Algorithm

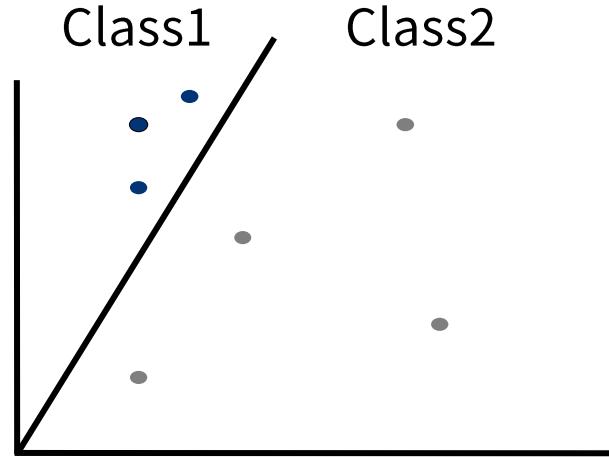
```
function PERCEPTRON-LEARNING(examples, network) returns a perceptron hypothesis
  inputs: examples, a set of examples, each with input  $\mathbf{x} = x_1, \dots, x_n$  and output  $y$ 
          network, a perceptron with weights  $W_j$ ,  $j = 0 \dots n$ , and activation function  $g$ 

  repeat
    for each e in examples do
       $in \leftarrow \sum_{j=0}^n W_j x_j[e]$ 
       $Err \leftarrow y[e] - g(in)$ 
       $W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j[e]$ 
  until some stopping criterion is satisfied
  return NEURAL-NET-HYPOTHESIS(network)
```

Figure 20.21 The gradient descent learning algorithm for perceptrons, assuming a differentiable activation function g . For threshold perceptrons, the factor $g'(in)$ is omitted from the weight update. NEURAL-NET-HYPOTHESIS returns a hypothesis that computes the network output for any given example.

3. 퍼셉트론의 한계

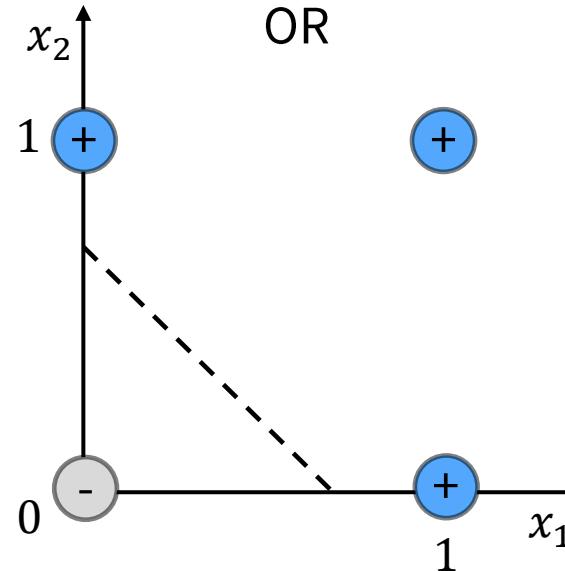
- 선형분류기로 (Linear separable function)로만 동작



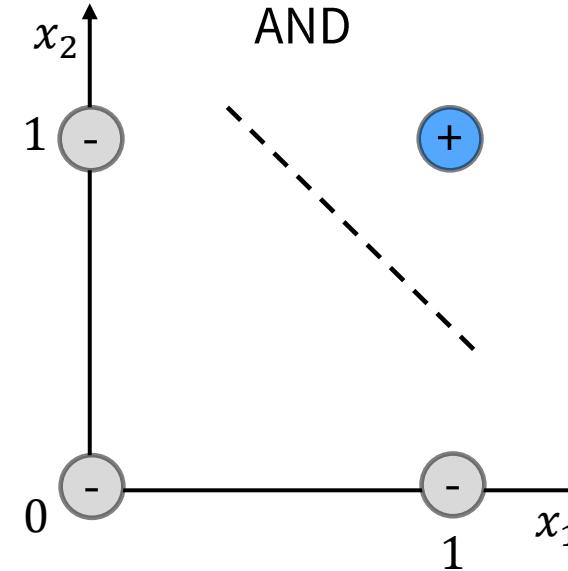


3. 퍼셉트론의 한계

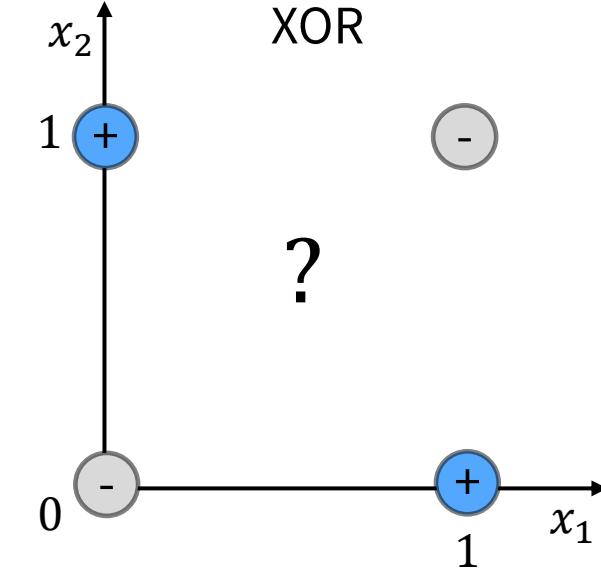
■ 선형분류기로 (Linear separable function)로만 동작



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



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



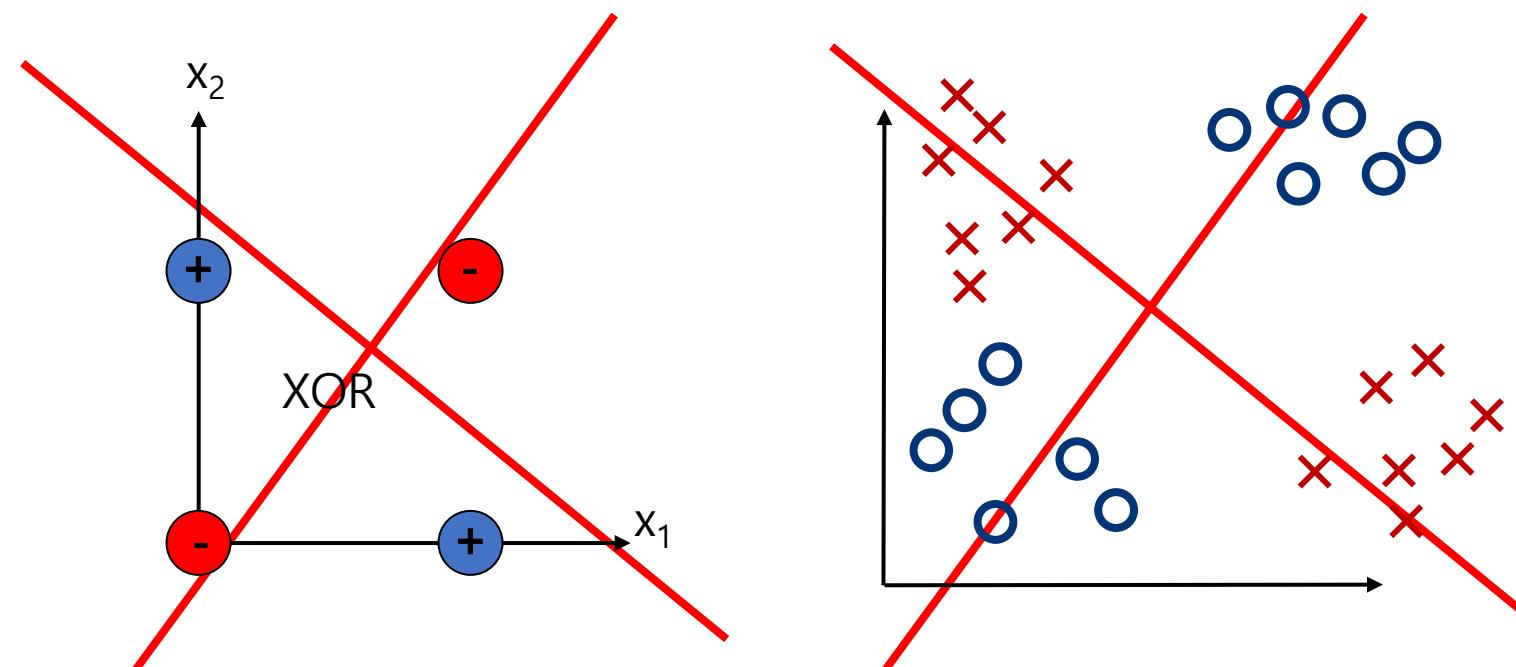
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



3. 퍼셉트론의 한계

선형분류기로 (Linear separable function)로만 동작

- Minsky & Papert (1969): Perceptrons can only represent linearly separable functions



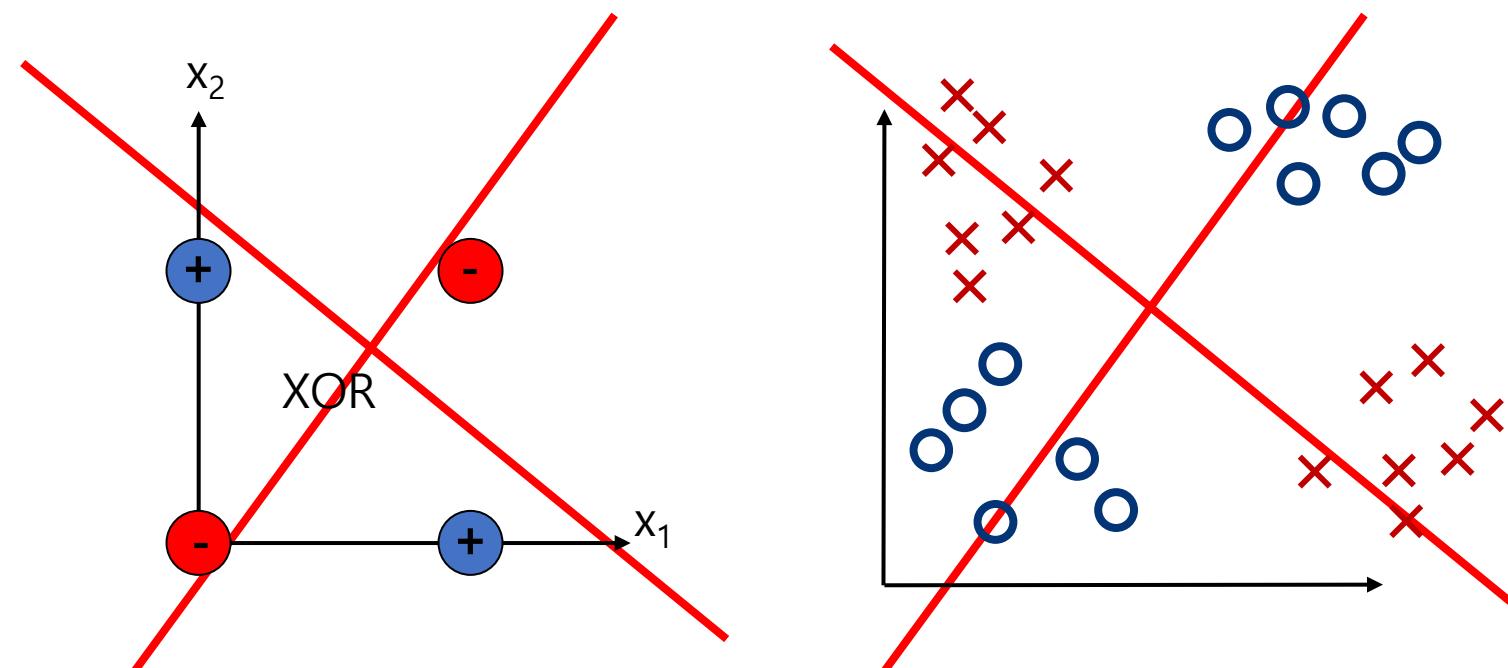
XOR problem
Not linearly separable



3. 퍼셉트론의 한계

선형분류기로 (Linear separable function)로만 동작

- Minsky & Papert (1969): Perceptrons can only represent linearly separable functions



XOR problem
Not linearly separable



3. 퍼셉트론의 한계

■ Minsky & Papert (1969)

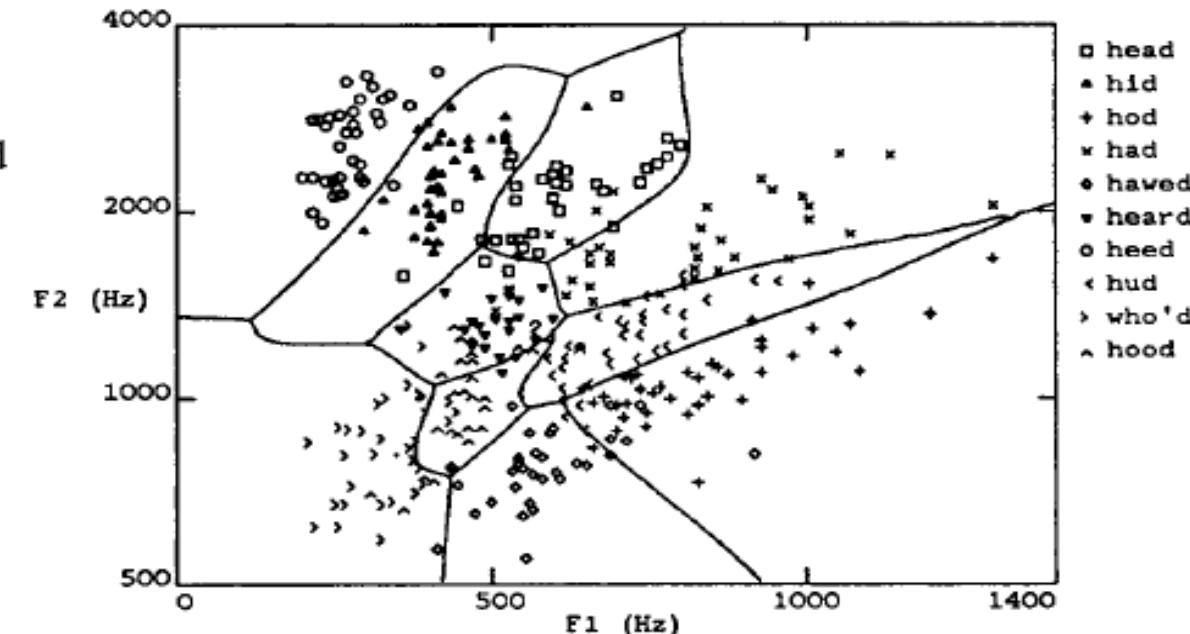
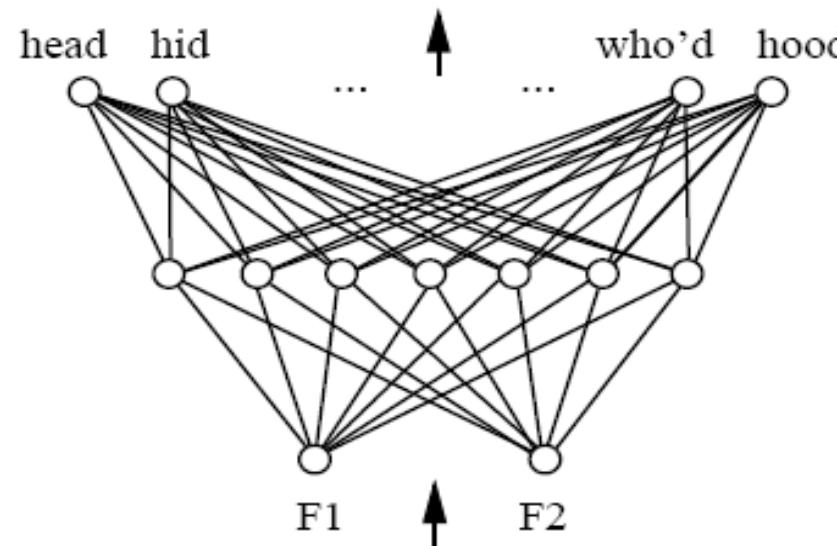
- ✓ 퍼셉트론은 선형분류 함수만을 표현할 수 있음
- ✓ 은닉층을 추가함으로써 더 복잡한 목표 함수를 표현할 수 있음을 증명



3. 퍼셉트론의 한계

Minsky & Papert (1969)

- ✓ 퍼셉트론은 선형분류 함수만을 표현할 수 있음
- ✓ 은닉층을 추가함으로써 더 복잡한 목표 함수를 표현할 수 있음을 증명





4. 다층퍼셉트론 (Multi-layer perceptron)

MLP의 필요성

Structure	Regions	XOR	Meshed Regions
Single layer	Halfplane bounded by hyperplane		
Two layers	Convex Open or Closed regions		
Three layers	Arbitrary (limired by # of nodes)		

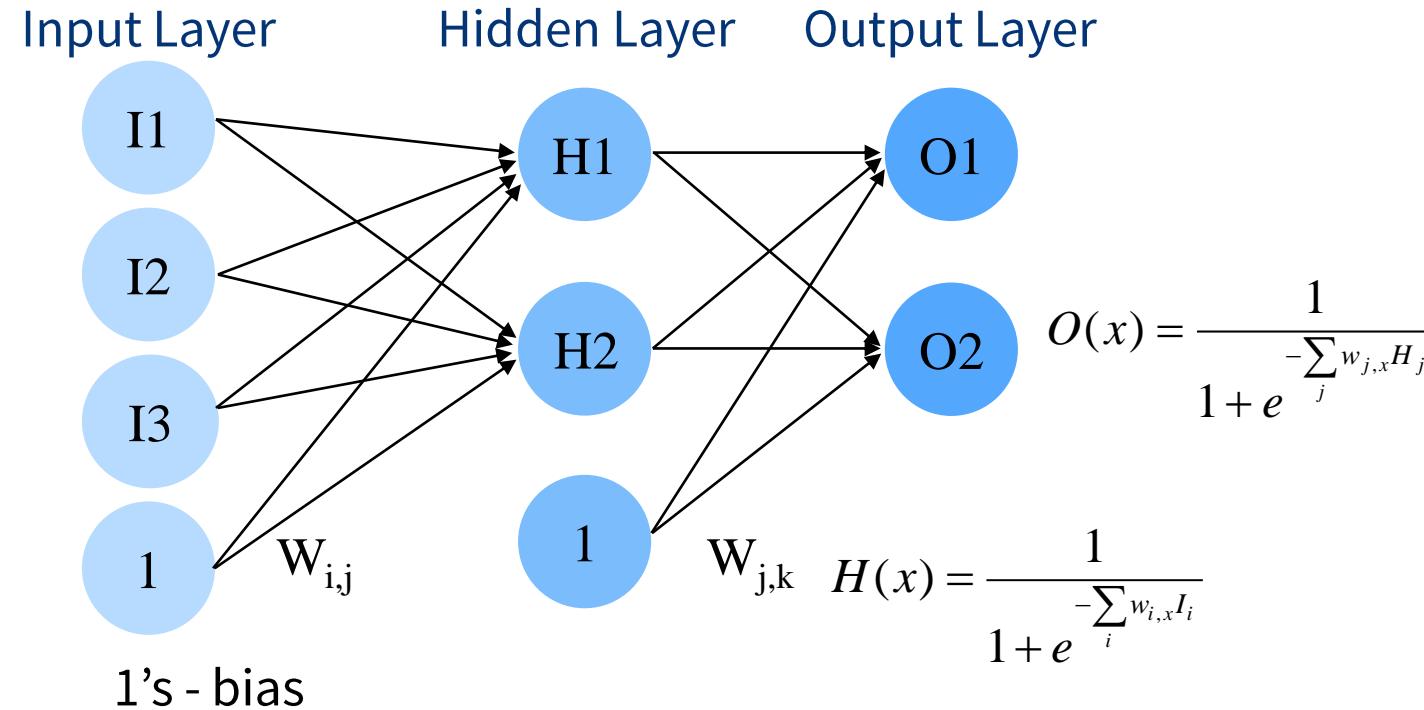
Multiple boundaries needed
(e.g. XOR problem)
→**Multiple units**

More complex regions needed
(e.g. polygons)
→**Multiple layers**

4. 다층퍼셉트론 (Multi-layer perceptron)

▣ 퍼셉트론

- ✓ 선형분류기로서의 한계를 극복하기 위해 여러개의 층을 가진 신경망을 고안
(Rumelhart and McClelland, late 70's)
- ✓ Fully connected, feedforward network





5. Hidden Units

■ Hidden units

- ✓ 입력노드와 출력 노드 사이에 위치
- ✓ 비선형 함수 (non-linear function) 의 학습을 가능하게 함
- ✓ 입력 특징들의 조합을 표현



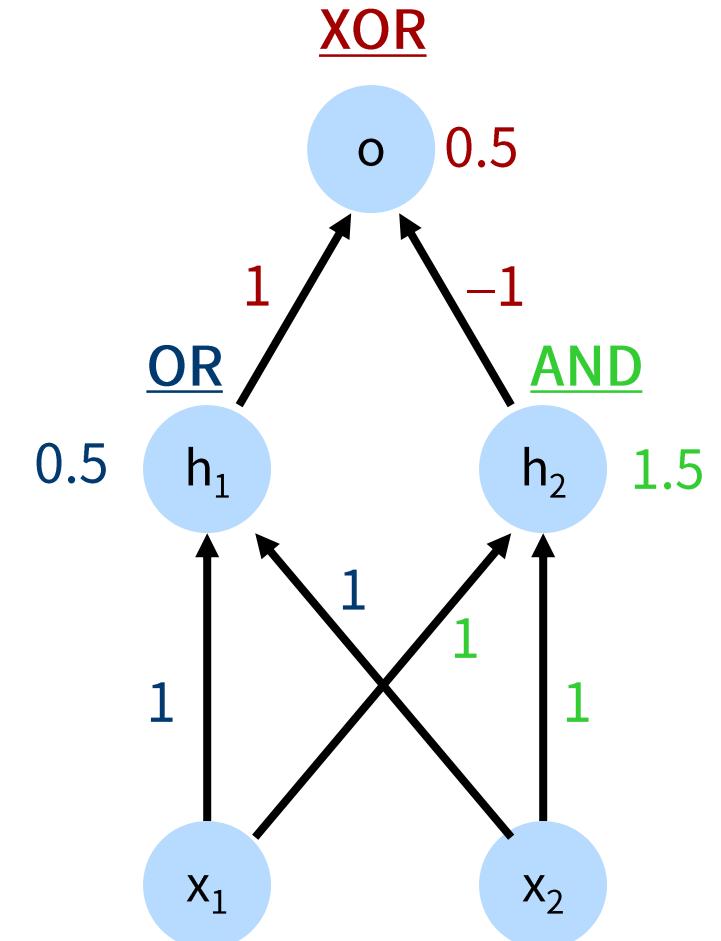
5. Hidden Units

▣ Boolean XOR

$$X_1 \oplus X_2 \Leftrightarrow (X_1 \vee X_2) \wedge \neg(X_1 \wedge X_2)$$

- ✓ 하나의 single layer perceptron으로는 표현할 수 없음
- ✓ Hidden layer를 추가하여, 함수의 building block으로 사용

$$w_1 X_1 + w_2 X_2 - w_0 > 0$$



정리하기

- ▶ 퍼셉트론 학습: 오차제곱합의 점진적 감소를 위해 gradient descent 사용
- ▶ 선형분류함수만을 생성, non-linear classification에 적용되기에 한계가 있음
- ▶ 은닉층을 추가함으로써 해결 → Next class

들어가기

| 학습목표 |

- MLP의 학습 알고리즘인 오류역전파 (back-propagation)을 이해한다.
- 은닉층 (hidden layer)과 은닉노드 (hidden node)를 통한 MLP의 expressive power를 설명할 수 있다.
- 다층 퍼셉트론을 분류기로 활용할 수 있다.

들어가기

| 학습내용 |

- 생물학적 신경망과 인공신경망의 특성
- 신경망의 역사
- 퍼셉트론의 구조와 원리

들어가기

| 학습내용 |

- 퍼셉트론의 학습
- 퍼셉트론의 문제점

들어가기

| 학습목차 |

- 1 | 생물학적 신경망 vs. 인공신경망
- 2 | 신경망의 역사
- 3 | 퍼셉트론의 구조와 원리 및 학습
- 4 | 다층 퍼셉트론
- 5 | 오류역전파 학습알고리즘 (Backpropagation learning algorithm)
- 6 | 신경망 설계 (NN design)
- 7 | Convolutional NN의 핵심 원리



1. 신경망 (Neural Network)

▣ 사람의 두뇌

- ✓ 뉴런으로 구성 (약 1011개, 약 1014 연결 (시냅스))
- ✓ 고도의 복잡한 명령어 처리기

▣ 인공 신경망 (ANN: Artificial Neural Network)

- ✓ 뇌의 정보처리를 모방하여 인간에 필적하는 지능 컴퓨터에 도전

▣ 컴퓨터 vs. 사람의 두뇌



(a) 폰 노이만 컴퓨터의 구조



(b) 사람 뇌의 정보 처리 단위인 뉴런



1. 신경망 (Neural Network)

인공 신경망 vs. 생물학적 신경망



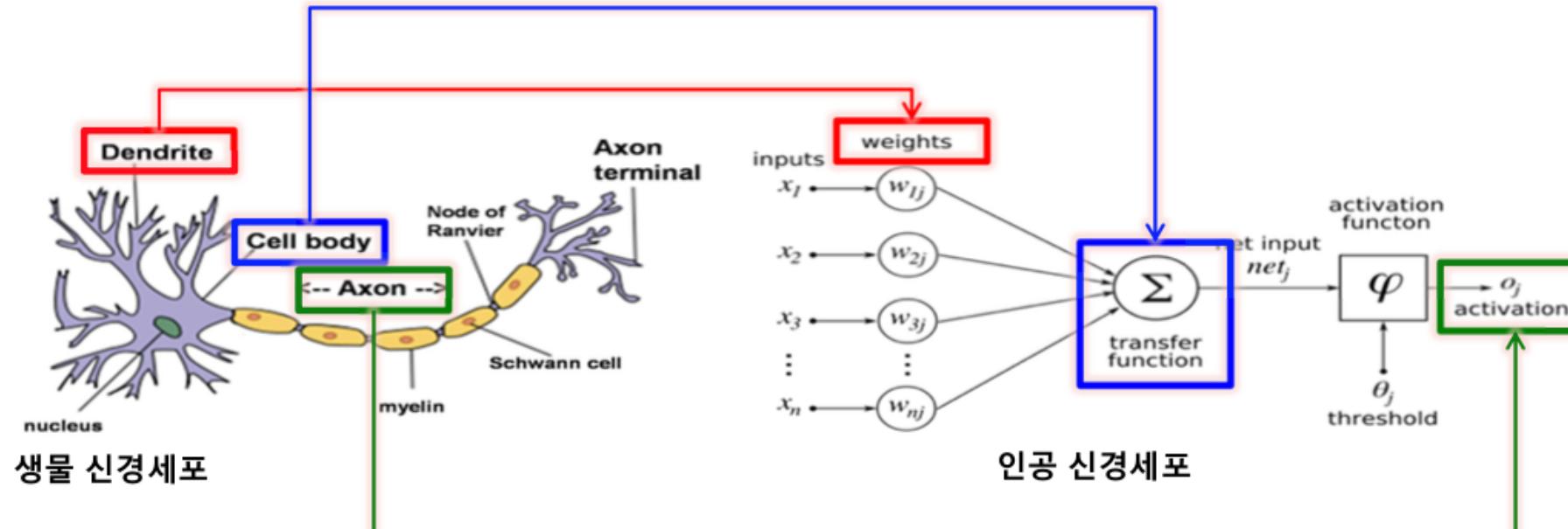
- 정확한 소자 (논리 소자)
- 소자 속도 매우 빠름 (10^{-9} 초)
- 디지털 회로망 (전자 회로망)
- 주소기반 메모리 (국지적/독립적)
- 논리/산술적 연산 (조작적)
- 중앙집중식 순차 처리
- 프로그래밍기반 명시적 지식



- 부정확한 소자 (10^{11} 개 뉴런)
- 소자 속도 느림 (10^{-3} 초)
- 아날로그 회로망 (10^{14} 개 연결선)
- 내용기반 메모리 (전역적/관계적)
- 패턴/영상기반 연산 (연관적)
- 분산적 병렬 처리
- 학습(경험/데이터)기반 암묵적 지식



1. 신경망 (Neural Network)

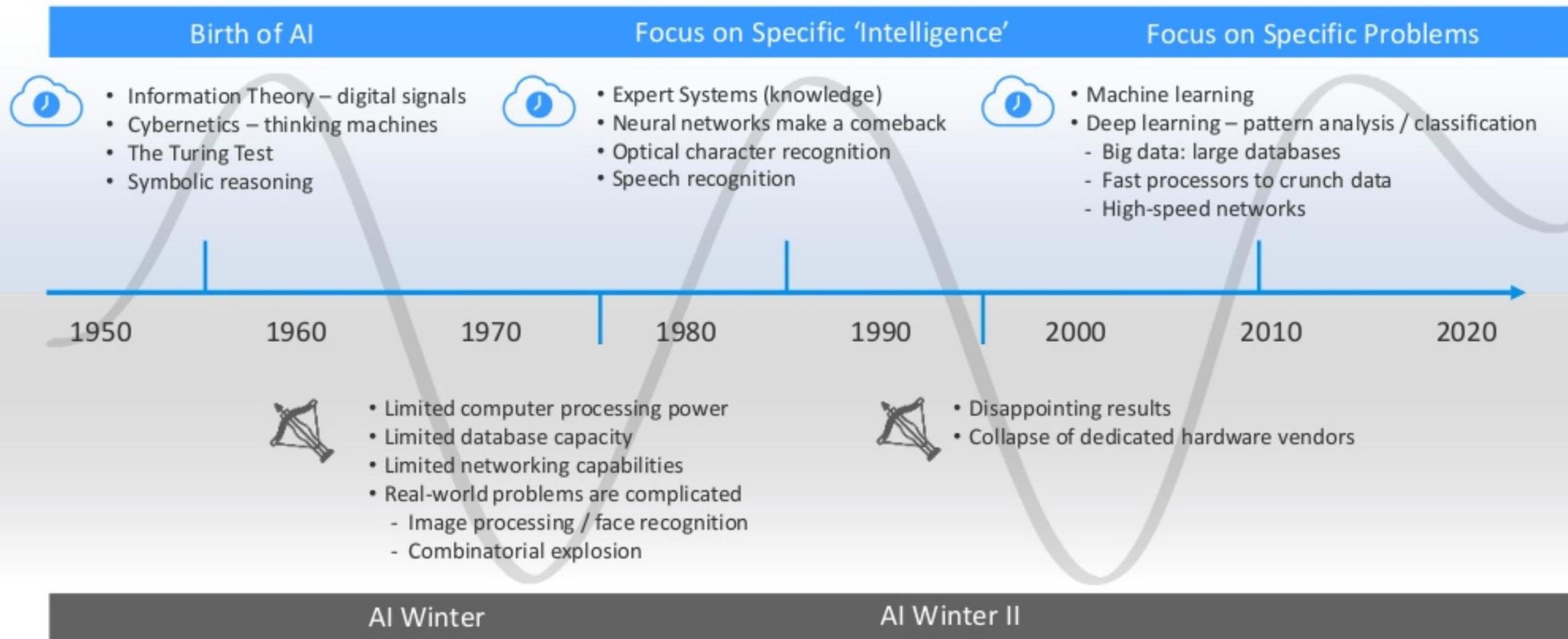


인공 신경세포 (Artificial Neuron)



1. 신경망 (Neural Network)

An AI Timeline





1. 신경망 (Neural Network)

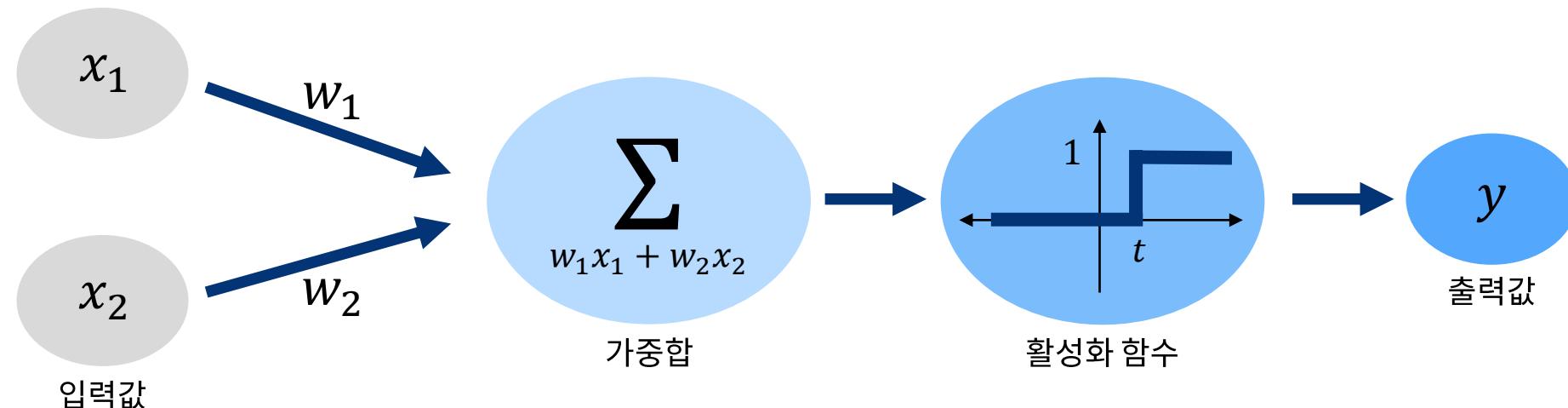
■ 신경망의 역사

- ✓ 1943, McCulloch과 Pitts 최초 신경망 제안
- ✓ 1949, Hebb의 학습 알고리즘
- ✓ 1958, Rosenblatt 퍼셉트론
- ✓ Widrow와 Hoff, Adaline과 Madaline
- ✓ 1960대, 신경망의 과대 포장
- ✓ 1969, Minsky와 Papert, Perceptrons라는 저서에서 퍼셉트론 한계 지적
- ✓ 1986, Rumelhart, Hinton, 그리고 Williams, 다층 퍼셉트론과 오류 역전파 학습 알고리즘
- ✓ 2010, 딥러닝 Deep Learning 알고리즘 출현

2. 퍼셉트론 (Perceptron)

퍼셉트론

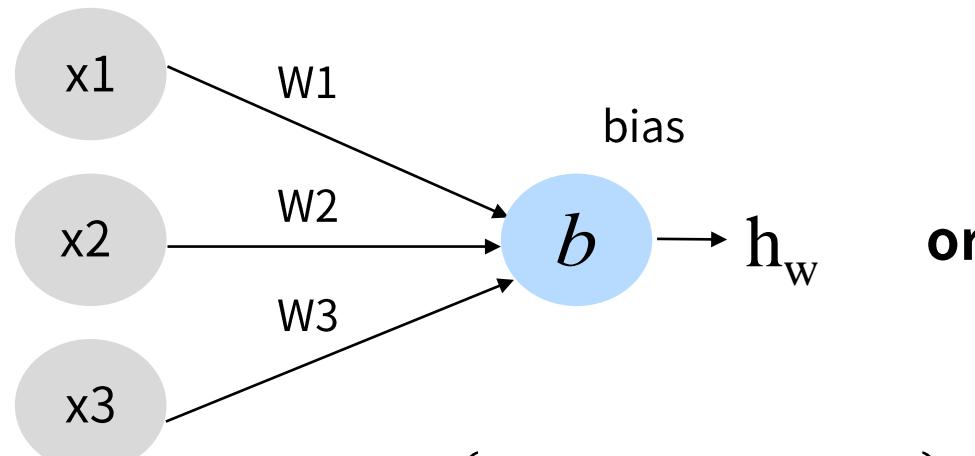
- ✓ 생물학적 신경세포의 작동원리를 수학적으로 모델링한 인공 뉴런
- ✓ 뇌의 구조를 모사한 정보처리 방식
 - 1958년 Rosenblatt가 학습의 가능함을 증명



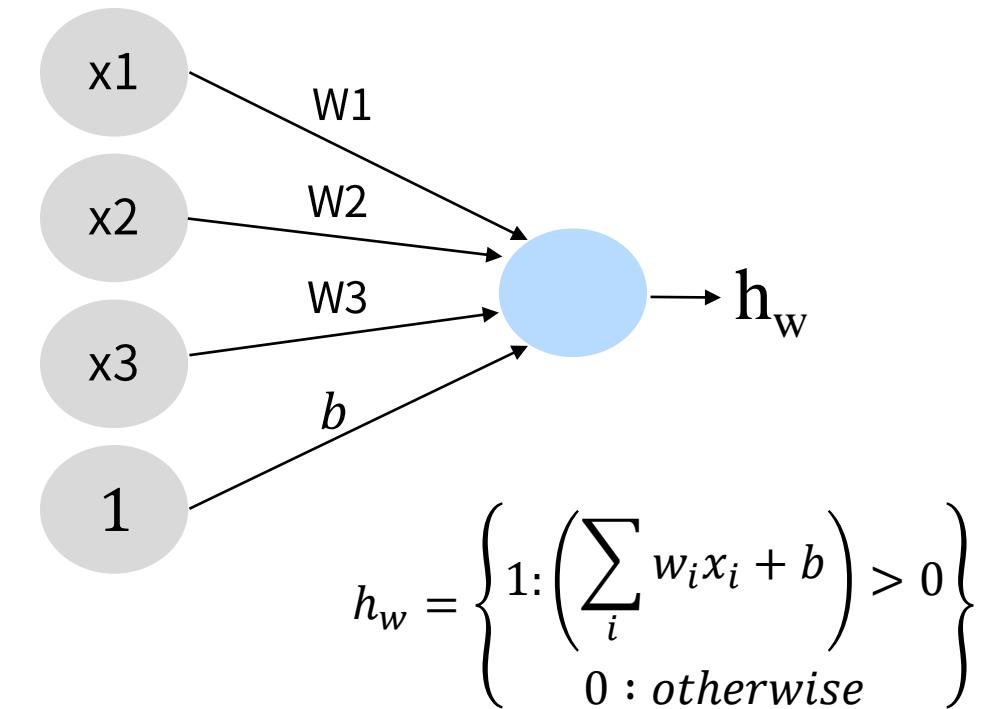
2. 퍼셉트론 (Perceptron)

▣ 구조와 원리

- ✓ 입력층: $d+1$ 개의 노드 (특징벡터 X)
- ✓ 출력층: 한 개의 노드 (따라서 2-부류 분류기)
- ✓ 에지와 가중치

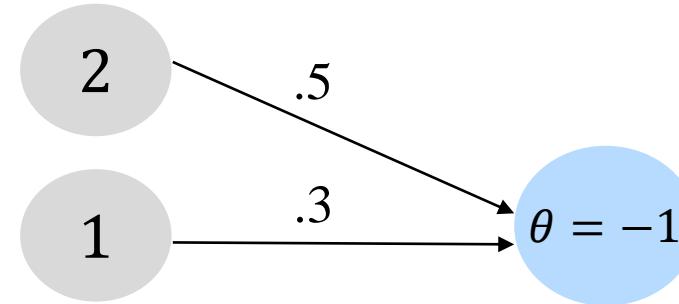


$$\text{Hypothesis } h_w = \begin{cases} 1: \left(\sum_i w_i x_i \right) + b > 0 \\ 0 : \text{otherwise} \end{cases}$$



$$h_w = \begin{cases} 1: \left(\sum_i w_i x_i + b \right) > 0 \\ 0 : \text{otherwise} \end{cases}$$

■ 예제



$$2(0.5) + 1(0.3) + -1 = 0.3 > 0, \text{ so } h_w = 1$$

■ Learning Procedure:

- ✓ Randomly assign weights (between 0 and 1)
- ✓ Present inputs from training data
- ✓ Get output h_w , nudge weights to give results toward our desired output T
- ✓ Repeat; stop when no errors, or enough epochs completed

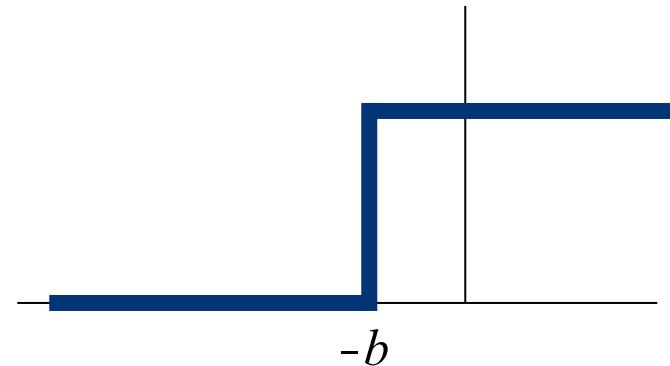


2. 퍼셉트론 (Perceptron)

■ Perceptron with Activation Function: sigmoid, Relu, Leaky Relu

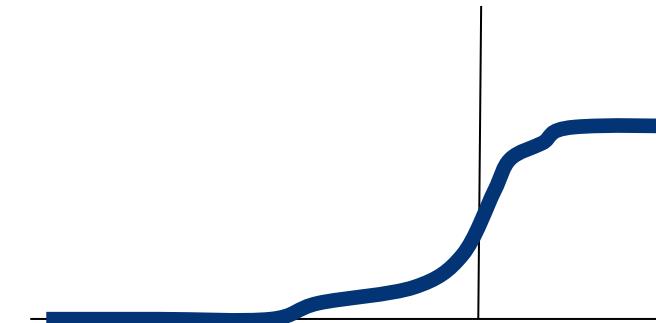
✓ Old

$$h_w = \begin{cases} 1: \sum_j w_j x_j + b > 0 \\ 0 : \text{otherwise} \end{cases}$$



✓ New

$$h_w = \frac{1}{1 + e^{-\sum_j w_j \times x_j + b}}$$



Activation Function: g

Perceptron is essentially linear classifier

$$h_w = \begin{cases} 1: \left(\sum_i w_i x_i + b \right) > 0 \\ 0 : \text{otherwise} \end{cases}$$



2. 퍼셉트론 (Perceptron)

■ 선형 분류기 (Linear classifier)

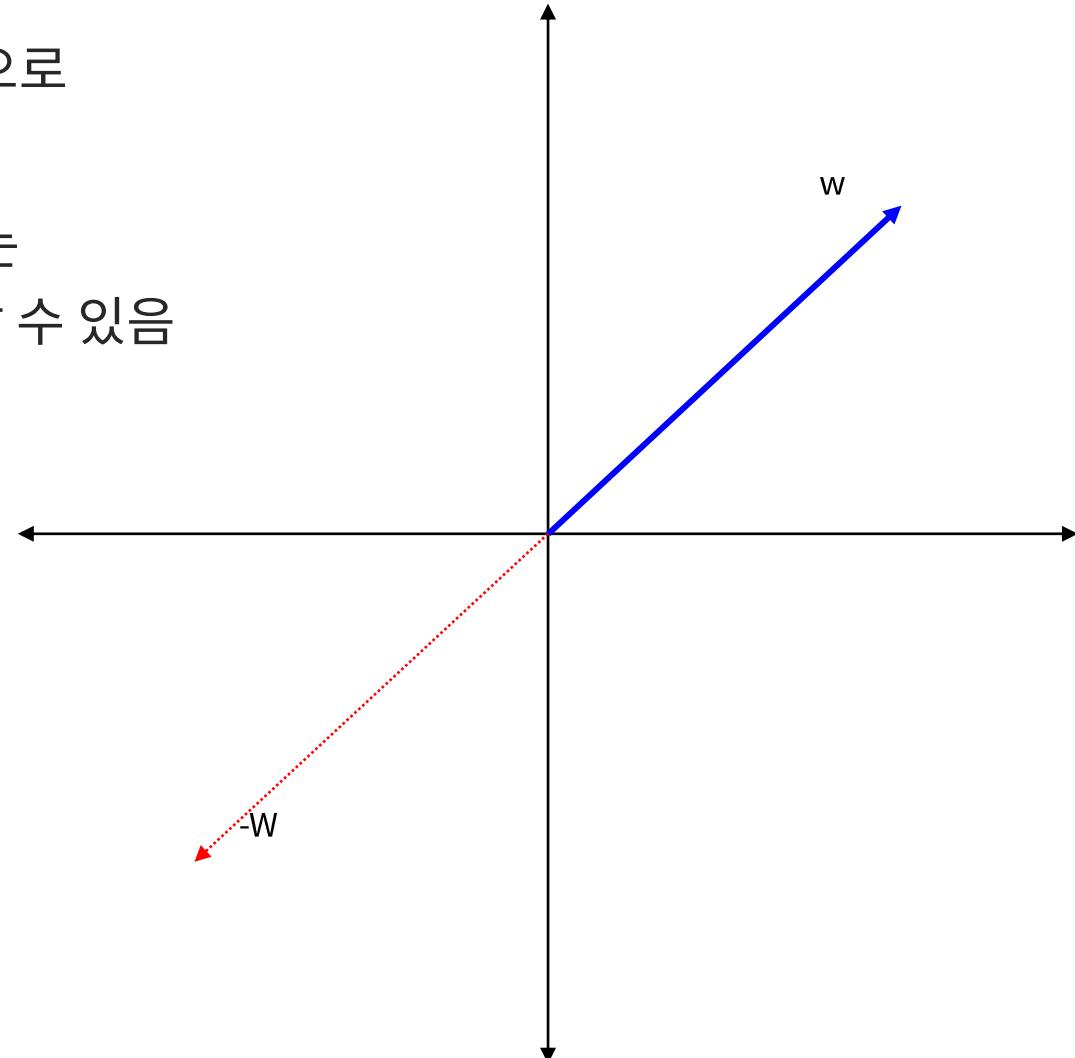
- A linear classifier is vector \mathbf{w} of the same dimension as \mathbf{x} that is used to make this prediction

$$\hat{y} = \text{sign}(w_1x_1 + w_2x_2 + \dots + w_nx_n) = \text{sign}(\mathbf{w} \cdot \mathbf{x})$$

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

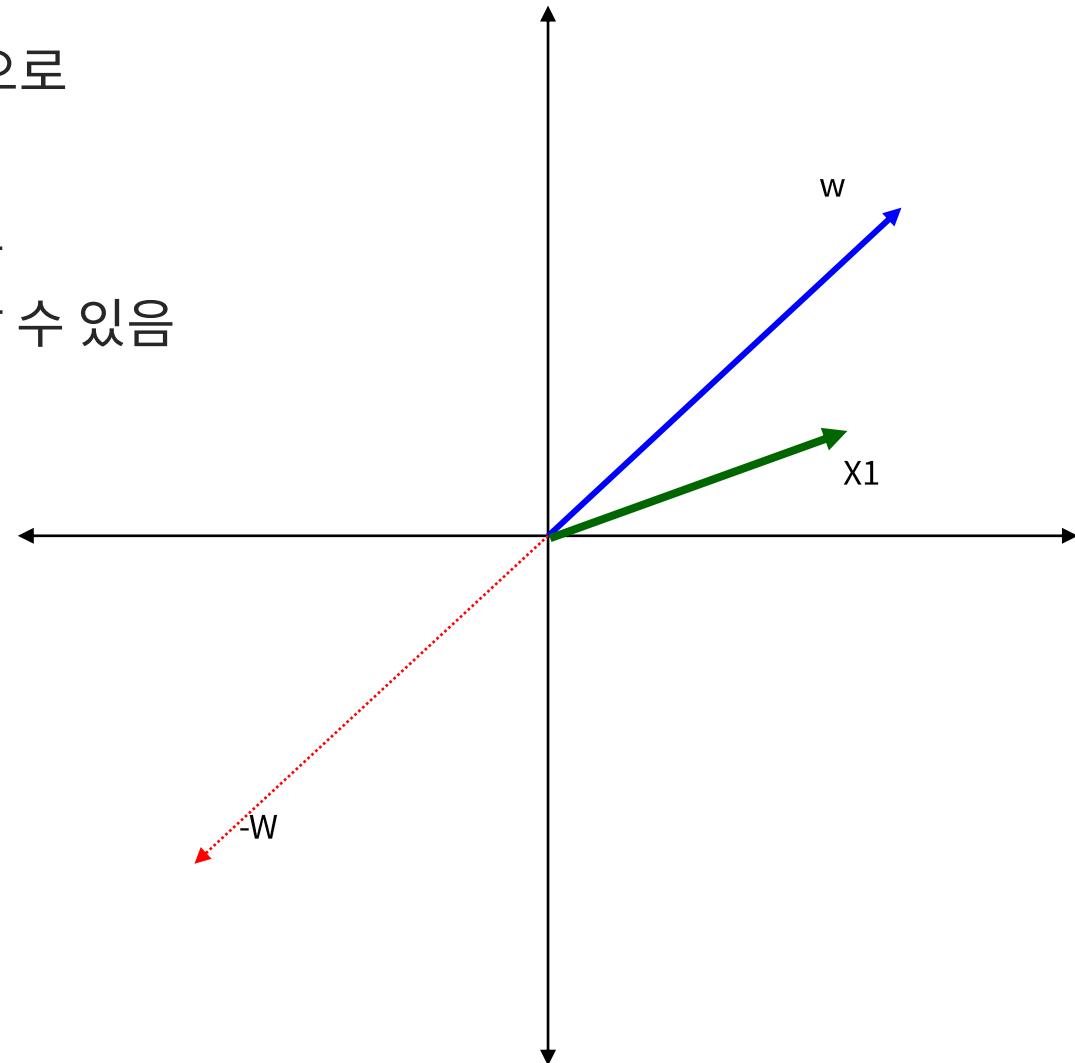
선형 분류기 (Linear classifier)

- ✓ 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- ✓ 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane



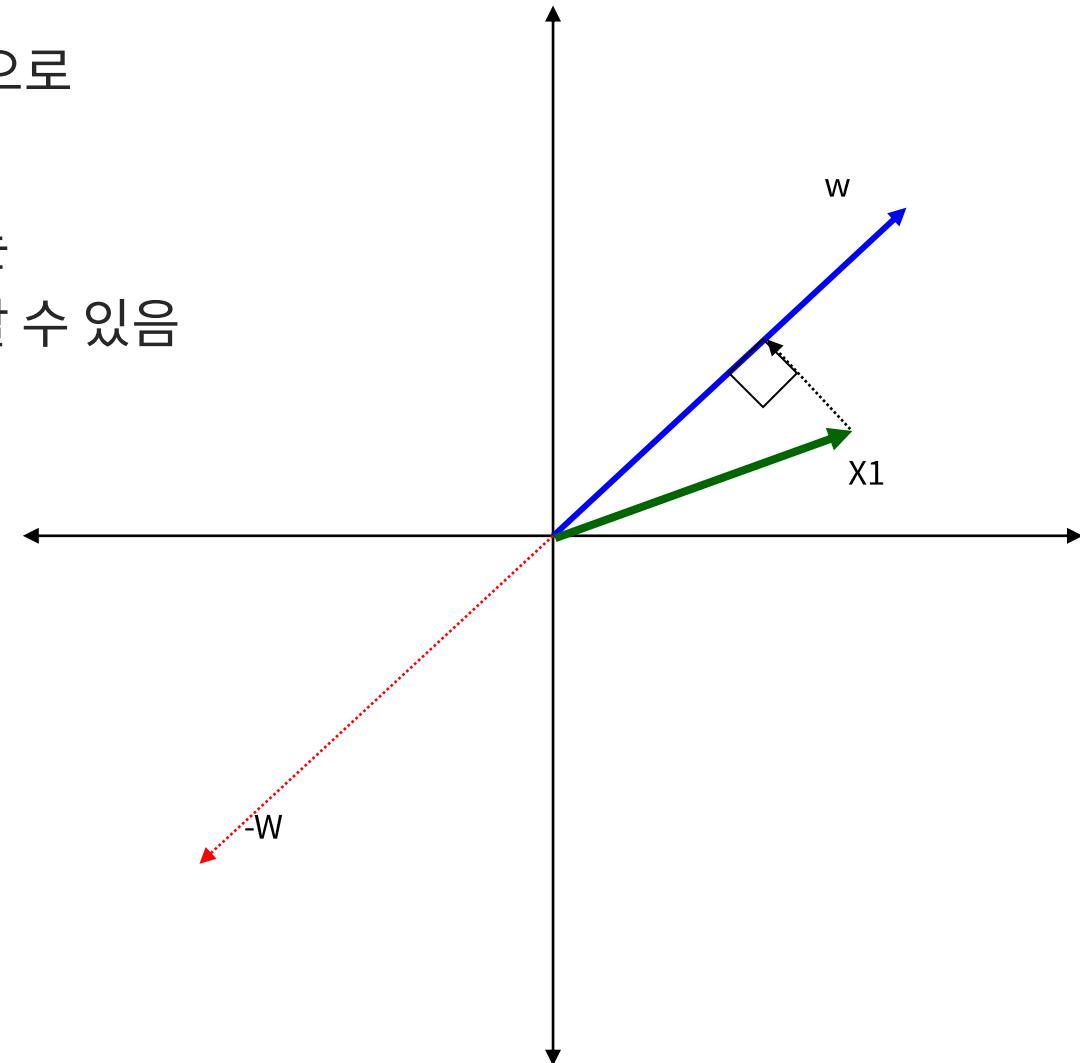
선형 분류기 (Linear classifier)

- ✓ 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- ✓ 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane



선형 분류기 (Linear classifier)

- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane

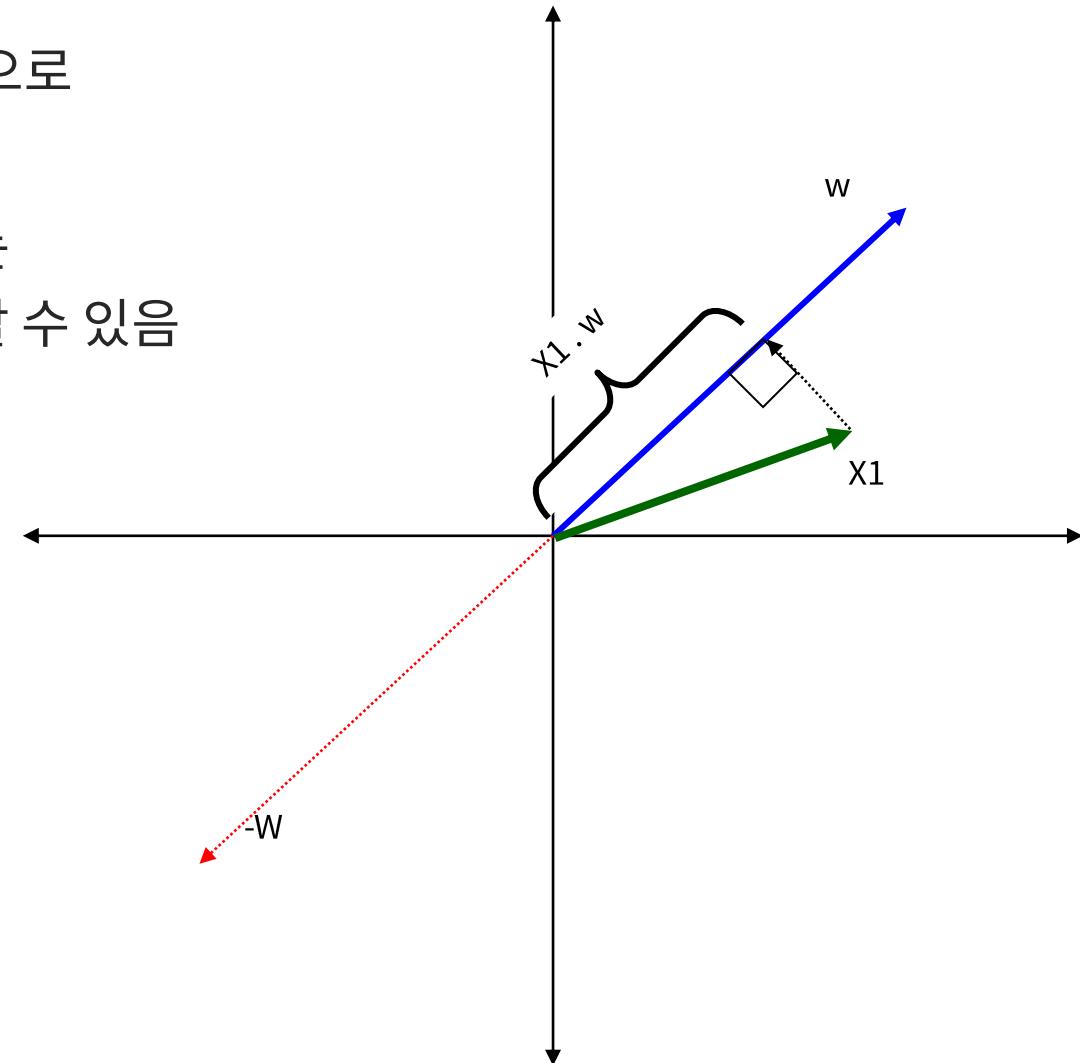




2. 퍼셉트론 (Perceptron)

선형 분류기 (Linear classifier)

- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane

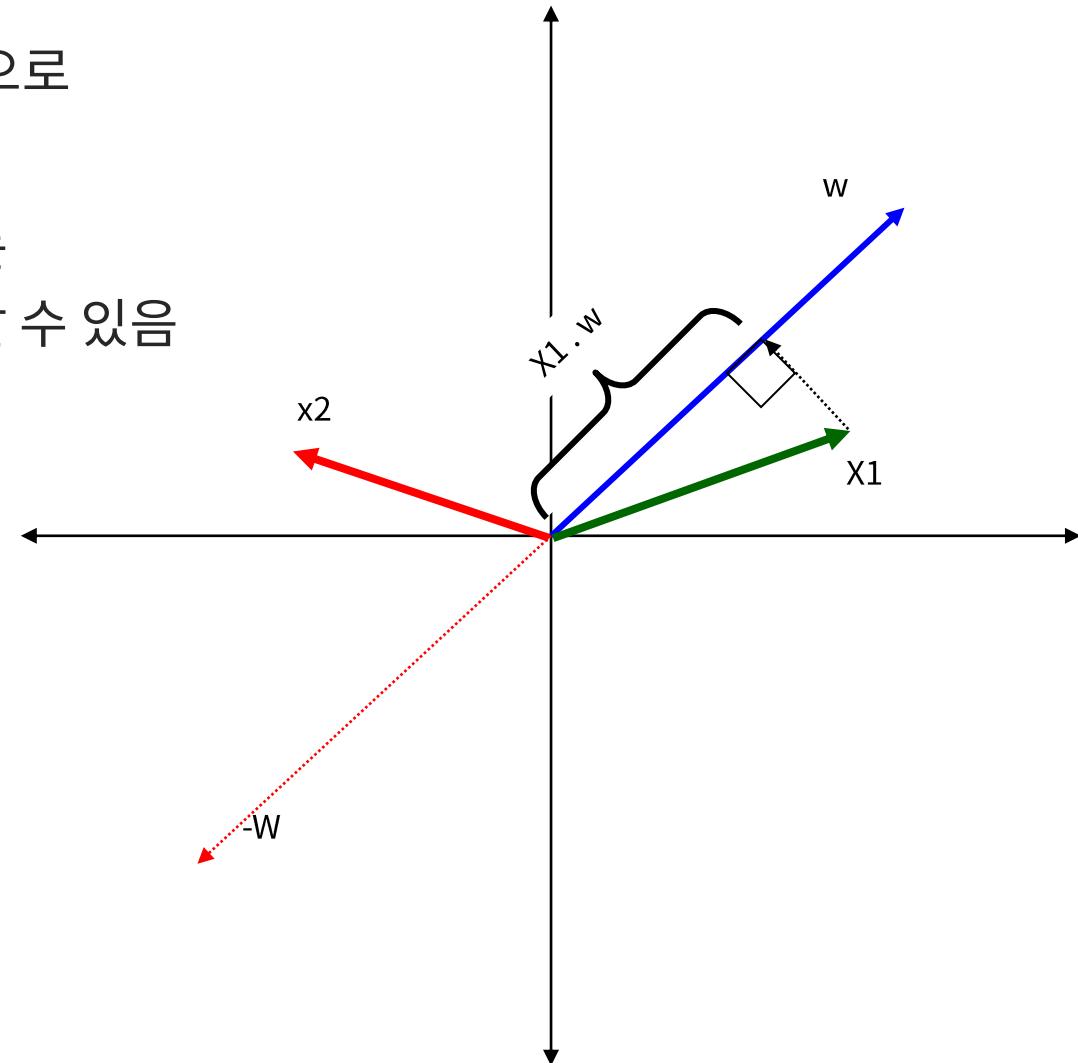




2. 퍼셉트론 (Perceptron)

선형 분류기 (Linear classifier)

- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane

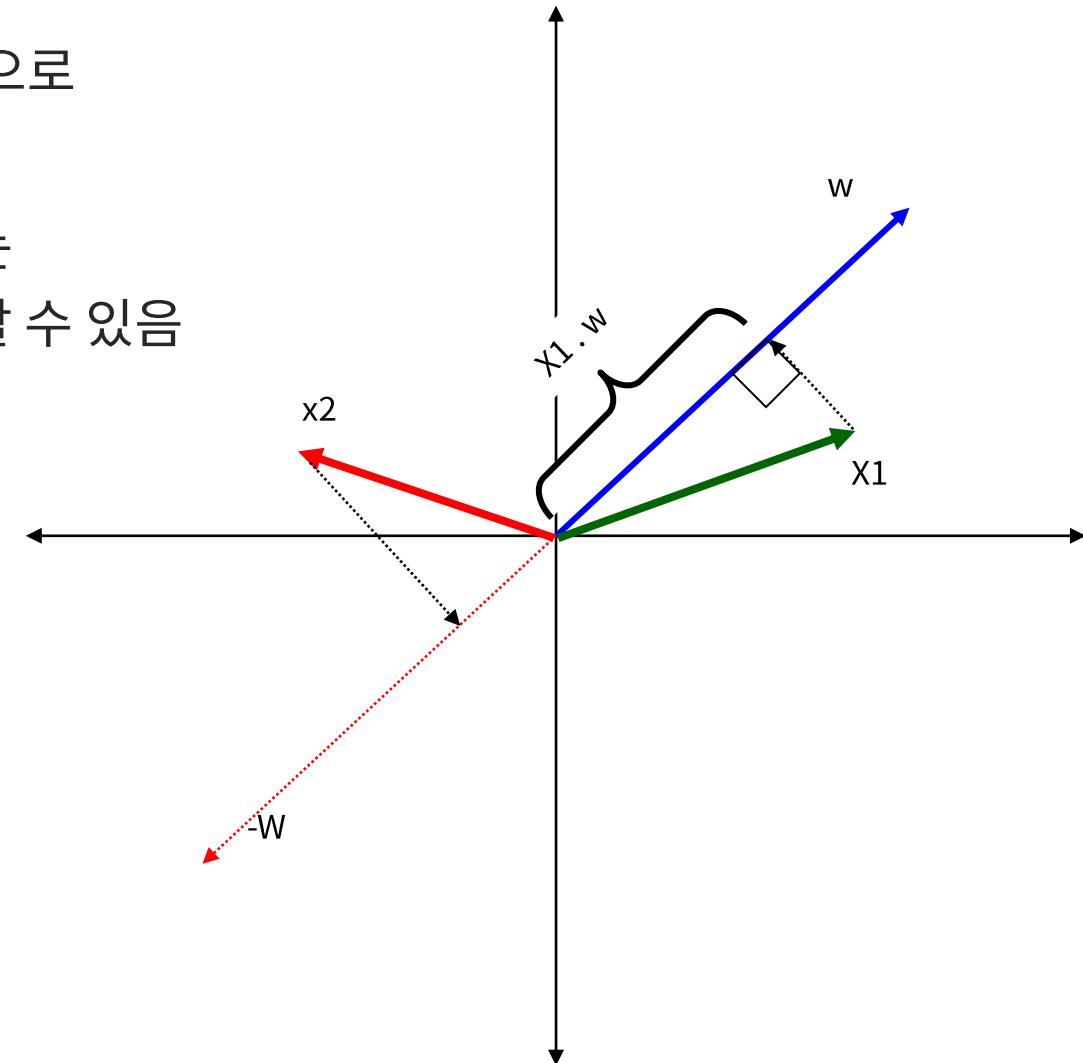




2. 퍼셉트론 (Perceptron)

선형 분류기 (Linear classifier)

- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane

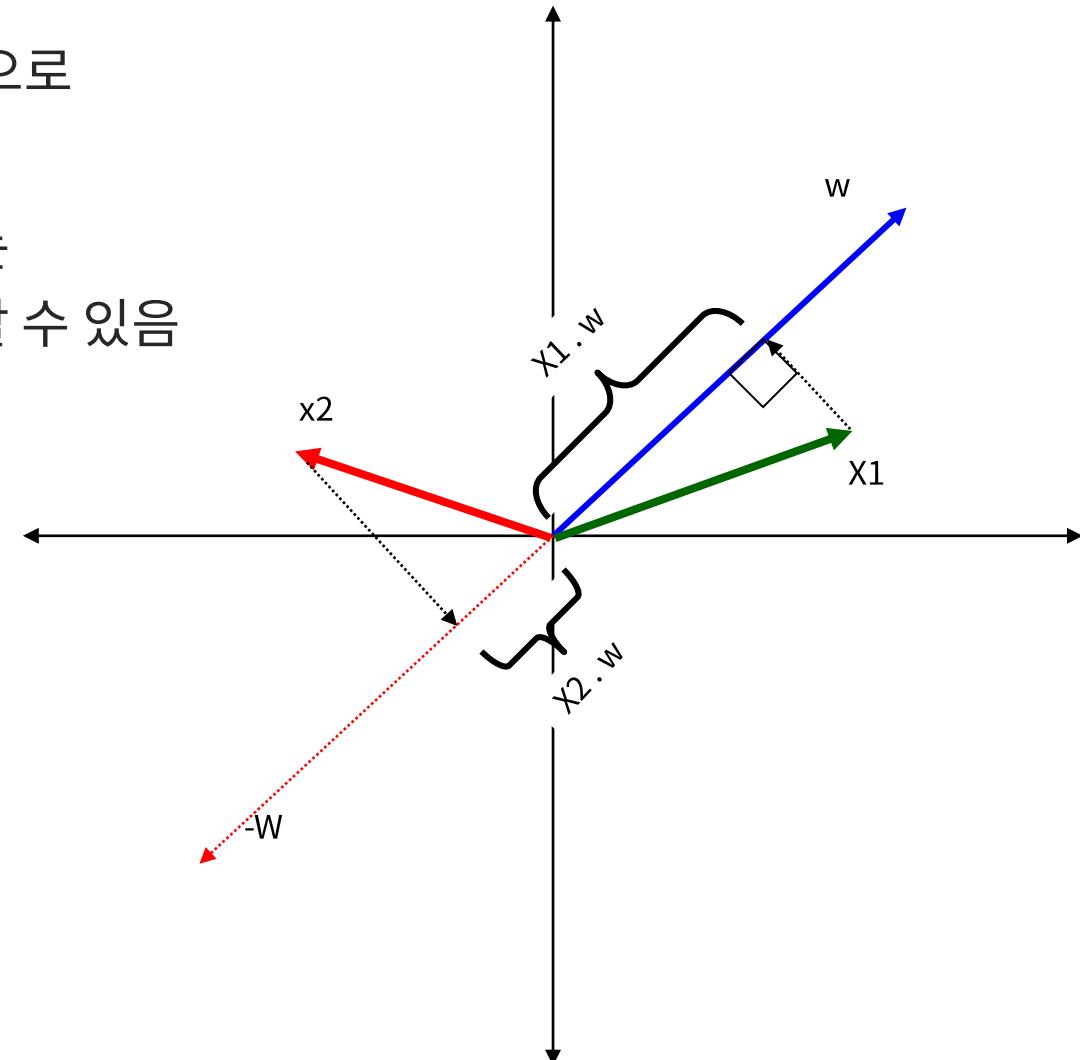




2. 퍼셉트론 (Perceptron)

선형 분류기 (Linear classifier)

- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane

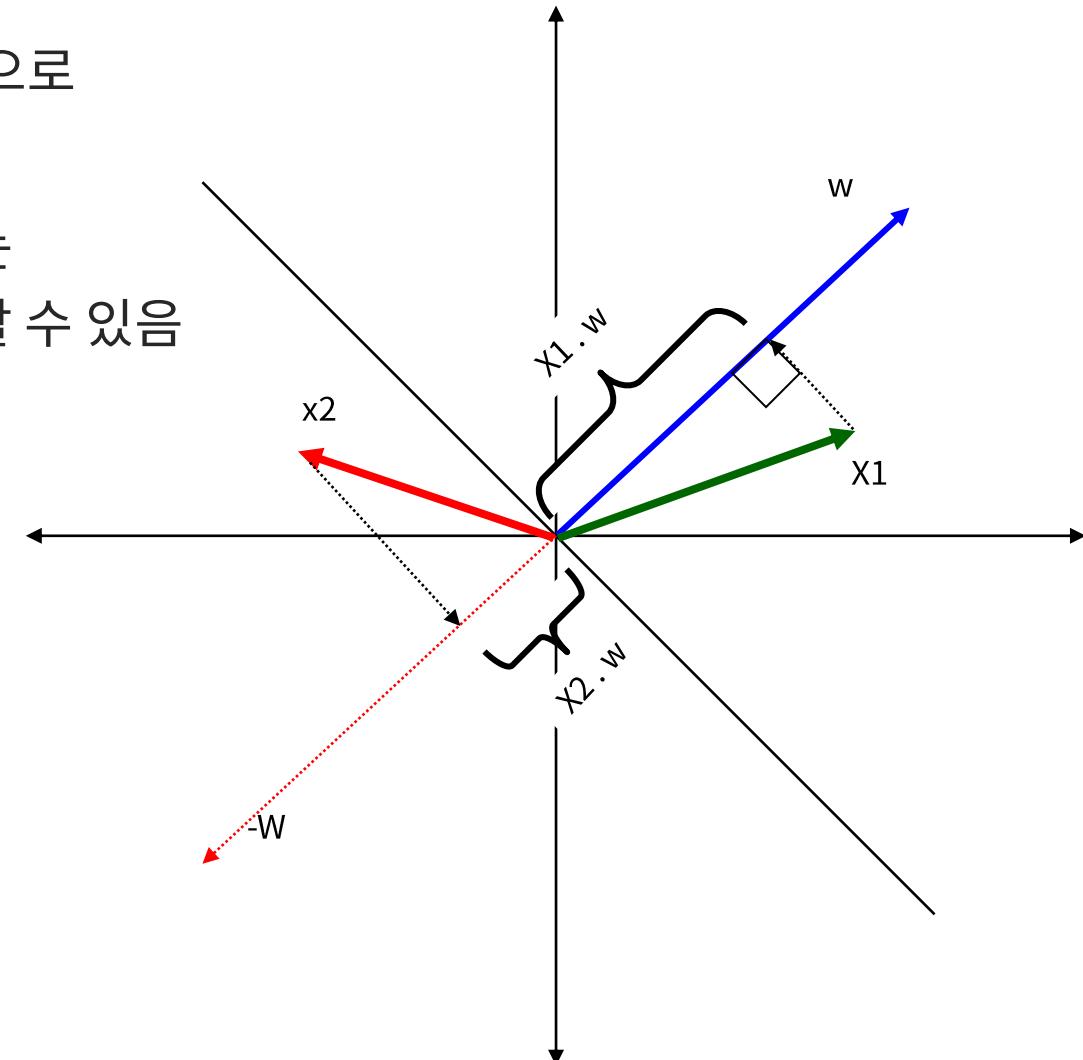




2. 퍼셉트론 (Perceptron)

선형 분류기 (Linear classifier)

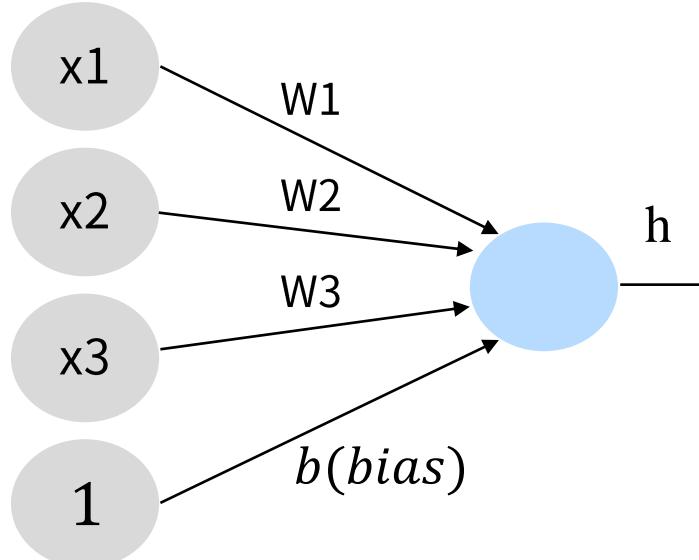
- 특징 공간 상에서, $x \cdot w$ 은 벡터 x 를 벡터 y 에 수직으로 투영(projection)하였을 때 벡터의 길이를 표현
- 벡터 w 에 수직인 직선은 양수 (positive)로 분류되는 벡터와 음수(negative)로 분류되는 벡터들을 분리할 수 있음
 - In 3d: line \rightarrow plane
 - In 4d: plane \rightarrow hyperplane



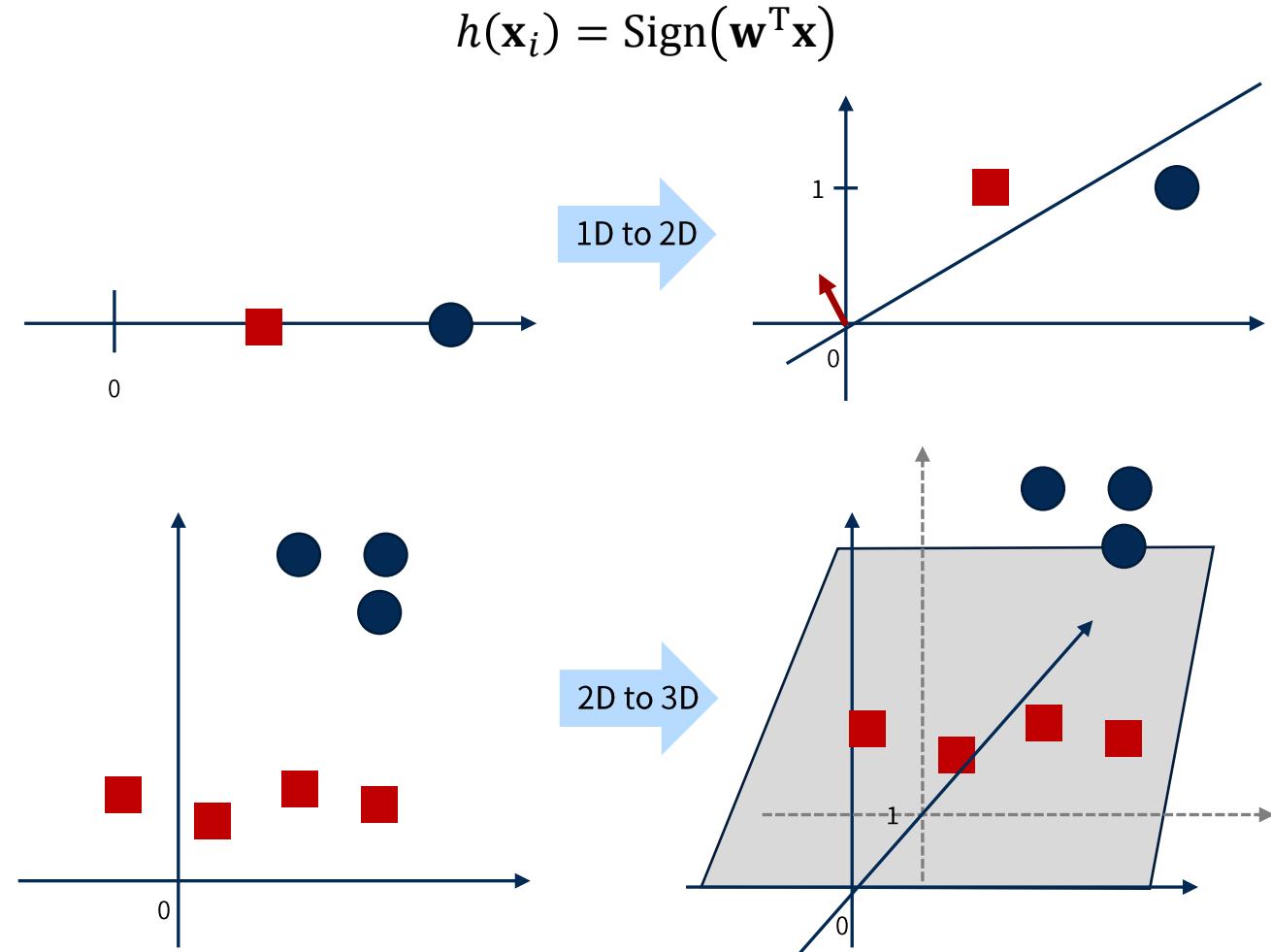


2. 퍼셉트론 (Perceptron)

■ 퍼셉트론은 선형분류기



$$h_w = \begin{cases} 1: g\left(\sum_i w_i x_i + b\right) > 0 \\ 0 : otherwise \end{cases}$$



정리하기

▶ 신경망의 최소단위인 퍼셉트론의 구조

▶ 퍼셉트론의 동작 원리

▶ 퍼셉트론은 선형분류기로 동작

출처

[출처01] gettyimagesbank 1305746141

[출처02] gettyimagesbank 1130604171

[출처03] <https://www.slideshare.net>

들어가기

| 학습목표 |

- 이론으로 배운 로지스틱 회귀 분석을 실습을 통해 구현하는 방법을 학습하고, 현실 데이터를 사용하여 선형 회귀 모델을 만들어 본다.

들어가기

| 학습내용 |

- 데이터셋

유방암 데이터 셋(Breast Cancer Wisconsin Database)을 불러오고,
트레이닝 데이터와 테스트 데이터로 분할하는 법을 학습한다.

- 로지스틱 모델 정의

로지스틱 회귀 모델, 교차 엔트로피, 그리고 함수 편미분의 구현 방법을 학습한다.

- 로지스틱 모델 학습

경사 하강법을 통해 로지스틱 모델을 학습하고 혼돈 행렬과 정확도를 통해 모델을 분석한다.

- 로지스틱 간단한 구현법

sklearn 모듈을 사용하여 간단하게 로지스틱 모델을 구현하는 방법을 학습한다.



1. 데이터셋

필요한 모듈 불러오기

- 1 numpy 모듈을 np라는 이름으로 불러오기
- 2 sklearn.datasets에서 load_breast_cancer 모듈 불러오기

```
1 import numpy as np  
2 from sklearn.datasets import load_breast_cancer
```

유방암 데이터셋 정의 (Breast Cancer Wisconsin Database)

- 1 load_breast_cancer()로 cancer변수에 유방암 데이터셋 할당
- 2 cancer은 dictionary형태로 저장되어 있기에 key값들을 출력하기

```
1 cancer = load_breast_cancer()    # dictionary  
2 print(cancer.keys())  
  
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR', 'feature_names', 'filename'])
```



1. 데이터셋

데이터셋 속성 확인

1

cancer중 feature_names에 해당하는 value값을 출력

2

cancer중 feature_names에 해당하는 value값의 개수 출력

- ✓ 총 30개의 속성 값이 존재

1

cancer중 target_names에 해당하는 value값 출력

- ✓ malignant(악성,0), benign(양성,1)

```
1 print(cancer['feature_names'])
2 print(len(cancer['feature_names']))
3 print(cancer['target_names']) # malignant(악성,0), benign(양성,1)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
```



1. 데이터셋

입력 데이터와 타깃 데이터

1

입력 데이터의 형태 출력

2

타깃 데이터의 형태 출력

- ✓ 입력 데이터는 총 569개가 있고 각 데이터 중 30개 요소 값이 있다.
- ✓ 타깃 데이터는 총 569개가 있다.

```
1 print('data shape:', cancer['data'].shape)
2 print('target shape:', cancer['target'].shape)
```

data shape: (569, 30)
target shape: (569,)



1. 데이터셋

입력 데이터와 타깃 데이터 출력

1

마지막 두 개 입력 데이터 출력

2

마지막 두 개 타깃 데이터 출력

- ✓ 입력 데이터는 각 30개 요소에 해당하는 값들이 출력 됨
- ✓ 타깃 데이터는 해당 30개 요소일때 악성(0)/양성(1) 값이 출력 됨

```
1 print(cancer['data'][-2:])
2 print(cancer['target'][-2:])
```

```
[2.060e+01 2.933e+01 1.401e+02 1.265e+03 1.178e-01 2.770e-01 3.514e-01
 1.520e-01 2.397e-01 7.016e-02 7.260e-01 1.595e+00 5.772e+00 8.622e+01
 6.522e-03 6.158e-02 7.117e-02 1.664e-02 2.324e-02 6.185e-03 2.574e+01
 3.942e+01 1.846e+02 1.821e+03 1.650e-01 8.681e-01 9.387e-01 2.650e-01
 4.087e-01 1.240e-01]
[7.760e+00 2.454e+01 4.792e+01 1.810e+02 5.263e-02 4.362e-02 0.000e+00
 0.000e+00 1.587e-01 5.884e-02 3.857e-01 1.428e+00 2.548e+00 1.915e+01
 7.189e-03 4.660e-03 0.000e+00 0.000e+00 2.676e-02 2.783e-03 9.456e+00
 3.037e+01 5.916e+01 2.686e+02 8.996e-02 6.444e-02 0.000e+00 0.000e+00
 2.871e-01 7.039e-02]
[0 1]
```



1. 데이터셋

데이터셋 섞기

1 | 2

x에 입력 데이터, y에 타깃 데이터 할당

4

0~568 인덱스가 있는 리스트 만들기

5

인덱스 리스트를 랜덤으로 섞기

6 | 7

입력 데이터와 타깃 데이터를 랜덤으로 섞인 인덱스순으로 섞기

```
1 x = cancer['data']      # (569, 30)
2 y = cancer['target']    # (569,)
3
4 index = [i for i in range(x.shape[0])]      # [0, 1, 2, ..., 568]
5 np.random.shuffle(index)                      # [108, 479, 89, ...]
6 x = x[index]
7 y = y[index]
```



1. 데이터셋

트레이닝 / 테스트 데이터 나누기

1

입력 데이터와 타깃 데이터 중 첫 400개 데이터를 트레이닝 데이터로 지정
(x_train, y_train)

2

입력 데이터와 타깃 데이터 중 나머지 데이터를 테스트 데이터로 지정
(x_test, y_test)

4 | 5

x_train, y_train, x_test, y_test의 형태 출력

```
1 x_train, y_train = x[:400], y[:400]
2 x_test, y_test = x[400:], y[400:]
3
4 print('train data:', x_train.shape, 'test data:', x_test.shape)
5 print('train target:', y_train.shape, 'test target:', y_test.shape)
```

train data: (400, 30) test data: (169, 30)
train target: (400,) test target: (169,)



2. 로지스틱 모델 정의

로지스틱 회귀 모델

1 | 2

로지스틱 모델 중 파라미터 w 와 b 정의

- ✓ w 는 13차원 벡터

4 | 5

선형 함수 정의

- ✓ 입력 x 와 w 는 내적 연산

7 | 8

시그모이드 함수 정의

```
1 w = 0.001 * np.random.randn(30)
2 b = 0.001 * np.random.randn(1)
3
4 def linear(x):
5     return x.dot(w) + b
6
7 def sigmoid(x):
8     return 1 / (1 + np.exp(-x))
```

linear

$$y = w_1x_1 + w_2x_2 + \cdots + w_{30}x_{30} + b$$
$$y = w^T x + b$$

sigmoid

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



2. 로지스틱 모델 정의

교차 엔트로피

2

log 함수가 0에는 값이 없기에 아주 작은 값 delta 지정

3

교차 엔트로피 함수 정의

```
1 def cross_entropy_error(p, y):  
2     delta = 1e-7  
3     return -np.average(y*np.log(p+delta) + (1-y)*np.log(1-p+delta))
```

Cross entropy error

$$loss = \frac{1}{m} \sum_{i=1}^m (y \cdot \log(p) + (1 - y) \cdot \log(1 - p))$$



2. 로지스틱 모델 정의

함수 편미분

2

아주 작은 값 h 를 0.0001로 정의

3

미분값을 저장할 grad 변수를 theta의 차원과
같고 0으로 구성된 numpy array로 만들기

5

theta에 있는 모든 값에 대해

6

tmp_val에 현재 theta값 저장

8

현재 theta값에 아주 작은 값 h 를 더하기

9

변화한 함수 값 계산

11

현재 theta값에 아주 작은 값 h 를 빼기

12

변화한 함수 값 계산

14

함수값 변화량과 입력값 변화량의 비율 구하여 grad 변수에 저장

16

theta값 복원

```
1 def numerical_gradient(f, theta):  
2     h = 1e-4                                # 0.0001  
3     grad = np.zeros_like(theta)  
4  
5     for idx in range(theta.size):  
6         tmp_val = theta[idx]  
7  
8         theta[idx] = tmp_val + h  
9         fxh1 = f(theta)                      # f(theta+h)  
10  
11        theta[idx] = tmp_val - h  
12        fxh2 = f(theta)                      # f(theta-h)  
13  
14        grad[idx] = (fxh1 - fxh2) / (2*h)  
15  
16        theta[idx] = tmp_val                # 값 복원  
17  
18    return grad
```



3. 로지스틱 모델 학습

▣ 경사 하강법으로 학습

1 | 2

학습할 epoch수와 학습률 정의

7 | 8

경사 하강법으로 w와 b에 대해 학습

10

학습한 로지스틱 모델로 암 예측

11

예측값과 타깃 데이터의 크로스 엔트로피 오차 계산

13 | 14

1000 epoch마다 오차값 출력

```
1 num_epoch = 20000
2 learning_rate = 0.00003
3
4 # 트레이닝
5 for epoch in range(num_epoch):
6     # 경사 하강법
7     w = w - learning_rate * numerical_gradient(lambda w: cross_entropy_error(sigmoid(linear(x_train)), y_train), w)
8     b = b - learning_rate * numerical_gradient(lambda b: cross_entropy_error(sigmoid(linear(x_train)), y_train), b)
9
10    pred = sigmoid(linear(x_train))
11    loss = cross_entropy_error(pred, y_train)
12
13    if epoch % 1000 == 0:
14        print("{0} epoch, train loss={1}".format(epoch, loss))
```

```
0 epoch, train loss=0.7280788589971133
1000 epoch, train loss=0.20093685881180728
2000 epoch, train loss=0.191804847156605
3000 epoch, train loss=0.1871782221433606
4000 epoch, train loss=0.18377622385569775
5000 epoch, train loss=0.18104656471500452
6000 epoch, train loss=0.17875645136566554
7000 epoch, train loss=0.17678304370968678
8000 epoch, train loss=0.17505236106587346
9000 epoch, train loss=0.17351577669629
10000 epoch, train loss=0.17213916070528357
11000 epoch, train loss=0.17089724486195812
12000 epoch, train loss=0.1697704836594723
13000 epoch, train loss=0.16874322208290743
14000 epoch, train loss=0.16780258422265462
15000 epoch, train loss=0.16693777253256153
16000 epoch, train loss=0.1661396067873156
17000 epoch, train loss=0.1654002063863464
18000 epoch, train loss=0.1647127608794851
19000 epoch, train loss=0.16407135675397483
```



3. 로지스틱 모델 학습

▣ 학습된 모델 테스트

2

로지스틱 모델로 테스트 데이터에 대해 암 예측

3

예측값과 타깃 데이터의 오차 계산

4

오차 출력

```
1 # 테스트  
2 pred = sigmoid(linear(x_test))  
3 loss = cross_entropy_error(pred, y_test)  
4 print("{0} epoch, test loss={1}".format(epoch, loss))
```

19999 epoch, test loss=0.2192639393653472



3. 로지스틱 모델 학습

▣ 혼돈 행렬과 정확도 계산

1 | 2

confusion_matrix와 classification_report 함수 불러오기

4

테스트 데이터로 예측한 값이 0.5보다 크면 1, 작으면 0으로 지정

6

confusion_matrix를 통해 혼돈 행렬 계산

8

classification_report를 통해 각 라벨 별 정확도 계산

```
1 from sklearn.metrics import confusion_matrix  
2 from sklearn.metrics import classification_report  
3  
4 pred = [0.0 if p<0.5 else 1.0 for p in pred]  
5  
6 conf_matrix = confusion_matrix(y_test, pred)  
7 print(conf_matrix)  
8  
9 class_report = classification_report(y_test, pred)  
10 print(class_report)
```

	[[56 8] [6 99]]		precision	recall	f1-score	support
0	0.90	0.88	0.89	64		
1	0.93	0.94	0.93	105		
		accuracy			0.92	169
		macro avg	0.91	0.91	0.91	169
		weighted avg	0.92	0.92	0.92	169



4. 로지스틱 모델 간단한 구현법 (sklearn)

필요한 모델 불러오기

- 1 암 데이터셋을 불러오는 load_breast_cancer
- 2 트레이닝/테스트 데이터를 나누는 train_test_split
- 3 로지스틱 회귀 모델 정의하는 LogisticRegression

```
1 from sklearn.datasets import load_breast_cancer  
2 from sklearn.model_selection import train_test_split  
3 from sklearn.linear_model import LogisticRegression
```



4. 로지스틱 모델 간단한 구현법 (sklearn)

데이터셋

1

cancer에 데이터셋 할당

2 | 3

X에 입력 데이터, y에 타깃 데이터 저장

5

트레이닝/테스트 데이터 나누기

6 | 7

트레이닝/테스트 데이터 형태 출력

```
1 cancer = load_breast_cancer()
2 x = cancer['data']      # (569, 30)
3 y = cancer['target']    # (569,)
4
5 x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.297)
6 print('train data:', x_train.shape, 'test data:', x_test.shape)
7 print('train target:', y_train.shape, 'test target:', y_test.shape)
```

train data: (400, 30) test data: (169, 30)
train target: (400,) test target: (169,)

$$\text{test size} = \frac{\text{test data}}{\text{all data}} = \frac{169}{569} \approx 0.297$$



4. 로지스틱 모델 간단한 구현법 (sklearn)

■ 로지스틱 모델 정의 및 학습

2

로지스틱 모델 정의

4

트레이닝 데이터로 로지스틱 모델 학습

6

테스트 데이터로 로지스틱 모델 테스트

```
1 # 2. model definition  
2 regression = LogisticRegression()  
3 # 3. train  
4 regression.fit(x_train, y_train)  
5  
6 pred = regression.predict(x_test)
```



4. 로지스틱 모델 간단한 구현법 (sklearn)

모델 테스트

1 | 2

confusion_matrix와 classification_report 함수 불러오기

4

confusion_matrix를 통해 혼돈 행렬 계산

7

classification_report를 통해 각 라벨 별 정확도 계산

```
1 from sklearn.metrics import confusion_matrix      [[63  5]
2 from sklearn.metrics import classification_report [ 3 98]]  
3  
4 conf_matrix = confusion_matrix(y_test, pred)          precision    recall   f1-score  support
5 print(conf_matrix)                                     0           0.95      0.93     0.94      68
6  
7 class_report = classification_report(y_test, pred)  1           0.95      0.97     0.96     101
8 print(class_report)                                    accuracy        0.95      0.95     0.95      169
                                         macro avg       0.95      0.95     0.95      169
                                         weighted avg    0.95      0.95     0.95      169
```



classification_report 함수를 사용하지 않고 아래 값 구하기

- ✓ malignant(악성,0), benign(양성,1) 두개 라벨에 대한 precision, recall, f1_score 값
- ✓ confusion_matrix 함수는 사용해도 됨

들어가기

| 학습목표 |

- 선형분류기로서의 Naïve Bayes를 이해
- 확률 추정 기법의 두 가지 방법을 이해
- k-nearest neighbor 의 알고리즘과 속성을 이해

들어가기

| 학습내용 |

- 선형분류기로서의 Naïve Bayes에 대한 개념
- 확률 추정 기법의 두 가지 방법을 이해
- k-nearest neighbor 의 알고리즘과 속성을 이해

들어가기

| 학습목차 |

1 Naïve Bayesian as a linear classifier

2 Non-parametric method

3 k-nearest neighbor (k-NN)의 개념

4 k-nearest neighbor (K-NN)의 성능개선 지표



1. 나이브 베이즈 (Naïve Bayes)

■ 서로 조건부 독립 (conditional independence)인 특징을 가정하고,
베이즈 이론을 기반으로 하는 기계학습 알고리즘

$$P(X_1, \dots, X_n | C) = \prod_{j=1}^n P(X_j | C)$$

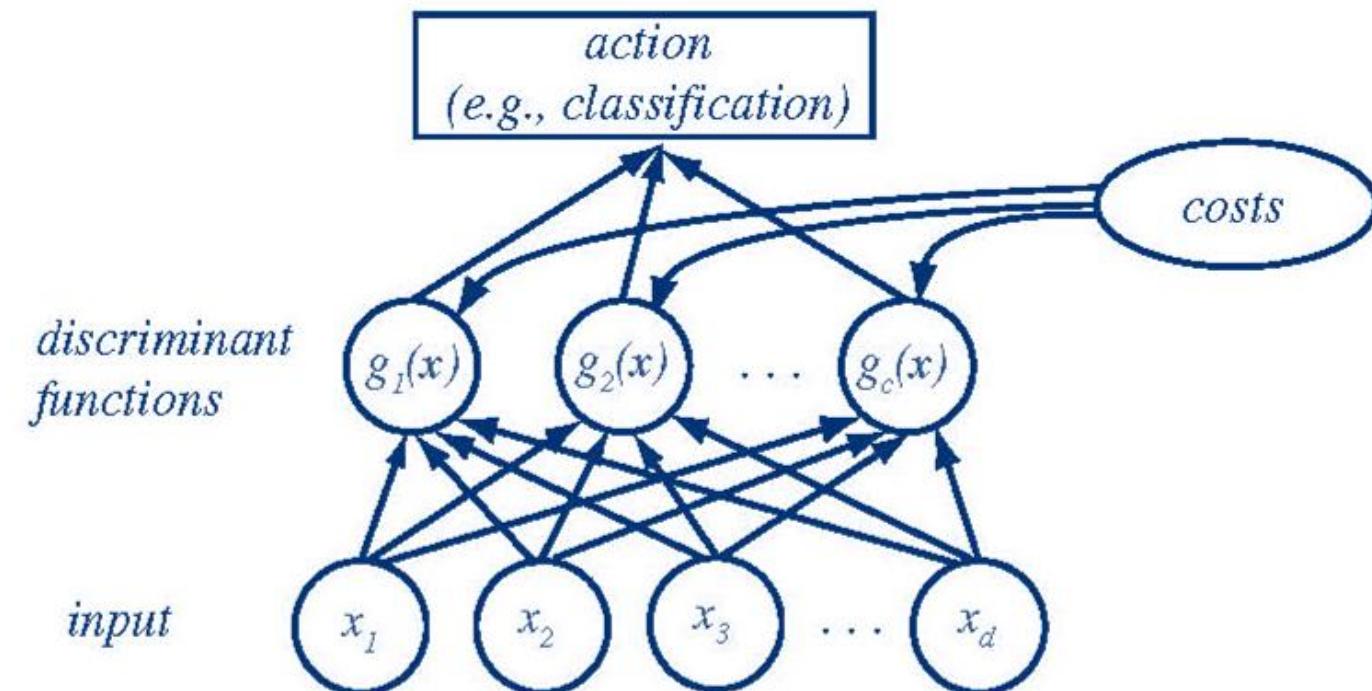
선형 분류기 (linear classifier)로 동작

▣ 식별함수 (discriminant function)

- ✓ $g_i(x) = P(\omega_i/x), i=1, \dots, c$

▣ 입력 특징 벡터 x 를 클래스 ω_i 로 배정

- ✓ $g_i(x) > g_j(x)$





1. 나이브 베이즈 (Naïve Bayes)

■ 우도함수 (likelihood)가 정규분포 $N(\mu, \sigma^2)$ 로 모델링 경우,

$$P(X_1, \dots, X_n | C) = \prod_{j=1}^n P(X_j | C) = \prod_{j=1}^n \left\{ \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\left(\frac{x - \mu_j}{\sqrt{2}\sigma_j}\right)^2\right) \right\}$$

■ 로그를 취하여 식별 함수를 만들어 보면,

- $g_i(\mathbf{x})$ 는 변수 \mathbf{x} 에 대한 1차식
$$g_i(\mathbf{x}) = \ln p(\mathbf{x} / \omega_i) + \ln P(\omega_i)$$

- If $p(x/\omega_i) \sim N(\mu_i, \Sigma_i)$, then

$$g_i(\mathbf{x}) = -\frac{1}{2} (\mathbf{x} - \boldsymbol{\mu}_i)^t \boldsymbol{\Sigma}_i^{-1} (\mathbf{x} - \boldsymbol{\mu}_i) - \frac{d}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{\Sigma}_i| + \ln P(\omega_i)$$

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2} [\mathbf{x}^t \mathbf{x} - 2\boldsymbol{\mu}_i^t \mathbf{x} + \boldsymbol{\mu}_i^t \boldsymbol{\mu}_i] + \ln P(\omega_i)$$



1. 나이브 베이즈 (Naïve Bayes)

■ 우도함수 (likelihood)가 정규분포 $N(\mu, \sigma^2)$ 로 모델링 경우,

$$P(X_1, \dots, X_n | C) = \prod_{j=1}^n P(X_j | C) = \prod_{j=1}^n \left\{ \frac{1}{\sqrt{2\pi\sigma_j^2}} \exp\left(-\left(\frac{x - \mu_j}{\sqrt{2}\sigma_j}\right)^2\right) \right\}$$

■ 로그를 취하여 식별 함수를 만들어 보면,

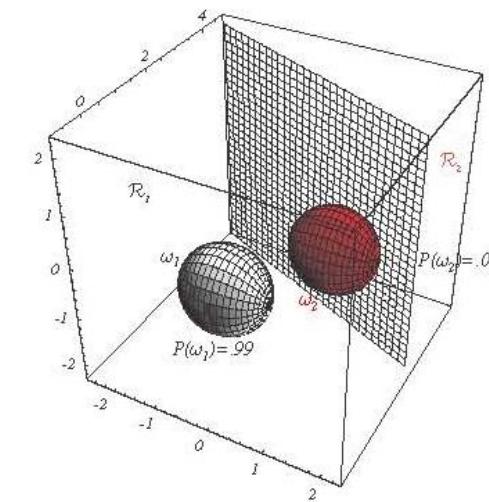
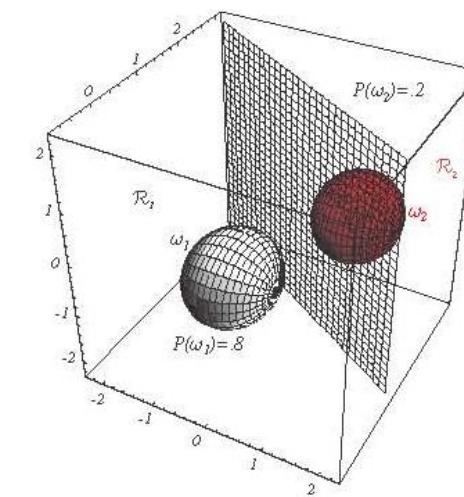
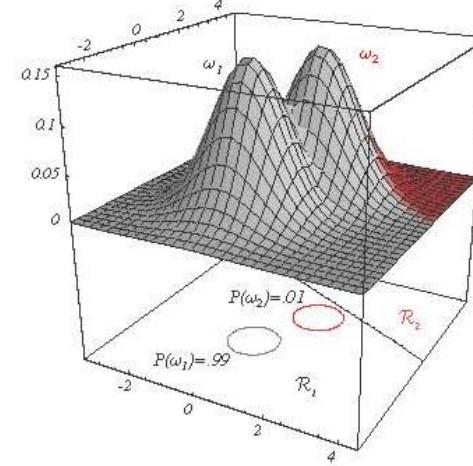
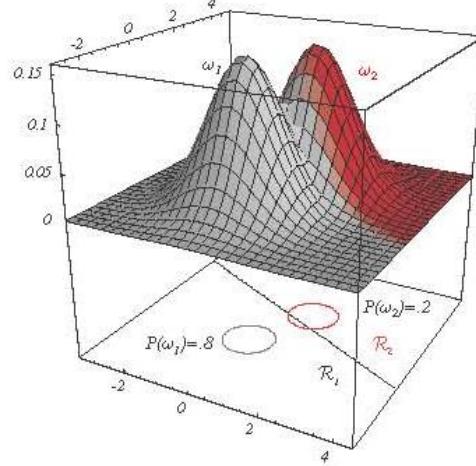
- ✓ $g_i(\mathbf{x})$ 는 변수 \mathbf{x} 에 대한 1차식

■ 선형 분류기 (linear classifier)로 동작

$$g_i(\mathbf{x}) = -\frac{1}{2\sigma^2} [\mathbf{x}^t \mathbf{x} - 2\mu_i^t \mathbf{x} + \mu_i^t \mu_i] + \ln P(\omega_i)$$



1. 나이브 베이즈 (Naïve Bayes)



결정경계선이 사전확률이 작은 쪽으로 가까워짐



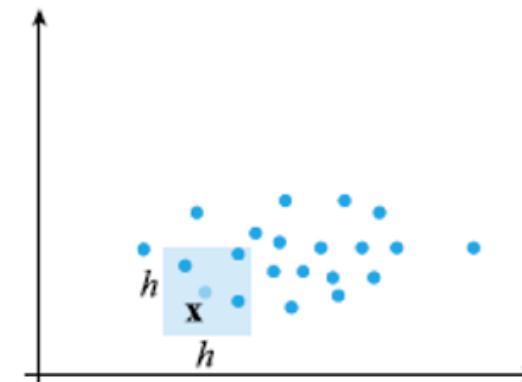
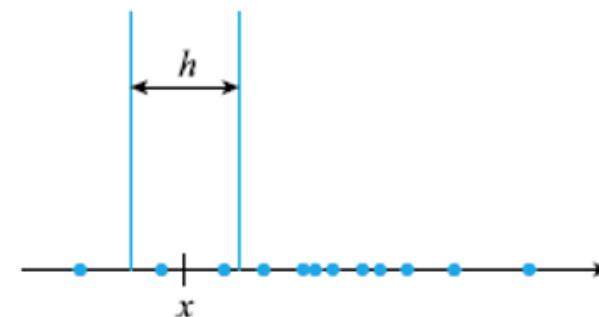
2. 확률 분포 추정 방법

▣ 모수적 방법 (parametric method)

- ✓ 확률 분포가 매개 변수 (파라미터)로 표현되는 형태
- ✓ ML, MAP 방법

▣ 비모수적 방법 (non-parametric method)

- ✓ 확률 분포가 임의의 형태
- ✓ 파젠창 (Pazen window), k-최근접 이웃 추정 (k-Nearest Neighbors) 등



▣ 파젠 창 (Parzen Window)

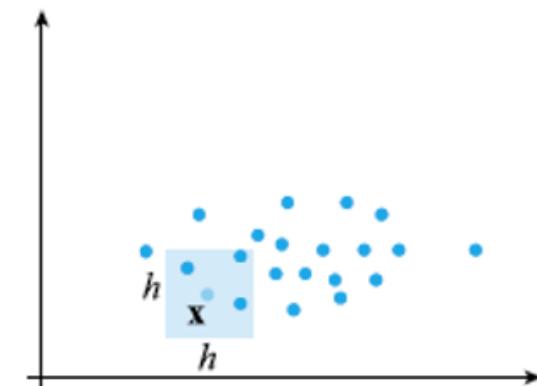
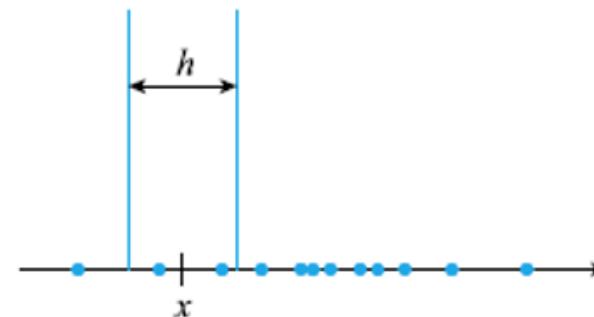
- ✓ 임의의 점 x 에서 확률 값 추정
- ✓ 크기 h 인 창을 씌우고 그 안의 샘플의 개수를 k 라고 할 때,

$$P(x) = \frac{1}{h} \frac{k}{m}$$

- ✓ d 차원으로 확대하면,

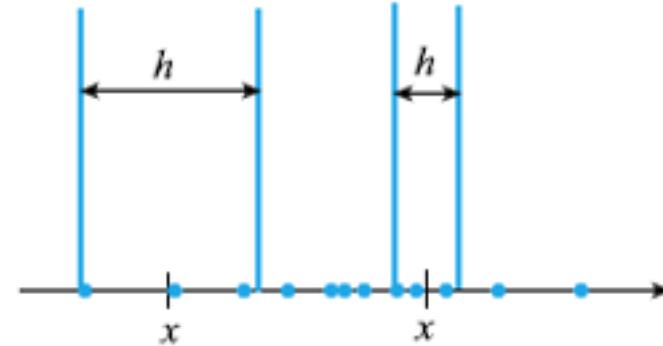
$$P(x) = \frac{1}{h^d} \frac{k}{m}$$

- ✓ h 가 고정이고 k 가 가변

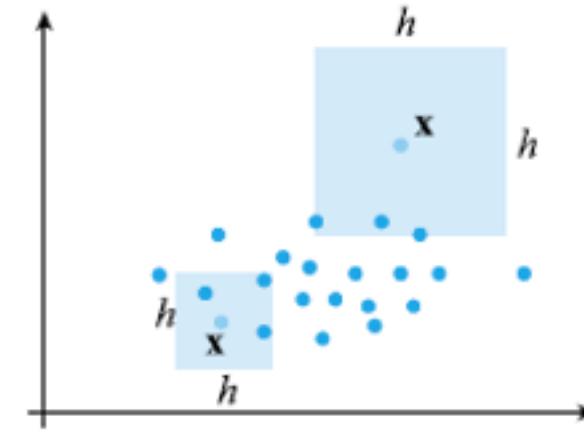


■ k -nearest neighbor (k -NN)

- ✓ x 를 중심으로 창을 씌우고 k 개 샘플이 안에 들어올 때까지 확장하고 그 순간의 창의 크기를 h 로 정의
- ✓ k -NN에서는 k 가 고정이고 h 가 가변



(a) 1 차원 특징 공간



(b) 2 차원 특징 공간



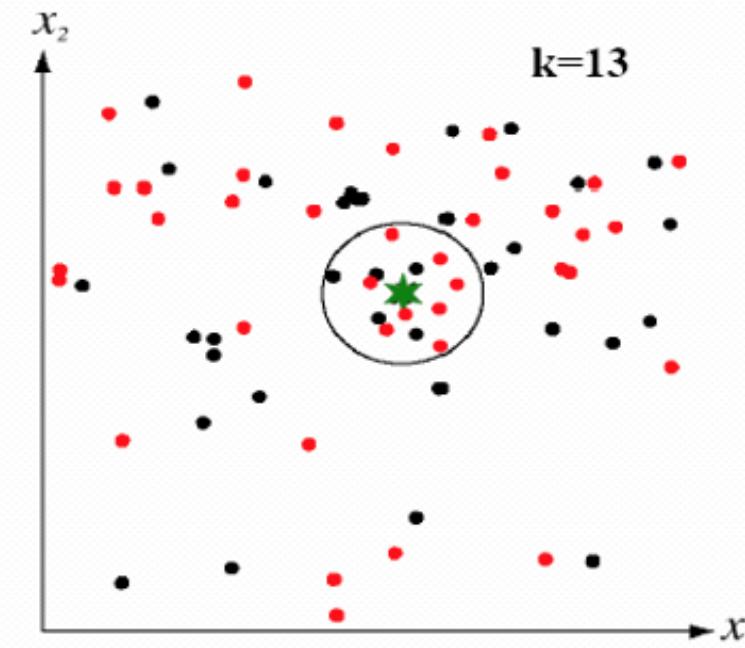
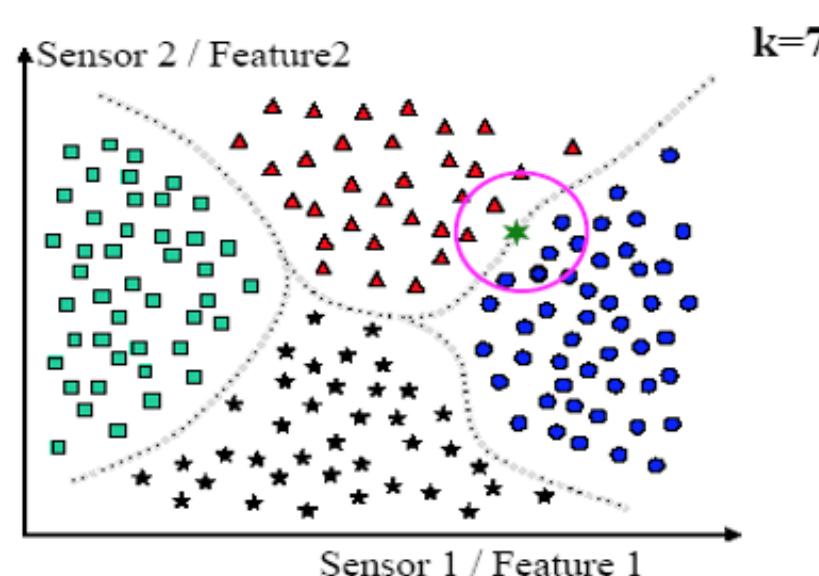
3. k -nearest neighbor (k -NN)

- 데이터간의 거리 측정 방법
- 가장 가까이 존재하는 이웃의 개수를 선택
- 가중치 부여

3. k -nearest neighbor (k -NN)

□ 이해하기 쉽고 자주 사용되는 방법

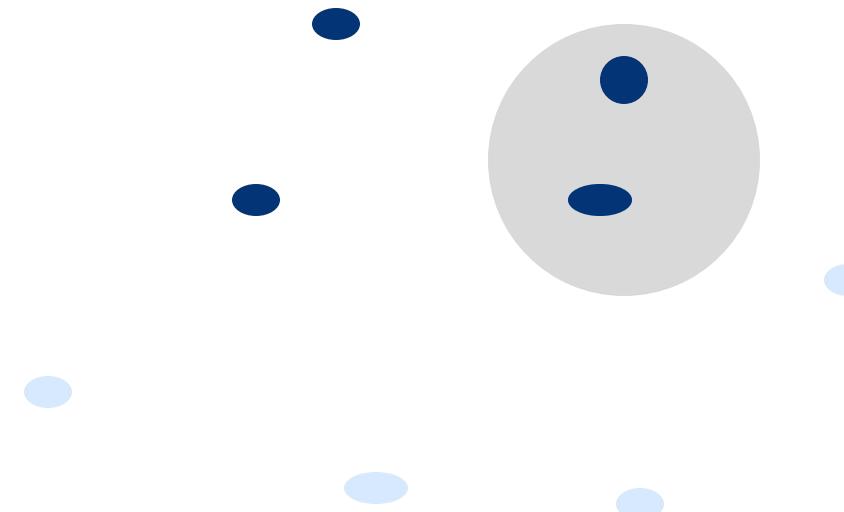
□ 비교 대상이 되는 데이터 주변에 가장 가까이 존재하는 k 개의 데이터와 비교해 가장 가까운 데이터 종류로 판별





3. k -nearest neighbor (k -NN)

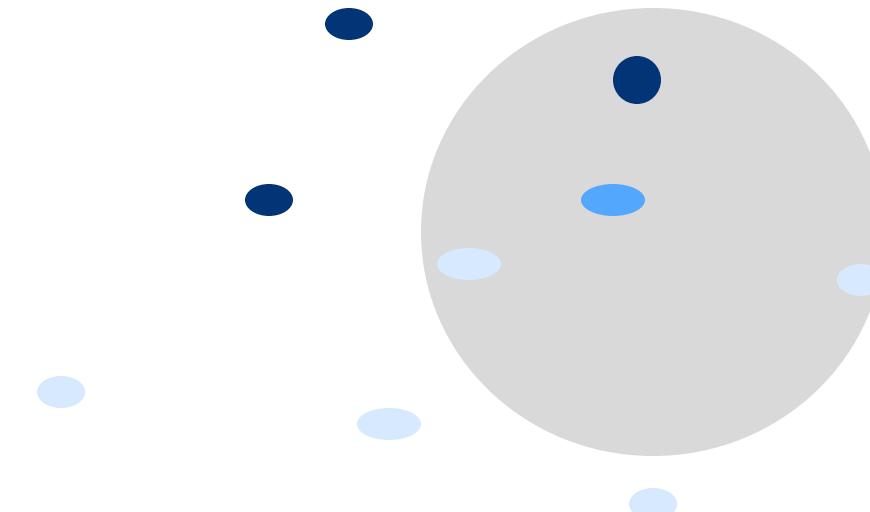
■ 1-Nearest Neighbor





3. k -nearest neighbor (k -NN)

3-Nearest Neighbor





3. *k*-nearest neighbor (*k*-NN)

■ Memory(instance) based learning

- 새로운 데이터 (unknown data)와 학습데이터에 있는 데이터들과의 비교를 통해 가장 가까운 데이터들을 선택, 따라서 트레이닝 데이터 전체를 메모리에 보관하고 매번 access하여 비교를 수행

■ 게으른 학습 방법 (lazy learning algorithm)

- 추가적인 학습 시간 없이, 테스트 데이터가 들어왔을 때 바로 학습을 수행
- Opposite to eager learning algorithm (예: neural network)

■ 장점: 학습시간 없이, 바로 결과를 얻을 수 있음

■ 단점: 예측 시 메모리상에 학습 데이터를 항상 보관하고 있어야 하므로, 큰 사이즈의 데이터에 대해서는 사용할 수 없음



3. k -nearest neighbor (k -NN)

- 데이터들간의 거리 척도 (distance metric)

- 인접한 데이터의 개수 k 선택

- 가중치 함수 (optional)



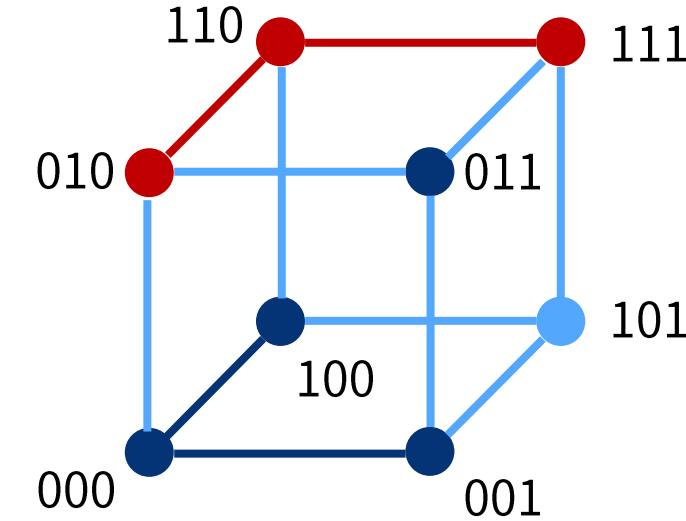
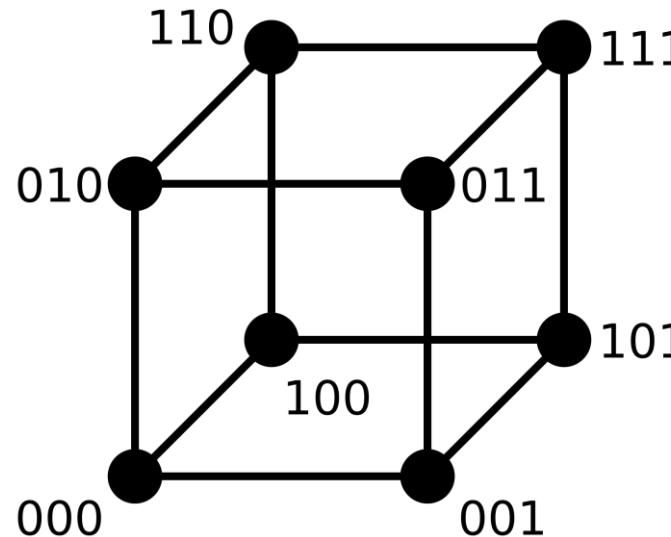
4. 거리 척도 (Distance metrics)

- 이산형 데이터 (discrete variable)
- 연속형 데이터 (continuous variable)

4. 거리 척도 (Distance metrics)

▣ 이산형 변수

- ✓ Hamming distance



▣ Example application: document classification



4. 거리 척도 (Distance metrics)

■ 연속형 변수

- ✓ L_2 -norm: Euclidean distance

$$D(x, x') = \sqrt{\sum_i (x_i - x'_i)^2}$$

- ✓ L_1 -norm: Sum of absolute difference

$$D(x, x') = \sum_i |x_i - x'_i|$$

- ✓ Scaled Euclidean distance

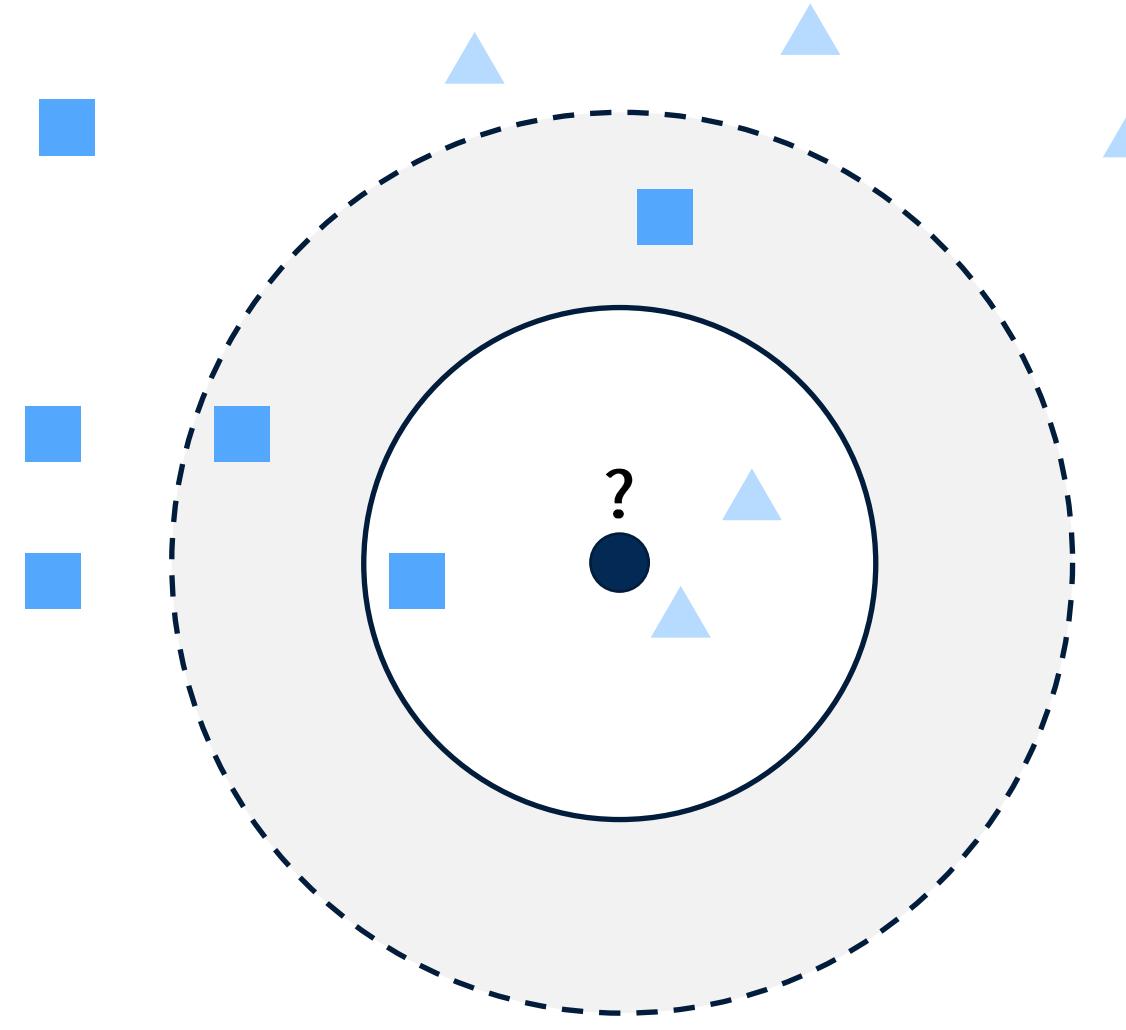
$$D(x, x') = \sqrt{\sum_i \sigma_i^2 (x_i - x'_i)^2}$$

- ✓ Mahalanobis distance

$$D(x, x') = \sqrt{(x - x')^\top A (x - x')}$$



5. 인접한 이웃의 개수, k 선택



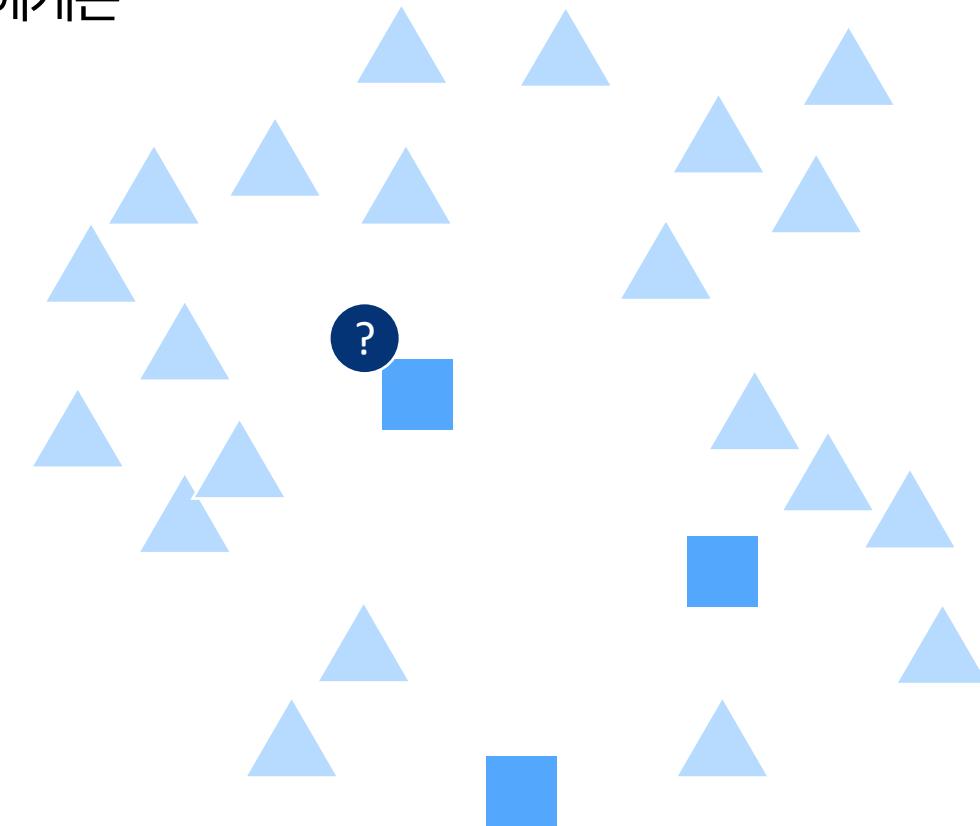
Issue: 편향된 (skewed) 클래스 분포

- ✓ kNN에서의 majority voting의 문제점
- ✓ 가까운 점들에게 가중치를 높게, 먼 거리의 데이터들에게는 가중치를 낮게 설정

가중치

$$w^{(i)} = \exp\left(-\frac{d(x^{(i)}, \text{query})^2}{\sigma^2}\right)$$

- σ^2 : Kernel width





7. K-NN의 문제

▣ 비용이 큼 (expensive)

- ✓ No learning, 테스트에서 대부분의 작업을 수행
- ✓ 매번 새로운 데이터 입력에 대해 전체 데이터셋을 탐색해야만 함

▣ 노이즈에 민감

- ✓ Noisy feature에 거리가 영향을 많이 받음

▣ 차원의 저주 (curse of dimensionality)

▣ 데이터의 크기 >> 메모리 용량 시, 불가능

정리하기

- ▶ 데이터들 간의 거리를 정의
- ▶ 인접한 데이터의 개수 k 선택
- ▶ 가중치 함수 (optional)

들어가기

| 학습목표 |

- 기계학습에서 자주 사용되는 통계학적인 개념을 이해한다.
- Naïve Bayes 이론을 이해한다.
- 주어진 데이터로부터 파라미터를 추정하여, generative classifier를 설계할 수 있다.

들어가기

| 학습내용 |

- Bayesian rule의 개념을 이해
- 우도함수와 사후확률 추정
- Naïve Bayes의 특징을 이해하고 분류기로 활용

들어가기

| 학습목차 |

1 | 확률 기초

2 | Bayesian Classifier

3 | 데이터로부터의 파라미터 추정

- 우도함수 최대화 (maximum likelihood)
- 사후확률 최대화 (maximum a posteriori: MAP)

4 | Naïve Bayes



0. 확률 기초 (recap)

▣ 사전확률 (prior probability) $P(X)$

▣ 조건부확률 (conditional probability) $P(X_1|X_2), P(X_2|X_1)$

▣ 결합확률 (joint probability) $\mathbf{X} = (X_1, X_2), P(\mathbf{X}) = P(X_1, X_2)$

- Relationship: $P(X_1, X_2) = P(X_2|X_1)P(X_1) = P(X_1|X_2)P(X_2)$
- Independence: $P(X_2|X_1) = P(X_2), P(X_1|X_2) = P(X_1), P(X_1, X_2) = P(X_1)P(X_2)$



1. Bayesian Classifier

■ Bayes rule

$$P(C|\mathbf{X}) = \frac{P(\mathbf{X}|C)P(C)}{P(\mathbf{X})} \quad Posterior = \frac{Likelihood \times Prior}{Evidence}$$

$$P(C|X) = \frac{P(X|C)P(C)}{P(X|C)P(C) + P(X|\sim C)P(\sim C)}$$

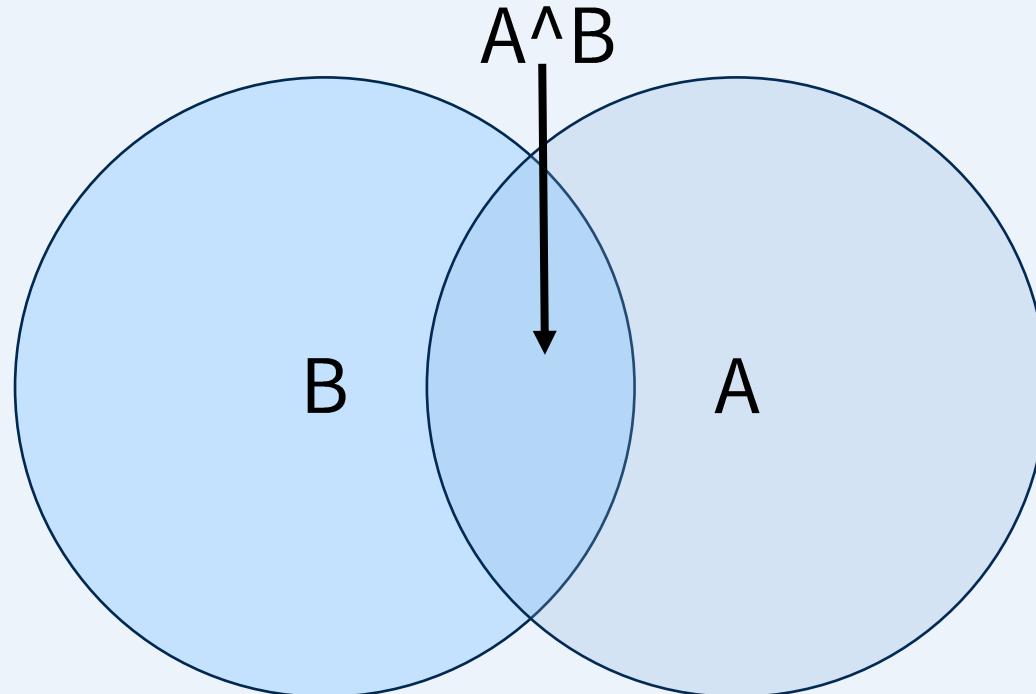


Thomas Bayes



1. Bayesian Classifier

■ Bayes rule



$$P(A|B) = \frac{P(A, B)}{P(B)}$$
$$= \frac{P(B|A)P(A)}{P(B)}$$

$$P(C|\mathbf{X}) = \frac{P(\mathbf{X}|C)P(C)}{P(\mathbf{X})}$$

Corollary: The chain rule

$$P(A, B) = P(A|B)P(B) = P(B)P(A|B)$$

■ Bayes rule의 응용 예

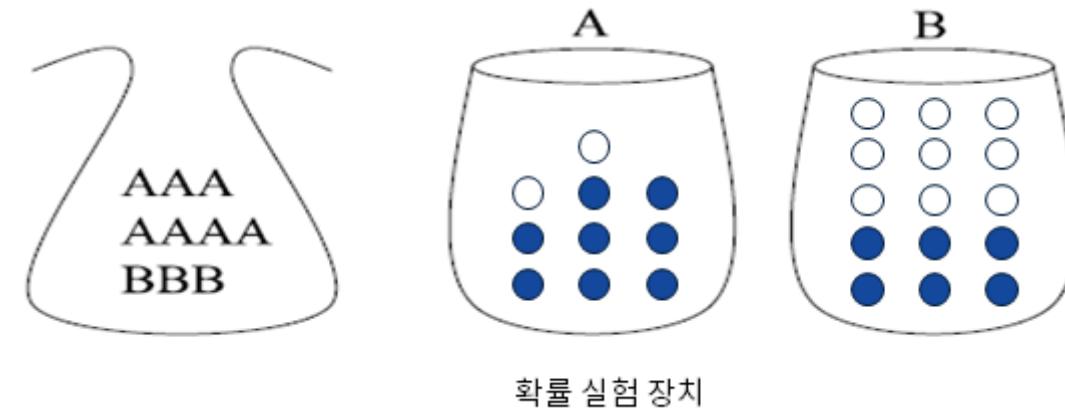
$$P(C|X) = \frac{P(X|C)P(C)}{P(X|C)P(C) + P(X|\sim C)P(\sim C)}$$

- ✓ C: 독감 (flu)로 진단
- ✓ X: 기침 증상이 있음
- ✓ 가정
 - $P(C) = 0.05$
 - $P(X|C) = 0.8$
 - $P(X|\sim C) = 0.2$
- ✓ 문제: 기침증상이 있을 때 독감에 걸렸을 확률은?
 $P(\text{flu} | \text{cough}) = P(C|X)?$



1. Bayesian Classifier

■ Bayes rule의 응용 예: 하얀 공이 뽑혔는데 어느 상자에서 나왔는지 맞추어라.



$$P(A | \text{하양}) = \frac{P(\text{하양} | A)P(A)}{P(\text{하양})} = \frac{(2/10)(7/10)}{(8/25)} = 0.4375$$

$$P(B | \text{하양}) = \frac{P(\text{하양} | B)P(B)}{P(\text{하양})} = \frac{(9/15)(3/10)}{(8/25)} = 0.5625$$



1. Bayesian Classifier

▣ Casual vs. Diagnostic reasoning

- ✓ $P(C|X)$: diagnostic
- ✓ $P(X|C)$: causal
- ✓ 일반적으로 causal knowledge 의 획득이 보다 쉬움 (표본데이터로부터 확률 계산 용이)
- ✓ Bayes rule을 통해 복잡한 diagnostic knowledge를 casual knowledge로 대치할 수 있음

$$P(\text{flu}|\text{cough}) = P(C|X) = \frac{0.8 \times 0.05}{0.8 \times 0.05 + 0.2 \times 0.95} \sim 0.17$$



1. Bayesian Classifier

■ Discriminative Classifier

- ✓ 주어진 입력에 대해 클래스에 소속될 확률을 모델링

$$P(C|\mathbf{X}) \quad C = c_1, \dots, c_L, \mathbf{X} = (X_1, \dots, X_n)$$

- ✓ k-NN (nearest neighbor), decision trees, perceptron, MLP, SVM

■ Generative Classifier

- ✓ 각 클래스에서 데이터의 분포도 모델을 생성, 이를 기반으로 분류를 수행

$$P(\mathbf{X}|C) \quad C = c_1, \dots, c_L, \mathbf{X} = (X_1, \dots, X_n)$$

- ✓ Bayesian classifier,, model - based classifiers



1. Bayesian Classifier

MAP classification rule

- ✓ MAP: Maximum A Posterior
 - Assign x to c^* if $P(C = c^*|X = x) > P(C = c|X = x)$ $c \neq c^*, c = c_1, \dots, c_L$
- ✓ ML: Maximum Likelihood
 - Assign x to c^* if $P(X = x|C = c^*) > P(X = x|C = c)$ $c \neq c^*, c = c_1, \dots, c_L$

Generative Classification with MAP

$$P(C|\mathbf{X}) = \frac{P(\mathbf{X}|C)P(C)}{P(\mathbf{X})} \propto P(\mathbf{X}|C)P(C)$$



2. 확률 추정

■ Objective (relative frequency) approach

- ✓ 실험 데이터로부터 직접적으로 확률을 계산

■ Subjective (Bayesian) approach

- ✓ 특정 불포 모델로 가정하여 확률을 계산



2. 확률 추정

▣ 예제

- ✓ 차량의 가격이 \$50K 이상인지 아닌지를 분류하는 문제
- ✓ 분류 클래스: C_1 if price > \$50K, C_2 if price <= \$50K
- ✓ 특징 (feature) : x , the **height** of a car
- ✓ 사후확률을 이용하여 클래스를 분류

$$P(C_i|X) = \frac{P(X|C_i)P(C_i)}{P(X)}$$

- ✓ $P(X|C_1), P(X|C_2), P(C_1), P(C_2)$ 의 확률이 필요



2. 확률 추정

■ Objective approach를 통한 확률 추정

- ✓ 데이터 수집
 - 운전자들에게 자신의 차의 가격과 높이를 질문
- ✓ 사전확률 추정 : $P(C_1), P(C_2)$
- ✓ 예를 들어, 1209개의 샘플데중 #. $C_1 = 221$, #. $C_2 = 988$

$$P(C_1) = \frac{221}{1209} = 0.183$$

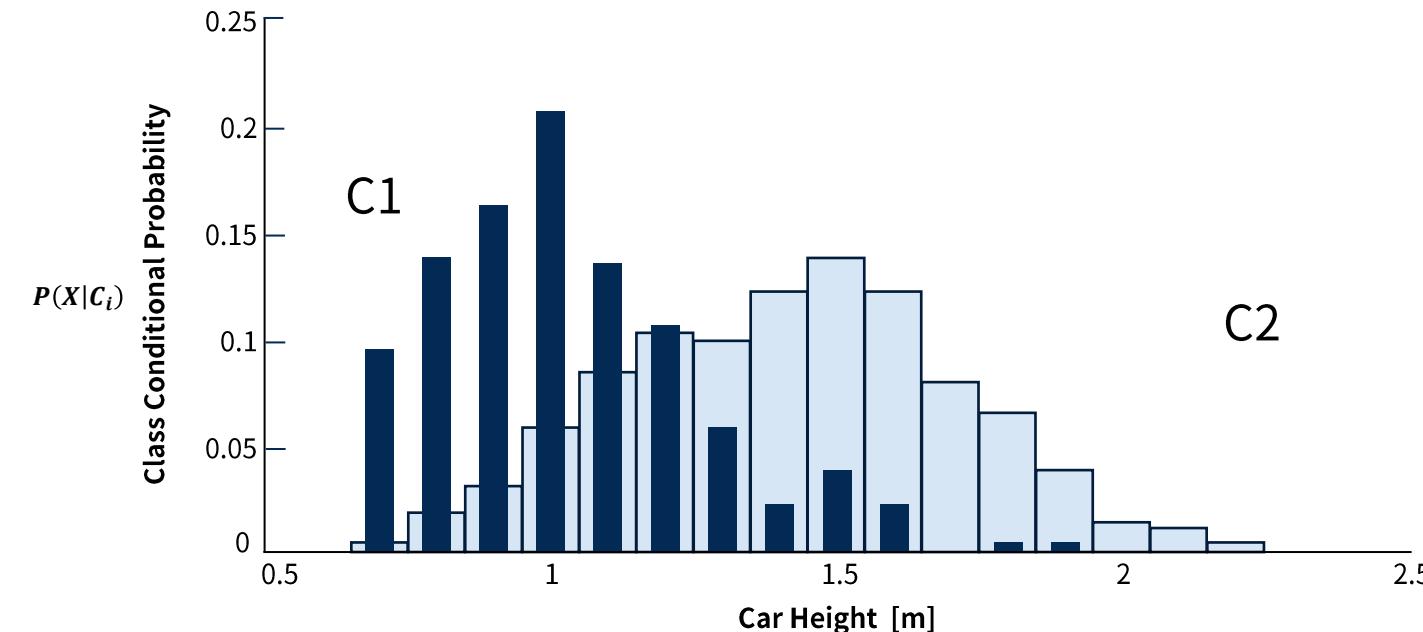
$$P(C_2) = \frac{988}{1209} = 0.817$$

■ Objective approach를 통한 확률 추정

- ✓ 데이터 수집
 - 운전자들에게 자신의 차의 가격과 높이를 질문
- ✓ 사전확률 추정 : $P(C_1), P(C_2)$
- ✓ 우도함수 (likelihood, class conditional probabilities) 계산 :

$$P(C_1) = \frac{221}{1209} = 0.183$$

$$P(C_2) = \frac{988}{1209} = 0.817$$





2. 확률 추정

■ Subjective approach를 통한 확률 추정

- ✓ 우도함수를 모델링 : uniform/normal/Bernoulli/binomial/multinomial 분포로 가정한 후, 모델 파라미터 추정
- ✓ 예를 들어, 다변량 정규 분포, $N(\mu, \Sigma)$ 로 모델링 한 경우, 평균 벡터와 과 공분산 행렬을 추정

$$\hat{P}(X = (X_1, X_2, \dots, X_n) | C_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}(X - \mu)^T \Sigma^{-1} (X - \mu)\right)$$

- ✓ 학습 단계: **우도함수를 보다 단순하게 표현할 수 있을까?**

- $X = (X_1, X_2, \dots, X_n)$ 과 $C = C_1, C_2, \dots, C_L$ 에 대해 L 개의 다변량 정규 분포를 학습

- ✓ 테스트 단계:

- 새로운 입력 X' 에 대해, 학습된 우도함수를 이용하여, 사후 확률을 계산
- 가장 큰 사후확률을 가지는 클래스로 결정



3. Naïve Bayes

□ Bayes classification

- ✓ 조건부 결합 확률, 우도함수의 학습 $\hat{P}(X = (X_1, X_2, \dots, X_n) | C_i)$ 이 어려움

□ Naïve Bayes classification

- ✓ 모든 특징들이 서로 독립적이라고 가정

$$P(X_1, \dots, X_n | C) = \prod_{j=1}^n P(X_j | C)$$

$$\begin{aligned} P(X_1, X_2, \dots, X_n | C) &= P(X_1 | X_2, \dots, X_n; C) P(X_2, \dots, X_n | C) \\ &= P(X_1 | C) P(X_2, \dots, X_n | C) \\ &= P(X_1 | C) P(X_2 | C) \cdots P(X_n | C) \end{aligned}$$

- ✓ MAP classification rule

$$[P(x_1 | c^*) \cdots P(x_n | c^*)] P(c^*) > [P(x_1 | c) \cdots P(x_n | c)] P(c), \quad c \neq c^*, c = c_1, \dots, c_L$$



3. Naïve Bayes

■ Naïve Bayes Algorithm (for discrete input features)

- ✓ 학습 단계 : 학습 데이터 셋, S 에 대해

For each target value of c_i ($c_i = c_1, \dots, c_L$)

$\hat{P}(C = c_i) \leftarrow$ estimate $P(C = c_i)$ with examples in S ;

For every attribute value a_{jk} of each attribute x_j ($j = 1, \dots, n$; $k = 1, \dots, N_j$)

$\hat{P}(X_j = a_{jk} | C = c_i) \leftarrow$ estimate $P(X_j = a_{jk} | C = c_i)$ with examples in S ;

Output: conditional probability tables; x_j 에 대해 $N_j \times L$ elements

- ✓ 테스트 단계: 새로운 샘플 $\mathbf{X}' = (a'_1, \dots, a'_n)$

$[\hat{P}(a'_1 | c^*) \cdots \hat{P}(a'_n | c^*)] \hat{P}(c^*) > [\hat{P}(a'_1 | c) \cdots \hat{P}(a'_n | c)] \hat{P}(c), c \neq c^*, c = c_1, \dots, c_L$

Output: X' 에게 c^* 을 할당



3. Naïve Bayes

■ Naïve Bayes Algorithm (for continuous input features)

- ✓ 학습 단계 : 학습 데이터 셋, S 에 대해 정규 분포, $N(\mu, \sigma^2)$ 의 평균과 분산을 추정

$$\hat{P}(X_j | C = c_i) = \frac{1}{\sqrt{2\pi}\sigma_{j_i}} \exp\left(-\frac{(X_j - \mu_{j_i})^2}{2\sigma_{j_i}^2}\right)$$

Output: $X = (X_1, X_2, \dots, X_n)$ 와 $C = C_1, C_2, \dots, C_L$ 에 대해 $n \times L$ 개의 정규 분포를 학습

- ✓ 테스트 단계:

- 새로운 입력 X' 에 대해, 학습된 우도함수를 이용하여, 사후 확률을 계산
- 가장 큰 사후확률을 가지는 클래스로 결정



4. Naïve Bayes의 응용

Example: Playing Tennis

PlayTennis: Training examples

Day	Outlook	Temperature	humidity	wind	Play tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No



4. Naïve Bayes의 응용

▣ 테스트 단계

- ✓ 새로운 입력 \mathbf{x}' =(Outlook=*Sunny*, Temperature=*Cool*, Humidity=*High*, Wind=*Strong*)
- ✓ Look-up tables

$$P(\text{Outlook}=\text{Sunny} \mid \text{Play}=\text{Yes}) = 2/9$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{Yes}) = 3/9$$

$$P(\text{Play}=\text{Yes}) = 9/14$$

$$P(\text{Outlook}=\text{Sunny} \mid \text{Play}=\text{No}) = 3/5$$

$$P(\text{Temperature}=\text{Cool} \mid \text{Play}=\text{No}) = 1/5$$

$$P(\text{Humidity}=\text{High} \mid \text{Play}=\text{No}) = 4/5$$

$$P(\text{Wind}=\text{Strong} \mid \text{Play}=\text{No}) = 3/5$$

$$P(\text{Play}=\text{No}) = 5/14$$

- ✓ MAP 추정

$$P(\text{Yes} \mid \mathbf{x}') = [P(\text{Sunny} \mid \text{Yes})P(\text{Cool} \mid \text{Yes})P(\text{High} \mid \text{Yes})P(\text{Strong} \mid \text{Yes})]P(\text{Play}=\text{Yes}) = 0.0053$$

$$P(\text{No} \mid \mathbf{x}') = [P(\text{Sunny} \mid \text{No})P(\text{Cool} \mid \text{No})P(\text{High} \mid \text{No})P(\text{Strong} \mid \text{No})]P(\text{Play}=\text{No}) = 0.0206$$

Given the fact $P(\text{Yes} \mid \mathbf{x}') < P(\text{No} \mid \mathbf{x}')$, we label \mathbf{x}' to be "No".

정리하기

- ▶ Bayesian classifier : Bayes rule을 이용하여 사후확률을 추정
- ▶ Naïve Bayes Classification: 입력 특징 간에 독립성을 가정한 Bayes classification
- ▶ Popular generative model

들어가기

| 학습목표 |

- 기계학습에서 자주 사용되는 통계학적인 개념을 이해한다.
- Naïve Bayes 이론을 이해한다.
- 주어진 데이터로부터 파라미터를 추정하여, generative classifier를 설계할 수 있다.

들어가기

| 학습내용 |

- 확률 변수
- 확률 분포
- 데이터로부터 평균과 분산 파라미터 추정

들어가기

| 학습내용 |

- 확률 변수간의 상관관계 및 상관계수
- 기계학습에서 자주 사용되는 확률 분포
- 확률 실험

들어가기

| 학습목차 |

1 | 확률 기초

2 | Bayesian Classifier

3 | 데이터로부터의 파라미터 추정

- 우도함수 최대화 (maximum likelihood)

- 사후확률 최대화 (maximum a posteriori: MAP)

4 | Naïve Bayes



1. 확률 변수

▣ 확률 변수 (random variable)

- ✓ 하나의 사상에 대하여 하나의 값을 부여하는 함수
- ✓ 예) 동전을 한번 던졌을 때, 앞면이 나오는 횟수 $X \rightarrow P(X = 0) \text{ or } P(X = 1)$
- ✓ 확률 변수는 X, Y 등의 영어 대문자로 표현하며, 그 값은 와 같이 영어 소문자로 표현

▣ 표본 공간 (sample space)

- ✓ 상호 배타적인 값들의 집합
- ✓ 변수 X 의 표본공간은 Ω_X 로 표시



1. 확률 변수

▣ 결합 확률 변수

- ✓ $Z = (X, Y)$: X 와 Y 의 곱집합 (conjunction)
- ✓ Z 의 표본공간: X 와 Y 의 표본공간의 곱, 즉 결합 표본공간 $\Omega_Z = \Omega_X \times \Omega_Y$
- ✓ 확률변수의 집합 $\{X_1, X_2, \dots, X_N\}$ 에 대해 결합 표본공간은

$$\Omega_{X_1} \times \Omega_{X_2} \times \cdots \times \Omega_{X_N} = \prod_{i=1}^N \Omega_{X_i}$$



2. 확률 분포

■ $P(S) = 1$

확률 분포 (probability distribution)

- ✓ 확률 변수가 특정값을 가질 확률의 함수
- ✓ 즉, 확률 변수가 특정값을 가질 확률이 얼마나 되느냐를 나타내는 것
- ✓ 예) 동전을 한번 던졌을 때, 앞면이 나오는 횟수 X 에 대한 확률 분포를 결정

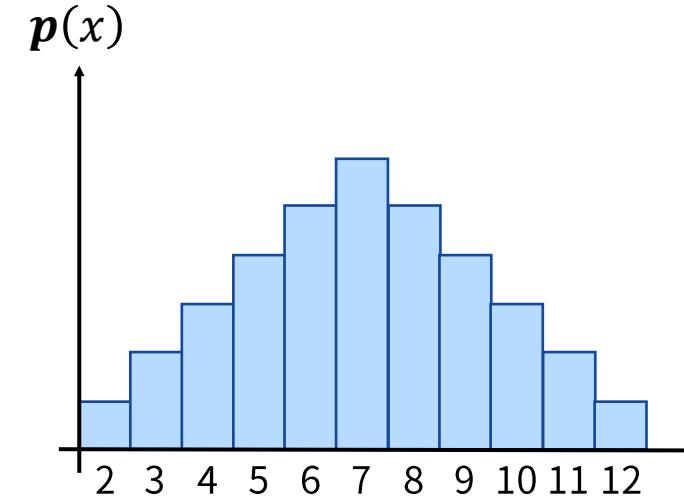
$$P(X = 0) = \frac{1}{2} \text{ or } P(X = 1) = 1/2$$

■ $0 \leq P(X) \leq 1$

■ $P(S) = 1$

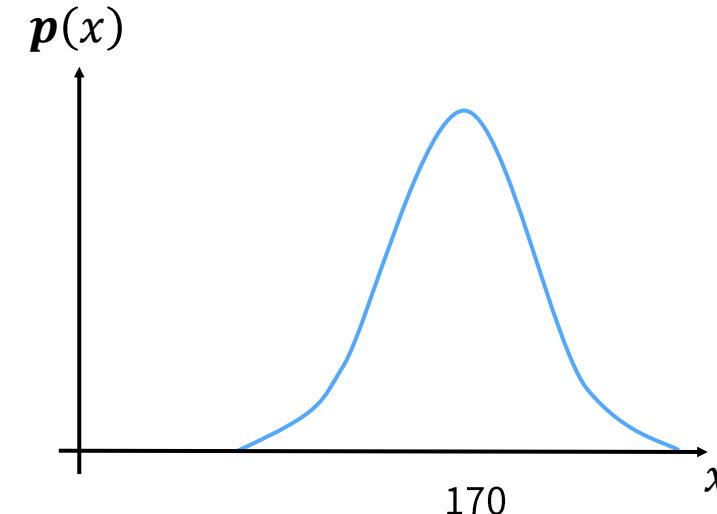
▣ 주사위

- ✓ 주사위 던졌을 때 3이 나올 확률 $P(X=3)=1/6$
- ✓ X 를 랜덤 변수라 부름
- ✓ 이 경우 X 는 이산 값을 가짐



▣ 사람 키

- ✓ 연속 값
- ✓ 확률 밀도 함수 $P(X)$



▣ 패턴 인식에서 특징 각각이 랜덤 변수에 해당



2. 확률분포

▣ 확률 분포 (probability distribution)



Sample space
Area = 1

$$P(X) + P(\sim X) = 1$$



3. 평균과 분산

▣ 평균 (mean)

- ✓ 산술 평균: 모든 데이터값을 덧셈한 후 데이터 개수로 나누는 것을 의미
- ✓ $E(X) = \mu$

- ✓ 샘플 확률 분포 : $\bar{X} = \frac{1}{m} \sum_{i=1}^m x_i$

- ✓ 이산 확률 분포 : $E(X) = \sum_x xP(x)$

- ✓ 연속 확률 분포 : $E(X) = \int_{-\infty}^{\infty} xf(x)dx$



3. 평균과 분산

▣ 분산 (variance)

- ✓ 데이터가 얼마나 퍼져 있는지를 수치화한 것
- ✓ 평균에 대한 편차 제곱의 평균으로 계산됨

$$Var(X) = E[(X - \mu)^2] = \sigma^2$$

- ✓ 이산 확률 분포 : $Var(X) = \sum_x (x - \mu)^2 P(x)$
- ✓ 연속 확률 분포 : $Var(X) = \int_{-\infty}^{\infty} (x - \mu)^2 f(x) dx$

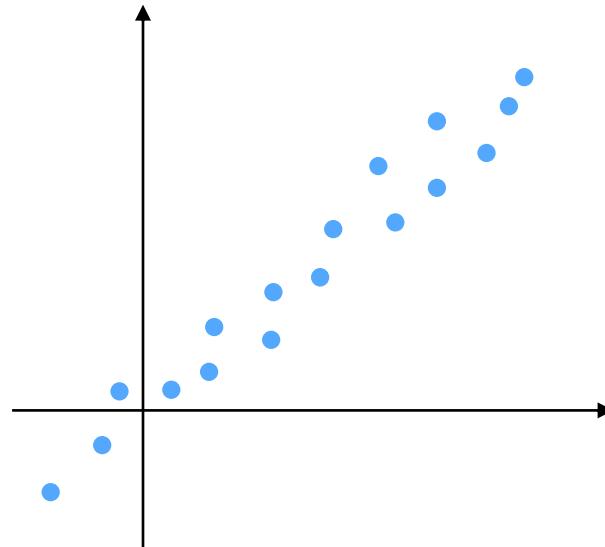


4. 상관관계

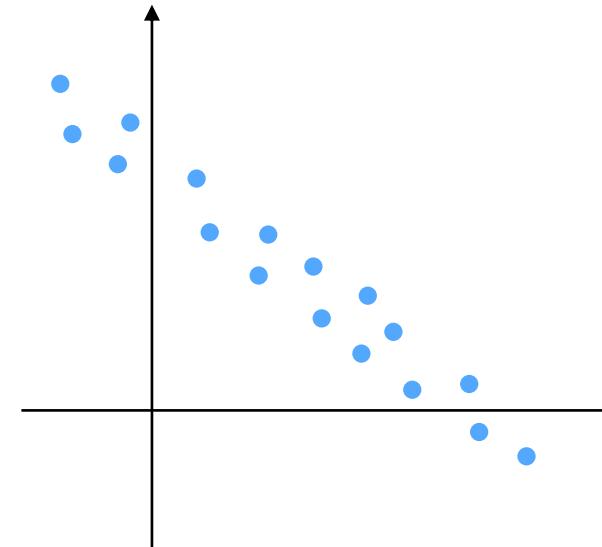
▣ 공분산 (covariance)

- 두 확률 변수의 상관관계를 나타내는 값

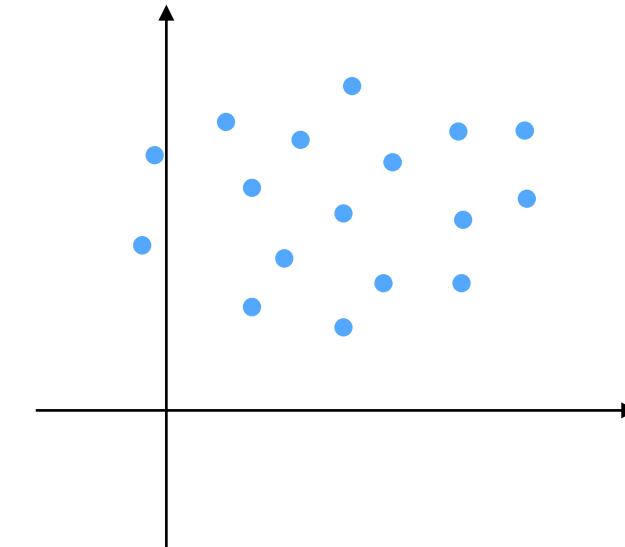
$$Cov(X, Y) = E[(X - \mu_X)(Y - \mu_Y)]$$



양의 상관관계
positive correlation



음의 상관관계
negative correlation



상관관계 없음
no correlation



4. 상관관계 및 상관계수

▣ 공분산행렬 (covariance matrix)

- ✓ 확률 변수간 분산, 공분산을 행렬로 표현한 것
- ✓ 기계학습에서 자주 쓰이며 특히 차원 축소할 때 자주 사용됨

$$\bar{X} = \begin{bmatrix} \bar{x}_1 & \bar{x}_2 & \cdots & \bar{x}_p \\ \bar{x}_1 & \bar{x}_2 & \cdots & \bar{x}_p \\ \vdots & \vdots & \ddots & \vdots \\ \bar{x}_1 & \bar{x}_2 & \cdots & \bar{x}_p \end{bmatrix}$$

$$X - \bar{X} = \begin{bmatrix} (x_{11} - \bar{x}_1) & (x_{12} - \bar{x}_2) & \cdots & (x_{1p} - \bar{x}_p) \\ (x_{21} - \bar{x}_1) & (x_{22} - \bar{x}_2) & \cdots & (x_{2p} - \bar{x}_p) \\ \vdots & \vdots & \ddots & \vdots \\ (x_{n1} - \bar{x}_1) & (x_{n2} - \bar{x}_2) & \cdots & (x_{np} - \bar{x}_p) \end{bmatrix}$$



4. 상관관계 및 상관계수

▣ 공분산행렬 (covariance matrix)

- ✓ 확률 변수간 분산, 공분산을 행렬로 표현한 것
- ✓ 기계학습에서 자주 쓰이며 특히 차원 축소할 때 자주 사용됨

$$\Sigma_X = E[(X - \bar{X})^T(X - \bar{X})]$$

$$= E \begin{bmatrix} (x_{11} - \bar{x}_1) & (x_{21} - \bar{x}_1) & \cdots & (x_{n1} - \bar{x}_1) \\ (x_{12} - \bar{x}_2) & (x_{22} - \bar{x}_2) & \cdots & (x_{n2} - \bar{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ (x_{1p} - \bar{x}_p) & (x_{2p} - \bar{x}_p) & \cdots & (x_{np} - \bar{x}_p) \end{bmatrix} \begin{bmatrix} (x_{11} - \bar{x}_1) & (x_{12} - \bar{x}_2) & \cdots & (x_{1p} - \bar{x}_p) \\ (x_{21} - \bar{x}_1) & (x_{22} - \bar{x}_2) & \cdots & (x_{21} - \bar{x}_p) \\ \vdots & \vdots & \ddots & \vdots \\ (x_{n1} - \bar{x}_1) & (x_{n2} - \bar{x}_2) & \cdots & (x_{np} - \bar{x}_p) \end{bmatrix}$$

$$= \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & \cdots & Cov(X_1, X_p) \\ Cov(X_2, X_1) & Var(X_2) & \cdots & Cov(X_2, X_p) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(X_p, X_1) & Cov(X_p, X_2) & \cdots & Var(X_p) \end{bmatrix}$$



4. 상관관계 및 상관계수

▣ 예제

- 다음의 8개 샘플이 주어진 상황에서 평균 벡터와 공분산 행렬 계산

학생	$\mathbf{x} = (x_1, x_2, x_3)^T$ (x_1 : 나이, x_2 : 몸무게, x_3 : 학점)
1	$\mathbf{x}_1 = (170, 60, 4.1)^T$
2	$\mathbf{x}_2 = (165, 55, 3.0)^T$
3	$\mathbf{x}_3 = (174, 75, 2.8)^T$
4	$\mathbf{x}_4 = (169, 67, 2.9)^T$
5	$\mathbf{x}_5 = (155, 49, 3.1)^T$
6	$\mathbf{x}_6 = (172, 63, 3.6)^T$
7	$\mathbf{x}_7 = (166, 58, 3.7)^T$
8	$\mathbf{x}_8 = (168, 61, 4.0)^T$

$$\mu = (167.375, \ 61.0, \ 3.4)^T$$

$$\Sigma = \begin{pmatrix} 33.696 & 39.429 & 0.371 \\ 39.429 & 60.857 & -0.943 \\ 0.371 & -0.943 & 0.263 \end{pmatrix}$$

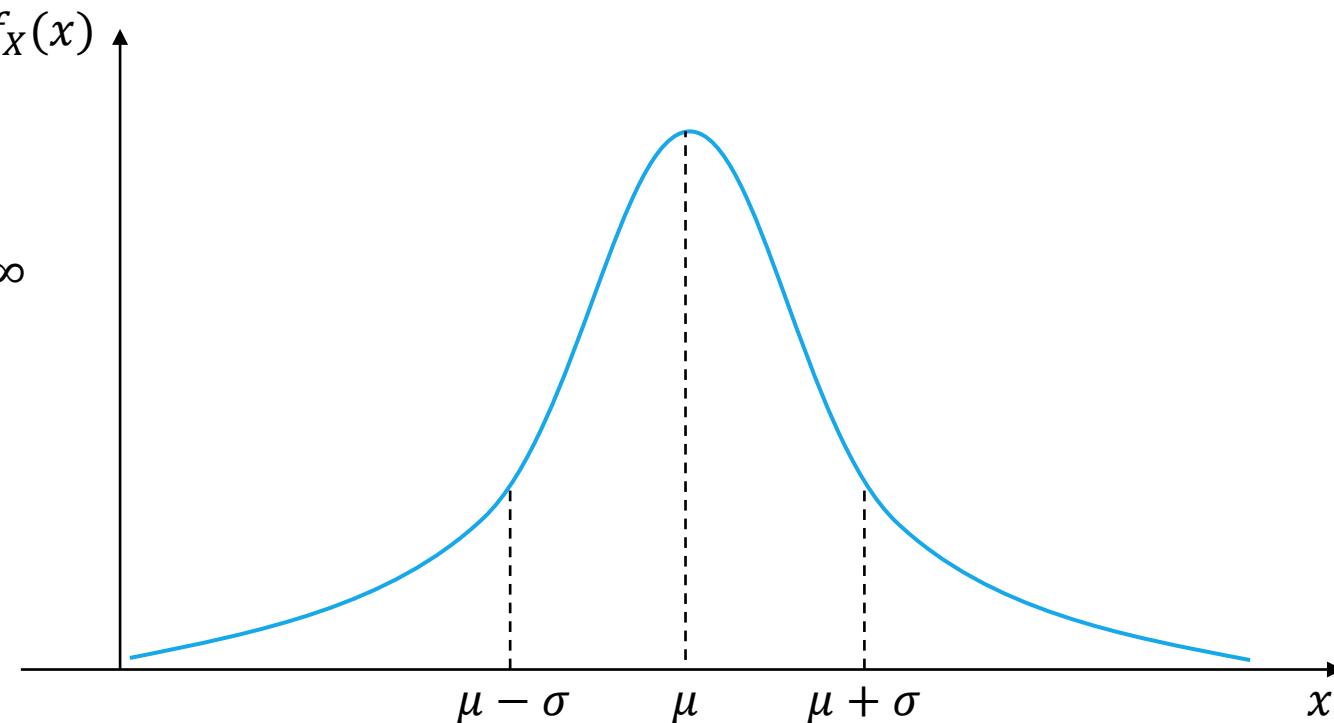
5. 정규분포 (normal distribution)

■ $N(\mu, \sigma^2)$

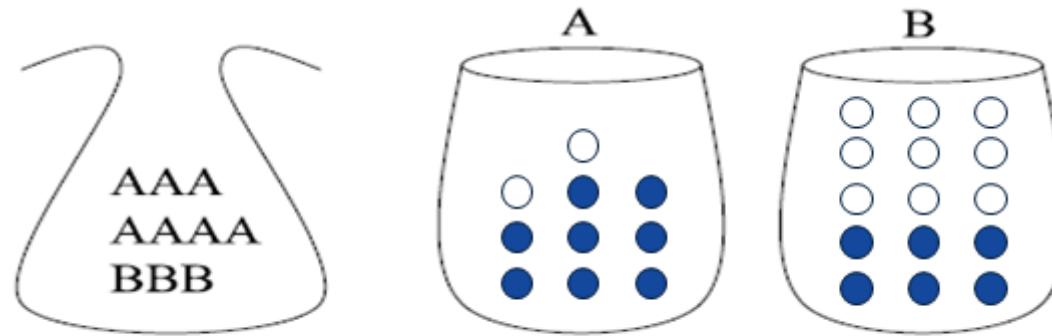
- ✓ 평균과 분산으로 분포를 설명
- ✓ Normal/Gaussian distribution

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}, -\infty < x < \infty$$

$$E(X) = \mu, \quad Var(X) = \sigma^2$$



▣ 확률 실험



확률 실험 장치

- ✓ 주머니에서 카드를 뽑아 상자를 선택하고 선택된 상자에서 공을 뽑아 관찰
- ✓ 랜덤 변수 $X \in \{A, B\}$, $Y = \{\text{파랑}, \text{하양}\}$



6. 확률 실험

▣ 확률

- ✓ 상자 A가 선택될 확률은?

$$P(X = A) = P(A) = 7/10$$

사전확률

$$P(A, \text{하양}) = P(\text{하양}|A)P(A) = (2/10)(7/10) = 7/50$$

- ✓ 상자 A에서 하얀 공이 뽑힐 확률은?

조건부 확률

$$P(Y = \text{하양}|X = A) = P(\text{하양}|A) = 2/10$$

- ✓ 상자는 A이고 공은 하양이 뽑힐 확률은?

결합확률

$$P(A, \text{하양}) = P(\text{하양}|A)P(A) = (2/10)(7/10) = 7/50$$

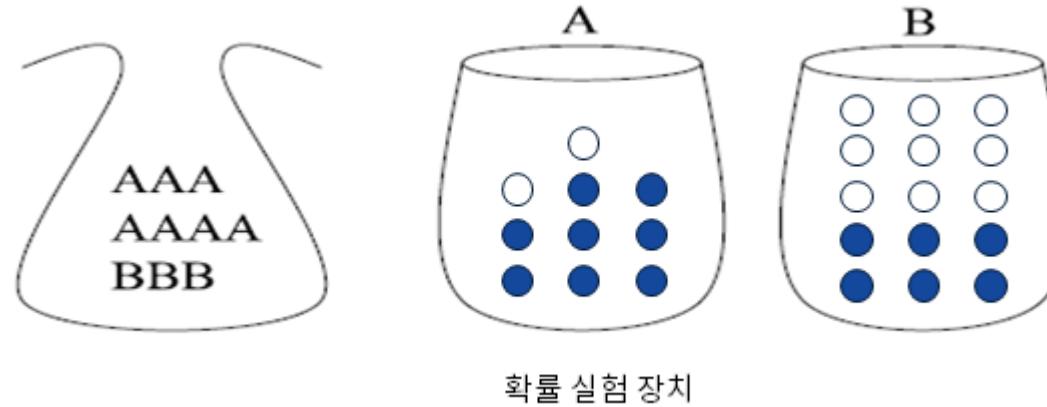
- ✓ 하얀 공이 나올 확률은?

주변확률

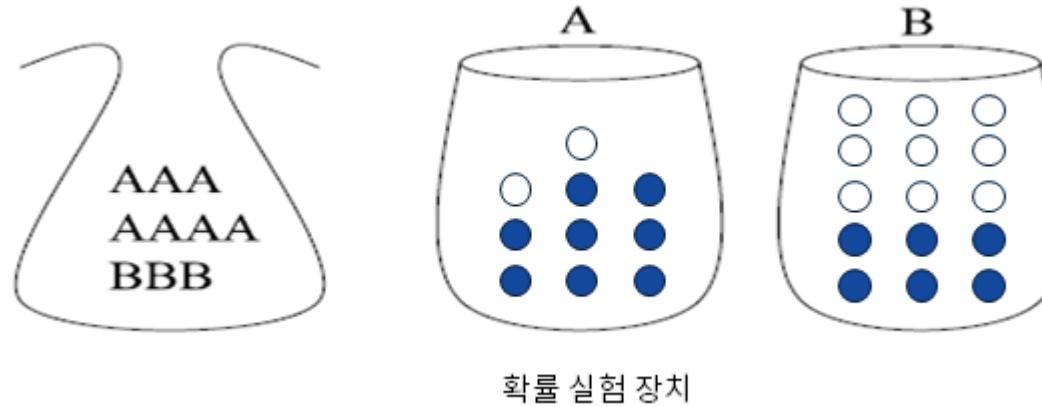
$$\begin{aligned}P(\text{하양}) &= P(\text{하양}|A)P(A) + P(\text{하양}|B)P(B) \\&= (2/10)(7/10) + (9/15)(3/10) = 8/25\end{aligned}$$

6. 확률 실험

▣ 문제: 하얀 공이 뽑혔는데 어느 상자에서 나왔는지 맞추어라.

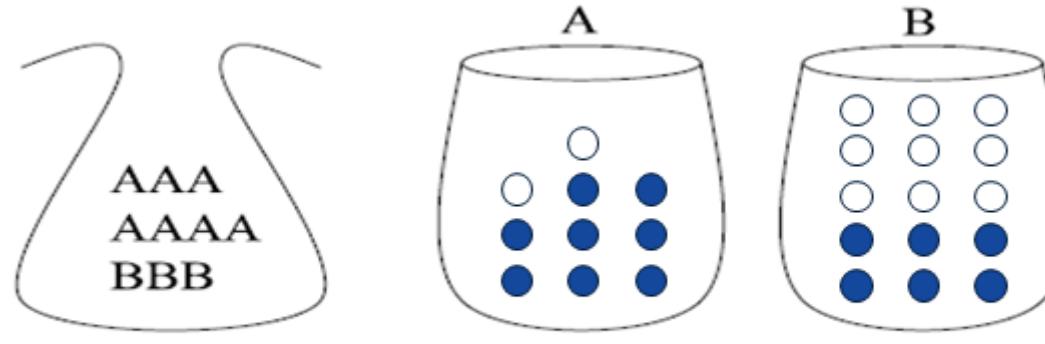


- ✓ 기본 전략: 상자 A와 B에서 나왔을 가능성 각각을 구하고 큰 가능성을 보인 상자를 답으로 취한다.
- ✓ 이렇게 해야 맞출 가능성이 최대 (오류 범할 가능성이 최소)가 됨
- ✓ 가능성은 어떻게 계산?

 생각 1

- ✓ 상자 A의 하얀 공 확률과 상자 B의 하얀 공 확률을 비교하여 큰 쪽을 취한다.
- ✓ $P(\text{하양}|B) = 9/15 > P(\text{하양}|A) = 2/10$ 이므로 ‘상자 B에서 나왔다’고 말함
- ✓ 조건부 확률 $P(Y|X)P(X)$ 를 사용한 셈이다. 타당한가?
- ✓ 이 조건부 확률을 우도 (likelihood) 라고 부름

▣ 생각 2



- ✓ 상자 A와 상자 B의 선택 가능성을 비교하여 큰 쪽을 취한다.
- ✓ $P(A)=7/10 > P(B)=3/10$ 이므로 ‘상자 A에서 나왔다’고 말함
- ✓ 사전 확률 $P(X)$ 를 사용한 셈이다. 타당한가?



6. 확률 실험

▣ 생각 1과 생각 2의 한계

- ✓ 극단적으로 $P(A)=0.999$ 라면 생각 1이 틀린 것이 확실하다.
- ✓ 극단적으로 $P(\text{하양}|A)=0.999$ 라면 생각 2가 틀린 것이 확실하다.
- ✓ 우도와 사전 확률을 모두 고려함이 타당해 보임

▣ 문제에 충실하자 !!!

- ✓ 조건부 확률 $P(A|\text{하양})$ 과 $P(B|\text{하양})$ 을 비교하여 큰 쪽을 취함
- ✓ 즉 $P(X|Y)$ 를 사용하겠다는 생각이 타당하다.
- ✓ $P(X|Y)$ 를 사후 확률 ((posterior probability)이라 함
- ✓ 어떻게 계산할 것인가?

Bayes rule 을 사용

정리하기

- ▶ 확률 변수와 확률 분포의 개념 이해
- ▶ 평균, 분산, 공분산 행렬, 상관관계 계수

정리하기

▶ 사전확률, 우도함수, 사후확률의 개념

▶ 데이터로부터 평균과 분산 계산