

2017 2학기 인공지능 강의노트

7주차

강사: 양일호

강의 계획

주차	강의	실습
1	과목 소개 인공지능 및 python 소개 Python 프로그래밍 환경 조성 및 기본 문법	Python 개발 환경 구축 간단한 expert system 구현
2	Search (Blind Search, Heuristic Search)	DFS, BFS, A* 구현
3	Learning / Simulated Annealing	Simulated Annealing 구현
4	Genetic Algorithm	Genetic Algorithm 구현
5	보강 주간 (개천절 / 추석 연휴)	
6	Neural Network	Perceptron 구현
7	Neural Network	Single-Layer Perceptron 구현
8	Neural Network	Multi-Layer Perceptron 구현
9	Deep Neural Network	DNN 구현
10	Deep Neural Network	DNN 구현
11	DNN 심화 (혹은 Particle Swarm Optimization)	관련 내용 구현
12	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
13	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
14	DNN 심화 (혹은 Bayesian Networks)	관련 내용 구현
15	DNN 심화 (혹은 Decision Networks)	관련 내용 구현
16	기말고사	

Python
숙달을 위한
준비 기간

예비 기간
(대체 가능)

Perceptron 구조

- Perceptron의 출력

입력 특징 벡터: $\vec{X} = \begin{bmatrix} X_0 \\ X_1 \\ \dots \\ X_m \end{bmatrix}$ 파라미터 (weight+bias): $\vec{W} = \begin{bmatrix} W_0 \\ W_1 \\ \dots \\ W_m \end{bmatrix}$

$$o = g(z) = g\left(\sum_{i=0}^m w_i x_i\right) = g(\vec{W} \cdot \vec{X}) = g(\vec{W}^T \vec{X})$$

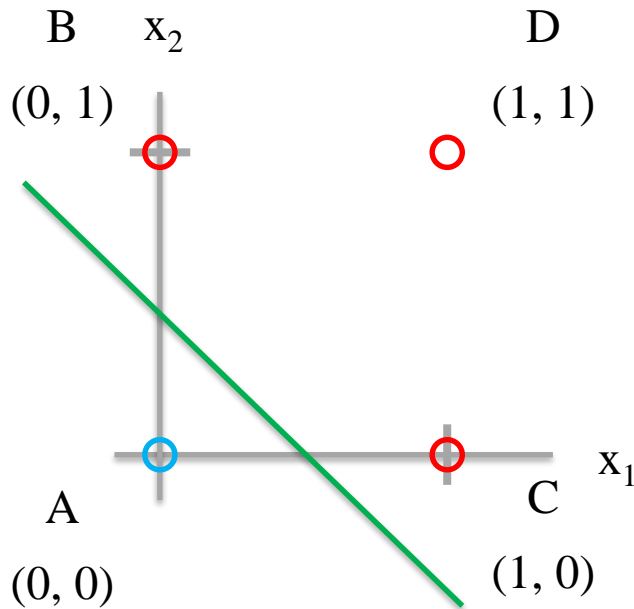
$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

$$z = \sum_{i=0}^m w_i x_i$$



Perceptron 분류 예시

- OR 게이트의 기능을 하는 perceptron

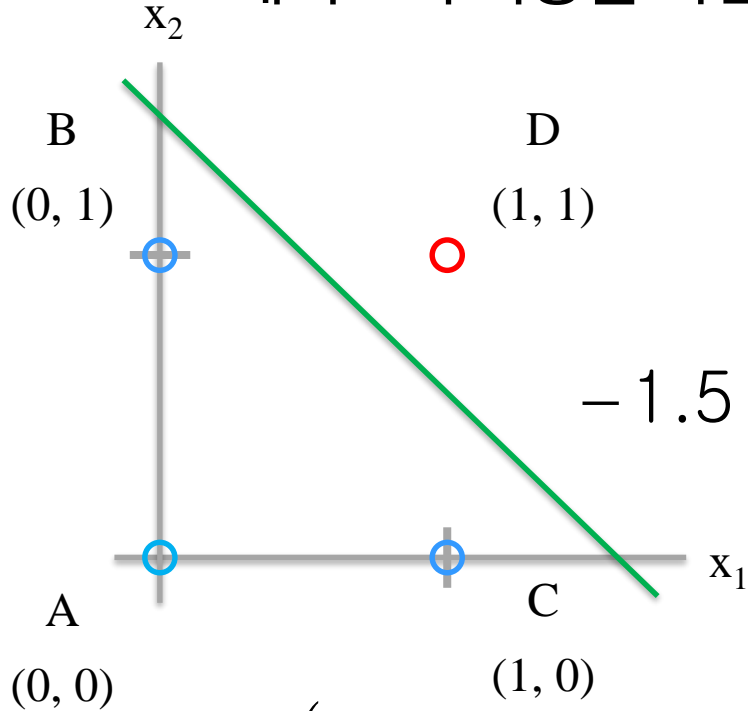


$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

$$\begin{aligned} \text{A: } o &= g(-0.5 + x_1 + x_2) = g(-0.5 + 0 + 0) = g(-0.5) = 0 \\ \text{B: } o &= g(-0.5 + x_1 + x_2) = g(-0.5 + 0 + 1) = g(0.5) = 1 \\ \text{C: } o &= g(-0.5 + x_1 + x_2) = g(-0.5 + 1 + 0) = g(0.5) = 1 \\ \text{D: } o &= g(-0.5 + x_1 + x_2) = g(-0.5 + 1 + 1) = g(1.5) = 1 \end{aligned}$$

Perceptron 분류 예시

- AND 게이트의 기능을 하는 perceptron



$$-1.5 + x_1 + x_2 = 0$$

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

$$A: o = g(-1.5 + x_1 + x_2) = g(-1.5 + 0 + 0) = g(-1.5) = 0$$

$$B: o = g(-1.5 + x_1 + x_2) = g(-1.5 + 0 + 1) = g(-0.5) = 0$$

$$C: o = g(-1.5 + x_1 + x_2) = g(-1.5 + 1 + 0) = g(-0.5) = 0$$

$$D: o = g(-1.5 + x_1 + x_2) = g(-1.5 + 1 + 1) = g(0.5) = 1$$

Perceptron learning rule

학습 반복 횟수: $t = 0$

perceptron의 파라미터(weight+bias) $\vec{w}(t)$ 랜덤 초기화

학습 데이터세트(특징 벡터, 정답 레이블): $\{(\vec{x}_1, d_1), (\vec{x}_2, d_2), \dots, (\vec{x}_n, d_n)\}$

for 충분히 오류가 감소할 때까지 반복 학습:

for 각 학습 데이터 (\vec{x}_k, d_k) 에 대해서 ($1 \leq k \leq n$):

$$o_k = g(\vec{w}(t)^T \vec{x}_k)$$

for 각 차원의 파라미터 $w_i(t)$ 에 대해서 ($0 \leq i \leq m$):

$$w_i(t+1) = w_i(t) + (d_k - o_k)x_{k,i}$$

$$t = t + 1$$

학습 종료

m : 입력 특징의 차원

$x_{k,i}$: k번째 학습 특징 벡터의 i번째 원소

$w_i(t)$: t번째 반복 학습한 파라미터의 i번째 원소

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Perceptron learning rule

- 학습이 제대로 되고 있는지 확인하는 방법

...

62-th training, errorRate=50.0% (50/100)

63-th training, errorRate=50.0% (50/100)

64-th training, errorRate=50.0% (50/100)

65-th training, errorRate=50.0% (50/100)

66-th training, errorRate=50.0% (50/100)

67-th training, errorRate=50.0% (50/100)

68-th training, errorRate=50.0% (50/100)

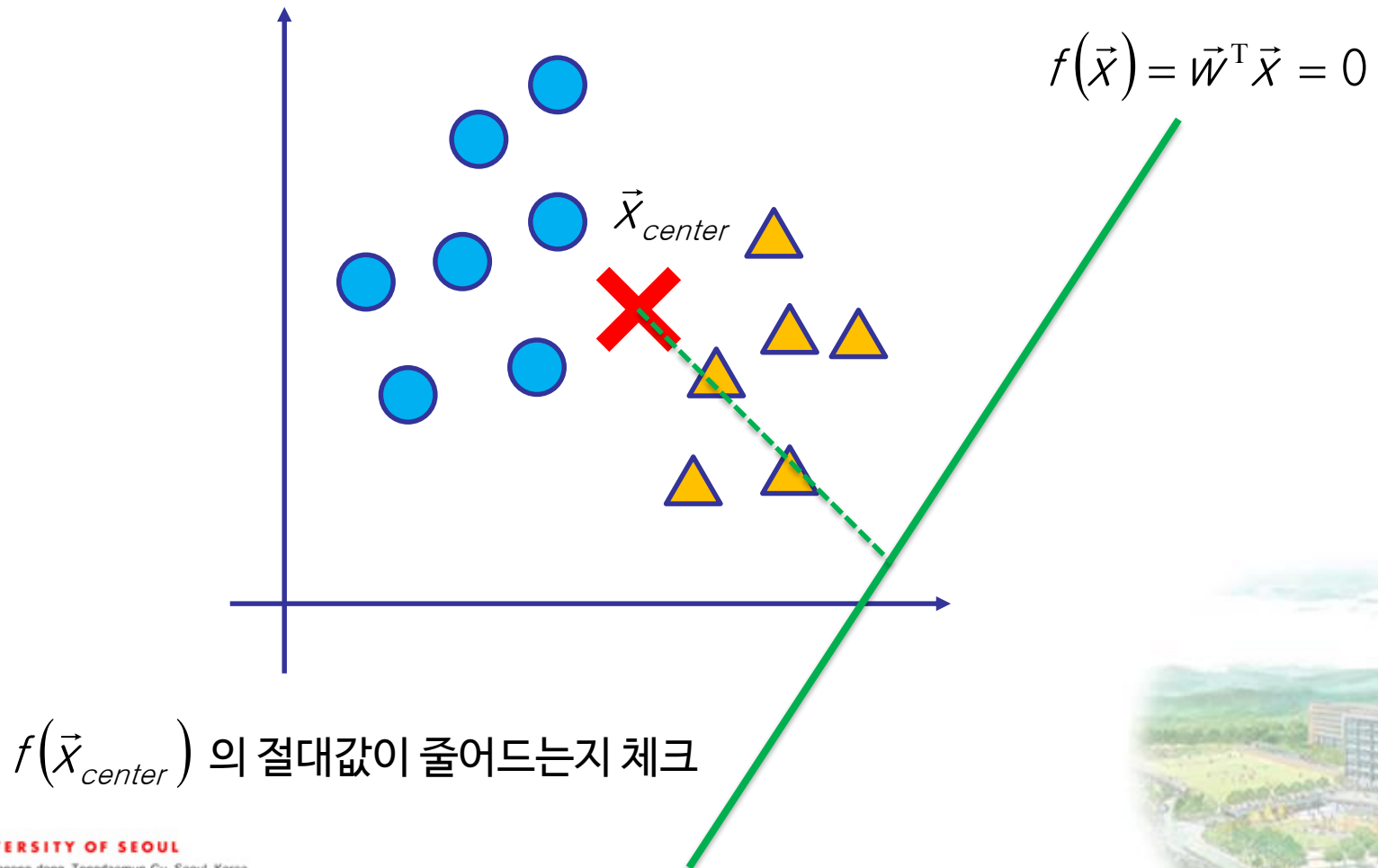
69-th training, errorRate=50.0% (50/100)

...



Perceptron learning rule

- 학습이 제대로 되고 있는지 확인하는 방법
 - 데이터 중심점과 선형 분류기와의 대략적인 거리 체크(임시 방법)



Perceptron learning rule

- 학습이 제대로 안되는 경우

```
# 선형 분류(step function)
def predictClass(paramArr, featArr):
    res = calcClassifierResult(paramArr, featArr)

    # class 1
    if res < 0:
        return 0
    # class 2
    return 1
```

왜 학습이 안되는가?



Perceptron learning rule

- Activation이 달라진 것

```
# 선형 분류(step function)
def predictClass(paramArr, featArr):
    res = calcClassifierResult(paramArr, featArr)

    # class 1
    if res < 0:
        return 0
    # class 2
    return 1
```

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$



$$g(z) = \begin{cases} 0 & z > 0 \\ 1 & z \leq 0 \end{cases}$$



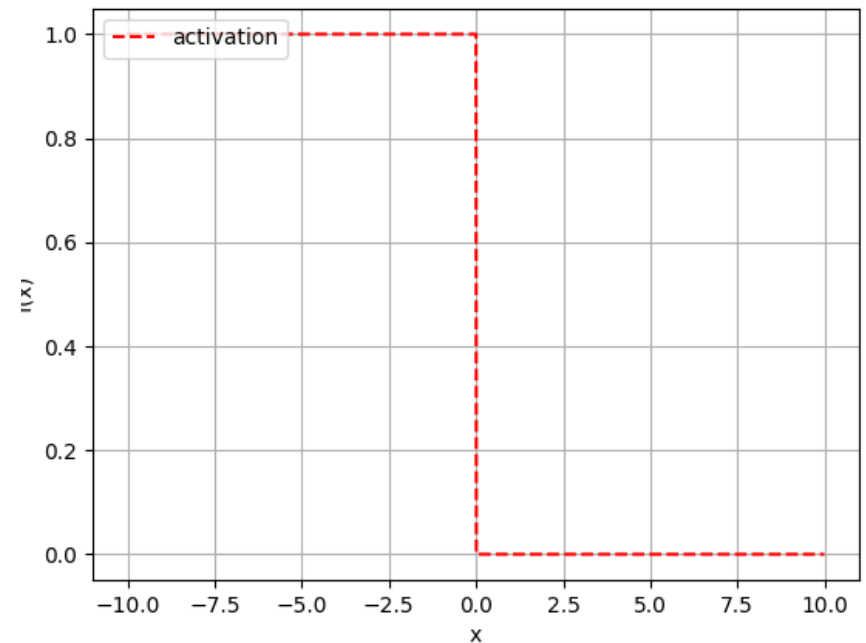
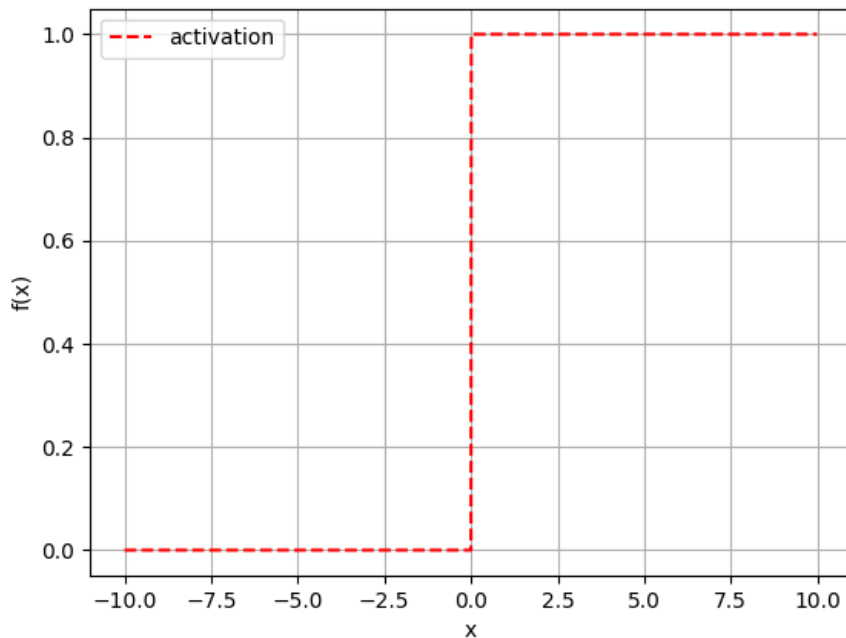
Perceptron learning rule

- Activation이 달라진 것

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



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



Perceptron learning rule

학습 반복 횟수: $t = 0$

perceptron의 파라미터(weight+bias) $\vec{w}(t)$ 랜덤 초기화

학습 데이터세트(특징 벡터, 정답 레이블): $\{(\vec{x}_1, d_1), (\vec{x}_2, d_2), \dots, (\vec{x}_n, d_n)\}$

for 충분히 오류가 감소할 때까지 반복 학습:

for 각 학습 데이터 (\vec{x}_k, d_k) 에 대해서 ($1 \leq k \leq n$):

$$o_k = g(\vec{w}(t)^T \vec{x}_k)$$

for 각 차원의 파라미터 $w_i(t)$ 에 대해서 ($0 \leq i \leq m$):

$$w_i(t+1) = w_i(t) - (d_k - o_k)x_{k,i}$$

$$t = t + 1$$

학습 종료

굳이 activation function을 바꿔야겠다면...

m : 입력 특징의 차원

$x_{k,i}$: k번째 학습 특징 벡터의 i번째 원소

$w_i(t)$: t번째 반복 학습한 파라미터의 i번째 원소

$$g(z) = \begin{cases} 0 & z > 0 \\ 1 & z \leq 0 \end{cases}$$

실습과제 (1 / 2) 5점

- 학습 과정

- 다음 학습용 2차원 어종 데이터를 파일로부터 읽기
 - “salmon_train.txt” / “seabass_train.txt”
- 선형 분류기로 이를 분류하고 학습 데이터에 대한 오류율 계산
 - Perceptron learning rule을 이용하여 반복 학습
 - Learning rate를 부여하도록 수정한 버전으로 학습
 - 다음 항목들을 적절히 설계하여 구현
 - Perceptron 파라미터 초기값 부여 방법
 - 학습 중단 판정 기준
 - Learning rate
 - 매 반복마다 오류율 콘솔 출력 + 텍스트 파일로 저장
 - “train_log_[learning_rate].txt”



실습과제 (2 / 2)

• 테스트 과정

- 다음 테스트용 2차원 어종 데이터를 파일로부터 읽고 분류
 - “salmon_test.txt” / “seabass_test.txt”
 - 앞에서 학습한 선형 분류기로 분류할 것
 - 학습 데이터에 대한 분류 오류율이 가장 낮은 파라미터 사용
 - 분류 결과를 출력 + 텍스트 파일로 저장
 - “test_output_[learning_rate].txt”

• 제출 형식

- 메인 소스코드에서 command line argument로 다음 항목 입력 받기
 - Learning rate
- 3가지 이상의 command line argument를 구성하여 배치 파일 동봉
 - “run_exp.bat”
- 입력 데이터 파일 + 출력 파일(학습 로그, 테스트 결과)과 함께 제출



강의개요

- 강의
 - Artificial neural network
- 실습
 - Single layer perceptron 구현



강의

강의 계획

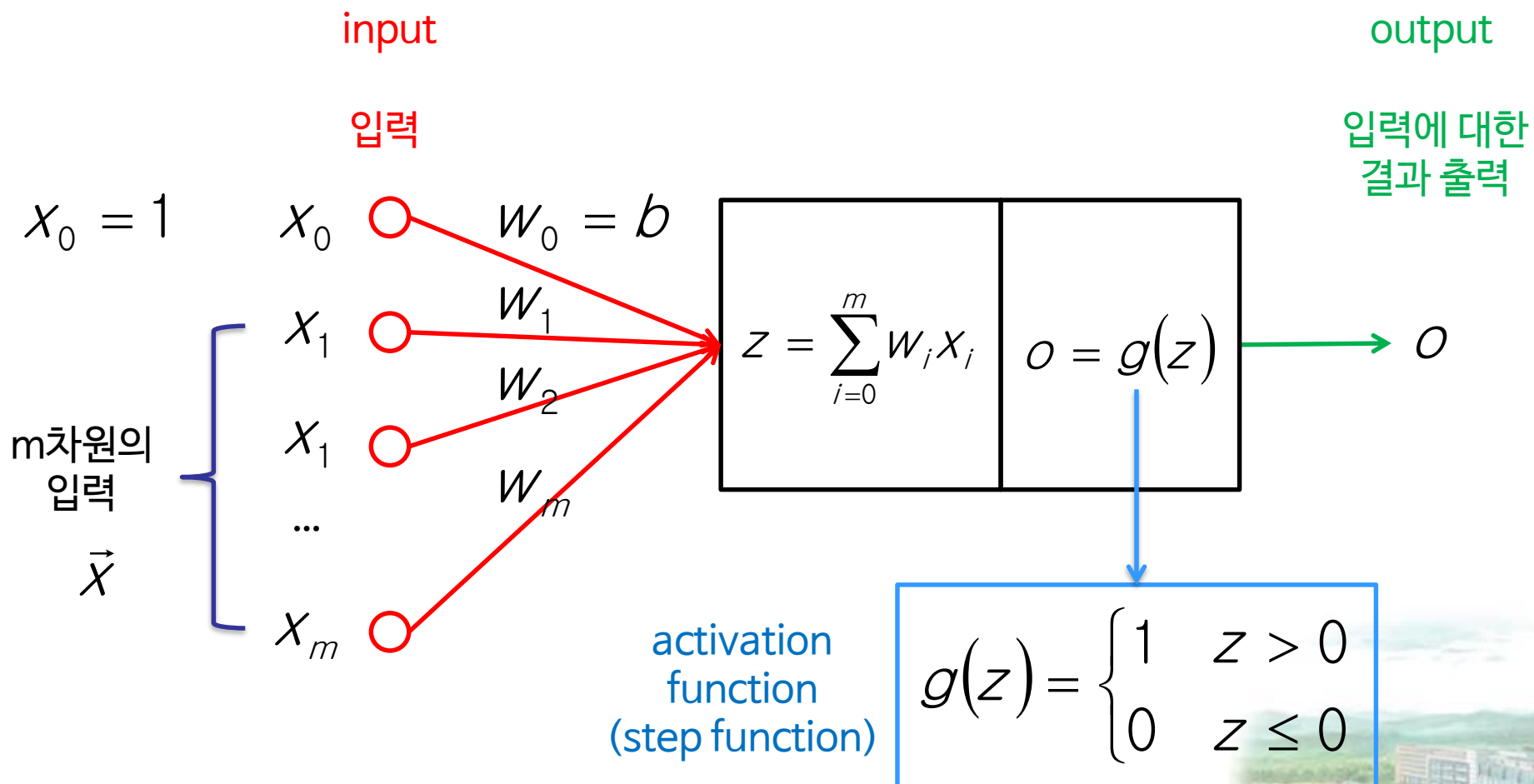
주차	강의	실습
1	과목 소개 인공지능 및 python 소개 Python 프로그래밍 환경 조성 및 기본 문법	Python 개발 환경 구축 간단한 expert system 구현
2	Search (Blind Search, Heuristic Search)	DFS, BFS, A* 구현
3	Learning / Simulated Annealing	Simulated Annealing 구현
4	Genetic Algorithm	Genetic Algorithm 구현
5	보강 주간 (개천절 / 추석 연휴)	
6	Neural Network	Perceptron 구현
7	Neural Network	Single-Layer Perceptron 구현
8	Neural Network	Multi-Layer Perceptron 구현
9	Deep Neural Network	DNN 구현
10	Deep Neural Network	DNN 구현
11	DNN 심화 (혹은 Particle Swarm Optimization)	관련 내용 구현
12	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
13	DNN 심화 (혹은 Gaussian Mixture Model)	관련 내용 구현
14	DNN 심화 (혹은 Bayesian Networks)	관련 내용 구현
15	DNN 심화 (혹은 Decision Networks)	관련 내용 구현
16	기말고사	

Python
숙달을 위한
준비 기간

예비 기간
(대체 가능)

Perceptron 구조

- 개략적인 도식



Perceptron learning rule

학습 반복 횟수: $t = 0$

perceptron의 파라미터(weight+bias) $\vec{w}(t)$ 랜덤 초기화

학습 데이터세트(특징 벡터, 정답 레이블): $\{(\vec{x}_1, d_1), (\vec{x}_2, d_2), \dots, (\vec{x}_n, d_n)\}$

for 충분히 오류가 감소할 때까지 반복 학습:

for 각 학습 데이터 (\vec{x}_k, d_k) 에 대해서 ($1 \leq k \leq n$):

$$o_k = g(\vec{w}(t)^T \vec{x}_k)$$

for 각 차원의 파라미터 $w_i(t)$ 에 대해서 ($0 \leq i \leq m$):

$$w_i(t+1) = w_i(t) + \underline{(d_k - o_k)} x_{k,i}$$

$$t = t + 1$$

실제 출력이 얼마나 틀렸는가?

원하는 출력(정답 레이블)과 실제 출력의 차이

학습 종료

m : 입력 특징의 차원

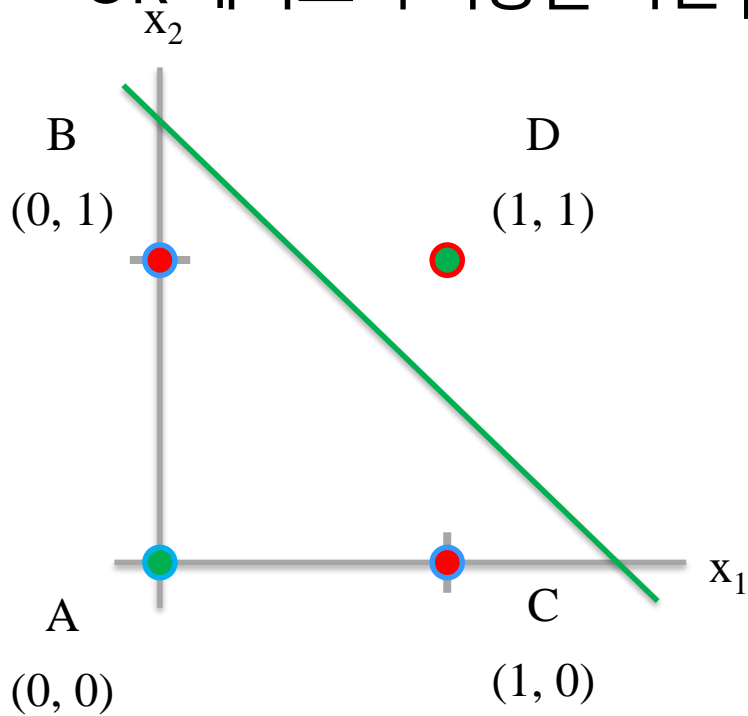
$x_{k,i}$: k번째 학습 특징 벡터의 i번째 원소

$w_i(t)$: t번째 반복 학습한 파라미터의 i번째 원소

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$

Perceptron 학습 예시

- OR 게이트의 기능을 하는 perceptron



$$\vec{w}(t=0) = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$

학습 데이터세트:

$$\left\{ \left(\vec{x}_A = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, d_A = 0 \right), \left(\vec{x}_B = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, d_B = 1 \right), \left(\vec{x}_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, d_C = 1 \right), \left(\vec{x}_D = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, d_D = 1 \right) \right\}$$

Perceptron 학습 예시

- OR 게이트의 기능을 하는 perceptron

- t=1

학습 데이터세트:

$$\left\{ \left(\vec{x}_A = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, d_A = 0 \right), \left(\vec{x}_B = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, d_B = 1 \right), \left(\vec{x}_C = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, d_C = 1 \right), \left(\vec{x}_D = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, d_D = 1 \right) \right\}$$

$$\begin{bmatrix} w_0(t+1) \\ w_1(t+1) \\ w_2(t+1) \end{bmatrix} = \begin{bmatrix} w_0(t) \\ w_1(t) \\ w_2(t) \end{bmatrix} + (d_A - o_A) \begin{bmatrix} x_{A,0} \\ x_{A,1} \\ x_{A,2} \end{bmatrix}$$

$$\begin{bmatrix} w_0(t+1) \\ w_1(t+1) \\ w_2(t+1) \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix} + \frac{(0 - g(-1.5 + 0 + 0))}{o_A = 0} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1.5 \\ 1 \\ 1 \end{bmatrix}$$

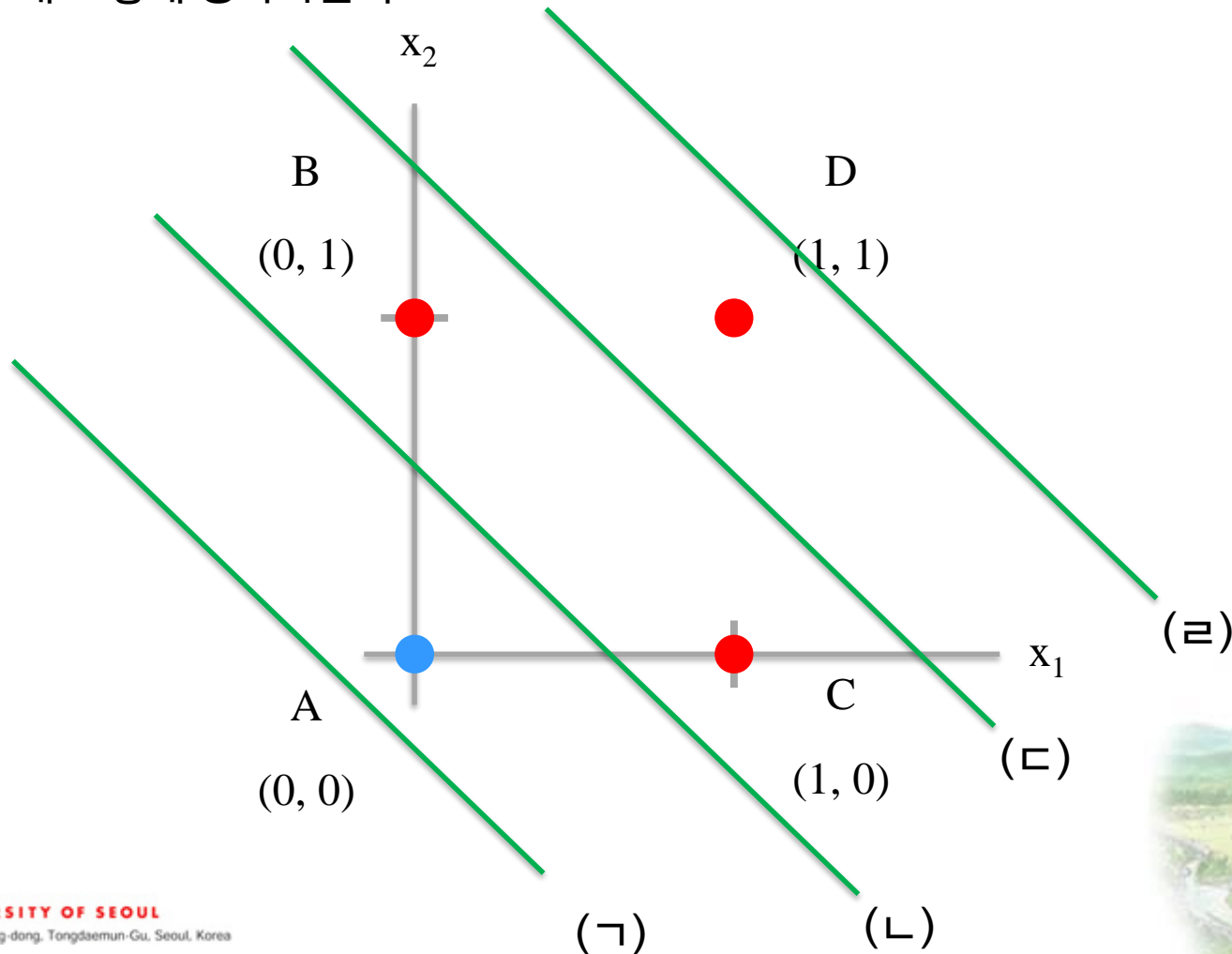
맞으면 0 / 틀리면 +1 혹은 -1

Perceptron 학습 예시

- OR 게이트 학습 문제

- Q) 다음 선형 분류기 중에서 가장 좋은 것은 무엇인가?

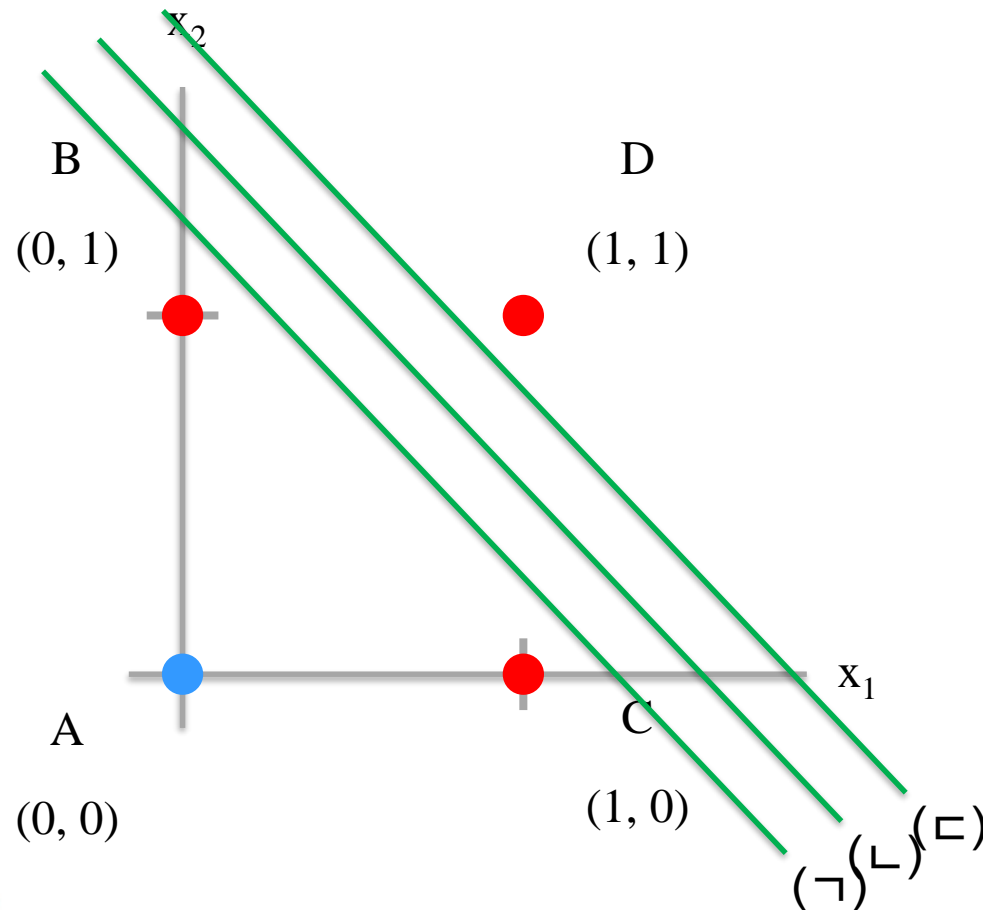
- 왜 그렇게 생각하는가?



Perceptron 학습 예시

- OR 게이트 학습 문제

- Q) 다음 선형 분류기 중에서 가장 좋은 것은 무엇인가?
 - 왜 그렇게 생각하는가?



Cost function의 설계

학습 데이터의 특징(N개): $X = \{x_1, x_2, \dots, x_N\}$

학습 데이터의 레이블(N개): $Y = \{y_1, y_2, \dots, y_N\}$

$f(x)$

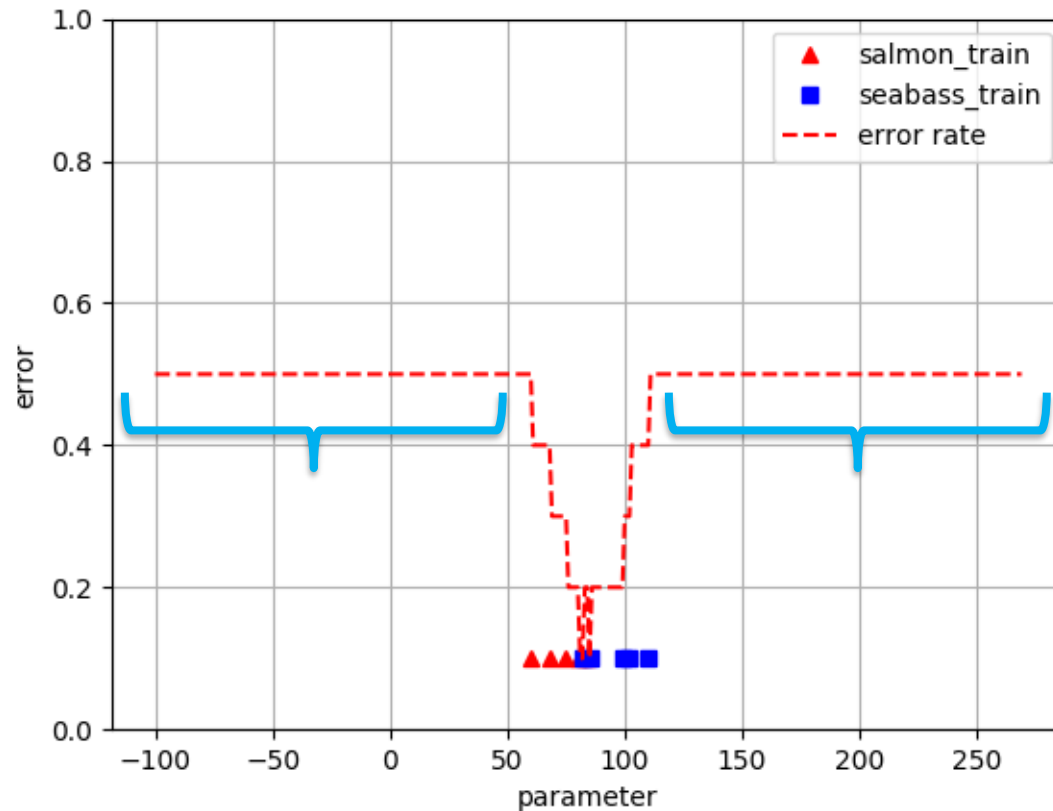
```
def func(x, param):  
    if x < param:  
        return 1  
    return 0
```

$$E = \frac{1}{N} \sum_{k=1}^N |f(x_k) - y_k|$$

X	Y
feature	label
60	1
68	1
80	1
75	1
84	1
99	0
102	0
110	0
85	0
82	0



Cost function의 설계



Cost function의 설계(수정)

학습 데이터의 특징(N개): $X = \{x_1, x_2, \dots, x_N\}$

학습 데이터의 레이블(N개): $Y = \{y_1, y_2, \dots, y_N\}$

$f(x)$

```
def func(x, param):  
    return x - param
```

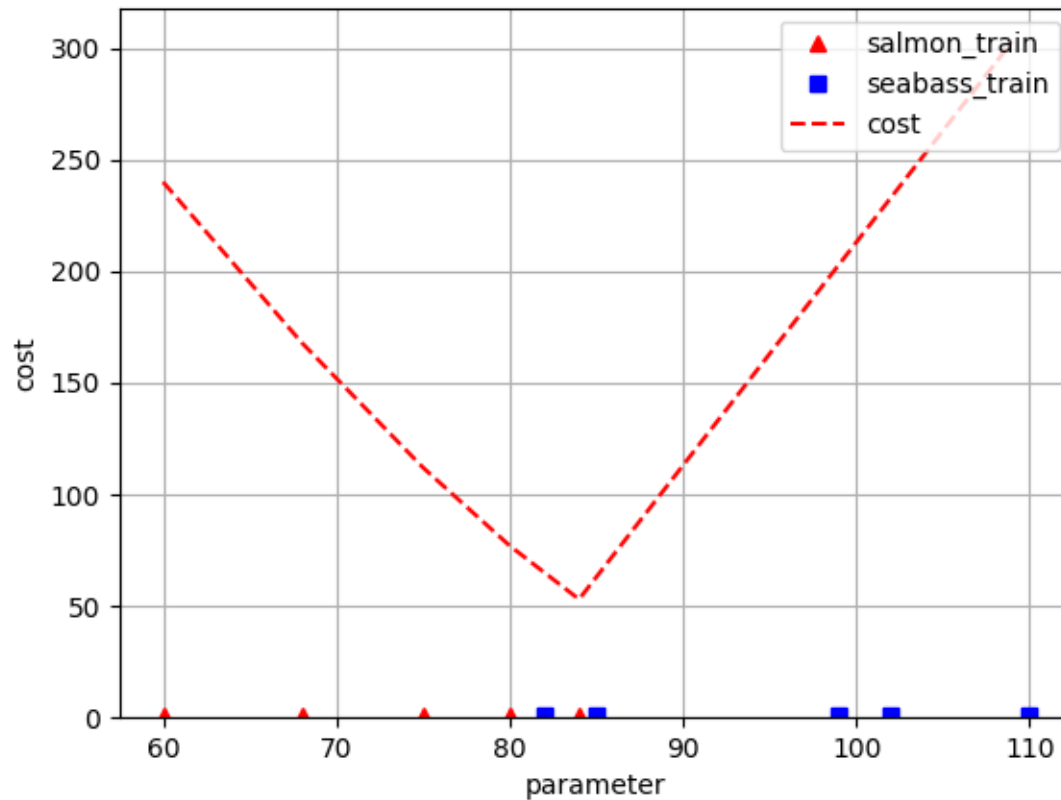
$H(n) = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases}$

```
def unitStep(x):  
    if x < 0:  
        return 0  
    return 1
```

$$E = \frac{1}{N} \sum_{k=1}^N \left\{ |H(f(x_k)) - y_k| \times |f(x_k)| \right\}$$

X	Y
feature	label
60	1
68	1
80	1
75	1
84	1
99	0
102	0
110	0
85	0
82	0

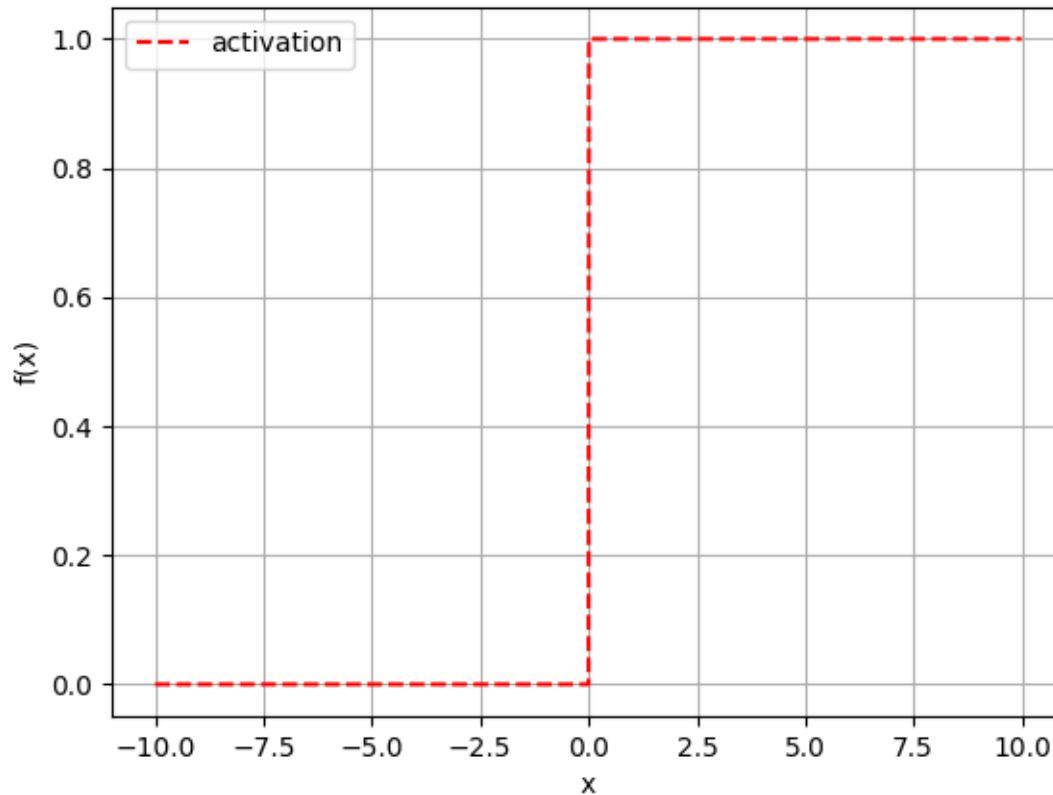
Cost function의 설계(수정)



Activation function

- Step function

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



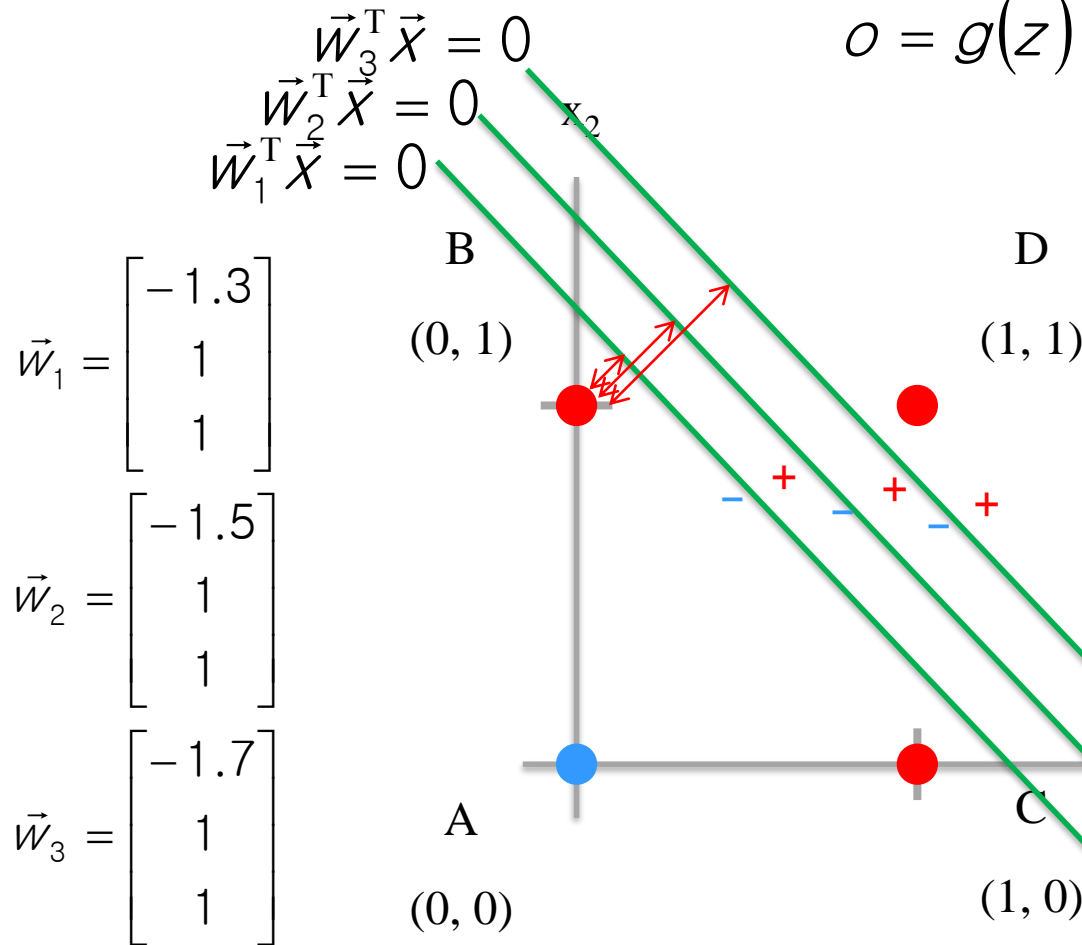
Activation function에 따른 오차

• OR 게이트 학습 문제

$$z = \vec{w}^T \vec{x}$$

$$o = g(z)$$

$$g(z) = \begin{cases} 1 & z > 0 \\ 0 & z \leq 0 \end{cases}$$



$$d_B = 1$$

$g(\vec{w}_1^T \vec{x}_B) = 0$	$d_B - o_B = 1$
$g(\vec{w}_2^T \vec{x}_B) = 0$	$d_B - o_B = 1$
$g(\vec{w}_3^T \vec{x}_B) = 0$	$d_B - o_B = 1$

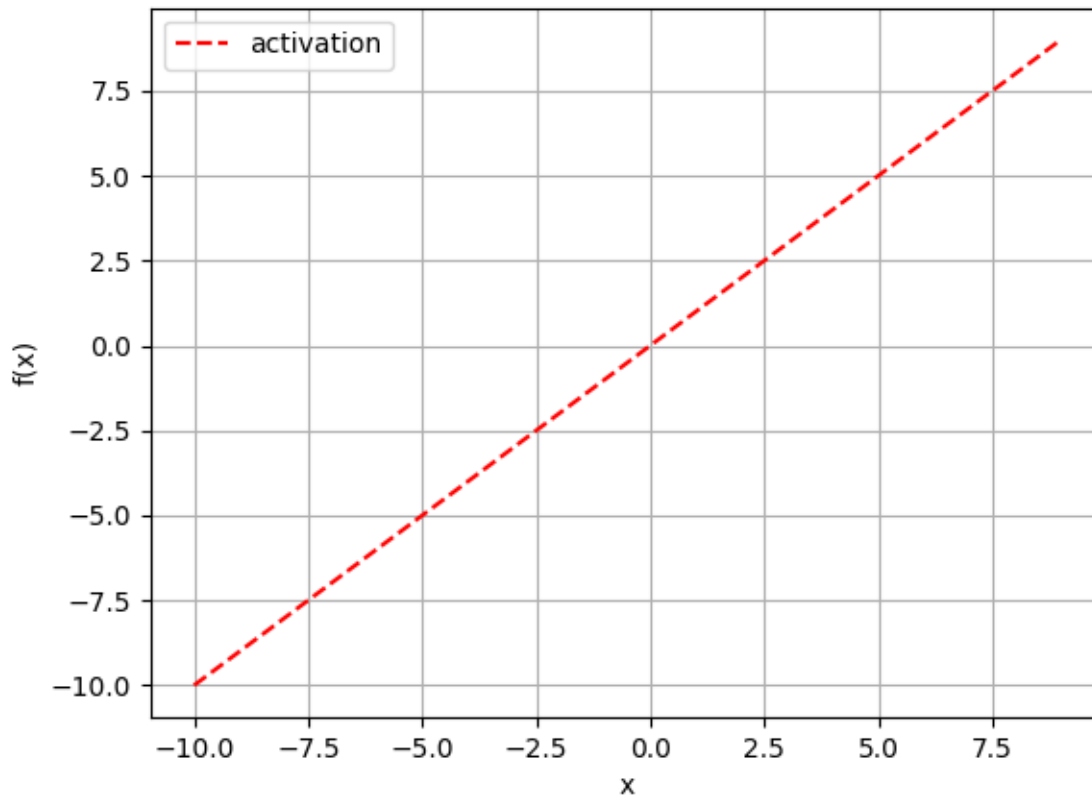
더 좋은 분류기는
오차를 더 적게 하고 싶음



Activation function

- Linear function

$$f(x) = x$$



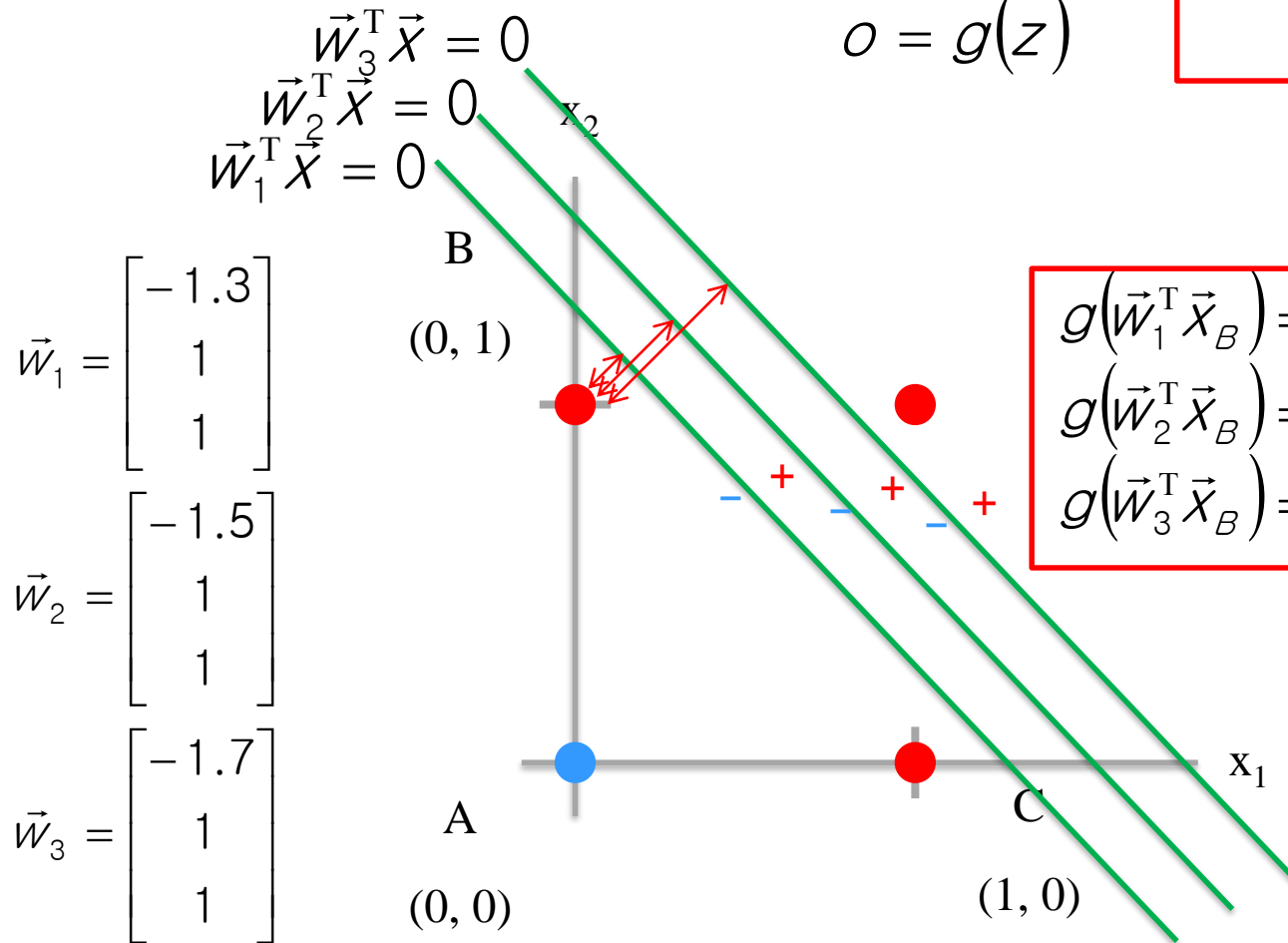
Activation function에 따른 오차

- OR 게이트 학습 문제

$$z = \vec{w}^T \vec{x}$$

$$o = g(z)$$

$$g(z) = z$$



$$g(\vec{w}_1^T \vec{x}_B) = -0.3$$

$$g(\vec{w}_2^T \vec{x}_B) = -0.5$$

$$g(\vec{w}_3^T \vec{x}_B) = -0.7$$

$$d_B = 1$$

$$d_B - o_B = 1.3$$

$$d_B - o_B = 1.5$$

$$d_B - o_B = 1.7$$

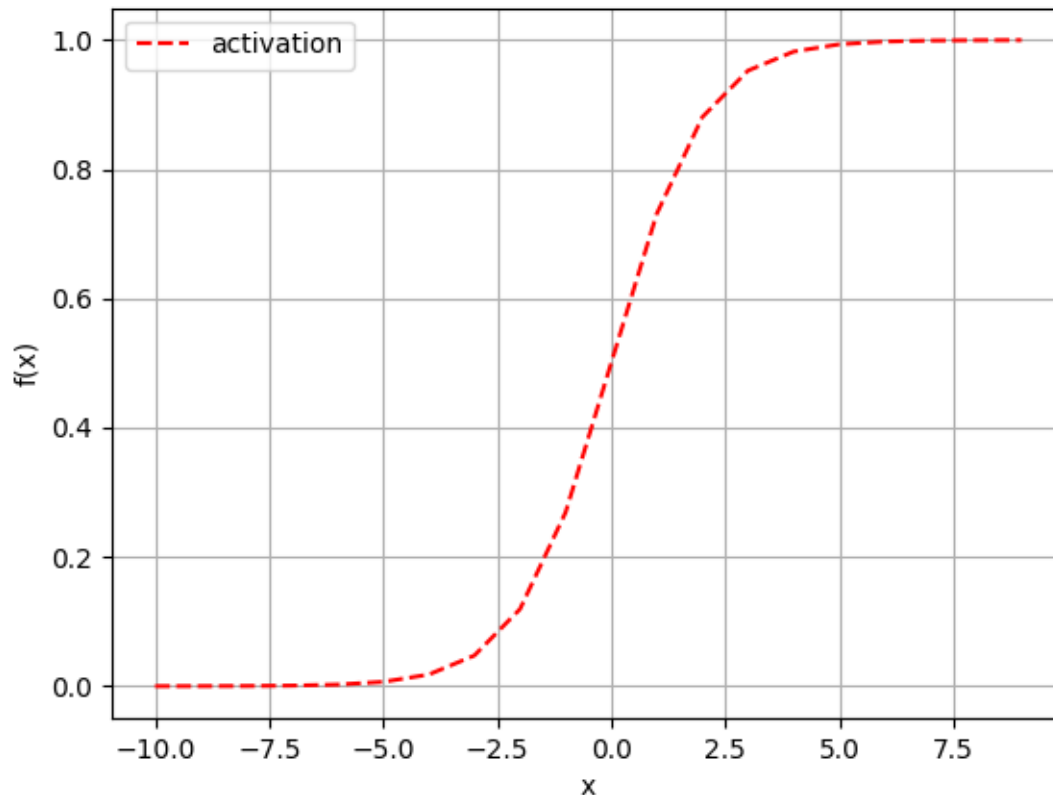
오차는 -1 ~ +1 사이로
계산하고 싶음



Activation function

- Sigmoid function (logistic function)

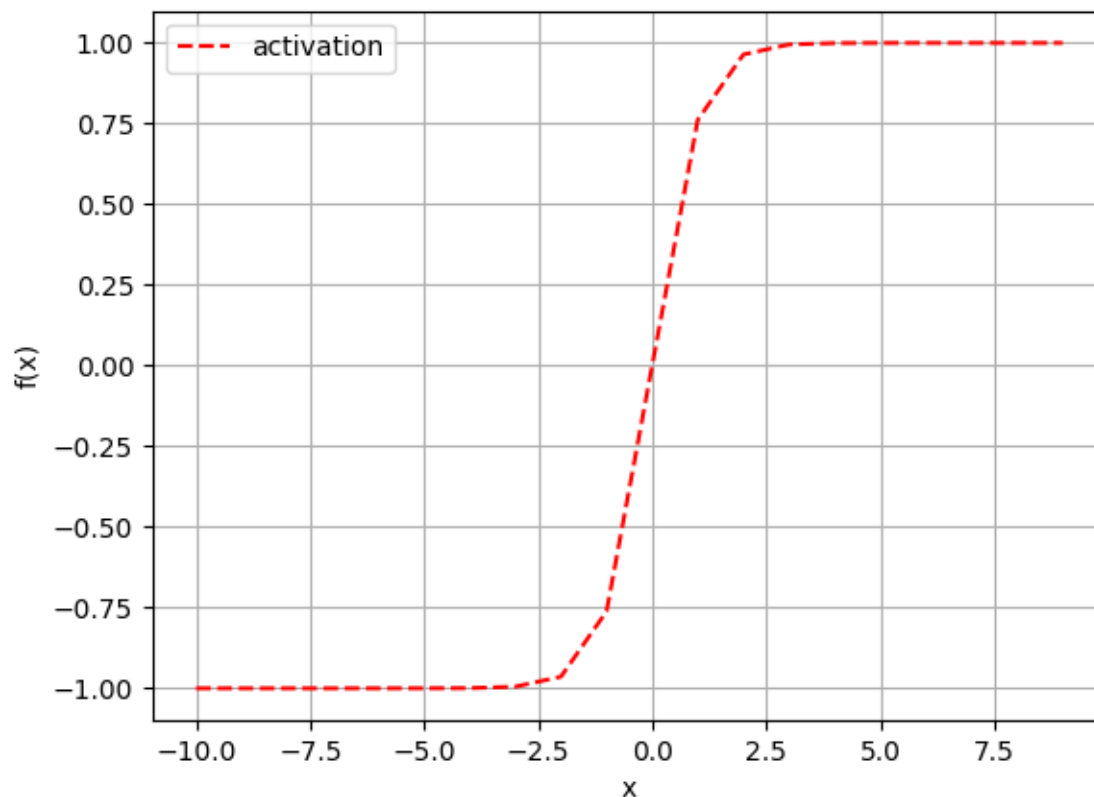
$$f(x) = \frac{1}{1 + \exp(-x)}$$



Activation function

- Sigmoid function (hyperbolic tangent function)

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



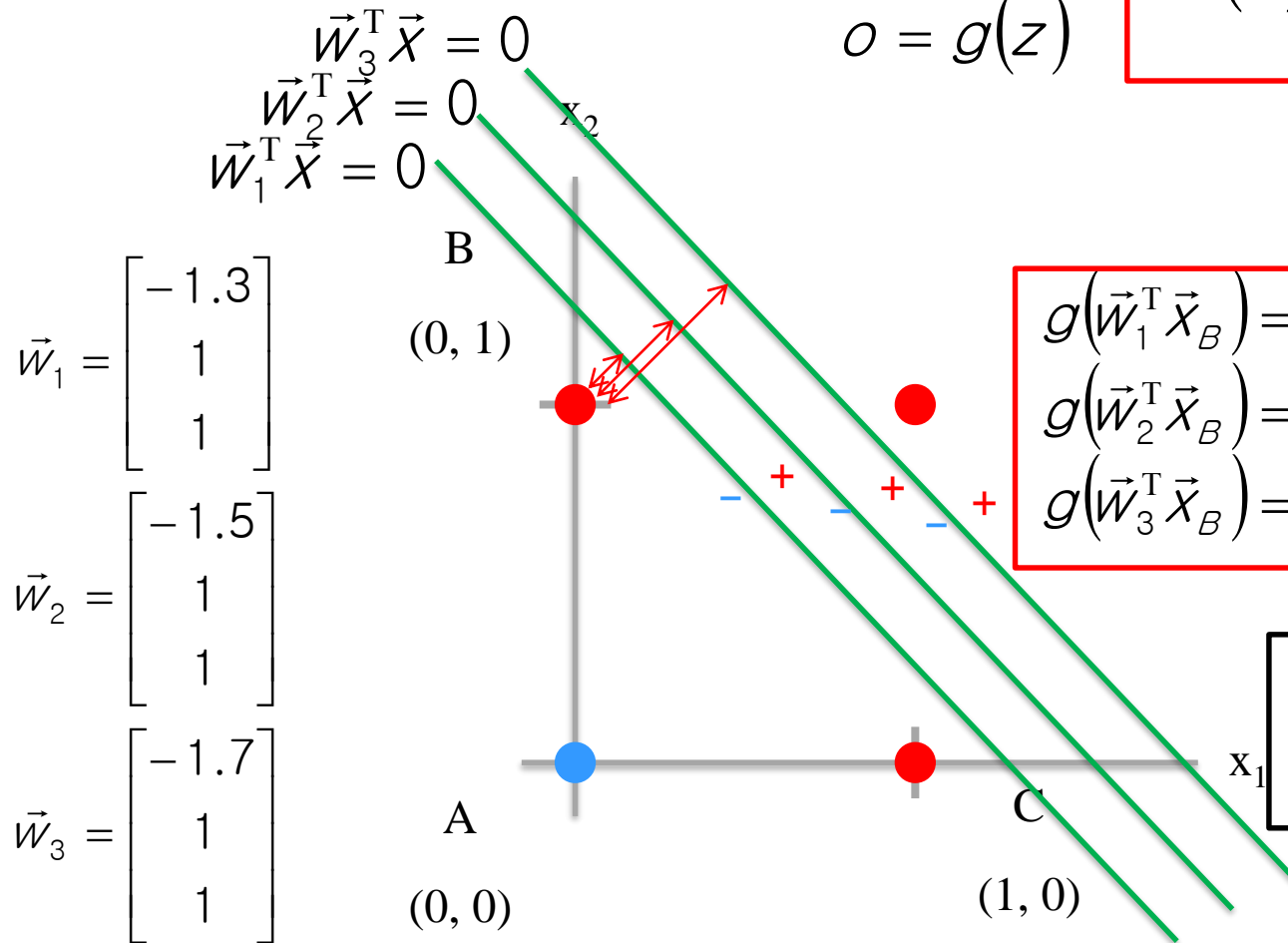
Activation function에 따른 오차

• OR 게이트 학습 문제

$$z = \vec{w}^T \vec{x}$$

$$o = g(z)$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$



$$g(\vec{w}_1^T \vec{x}_B) = 0.42$$

$$g(\vec{w}_2^T \vec{x}_B) = 0.37$$

$$g(\vec{w}_3^T \vec{x}_B) = 0.33$$

$$d_B = 1$$

$$d_B - o_B = 0.58$$

$$d_B - o_B = 0.63$$

$$d_B - o_B = 0.67$$

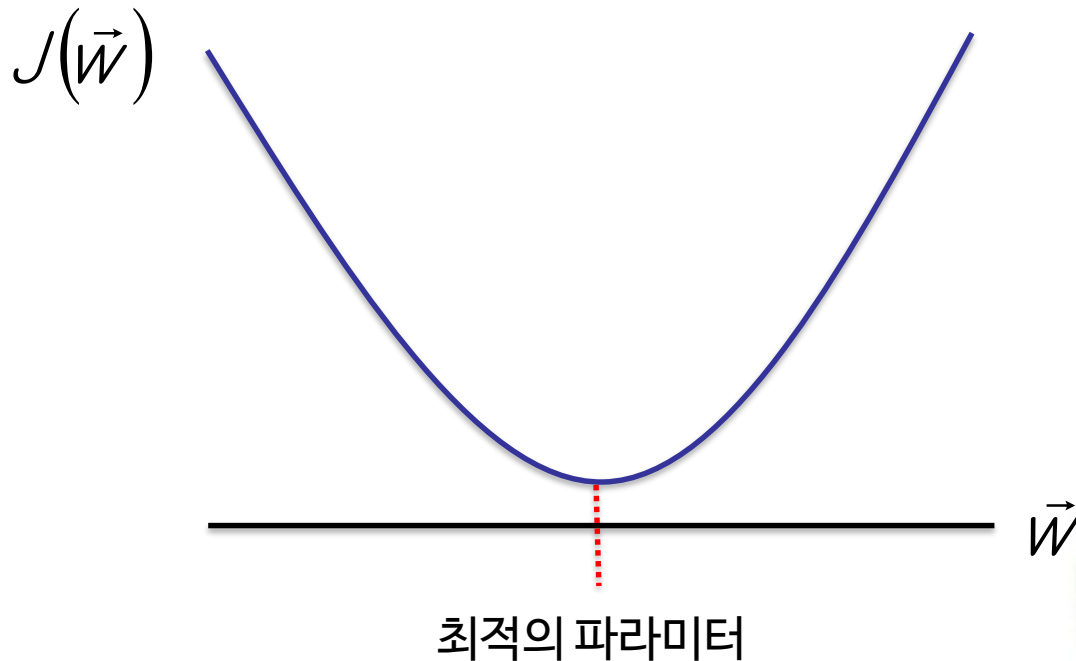
연속 함수
 (+기왕이면 미분 가능 함수)
 +출력의 범위가 한정된 함수

Cost function의 정의

- 학습 데이터 샘플에 대한 코스트

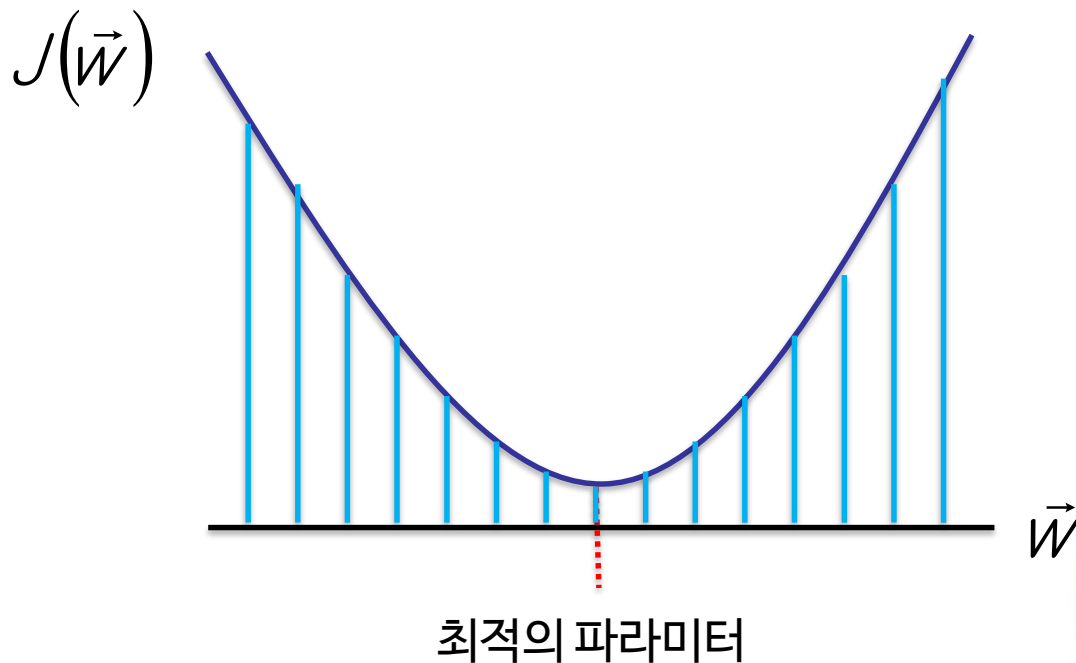
i번째 학습 데이터(특징 벡터, 정답 레이블): $\{(\vec{x}_i, d_i)\}$

코스트: $J(\vec{w}) = \frac{1}{2} (d_i - g(\vec{w}^T \vec{x}_i))^2$



Cost를 줄이는 방향 찾기

- 최적의(혹은 더 좋은) 파라미터를 어떻게 찾을 수 있는가?
 - 가능한 모든 파라미터에 대해서 코스트를 계산?
 - 파라미터의 간격을 얼마나 촘촘히 탐색해야 하나?
 - 특징의 차원이 늘어나면 계산량이 얼마나 늘어나는가?

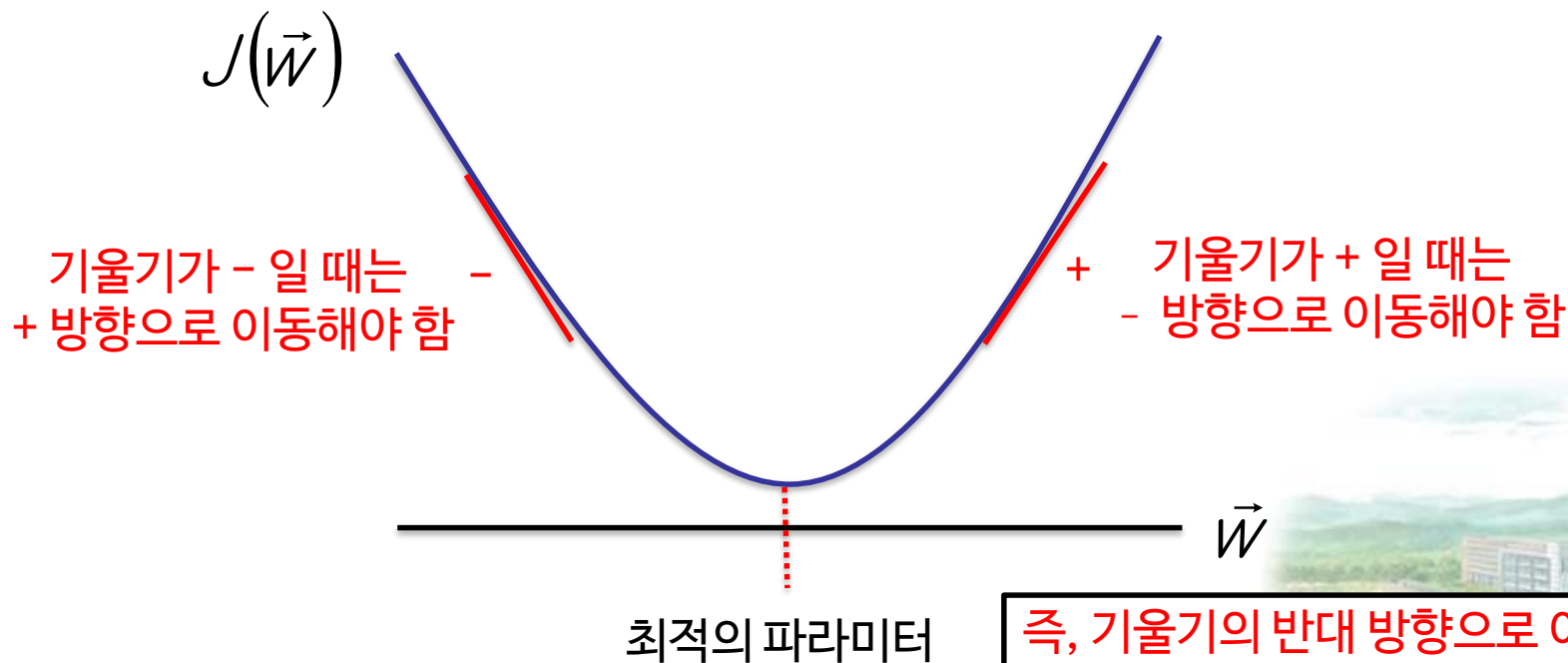


Cost를 줄이는 방향 찾기

- 학습 데이터 샘플에 대한 코스트

i번째 학습 데이터(특징 벡터, 정답 레이블): $\{(\vec{x}_i, d_i)\}$

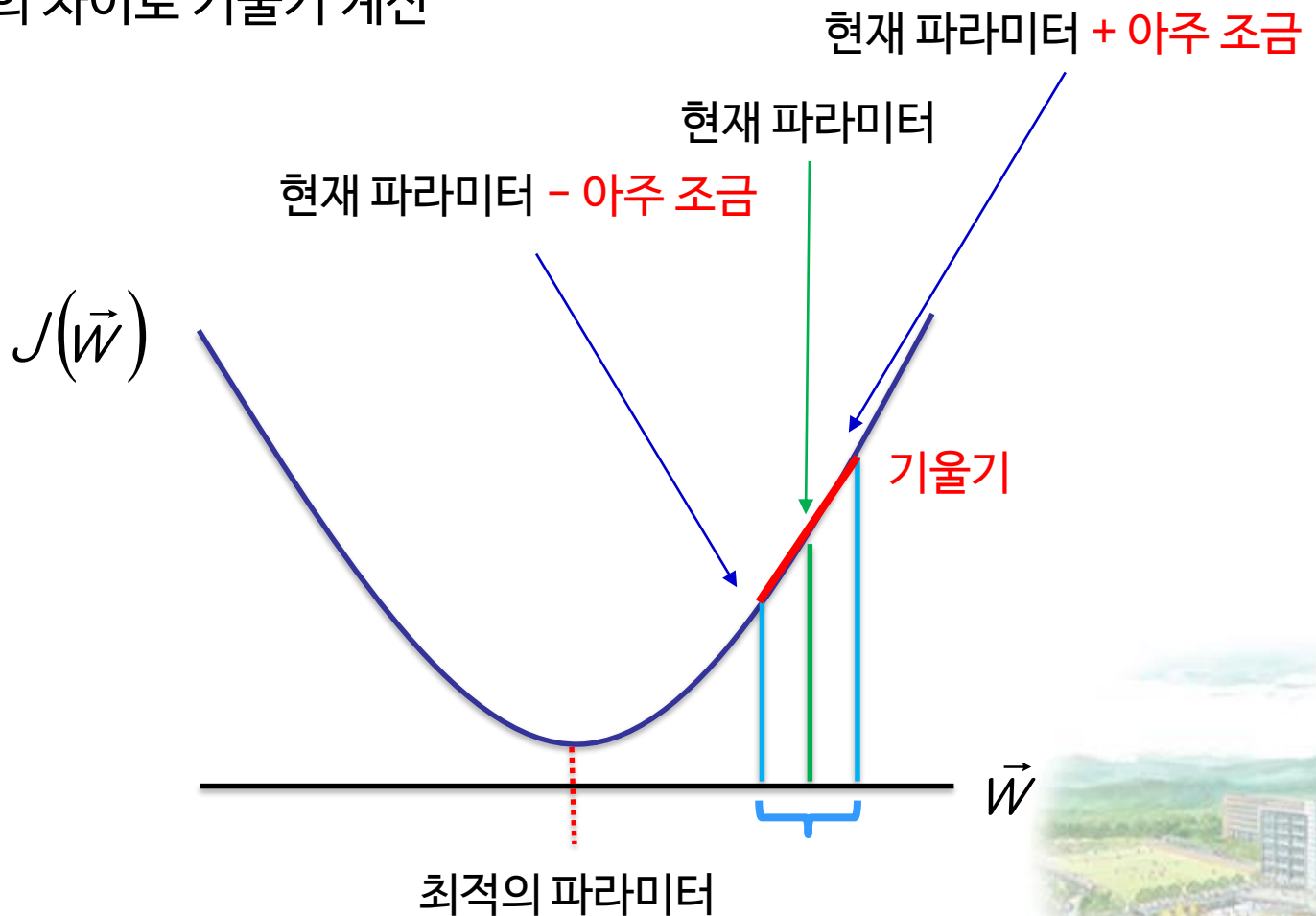
코스트: $J(\vec{w}) = \frac{1}{2} (d_i - g(\vec{w}^T \vec{x}_i))^2$



Cost를 줄이는 방향 찾기

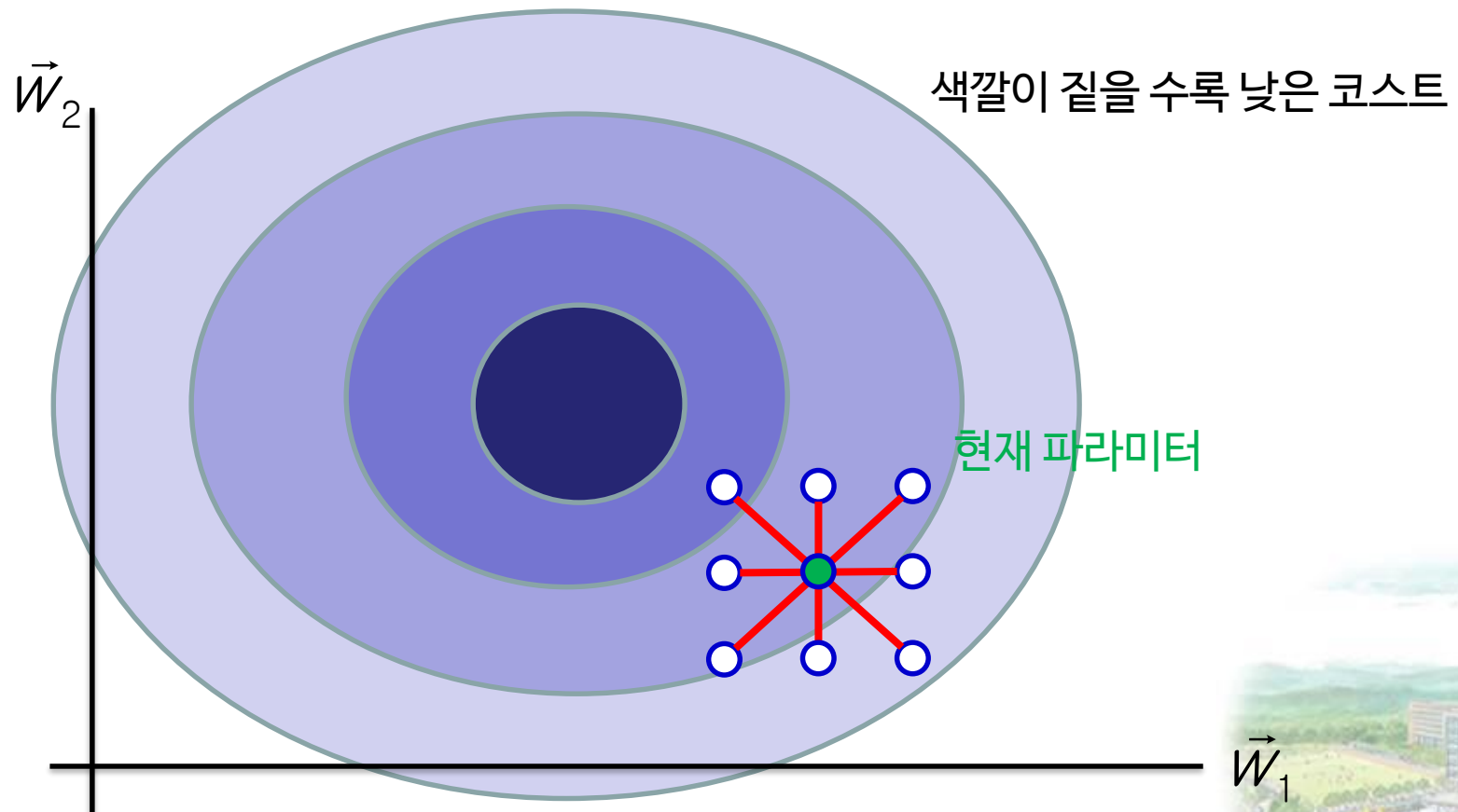
- Numerical gradient

- 현재 파라미터에서 아주 약간 떨어진 위치의 코스트 계산해보기
- 두 코스트의 차이로 기울기 계산



Cost를 줄이는 방향 찾기

- Numerical gradient
 - 2차원일 경우: 대각선 방향도 확인해 볼 건가?
 - 특징의 **차원이 늘어나면** 계산량이 얼마나 늘어나는가?



Cost를 줄이는 방향 찾기

- Delta rule (gradient descent를 이용한 perceptron 학습 방법)
 - 현재 파라미터에서의 gradient 계산
 - 각 차원별 기울기 벡터
 - gradient의 반대 방향으로 gradient 갱신

cost function 예시:

$$J(\vec{w}) = \frac{1}{2} (d_i - g(\vec{w}^T \vec{x}_i))^2$$

Update:

$$\vec{w} = \vec{w} - \rho \nabla J(\vec{w})$$

gradient:

$$\nabla J(\vec{w}) = \begin{bmatrix} \frac{\partial J(\vec{w})}{\partial w_0} \\ \frac{\partial J(\vec{w})}{\partial w_1} \\ \vdots \\ \frac{\partial J(\vec{w})}{\partial w_m} \end{bmatrix}$$



Gradient descent

- Perceptron의 출력: $z = \vec{w}^T \vec{x}$

weighted sum

$$o = g(z) \quad \text{-----} \quad g(z)$$

activation function

- Cost function:

$$J(\vec{w}) = \frac{1}{2} (d - o)^2$$

squared error

- 파라미터 업데이트 (gradient descent):

$$\vec{w} = \vec{w} - \rho \nabla J(\vec{w})$$

learning rate

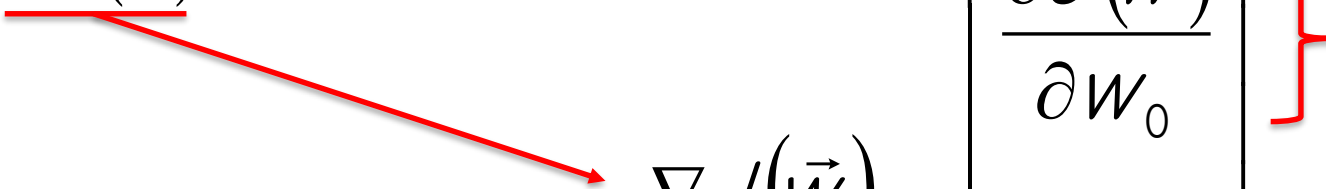
gradient

$$\nabla J(\vec{w}) =$$

$$\begin{bmatrix} \frac{\partial J(\vec{w})}{\partial w_0} \\ \frac{\partial J(\vec{w})}{\partial w_m} \end{bmatrix}$$

Gradient descent

- 파라미터 업데이트 (gradient descent) 수식 유도:

$$\vec{W} = \vec{W} - \rho \nabla J(\vec{W})$$
$$\nabla J(\vec{W}) = \begin{bmatrix} \frac{\partial J(\vec{W})}{\partial w_0} \\ \frac{\partial J(\vec{W})}{\partial w_m} \end{bmatrix}$$




Gradient descent

- 파라미터 업데이트 (gradient descent) 수식 유도:

$$\frac{\partial J(\vec{w})}{\partial w_i} = \frac{\partial \left(\frac{1}{2} (d - o)^2 \right)}{\partial w_i}$$

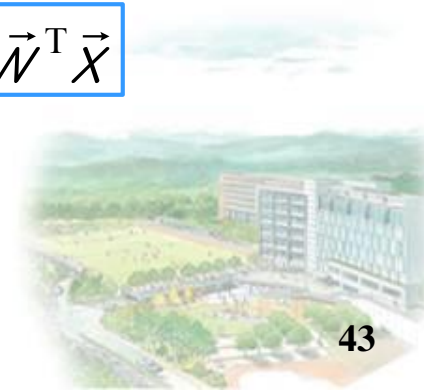
$$J(\vec{w}) = \frac{1}{2} (d - o)^2$$

$$= \frac{\partial \left(\frac{1}{2} (d - o)^2 \right)}{\partial o} \frac{\partial o}{\partial w_i} = -(d - o) \frac{\partial o}{\partial w_i}$$

$$o = g(z)$$

$$= -(d - o) \frac{\partial g(z)}{\partial w_i} = -(d - o) \frac{\partial g(z)}{\partial z} \frac{\partial z}{\partial w_i}$$

$$z = \vec{w}^T \vec{x}$$



Gradient descent

- 파라미터 업데이트 (gradient descent) 수식 유도:

$$\frac{\partial J(\vec{w})}{\partial w_i} = -(d - o) \frac{\partial g(z)}{\partial w_i} = -(d - o) \frac{\partial g(z)}{\partial z} \frac{\partial z}{\partial w_i} \quad \boxed{z = \vec{w}^T \vec{x}}$$

$$= -(d - o) g'(z) \frac{\partial (\vec{w}^T \vec{x})}{\partial w_i} \quad \boxed{\vec{w}^T \vec{x} = \sum_{k=0}^d w_k x_k = w_0 x_0 + w_1 x_1 + \dots + w_d x_d}$$

$$= -(d - o) g'(z) x_i$$

$$\boxed{\frac{\partial J(\vec{w})}{\partial w_i} = -(d - o) g'(z) x_i}$$

만약 cost function이 바뀌면 다시 미분해서 유도해야 함



Gradient descent

- 파라미터 업데이트 (gradient descent) 수식 유도:

$$\vec{W} = \vec{W} - \rho \nabla J(\vec{W})$$

$$\nabla J(\vec{W}) = \begin{bmatrix} \frac{\partial J(\vec{W})}{\partial w_0} \\ \frac{\partial J(\vec{W})}{\partial w_m} \end{bmatrix} = \begin{bmatrix} -(d - o)g'(z)x_0 \\ \dots \\ -(d - o)g'(z)x_m \end{bmatrix}$$

$$= -(d - o)g'(z) \begin{bmatrix} x_0 \\ \dots \\ x_m \end{bmatrix} = -(d - o)g'(z)\vec{x}$$

$$\vec{W} = \vec{W} + \rho(d - o)g'(z)\vec{x}$$



Gradient descent

- Logistic function 0| activation function인 경우:

$$\vec{w} = \vec{w} + \rho(d - o) \underline{g'(z)} \vec{x}$$

$$g'(z) = \frac{d}{dz} g(z) = \frac{d}{dz} \frac{1}{1 + e^{-z}}$$

$$= \frac{d}{dz} (1 + e^{-z})^{-1}$$

$$= -(1 + e^{-z})^{-2} (-e^{-z})$$

$$= -\frac{(-e^{-z})}{(1 + e^{-z})^2} = \frac{e^{-z}}{(1 + e^{-z})^2}$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$

Logistic function

$$\frac{d}{dx} e^{ax} = a e^{ax}$$



Gradient descent

- Logistic function | activation function인 경우:

$$\vec{w} = \vec{w} + \rho(d - o) \underline{g'(z)} \vec{x}$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$

Logistic function

$$g'(z) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

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

$$= g(z) \left(\frac{1 + e^{-z} - 1}{1 + e^{-z}} \right) = g(z) \left(\frac{1 + e^{-z}}{1 + e^{-z}} - \frac{1}{1 + e^{-z}} \right)$$

$$= g(z)(1 - g(z))$$

만약 activation function이 바뀌면
다시 미분해서 유도해야 함



Gradient descent

- Logistic function | activation function인 경우:

$$\vec{w} = \vec{w} + \rho(d - o) \underline{g'(z)} \vec{x}$$

$$g'(z) = g(z)(1 - g(z))$$

$$g(z) = \frac{1}{1 + \exp(-z)}$$

Logistic function

$$\vec{w} = \vec{w} + \rho(d - o) g(z)(1 - g(z)) \vec{x}$$

$$= \vec{w} + \rho(d - o) \rho(1 - o) \vec{x}$$



Gradient descent

- Online training
 - 학습 샘플 하나 하나마다 파라미터 업데이트
 - batch training
 - Mini-batch: 일부 학습 샘플에 대해 계산한 뒤 한 번에 업데이트
 - Batch size: 한 번에 처리할 학습 샘플의 수
 - 한 mini-batch에 포함될 샘플들을 매 번 골고루 섞어주면 약간 더 좋음
 - Full-batch: 모든 학습 샘플에 대해 계산한 뒤 한 번에 업데이트
- stochastic gradient descent
- batch gradient descent



Online training

학습 반복 횟수: $t = 0$

perceptron의 파라미터(weight+bias) $\vec{w}(t)$ 랜덤 초기화

학습 데이터세트(특징 벡터, 정답 레이블): $\{(\vec{x}_1, d_1), (\vec{x}_2, d_2), \dots, (\vec{x}_n, d_n)\}$

for 충분히 오류가 감소할 때까지 반복 학습:

for 각 학습 데이터 (\vec{x}_k, d_k) 에 대해서 ($1 \leq k \leq n$):

$$o_k = g(\vec{w}(t)^T \vec{x}_k)$$

for 각 차원의 파라미터 $w_i(t)$ 에 대해서 ($0 \leq i \leq m$):

$$w_i(t+1) = w_i(t) + \rho(d_k - o_k)o_k(1 - o_k)x_{k,i}$$

$$t = t + 1$$

학습 종료

m : 입력 특징의 차원

$x_{k,i}$: k번째 학습 특징 벡터의 i번째 원소

$w_i(t)$: t번째 반복 학습한 파라미터의 i번째 원소

Full-batch training

학습 반복 횟수: $t = 0$

perceptron의 파라미터(weight+bias) $\vec{w}(t)$ 랜덤 초기화

학습 데이터세트(특징 벡터, 정답 레이블): $\{(\vec{x}_1, d_1), (\vec{x}_2, d_2), \dots, (\vec{x}_n, d_n)\}$

for 충분히 오류가 감소할 때까지 반복 학습:

for 각 차원에 대해서 ($0 \leq i \leq m$):

$$\Delta w_i = 0$$

for 각 학습 데이터 (\vec{x}_k, d_k) 에 대해서 ($1 \leq k \leq n$):

$$o_k = g(\vec{w}(t)^T \vec{x}_k)$$

for 각 차원에 대해서 ($0 \leq i \leq m$):

$$\Delta w_i += \rho(d_k - o_k)o_k(1 - o_k)x_{k,i}$$

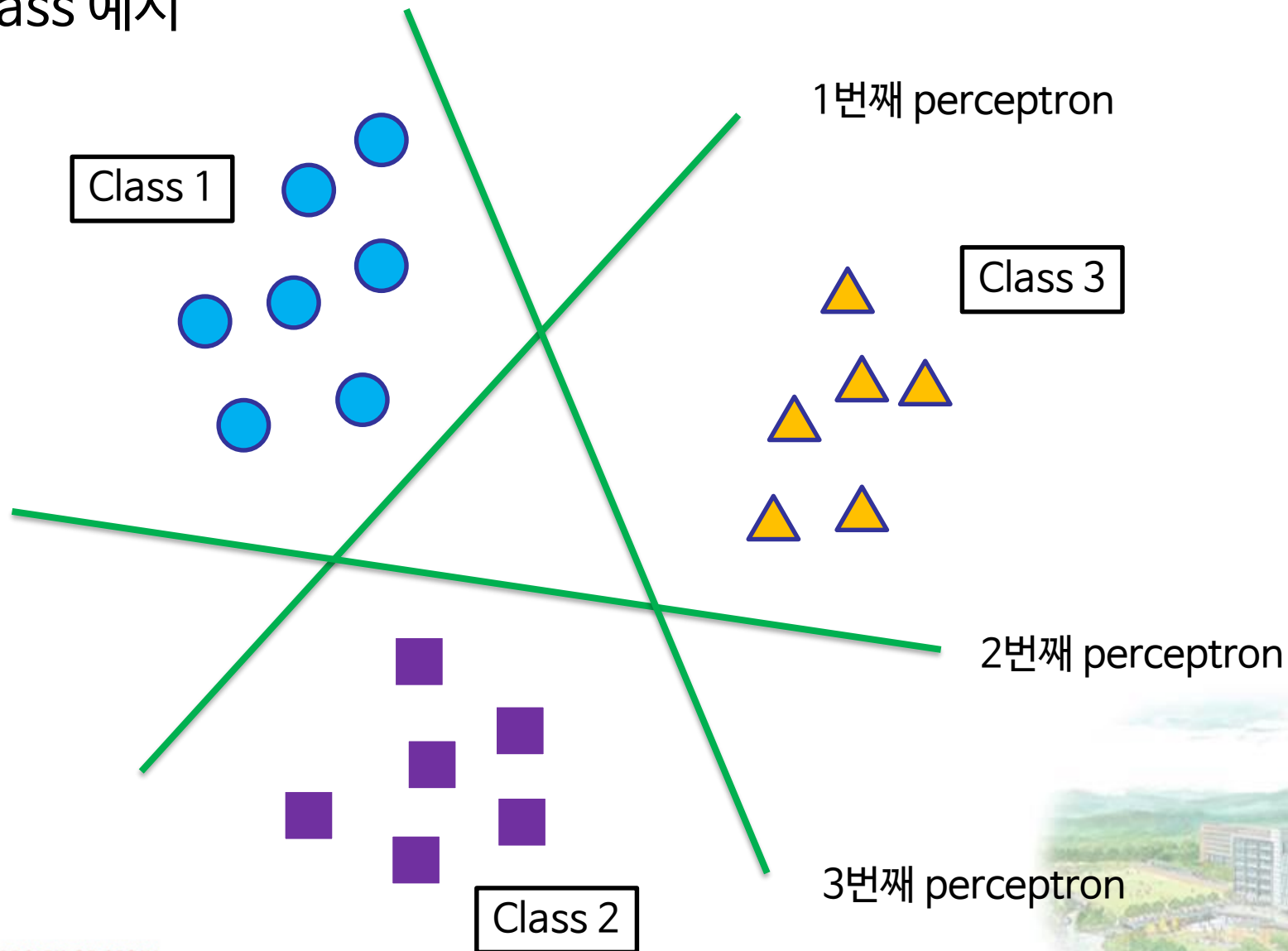
for 각 차원에 대해서 ($0 \leq i \leq m$):

$$w_i(t+1) = w_i(t) + \Delta w_i$$

$$t = t + 1$$

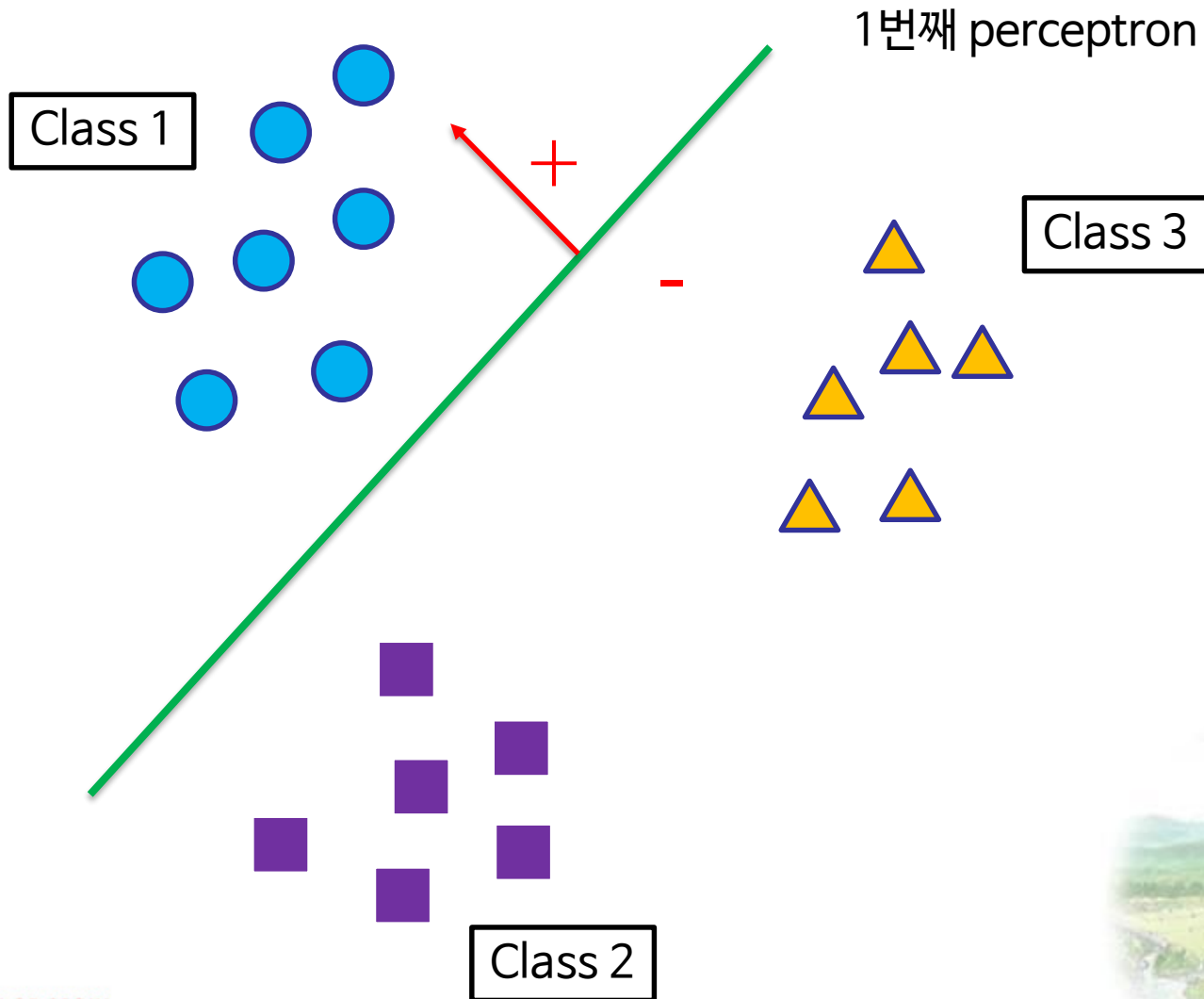
Single layer perceptron

- 3-class 예시



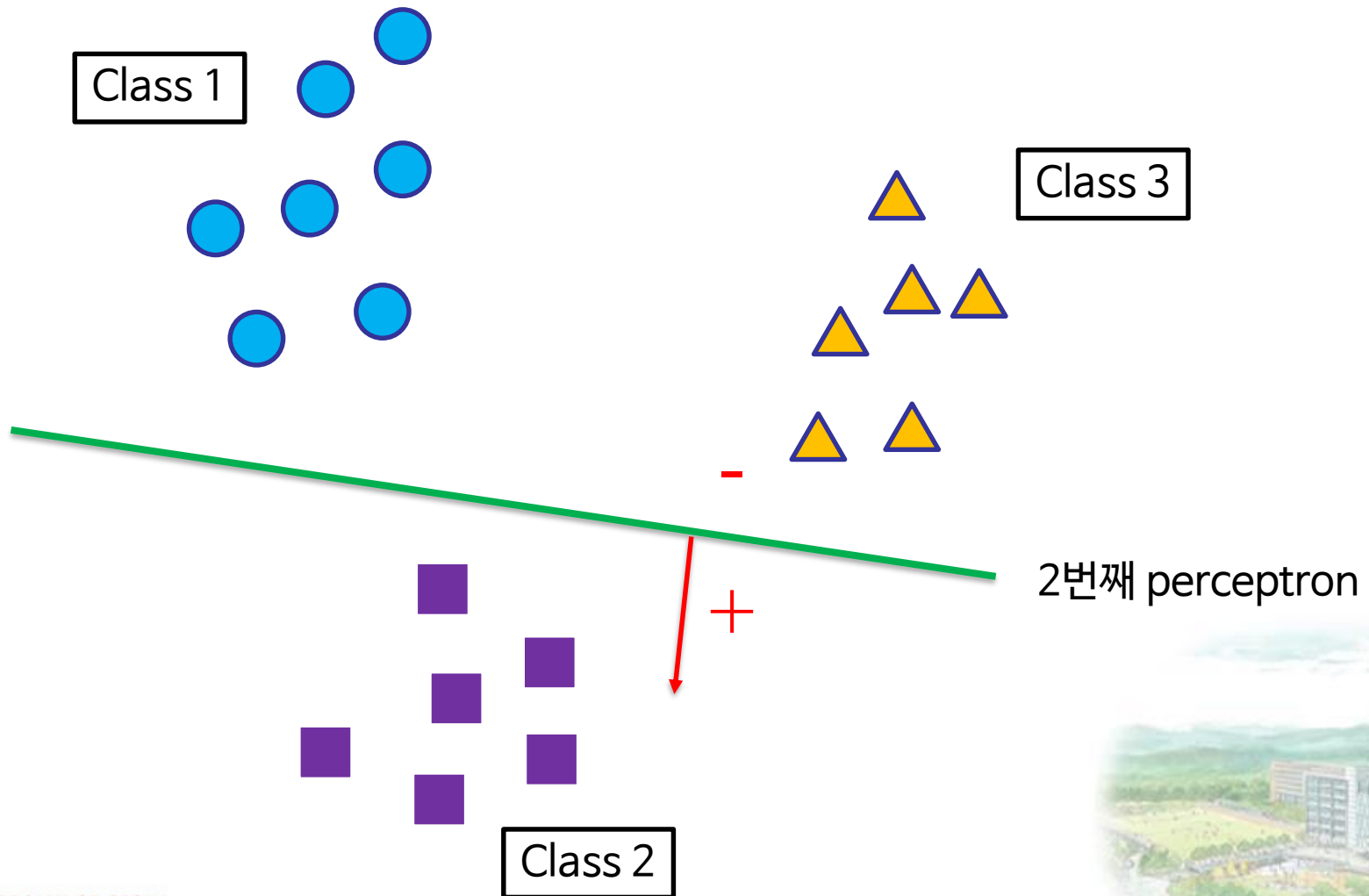
Single layer perceptron

- 3-class 예시



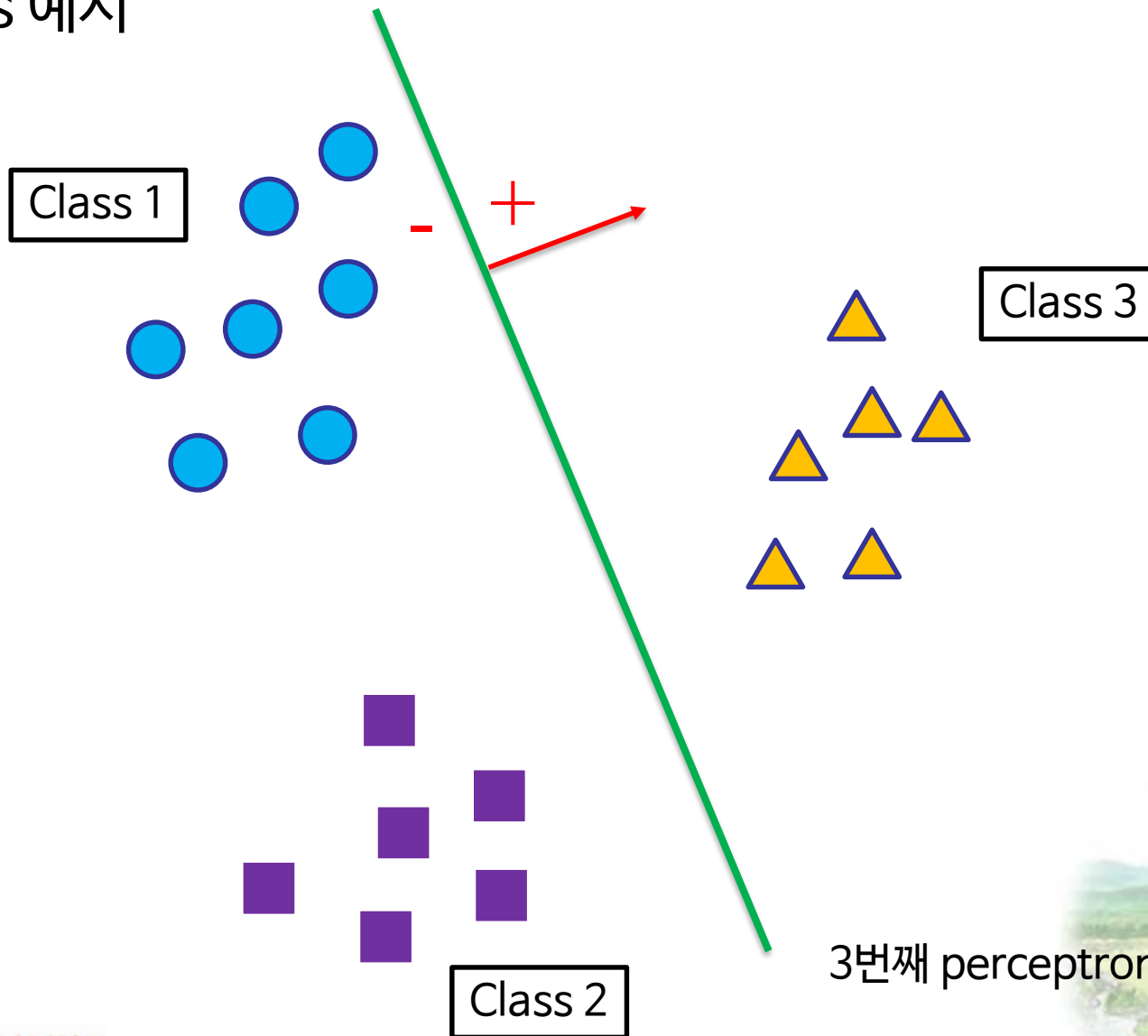
Single layer perceptron

- 3-class 예시



Single layer perceptron

- 3-class 예시



Single layer perceptron

- 3-class 예시

입력 특징 벡터

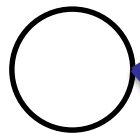
\vec{x}_i

Single layer perceptron

$$g(\vec{w}_1^T \vec{x}_i)$$

출력 벡터

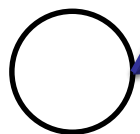
$O_{i,1}$



$$g(\vec{w}_2^T \vec{x}_i)$$

$O_{i,2}$

...



$$g(\vec{w}_3^T \vec{x}_i)$$

$O_{i,3}$



Single layer perceptron

- One-hot representation

Perceptron의 출력 벡터: $O_i = \begin{bmatrix} O_{i,1} \\ O_{i,2} \\ O_{i,3} \end{bmatrix}$

정답 레이블: $d_i = \begin{bmatrix} d_{i,1} \\ d_{i,2} \\ d_{i,3} \end{bmatrix}$

Class 1 데이터에 대한
정답 레이블

$$d = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

Class 2 데이터에 대한
정답 레이블

$$d = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Class 3 데이터에 대한
정답 레이블

$$d = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



Single layer perceptron

- Mean squared error

k번째 학습 데이터(특징 벡터, 정답 레이블): $\{(\vec{x}_k, d_k)\}$

코스트:
$$J(\vec{w}) = \frac{1}{2} \sum_{j=1}^c (d_{k,j} - g(\vec{w}_j^T \vec{x}_k))^2$$



Single layer perceptron

- Mean squared error

k번째 학습 데이터(특징 벡터, 정답 레이블): $\{(\vec{x}_k, d_k)\}$

코스트:
$$J(\vec{w}) = \frac{1}{2} \sum_{j=1}^c (d_{k,j} - g(\vec{w}_j^T \vec{x}_k))^2$$

k-번째 학습 데이터

j-번째 perceptron

i-번째 입력 차원

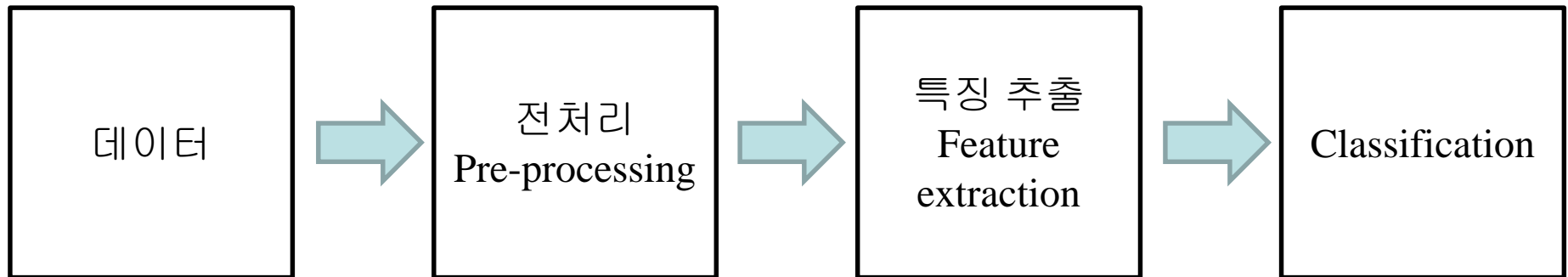
$$o_{k,j} = g(\vec{w}_j(t)^T \vec{x}_k)$$

$$\Delta w_j = (d_{k,j} - o_{k,j}) o_{k,j} (1 - o_{k,j}) x_{k,i}$$

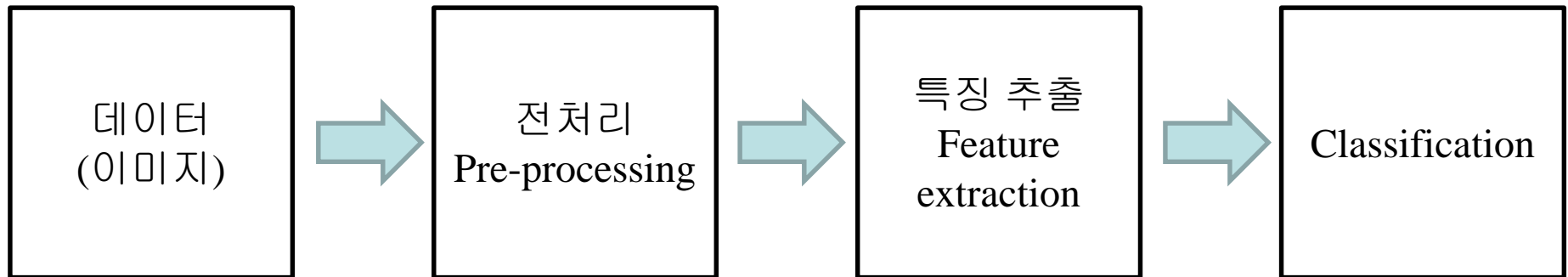
$$w_{j,i}(t+1) = w_{j,i}(t) + \Delta w_j(t)$$



Pattern classification의 흐름

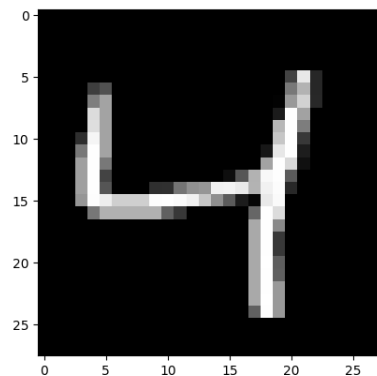
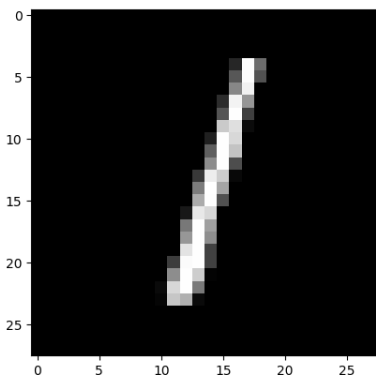
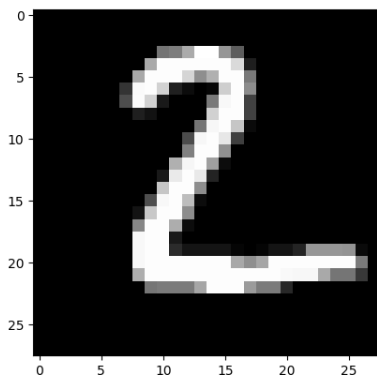
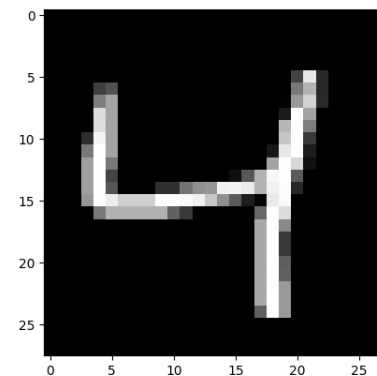
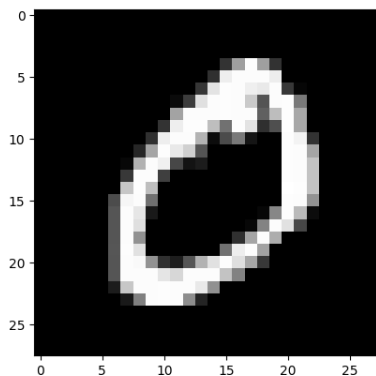
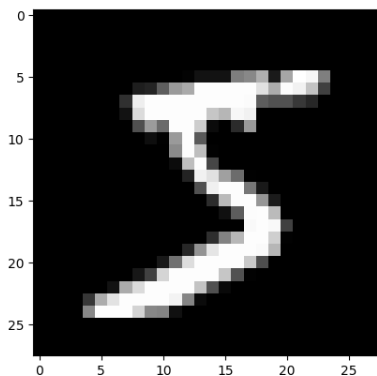


숫자인식기 만들기



MNIST

- THE MNIST DATABASE of handwritten digits
 - <http://yann.lecun.com/exdb/mnist/>



QnA

실습

실습 과제

- Single-layer perceptron으로 숫자인식해보기
 - Stochastic gradient descent로 학습
 - MNIST DB 사용



Python 실습

- 클래스 멤버변수 선언 위치는 신중히 결정

```
# -*- coding: utf-8 -*-
class MyClass:
    m = [0, 1, 2]

    def Add(self, acc):
        self.m += [acc]

if __name__ == '__main__':
    a = MyClass()
    b = MyClass()

    print a.m, b.m

    a.Add(3)
    print a.m, b.m
```

```
# -*- coding: utf-8 -*-
class MyClass:
    def __init__(self):
        self.m = [0, 1, 2]

    def Add(self, acc):
        self.m += [acc]

if __name__ == '__main__':
    a = MyClass()
    b = MyClass()

    print a.m, b.m

    a.Add(3)
    print a.m, b.m
```

Python 실습

- numpy.array
 - 벡터/행렬 연산을 간편하게

```
# -*- coding: utf-8 -*-
import numpy as np

if __name__ == '__main__':
    # numpy array형으로 변환
    a = np.array([1, 2, 3, 4, 5, 6]).reshape(3, -1)

    print a
    print a[2, 1]
    print a[2, :]
    print a[:, 1]
    print a[:2, 1]
```

[주의사항]

숫자만 다를 수 있음

한 array 내의 모든 숫자는 같은 형을 지님(ex) float32, int32)



Python 실습

- numpy.array
 - 벡터/행렬 연산을 간편하게

```
# -*- coding: utf-8 -*-  
import numpy as np  
  
if __name__ == '__main__':  
    # numpy array형으로 변환  
    a = np.array([1, 2, 3])  
    b = np.array([-1, 0, 1])  
    c = 2  
  
    print a  
    print b  
    print a + b  
    print (a + b) * c
```

```
# -*- coding: utf-8 -*-  
import numpy as np  
  
if __name__ == '__main__':  
    # numpy array형으로 변환  
    a = np.array([1, 2, 3]).reshape(3, 1)  
    b = np.array([-1, 0, 1]).reshape(3, 1)  
    c = 2  
  
    print a  
    print b  
    print a + b  
    print (a + b) * c
```



Python 실습

- numpy.array
 - 벡터/행렬 연산을 간편하게

```
# -*- coding: utf-8 -*-
import numpy as np

if __name__ == '__main__':
    # numpy array형으로 변환 + float형으로 변환
    a = np.array([1, 2, 3]).astype('float32')
    b = np.array([-1, 0, 1]).astype('float32')

    # (3 x 1) 행렬로 크기 변환
    a = a.reshape(3, 1)
    b = b.reshape(3, 1)

    # numpy array형의 크기 출력
    print a.shape, b.shape

    # a, b의 dot product 결과 계산
    print np.dot(a.T, b)
    print np.dot(a.T, b)[0][0]
```



QnA