

인공지능

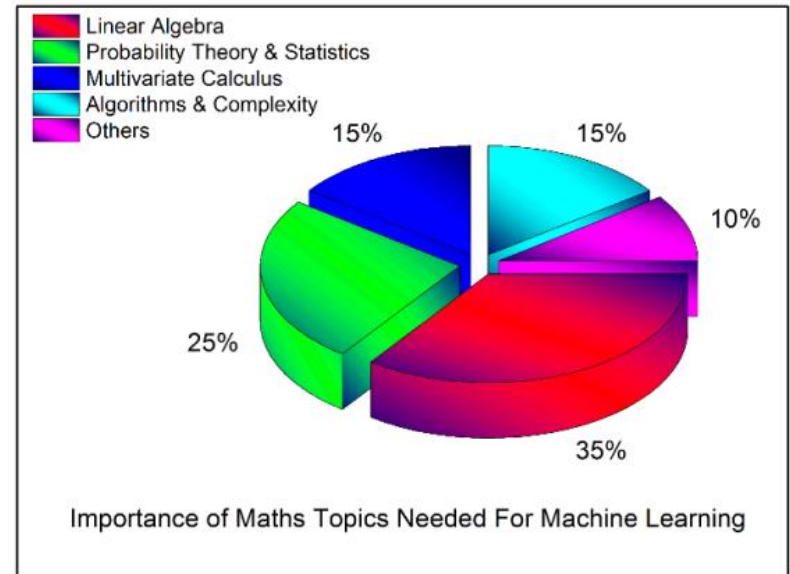
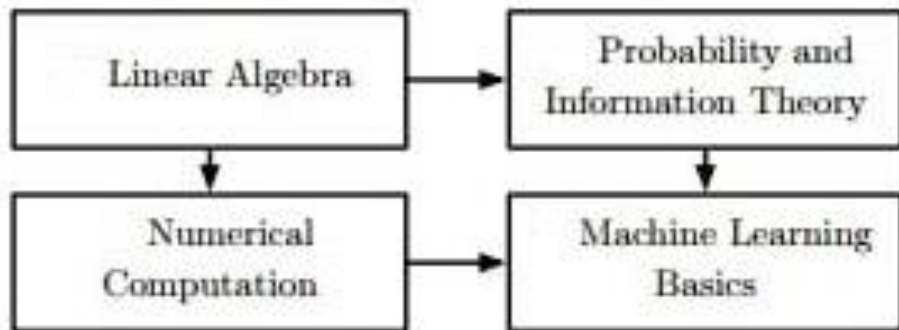
[3. 다층 퍼셉트론]

소프트웨어융합대학
소프트웨어학부

본 자료는 해당 수업의 교육 목적으로만 활용될 수 있음.
일부 내용은 다른 교재와 논문으로부터 인용되었으며, 모든 저작권은 원 교재와 논문에 있음.

지난 장에는 뭐했지?

2. 기계 학습과 수학



오늘 수업에는 뭐하지?

PREVIEW

■ 신경망

- 기계 학습 역사에서 가장 오래된 기계 학습 모델이며, 현재 가장 다양한 형태를 가짐
- 1950년대 퍼셉트론 (인공두뇌학cybernetics) → 1980년대 다층 퍼셉트론 (결합설connectionism)
- 3장은 4장 딥러닝의 기초가 됨

각 절에서 다루는 내용

3.1절 신경망의 역사와 종류를 간략히 소개한다.

3.2절 1950년대 개발된 퍼셉트론의 구조와 동작, 학습 알고리즘을 자세히 설명한다.

3.3절 선형 분류기로서 퍼셉트론의 한계를 지적하고, 퍼셉트론을 여러 개 이어 붙여 비선형 공간 분할이 가능한 다층 퍼셉트론으로 확장한다.

3.4절 다층 퍼셉트론의 그레이디언트를 효율적으로 계산하는 오류 역전파 알고리즘을 유도한다.

3.5절 현대 기계 학습이 널리 활용하는 미니배치 스토캐스틱 경사 하강법을 설명한다.

3.6절 학습을 마친 다층 퍼셉트론이 인식하는 단계를 설명한다.

3.7절 다층 퍼셉트론의 특성을 기술한다.



3.1 신경망 기초

- 3.1.1 **인공신경망**과 **생물신경망**
- 3.1.2 신경망의 간략한 **역사**
- 3.1.3 신경망의 **종류**

3.1.1 인공신경망과 생물신경망

■ 사람의 뉴런

- 두뇌의 가장 작은 정보처리 단위
- 세포체는 cell body 간단한 연산, 수상돌기는 dendrite 신호 수신, 축삭은 axon 처리 결과를 전송
- 사람은 10^{11} 개 정도의 뉴런을 가지며,
뉴런은 1000개 가량 다른 뉴런과 연결되어 있어 10^{14} 개 정도의 연결

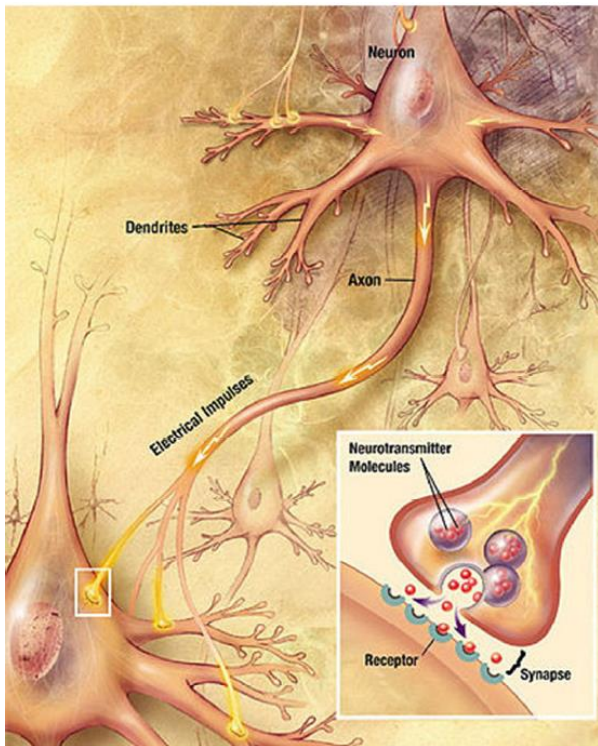
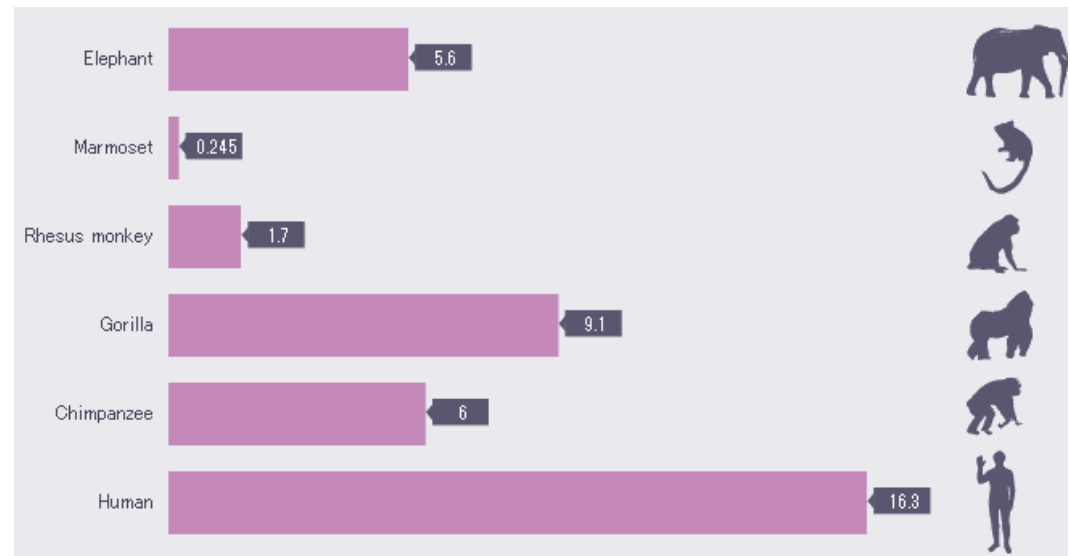


그림 3-1 사람의 뉴런의 구조와 동작

대뇌피질의 뉴런의 수 비교



3.1.1 인공신경망과 생물신경망

■ 두 줄기 연구의 시너지|synergy 효과

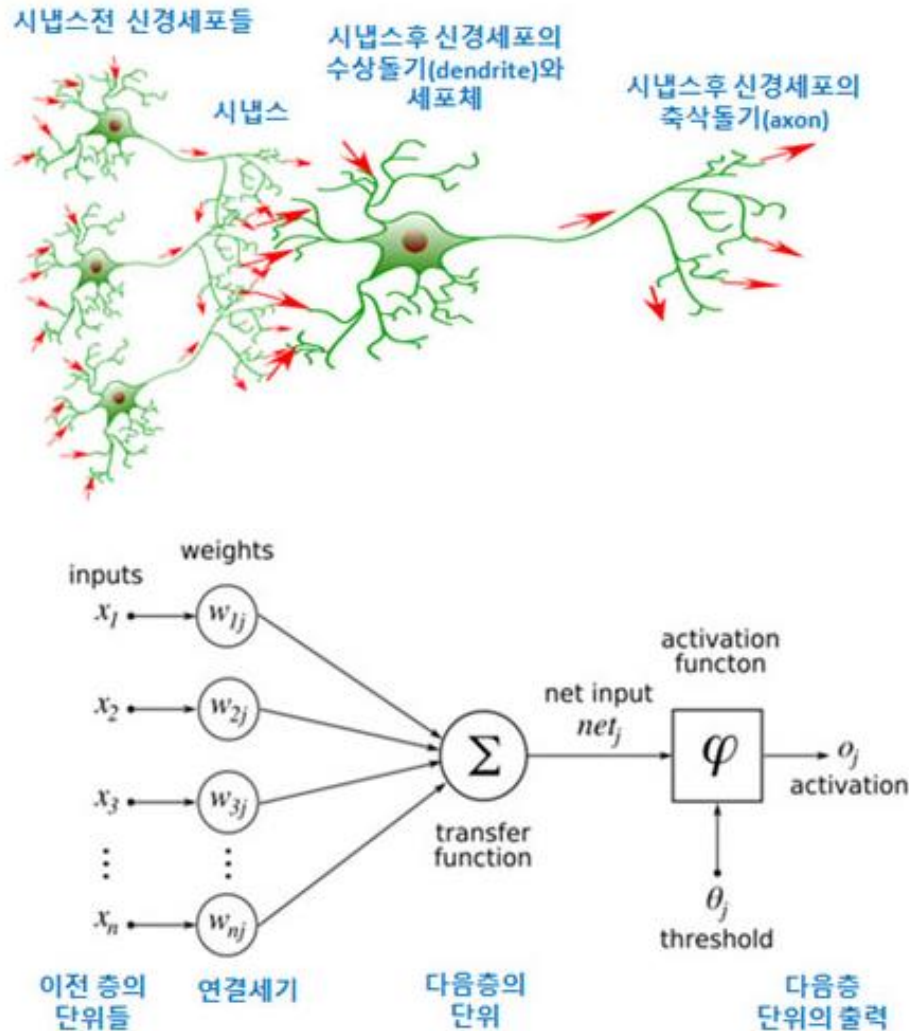
- 컴퓨터 과학
 - 계산 (연산) 능력의 획기적 발전으로 지능 처리에 대한 욕구 확대
- 뇌 (의학) 과학
 - 뇌의 정보처리 방식 연구

→ 뇌의 정보처리 모방하여 사람의 지능 처리할 수 있는 인공지능 도전

- 뉴런의 동작 이해를 모방한 인공 신경망artificial neural networks (ANN) 연구 수행됨
- 페셉트론 고안

3.1.1 인공신경망과 생물신경망

■ 사람의 신경망과 인공신경망 비교



사람 신경망	인공 신경망
세포체	노드
수상돌기	입력
축삭	출력
시냅스	가중치

3.1.2 신경망의 간략한 역사

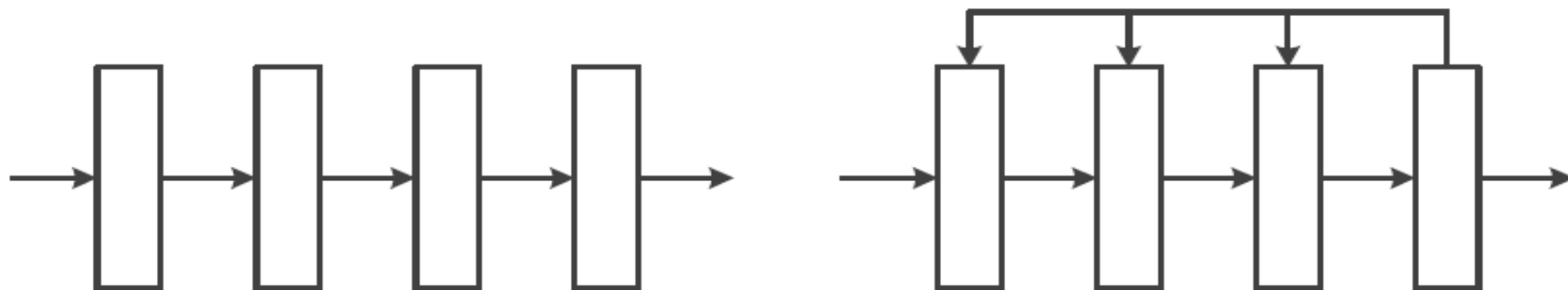
■ 신경망의 역사

- 1943년 매컬락과 피츠의 최초의 신경망
- 1949년 헤브는 최초로 학습 알고리즘 제안
- 1958년 로젠블렛은 퍼셉트론 제안 [3.2절에서 다룸]
- 위드로와 호프의 Adaline 과 Madaline
- 1960년대의 과대 평가
- 1969년 민스키와 페퍼트의 저서 『Perceptrons』는 퍼셉트론의 한계를 수학적으로 입증
 - 퍼셉트론은 선형분류기에 불과하여 XOR 문제조차 해결 못함
- 신경망 연구 퇴조
- 1986년 루멜하트의 저서 『Parallel Distributed Processing』은 다층 퍼셉트론 제안 [3.3~3.4절에서 다룸]
- 신경망 연구 부활
- 1990년대 SVM [11장에서 다룸]에 밀리는 형국
- 2000년대 딥러닝이 실현되어 신경망이 기계 학습의 주류 기술로 자리매김
- 보다 상세한 신경망 역사는 [Kurenkov2015] 참조

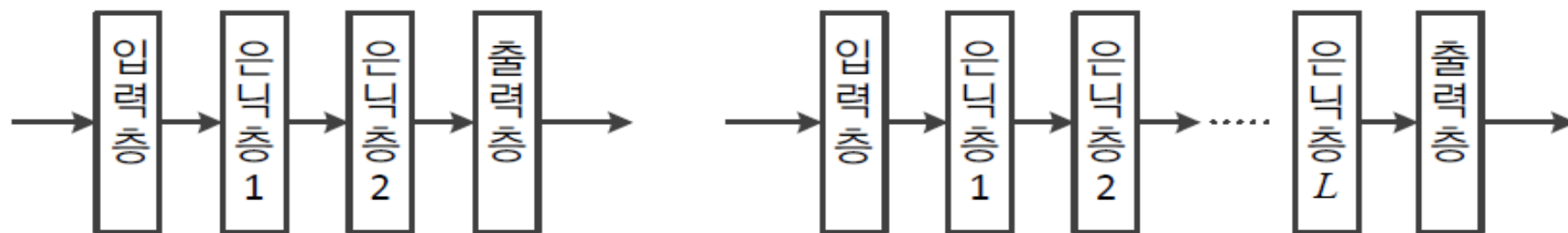
3.1.3 신경망의 종류

■ 신경망에는 아주 다양한 모델이 존재함

- 전방forward 신경망과 순환recurrent 신경망
- 얇은shallow 신경망과 깊은deep 신경망



(a) 전방 신경망과 순환 신경망

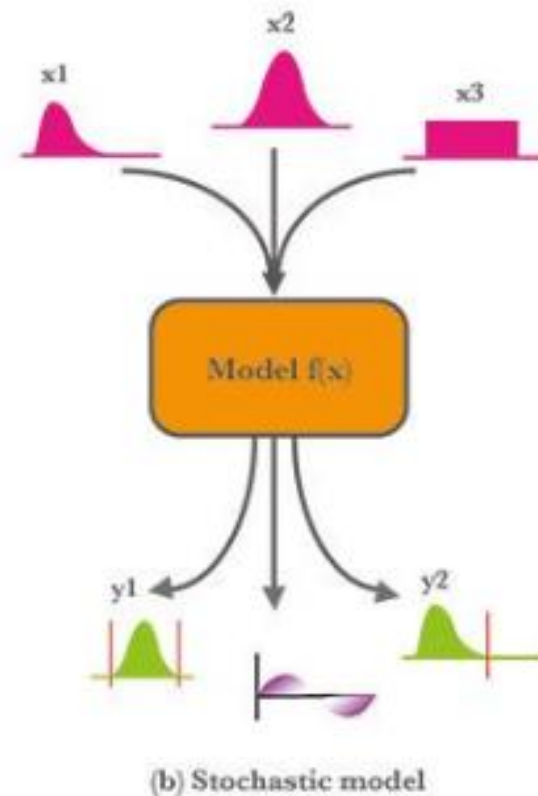
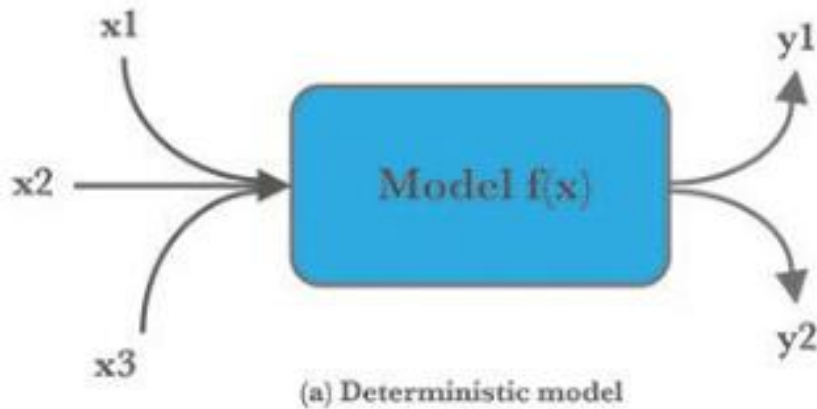


(b) 얇은 신경망과 깊은 신경망

그림 3-2 신경망의 종류

3.1.3 신경망의 종류

- 결정론(deterministic) 신경망과 스토캐스틱(stochastic) 신경망
 - 결정론 신경망: 모델의 매개변수와 조건에 의해 출력이 완전히 결정되는 신경망
 - 스토캐스틱 신경망: 고유의 임의성을 가지고 매개변수와 조건이 같더라도 다른 출력의 가지는 신경망

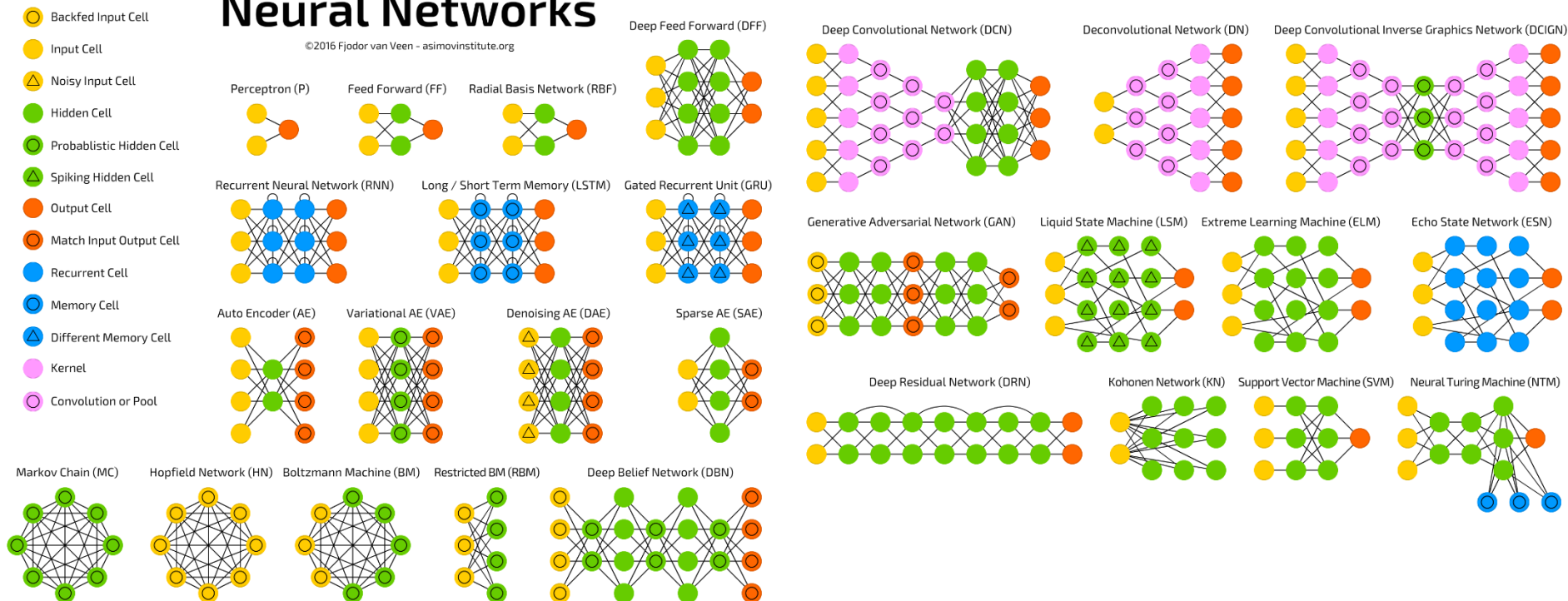


3.1.3 신경망의 종류

■ 다양한 신경망 종류

A mostly complete chart of Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org



3.2 퍼셉트론

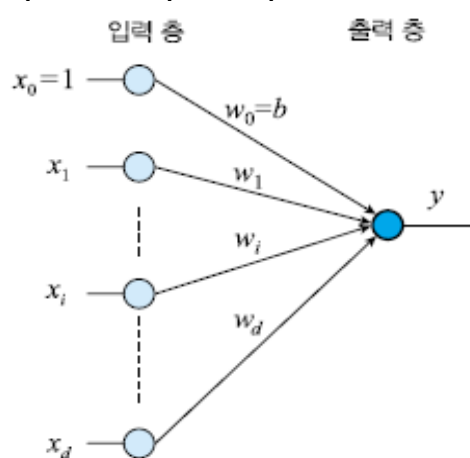
- 3.2.1 구조
- 3.2.2 동작
- 3.2.3 학습

- 퍼셉트론은 **노드**^{node}, **가중치**^{weight}, **층**^{layer}과 같은 새로운 개념을 도입하고 **학습 알고리즘**을 창안함
- **퍼셉트론**은 원시적 신경망이지만,
딥러닝을 포함한 현대 신경망은 퍼셉트론을 병렬과 순차 구조로 결합하여 만듦
→ 현대 신경망의 중요한 구성 요소

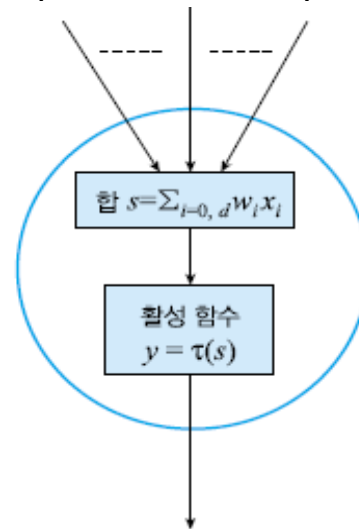
3.2.1 구조

■ 퍼셉트론의 구조

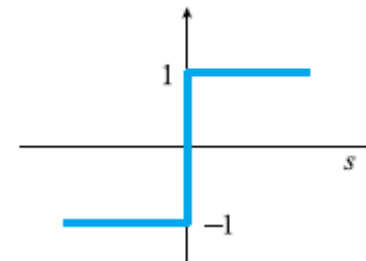
- 입력층과 출력층을 가짐
 - 입력층은 연산을 하지 않으므로 퍼셉트론은 **단일 층 구조**라고 간주
- 입력층**의 i 번째 노드는 특징 벡터 $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$ 의 요소 x_i 를 담당
- 항상 1이 입력되는 바이어스^{bias} 노드
- 출력층**은 한 개의 노드
- i 번째 입력 노드와 출력 노드를 연결하는 변^{edge}은 가중치 w_i 를 가짐



(a) 전체 구조



(b) 출력 노드의 연산



(c) 활성화 함수

3.2.2 동작

■ 퍼셉트론의 동작

- 해당하는 입력(특징)값과 가중치를 곱한 결과를 모두 더하여 s 를 구하고, 활성화함수 τ 를 적용함

- 활성화함수 τ 로 계단함수^{step function}를 사용하므로 최종 출력 y 는 +1 또는 -1

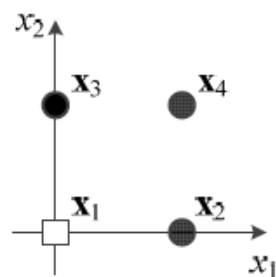
$$y = \tau(s)$$
$$\text{이때 } s = w_0 + \sum_{i=1}^d w_i x_i, \quad \tau(s) = \left\{ \begin{array}{ll} 1 & s \geq 0 \\ -1 & s < 0 \end{array} \right\} \quad (3.1)$$

3.2.2 동작

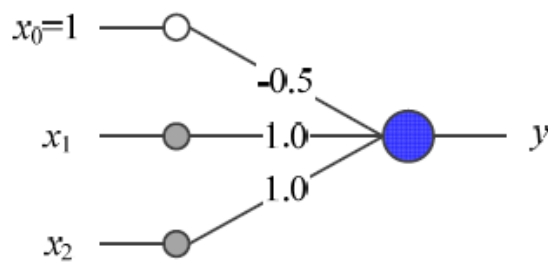
예제 3-1 퍼셉트론의 동작

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

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



(a) 훈련집합



(b) 퍼셉트론

그림 3-4 OR 논리 게이트를 이용한 퍼셉트론의 동작 예시

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

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

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

3.2.2 동작

■ 행렬 표기 | matrix vector notation

$$s = \mathbf{w}^T \mathbf{x} + w_0, \quad \text{여기서 } \mathbf{x} = (x_1, x_2, \dots, x_d)^T, \mathbf{w} = (w_1, w_2, \dots, w_d)^T \quad (3.2)$$

- 바이어스 항을 벡터에 추가하면,

$$s = \mathbf{w}^T \mathbf{x}, \quad \text{여기서 } \mathbf{x} = (1, x_1, x_2, \dots, x_d)^T, \mathbf{w} = (w_0, w_1, w_2, \dots, w_d)^T \quad (3.3)$$

- 퍼셉트론의 동작을 식 (3.4)로 표현할 수 있음

$$y = \tau(\mathbf{w}^T \mathbf{x}) \quad (3.4)$$

3.2.2 동작

■ [그림 3-4(b)]를 기하학적으로 설명하면,

- 결정 직선 $d(\mathbf{x}) = d(x_1, x_2) = w_1x_1 + w_2x_2 + w_0 = 0 \rightarrow x_1 + x_2 - 0.5 = 0$
 - w_1 과 w_2 는 직선의 방향, w_0 은 절편^{intercept}을 결정
 - 결정 직선은 전체 공간을 +1과 -1의 두 부분공간으로 분할하는 **분류기** 역할

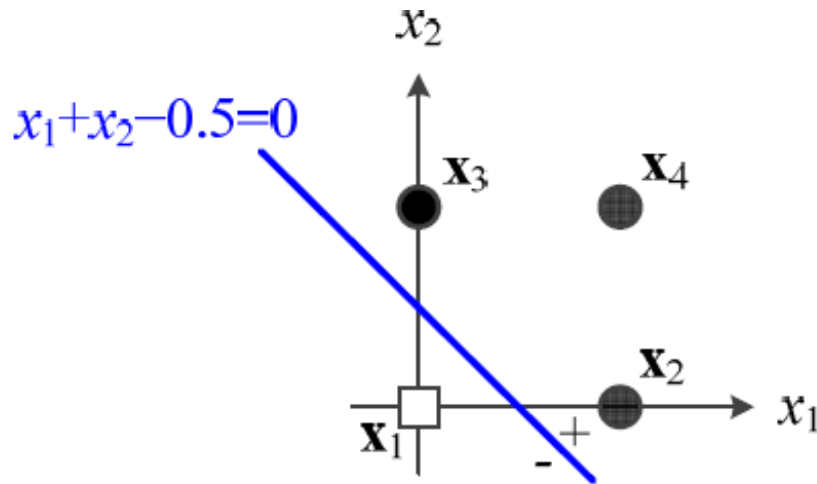


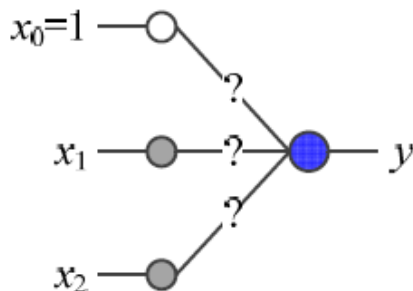
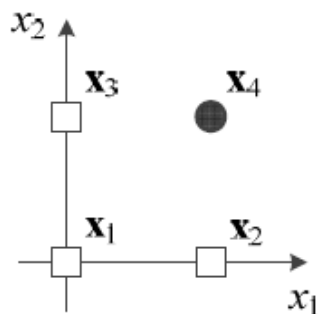
그림 3-5 [그림 3-4(b)]의 퍼셉트론에 해당하는 결정 직선

- d 차원 공간에서는 $d(\mathbf{x}) = w_1x_1 + w_2x_2 + \dots + w_dx_d + w_0 = 0$
 - 2차원은 결정 직선^{decision line}, 3차원은 결정 평면^{decision plane},
 - 4차원 이상은 결정 초평면^{decision hyperplane}

3.2.3 학습

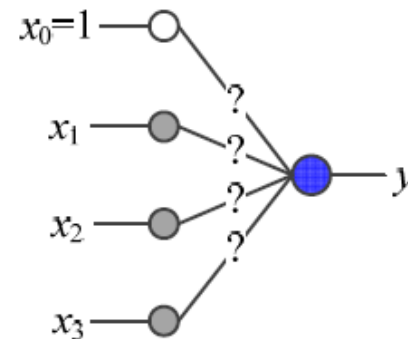
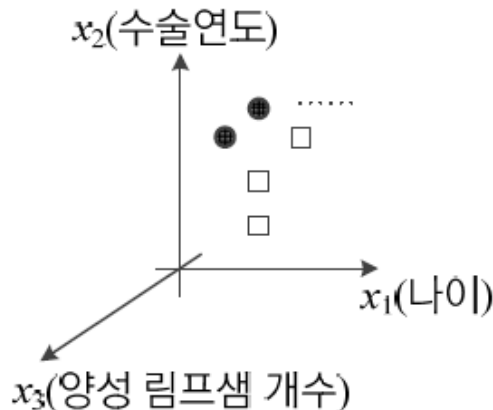
■ 학습 문제

- 지금까지는 **학습을 마친 퍼셉트론**을 가지고 **동작**을 설명한 셈
- [그림 3-6]은 **학습 문제**: w_1 과 w_2 , w_0 이 어떤 값을 가져야 100% 옳게 분류할까?
- [그림 3-6]은 2차원 공간에 4개 샘플이 있는 훈련집합이지만, 현실 세계는 d 차원 공간에 수백~수만 개의 샘플이 존재
(예, MNIST는 784차원에 6만개 샘플)



(a) AND 분류 문제

그림 3-6 어떻게 학습시킬 것인가?



(b) Haberman survival 분류 문제

UCI 데이터 (유방암 수술 생존 관련 데이터)

3.2.3 학습

■ 학습 문제

- 일반적인 분류기의 학습 수행 과정
 - 단계1: 분류기의 정의와 분류 과정의 수학적 정의
 - 단계2: 해당 분류기의 목적함수 $J(\theta)$ 정의
 - 단계3: $J(\theta)$ 를 최소화하는 θ 를 찾기 위한 최적화 방법 수행

3.2.3 학습

■ 목적함수 설계 (단계1과 단계2)

- 퍼셉트론의 매개변수를 $\mathbf{w} = (w_0, w_1, w_2, \dots, w_d)^T$ 라 표기하면, 매개변수 집합은 $\Theta = \{\mathbf{w}\}$

- 목적함수를 $J(\Theta)$ 또는 $J(\mathbf{w})$ 로 표기함

- 목적함수의 조건

- $J(\mathbf{w}) \geq 0$ 이다. (1)

- \mathbf{w} 가 최적이면, 즉 모든 샘플을 맞히면 $J(\mathbf{w}) = 0$ 이다. (2)

- 틀리는 샘플이 많은 \mathbf{w} 일수록 $J(\mathbf{w})$ 는 큰 값을 가진다. (3)

3.2.3 학습

■ 목적함수 설계 (단계1과 단계2)

- 식 (3.7)은 세 가지 조건을 만족하므로, 퍼셉트론의 목적함수로 적합
 - Y 는 \mathbf{w} 가 틀리는 샘플의 집합

$$J(\mathbf{w}) = \sum_{\mathbf{x}_k \in Y} -y_k (\mathbf{w}^T \mathbf{x}_k) \quad (3.7)$$

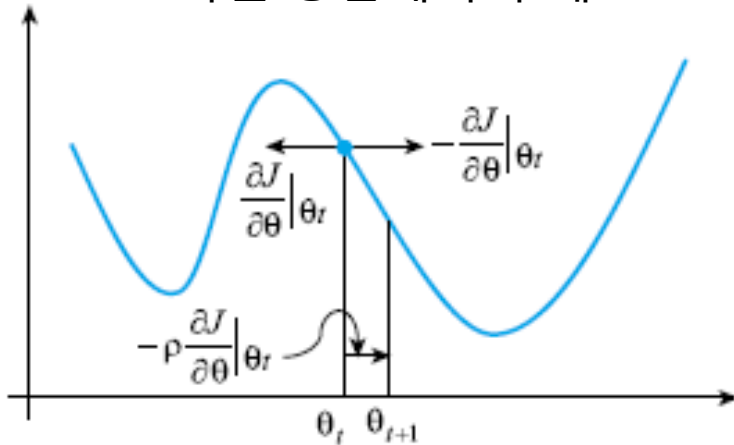
- 조건 (1), (2), (3)을 만족
 - 임의의 샘플 \mathbf{x}_k 가 Y 에 속한다면, 퍼셉트론의 예측값 $\mathbf{w}^T \mathbf{x}_k$ 와 실제값 y_k 는 부호가 다름
→ $-y_k(\mathbf{w}^T \mathbf{x}_k)$ 는 항상 양수를 가짐: **조건(1) 만족**
 - 결국 Y 가 클수록 (틀린 샘플이 많을수록), $J(\mathbf{w})$ 는 큰 값을 가짐: **조건 (3) 만족**
 - Y 가 공집합일 때 (퍼셉트론이 모든 샘플을 맞출 때), $J(\mathbf{w}) = 0$ 임: **조건 (2) 만족**

3.2.3 학습

■ 경사 하강법 gradient descent (3단계)

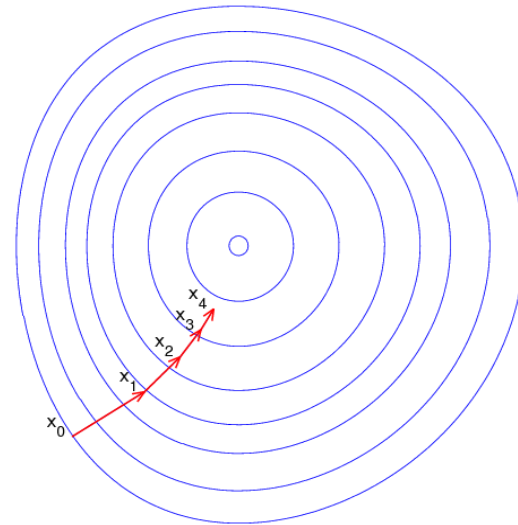
- 최소 $J(\theta)$ 기울기를 이용하여 반복 탐색하여 극값을 찾음

1차원 공간에서의 예



내리막 경사법

2차원 공간에서의 예



3.2.3 학습

■ 그레이디언트 계산

- 식 (2.58)의 **가중치 갱신 규칙** $\Theta = \Theta - \rho \mathbf{g}$ 를 적용하려면 그레이디언트 \mathbf{g} 가 필요
- 식 (3.7)을 **편미분**하면,

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} \frac{\partial(-y_k(w_0x_{k0} + w_1x_{k1} + \dots + w_ix_{ki} + \dots + w_dx_{kd}))}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} -y_kx_{ki}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} -y_kx_{ki}, \quad i = 0, 1, \dots, d \quad (3.8)$$

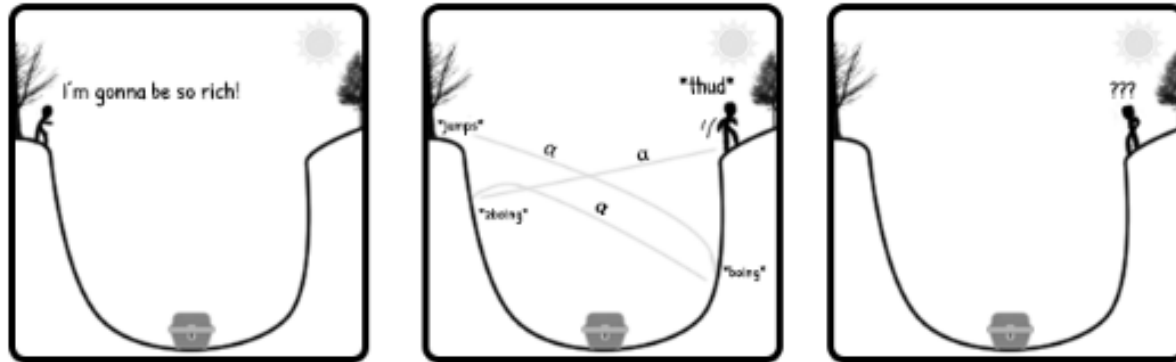
- x_{ki} 는 \mathbf{x}_k 의 i 번째 요소임
- $\mathbf{x}_k = (x_{k0}, x_{k1}, \dots, x_{kd})^T$
- 편미분 결과인 식 (3.8)을 식 (2.58)에 대입하면,

$$\text{델타 규칙: } w_i = w_i + \rho \sum_{\mathbf{x}_k \in Y} y_kx_{ki}, \quad i = 0, 1, \dots, d \quad (3.9)$$

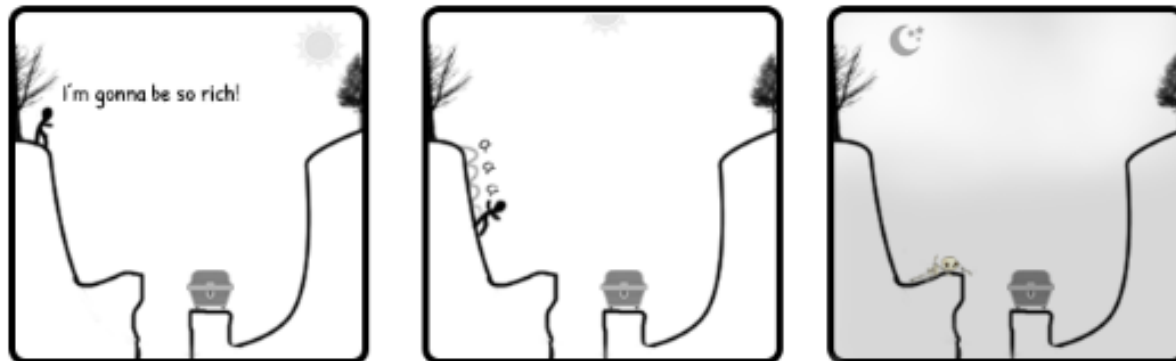
- 델타규칙(delta rule)은 퍼셉트론 학습 규칙

3.2.3 학습

■ 학습률 learning rate의 중요성



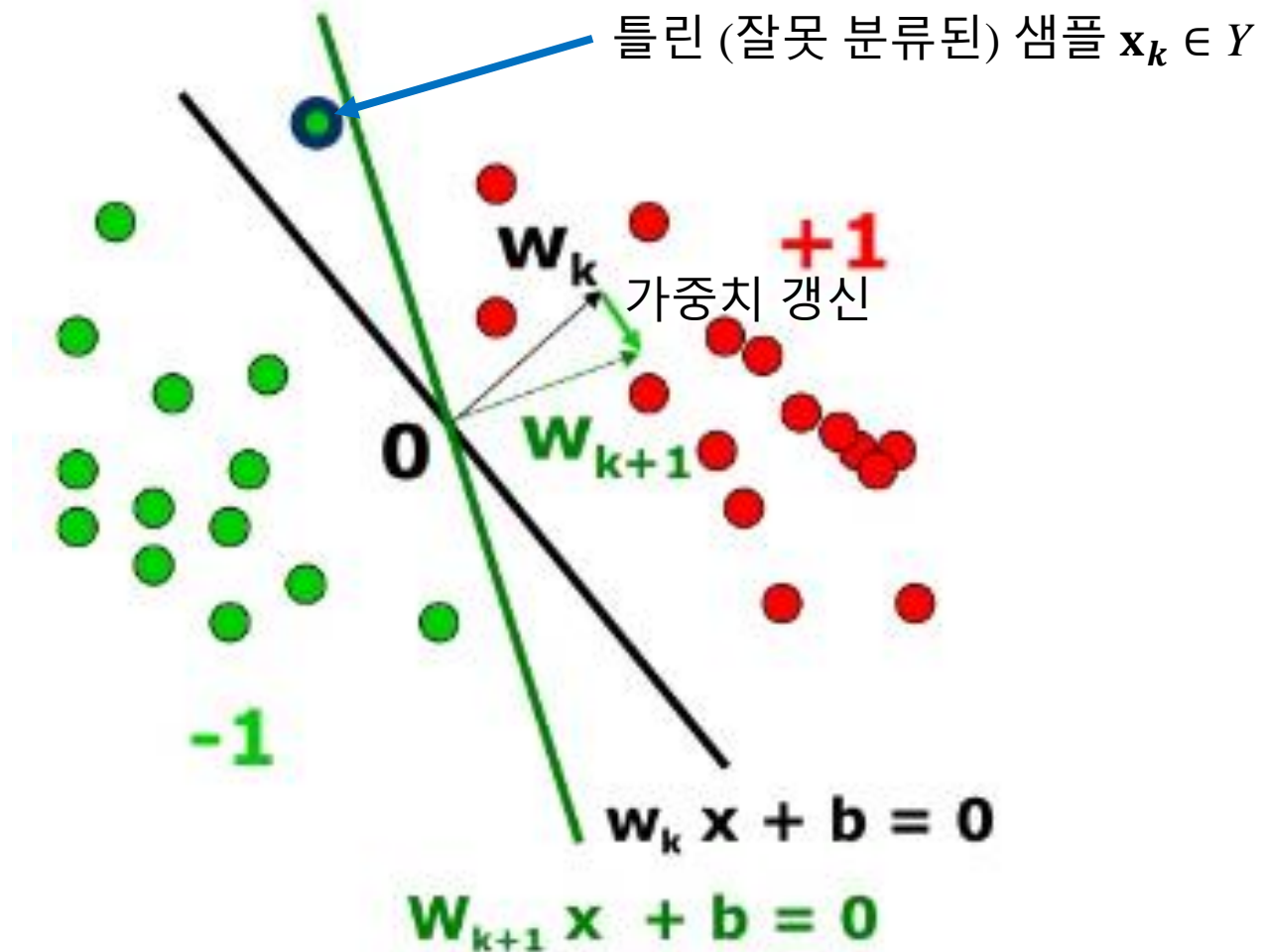
α too big



α too small

3.2.3 학습

■ 퍼셉트론 학습 알고리즘 동작



3.2.3 학습

■ 퍼셉트론 학습 알고리즘

- 식 (3.9)를 이용하여 학습 알고리즘을 쓰면,
 - 훈련집합의 샘플을 모두 맞출(즉 $Y = \emptyset$) 때까지 세대^{epoch}(라인 3~9)를 반복함

알고리즘 3-1 퍼셉트론 학습(배치 버전)

입력: 훈련집합 \mathbb{X} 와 \mathbb{Y} , 학습률 ρ

출력: 최적 가중치 $\hat{\mathbf{w}}$

```
1  난수를 생성하여 초기해  $\mathbf{w}$ 를 설정한다.
2  repeat
3       $Y = \emptyset$   // 틀린 샘플 집합
4      for  $j=1$  to  $n$ 
5           $y = \tau(\mathbf{w}^T \mathbf{x}_j)$           // 식 (3.4)
6          if( $y \neq y_j$ )  $Y = Y \cup \mathbf{x}_j$   // 틀린 샘플을 집합에 추가한다.
7      if( $Y \neq \emptyset$ )
8          for  $i=0$  to  $d$           // 식 (3.9)
9               $w_i = w_i + \rho \sum_{\mathbf{x}_k \in Y} y_k x_{ki}$ 
10 until ( $Y = \emptyset$ )
11  $\hat{\mathbf{w}} = \mathbf{w}$ 
```

3.2.3 학습

■ 퍼셉트론 학습 알고리즘의 **스토캐스틱 형태**

- 샘플 순서를 섞음. 틀린 샘플이 발생하면 즉시 갱신

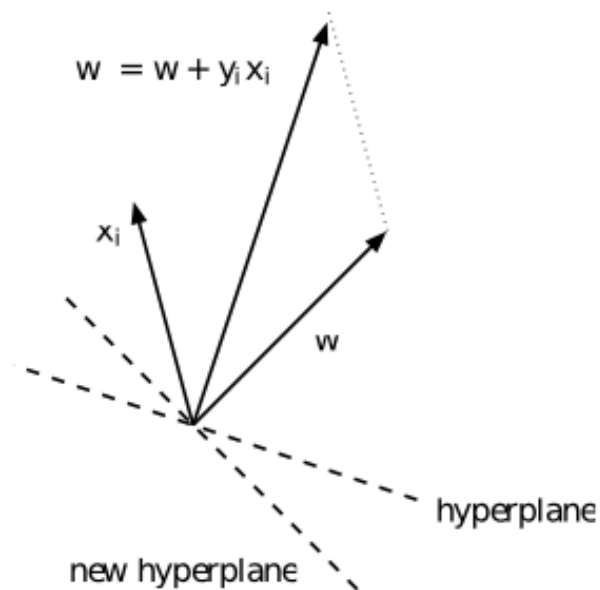
알고리즘 3-2 퍼셉트론 학습(스토캐스틱 버전)

입력: 훈련집합 \mathbb{X} 와 \mathbb{Y} , 학습률 ρ

출력: 최적 가중치 $\hat{\mathbf{w}}$

```
1  난수를 생성하여 초기해  $\mathbf{w}$ 을 설정한다.
2  repeat
3     $\mathbb{X}$ 의 샘플 순서를 섞는다.
4    quit=true
5    for  $j=1$  to  $n$ 
6       $y = \tau(\mathbf{w}^T \mathbf{x}_j)$  // 식 (3.4)
7      if( $y \neq y_j$ )
8        quit=false
9        for  $i=0$  to  $d$ 
10          $w_i = w_i + \rho y_j x_{ji}$ 
11  until(quit) // 틀린 샘플이 없을 때까지
12   $\hat{\mathbf{w}} = \mathbf{w}$ 
```

갱신의 기하학적 의미



3.2.3 학습

■ 행렬 표기

- 행렬을 사용하여 간결하게 표기: 델타 규칙: $\mathbf{w} = \mathbf{w} + \rho \sum_{\mathbf{x}_k \in Y} y_k \mathbf{x}_k$
- 행렬 표기로 [알고리즘 3-1]을 수정하면,
8. for $i = 0$ to d
9. $w_i = w_i + \rho \sum_{\mathbf{x}_k \in Y} y_k x_{ki}$ } \rightarrow 8. $\mathbf{w} = \mathbf{w} + \rho \sum_{\mathbf{x}_k \in Y} y_k \mathbf{x}_k$
- 행렬 표기로 [알고리즘 3-2]를 수정하면,
9. for $i = 0$ to d
10. $w_i = w_i + \rho y_j x_{ji}$ } \rightarrow 9. $\mathbf{w} = \mathbf{w} + \rho y_j \mathbf{x}_j$

■ 선형분리 불가능한 경우에는 무한 반복

- until($Y = \emptyset$) 또는 until(quit)를 until(더 이상 개선이 없다면)으로 수정해야 함

3.2.3 학습

■ 퍼셉트론 학습 예제

$$\mathbf{w}(0) = (-0.5, 0.75)^T, b(0) = 0.375$$

t_a : 샘플 a의 목표치 (실제값 == y_a)

$$\textcircled{1} d(\mathbf{x}) = -0.5x_1 + 0.75x_2 + 0.375$$

$Y = \{\mathbf{a}, \mathbf{b}\}$

$$\mathbf{w}(1) = \mathbf{w}(0) + 0.4(t_a \cdot \mathbf{a} + t_b \cdot \mathbf{b}) = \begin{pmatrix} -0.5 \\ 0.75 \end{pmatrix} + 0.4 \left[-\begin{pmatrix} 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix}$$

$$b(1) = b(0) + 0.4(t_a + t_b) = 0.375 + 0.4 * 0 = 0.375$$

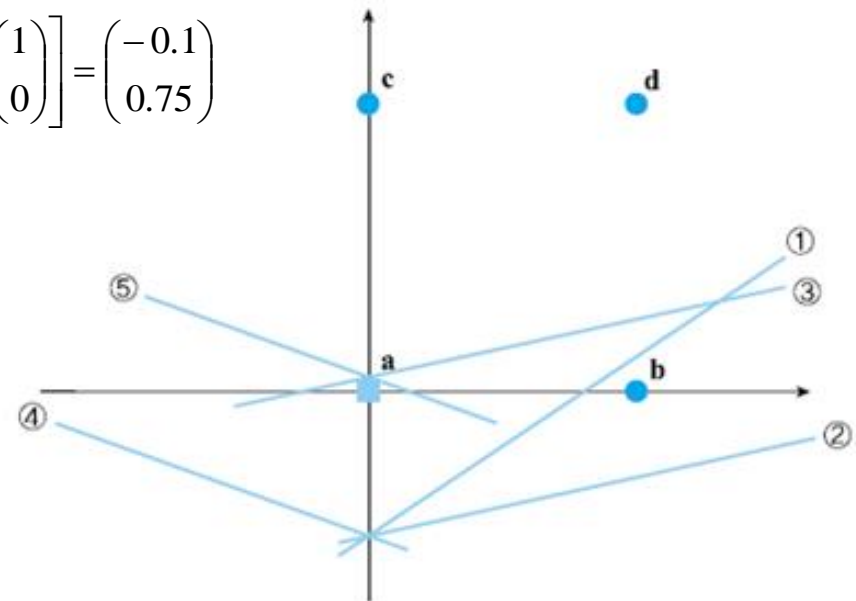
$$\textcircled{2} d(\mathbf{x}) = -0.1x_1 + 0.75x_2 + 0.375$$

$Y = \{\mathbf{a}\}$

$$\mathbf{w}(2) = \mathbf{w}(1) + 0.4(t_a \mathbf{a}) = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix} + 0.4 \left[-\begin{pmatrix} 0 \\ 0 \end{pmatrix} \right] = \begin{pmatrix} -0.1 \\ 0.75 \end{pmatrix}$$

$$b(2) = b(1) + 0.4(t_a) = 0.375 - 0.4 = -0.025$$

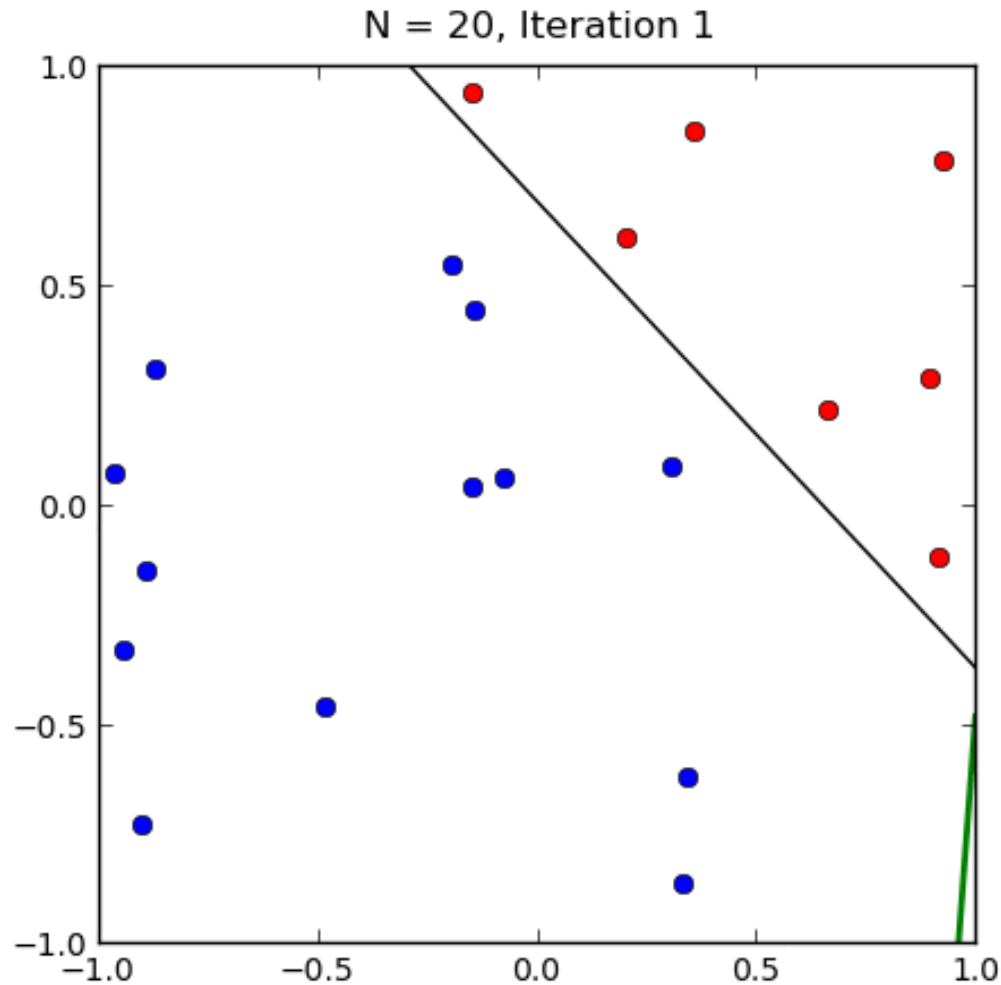
⋮



퍼셉트론 학습 과정의 시각화

3.2.3 학습

■ 퍼셉트론 학습 알고리즘의 동작 예



3.3 다층 퍼셉트론

- 3.3.1 특징 공간 변환
- 3.3.2 활성화함수
- 3.3.3 구조
- 3.3.4 동작

3.3 다층 퍼셉트론

■ 퍼셉트론은 선형 분류기(linear classifier)라는 한계

- [그림 3-7(b)]의 선형 분리 불가능한 상황에서는 일정한 양의 오류
- 예) XOR 문제에서는 75%가 정확률 한계

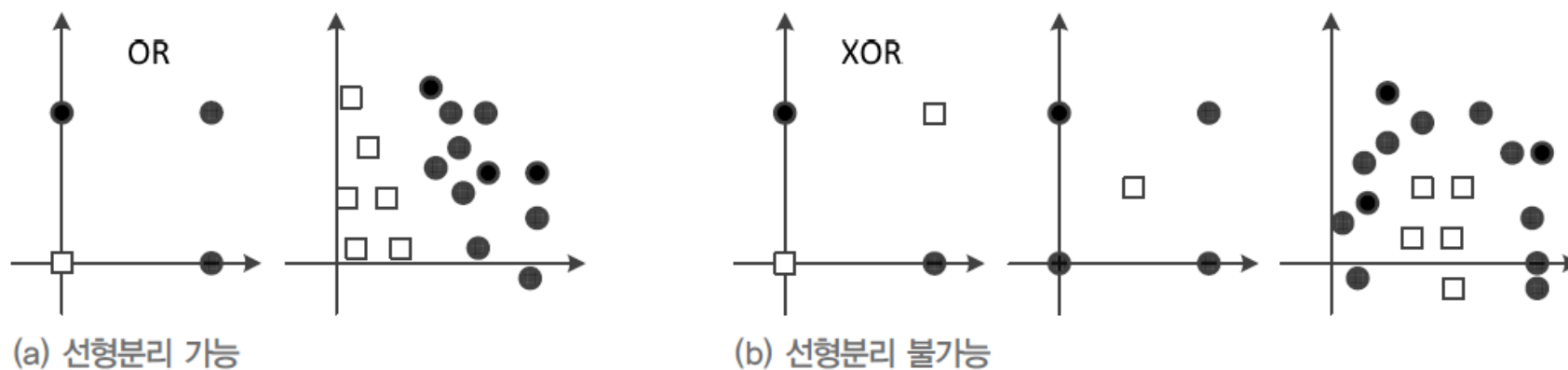


그림 3-7 선형분리가 가능한 상황과 불가능한 상황

- 1969년 민스키의 『Perceptrons』
 - 퍼셉트론의 한계를 지적하고 다층 구조를 이용한 극복 방안 제시.
당시 기술로 실현 불가능
- 1974년 웨어보스는 박사 논문에서 오류 역전파(error backpropagation) 알고리즘 제안
- 1986년 루멜하트의 저서 『Parallel Distributed Processing』 다층 퍼셉트론 이론 정립
신경망 부활

3.3 다층 퍼셉트론

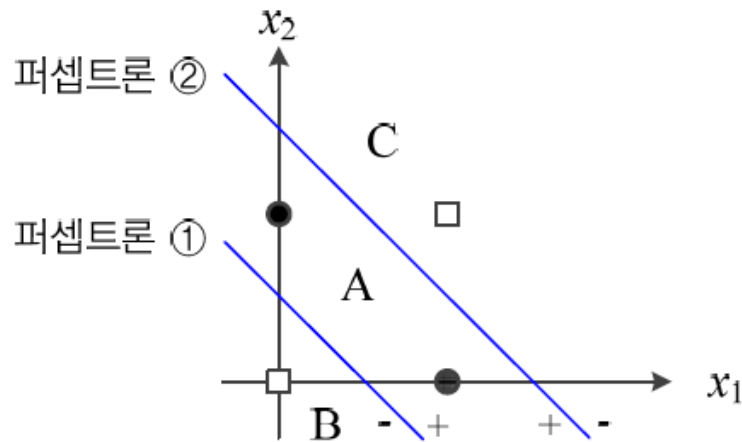
■ 다층 퍼셉트론의 핵심 아이디어

- 은닉층을 둔다. 은닉층은 원래 특징 공간을 분류하는 데 훨씬 유리한 새로운 특징 공간으로 변환한다. 3.3.1절에서 다층 퍼셉트론의 공간 변환 능력을 설명한다.
- 시그모이드 활성화함수를 도입한다. 퍼셉트론은 [그림 3-3(b)]의 계단함수를 활성화함수로 사용하였다. 이 함수는 경성^{hard} 의사결정에 해당한다. 반면, 다층 퍼셉트론은 연성^{soft} 의사결정이 가능한 [그림 3-12]의 시그모이드함수를 활성화함수로 사용한다. 연성에서는 출력이 연속값인데, 출력을 신뢰도로 간주함으로써 더 융통성 있게 의사결정을 할 수 있다. 3.3.2절에서 시그모이드 활성화함수를 자세히 설명한다.
- 오류 역전파 알고리즘을 사용한다. 다층 퍼셉트론은 여러 층이 순차적으로 이어진 구조이므로, 역방향으로 진행하면서 한 번에 한 층씩 그레디언트를 계산하고 가중치를 갱신하는 방식의 오류 역전파 알고리즘을 사용한다. 이 학습 알고리즘에 대해서는 3.4절에서 다룬다.

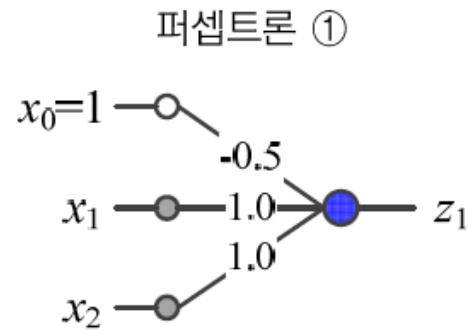
3.3.1 특징 공간 변환

■ 퍼셉트론 2개를 사용한 XOR 문제의 해결

- 퍼셉트론①과 퍼셉트론②가 모두 +1이면 ● 부류이고 그렇지 않으면 □ 부류임



(a) 퍼셉트론 2개를 이용한 공간분할



(b) 퍼셉트론 2개

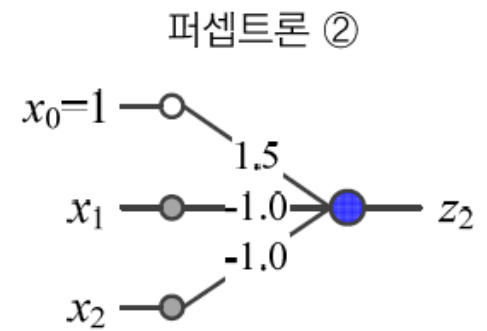
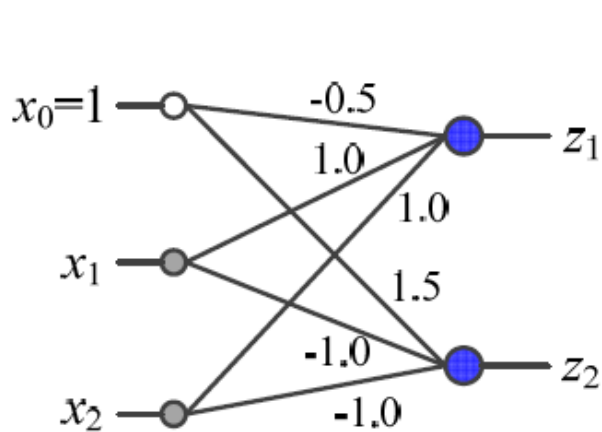


그림 3-8 XOR 문제의 해결

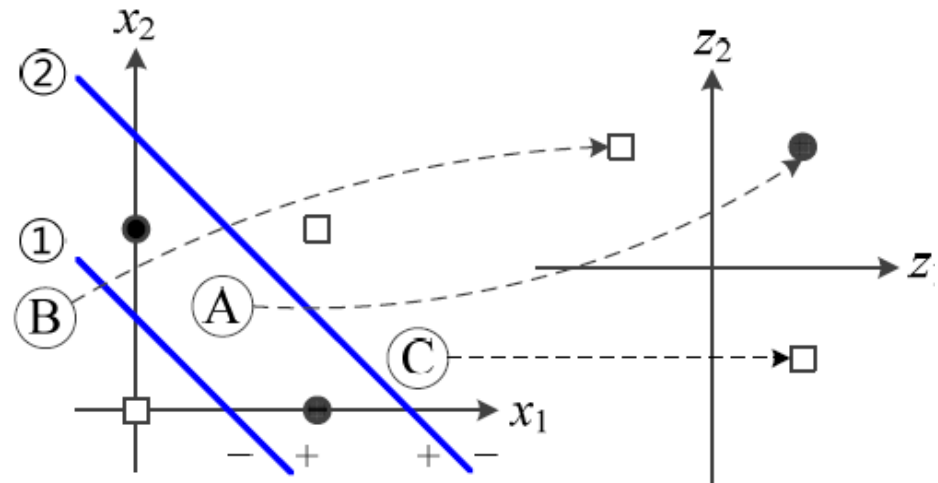
3.3.1 특징 공간 변환

■ 퍼셉트론 2개를 병렬 결합하면,

- 원래 공간 $\mathbf{x} = (x_1, x_2)^T$ 를 새로운 특징 공간 $\mathbf{z} = (z_1, z_2)^T$ 로 변환
- 새로운 특징 공간 \mathbf{z} 에서는 선형 분리 가능함



(a) 두 퍼셉트론을 병렬로 결합



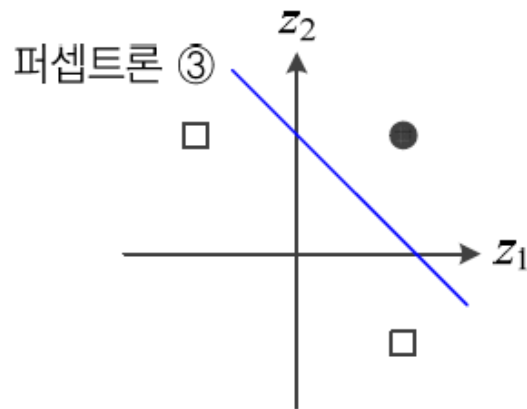
(b) 원래 특징 공간 \mathbf{x} 를 새로운 특징 공간 \mathbf{z} 로 변환

그림 3-9 특징 공간의 변환

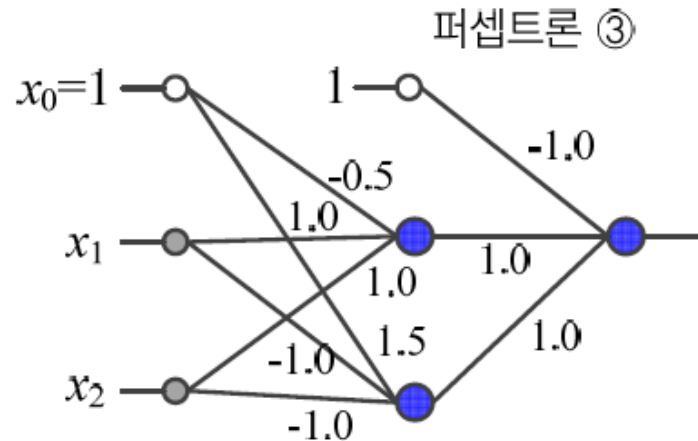
- 사람이 수작업으로 특징 학습을 수행한 것과 유사함

3.3.1 특징 공간 변환

- 이후, 퍼셉트론 1개를 순차 결합하면,
 - 새로운 특징 공간 z 에서 선형 분리를 수행하는 퍼셉트론③을 순차 결합하면,
[그림 3-10(b)]의 다층 퍼셉트론이 됨



(a) 새로운 특징 공간에서 분할



(b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

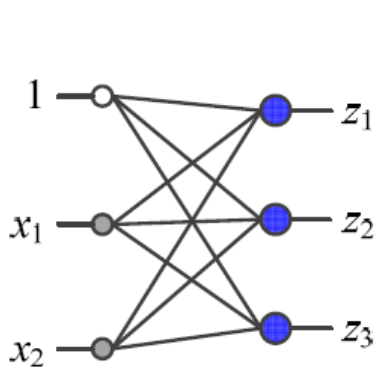
그림 3-10 다층 퍼셉트론

- 이 다층 퍼셉트론은 훈련집합에 있는 4개 샘플 $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ 을 제대로 분류하나?

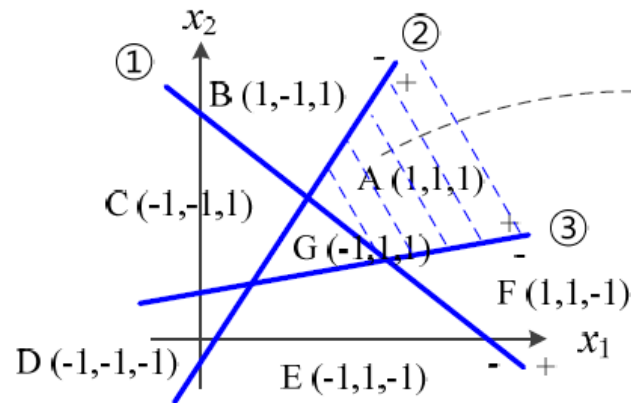
3.3.1 특징 공간 변환

■ 다층 퍼셉트론의 용량

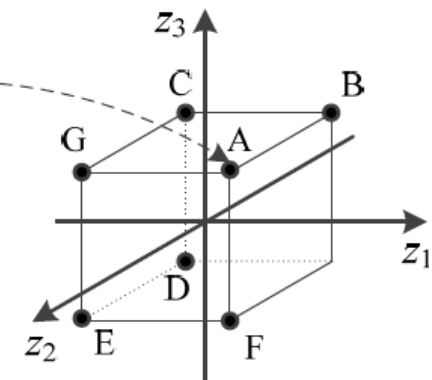
- [그림 3-11]처럼 3개 퍼셉트론을 결합하면, 2차원 공간을 7개 영역으로 나누고 각 영역을 3차원 점으로 변환
- 활성화함수 τ 로 계단함수를 사용하므로 영역을 점으로 변환



(a) 퍼셉트론 3개를 결합



(b) 7개 부분공간으로 나눔




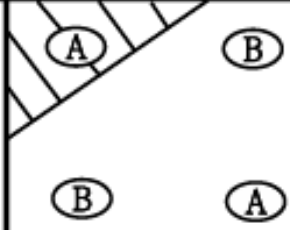
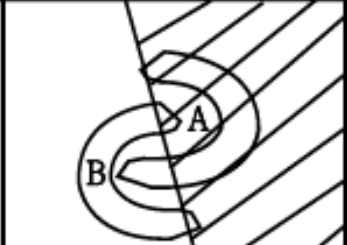
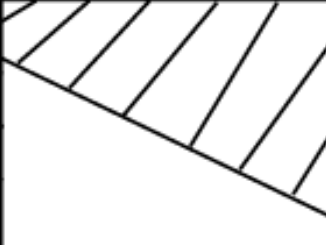

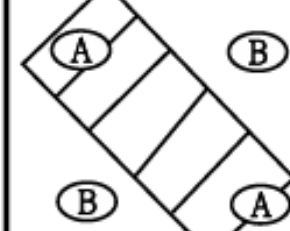
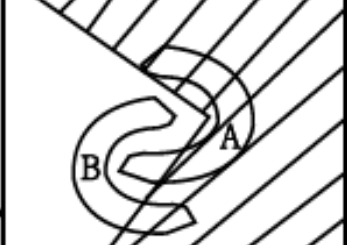


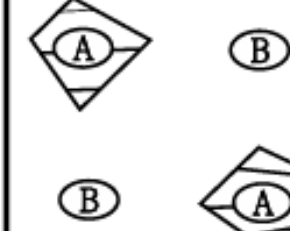
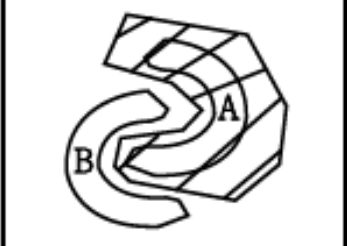
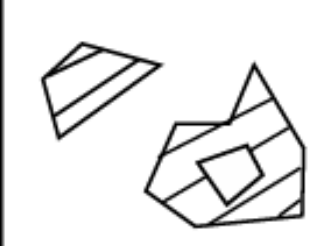
(c) 3차원 공간의 점으로 매핑

그림 3-11 퍼셉트론을 3개 결합했을 때 공간 변환

- 일반화하여, p 개 퍼셉트론을 결합하면 p 차원 공간으로 변환
 - $1 + \sum_{i=1}^p i$ 개의 영역으로 분할

3.3.1 특징 공간 변환

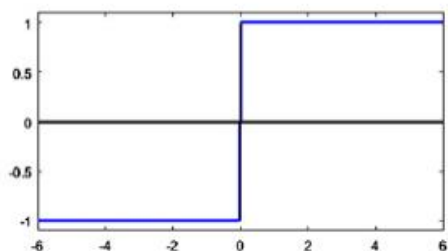
■ 다층 퍼셉트론의 용량

구 조	결정 구역	Exclusive -or	Classes with Meshed Regions	Most General Region Shapes
Single-layer j i 	Half Plane Bounded by Hyperplane			
Two-layer k j i 	Convex Open or Closed Regions			
Three-layer l k j i 	Arbitrary (Complexity limited by Number of Units)			

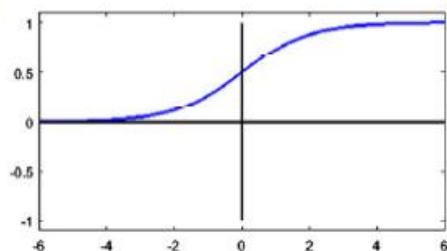
3.3.2 활성화함수

■ 딱딱한 공간 분할과 부드러운 공간 분할

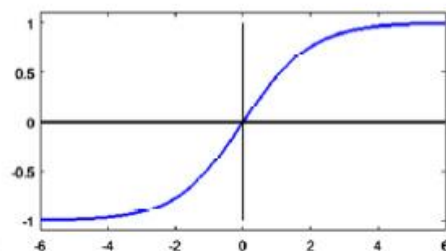
- 계단함수는 딱딱한 의사결정(영역을 점으로 변환).
- 나머지 활성화함수는 부드러운 의사결정(영역을 영역으로 변환)



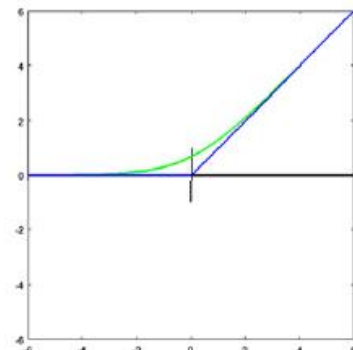
(a) 계단 함수



(b) 로지스틱 시그모이드

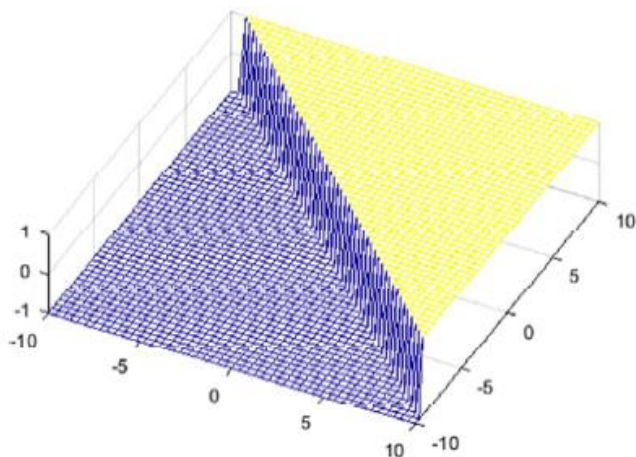


(c) 하이퍼볼릭 탄젠트 시그모이드

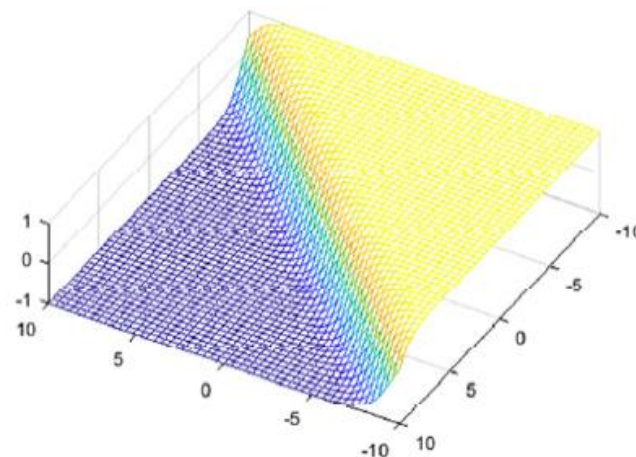


(d) softplus와 rectifier

그림 3-12 신경망이 사용하는 활성화함수



(a) 계단함수의 딱딱한 공간 분할



(b) 로지스틱 시그모이드의 부드러운 공간 분할

그림 3-13 퍼셉트론의 공간 분할 유형

3.3.2 활성화함수

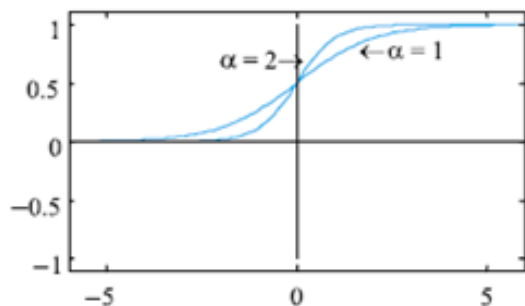
■ 대표적인 비선형 함수인 시그모이드를 활성화함수로 사용

이진 시그모이드 함수:

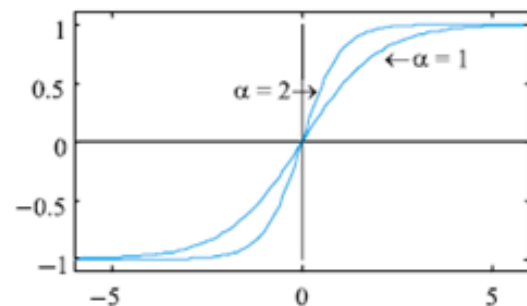
$$\left. \begin{aligned} \tau_1(x) &= \frac{1}{1+e^{-\alpha x}} \\ \tau_1'(x) &= \alpha \tau_1(x)(1-\tau_1(x)) \end{aligned} \right\}$$

양극 시그모이드 함수:

$$\left. \begin{aligned} \tau_2(x) &= \frac{2}{1+e^{-\alpha x}} - 1 \\ \tau_2'(x) &= \frac{\alpha}{2} (1+\tau_2(x))(1-\tau_2(x)) \end{aligned} \right\}$$



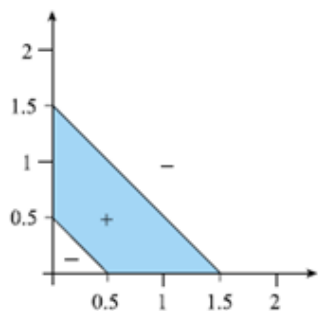
(a) 이진 시그모이드



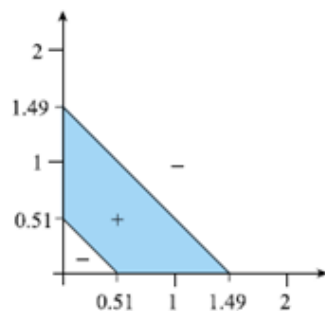
(b) 양극 시그모이드

활성 함수로 널리 사용되는 두 가지 시그모이드 함수

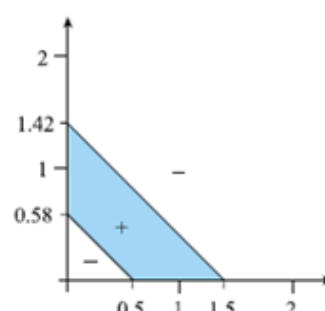
■ 활성화 함수에 따른 다층 퍼셉트론의 공간 분할 능력 변화



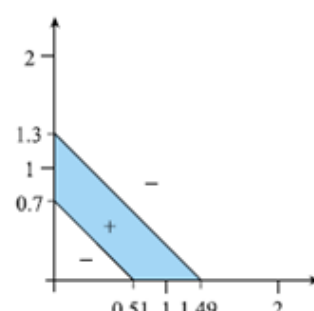
(a) 계단 함수 (양극 시그모이드 $\alpha=\infty$)



(b) 양극 시그모이드 $\alpha=5$



(c) 양극 시그모이드 $\alpha=3$



(d) 양극 시그모이드 $\alpha=2.5$

활성 함수에 따른 다층 퍼셉트론의 공간 분할 능력

3.3.2 활성화함수

■ 신경망이 사용하는 다양한 활성화함수

- 로지스틱 시그모이드와 하이퍼볼릭 탄젠트는 a 가 커질수록 계단함수에 가까워짐
- 모두 1차 도함수 계산이 빠름 (특히 ReLU는 비교 연산 한 번)

표 3-1 활성화함수로 사용되는 여러 함수

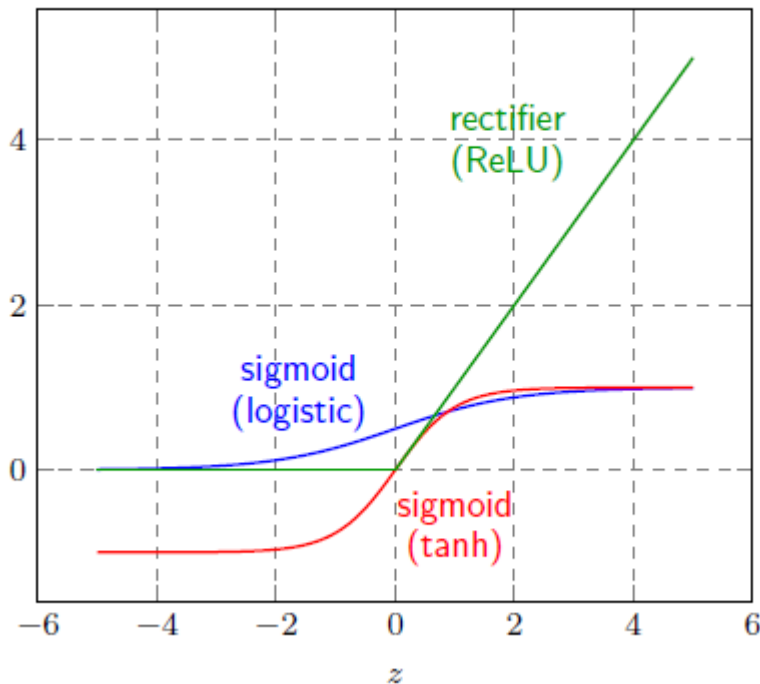
함수 이름	함수	1차 도함수	범위
계단	$\tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$	$\tau'(s) = \begin{cases} 0 & s \neq 0 \\ \text{불가} & s = 0 \end{cases}$	-1과 1
로지스틱 시그모이드	$\tau(s) = \frac{1}{1 + e^{-as}}$	$\tau'(s) = a\tau(s)(1 - \tau(s))$	(0,1)
하이퍼볼릭 탄젠트	$\tau(s) = \frac{2}{1 + e^{-as}} - 1$	$\tau'(s) = \frac{a}{2}(1 - \tau(s)^2)$	(-1,1)
소프트플러스	$\tau(s) = \log_e(1 + e^s)$	$\tau'(s) = \frac{1}{1 + e^{-s}}$	$(0, \infty)$
렉티파이어(ReLU)	$\tau(s) = \max(0, s)$	$\tau'(s) = \begin{cases} 0 & s < 0 \\ 1 & s > 0 \\ \text{불가} & s = 0 \end{cases}$	$[0, \infty)$

- 퍼셉트론은 계단함수, 다층 퍼셉트론은 로지스틱 시그모이드와 하이퍼볼릭 탄젠트, 딥러닝은 ReLU^{rectified linear activation}를 주로 사용

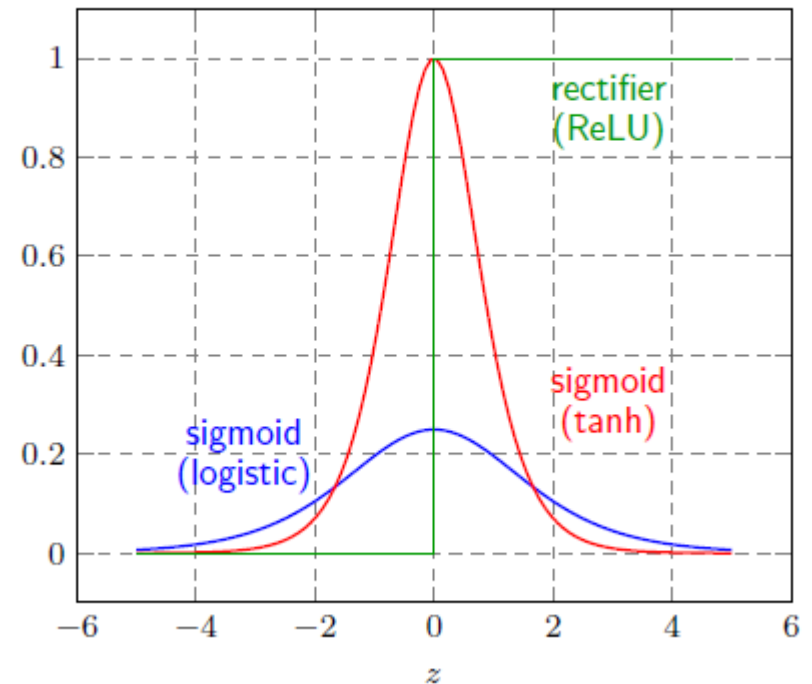
3.3.2 활성화함수

- 일반적으로 은닉층에서 로지스틱 시그모이드를 활성화 함수로 많이 사용
 - 시그모이드의 넓은 포화곡선은 그레디언트 기반한 학습을 어렵게 함
 - ReLU 대두됨

activation function: $g(z)$



gradient: $g'(z)$

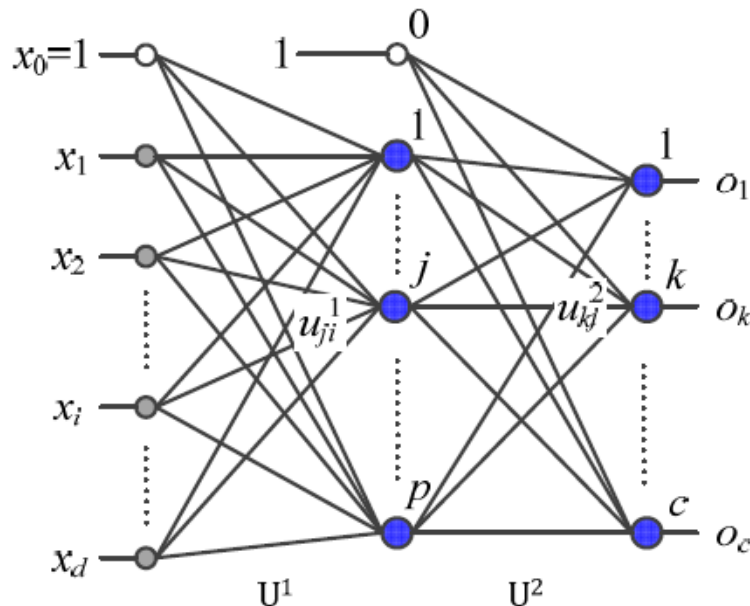


3.3.3 구조

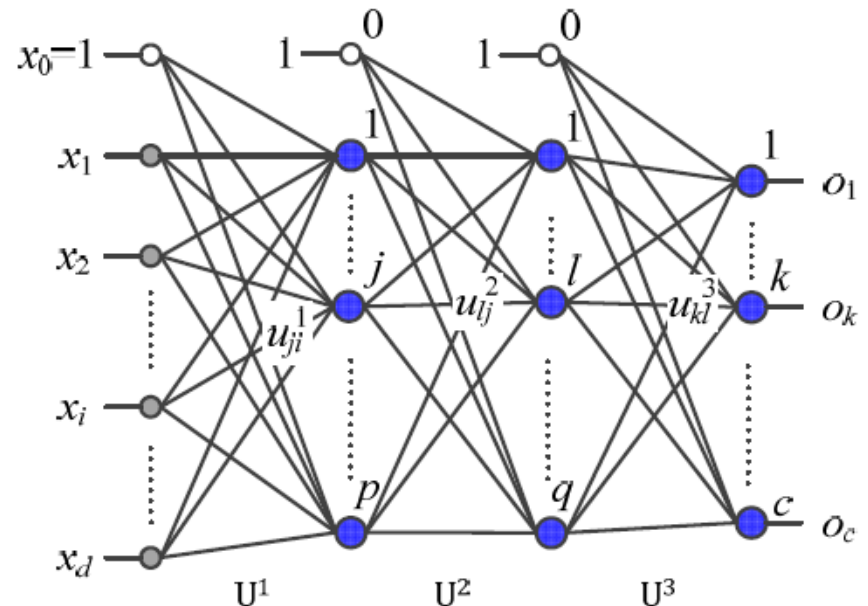
■ [그림 3-14(a)]는 입력층-은닉층-출력층의 2층 구조

- $d+1$ 개의 입력 노드 (d 는 특징의 개수). c 개의 출력 노드 (c 는 부류 개수)
- p 개의 은닉 노드: p 는 하이퍼 매개변수(사용자가 정해주는 매개변수)
 - p 가 너무 크면 과잉적합, 너무 작으면 과소적합 → 5.5절의 하이퍼 매개변수 hyper-parameter 최적화

■ [그림 3-14(b)]는 입력층-은닉층-은닉층-출력층의 3층 구조



(a) 2층 퍼셉트론



(b) 3층 퍼셉트론

그림 3-14 다층 퍼셉트론의 구조

3.3.3 구조

■ 다층 (2층) 퍼셉트론의 매개변수 (가중치)

- **입력층-은닉층**을 연결하는 \mathbf{U}^1 (u_{ji}^1 은 입력층의 i 번째 노드를 은닉층의 j 번째 노드와 연결)
← j
- **은닉층-출력층**을 연결하는 \mathbf{U}^2 (u_{kj}^2 은 은닉층의 j 번째 노드를 출력층의 k 번째 노드와 연결)
← k

2층 퍼셉트론의 가중치 행렬:

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}, \quad \mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} \quad (3.11)$$

- 일반화하면 u_{ji}^l 은 $l-1$ 번째 은닉층의 i 번째 노드를 l 번째 은닉층의 j 번째 노드와 연결하는 가중치
 - 입력층을 0번째 은닉층, 출력층을 마지막 은닉층으로 간주

3.3.4 동작

- 특징 벡터 \mathbf{x} 를 출력 벡터 \mathbf{o} 로 매핑(mapping)하는 함수로 간주할 수 있음

$$\left. \begin{array}{l} \text{2층 퍼셉트론: } \mathbf{o} = \mathbf{f}(\mathbf{x}) = \mathbf{f}_2(\mathbf{f}_1(\mathbf{x})) \\ \text{3층 퍼셉트론: } \mathbf{o} = \mathbf{f}(\mathbf{x}) = \mathbf{f}_3(\mathbf{f}_2(\mathbf{f}_1(\mathbf{x}))) \end{array} \right\} \quad (3.12)$$

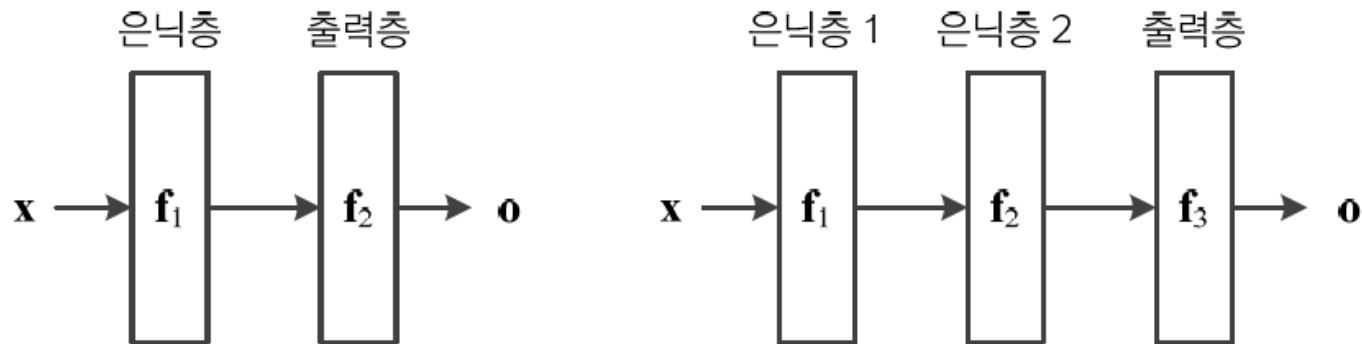


그림 3-15 다층 퍼셉트론을 간략화한 구조

- 깊은 신경망(deep neural networks)은 $\mathbf{o} = \mathbf{f}_L(\cdots \mathbf{f}_2(\mathbf{f}_1(\mathbf{x})))$, $L \geq 4$

← 4장의 주제(딥러닝)

3.3.4 동작

- 노드가 수행하는 연산을 구체적으로 쓰면,

j 번째 은닉 노드의 연산:

$$z_j = \tau(zsum_j), j = 1, 2, \dots, p \quad (3.13)$$

이때 $zsum_j = \mathbf{u}_j^1 \mathbf{x}$ 이고 $\mathbf{u}_j^1 = (u_{j0}^1, u_{j1}^1, \dots, u_{jd}^1)$, $\mathbf{x} = (1, x_1, x_2, \dots, x_d)^T$

k 번째 출력 노드의 연산:

$$o_k = \tau(osum_k), k = 1, 2, \dots, c \quad (3.14)$$

이때 $osum_k = \mathbf{u}_k^2 \mathbf{z}$ 이고 $\mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \dots, u_{kp}^2)$, $\mathbf{z} = (1, z_1, z_2, \dots, z_p)^T$

- \mathbf{u}_j^1 은 j 번째 은닉 노드에 연결된 가중치 벡터 (식 (3.11)의 \mathbf{U}^1 의 j 번째 행)
- \mathbf{u}_k^2 는 k 번째 출력 노드에 연결된 가중치 벡터 (식 (3.11)의 \mathbf{U}^2 의 k 번째 행)

- 다층 퍼셉트론의 동작을 행렬로 표기하면,

$$\mathbf{o} = \tau(\mathbf{U}^2 \tau_h(\mathbf{U}^1 \mathbf{x})) \quad (3.15)$$

3.3.4 동작

■ 은닉층은 특징 추출기

- 은닉층은 특징 벡터를 분류에 더 유리한 새로운 특징 공간으로 변환
- 현대 기계 학습에서는 특징 학습이라 feature learning 혹은 data-driven features 부름
(딥러닝은 더 많은 단계를 거쳐 특징학습을 함)

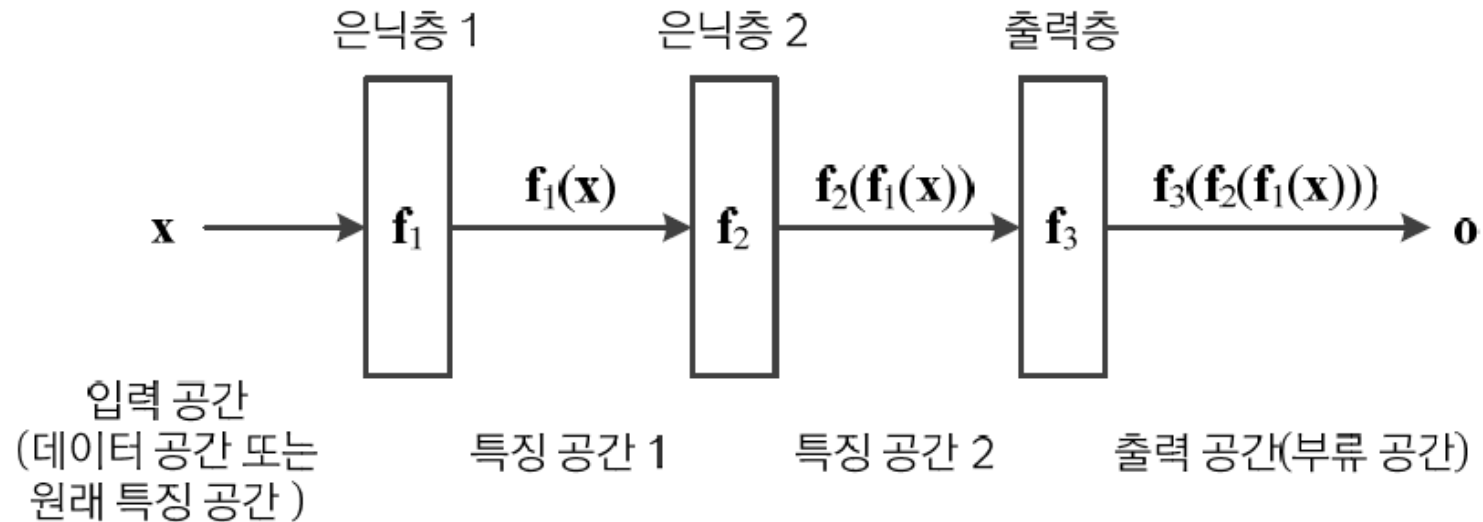
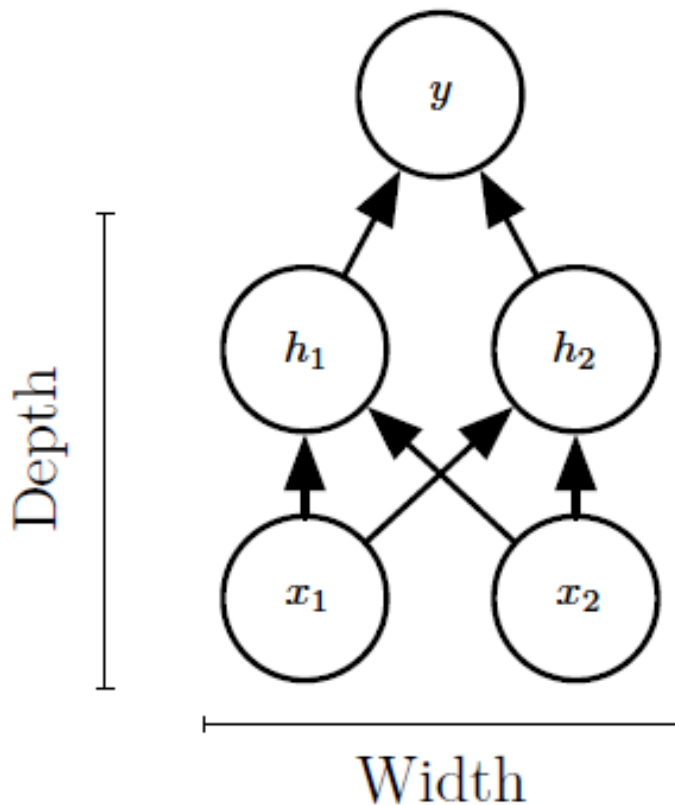


그림 3-16 특징 추출기로서의 은닉층

3.3.3 구조

■ 기본 구조

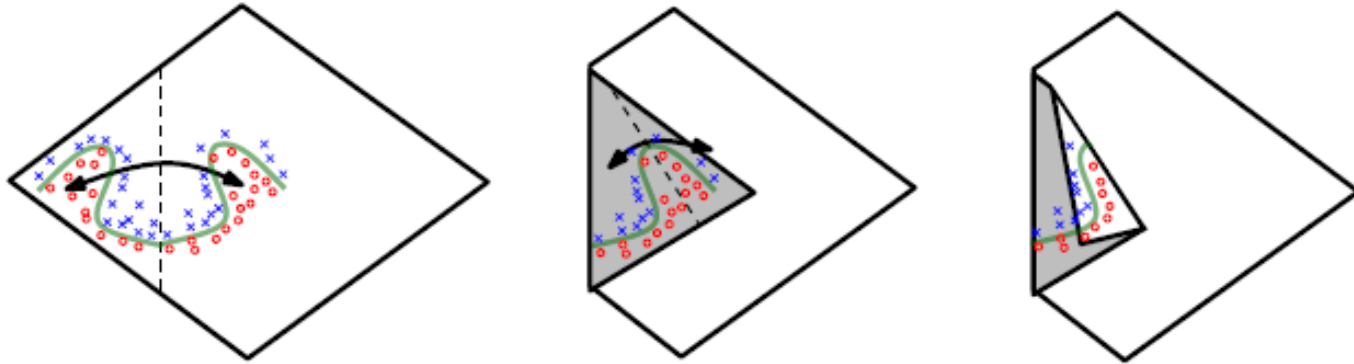


- 범용적 근사 이론 universal approximation theorem
 - 하나의 은닉층은 함수의 근사를 표현
- 다층 퍼셉트론도 공간을 변환하는 함수를 근사함
- 얇은 은닉층의 구조보다
 - 지수적으로 더 넓은 폭^{width}이 필요할 수 있음
 - 더 과잉적합 되기 쉬움
- 일반적으로 깊은 은닉층의 구조가 좋은 성능을 가짐

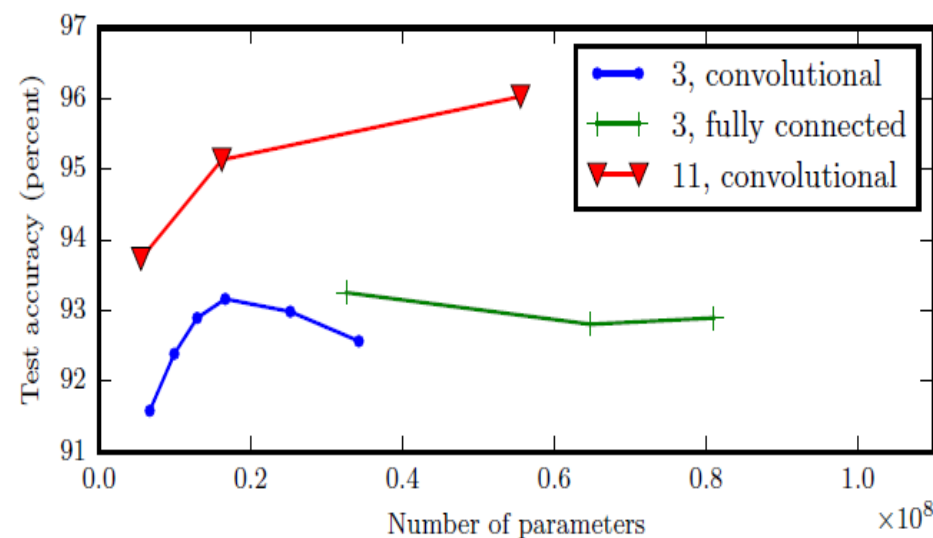
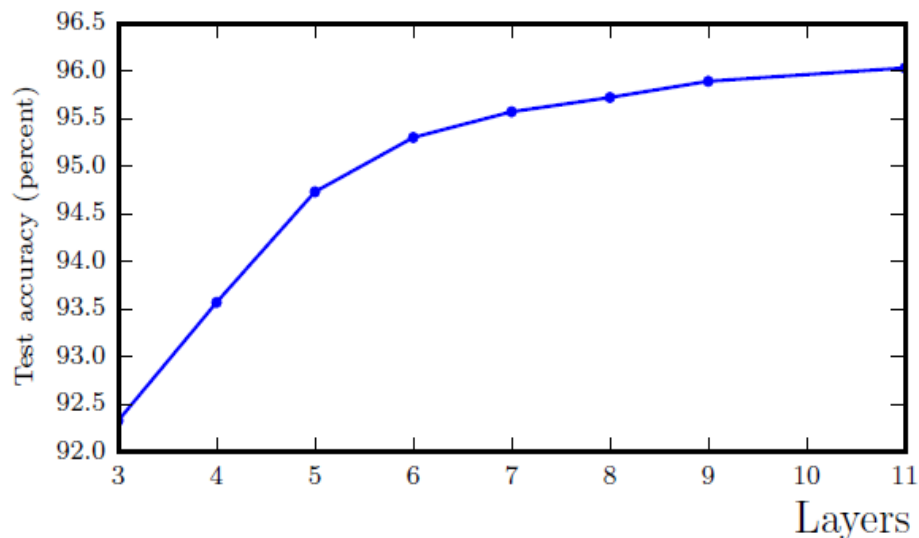
3.3.3 구조

■ 은닉층의 깊이에 따른 이점

- 지수의 표현 exponential representation



- 각 은닉층은 입력 공간을 어디서 접을지 지정 → 지수적으로 많은 선형적인 영역 조각들
- 일반화 성능 향상과 과잉적합 해소



3.4 오류 역전파 알고리즘

- 3.4.1 목적함수의 정의
- 3.4.2 오류 역전파 알고리즘 설계
- 3.4.3 오류 역전파를 이용한 학습 알고리즘

3.4.1 목적함수의 정의

■ 훈련집합

- 특징 벡터 집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ 과 부류 벡터 집합 $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n\}$
- 부류 벡터는 원핫one-hot 코드로 표현됨. 즉 $\mathbf{y}_i = (0, 0, \dots, 1, \dots, 0)^T$
- 설계 행렬로 쓰면,

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{pmatrix}, \quad \mathbf{Y} = \begin{pmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_n^T \end{pmatrix} \quad (3.16)$$

■ 기계 학습의 목표

- 모든 샘플을 옳게 분류하는 (식 (3.17)을 만족하는) 함수 \mathbf{f} 를 찾는 일

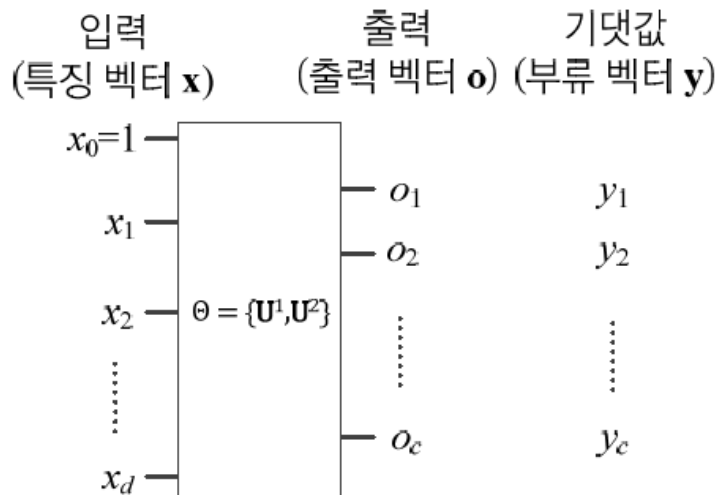
$$\left. \begin{array}{l} \mathbf{Y} = \mathbf{f}(\mathbf{X}) \\ \text{풀어 쓰면 } \mathbf{y}_i = \mathbf{f}(\mathbf{x}_i), i = 1, 2, \dots, n \end{array} \right\} \quad (3.17)$$

3.4.1 목적함수의 정의

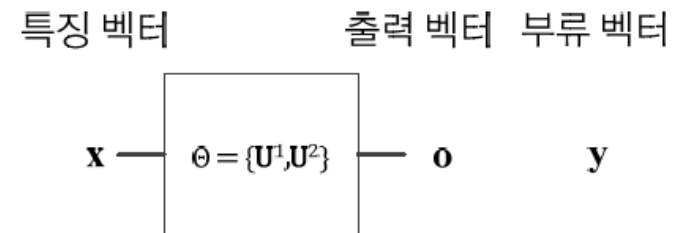
■ 목적함수

- **평균 제곱 오차** mean squared error(MSE)로 정의

$$\left. \begin{array}{l} \text{온라인 모드: } e = \frac{1}{2} \|\mathbf{y} - \mathbf{o}\|_2^2 \\ \text{배치 모드: } e = \frac{1}{2n} \sum_{i=1}^n \|\mathbf{y}_i - \mathbf{o}_i\|_2^2 \end{array} \right\} \quad (3.19)$$



(a) 블록 다이어그램



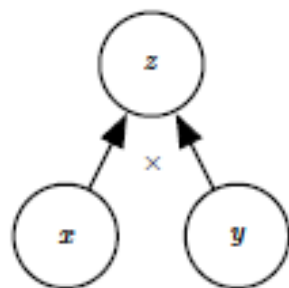
(b) 축약형 블록 다이어그램

그림 3-17 목적함수 정의에 사용하는 입력과 출력, 기댓값

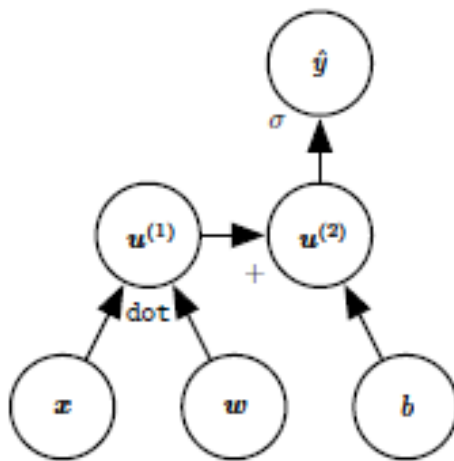
3.4.1 목적함수의 정의

■ 연산 그래프 computational graph의 예

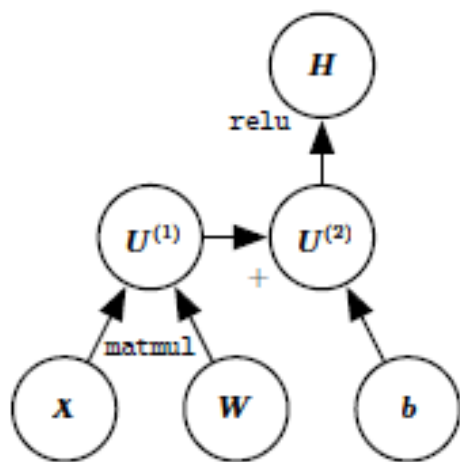
- 연산을 그래프로 표현



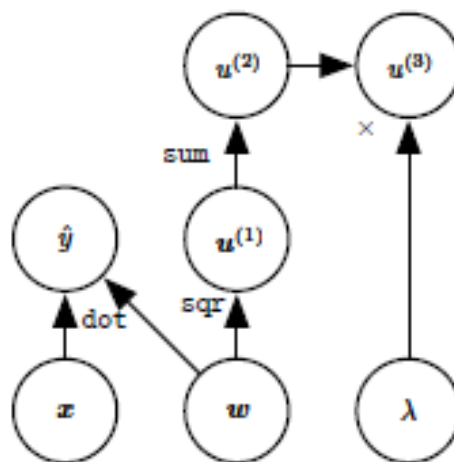
(a)



(b)



(c)



(d)

(a) $z = xy$

(b) logistic regression

▶ $\hat{y} = \sigma(x^T w + b)$

(c) ReLU activation

▶ $H = \max\{0, XW + b\}$

(d) linear regression

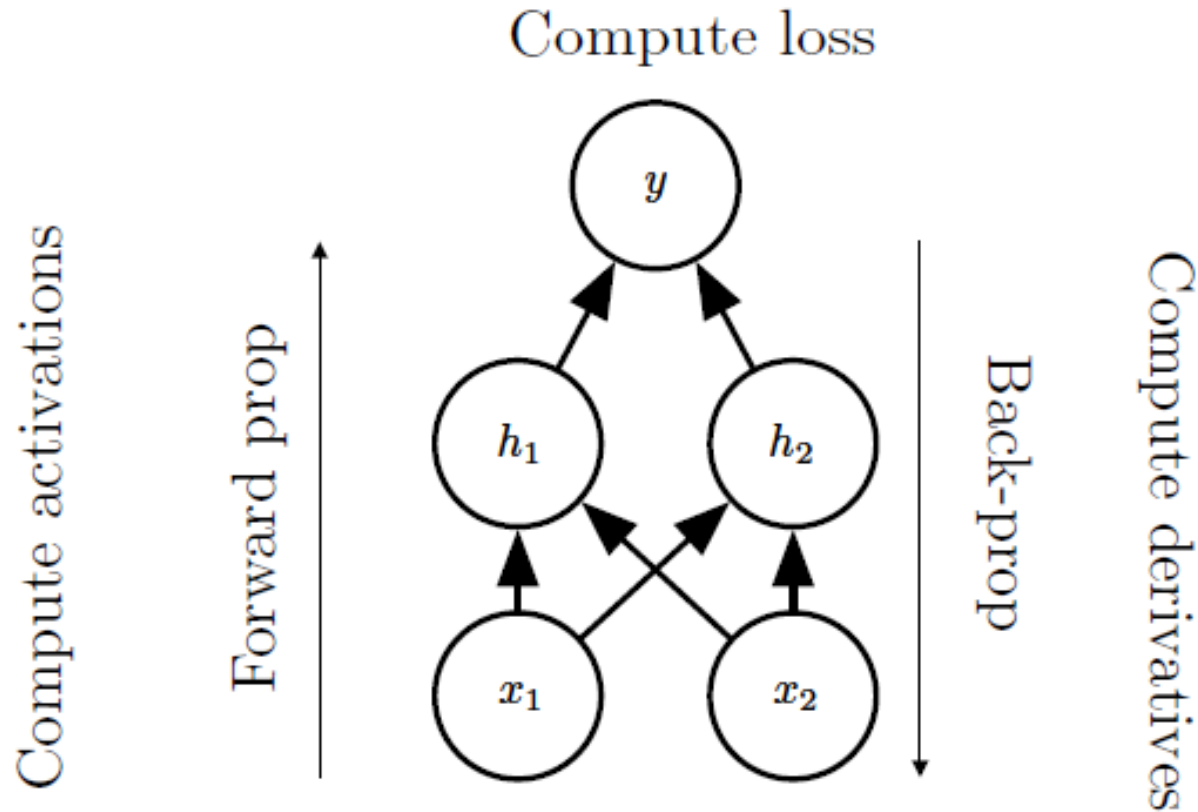
▶ w : weights

▶ \hat{y} : prediction

▶ $\lambda \sum_i w_i^2$: weight decay penalty

3.4.2 오류 역전파 알고리즘의 설계

- 간단한 오류 역전파의 연산 그래프 예



3.4.2 오류 역전파 알고리즘의 설계

■ 오류 역전파(error back-propagation) 미분의 연쇄 법칙을 이용

■ 연쇄 법칙

• 수인 경우, $\frac{dz}{dx} = \frac{dz}{dy} \frac{dy}{dx}$.

• 벡터인 경우, $\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \right)^{\top} \nabla_{\mathbf{y}} z$.

$\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$ 라고 가정하면

$$g : \mathbb{R}^m \mapsto \mathbb{R}^n$$

$$f : \mathbb{R}^n \mapsto \mathbb{R}$$

$$\rightarrow \mathbf{y} = g(\mathbf{x}) : \mathbf{x} \in \mathbb{R}^m \mapsto \mathbb{R}^n \ni \mathbf{y}$$

$$\begin{aligned} \frac{\partial \mathbf{y}}{\partial \mathbf{x}} &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \in \mathbb{R}^{n \times m} \end{aligned}$$

3.4.2 오류 역전파 알고리즘의 설계

$$\begin{aligned}
 \nabla_y z &= \begin{bmatrix} \frac{\partial z}{\partial y_1} & \cdots & \frac{\partial z}{\partial y_n} \end{bmatrix}^T \in \mathbb{R}^{n \times 1} \\
 \left(\frac{\partial y}{\partial x} \right)^T \nabla_y z &= \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_n}{\partial x_1} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_1}{\partial x_m} & \cdots & \frac{\partial y_n}{\partial x_m} \end{bmatrix} \begin{bmatrix} \frac{\partial z}{\partial y_1} \\ \vdots \\ \frac{\partial z}{\partial y_n} \end{bmatrix} \\
 &= \begin{bmatrix} \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x_1} + \cdots + \frac{\partial z}{\partial y_n} \frac{\partial y_n}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial y_1} \frac{\partial y_1}{\partial x_m} + \cdots + \frac{\partial z}{\partial y_n} \frac{\partial y_n}{\partial x_m} \end{bmatrix} = \begin{bmatrix} \frac{\partial z}{\partial x_1} \\ \vdots \\ \frac{\partial z}{\partial x_m} \end{bmatrix} = \nabla_x z
 \end{aligned}$$

- 야코비안 행렬과 그레디언트를 곱한 연쇄 법칙을 얻어서 구해짐

$$\nabla_x z = \left(\frac{\partial y}{\partial x} \right)^T \nabla_y z$$

$\underbrace{x \in \mathbb{R}^m \xrightarrow{g} y \in \mathbb{R}^n}_{\text{Jacobian: } \frac{\partial y}{\partial x}}$

$\underbrace{y \in \mathbb{R}^n \xrightarrow{f} \mathbb{R} \ni z}_{\text{gradient: } \nabla_y z}$

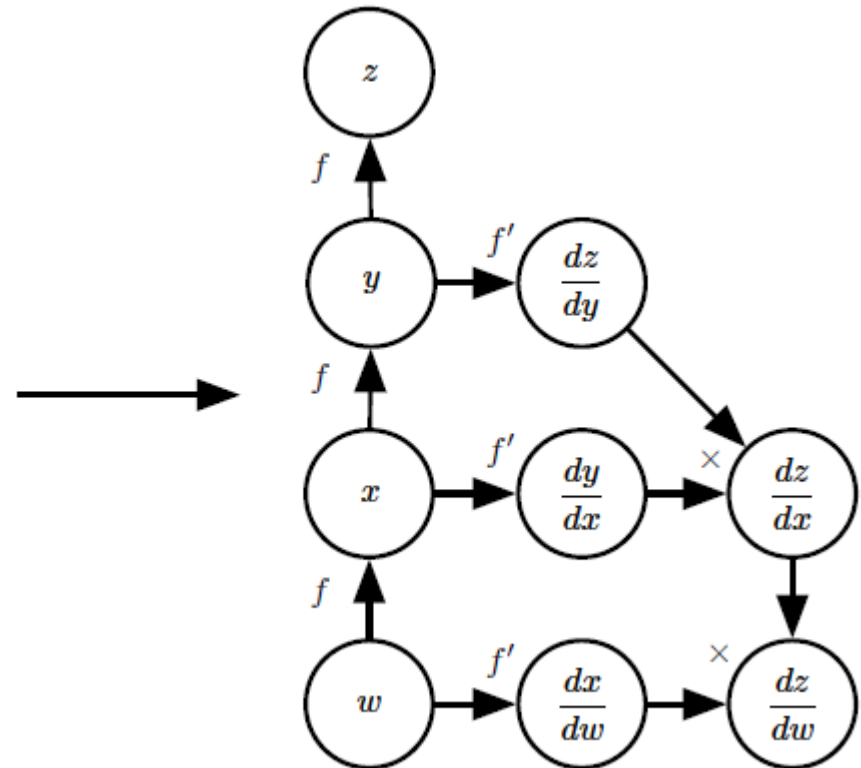
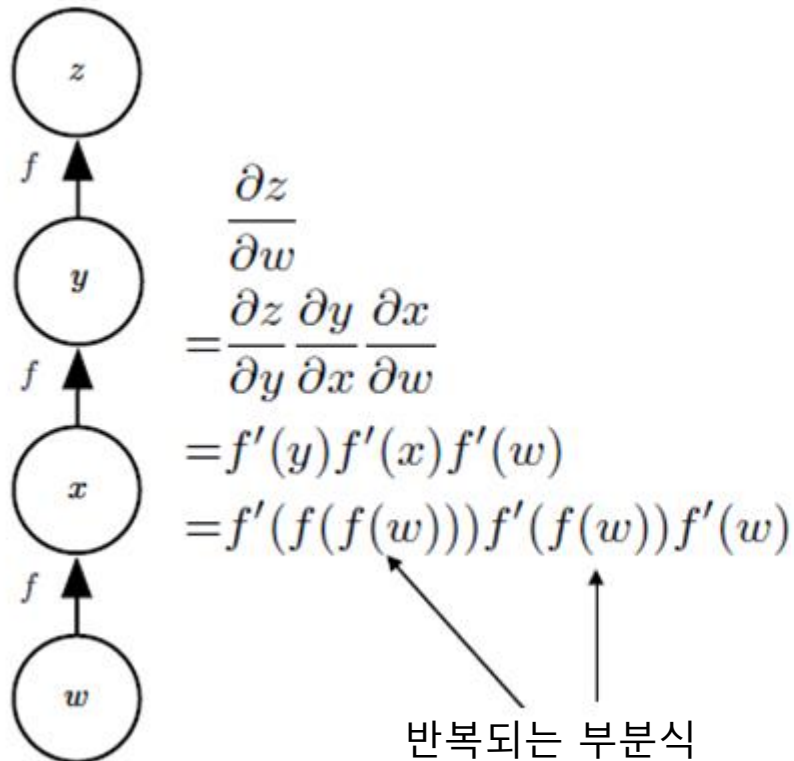
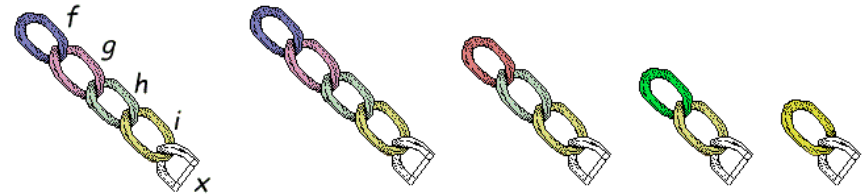
3.4.2 오류 역전파 알고리즘의 설계

■ 연쇄 법칙의 구현

- 반복되는 부분식들 subexpressions을 저장하거나 재연산을 최소화

- 예. 동적 프로그래밍 dynamic programming
- 연산 속도와 저장 공간의 트레이드오프
- 그래디언트 계산을 위한 연산 그래프 예

$$(f \circ g \circ h \circ i)'(x) = (f' \circ g \circ h \circ i)(x) \times (g' \circ h \circ i)(x) \times (h' \circ i)(x) \times i'(x)$$



3.4.2 오류 역전파 알고리즘의 설계

- 식 (3.19)의 목적함수를 다시 쓰면,

- 2층 퍼셉트론의 경우 $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$

$$J(\Theta) = \frac{1}{2} \|\mathbf{y} - \mathbf{o}(\Theta)\|_2^2 \quad (3.20)$$

- $J(\Theta) = J(\{\mathbf{U}^1, \mathbf{U}^2\})$ 의 최저점을 찾아주는 경사 하강법

$$\left. \begin{aligned} \mathbf{U}^1 &= \mathbf{U}^1 - \rho \frac{\partial J}{\partial \mathbf{U}^1} \\ \mathbf{U}^2 &= \mathbf{U}^2 - \rho \frac{\partial J}{\partial \mathbf{U}^2} \end{aligned} \right\} \quad (3.21)$$

3.4.2 오류 역전파 알고리즘의 설계

- 식 (3.21)을 알고리즘 형태로 쓰면,

알고리즘 3-3 다층 퍼셉트론을 위한 스토케스틱 경사 하강법

입력: 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, $\mathbb{Y} = \{y_1, y_2, \dots, y_n\}$, 학습률 ρ

출력: 가중치 행렬 \mathbf{U}^1 과 \mathbf{U}^2

```
1   $\mathbf{U}^1$ 과  $\mathbf{U}^2$ 를 초기화한다.
2  repeat
3       $\mathbb{X}$ 의 순서를 섞는다.
4      for ( $\mathbb{X}$ 의 샘플 각각에 대해)
5          식 (3.15)로 전방 계산을 하여  $\mathbf{o}$ 를 구한다.
6           $\frac{\partial J}{\partial \mathbf{U}^1}$ 와  $\frac{\partial J}{\partial \mathbf{U}^2}$ 를 계산한다.
7          식 (3.21)로  $\mathbf{U}^1$ 과  $\mathbf{U}^2$ 를 갱신한다.
8  until (멈춤 조건)
```

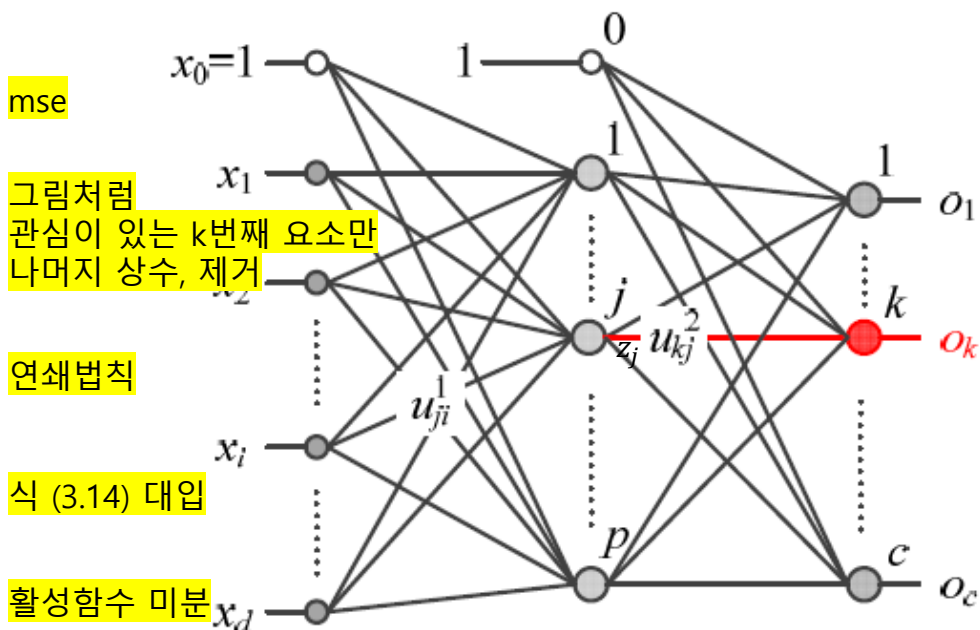


3.4.2 오류 역전파 알고리즘의 설계

■ 오류 역전파의 유도

- [알고리즘 3-3]의 라인 6을 위한 도함수 값 $\frac{\partial J}{\partial \mathbf{U}^1}$ 와 $\frac{\partial J}{\partial \mathbf{U}^2}$ 의 계산 과정
- 먼저 \mathbf{U}^2 를 구성하는 u_{kj}^2 로 미분하면,

$$\begin{aligned}
 \frac{\partial J}{\partial u_{kj}^2} &= \frac{\partial (0.5 \|\mathbf{y} - \mathbf{o}(\mathbf{U}^1, \mathbf{U}^2)\|_2^2)}{\partial u_{kj}^2} \\
 &= \frac{\partial \left(0.5 \sum_{q=1}^c (y_q - o_q)^2 \right)}{\partial u_{kj}^2} \\
 &= \frac{\partial (0.5 (y_k - o_k)^2)}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \frac{\partial o_k}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \frac{\partial \tau(\text{osum}_k)}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \tau'(\text{osum}_k) \frac{\partial \text{osum}_k}{\partial u_{kj}^2} \\
 &= -(y_k - o_k) \tau'(\text{osum}_k) z_j
 \end{aligned}$$



(a) u_{kj}^2 가 미치는 영향

osum 미분 그림 3-18 매개변수가 미치는 영향

$$o_k = \tau(\text{osum}_k), k = 1, 2, \dots, c$$

$$\text{이때 } \text{osum}_k = \mathbf{u}_k^2 \mathbf{z} \text{이고 } \mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \dots, u_{kp}^2), \mathbf{z} = (1, z_1, z_2, \dots, z_p)^T$$

3.4.2 오류 역전파 알고리즘의 설계

■ 오류 역전파의 유도

- \mathbf{U}^1 을 구성하는 u_{ji}^1 로 미분하면,

$$\begin{aligned}
 \frac{\partial J}{\partial u_{ji}^1} &= \frac{\partial (0.5 \|\mathbf{y} - \mathbf{o}(\mathbf{U}^1, \mathbf{U}^2)\|_2^2)}{\partial u_{ji}^1} \\
 &= \frac{\partial (0.5 \sum_{q=1}^c (y_q - o_q)^2)}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \frac{\partial o_q}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) \frac{\partial osum_q}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) \frac{\partial osum_q}{\partial z_j} \frac{\partial z_j}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2 \frac{\partial z_j}{\partial u_{ji}^1} \\
 &= - \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2 \tau'(zsum_j) x_i
 \end{aligned}$$

$$= -\tau'(zsum_j) x_i \sum_{q=1}^c (y_q - o_q) \tau'(osum_q) u_{qj}^2$$

$$o_k = \tau(osum_k), k = 1, 2, \dots, c \quad (3.14)$$

$$\text{이때 } osum_k = \mathbf{u}_k^2 \mathbf{z} \text{이고 } \mathbf{u}_k^2 = (u_{k0}^2, u_{k1}^2, \dots, u_{kp}^2), \mathbf{z} = (1, z_1, z_2, \dots, z_p)^T$$

$$z_j = \tau(zsum_j), j = 1, 2, \dots, p \quad (3.13)$$

$$\text{이때 } zsum_j = \mathbf{u}_j^1 \mathbf{x} \text{이고 } \mathbf{u}_j^1 = (u_{j0}^1, u_{j1}^1, \dots, u_{jd}^1), \mathbf{x} = (1, x_1, x_2, \dots, x_d)^T$$

연쇄법칙
연결된 것만 관심
나머진 상수, 제거

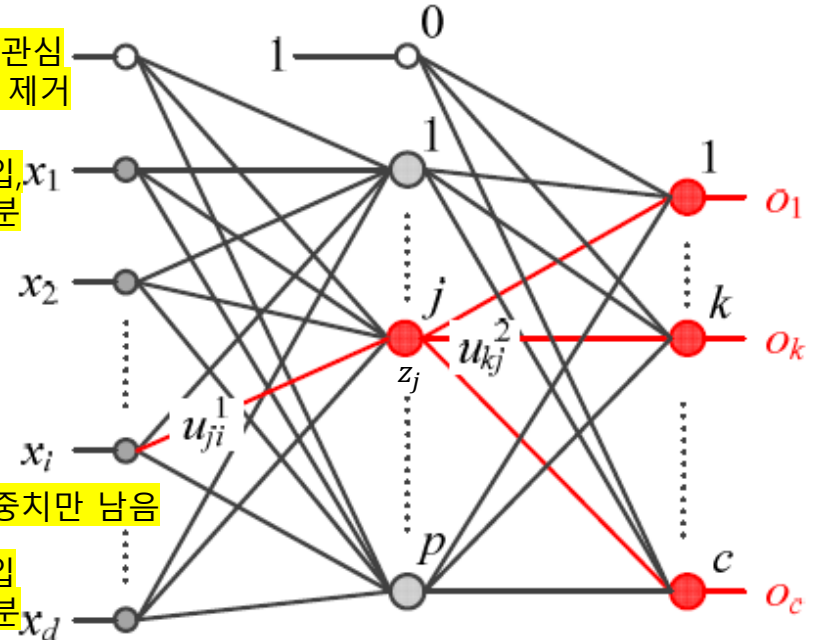
식 (3.14) 대입,
활성함수 미분

미분하면 가중치만 남음

식 (3.13) 대입
활성함수 미분
zsum 미분

(b) u_{ji}^1 이 미치는 영향

항 정리



3.4.2 오류 역전파 알고리즘의 설계

- 지금까지 유도한 식을 정리하면,

$$\delta_k = (y_k - o_k)\tau'(osum_k), \quad 1 \leq k \leq c \quad (3.22)$$

반복되는 부분식

$$\frac{\partial J}{\partial u_{kj}^2} = \Delta u_{kj}^2 = -\delta_k z_j, \quad 0 \leq j \leq p, 1 \leq k \leq c \quad (3.23)$$

$$\eta_j = \tau'(zsum_j) \sum_{q=1}^c \delta_q u_{qj}^2, \quad 1 \leq j \leq p \quad (3.24)$$

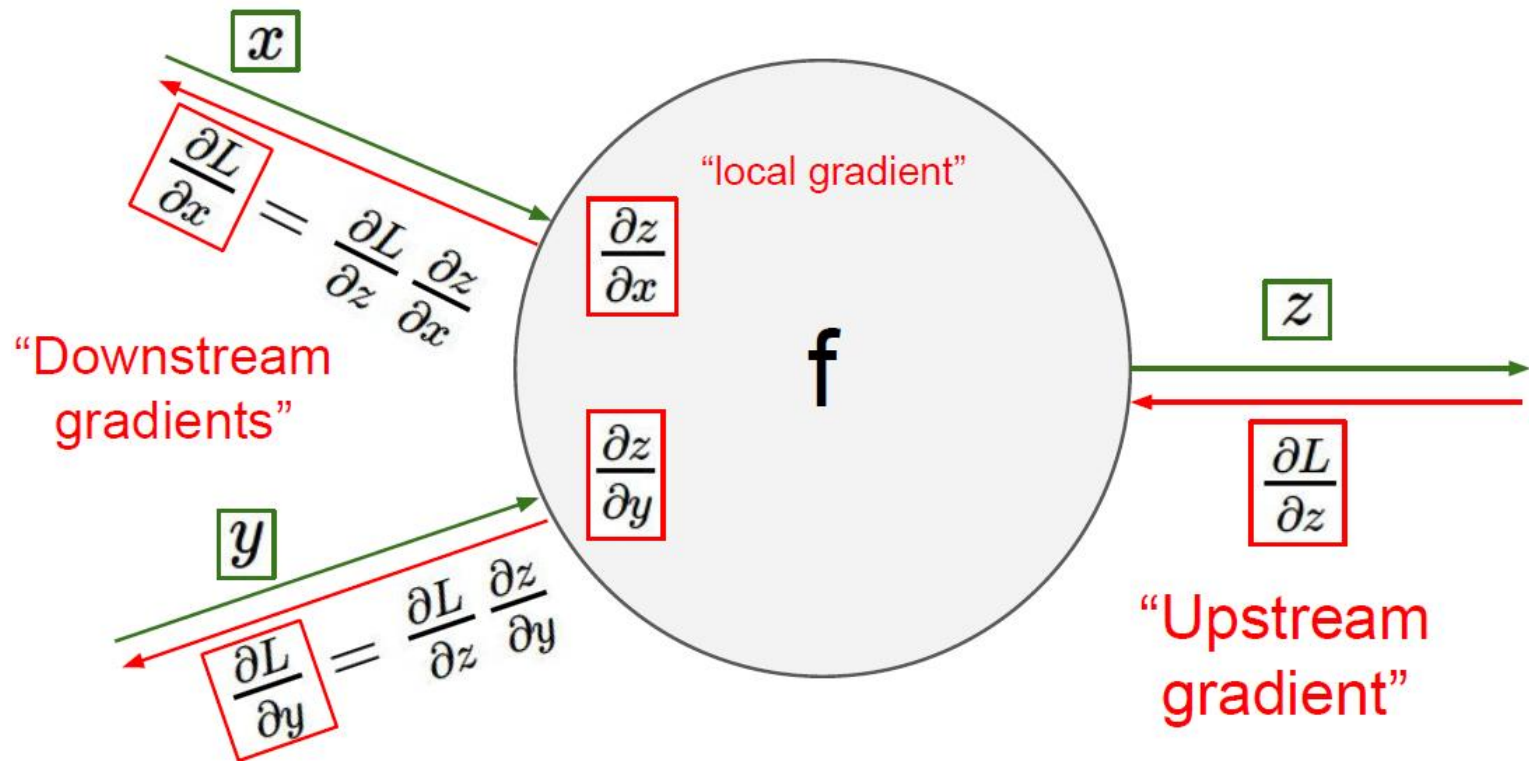
$$\frac{\partial J}{\partial u_{ji}^1} = \Delta u_{ji}^1 = -\eta_j x_i, \quad 0 \leq i \leq d, 1 \leq j \leq p \quad (3.25)$$

- 오류 역전파 알고리즘

- 식 (3.22)~(3.25)를 이용하여 출력층의 오류를 역방향(왼쪽)으로 전파하며 그레이디언트를 계산하는 알고리즘
- 반복되는 부분식들의 지수적 폭발(exponential explosion)을 피해야 함

3.4.2 오류 역전파 알고리즘의 설계

■ 역전파 분해

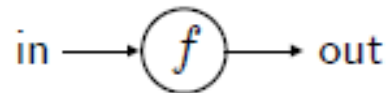


3.4.2 오류 역전파 알고리즘의 설계

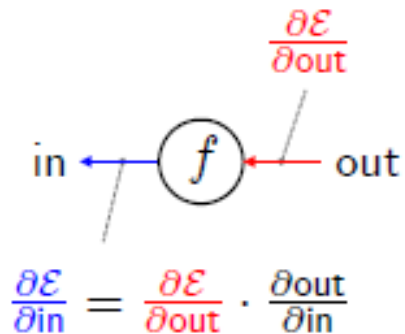
■ 단일 노드의 역전파 예

$$\text{out} = f(\text{in})$$

forward



backprop



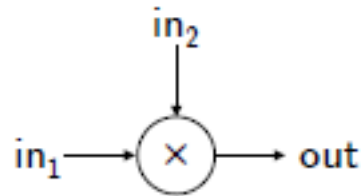
$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \text{in}} &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{\text{local gradient}} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot f'(\text{in}) \end{aligned}$$

3.4.2 오류 역전파 알고리즘의 설계

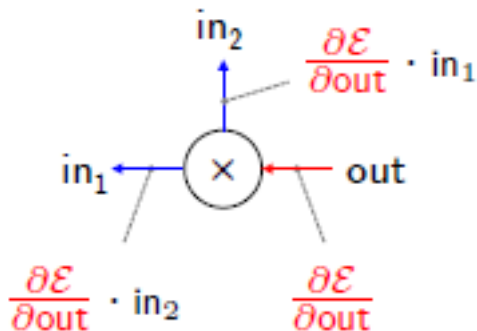
■ 곱셈의 역전파 예

$$\text{out} = \text{in}_1 \cdot \text{in}_2$$

forward



backprop

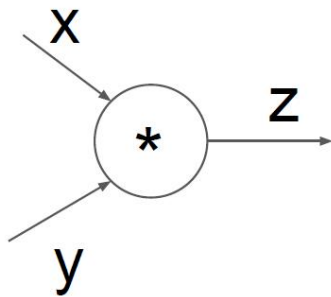


$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \text{in}_1} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_1} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\text{in}_2}_{\text{local gradient}}\end{aligned}$$

$$\begin{aligned}\frac{\partial \mathcal{E}}{\partial \text{in}_2} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_2} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \text{in}_1\end{aligned}$$

3.4.2 오류 역전파 알고리즘의 설계

■ 곱셈의 역전파 PyTorch 구현 예



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):  
    @staticmethod  
    def forward(ctx, x, y):  
        ctx.save_for_backward(x, y)  
        z = x * y  
        return z  
    @staticmethod  
    def backward(ctx, grad_z):  
        x, y = ctx.saved_tensors  
        grad_x = y * grad_z # dz/dx * dL/dz  
        grad_y = x * grad_z # dz/dy * dL/dz  
        return grad_x, grad_y
```

Need to stash
some values for
use in backward

Upstream
gradient

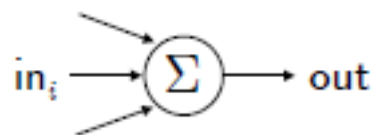
Multiply upstream
and local gradients

3.4.2 오류 역전파 알고리즘의 설계

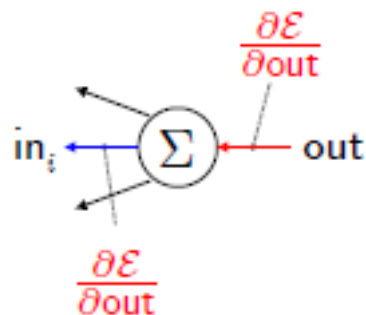
■ 덧셈의 역전파 예

$$\text{out} = \sum_i \text{in}_i$$

forward



backprop



$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \text{in}_i} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_i} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{1}_{\text{local gradient}} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \\ &\triangleq \delta \end{aligned}$$

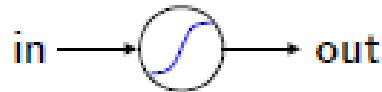
sum (forward) \Leftrightarrow fanout (backprop)

3.4.2 오류 역전파 알고리즘의 설계

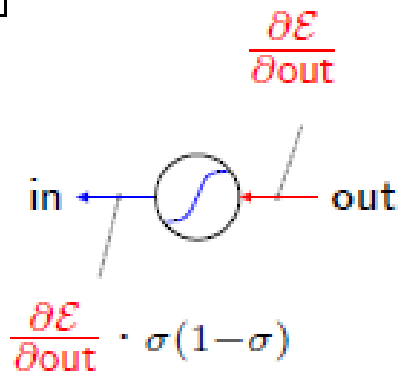
■ 시그모이드의 역전파 예

$$\text{out} = \sigma(\text{in})$$

forward



backprop



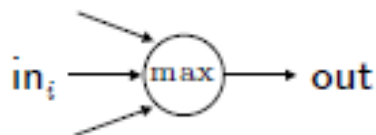
$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \text{in}} &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}} \\ &= \underbrace{\frac{\partial \mathcal{E}}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\sigma'(\text{in})}_{\text{local gradient}} \\ &= \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot [\sigma(\text{in}) (1 - \sigma(\text{in}))] \end{aligned}$$

3.4.2 오류 역전파 알고리즘의 설계

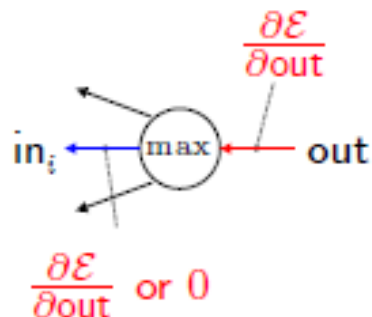
■ 최대화의 역전파 예

$$\text{out} = \max_i \{ \text{in}_i \}$$

forward



backprop



$$\frac{\partial \mathcal{E}}{\partial \text{in}_i} = \frac{\partial \mathcal{E}}{\partial \text{out}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{1 \text{ or } 0}$$

$$= \begin{cases} \frac{\partial \mathcal{E}}{\partial \text{out}} & \text{if } \text{in}_i \text{ is max} \\ 0 & \text{otherwise} \end{cases}$$

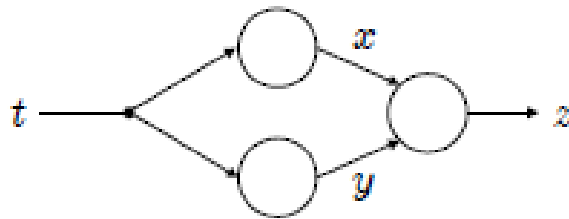
$\text{max (forward)} \Leftrightarrow \text{mux (backprop)}$

3.4.2 오류 역전파 알고리즘의 설계

■ 전개 fanout의 역전파 예

multivariable chain rule

forward

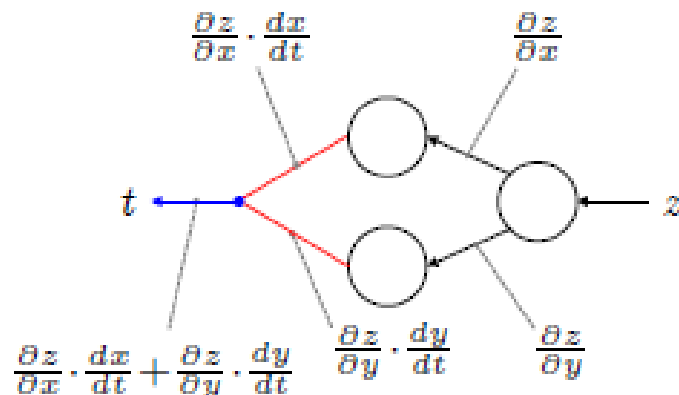


let

$$x = x(t), \quad y = y(t)$$

$$z = f(x, y)$$

backprop



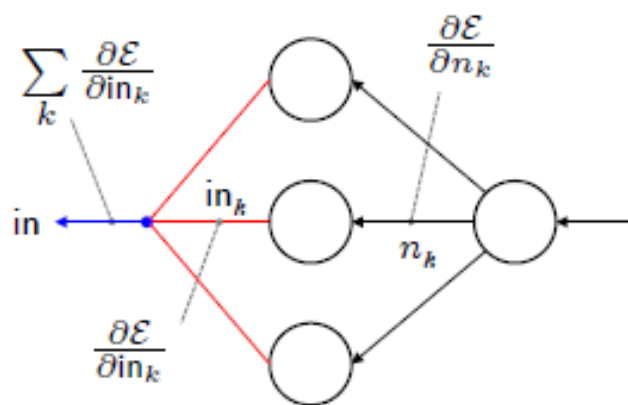
then

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial t} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial t}$$

3.4.2 오류 역전파 알고리즘의 설계

■ 전개의 역전파 예

fanout



assuming

$$\mathcal{E} = f(n_1, \dots, n_k, \dots)$$

and

$$n_k = n_k(\text{in})$$

gives

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial \text{in}} &= \sum_k \frac{\partial \mathcal{E}}{\partial n_k} \cdot \frac{\partial n_k}{\partial \text{in}} \\ &= \sum_k \frac{\partial \mathcal{E}}{\partial \text{in}_k} \end{aligned}$$

where

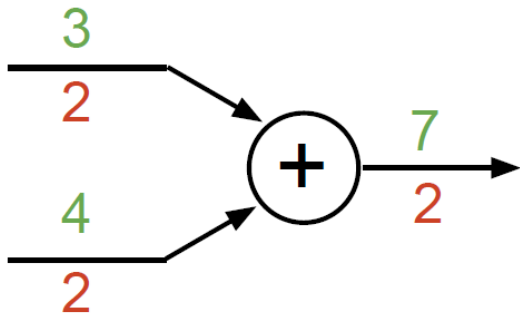
$$\text{in}_k \triangleq \text{input to } n_k$$

fanout (forward) \Leftrightarrow sum (backprop)

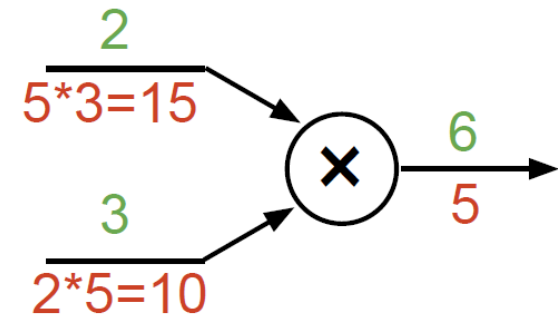
3.4.2 오류 역전파 알고리즘의 설계

■ 역전파 주요 예

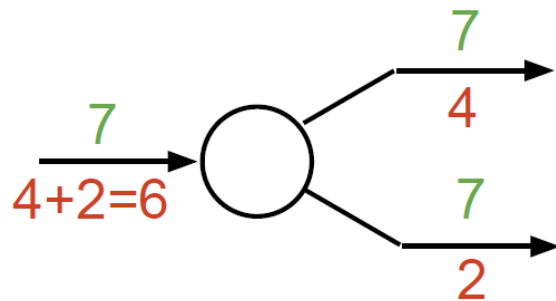
add gate: gradient distributor



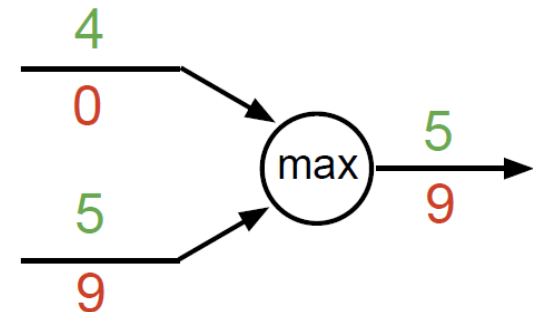
mul gate: “swap multiplier”



copy gate: gradient adder



max gate: gradient router

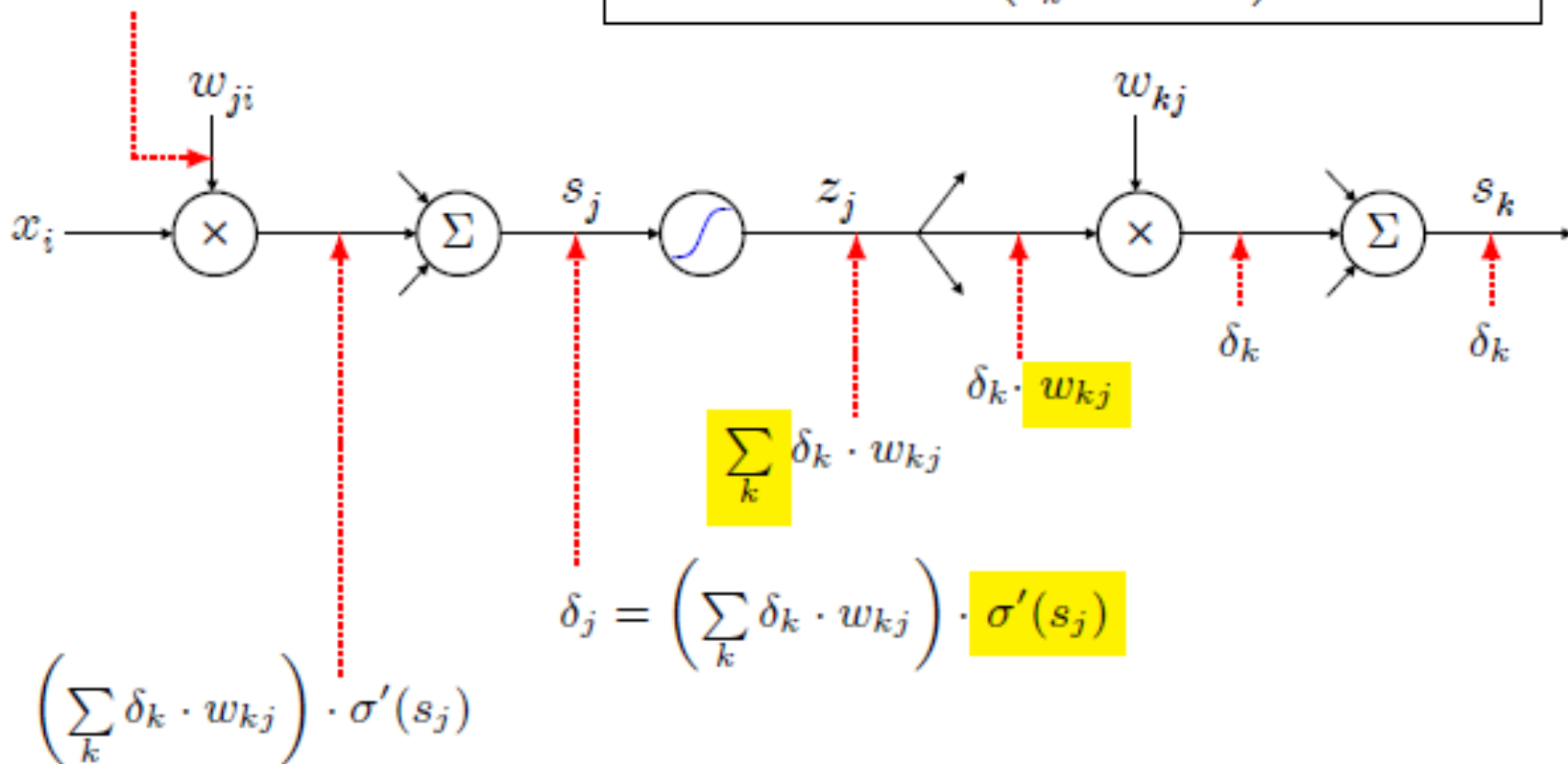


3.4.2 오류 역전파 알고리즘의 설계

■ 실제 역전파 예

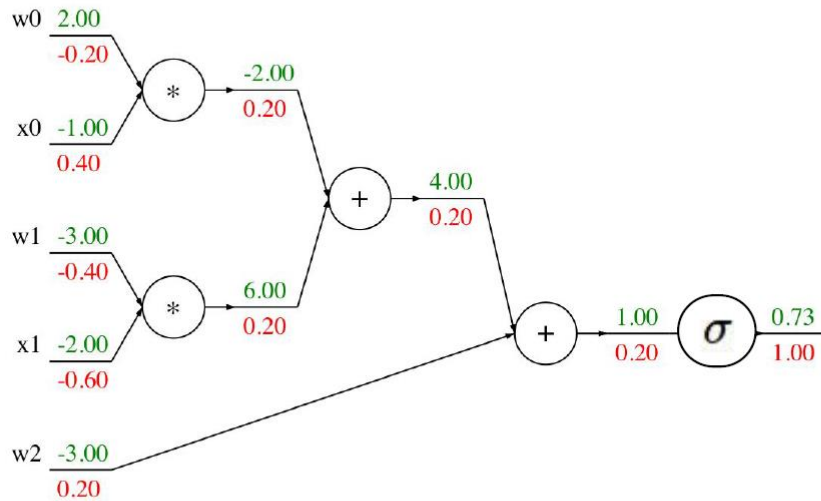
computing $\frac{\partial \mathcal{E}}{\partial w_{ji}}$

$$\left(\sum_k \delta_k \cdot w_{kj} \right) \cdot \sigma'(s_j) \cdot x_i \Rightarrow \boxed{\frac{\partial \mathcal{E}}{\partial w_{ji}} = \delta_j \cdot x_i = \left(\sum_k \delta_k \cdot w_{kj} \right) \cdot \sigma'(s_j) \cdot x_i}$$



3.4.2 오류 역전파 알고리즘의 설계

■ 역전파의 간단한 구현



Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
```

```
    s0 = w0 * x0  
    s1 = w1 * x1  
    s2 = s0 + s1  
    s3 = s2 + w2  
    L = sigmoid(s3)
```

Backward pass:
Compute grads

```
    grad_L = 1.0  
    grad_s3 = grad_L * (1 - L) * L  
    grad_w2 = grad_s3  
    grad_s2 = grad_s3  
    grad_s0 = grad_s2  
    grad_s1 = grad_s2  
    grad_w1 = grad_s1 * x1  
    grad_x1 = grad_s1 * w1  
    grad_w0 = grad_s0 * x0  
    grad_x0 = grad_s0 * w0
```

3.4.3 오류 역전파를 이용한 학습 알고리즘

■ 식 (3.22)~(3.25)를 이용한 스토캐스틱 경사 하강법

알고리즘 3-4 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법

입력: 훈련집합 \mathbb{X} 와 \mathbb{Y} , 학습률 ρ

출력: 가중치 행렬 \mathbf{U}^1 과 \mathbf{U}^2

```
1   $\mathbf{U}^1$ 과  $\mathbf{U}^2$ 를 초기화한다.
2  repeat
3     $\mathbb{X}$ 의 순서를 섞는다.
4    for ( $\mathbb{X}$ 의 샘플 각각에 대해)
5      현재 처리하는 샘플을  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$ ,  $\mathbf{y} = (y_1, y_2, \dots, y_c)^T$ 라 표기한다.
6       $x_0$ 과  $z_0$ 을 1로 설정한다. // 바이어스
          // 전방 계산
7      for ( $j=1$  to  $\rho$ )  $zsum_j = \mathbf{u}_j^1 \mathbf{x}$ ,  $z_j = \tau(zsum_j)$  // 식 (3.13)
8      for ( $k=1$  to  $c$ )  $osum_k = \mathbf{u}_k^2 \mathbf{z}$ ,  $o_k = \tau(osum_k)$  // 식 (3.14)
          // 오류 역전파
9      for ( $k=1$  to  $c$ )  $\delta_k = (y_k - o_k) \tau'(osum_k)$  // 식 (3.22)
10     for ( $k=1$  to  $c$ ) for ( $j=0$  to  $\rho$ )  $\Delta u_{kj}^2 = -\delta_k z_j$  // 식 (3.23)
11     for ( $j=1$  to  $\rho$ )  $\eta_j = \tau'(zsum_j) \sum_{q=1}^c \delta_q u_{qj}^2$  // 식 (3.24)
12     for ( $j=1$  to  $\rho$ ) for ( $i=0$  to  $d$ )  $\Delta u_{ji}^1 = -\eta_j x_i$  // 식 (3.25)
          // 가중치 갱신
13     for ( $k=1$  to  $c$ ) for ( $j=0$  to  $\rho$ )  $u_{kj}^2 = u_{kj}^2 - \rho \Delta u_{kj}^2$  // 식 (3.21)
14     for ( $j=1$  to  $\rho$ ) for ( $i=0$  to  $d$ )  $u_{ji}^1 = u_{ji}^1 - \rho \Delta u_{ji}^1$  // 식 (3.21)
15 until (멈춤 조건)
```

3.4.3 오류 역전파를 이용한 학습 알고리즘

■ 임의 샘플링 방식으로 바꾸려면,

- 3. \mathbb{X} 의 순서를 섞는다.
- 4. for (\mathbb{X} 의 샘플 각각에 대해)
- 5. 현재 처리하는 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$, $\mathbf{y} = (y_1, y_2, \dots, y_c)^T$ 라 표기한다.
- 6. x_0 과 z_0 을 1로 설정한다.

-
- 3. \mathbb{X} 에서 임의로 샘플 하나를 뽑는다.
 - 4. 뽑힌 샘플을 $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$, $\mathbf{y} = (y_1, y_2, \dots, y_c)^T$ 라 표기한다.
 - 5. x_0 과 z_0 을 1로 설정한다.

3.4.3 오류 역전파를 이용한 학습 알고리즘

■ 도함수 derivatives의 종류

Scalar to Scalar

$$x \in \mathbb{R}, y \in \mathbb{R}$$

Regular derivative:

$$\frac{\partial y}{\partial x} \in \mathbb{R}$$

If x changes by a small amount, how much will y change?

Vector to Scalar

$$x \in \mathbb{R}^N, y \in \mathbb{R}$$

Derivative is **Gradient**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^N \quad \left(\frac{\partial y}{\partial x} \right)_n = \frac{\partial y}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will y change?

Vector to Vector

$$x \in \mathbb{R}^N, y \in \mathbb{R}^M$$

Derivative is **Jacobian**:

$$\frac{\partial y}{\partial x} \in \mathbb{R}^{N \times M} \quad \left(\frac{\partial y}{\partial x} \right)_{n,m} = \frac{\partial y_m}{\partial x_n}$$

For each element of x, if it changes by a small amount then how much will each element of y change?

3.4.3 오류 역전파를 이용한 학습 알고리즘

■ 행렬 표기: GPU를 사용한 **고속 행렬 연산**에 적합

알고리즘 3-5 다층 퍼셉트론 학습을 위한 스토캐스틱 경사 하강법(행렬 표기)

입력: 훈련집합 \mathbb{X} 와 \mathbb{Y} , 학습률 ρ

출력: 가중치 행렬 \mathbf{U}^1 과 \mathbf{U}^2

```
1   $\mathbf{U}^1$ 과  $\mathbf{U}^2$ 를 초기화한다.
2  repeat
3     $\mathbb{X}$ 의 순서를 섞는다.
4    for ( $\mathbb{X}$ 의 샘플 각각에 대해)
5      현재 처리하는 샘플을  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$ ,  $\mathbf{y} = (y_1, y_2, \dots, y_c)^T$ 라 표기한다.
6       $x_0$ 과  $z_0$ 을 1로 설정한다. // 바이어스
          // 전방 계산
7       $\mathbf{zsum} = \mathbf{U}^1 \mathbf{x}$ ,  $\tilde{\mathbf{z}} = \tau(\mathbf{zsum})$  // 식 (3.13),  $\mathbf{zsum}_{p \times 1}$ ,  $\mathbf{U}^1_{p \times (d+1)}$ ,  $\mathbf{x}_{(d+1) \times 1}$ ,  $\tilde{\mathbf{z}}_{p \times 1}$ 
8       $\mathbf{osum} = \mathbf{U}^2 \tilde{\mathbf{z}}$ ,  $\mathbf{o} = \tau(\mathbf{osum})$  // 식 (3.14),  $\mathbf{osum}_{c \times 1}$ ,  $\mathbf{U}^2_{c \times (p+1)}$ ,  $\tilde{\mathbf{z}}_{(p+1) \times 1}$ ,  $\mathbf{o}_{c \times 1}$ 
          // 오류 역전파
9       $\delta = (\mathbf{y} - \mathbf{o}) \odot \tau'(\mathbf{osum})$  // 식 (3.22),  $\delta_{c \times 1}$ 
10      $\Delta \mathbf{U}^2 = -\delta \mathbf{z}^T$  // 식 (3.23),  $\Delta \mathbf{U}^2_{c \times (p+1)}$ 
11      $\boldsymbol{\eta} = (\delta^T \tilde{\mathbf{U}}^2)^T \odot \tau'(\mathbf{zsum})$  // 식 (3.24),  $\tilde{\mathbf{U}}^2_{c \times p}$ ,  $\boldsymbol{\eta}_{p \times 1}$ 
12      $\Delta \mathbf{U}^1 = -\boldsymbol{\eta} \mathbf{x}^T$  // 식 (3.25),  $\Delta \mathbf{U}^1_{p \times (d+1)}$ 
          // 가중치 갱신
13      $\mathbf{U}^2 = \mathbf{U}^2 - \rho \Delta \mathbf{U}^2$  // 식 (3.21)
14      $\mathbf{U}^1 = \mathbf{U}^1 - \rho \Delta \mathbf{U}^1$  // 식 (3.21)
15 until (멈춤 조건)
```

3.5 미니배치 스토캐스틱 경사 하강법

■ 미니배치 방식

- 한번에 t 개의 샘플을 처리함 (t 는 미니배치 크기)
 - $t=1$ 이면 스토캐스틱 경사 하강법 ([알고리즘 3-4])
 - $t=n$ (전체)이면 배치 경사 하강법
- 미니배치 방식은 보통 t =수십~수백
 - 그레이디언트의 잡음을 줄여주는 효과 때문에 수렴이 빨라짐
 - GPU를 사용한 병렬처리에도 유리함
- 현대 기계 학습은 미니배치를 표준처럼 여겨 널리 사용함

3.5 미니배치 스토캐스틱 경사 하강법

알고리즘 3-6 다층 퍼셉트론 학습을 위한 ‘미니배치’ 스토캐스틱 경사 하강법

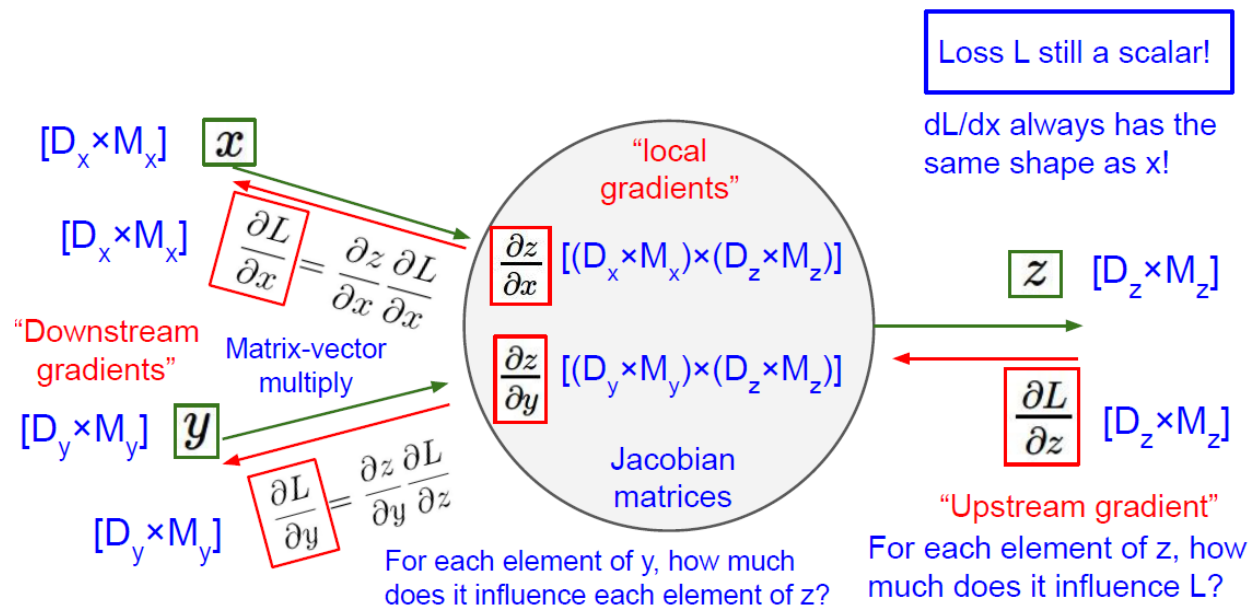
입력: 훈련집합 \mathbb{X} 와 \mathbb{Y} , 학습률 ρ , 미니배치 크기 t

출력: 가중치 행렬 \mathbf{U}^1 과 \mathbf{U}^2

```
1   $\mathbf{U}^1$ 과  $\mathbf{U}^2$ 를 초기화한다.
2  repeat
3       $\mathbb{X}$ 와  $\mathbb{Y}$ 에서  $t$ 개의 샘플을 무작위로 뽑아 미니배치  $\mathbb{X}'$ 와  $\mathbb{Y}'$ 를 만든다.
4       $\Delta \mathbf{U}^2 = \mathbf{0}$ ,  $\Delta \mathbf{U}^1 = \mathbf{0}$ 
5      for ( $\mathbb{X}'$ 의 샘플 각각에 대해)
6          현재 처리하는 샘플을  $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$ ,  $\mathbf{y} = (y_1, y_2, \dots, y_c)^T$ 라 표기한다.
7           $x_0$ 와  $z_0$ 를 1로 설정한다. // 바이어스
              // 전방 계산
8           $\mathbf{zsum} = \mathbf{U}^1 \mathbf{x}$ ,  $\tilde{\mathbf{z}} = \tau(\mathbf{zsum})$  // 식 (3.13),  $\mathbf{zsum}_{p \times 1}$ ,  $\mathbf{U}^1_{p \times (d+1)}$ ,  $\mathbf{x}_{(d+1) \times 1}$ ,  $\tilde{\mathbf{z}}_{p \times 1}$ 
9           $\mathbf{osum} = \mathbf{U}^2 \mathbf{z}$ ,  $\mathbf{o} = \tau(\mathbf{osum})$  // 식 (3.14),  $\mathbf{osum}_{c \times 1}$ ,  $\mathbf{U}^2_{c \times (p+1)}$ ,  $\mathbf{z}_{(p+1) \times 1}$ ,  $\mathbf{o}_{c \times 1}$ 
              // 오류 역전파
10          $\boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \odot \tau'(\mathbf{osum})$  // 식 (3.22),  $\boldsymbol{\delta}_{c \times 1}$ 
11          $\Delta \mathbf{U}^2 = \Delta \mathbf{U}^2 + (-\boldsymbol{\delta} \mathbf{z}^T)$  // 식 (3.23)을 누적,  $\Delta \mathbf{U}^2_{c \times (p+1)}$ 
12          $\boldsymbol{\eta} = (\boldsymbol{\delta}^T \tilde{\mathbf{U}}^2)^T \odot \tau'(\mathbf{zsum})$  // 식 (3.24),  $\tilde{\mathbf{U}}^2_{c \times p}$ ,  $\boldsymbol{\eta}_{p \times 1}$ 
13          $\Delta \mathbf{U}^1 = \Delta \mathbf{U}^1 + (-\boldsymbol{\eta} \mathbf{x}^T)$  // 식 (3.25)를 누적,  $\Delta \mathbf{U}^1_{p \times (d+1)}$ 
              // 가중치 갱신
14          $\mathbf{U}^2 = \mathbf{U}^2 - \rho \left(\frac{1}{t}\right) \Delta \mathbf{U}^2$  // 식 (3.21) - 평균 그래디언트로 갱신
15          $\mathbf{U}^1 = \mathbf{U}^1 - \rho \left(\frac{1}{t}\right) \Delta \mathbf{U}^1$  // 식 (3.21) - 평균 그래디언트로 갱신
16 until (멈춤 조건)
```

3.5 미니배치 스토캐스틱 경사 하강법

■ 행렬의 역전파를 위한 도함수



Backprop with Matrices

x: $[N \times D]$
 $\begin{bmatrix} 2 & 1 & -3 \\ -3 & 4 & 2 \end{bmatrix}$
 w: $[D \times M]$
 $\begin{bmatrix} 3 & 2 & 1 & -1 \\ 2 & 1 & 3 & 2 \\ 3 & 2 & 1 & -2 \end{bmatrix}$

Matrix Multiply

$$y_{n,m} = \sum_d x_{n,d} w_{d,m}$$

Jacobians:
 $dy/dx: [(N \times D) \times (N \times M)]$
 $dy/dw: [(D \times M) \times (N \times M)]$

y: $[N \times M]$
 $\begin{bmatrix} 13 & 9 & -2 & -6 \\ 5 & 2 & 17 & 1 \end{bmatrix}$
 $dL/dy: [N \times M]$
 $\begin{bmatrix} 2 & 3 & -3 & 9 \\ -8 & 1 & 4 & 6 \end{bmatrix}$

For a neural net we may have
 $N=64, D=M=4096$
 Each Jacobian takes 256 GB of memory!

3.6 다층 퍼셉트론에 의한 인식

■ 예측 (또는 테스트) 단계

- 학습을 마친 후 현장 설치하여 사용 (또는 테스트 집합으로 성능 테스트)

알고리즘 3-7 다층 퍼셉트론을 이용한 인식

입력: 테스트 샘플 \mathbf{x} // 신경망의 가중치 \mathbf{U}^1 과 \mathbf{U}^2 는 이미 설정되었다고 가정함.

출력: 부류 y

- 1 $\mathbf{x} = (x_0, x_1, x_2, \dots, x_d)^T$ 로 확장하고, x_0 과 z_0 을 1로 설정한다.
- 2 $\mathbf{zsum} = \mathbf{U}^1 \mathbf{x}$
- 3 $\tilde{\mathbf{z}} = \tau(\mathbf{zsum})$ // 식 (3.13)
- 4 $\mathbf{osum} = \mathbf{U}^2 \tilde{\mathbf{z}}$
- 5 $\mathbf{o} = \tau(\mathbf{osum})$ // 식 (3.14)
- 6 \mathbf{o} 에서 가장 큰 값을 가지는 노드에 해당하는 부류 번호를 y 에 대입한다.

- 라인 6을 수식으로 표현하면, $y = \underset{k}{\operatorname{argmax}} o_k$
- 전방 계산 한번만 사용하므로 빠름 (표 3-2 참조)

3.7 다층 퍼셉트론의 특성

- 3.7.1 오류 역전파 알고리즘의 빠른 속도
- 3.7.2 모든 함수를 정확하게 근사할 수 있는 능력
- 3.7.3 성능 향상을 위한 휴리스틱의 중요성

3.7.1 오류 역전파 알고리즘의 빠른 속도

■ 연산 횟수 비교

표 3-2 전방 계산과 오류 역전파 과정이 사용하는 연산 횟수

과정	수식	덧셈	곱셈
전방 계산	식 (3.13)	dp	dp
	식 (3.14)	pc	pc
오류 역전파	식 (3.22)	c	c
	식 (3.23)		cp
	식 (3.24)	cp	cp
	식 (3.25)		dp
	식 (3.21)	$cp+dp$	$cp+dp$

- 오류 역전파는 전방 계산보다 약 1.5~2배의 시간 소요 → 빠른 계산 가능
- 하지만 학습 알고리즘은 수렴할 때까지 **오류 역전파**를 **반복**해야 하므로
점근적 **시간 복잡도**는 $\theta((dp + pc)nq)$ (n 은 훈련집합의 크기, q 는 에포크^{epoch} 수)
 - 에포크는 전체 학습 집합을 수행한 단위

3.7.2 모든 함수를 정확하게 근사할 수 있는 능력

■ 호닉의 주장[Hornik1989]

- 은닉층을 하나만 가진 다층 퍼셉트론은 범용근사자universal approximator

“... standard multilayer feedforward network architectures using arbitrary squashing functions can approximate virtually any function of interest to any desired degree of accuracy, provided sufficiently many hidden units are available. ... 은닉 노드가 충분히 많다면, 포화함수(활성함수)로 무엇을 사용하든 표준 다층 퍼셉트론은 어떤 함수라도 원하는 정확도만큼 근사화할 수 있다.”

- 은닉 노드를 무수히 많이 할 수 없으므로,
실질적으로는 복잡한 구조의 데이터에서는 성능 한계

3.7.3 성능 향상을 위한 휴리스틱의 중요성

■ 순수한 최적화 알고리즘으로는 높은 성능 불가능

- 데이터 희소성, 잡음, 미숙한 신경망 구조 등의 이유
- **성능 향상**을 위한 갖가지 **휴리스틱** heuristics을 개발하고 공유함

→ 예) 『Neural Networks: Tricks of the Trade』 [Montavon2012]

■ 휴리스틱 개발에서 중요 쟁점

- **아키텍처**: 은닉층과 은닉 노드의 개수를 정해야 한다. 은닉층과 은닉 노드를 늘리면 신경망의 용량은 커지는 대신, 추정할 매개변수가 많아지고 학습 과정에서 과잉적합할 가능성이 커진다. 1.6절에서 소개한 바와 같이 현대 기계 학습은 복잡한 모델을 사용하되, 적절한 규제 기법을 적용하는 경향이 있다.
- **초깃값**: [알고리즘 3-4]의 라인 1에서 가중치를 초기화한다. 보통 난수를 생성하여 설정하는데, 값의 범위와 분포가 중요하다. 이 주제는 5.2.2절에서 다룬다.
- **학습률**: 처음부터 끝까지 같은 학습률을 사용하는 방식과 처음에는 큰 값으로 시작하고 점점 줄이는 적응적 방식이 있다. 5.2.4절에서 여러 가지 적응적 학습률 기법을 소개한다.
- **활성함수**: 초창기 다층 퍼셉트론은 주로 로지스틱 시그모이드나 tanh 함수를 사용했는데, 은닉층의 개수를 늘림에 따라 그레이디언트 소멸과 같은 몇 가지 문제가 발생한다. 따라서 깊은 신경망은 주로 ReLU 함수를 사용한다. 5.2.5절에서 여러 가지 ReLU 함수를 설명한다.

3.7.3 성능 향상을 위한 휴리스틱의 중요성

■ 실용적인 성능

- 1980~1990년대에 다층 퍼셉트론은 실용 시스템 제작에 크게 기여
 - 인쇄/필기 문자 인식으로 우편물 자동 분류기, 전표 인식기, 자동차 번호판 인식기 등
 - 음성 인식, 게임, 주가 예측, 정보 검색, 의료 진단, 유전자 검색, 반도체 결함 검사 등

■ 하지만 한계 노출

- 잡음이 섞인 상황에서 음성인식 성능 저하
- 필기 주소 인식 능력 저하
- 바둑 등의 복잡한 문제에서의 한계

■ 이러한 한계를 딥러닝은 극복함