한빛아카데미
Hanbit Academy, Inc.

MACHINE 기계 학습 LEARNING

오일석 지음

3장. 다층 퍼셉트론

PREVIEW

신경망

- 기계 학습 역사에서 가장 오래된 기계 학습 모델이며, 현재 가장 다양한 형태를 가짐
- 1950년대 퍼셉트론 → 1980년대 다층 퍼셉트론
- 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 인공신경망과 생물신경망
- 3.1.2 신경망의 간략한 역사
- 3.1.3 신경망의 종류

3.1.1 인공신경망과 생물신경망

- 사람의 뉴런
  - 두뇌의 가장 작은 정보처리 단위
  - 세포체는cell body 간단한 연산, 수상돌기는dendrite 신호 수신, 축삭은axon 처리 결과를 전송
  - 사람은 10^{11}개 정도의 뉴런을 가지며, 뉴런은 1000개 가량 다른 뉴런과 연결되어 있어 10^{14}개 정도의 연결

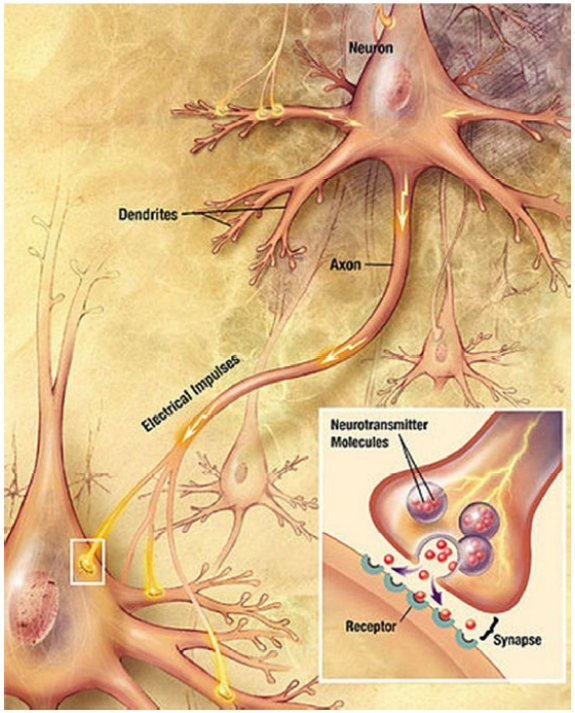
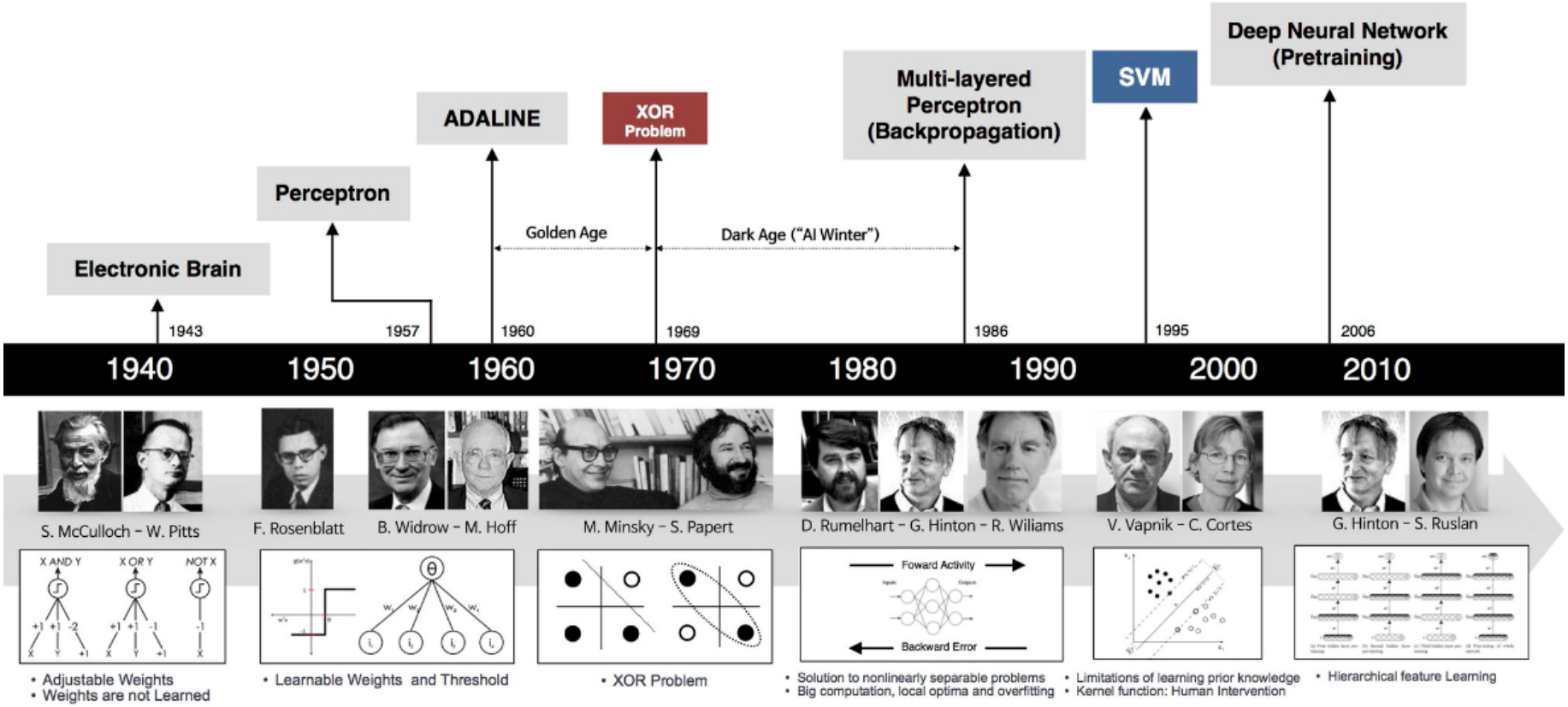


그림 3-1 사람의 뉴런의 구조와 동작

3.1.2 신경망의 간략한 역사

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

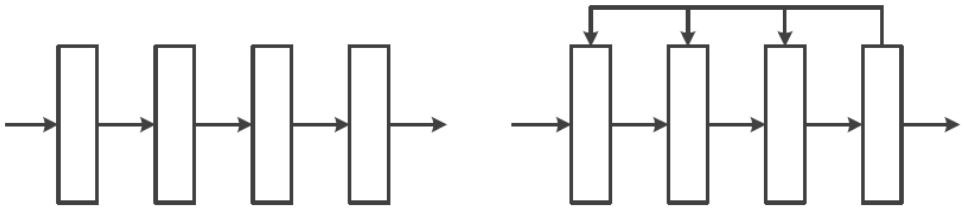
3.1.2 신경망의 간략한 역사



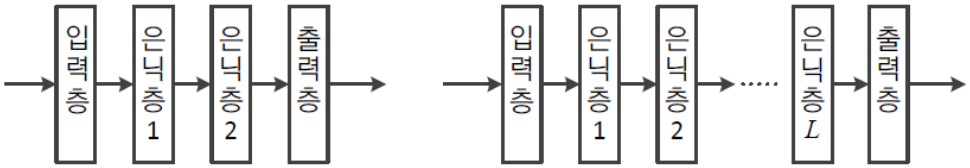
source: https://medium.com/ibm-data-science-experience/deep-learning-with-data-science-experience-8478cc0f81ac

3.1.3 신경망의 종류

- 신경망에는 아주 다양한 모델이 존재함
  - **전방(feedforward) 신경망과 순환(recurrent) 신경망:**
  - [그림3-2(a)]의 왼쪽 그림에서 보는 바와 같이 전방 신경망은 모든 계산이 왼쪽에서 오른쪽으로 진행된다. 반면, 오른쪽 그림의 순환 신경망은 오른쪽에서 왼쪽으로 진행되는 피드백 계산도 포함한다.
  - 3장에서 학습할 퍼셉트론과 다중 퍼셉트론, 4장에서 학습할 깊은 다중퍼셉트론(DMLP)과 컨볼루션 신경망(CNN)은 모두 전방 신경망에 속한다.
  - 8장에서 학습할 순환 신경망(RNN)과 LSTM은 순환 신경망에 속한다.



(a) 전방 신경망과 순환 신경망

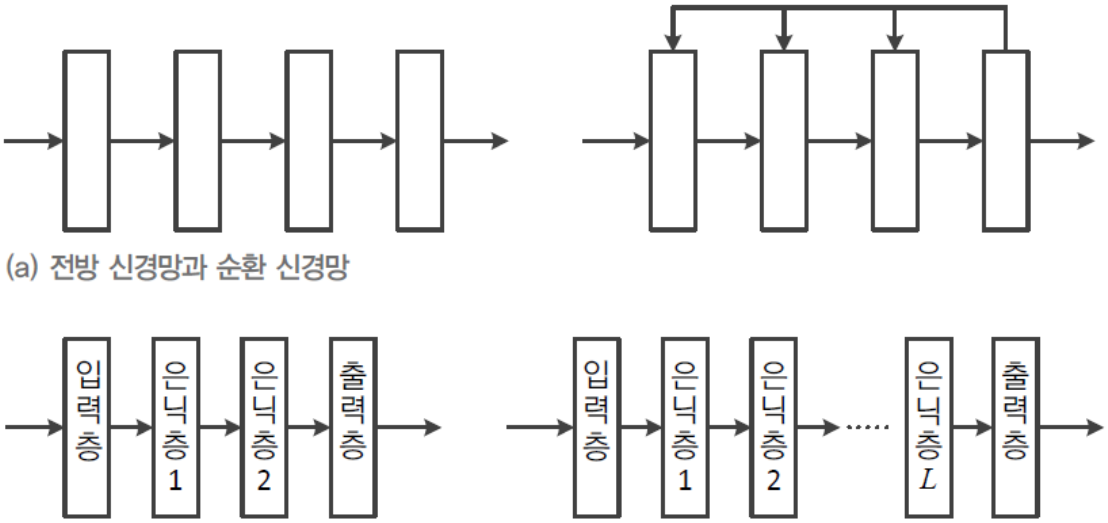


(b) 얕은 신경망과 깊은 신경망

그림 3-2 신경망의 종류

3.1.3 신경망의 종류

- 신경망에는 아주 다양한 모델이 존재함
  - **얕은(shallow) 신경망과 깊은(deep) 신경망:**
  - [그림 3-2(b)]의 왼쪽 그림처럼 은닉층이 1~2개정도인 신경망을 얕은 신경망이라고 하고,
  - 오른쪽 그림처럼 더 많은 신경망을 가진 신경망을 깊은 신경망이라고 한다.
  - 하지만 몇 개의 은닉층을 기준으로 구분하는 지는 명확하지 않다.
  - 3장에서는 얕은 신경망을 학습하고, 4장은 깊은 신경망을 학습한다.



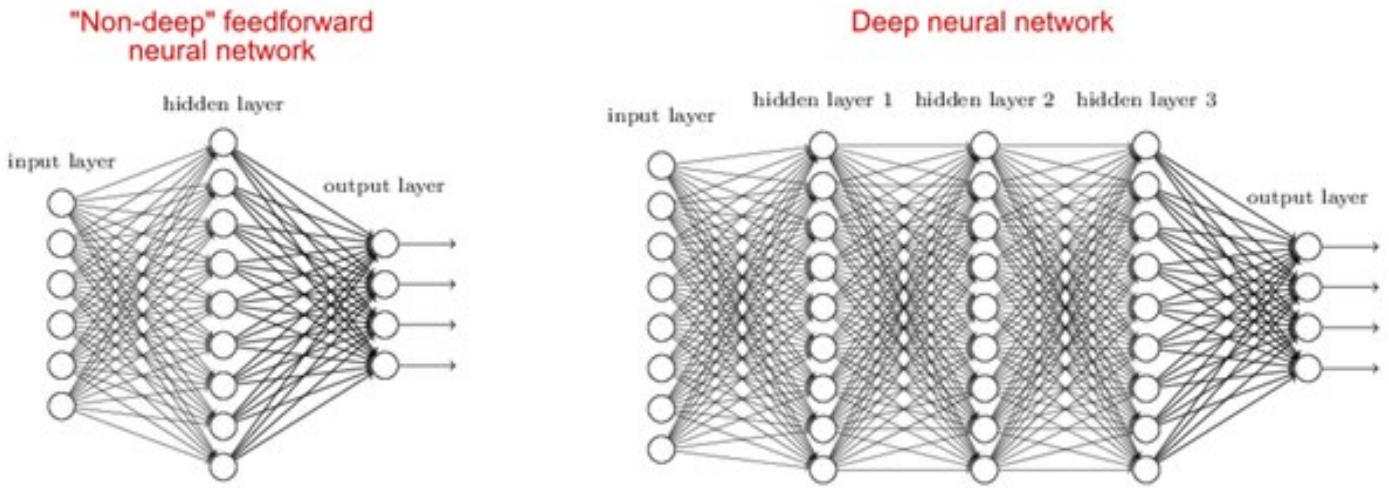
(a) 전방 신경망과 순환 신경망

(b) 얕은 신경망과 깊은 신경망

그림 3-2 신경망의 종류

3.1.3 신경망의 종류

- 신경망에는 아주 다양한 모델이 존재함
  - **결정론(deterministic) 신경망과 스토캐스틱(stochastic) 신경망:**
  - 결정론 신경망에서는 입력이 같으면 항상 같은 출력이 나온다. 계산식에서 임의성이 전혀 없기 때문이다.
  - 반면, 스토캐스틱 신경망에서는 계산식이 확률에 따른 난수를 사용하므로 입력이 같아도 매번 다른 출력이 나온다.
  - 10장에서 다룰 RBM(restricted Boltzmann machine)과 DBN(deep belief network)은 대표적인 스토캐스틱 신경망이며, 나머지는 결정론이다.
  - 결정론 신경망은 분류나 회귀와 같은 예측만 할 수 있지만, 스토캐스틱 신경망은 예측뿐 아니라 유사한 패턴을 생성하는 능력을 부여하면 생성모델로 활용 할 수 있다.



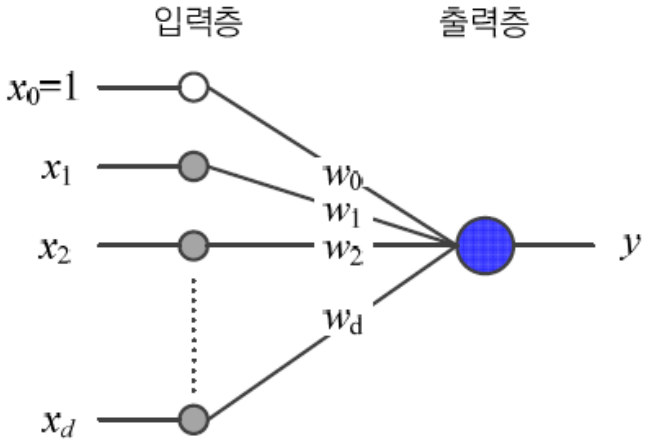
3.2 퍼셉트론

- 3.2.1 구조
- 3.2.2 동작
- 3.2.3 학습

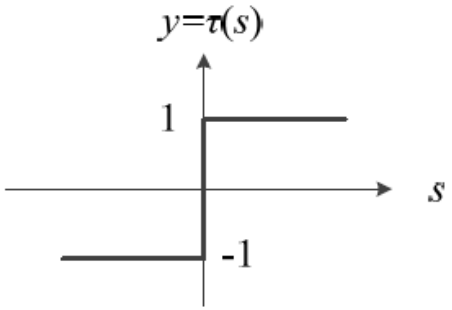
- 퍼셉트론은 노드, 가중치, 층과 같은 새로운 개념을 도입하고 학습 알고리즘을 창안함
- 퍼셉트론은 원시적 신경망이지만, 딥러닝을 포함한 현대 신경망은 퍼셉트론을 병렬과 순차 구조로 결합하여 만듬 → 현대 신경망의 중요한 구성요소

3.2.1 구조

- 퍼셉트론의 구조 [그림 3-3(a)]
  - 입력층과 출력층을 가짐
    - 입력층은 연산을 하지 않으므로 퍼셉트론은 단일 층 구조라고 간주 (입력층은 아무런 연산을 하지 않으므로 층의 개수를 셀 때 제외한다)
  - d 개의 노드를 가지는 입력층의 i번째 노드는 특징 벡터 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$의 요소 x_i를 담당
  - 항상 1이 입력되는 바이어스 노드(x_0=1). 총 d+1 개의 입력노드
  - 출력층은 한 개의 노드
  - i번째 입력층 노드와 출력층을 연결하는 에지는 가중치(weight) w_i를 가짐. 총 d+1 개의 가중치.



(a) 퍼셉트론의 구조



(b) 계단함수를 활성함수 $\tau(s)$로 이용함

그림 3-3 퍼셉트론의 구조와 동작

3.2.2 동작

퍼셉트론의 동작

- 입력층에 특징벡터 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$가 들어오면, 서로 연결된 특징값과 가중치를 곱한 결과를 모두 더하여 s를 구하고, 활성함수(activation function) τ를 적용함
- 활성함수 τ로 계단함수를 사용하므로 최종 출력 y는 +1 또는 -1 [그림 3-3(b)]

$$y = \tau(s)$$

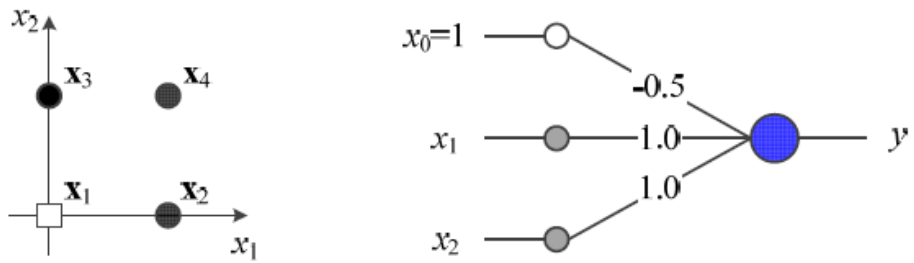
$$\text{이때 } s = w_0 + \sum_{i=1}^{d} w_i x_i, \qquad \tau(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases} \tag{3.1}$$

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.

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



(a) 훈련집합 (b) 퍼셉트론

그림 3-4 OR 논리 게이트를 이용한 퍼셉트론의 동작 예시

	x_1	x_2	y
$\mathbf{x}_1$	0	0	-1
$\mathbf{x}_2$	1	0	1
$\mathbf{x}_3$	0	1	1
$\mathbf{x}_4$	1	1	1

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

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

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

X 를 100% 정확하게 분류하는 퍼셉트론은 무수히 많다.
예를 들어, x_2 노드에 연결된 가중치를 0.9로 바꾸어도 성능은 같다.
즉, 훈련집합을 같은 성능으로 분류하는 퍼셉트론은 무수히 많다.

3.2.2 동작

- 행렬 표기

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

  - 바이어스 항을 벡터에 추가하면,

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

  - 퍼셉트론의 동작을 식 (3.4)로 표현할 수 있음

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

3.2.2 동작

- [그림 3-4(b)]를 기하학적으로 설명하면,
  - 결정 직선 $d(\mathbf{x}) = d(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0 = 0$ → $x_1 + x_2 - 0.5 = 0$
    - w_1과 w_2는 직선의 방향, w_0은 절편을 결정
    - 결정 직선은 전체 공간을 +1과 -1의 두 부분공간으로 분할하는 분류기(classifier) 역할

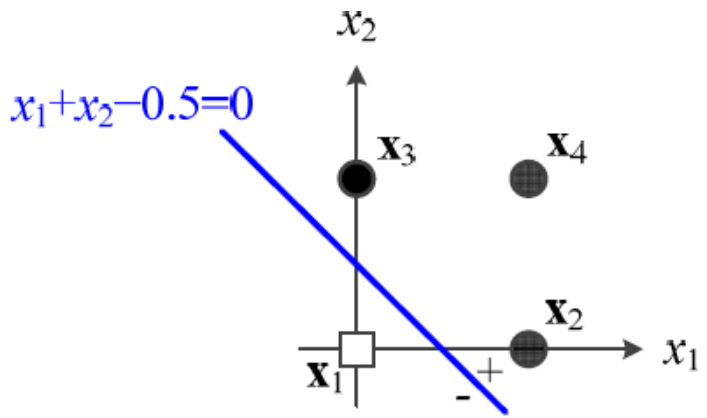


그림 3-5 [그림 3-4(b)]의 퍼셉트론에 해당하는 결정 직선

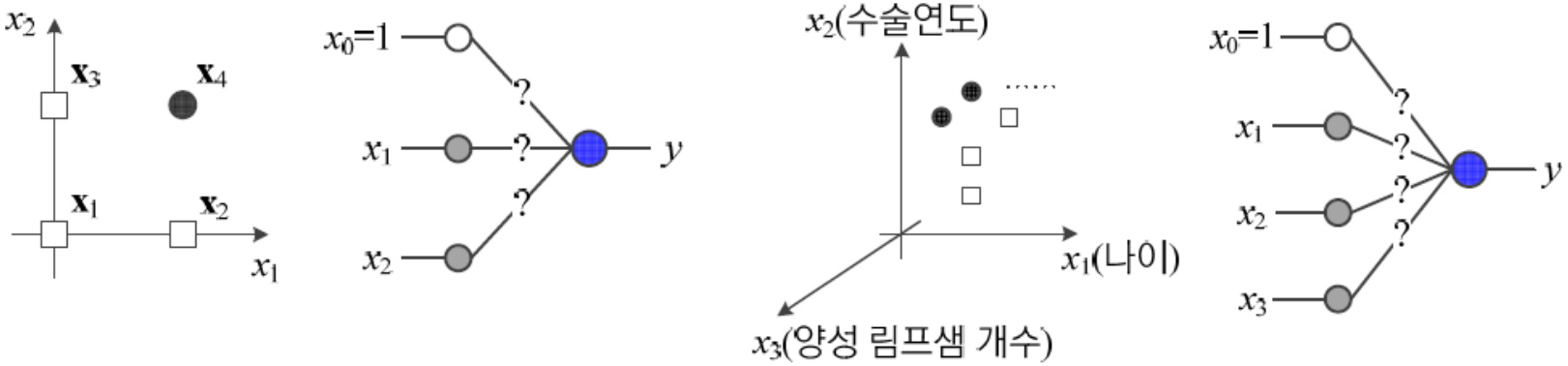
  - D 차원 공간에서는 $d(\mathbf{x}) = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + w_0 = 0$
    - w_1~wd는 초평면의 방향, w_0는 절편을 나타냄. w_0은 특징이 모두 0인 신호로 들어올 때 퍼셉트론이 출력할 기본값으로, 퍼셉트론이 0으로부터 벗어난 정도를 나타내는 바이어스(bias)라고 함.
    - 2차원은 결정 직선(line), 3차원은 결정 평면(plane), 4차원 이상은 결정 초평면(hyperplane)

3.2.3 학습

학습 문제:

- 지금까지는 퍼셉트론이 데이터를 제대로 분류하였는지 확인하는 것이었음. 즉, 학습을 마친 퍼셉트론을 가지고 동작을 설명한 셈
- 이제는 훈련집합으로 퍼셉트론을 어떻게 학습할 것인가? 학습문제.
- [그림 3-6(a)]는 [그림3-4(a)]의 OR 분류문제를 AND 분류문제로 바꾼 것임.
- 여기에서 학습 문제: w_1과 w_2, w_0이 어떤 값을 가져야 100% 옳게 분류할까?
- [그림 3-6(a)]는 2차원 공간에 4개 샘플이 있는 훈련집합이지만, 현실 세계는 d차원 공간에 수백~수만 개의 샘플이 존재 (예, MNIST는 784차원에 6만개 샘플)

	x_1	x_2	y
$\mathbf{x_1}$	0	0	-1
$\mathbf{x_2}$	1	0	-1
$\mathbf{x_3}$	0	1	-1
$\mathbf{x_4}$	1	1	1



(a) AND 분류 문제 (b) Haberman survival 분류 문제

그림 3-6 어떻게 학습시킬 것인가?

자동으로 최적 매개변수값을 찾아주는 학습 알고리즘을 알아보자.

3.2.3 학습

목적함수 설계

- 퍼셉트론의 매개변수를 $\mathbf{w} = (w_0, w_1, w_2, \cdots, w_d)^{\mathrm{T}}$라 표기하면, 매개변수 집합은 $\Theta = \{\mathbf{w}\}$
- 목적함수를 $J(\Theta)$ 또는 $J(\mathbf{w})$로 표기함. ($J(\mathbf{w})$는 일종의 에러 함수임. 작을수록 모형이 정확함)
- 바람직한 목적함수의 조건
  - $J(\mathbf{w}) \geq 0$이다.
  - $\mathbf{w}$가 최적이면, 즉 모든 샘플을 맞히면 $J(\mathbf{w}) = 0$이다.
  - 틀리는 샘플이 많은 $\mathbf{w}$일수록 $J(\mathbf{w})$는 큰 값을 가진다.
- 식 (3.7)은 세 가지 조건을 만족하므로, 퍼셉트론의 목적함수로 적합
  - Y는 $\mathbf{w}$가 틀리는 샘플의 집합 (뒤의 슬라이드 [알고리즘3-1] 참조)
  - 만약 어떤 샘플 $\mathbf{x}_k$가 주어진 $\mathbf{w}$에 의해 잘못 분류되었다면(즉, $\mathbf{x}_k$가 Y(*틀린샘플집단*)에 속한다면), 퍼셉트론이 계산한 값 $\mathbf{w}^{\mathrm{T}}\mathbf{x}_k$ 와 $\mathbf{x}_k$ 의 부류를 뜻하는 y_k의 부호가 달라야 한다. 따라서 $-y(\mathbf{w}^{\mathrm{T}}\mathbf{x}_k)$는 항상 양수이다. 틀린 샘플이 많을수록 $J(\boldsymbol{w})$는 큰 값을 가진다. 일반적으로 y_k 는 +1 또는 -1 로 한다.

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

3.2.3 학습

그레이디언트 계산

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

(여기에서 x_{ki} 는 $\mathbf{x}_k$ 의 i 번째 요소이다. 즉, $\mathbf{x}_k = (x_{k0}, x_{k1}, \cdots, x_{ki}, \cdots, x_{kd})^{\mathrm{T}}$ 이다.

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} \frac{\partial(-y_k(w_0 x_{k0} + w_1 x_{k1} + \cdots + w_i x_{ki} + \cdots + w_d x_{kd}))}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} -y_k x_{ki}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = \sum_{\mathbf{x}_k \in Y} -y_k x_{ki}, \qquad i = 0,1,\cdots,d \tag{3.8}$$

- 편미분 결과인 식 (3.8)을 식 (2.58)에 대입하면,

(2.3.3절에서 학습했듯이 $\frac{\partial J(\mathbf{w})}{\partial w_i}$ 는 목적함수의 값이 커지는 방향이므로 –를 붙여 더한다).

$$\text{델타 규칙:} \quad w_i = w_i + \rho \sum_{\mathbf{x}_k \in Y} y_k x_{ki}, \qquad i = 0,1,\cdots,d \tag{3.9}$$

3.2.3 학습

- 퍼셉트론 학습 알고리즘 (틀린 샘플을 모두 모아 한꺼번에 매개변수를 갱신하는 배치버전)
  - 식 (3.9)를 이용하여 학습 알고리즘을 쓰면, (참조:[알고리즘2-4])
    - 훈련집합의 샘플을 모두 맞출(즉 $Y = \emptyset$) 때까지 세대epoch(라인 3~9)를 반복함. 즉 반복할 때마다 훈련집합의 모든 샘플을 한 번씩 방문하여 처리한다.

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 w를 설정한다.
2  repeat
3     Y = ∅    // 틀린 샘플 집합
4     for j=1 to n
5        y = τ(w^T x_j)                  // 식 (3.4)
6        if(y ≠ y_j) Y = Y ∪ x_j         // 틀린 샘플을 집합에 추가한다.
7     if(Y ≠ ∅)
8        for i=0 to d                    // 식 (3.9)
9           w_i = w_i + ρ Σ_{x_k∈Y} y_k x_ki
10 until (Y = ∅)
11 ŵ = w
```

3.2.3 학습

- 퍼셉트론 학습 알고리즘의 스토캐스틱 버전(참조:[알고리즘2-5])
  - 샘플 순서를 섞음(스토캐스틱). 틀린 샘플이 발생하면 즉시 갱신
  - 현대기계학습에서는 배치버전보다 스토캐스틱 버전 채택

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 w을 설정한다.
2  repeat
3     𝕏의 샘플 순서를 섞는다.
4     quit=true
5     for j=1 to n
6        y = τ(w^T x_j)      // 식 (3.4)
7        if(y ≠ y_j)
8           quit=false
9           for i=0 to d
10             w_i = w_i + ρ y_j x_ji
11 until(quit)     // 틀린 샘플이 없을 때까지
12 ŵ = w
```

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

3.3 다층 퍼셉트론

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

3.3 다층 퍼셉트론

퍼셉트론은 선형 분류기라는 한계

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

XOR (exclusive OR): 배타적 논리합(입력된 2개 중 1개만 참일 때 참이 되는 논리 연산자)

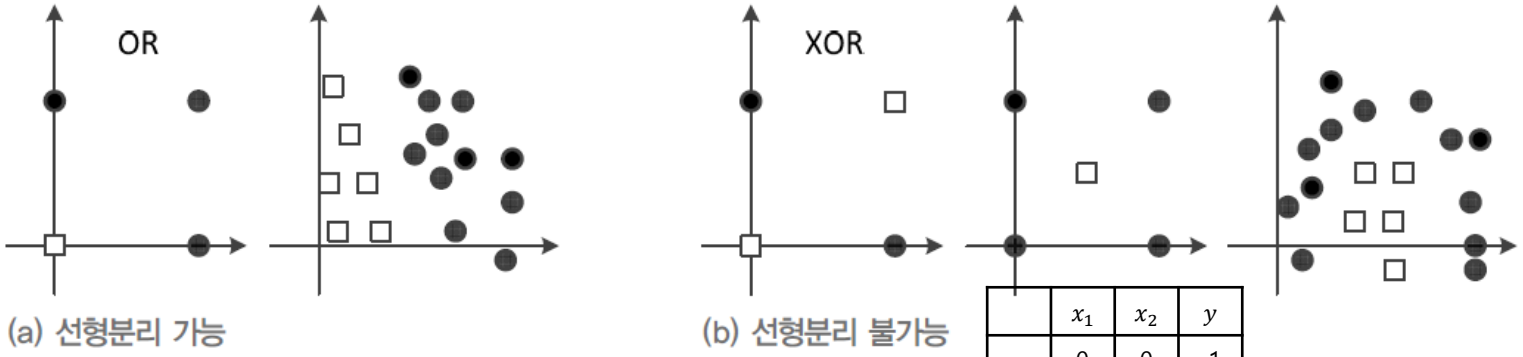


그림 3-7 선형분리가 가능한 상황과 불가능한 상황

	x_1	x_2	y
$\mathbf{x_1}$	0	0	-1
$\mathbf{x_2}$	1	0	1
$\mathbf{x_3}$	0	1	1
$\mathbf{x_4}$	1	1	-1

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

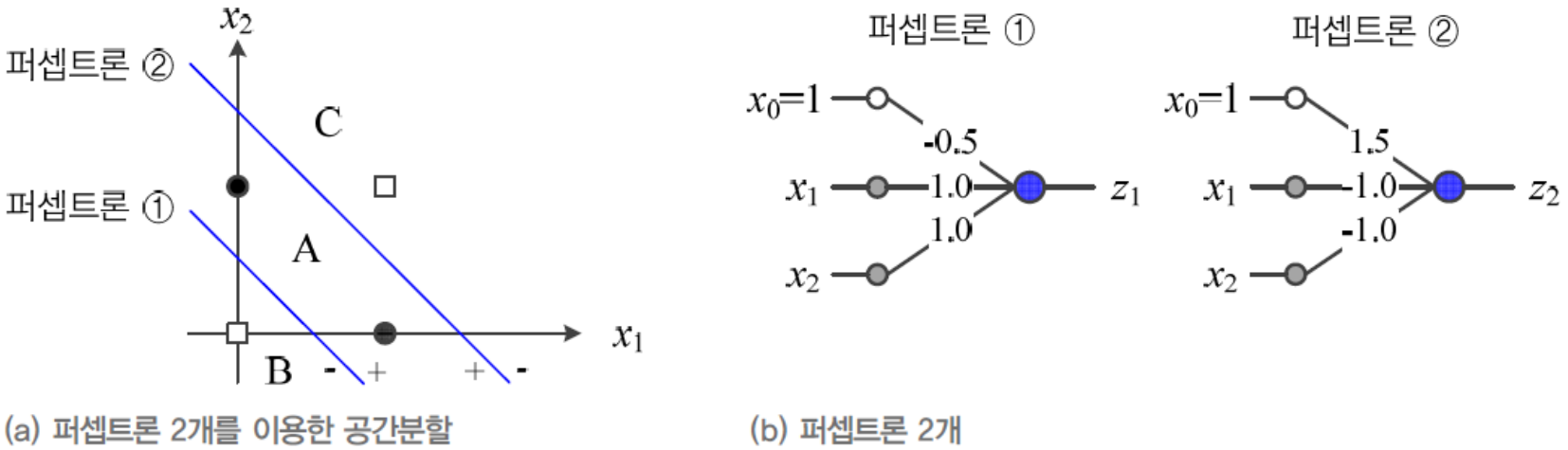
3.3 다층 퍼셉트론

다층 퍼셉트론(여러 개의 퍼셉트론을 결합한 다층구조를 이용하여 선형분리가 불가능한 상황 해결) 의 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 활성함수를 도입한다. 퍼셉트론은 [그림 3-3(b)]의 계단함수를 활성함수로 사용하였다. 이 함수는 경성hard 의사결정에 해당한다. 반면, 다층 퍼셉트론은 연성soft 의사결정이 가능한 [그림 3-12]의 시그모이드함수를 활성함수로 사용한다. 연성에서는 출력이 연속값인데, 출력을 신뢰도로 간주함으로써 더 융통성 있게 의사결정을 할 수 있다. 3.3.2절에서 시그모이드 활성함수를 자세히 설명한다.
- 오류 역전파 알고리즘을 사용한다. 다층 퍼셉트론은 여러 층이 순차적으로 이어진 구조이므로, 역방향으로 진행하면서 한 번에 한 층씩 그레이디언트를 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

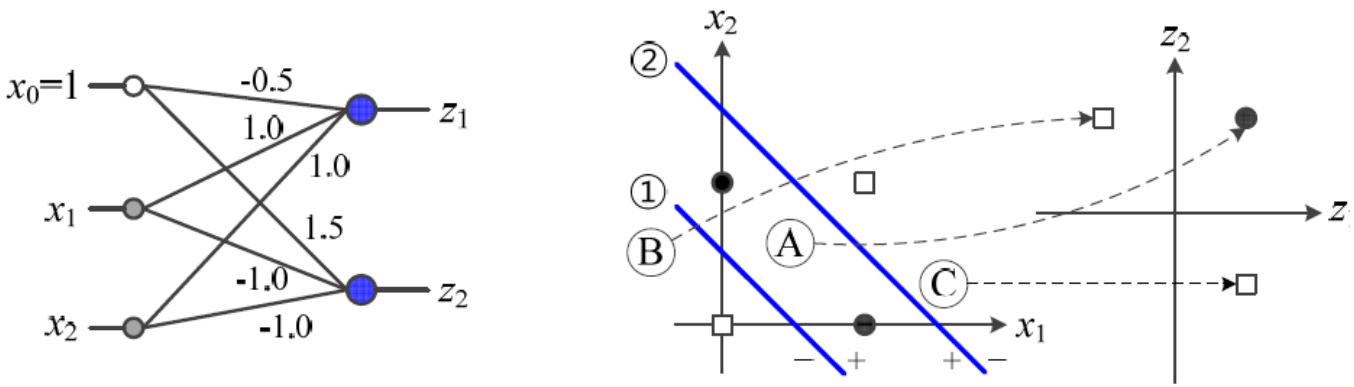


(a) 퍼셉트론 2개를 이용한 공간분할 (b) 퍼셉트론 2개

그림 3-8 XOR 문제의 해결

3.3.1 특징 공간 변환

- 퍼셉트론 2개를 병렬로 결합하면,
  - 원래 공간 $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$를 새로운 특징 공간 $\mathbf{z} = (z_1, z_2)^{\mathrm{T}}$로 변환
  - 새로운 특징 공간 $\mathbf{z}$에서는 선형 분리 가능함



(a) 두 퍼셉트론을 병렬로 결합 (b) 원래 특징 공간 x를 새로운 특징 공간 z로 변환

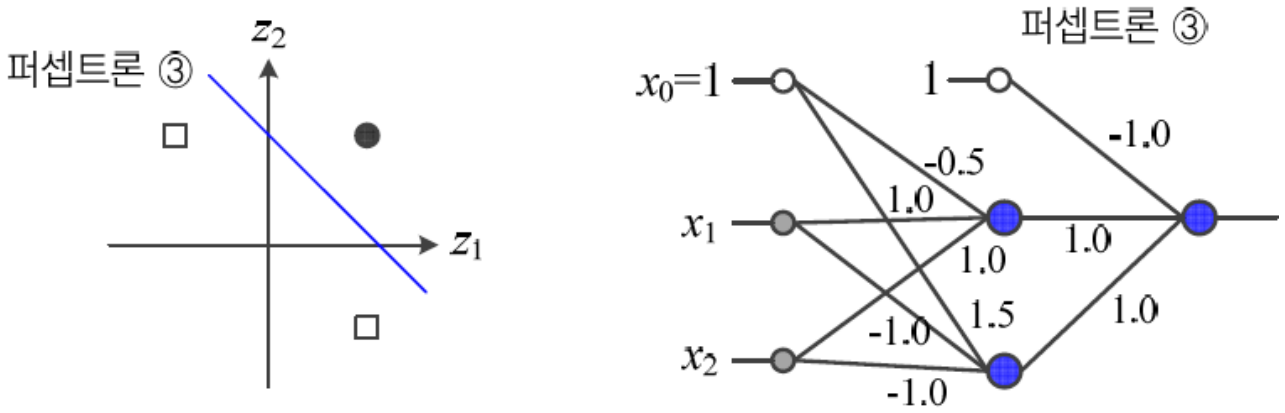
그림 3-9 특징 공간의 변환

  - 사람이 수작업으로 특징 학습을 수행한 셈

	x_1	x_2	z_1	z_2	y
$\mathbf{x_1}$	0	0	-1	1	-1
$\mathbf{x_2}$	1	0	1	1	1
$\mathbf{x_3}$	0	1	1	1	1
$\mathbf{x_4}$	1	1	1	-1	-1

3.3.1 특징 공간 변환

- 퍼셉트론 1개를 순차 결합하면,
  - 새로운 특징 공간 $\mathbf{z}$에서 선형 분리를 수행하는 퍼셉트론③을 순차 결합하면, [그림 3-10(b)]의 다층 퍼셉트론이 됨.
  - 즉, [그림 3-9(a)]의 출력을 ③의 입력에 연결하면 [그림 3-10(b)]의 다층 퍼셉트론이 됨.



(a) 새로운 특징 공간에서 분할 (b) 퍼셉트론 3개를 결합한 다층 퍼셉트론

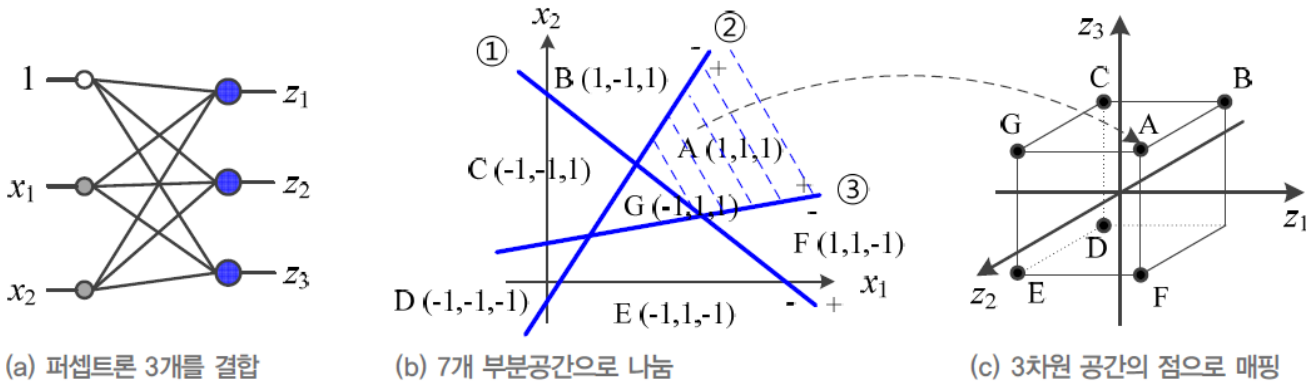
그림 3-10 다층 퍼셉트론

- 이 다층 퍼셉트론은 훈련집합에 있는 4개 샘플 $\binom{0}{0}\binom{0}{1}\binom{1}{0}\binom{1}{1}$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
- 활성함수 τ로 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, p개 퍼셉트론을 결합하면 p차원 공간으로 변환
  - $1+\sum_{i=1}^{p} i$개의 영역으로 분할

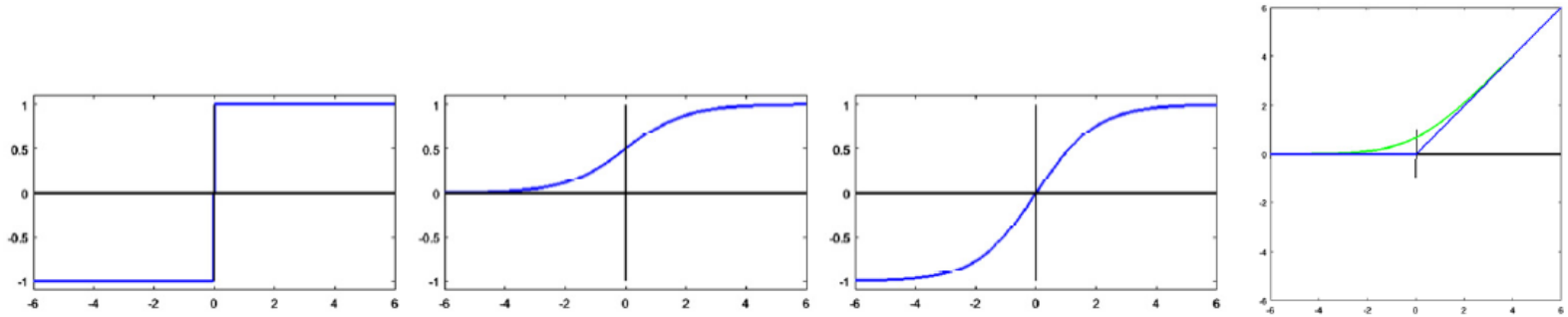
3.3.2 활성함수

- 딱딱한 공간 분할과 부드러운 공간 분할
  - 앞의 퍼셉트론이 사용하는 계단형 활성함수 [그림 3-3(b)]는 +1 또는 -1만 출력한다.
  - 따라서 [그림 3-9(b)]의 영역 A에 있는 점은 모두 (1,1)이라는 점으로 매핑된다.

    이는 경성(딱딱한, hard) 의사결정이라고 할 수 있다.
  - 경계선에 가까울수록 0에 가깝고 멀어질수록 +1 또는 -1에 가까워지는 연성(부드러운, soft) 의사결정으로 확장 고려 => 활성함수의 다양화

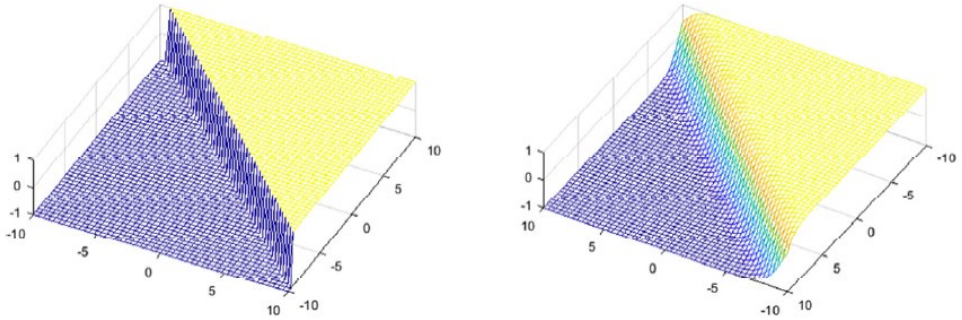
3.3.2 활성함수

- 딱딱한 공간 분할과 부드러운 공간 분할
  - 계단함수는 딱딱한 의사결정(영역을 점으로 변환). 나머지 활성함수는 부드러운 의사결정(영역을 영역으로 변환)



(a) 계단 함수 (b) 로지스틱 시그모이드 (c) 하이퍼볼릭 탄젠트 시그모이드 (d) softplus와 rectifier

그림 3-12 신경망이 사용하는 활성함수



(a) 계단함수의 딱딱한 공간 분할 (b) 로지스틱 시그모이드의 부드러운 공간 분할

그림 3-13 퍼셉트론의 공간 분할 유형

3.3.2 활성함수

교재 p.151 -3:
... 퍼셉트론은 (-1, 1) 사이의..
=> ... 퍼셉트론은 (0, 1) 사이의..

신경망이 사용하는 다양한 활성함수

- 로지스틱 시그모이드와 하이퍼볼릭 탄젠트(모든 구간에서 미분가능)는 a가 커질수록 계단함수에 가까워짐
- 모두 1차 도함수 계산이 빠름. 예를 들어, 로지스틱 시그모이드의 1차 도함수는 $\tau'(s) = a\tau(s)(1-\tau(s))$로서 함수 자체를 포함한다. 따라서 함수값을 저장하면 한번의 뺄셈과 한 번의 곱셈으로 도함수 계산을 마칠 수 있다.
- ReLU는 더 빠르다. 함수값은 한 번의 비교 연산만으로 구할 수 있고, 1차 도함수는 비교 연산 1개로 구할 수 있다.

표 3-1 활성함수로 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)	$\tau(s) = \max(0, s)$	$\tau'(s) = \begin{cases} 0 & s < 0 \\ 1 & s > 0 \\ \text{불가} & s = 0 \end{cases}$	$[0, \infty)$

3.3.2 활성함수

신경망이 사용하는 다양한 활성함수

- 퍼셉트론은 계단함수,
- 다층 퍼셉트론은 로지스틱 시그모이드와 하이퍼볼릭 탄젠트,
- 딥러닝은 ReLU를 사용

- 딥러닝이 로지스틱이나 하이퍼볼릭 탄젠트를 사용하면, 학습 도중에 그레이디언트가 점점 작아져 결국 0에 가까워지는 그레이디언트 소멸(gradient vanishing) 문제가 발생한다. 이 때 ReLU로 대치하면 문제가 크게 완화된다. 또한 ReLU는 비교 연산 한 번으로 계산할 수 있어 딥러닝의 속도 향상에 무척 유용하다.
- 하지만 음수를 모두 0으로 대치하기 때문에 문제가 발생하는데, 이 문제를 해결하기 위한 여러 변종(leaky ReLU, PReLU 등)이 개발 되어 있다. 5.2.5절 참조.

- 용어 참고:

하이퍼 매개변수: 신경망을 설계할 때 사용자가 지정하는 매개변수를 뜻한다.

예를 들어, 은닉 노드의 개수(p), 활성함수의 종류, 학습률 등이 여기에 속한다.

3.3.2 활성함수

The derivative of $\pi(x) = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}} = \frac{1}{1+e^{-(\alpha+\beta x)}}$ with respect to x is

$$\frac{d}{dx}\frac{1}{1+e^{-(\alpha+\beta x)}} = \beta e^{-(\alpha+\beta x)}\left[\frac{1}{1+e^{-(\alpha+\beta x)}}\right]^2$$

$$= \beta e^{-(\alpha+\beta x)}\left[\frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}\right]^2 = \beta\left[\frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}\right]\left[\frac{1}{1+e^{\alpha+\beta x}}\right]$$

$$= \beta\pi(x)[1-\pi(x)]$$

because $1-\pi(x) = 1/[1+e^{\alpha+\beta x}]$

3.3.3 구조

- 퍼셉트론과 달리 층이 여러 개이기 때문에 다층퍼셉트론(multi-layer perceptron, MLP)이라고 함
- 은닉층이 대략 4층 이상이면 깊은 신경망이라 하며, 깊은 신경망을 학습시키는 알고리즘을 딥러닝이라고 함
- [그림 3-14(a)]는 입력층-은닉층-출력층의 2층 구조
  - $d+1$개의 입력 노드 (d는 특징의 개수). c개의 출력 노드 (c는 부류 개수)
  - p개의 은닉 노드 : p는 하이퍼 매개변수(사용자가 정해주는 매개변수)
    - p가 너무 크면 과잉적합, 너무 작으면 과소적합 → 5.5절의 하이퍼 매개변수 최적화

  - 용어 참고:

    하이퍼 매개변수: 신경망을 설계할 때 사용자가 지정하는 매개변수를 뜻한다.

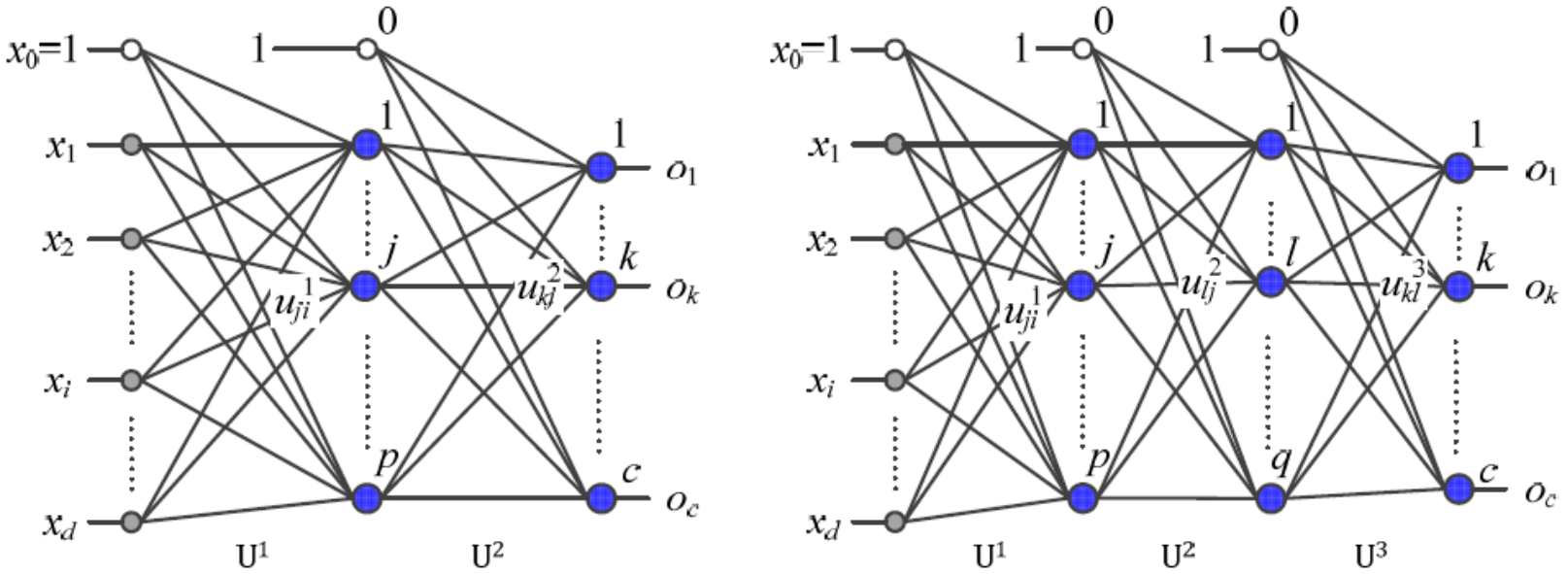
    예를 들어, 은닉 노드의 개수(p), 활성함수의 종류, 학습률 등이 여기에 속한다.

※ 교재에서 노드의 개수를 나타내는 기호:

특징(x)의 개수 : d 은닉노드(z)의 개수: p 또는 q 출력노드(부류)(o)의 개수: c

3.3.3 구조

- [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 퍼셉트론의 매개변수(가중치)
  - 입력층-은닉층을 연결하는 $\mathbf{U}^1$ (u_{ji}^1은 입력층의 i번째 노드를 은닉층의 j번째 노드와 연결)
  - 은닉층-출력층을 연결하는 $\mathbf{U}^2$ (u_{kj}^2는 은닉층의 j번째 노드를 출력층의 k번째 노드와 연결)

2층 퍼셉트론의 가중치 행렬:

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

  - 일반화하면 u_{ji}^l은 l-1번째 은닉층의 i번째 노드를 l번째 은닉층의 j번째 노드와 연결하는 가중치
    - 입력층을 0번째 은닉층 , 출력층을 마지막 은닉층으로 간주

※ 교재에 따라서는 매개변수(가중치) 기호가 $\mathbf{U}^1$, $\mathbf{U}^2$ 가 아니라 $\mathbf{W}^1$, $\mathbf{W}^2$ 로 사용하기도 함.

3.3.4 동작

- 특징 벡터 $\mathbf{x}$를 출력 벡터 $\mathbf{o}$로 매핑하는 함수로 간주할 수 있음

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

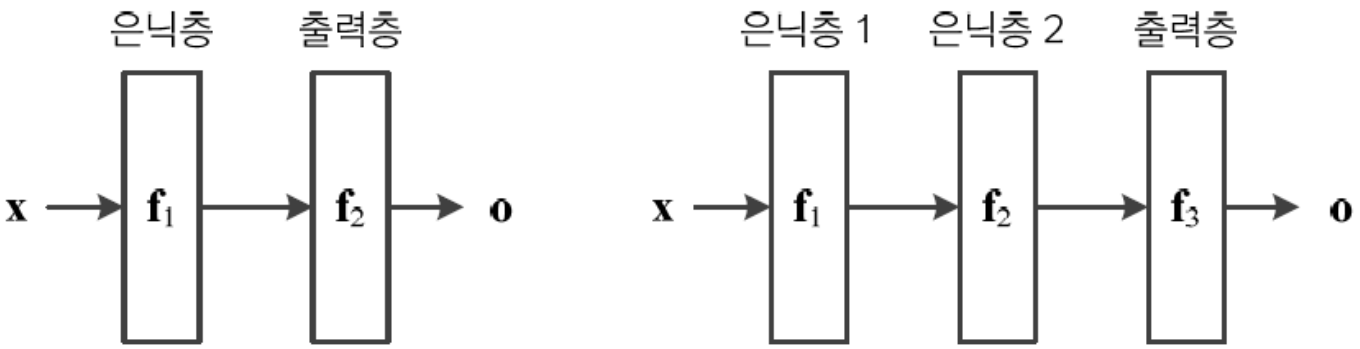


그림 3-15 다층 퍼셉트론을 간략화한 구조

- 깊은 신경망은 $\mathbf{o} = \mathbf{f}_L\left(\cdots \mathbf{f}_2\left(\mathbf{f}_1(\mathbf{x})\right)\right), L \geq 4$ ← 4장의 주제(딥러닝)

참고: 특징벡터 $\mathbf{x} = (x_1, x_2, \cdots, x_d)^{\mathrm{T}}$ 출력벡터 $\mathbf{o} = (o_1, o_2, \cdots, o_c)^{\mathrm{T}}$

3.3.4 동작

- 노드가 수행하는 연산을 구체적으로 쓰면,

j번째 은닉 노드의 연산:

$$z_j = \tau(zsum_j),\ j = 1,2,\cdots,p \qquad (3.13)$$

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

k번째 출력 노드의 연산:

$$o_k = \tau(osum_k),\ k = 1,2,\cdots,c \qquad (3.14)$$

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

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

※ $zsum_j$ 는 j번째 은닉 노드(z_j)로 가기 위해서 그 전의 노드들의 값을 sum 한 것임. sum_j 는 다른 교재에서는 net_j 로도 표시함.

- 다층 퍼셉트론의 동작을 행렬로 표기하면,

$$\mathbf{o} = \boldsymbol{\tau}(\mathbf{U}^2\boldsymbol{\tau}_h(\mathbf{U}^1\mathbf{x})) \qquad (3.15)$$

3.3.4 동작

- 은닉층은 특징 추출기
  - 은닉층은 특징 벡터를 분류에 더 유리한 새로운 특징 공간으로 변환하는 특징 추출기(feature extractor)
  - 현대 기계 학습에서는 특징 학습이라feature learning 부름 (딥러닝은 더 많은 단계를 거쳐 특징학습을 함)
  - 딥러닝에서는 계층적인 특징을 추출하는 방법을 사용하는데, "계층적"은 앞쪽에 있는 은닉층들이 추출한 저급특징(low-level feature)을 뒤쪽에 있는 은닉층들이 입력받아 추상화된 고급특징(high-level feature)을 추출하는 방식이다.

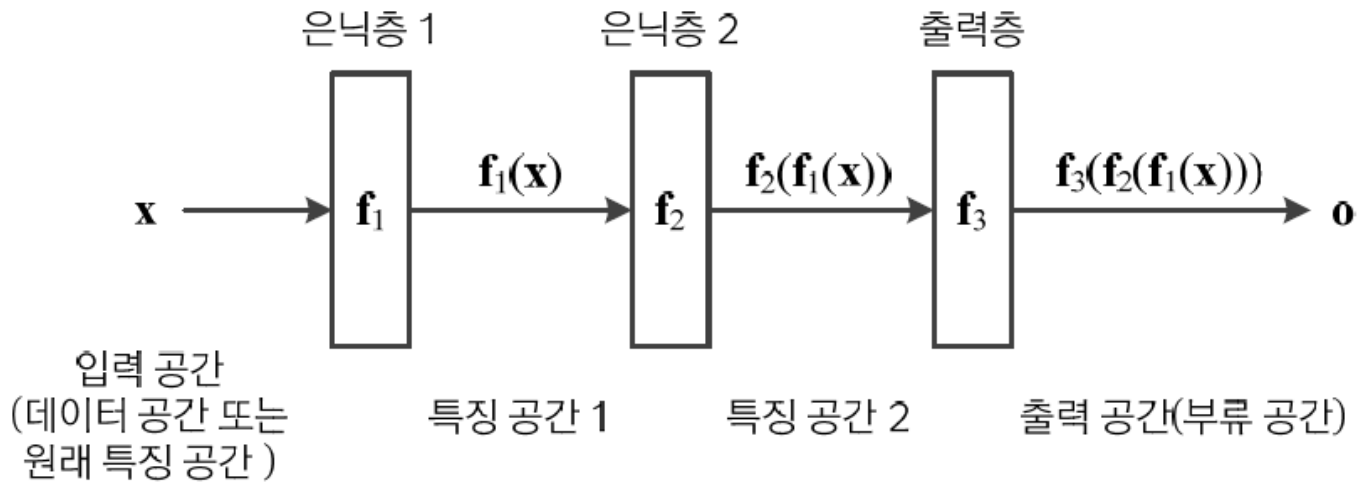
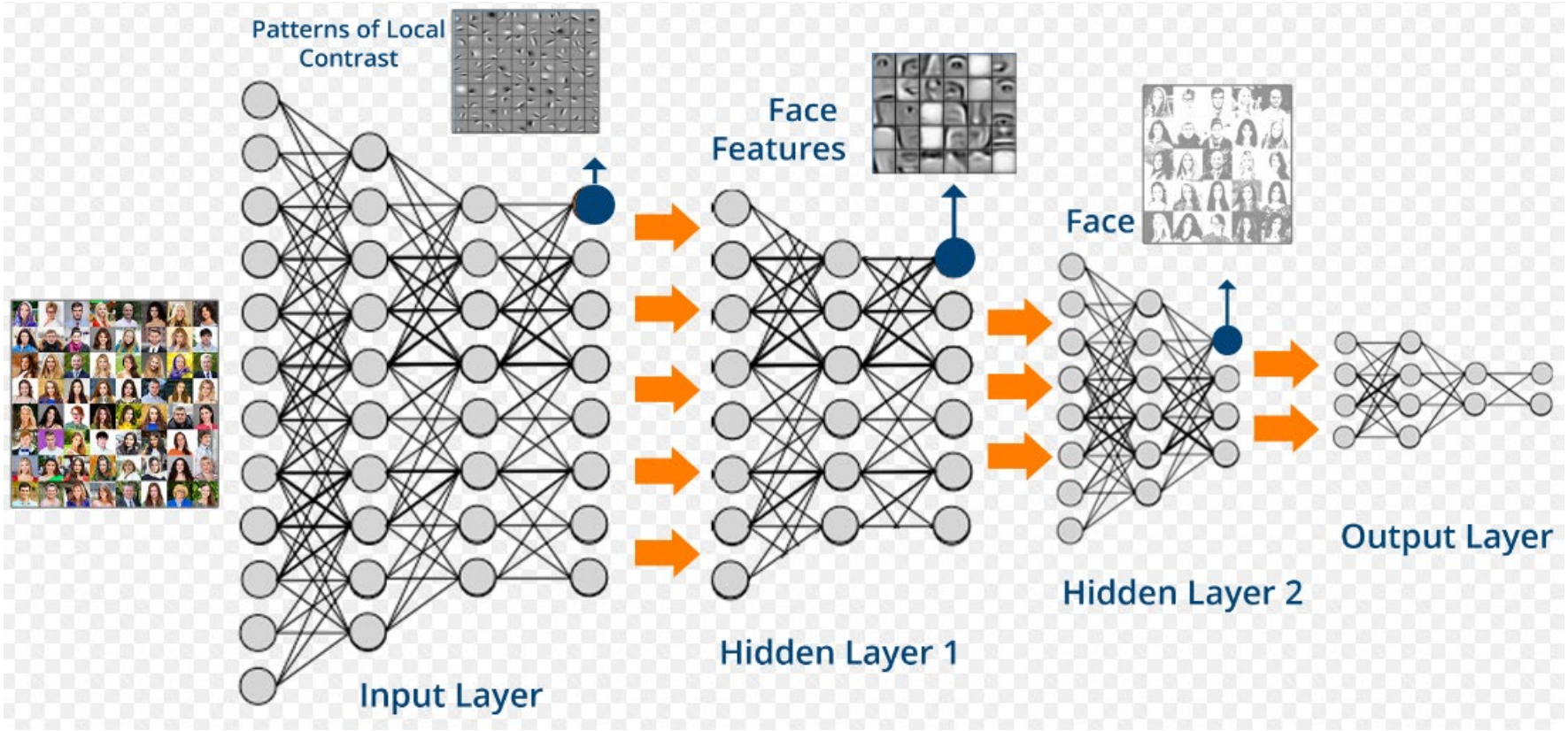


그림 3-16 특징 추출기로서의 은닉층

3.3.4 동작



source: https://cdn.edureka.co/blog/wp-content/uploads/2017/05/Deep-Neural-Network-What-is-Deep-Learning-Edureka.png

3.4 오류 역전파 알고리즘 (error backpropagation, Rumelhart 1986)

- 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 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$과 부류 벡터 집합 $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$
  - 부류 벡터는 원핫 코드로 표현됨. 즉 $\mathbf{y}_i = (0,0,\cdots,1,\cdots,0)^{\mathrm{T}}$ 만일 하이퍼볼릭탄젠트와 같이 범위가 (-1,1)인 활성함수를 사용한다면 0을 -1로 바꾸면 된다.
  - 설계 행렬로 쓰면,

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

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

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

훈련집합 기호

데이터 $\mathbf{X}$	$\mathbf{X}$ $(x_1, x_2, \ldots, x_j, \ldots, x_d)$	$\mathbf{Y}$ $(y_1, y_2, \ldots, y_k, \ldots, y_c)$	데이터의 y 부류값
$\mathbf{x}_1^T$	$\mathbf{x}_1^T = (x_{11}, x_{12}, \ldots, x_{1j}, \ldots x_{1d})$	$\boldsymbol{y}_1^T = (y_{11}, y_{12}, \ldots, y_{1k}, \ldots y_{1c})$ $= (0, 1, \ldots, 0, \ldots, 0)$	$y_1 = 2$
$\mathbf{x}_2^T$	$\mathbf{x}_2^T = (x_{21}, x_{22}, \ldots, x_{2j}, \ldots x_{2d})$	$\boldsymbol{y}_2^T = (y_{21}, y_{22}, \ldots, y_{2k}, \ldots y_{2c})$ $= (0, 0, \ldots, 0, \ldots, 1)$	$y_2 = c$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}_i^T$	$\mathbf{x}_i^T = (x_{i1}, x_{i2}, \ldots, x_{ij}, \ldots x_{id})$	$\boldsymbol{y}_i^T = (y_{i1}, y_{i2}, \ldots, y_{ik}, \ldots y_{ic})$ $= (1, 0, \ldots, 0, \ldots, 0)$	$y_i = 1$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$\mathbf{x}_n^T$	$\mathbf{x}_n^T = (x_{n1}, x_{n2}, \ldots, x_{nj}, \ldots x_{nd})$	$\boldsymbol{y}_n^T = (y_{n1}, y_{n2}, \ldots, y_{nk}, \ldots y_{nc})$ $= (0, 0, \ldots, 1, \ldots, 0)$	$y_n = k$

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

3.4.1 목적함수의 정의

p.159 식 (3.18) argmax => argmin

- 앞의 식(3.17)은 $\mathbf{f(X)}$ 로 표기했지만 엄밀히는 $\mathbf{f(X;\theta)}$ 와 같다. $\mathbf{f}$가 $\boldsymbol{\theta}$를 매개변수화 되어 있다는 것이다. [그림 3-14(a)]의 2층 퍼셉트론에서는 $\boldsymbol{\theta} = \{\mathbf{U}^1, \mathbf{U}^2\}$ 이다 기계학습이 학습해야 할 일은

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} ||\mathbf{f}(\boldsymbol{X}; \boldsymbol{\theta}) - \mathbf{Y}||_2^2$$

다시 말해, 기계학습은 다층 퍼셉트론의 출력 $\mathbf{f(X;\theta)}$ 와 주어진 부류정보 $\mathbf{Y}$ 와의 차이를 최소로 하는 매개변수 $\hat{\boldsymbol{\theta}}$ 을 찾아야 한다.

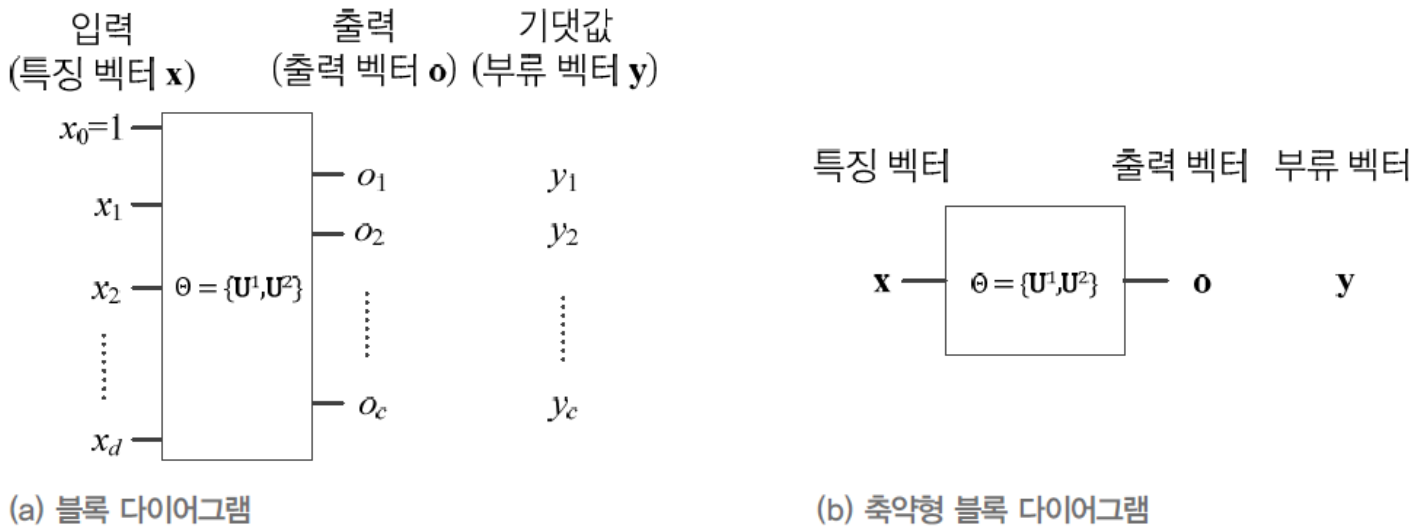
온라인모드: 하나의 샘플에 대해 그레이디언트를 계산하고 바로 가중치를 갱신하는 방식
=> [알고리즘 2-5](스토캐스틱 경사 하강법)

배치모드: 모든 샘플의 그레이디언트를 계산한 후 한꺼번에 가중치를 갱신하는 방식
=> [알고리즘 2-45](배치 경사 하강법)

3.4.1 목적함수의 정의

- 목적함수
- 평균 제곱 오차MSE(mean squared error) 로 정의 (다층 퍼셉트론: 주로 MSE 사용,
  딥러닝: 교차엔트로피 또는 로그우드 사용(5.1절))
  - 미분할 때 상수항을 1로 유지하여 수식 표현을 간단하게 하려고 ½을 곱해준다.

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



(a) 블록 다이어그램 (b) 축약형 블록 다이어그램

그림 3-17 목적함수 정의에 사용하는 입력과 출력, 기댓값

3.4.2 오류 역전파 알고리즘의 설계

- 식 (3.19)의 목적함수를 다시 쓰면,
  - 2층 퍼셉트론의 경우 $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$

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

  - 기계학습 알고리즘은 식(3.20)의 목적함수값이 줄어드는 방향으로 $\Theta = \{\mathbf{U}^1, \mathbf{U}^2\}$ 의 값을 정해야 함.
- $J(\Theta) = J(\{\mathbf{U}^1, \mathbf{U}^2\})$의 최저점을 찾아주는 경사 하강법

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

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 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$, $\mathbb{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$, 학습률 ρ
출력: 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$

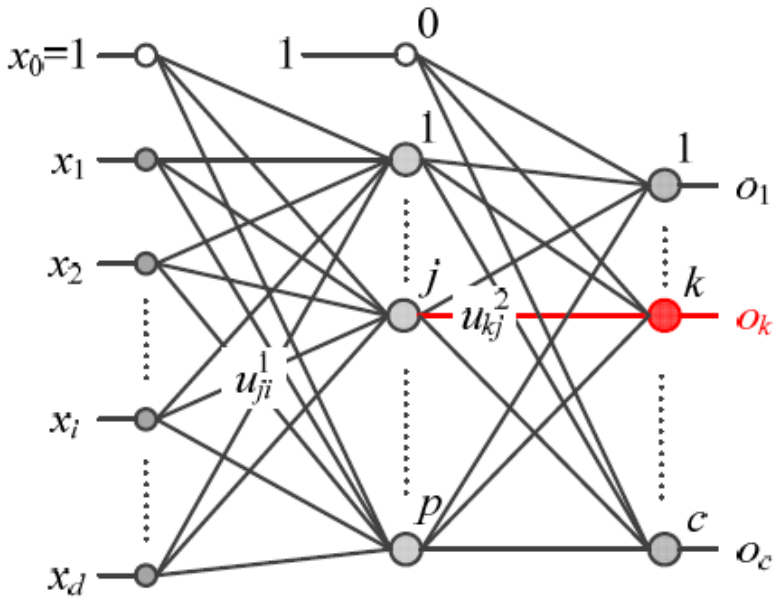
```
1  U¹과 U²를 초기화한다.
2  repeat
3     𝕏의 순서를 섞는다.
4     for (𝕏의 샘플 각각에 대해)
5           식 (3.15)로 전방 계산을 하여 o를 구한다.
6           ∂J/∂U¹와 ∂J/∂U²를 계산한다.
7           식 (3.21)로 U¹과 U²를 갱신한다.
8  until (멈춤 조건)
```

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}^2로 미분하면,

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



(a) u_{kj}^2가 미치는 영향

그림 3-18 매개변수가 미치는 영향

3.4.2 오류 역전파 알고리즘의 설계

오류 역전파의 유도

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

$$\frac{\partial J}{\partial u_{ji}^1} = \frac{\partial (0.5\|\mathbf{y}-\mathbf{o}(\mathbf{U}^1,\mathbf{U}^2)\|_2^2)}{\partial u_{ji}^1}$$

$$= \frac{\partial \left(0.5\sum_{q=1}^{c}(y_q-o_q)^2\right)}{\partial u_{ji}^1}$$

$$= -\sum_{q=1}^{c}(y_q-o_q)\frac{\partial o_q}{\partial u_{ji}^1}$$

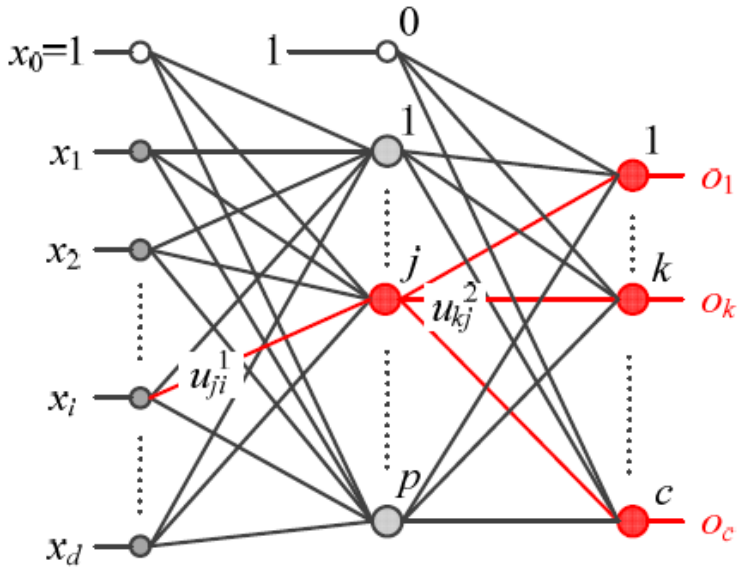
$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)\frac{\partial osum_q}{\partial u_{ji}^1}$$

$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)\frac{\partial osum_q}{\partial z_j}\frac{\partial z_j}{\partial u_{ji}^1}$$

$$= -\sum_{q=1}^{c}(y_q-o_q)\tau'(osum_q)u_{qj}^2\frac{\partial z_j}{\partial u_{ji}^1}$$

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

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



(b) u_{ji}^1이 미치는 영향

3.4.2 오류 역전파 알고리즘의 설계

- 지금까지 유도한 식을 정리하면, (교재 p.162~165 읽어볼 것!)
  (이 식들은 알고리즘에 사용할 아주 중요한 식임)

$$\delta_k = (y_k - o_k)\tau'(osum_k), \qquad 1 \le k \le c \tag{3.22}$$

$$\frac{\partial J}{\partial u_{kj}^2} = \Delta u_{kj}^2 = -\delta_k z_j, \qquad 0 \le j \le p, 1 \le k \le c \tag{3.23}$$

- (교재 p.163) 지금까지 유도한 식을 중간과정을 빼고 다시 쓰면 식(3.22)와 식(3.23)이 된다.
- 여기서 중요한 점은 두 항의 곱, 즉 $(y_k - o_k)\tau'(osum_k)$ 를 미리 계산하여 δ_k 라는 변수에 따로 저장하는 것이다. 이렇게 하면 j 가 달라질 때마다 같은 계산을 반복하는 비효율성을 피할 수 있다. $osum_k$ 와 z_j는 식(3.13)과 식(3.14)의 전방 계산에서 구했던 값인데, 전방 계산 과정에서 따로 저장해야 한다. 식(3.22)와 식 (3.23)은 알고리즘이 사용할 아주 중요한 식이다.

3.4.2 오류 역전파 알고리즘의 설계

- (교재 p.164) 식 (3.24) 와 식(3.25):

식 (3.24)는 앞에서와 마찬가지로 중복 계산을 방지하기 위해 η_j 를 미리 계산한다. δ_q는 식 (3.22)에서 계산한 값이고, $zsum_j$는 식 (3.13) 의 전방계산에서 계산한 값이다.

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

$$\frac{\partial J}{\partial u_{ji}^1} = \Delta u_{ji}^1 = -\eta_j x_i, \qquad 0 \le i \le d, 1 \le j \le p \tag{3.25}$$

- 오류 역전파error back-propagation 알고리즘
  - 식 (3.22)~(3.25)를 이용하여 출력층의 오류를 역방향(왼쪽)으로 전파하며 그레이디언트를 계산하는 알고리즘

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  U¹과 U²를 초기화한다.
2  repeat
3     𝕏의 순서를 섞는다.
4     for (𝕏의 샘플 각각에 대해)
5        현재 처리하는 샘플을 x = (x_0, x_1, x_2, ..., x_d)^T, y = (y_1, y_2, ..., y_c)^T라 표기한다.
6        x_0과 z_0을 1로 설정한다.   // 바이어스
                        // 전방 계산
7        for (j=1 to p)  zsum_j = u_j^1 x, z_j = τ(zsum_j)     // 식 (3.13)
8        for (k=1 to c)  osum_k = u_k^2 z, o_k = τ(osum_k)     // 식 (3.14)
                        // 오류 역전파
9        for (k=1 to c)  δ_k = (y_k − o_k)τ'(osum_k)           // 식 (3.22)
10       for (k=1 to c) for (j=0 to p)  Δu_kj^2 = −δ_k z_j     // 식 (3.23)
11       for (j=1 to p)  η_j = τ'(zsum_j) Σ_{q=1}^{c} δ_q u_qj^2   // 식 (3.24)
12       for (j=1 to p) for (i=0 to d)  Δu_ji^1 = −η_j x_i     // 식 (3.25)
                        // 가중치 갱신
13       for (k=1 to c) for (j=0 to p)  u_kj^2 = u_kj^2 − ρΔu_kj^2   // 식 (3.21)
14       for (j=1 to p) for (i=0 to d)  u_ji^1 = u_ji^1 − ρΔu_ji^1   // 식 (3.21)
15 until (멈춤 조건)
```

3.4.3 오류 역전파를 이용한 학습 알고리즘

- 라인 3~14는 한 세대를 구성하는데 라인 3의 X의 순서를 섞고 한 번에 샘플 하나씩 순서대로 처리한다. 따라서 모든 샘플에 한 번씩 기회가 주어진다. 그러나 [알고리즘 3-4]의 라인 3~6을 임의 샘플링 방식으로 바꾸어서 다음처럼 수정하여 사용할 수도 있다. 수정된 코드에서는 임의로 샘플을 하나 뽑아 처리한다. 따라서 샘플마다 다른 빈도로 처리 될 수 있다.

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

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

- [알고리즘 3-4]는 샘플 하나의 그레이디언트를 계산한 후 즉시 가중치를 갱신한다. 이러한 방식을 스토캐스틱 학습 또는 온라인 학습(online learning)이라고 한다. 3.5절에서는 여러 샘플의 그레이디언트 평균을 구한 다음 한꺼번에 가중치를 갱신하는 미니배치 방식을 학습한다. 미니배치는 온라인과 배치 방식의 중간 형태인데, 현대 기계학습은 미니배치 방식을 주로 사용한다.

3.4.3 오류 역전파를 이용한 학습 알고리즘

- 행렬 표기: GPU를 사용한 고속 행렬 연산에 적합 (교재 p.167 읽어볼 것!)

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 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^T$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^T$라 표기한다.
6 x_0과 z_0을 1로 설정한다. // 바이어스
  // 전방 계산
7 $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, ${\mathbf{U}^1}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
8 $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, ${\mathbf{U}^2}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
  // 오류 역전파
9 $\boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
10 $\Delta\mathbf{U}^2 = -\boldsymbol{\delta}\mathbf{z}^T$ // 식 (3.23), ${\Delta\mathbf{U}^2}_{c*(p+1)}$
11 $\boldsymbol{\eta} = (\boldsymbol{\delta}^T\tilde{\mathbf{U}}^2)^T \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), ${\tilde{\mathbf{U}}^2}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
12 $\Delta\mathbf{U}^1 = -\boldsymbol{\eta}\mathbf{x}^T$ // 식 (3.25), ${\Delta\mathbf{U}^1}_{p*(d+1)}$
  // 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)

라인 9와 11에 있는 ⊙ 는 요소별 곱을 뜻한다. 예를 들어, $(2\ 1\ 6\ 3)^T \odot (2\ 2\ 1\ 3)^T = (4\ 2\ 6\ 9)^T$ 이다.
라인 7에 있는 $\tilde{\mathbf{z}}$ 은 바이어스에 해당하는 z_0를 제거한 p개 요소를 가진 벡터이다. 라인 8의 $\mathbf{z}$는 z_0 까지 포함한 벡터이다.

3.5 미니배치 스토캐스틱 경사 하강법

미니배치 방식

- 한번에 t개의 샘플을 처리함 (t는 미니배치 크기)
  - t=1이면 스토캐스틱 경사 하강법(SGD(stochastic gradient descent method)) ([알고리즘 3-4])
  - t=n이면 배치 경사 하강법 : 모든 샘플의 그레이디언트를 계산한 다음 평균 그레이디언트를 구하여 한꺼번에 갱신
- 미니배치(mini batch) 방식은 보통 t=수십~수백
  - 훈련집합 $\mathbf{X}$에서 t개의 샘플을 무작위로 뽑아 미니배치를 구성하고, 미니배치에 속한 샘플의 그레이디언트 평균으로 가중치를 갱신하다.
  - 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 t개의 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 $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^{\mathrm{T}}$, $\mathbf{y} = (y_1, y_2, \cdots, y_c)^{\mathrm{T}}$라 표기한다.
7 x_0와 z_0를 1로 설정한다. // 바이어스
  // 전방 계산
8 $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$, $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13), $\mathbf{zsum}_{p*1}$, ${\mathbf{U}^1}_{p*(d+1)}$, $\mathbf{x}_{(d+1)*1}$, $\tilde{\mathbf{z}}_{p*1}$
9 $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$, $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14), $\mathbf{osum}_{c*1}$, ${\mathbf{U}^2}_{c*(p+1)}$, $\mathbf{z}_{(p+1)*1}$, $\mathbf{o}_{c*1}$
  // 오류 역전파
10 $\boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \odot \boldsymbol{\tau}'(\mathbf{osum})$ // 식 (3.22), $\boldsymbol{\delta}_{c*1}$
11 $\Delta\mathbf{U}^2 = \Delta\mathbf{U}^2 + (-\boldsymbol{\delta}\mathbf{z}^{\mathrm{T}})$ // 식 (3.23)을 누적, ${\Delta\mathbf{U}^2}_{c*(p+1)}$
12 $\boldsymbol{\eta} = \left(\boldsymbol{\delta}^{\mathrm{T}}\tilde{\mathbf{U}}^2\right)^{\mathrm{T}} \odot \boldsymbol{\tau}'(\mathbf{zsum})$ // 식 (3.24), ${\tilde{\mathbf{U}}^2}_{c*p}$, $\boldsymbol{\eta}_{p*1}$
13 $\Delta\mathbf{U}^1 = \Delta\mathbf{U}^1 + (-\boldsymbol{\eta}\mathbf{x}^{\mathrm{T}})$ // 식 (3.25)를 누적, ${\Delta\mathbf{U}^1}_{p*(d+1)}$
  // 가중치 갱신
14 $\mathbf{U}^2 = \mathbf{U}^2 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^2$ // 식 (3.21) - 평균 그레이디언트로 갱신
15 $\mathbf{U}^1 = \mathbf{U}^1 - \rho\left(\frac{1}{t}\right)\Delta\mathbf{U}^1$ // 식 (3.21) - 평균 그레이디언트로 갱신
16 until (멈춤 조건)

[알고리즘 3-5]와 차이점

라인3: **X'**, **Y'**
라인 4: 초기화 =0
라인5: **X'**
라인11: 누적
라인12: 누적
라인14,15: t로 나눔.

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 x // 신경망의 가중치 $\mathbf{U}^1$과 $\mathbf{U}^2$는 이미 설정되었다고 가정함.
출력: 부류 y

1. $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)^{\mathrm{T}}$로 확장하고, x_0과 z_0을 1로 설정한다.
2. $\mathbf{zsum} = \mathbf{U}^1\mathbf{x}$
3. $\tilde{\mathbf{z}} = \boldsymbol{\tau}(\mathbf{zsum})$ // 식 (3.13)
4. $\mathbf{osum} = \mathbf{U}^2\mathbf{z}$
5. $\mathbf{o} = \boldsymbol{\tau}(\mathbf{osum})$ // 식 (3.14)
6. $\mathbf{o}$에서 가장 큰 값을 가지는 노드에 해당하는 부류 번호를 y에 대입한다.

- 라인 6을 수식으로 표현하면, $y = \underset{k}{\operatorname{argmax}}\, o_k$
- 전방 계산 한번만 사용하므로 빠름 (뒤의 슬라이드 표 3-2 참조)
- 신경망은 학습을 마치면 훈련집합을 가지고 있을 필요가 없다. 대신 가중치를 저장한다. 가중치만 저장하면 되기 때문에 메모리 효율이 매우 높다.

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

※ 교재에서 노드의 개수를 나타내는 기호:

특징(x)의 개수 : d 은닉노드(z)의 개수: p 또는 q 출력노드(부류)(o)의 개수: c

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 역전파를 반복해야 하므로 점근적 시간 복잡도는 $\Theta((dp+pc)nq)$

3.7.2 모든 함수를 정확하게 근사할 수 있는 능력

- 호닉의 주장[Hornik1989] : 수학적으로 증명
  - 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."

  - 이 말은 어떤 데이터든 은닉층이 하나인 다층 퍼셉트론으로 100% 분류할 수 있다고 말할 수 있다. 하지만 현실은 그렇지 않다. 사실 이 이론은 함수를 근사화하는 신경망이 있다는 사실을 보인 것이지, 신경망이 모든 함수를 근사화하는 데 적절한 도구라고 주장하는 것은 아니며, 더구나 그런 신경망을 어떻게 찾을 것인지에 대한 어떠한 아이디어도 제시하지 못한다. 그런데도 이 이론은 신경망의 성공적인 응용이 요행이 아니라 이론적인 근거에서 발생한 것이라는 믿음을 주며, 신경망의 개선을 통해 현재 풀리지 않는 문제를 추가로 풀 수 있다는 희망을 준다.
  - 은닉 노드를 무수히 많게 할 수 없으므로, 실질적으로는 복잡한 구조의 데이터에서는 성능 한계
  - 은닉층이 하나뿐이어도 충분하다는 이론에 따라 은닉층이 하나인 신경망을 개선하려는 노력과 은닉층을 여럿으로 늘려 깊은 신경망으로 확장하여 개선하는 두 줄기의 접근방법이 있다. 현재로서는 분명 4장에서 학습할 깊은 신경망이 우세를 점하고 있다. (교재 p.175)

3.7.3 성능 향상을 위한 휴리스틱의 중요성

- 순수한 최적화 알고리즘으로는 높은 성능 불가능
  - 데이터 희소성, 잡음, 미숙한 신경망 구조 등의 이유
  - 성능 향상을 위한 갖가지 휴리스틱을 개발하고 공유함 → 예) 『Neural Networks: Tricks of the Trade』[Montavon2012]
- 휴리스틱 개발에서 중요 쟁점: 하이퍼 매개변수

  • 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

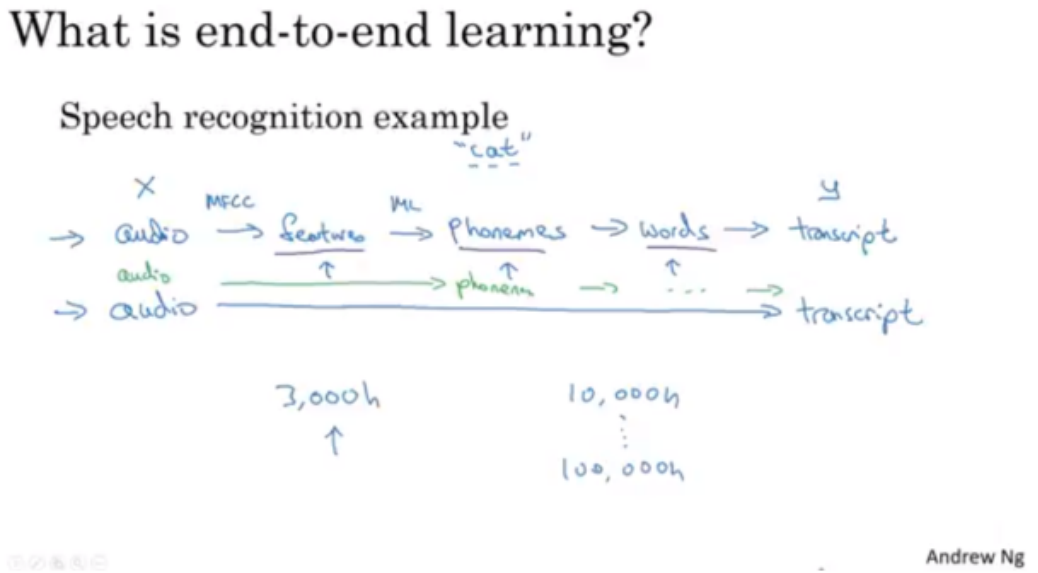
- 실용적인 성능
  - 1980~1990년대에 다층 퍼셉트론은 실용 시스템 제작에 크게 기여
    - 인쇄/필기 문자 인식으로 우편물 자동 분류기, 전표 인식기, 자동차 번호판 인식기 등
    - 음성 인식, 게임, 주가 예측, 정보 검색, 의료 진단, 유전자 검색, 반도체 결함 검사 등

- 하지만 한계 노출
  - 잡음이 섞인 상황에서 음성인식 성능 저하
  - 필기 주소 인식 능력 저하
  - 바둑에서의 한계

- 딥러닝은 이들 한계를 극복함

4. End-to-End Learning

정말 간단한 개념이다. 기존 전통적인 학습 방법은 인간이 특정 데이터를 보고 데이터 분류를 위한 Feature를 직접 지정해 분류기 모델을 만드는 형식이었다. 하지만 end-to-end 학습 방식은 Input X 데이터와 정답 Y만을 이용해 분류를 하는 즉 인간이 feature나 중간과정을 만들 필요가 없는 모델이다. 하지만, 막상 end-to-end의 내부를 들여다보면 결국, 각 neural network layer의 neuron 자체가 feature를 분류하는 분류기가 되기 때문에 실상은 자동으로 feature를 system이 정답을 보고 만든다 라고 이해할수 있다. end-to-end를 할수 있는 조건은 일단 데이터가 많을때 가능하다. (정량적으로 어느정도라고는 말할수 없지만), 하지만 데이터가 적을 경우 사람이 feature나 rule을 정해주는것이 더 성능이 좋을때가 많다.



출처: https://blog.naver.com/jujbob/221146361024

[Deep Learning] end-to-end trainable neural network

Machine Learning/Deep Learning 2017.08.04 13:52

모델의 모든 매개변수가 하나의 손실함수에 대해 동시에 훈련되는 경로가 가능한 네트워크로써 역전파 알고리즘 (Backpropagation Algorithm) 과 함께 최적화 될 수 있다는 의미이다.

예를들어 인코더(언어의 입력을 벡터로)와 디코더(벡터의 입력을 언어로)에서 모두가 동시에 학습되는 기계 번역 문제에서 효과적으로 적용 될 수 있다.

즉, 신경망은 한쪽 끝에서 입력을 받아들이고 다른 쪽 끝에서 출력을 생성하는데, 입력 및 출력을 직접 고려하여 네트워크 가중치를 최적화 하는 학습을 종단 간 학습(End-to-end Learning) 이라고 한다.

Convolutional neural network 가 카메라의 원시 픽셀을 명령어에 직접 매핑하는 과정을 거치게 될 때 역전파는 종종 입력을 해당 출력으로 매핑하는 것과 관련하여 네트워크 가중치를 학습하는 효율적인 방법으로 사용된다.

만약, 신경망에 너무 많은 계층의 노드가 있거나 메모리가 적합하지 않을 경우 종단 간(end-to-end) 방식으로 훈련 시킬 수 없다. 이 경우 네트워크를 더 작은 네트워크의 파이프라인으로 나누어 해결 할 수 있는데 각 작은 네트워크는 독립적으로 훈련을 하게 되고 원하는 출력을 얻기 위해 연결 될 수 있다. 이러한 방식을 Divide and train 이라고 한다. 최적화가 중간 산출물에 의존한다는 점에서 국지적으로 수행이 되기 때문에 최적의 결과를 보장할 수 없다.

출처: http://eehoeskrap.tistory.com/183