인공지능

Artificial Intelligence

신경망 기초

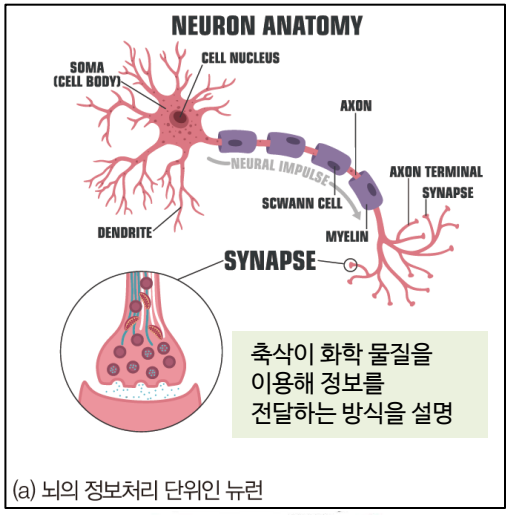
본 강의자료는 한빛아카데미 〈파이썬으로 만드는 인공지능〉을 내용을 기반으로 제작 및 보완되었음을 알립니다.

4.1 인공 신경망의 태동

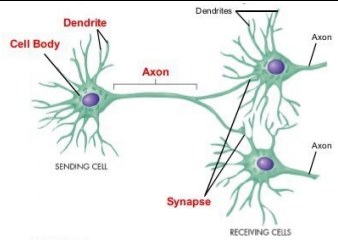
- 인공 신경망은 생물 신경망에서 영감을 얻었지만 실제 구현은 생물 신경망의 원리와 상당히 다름
- 인공 신경망을 구현하는데 쓸 수 있는 기계는 컴퓨터가 유일한데 컴퓨터의 기본 구조와 원리는 생물의 작동 원리와 근본적으로 다르기 때문임

4.1.1 생물 신경망

- 뉴런은 뇌가 수행하는 정보처리의 단위로서, 세포체cell body, 수상돌기dendrite, 축삭axon으로 구성됨
  - 세포체는 간단한 연산을 수행함
  - 수상돌기는 다른 뉴런으로부터 정보를 받음
  - 축삭은 세포체가 처리한 정보를 다른 뉴런에 전달함
- 시냅스synapse 는 한 뉴런에서 다른 뉴런으로 신호를 전달하는 연결 지점
- 뉴런은 서로 연결된 망을 형성하므로 신경망neural network이라 부르는데, 인공 신경망이 등장한 이후에는 서로 구별하기 위해 생물 신경망이라 부름



(a) 뇌의 정보처리 단위인 뉴런

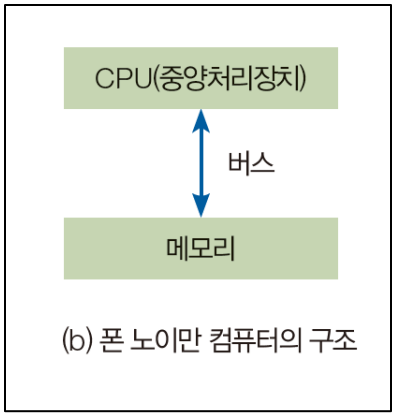


4.1 인공 신경망의 태동

- 인공 신경망은 생물 신경망에서 영감을 얻었지만 실제 구현은 생물 신경망의 원리와 상당히 다름
  - 인공 신경망을 구현하는데 쓸 수 있는 기계는 컴퓨터가 유일한데 컴퓨터의 기본 구조와 원리는 생물의 작동 원리와 근본적으로 다르기 때문임

4.1.1 폰 노이만 컴퓨터Von Neumann computer의 구조

- 현재 우리가 사용하고 있는 컴퓨터는 모두 폰 노이만 구조임
- 메모리는 프로그램(명령어 집합)과 데이터를 저장함
- CPU는 메모리에서 명령어를 한 번에 하나씩 가져가 수행하고 결과를 메모리에 저장함
- 컴퓨터는 아주 빠른 순차 명령어 처리기
- 컴퓨터는 뇌의 병렬 처리와는 근본적으로 다른 정보처리 방식을 사용함



(b) 폰 노이만 컴퓨터의 구조

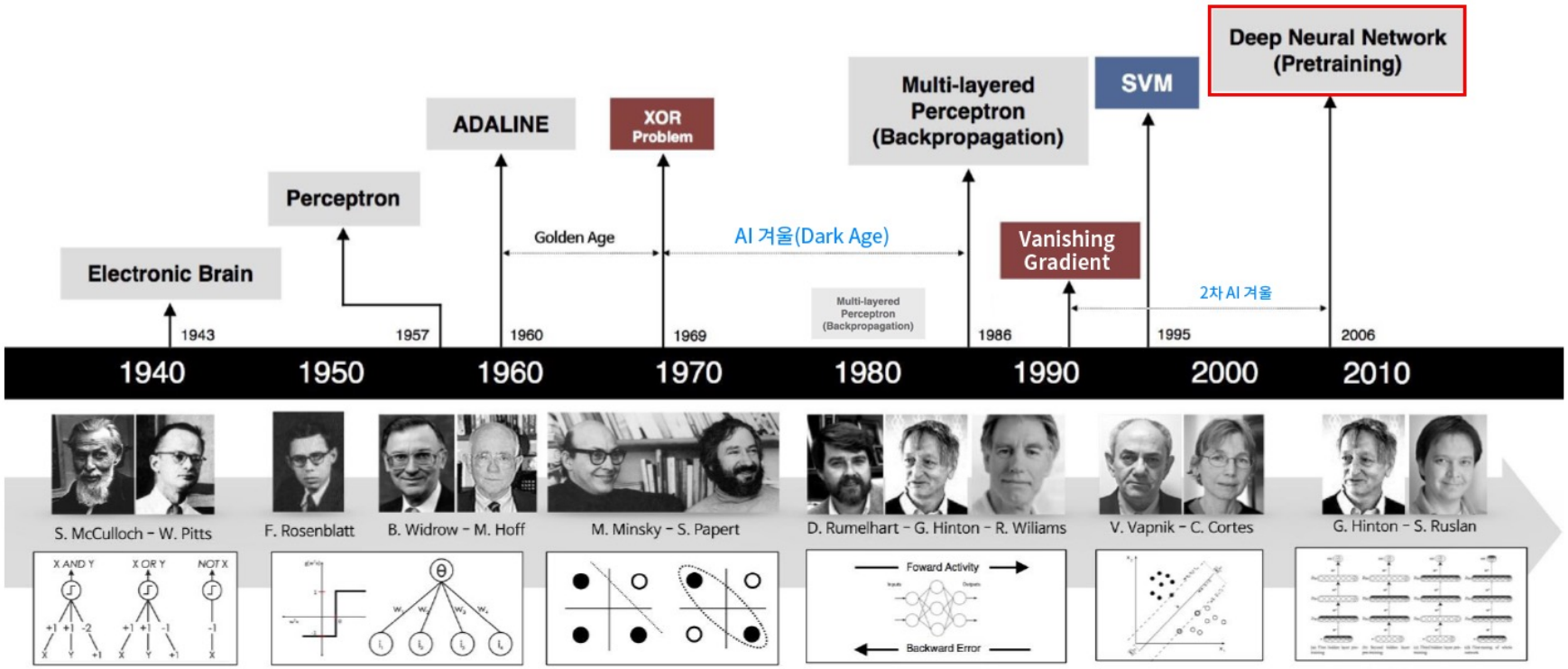
4.1 인공 신경망의 태동

4.1.2 인공 신경망의 발상과 전개

- **1940년대**에 생물 신경망 연구자 중 일부는 신경망 동작을 인공적으로 모방하는 일에 관심을 두게 됨
  - 심리학자와 신경의학자가 연구를 주도함
- **1943년에는** 맥컬록과 피츠가 신경망의 시초로 볼 수 있는 계산 모형을 발표함
- **1949년에는** 헤브가 최초로 학습 알고리즘을 발표함
- **1958년에는** 로젠블랫이 **퍼셉트론을** 제안함
- 위드로와 호프는 퍼셉트론의 학습 알고리즘을 개선한 아달린과 아달린을 이어 붙인 마달린을 발표함
- **1960년대에는** 신경망으로 모든 문제를 해결할 수 있는 것처럼 과장되어 주목을 받음
- **1969년** 민스키와 페퍼트는 **퍼셉트론의 한계를 수학적으로 입증함**
  - 퍼셉트론은 선형 분류기에 불과, XOR 분류도 못한다고 단정함, 이후 신경망에 대한 연구가 크게 퇴조
- **1986년** 루멜하트와 동료들이 은닉층을 가진 **다층 퍼셉트론**과 **오류 역전파 알고리즘**을 제시함
  - 이 신경망은 필기 숫자 인식과 같은 복잡한 분류 문제를 훌륭하게 해결할 수 있음을 입증하였으며,
    이후 신경망 연구는 다시 활기를 찾았고 분류를 포함한 문제해결에 가장 널리 사용되는 도구가 됨
- **1990년대** SVM이 신경망보다 좋은 성능을 보여 신경망이 SVM에 밀리는 형국이 되지만, 딥러닝이 실현되어 다시 신경망이 기계 학습의 주류로 자리 잡게 됨

4.1 인공 신경망의 태동

4.1.2 인공 신경망의 발상과 전개



4.1 인공 신경망의 태동

Note. 디지털 컴퓨터와 역사를 같이 하는 인공 신경망

1940년대에는 컴퓨터과학이라는 학문 분야가 없었음. 당시의 컴퓨터는 기계가 아니라 계산을 전문으로 하는 전문 직업인을 지칭했음. 1946년 세계 최초의 전자식 컴퓨터인 에니악이 탄생했고, 1950년대에는 상업용 컴퓨터가 출시됨. 이때쯤 인공 신경망이 등장하니 컴퓨터와 인공 신경망은 얼추 역사를 같이하는 셈. 1950년 들어서야 대학에 컴퓨터과학이라는 학문 분야가 태동하였으며, 1960년대에 제대로 된 교육과정으로 자리를 잡음. 컴퓨터의 속도와 메모리 용량이 발전함에 따라 신경망의 성능 또한 크게 발전함. 현재는 고도의 병렬 처리기인 GPU의 도움으로 딥러닝이 꽃을 피우고 있음

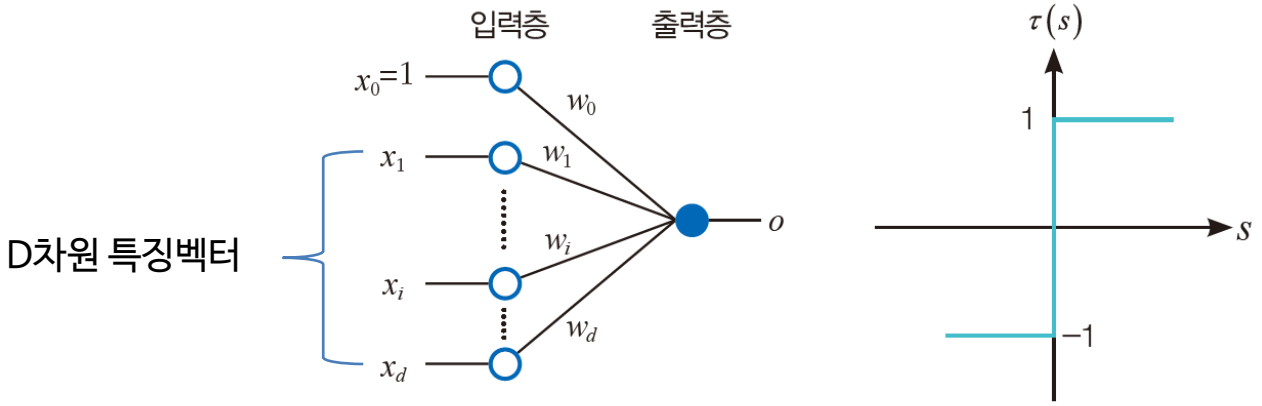
4.2 퍼셉트론의 원리

- 퍼셉트론은 1958년에 탄생했으니 세월로만 판단하면 아주 낡은 기술임
- 하지만 신경망을 공부하려면 퍼셉트론의 원리를 이해하는 것이 아주 중요함
- 왜냐하면 퍼셉트론은 이후에 나온 다층 퍼셉트론과 현재 인공지능 기술을 구현하는데 핵심 역할을 담당하는 딥러닝의 구성요소로 작용하기 때문임
- 퍼셉트론을 제대로 이해하면 현대 인공지능의 중심에 있는 딥러닝을 이해하는데 크게 도움이 됨
- 또한 퍼셉트론은 단순한 모델이어서 기계 학습의 용어와 원리를 설명하는데 적합한데, 이 용어들은 원리가 딥러닝에 그대로 쓰이거나 일부 변형되어 적용됨

4.2 퍼셉트론의 원리

4.2.1 퍼셉트론의 구조와 연산

- 그림(a)와 같이 퍼셉트론은 입력층과 출력층으로 구성됨
- 왼쪽에 있는 층은 외부로부터 입력을 받는 입력층input layer이고, 오른쪽은 계산 결과를 외부로 내부내는 출력층output layer 임
- 입력층에는 d+1개의 노드node가 있음. d는 특징 벡터의 차원인데, iris의 경우 d는 4 이고, 필기 숫자에서 화소를 특징으로 사용하는 경우 d는 64임



(a) 퍼셉트론의 구조 (b) 계단 함수를 활성 함수로 사용

그림 4-2 퍼셉트론의 구조와 동작

4.2 퍼셉트론의 원리

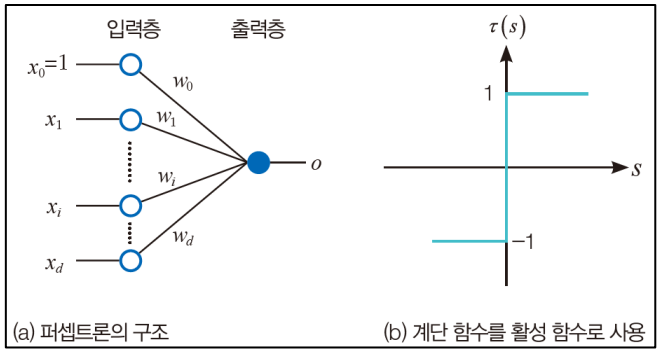
4.2.1 퍼셉트론의 구조와 연산

- 입력층의 i 번째 노드와 출력층의 노드는 가중치 w_i 를 갖는 에지edge로 연결됨
- i 번째 에지는 특징 x_i와 가중치 w_i 를 곱해 출력 노드로 전달함
- 0번째 노드의 입력 x_0은 항상 1인데, 이 노드를 바이어스 노드bias node라 부름
- 출력 노드는 d+1개의 곱셈 결과를 모두 더한 s 를 계산함
- s 를 활성 함수 (τ)에 적용한 결과를 출력 o 로 내보냄

$$o = \tau(s) = \tau\left(\sum_{i=1}^{d} w_i x_i + w_0\right)$$

$$\text{이때 } \tau(s) = \begin{cases} +1, s > 0 \\ -1, s \le 0 \end{cases}$$

← 계단 함수



(a) 퍼셉트론의 구조 (b) 계단 함수를 활성 함수로 사용

- 활성 함수activation function는 두뇌가 뉴런을 활성화하는 과정을 모방함
- 퍼셉트론은 활성 함수로의 계단 함수step function를 사용함
- 퍼셉트론은 특징 벡터를 위의 식에 따라 1 또는 −1로 변환하는 장치로 볼 수 있음

4.2 퍼셉트론의 원리

4.2.2 퍼셉트론으로 인식하기

- 퍼셉트론의 동작은 인공지능과 무슨 관련이 있을까?
  - 퍼셉트론을 통해 두 부류로 구분하는 인식 문제를 해결할 수 있음

- 두 부류로 구분해야 하는 문제의 예
  - 생산 라인에서 나오는 제품을 불량/우량으로 구분하는 문제
  - 병원에 온 사람을 환자와 정상으로 구분하는 문제
  - 내가 등장하는 사진만 구분하는 문제

4.2 퍼셉트론의 원리

4.2.2 퍼셉트론으로 인식하기

[예시 4-1] 퍼셉트론의 인식 능력

- 어떤 제품이 크기와 색상을 나타내는 2개의 특징으로 표현
  - 특징 벡터는 X=(크기, 색상)=(x_1, x_2)로 표현할 수 있음
  - 입력 값(특징 값)은 0 또는 1을 갖는다고 가정, 출력 값은 -1 또는 1을 갖는다고 가정

- 생산 라인에서 제품 4개를 수집해 관찰한 결과, 다음과 같이 데이터를 얻고 불량 3개, 정상 1개로 판정

$\mathbf{x}_1=(0,0),\quad \mathbf{x}_2=(0,1),\quad \mathbf{x}_3=(1,0),\quad \mathbf{x}_4=(1,1)$ ← 특징 벡터

$y_1=-1,\quad y_2=1,\quad y_3=1,\quad y_4=1$ ← 레이블 (y=1은 불량, y=-1은 우량)

4.2 퍼셉트론의 원리

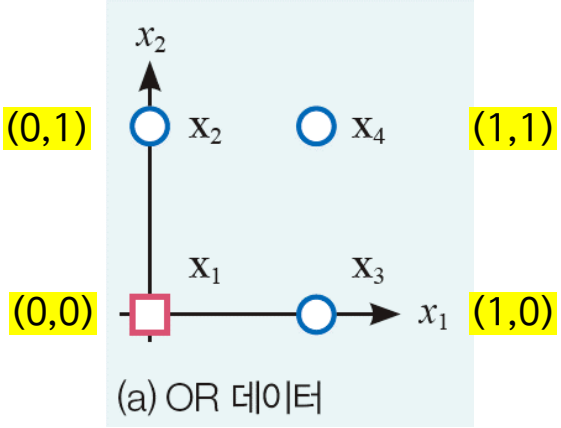
4.2.2 퍼셉트론으로 인식하기

[예시 4-1] 퍼셉트론의 인식 능력

$\mathbf{x}_1=(0,0), \quad \mathbf{x}_2=(0,1), \quad \mathbf{x}_3=(1,0), \quad \mathbf{x}_4=(1,1)$ ← 특징 벡터

$y_1=-1, \quad y_2=1, \quad y_3=1, \quad y_4=1$ ← 레이블 (y=1은 불량, y=-1은 우량)

〉〉 데이터를 그림으로 표현



(a) OR 데이터

[NOTE] 논리 연산을 본뜬 OR 데이터

- 왼쪽의 그림 데이터를 분류하는 문제를 OR 분류라고 함
- 1을 참, 0을 거짓으로 간주하면 이진 논리의 OR 연산에 해당되기 때문
- 단순하여 설명하기 쉽기 때문에 초심자에게 신경망의 동작을 설명할 때 종종 사용함

4.2 퍼셉트론의 원리

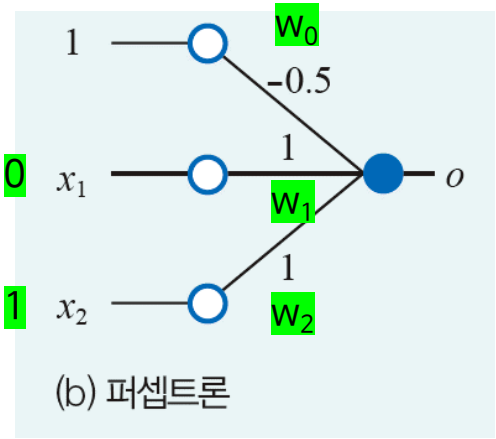
4.2.2 퍼셉트론으로 인식하기

[예시 4-1] 퍼셉트론의 인식 능력

$\mathbf{x}_1=(0,0),\quad \mathbf{x}_2=(0,1),\quad \mathbf{x}_3=(1,0),\quad \mathbf{x}_4=(1,1)$ ← 특징 벡터

$y_1=-1,\quad y_2=1,\quad y_3=1,\quad y_4=1$ ← 레이블 (y=1은 불량, y=-1은 우량)

〉〉 샘플 4개를 인식하는 퍼셉트론



(b) 퍼셉트론

$$o = \tau(s) = \tau\left(\sum_{i=1}^{d} w_i x_i + w_0\right)$$

$$\text{이때}\quad \tau(s) = \begin{cases} +1, s > 0 \\ -1, s \le 0 \end{cases}$$

➔ $x_2=(0,1)$ 샘플을 퍼셉트론에 입력으로 넣으면, 아래와 같은 계산에 의해 예측 값은 1이 나옴 (제대로 인식함, 예측값==참값)

$$o = \tau(w_1 x_1 + w_2 x_2 + w_0)$$
$$= \tau(1 * 0 + 1 * 1 - 0.5) = \tau(0.5) = 1$$

4.2 퍼셉트론의 원리

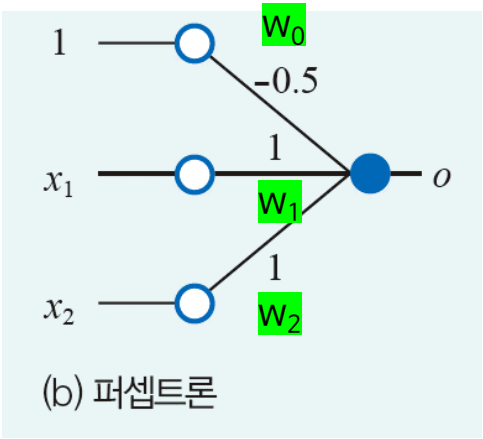
4.2.2 퍼셉트론으로 인식하기

[예시 4-1] 퍼셉트론의 인식 능력

$\mathbf{x}_1=(0,0),\quad \mathbf{x}_2=(0,1),\quad \mathbf{x}_3=(1,0),\quad \mathbf{x}_4=(1,1)$ ← 특징 벡터

$y_1=-1,\quad y_2=1,\quad y_3=1,\quad y_4=1$ ← 레이블 (y=1은 불량, y=-1은 우량)

〉〉 샘플 4개를 인식하는 퍼셉트론



(b) 퍼셉트론

➔ x_1=(0,0) 샘플을 퍼셉트론에 입력으로 넣으면,

$o = \tau(w_1x_1 + w_2x_2 + w_0)$
$= \tau(1*0 + 1*0 - 0.5) = \tau(-0.5) = -1$

➔ x_3=(1,0) 샘플을 퍼셉트론에 입력으로 넣으면,

$o = \tau(w_1x_1 + w_2x_2 + w_0)$
$= \tau(1*1 + 1*0 - 0.5) = \tau(0.5) = 1$

➔ x_4=(1,1) 샘플을 퍼셉트론에 입력으로 넣으면,

$o = \tau(w_1x_1 + w_2x_2 + w_0)$
$= \tau(1*1 + 1*1 - 0.5) = \tau(1.5) = 1$

모두 맞게 예측

4.2 퍼셉트론의 원리

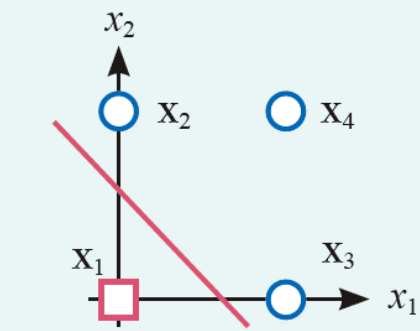
4.2.2 퍼셉트론으로 인식하기

[예시 4-1] 퍼셉트론의 인식 능력

$\mathbf{x}_1=(0,0), \quad \mathbf{x}_2=(0,1), \quad \mathbf{x}_3=(1,0), \quad \mathbf{x}_4=(1,1)$ ← 특징 벡터

$y_1=-1, \quad y_2=1, \quad y_3=1, \quad y_4=1$ ← 레이블 (y=1은 불량, y=-1은 우량)

〉〉 퍼셉트론 결정 경계



(c) 퍼셉트론의 결정 경계

➔ s 〉 0 이면 +1 영역, s 〈 0 이면 −1 영역, s = 0 이면 결정 경계

$$s = w_1x_1 + w_2x_2 + w_0$$
$$= x_1 + x_2 - 0.5$$

$x_1 + x_2 - 0.5 = 0$ (2개의 부류를 분별하는 결정 직선)

$\therefore x_2 = -x_1 + 0.5$

특징 공간을 두 공간으로 분할하는 이진 분류기이며, 퍼셉트론의 결정 경계는 선형에 국한됨

4.2 퍼셉트론의 원리

4.2.3 행렬 표기

- 퍼셉트론의 합 s 를 구하는 식을 Σ로 표기하고 있으며, 이 식을 코딩하려면 반복문을 사용함

[C 코드]

```
s = 0;
for(I = 0; I <= d; i++)
  s = s + x[i]*w[i];
```

4.2.3.1 퍼셉트론 동작의 행렬 표기

- Σ로 표기하는 방식에 머물면 두 가지 문제가 발생함
  - 첫째, 기계 학습을 다루는 대다수의 책, 논문, 보고서는 행렬을 사용함
  - 둘째, 파이썬 언어는 반복문 대신 행렬 연산을 주로 사용함
- 파이썬은 x와 w가 벡터라는 사실을 알고 있어 요소별 곱셈의 합을 수행함
  - 내부적으로는 C와 마찬가지로 for 문으로 변환해 반복을 통해 실행하는데, 프로그래머는 내부적인 실행 절차에 신경 쓰지 않고 코딩할 수 있음

[파이썬 코드]

```
# 요소별로 곱한 후 요소를 합산
s = sum(x*w)
```

4.2 퍼셉트론의 원리

4.2.3.1 퍼셉트론 동작의 행렬 표기

- 퍼셉트론의 식을 선형 대수가 사용하는 행렬 표기로 다시 쓰기

$$o = \tau\left(\sum_{i=1}^{d} w_i x_i + b\right) = \tau(\mathbf{W}\mathbf{X}^{\mathrm{T}} + w_0) = \tau(\mathbf{W}\mathbf{X}^{\mathrm{T}})$$

행렬 표기 (1)　　행렬 표기 (2)

X는 특징 벡터, **W**는 가중치 벡터

X와 **W**는 1xd 행렬로 간주할 수 있음

X는 1xd 행렬, **X**의 전치 행렬 $\mathbf{X}^{\mathrm{T}}$는 dx1 행렬

1xd 행렬 w와 dx1 행렬 $\mathbf{X}^{\mathrm{T}}$의 곱은 1x1 행렬

즉 스칼라!!

가중치 벡터 **W**에 w_0를 추가해 1x(d+1) 행렬

W = $(w_0, w_1, w_2, \cdots, w_d)$로 만들고

특징 벡터 x에 x0를 추가해 1x(d+1) 행렬

X = $(x_0, x_1, x_2, \cdots, x_d)$로 만들면

행렬 표기(2)와 같이 간결하게 정리 가능

4.2 퍼셉트론의 원리

4.2.3.2 파이썬 코딩

- 파이썬에서는 numpy 라이브러리가 제공하는 array를 이용해 벡터와 행렬 연산을 간편하게 수행할 수 있음
- [예제 4-1]에 샘플 x_2 에 대해 계산하는 파이썬 코드

```
1  import numpy as np              # numpy 라이브러리를 불러옴
2  x2 = np.array([1,0,1])         # 샘플 X2
3  w=np.array([-0.5, 1.0,1.0])    # 퍼셉트론 가중치 벡터
4  s=sum(x2*w)                    # 요소별로 곱한 후 합산
```

- [예제4-1]에 있는 샘플 4개를 모두 계산하는 파이썬 코드

```
7  import numpy as np                                  # numpy 라이브러리를 불러옴
8  x=np.array([[1,0,0],[1,0,1],[1,1,0],[1,1,1]])      # 샘플 4개를 불러옴
9  w=np.array([-0.5, 1.0,1.0])                        # 퍼셉트론 가중치 벡터
10 s=np.sum(x*w,axis=1)                               # 샘플 각각에 대해 요소별로 곱한 후 요소를 합산
```

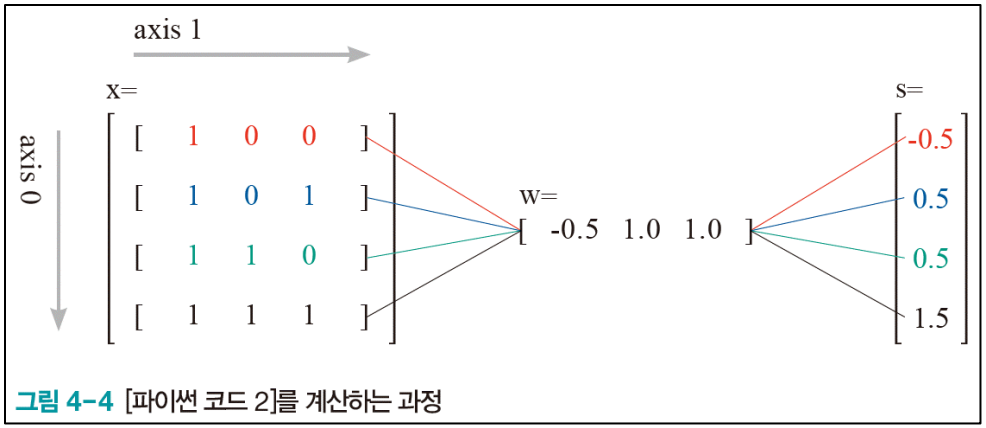
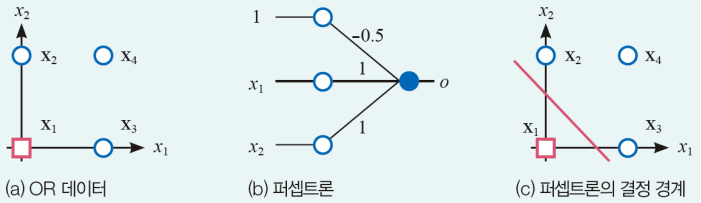


그림 4-4 [파이썬 코드 2]를 계산하는 과정

4.3 사람의 학습과 신경망의 학습



(a) OR 데이터 (b) 퍼셉트론 (c) 퍼셉트론의 결정 경계

그림 4-3 퍼셉트론의 인식 능력(OR 분류 문제)

- [그림4-3]에서는 데이터와 함께 가중치도 함께 주어짐
- 하지만 기계 학습에서는 데이터만 주어지고, 데이터를 제대로 인식하는 퍼셉트론, 즉 가중치를 알아내야 함. 이 과정이 퍼셉트론의 학습과정에 해당함

- [그림4-3]의 OR 데이터에서는 특징 벡터가 2차원이므로 알아내야 하는 가중치는 3개이고 샘플은 4개 뿐이라 최적의 가중치를 쉽게 찾을 수 있음
- iris 데이터의 경우 특징 벡터는 4차원이고 샘플은 150개이므로 사람이 종이와 연필을 들고 가중치 5개를 알아내는 길은 거의 불가능
- 숫자 데이터의 경우는 특징 벡터가 64차원이므로 65개의 가중치를 알아내야 하며, 샘플은 1796개여서 사람이 가중치를 알아내야 일은 불가능

- 따라서, 차원이 높고 샘플의 개수가 아주 많은 상황에서 〈퍼셉트론의 가중치를 알아내는 학습 알고리즘〉을 구상해야 함

4.3 사람의 학습과 신경망의 학습

4.3.1 사람의 학습 알고리즘

- 기계 학습의 근본 원리는 어떤 면에서는 사람의 학습과 비슷함
- 알고리즘 [4-1]은 사람이 수영을 학습하는 과정을 알고리즘 형식에 맞춰 기술한 것

[알고리즘 4-1] 사람의 수영 학습

```
01. 적절한 동작을 취한다.
02. while (true)
03.       동작에 따라 수영을 하고 평가한다.
04.       if (만족스러움) break                    # break문으로 루프 탈출
05.       더 나은 방향으로 동작을 수정한다.
06. 동작을 기억한다.
```

- (01행) 처음에는 어떤 동작이 좋은지 모르니 적절하다고 생각되는 동작을 설정하고 수영해 봄
- (03행) 그리고 잘 되는지 평가
- (05행) 만족스럽지 않다면 더 나은 방향으로 동작을 수정한 다음 같은 과정을 반복
- (04행) 이 과정을 반복하다가 만족스러운 상황이 되면 반복 루프를 빠져 나옴
- (06행) 이후에 멋진 수영을 하기 위해 찾아낸 동작을 기억함

4.3 사람의 학습과 신경망의 학습

4.3.1 사람의 학습 알고리즘

- 기계 학습의 근본 원리는 어떤 면에서는 사람의 학습과 비슷함
- 알고리즘 [4-1]을 좀 더 수학적으로 기술하면 [알고리즘 4-2]가 됨

[알고리즘 4-2] 사람의 수영 학습(수학적 기술)

01. 초기 동작 벡터 w=(팔 돌리는 속도, 팔꿈치 각도, 팔과 귀의 거리)를 초기화한다.
02. while (true)
03. w에 따라 수영을 하고 동작 w의 점수 $J(\mathbf{w})$를 계산한다.
04. if ($J(\mathbf{w})$가 만족스러움) break
05. 더 나은 방향 $\Delta\mathbf{w}$를 계산한다.
06. $\mathbf{w}=\mathbf{w}+\Delta\mathbf{w}$
07. w를 기억한다.

- (01행) 수영을 잘 하기 위해 세 가지 동작의 조합(w)만 생각하겠음
- (01행) 동작을 나타내는 벡터 w를 적절한 값으로 초기화

- (03행) w에 따라 실제로 수영을 해본 다음 w가 얼마나 좋은지 평가, 이때 평가 함수(목적 함수) J가 있어야 함
  - 가령 코치가 매기는 점수를 함수 값으로 쓸 수 있음
  - 프로 선수의 경우에는 일정 거리를 가는데 걸리는 시간을 쓸 수도 있음
- (04행) 수영이 만족스럽다면 루프를 탈출하고 학습 종료
- (05행) 그렇지 않으면, 수영을 더 잘 할 수 있는 값을 찾음
  - 가령 팔을 돌리는 속도를 0.5만큼 줄이고 팔꿈치 각도는 3도만큼 늘리고 팔과 귀의 거리는 1.2만큼 가까이 함. ➔ 이 경우가 $\Delta\boldsymbol{w} = (-0.5, 3, -1.2)$임.
- (06행) $\Delta\boldsymbol{w}$를 현재 동작 벡터에 더하여 다음에 쓸 동작 벡터를 계산
- (03행) 부터 같은 과정 반복

4.3 사람의 학습과 신경망의 학습

4.3.2 신경망의 학습 알고리즘

- 신경망의 학습은 사람의 학습과 비슷한 점도 있고 다른 점도 있음
  - 조금씩 나은 방향으로 찾아 개선해 나가는 절차는 같음
  - 기계 학습은 철저히 수학에 의존한다는 점에서 사람의 학습과 크게 다름

[알고리즘 4-3] 신경망의 학습

입력: 훈련 데이터

출력: 최적의 매개변수 값

```
01. 난수로 매개변수 벡터 w를 초기화한다.    # w는 신경망의 가중치([그림 4-2(a)]의 (w0,w1,…,wd))
02. while (true)
03.     w에 따라 데이터를 인식하고 손실 함수 J(w)를 계산한다.
04.     if (J(w)가 만족스러움) break
05.     손실 함수 값을 낮추는 방향 Δw를 계산한다.
06.     w=w+Δw
07. w를 저장한다.
```

- (01행) 난수를 생성해 매개변수를 초기화
  - 처음에는 매개변수 값을 어떻게 설정해야 좋을 지 알지 못하므로 난수 사용
  - 물론 난수로 설정했기 때문에 성능이 형편없는 신경망 일 것임
- (03행) 손실 함수 J를 이용해 현재 매개변수 값 W의 성능을 측정함.
  - 손실 함수는 여러 행태가 있는데 기본적으로 신경망이 범하는 오류와 비례 관계임
- (04행) 현재 매개변수 값의 성능, 즉 J(w)가 충분히 낮으면 루프를 탈출, 그렇지 않으면
- (05행)으로 가서 손실 함수 값이 낮아지는, 즉 신경망이 오류를 덜 범하는 방향을 탐색
- (06행) 탐색을 반영하여 매개변수 값을 갱신
- (07행) 나중에 사용할 목적으로 매개변수 값을 저장

4.3 사람의 학습과 신경망의 학습

4.3.2 신경망의 학습 알고리즘

- 신경망을 학습하는 일은 전형적인 최적화 문제optimization problem임
  - [알고리즘 4-3]은 전형적인 최적화 알고리즘임
- 최적화 알고리즘은 매개변수에 대해 최적의 값을 알아내야 하는데, [알고리즘4-3]의 경우 매개변수는 신경망의 가중치임
  - 퍼셉트론의 경우 $w_0, w_1, \cdots, w_d$ 가 매개변수임

[알고리즘 4-3] 신경망의 학습

입력: 훈련 데이터

출력: 최적의 매개변수 값

```
01. 난수로 매개변수 벡터 w를 초기화한다.    # w는 신경망의 가중치([그림 4-2(a)]의 (w0,w1,⋯,wd))
02. while (true)
03.     w에 따라 데이터를 인식하고 손실 함수 J(w)를 계산한다.
04.     if (J(w)가 만족스러움) break
05.     손실 함수 값을 낮추는 방향 Δw를 계산한다.
06.     w=w+Δw
07. w를 저장한다.
```

4.3 사람의 학습과 신경망의 학습

4.3.2 신경망의 학습 알고리즘

- [알고리즘 4-3]을 구현하려면 몇 가지를 구체화 해야함
  - (01행) 매개변수 초기화 방법
  - (03행) 손실 함수 정의
  - (04행) 학습 종료를 위한 조건
  - (05행) 손실 함수의 값을 낮추는, 즉 신경망의 오류를 낮추는 방향을 찾는 것** (매우 중요)
    - 신경망은 이 방향을 찾기 위해 미분을 사용함

[알고리즘 4-3] 신경망의 학습

입력: 훈련 데이터

출력: 최적의 매개변수 값

01. 난수로 매개변수 벡터 **w**를 초기화한다. # **w**는 신경망의 가중치([그림 4-2(a)]의 $(w_0, w_1, \cdots, w_d)$)
02. while (true)
03. **w**에 따라 데이터를 인식하고 손실 함수 $J(\mathbf{w})$를 계산한다.
04. if ($J(\mathbf{w})$가 만족스러움) break
05. 손실 함수 값을 낮추는 방향 $\Delta\mathbf{w}$를 계산한다.
06. $\mathbf{w}=\mathbf{w}+\Delta\mathbf{w}$
07. **w**를 저장한다.

4.6 퍼셉트론 프로그래밍

4.6.1 OR 데이터 인식

- 〉〉 실습 4-1. OR 데이터에 퍼셉트론 적용

```
1  from sklearn.linear_model import Perceptron
2
3  # 훈련 집합 구축
4  X = [[0,0],[0,1],[1,0],[1,1]]
5  y = [-1,1,1,1]
6
7  # fit 함수로 Perceptron 학습
8  p = Perceptron()
9  p.fit(X,y)
10
11 print("학습된 퍼셉트론의 매개변수: ",p.coef_, p.intercept_)
12 print("훈련집합에 대한 예측: ",p.predict(X))
13 print("정확률 측정: ",p.score(X,y)*100, "%")
```

```
학습된 퍼셉트론의 매개변수:  [[2. 2.]] [-1.]
훈련집합에 대한 예측:  [-1  1  1  1]
정확률 측정:  100.0 %
```

04행-05행 훈련 데이터셋 만들기

08행 Perceptron 함수를 호출해 객체 P를 생성

09행 fit 함수를 이용해 데이터 X와 y에 대한 학습 수행

11행 퍼셉트론의 매개변수 가중치와 바이어스 출력

12행 predict 함수를 이용해 훈련집합 X를 테스트용으로 간주하고 예측 수행

13행 score함수는 X를 퍼셉트론으로 예측한 값과 y 레이블을 비교해 맞힌 샘플 개수를 세어 정확률을 계산

https://www.kaggle.com/yukyungchoi/2023-ai-w3-p1

4.6 퍼셉트론 프로그래밍

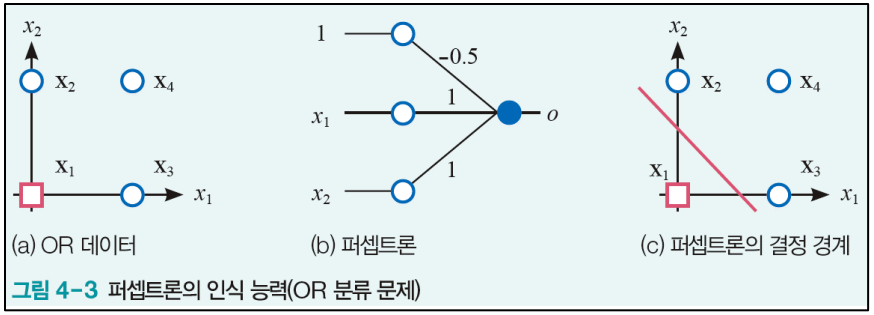


그림 4-3 퍼셉트론의 인식 능력(OR 분류 문제)

4.6.1 OR 데이터 인식

- 〉〉 실습 4-1. OR 데이터에 퍼셉트론 적용

```python
from sklearn.linear_model import Perceptron

# 훈련 집합 구축
X = [[0,0],[0,1],[1,0],[1,1]]
y = [-1,1,1,1]

# fit 함수로 Perceptron 학습
p = Perceptron()
p.fit(X,y)

print("학습된 퍼셉트론의 매개변수: ",p.coef_, p.intercept_)
print("훈련집합에 대한 예측: ",p.predict(X))
print("정확률 측정: ",p.score(X,y)*100, "%")
```

```
학습된 퍼셉트론의 매개변수:  [[2. 2.]] [-1.]
훈련집합에 대한 예측:  [-1  1  1  1]
정확률 측정:  100.0 %
```

직선 방정식의 계수에 상수를 곱해도 같은 직선이므로
퍼셉트론 학습결과로 얻어진 결정 경계 $2*x_1+2*x_2-1=0$와
그림4-3의 결정 경계 $1*x_1+1*x_2-0.5=0$ 는 동일함을 알 수 있음

https://www.kaggle.com/yukyungchoi/2023-ai-w3-p1

4.6 퍼셉트론 프로그래밍

4.6.1 필기 숫자 데이터 인식

- 〉〉 실습 4-2. 필기 숫자 데이터에 퍼셉트론 적용

```
1  from sklearn import datasets
2  from sklearn.linear_model import Perceptron
3  from sklearn.model_selection import train_test_split
4  import numpy as np
5
6  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
7  np.random.seed(0)
8  digit = datasets.load_digits()
9  x_train, x_test, y_train, y_test = train_test_split(digit.data, digit.target, train_size=0.6)
10
11 # fit 함수로 Perceptron 학습
12 p = Perceptron(max_iter=100, eta0=0.001, verbose=0)
13 # digit 데이터로 모델링
14 p.fit(x_train,y_train)
15 # 테스트 집합으로 예측
16 res = p.predict(x_test)
17
18 # 혼동 행렬
19 conf = np.zeros((10,10))
20 for i in range(len(res)):
21     conf[res[i]][y_test[i]]+=1
22 print(conf)
23
24 # 정확률 계산
25 no_correct=0
26 for i in range(10):
27     no_correct+=conf[i][i]
28 accuracy=no_correct/len(res)
29 print("테스트 집합에 대한 정확률은 ",accuracy*100, "%입니다.")
```

02행 sklean의 linear_model 클래스가 제공하는 Perceptron 함수를 불러옴

09행 train_test_split 함수로 digit 데이터를 훈련 집합과 테스트 집합으로 분할

12행 Perceptron 객체를 생성해 p에 저장

14행 훈련 집합으로 퍼셉트로을 학습

16행 테스트 집합에 대해 예측을 수행

19행-22행 혼동 행렬을 계산

25행-28행 정확률을 계산

```
[[60.  0.  0.  0.  0.  0.  1.  0.  0.  1.]
 [ 0. 70.  5.  0.  2.  2.  1.  2.  4.  3.]
 [ 0.  0. 64.  1.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0. 54.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0. 60.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  3.  0. 86.  2.  0.  0.  3.]
 [ 0.  0.  0.  0.  0.  1. 71.  0.  0.  0.]
 [ 0.  0.  0.  1.  1.  0.  0. 62.  0.  0.]
 [ 0.  3.  2. 10.  0.  0.  1.  0. 72. 13.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0. 54.]]
테스트 집합에 대한 정확률은  90.82058414464534 %입니다.
```

https://www.kaggle.com/yukyungchoi/2023-ai-w3-p1

4.6 퍼셉트론 프로그래밍

기계학습의 디자인 패턴을 기억하자

4.6.1 필기 숫자 데이터 인식

- 〉〉 실습 4-2. 필기 숫자 데이터에 퍼셉트론 적용 vs 실습 3-5

〉〉 실습 4-2. 필기 숫자 데이터에 퍼셉트론 적용

```
1  from sklearn import datasets
2  from sklearn.linear_model import Perceptron
3  from sklearn.model_selection import train_test_split
4  import numpy as np
5
6  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
7  np.random.seed(0)
8  digit = datasets.load_digits()
9  x_train, x_test, y_train, y_test = train_test_split(digit.data, digit.target, train_size=0.6)
10
11 # fit 함수로 Perceptron 학습
12 p = Perceptron(max_iter=100, eta0=0.001, verbose=0)
13 # digit 데이터로 모델링
14 p.fit(x_train,y_train)
15 # 테스트 집합으로 예측
16 res = p.predict(x_test)
17
18 # 혼동 행렬
19 conf = np.zeros((10,10))
20 for i in range(len(res)):
21     conf[res[i]][y_test[i]]+=1
22 print(conf)
23
24 # 정확률 계산
25 no_correct=0
26 for i in range(10):
27     no_correct+=conf[i][i]
28 accuracy=no_correct/len(res)
29 print("테스트 집합에 대한 정확률은 ",accuracy*100, "%입니다.")
```

02행 sklean의 linear_model 클래스가 제공하는 Perceptron 함수를 불러옴

09행 train_test_split 함수로 digit 데이터를 훈련 집합과 테스트 집합으로 분할

12행 Perceptron 객체를 생성해 p에 저장

14행 훈련 집합으로 퍼셉트로을 학습

16행 테스트 집합에 대해 예측을 수행

19행-22행 혼동 행렬을 계산

25행-28행 정확률을 계산

```
[[60.  0.  0.  0.  0.  0.  1.  0.  0.  1.]
 [ 0. 70.  5.  0.  2.  2.  1.  2.  4.  3.]
 [ 0.  0. 64.  1.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0. 54.  0.  0.  0.  0.  1.  0.]
 [ 0.  0.  0.  0. 60.  0.  0.  1.  0.  0.]
 [ 0.  0.  0.  3.  0. 86.  2.  0.  0.  3.]
 [ 0.  0.  0.  0.  0.  1. 71.  0.  0.  0.]
 [ 0.  0.  0.  1.  1.  0.  0. 62.  0.  0.]
 [ 0.  3.  2. 10.  0.  0.  1.  0. 72. 13.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0. 54.]]
테스트 집합에 대한 정확률은  90.82058414464534 %입니다.
```

4.6 퍼셉트론 프로그래밍

기계학습의 디자인 패턴을 기억하자

4.6.1 필기 숫자 데이터 인식

- 〉〉 실습 4-2. 필기 숫자 데이터에 퍼셉트론 적용 vs 실습 3-5

〉〉 실습 3-5. 필기 숫자 인식-훈련 집합으로 학습하고 테스트 집합으로 성능 측정 (혼동 행렬, 정확률)

```
1  from sklearn import datasets
2  from sklearn import svm
3  from sklearn.model_selection import train_test_split
4  import numpy as np
5
6
7  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
8  np.random.seed(0)
9  digit = datasets.load_digits()
10 x_train, x_test, y_train, y_test = train_test_split(digit.data, digit.target, train_size=0.6)
11
12
13 # svm의 분류 모델 SVC를 학습
14 s=svm.SVC(gamma=0.001)
15 s.fit(x_train, y_train)
16 res = s.predict(x_test)
17
18
19 # 혼동 행렬 구함
20 conf=np.zeros((10,10))
21 for i in range(len(res)):
22     conf[res[i]][y_test[i]]+=1
23 print(conf)
24
25
26 # 정확률 측정하고 출력
27 no_correct=0
28 for i in range(10):
29     no_correct+=conf[i][i]
30 accuracy=no_correct/len(res)
31 print("테스트 집합에 대한 정확률은",accuracy*100, "%입니다.")
```

03행 model_selection 클래스가 제공하는 train_test_split 함수를 불러옴

08행 난수 생성 고정을 위해 사용 (train_test_split 함수가 난수를 사용)

10행 train_test_split함수를 이용하여 digit 데이터를 훈련 집합과 테스트 집합으로 분할. 첫 번째와 두 번째 매개변수는 분할할 데이터의 특징 벡터와 레이블이며, 세 번째 매개변수는 훈련 집합의 비율

14~15행 훈련 집합으로 SVM을 학습

16행 테스트 집합으로 예측을 수행

20~22행 혼동 행렬 계산

27~30행 정확률을 계산

```
[[60.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 73.  1.  0.  0.  0.  0.  0.  1.  0.]
 [ 0.  0. 69.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 70.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 63.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 87.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  1. 76.  0.  0.  0.]
 [ 0.  0.  1.  0.  0.  0.  0. 65.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 77.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0. 74.]]
테스트 집합에 대한 정확률은 99.30458970792768 %입니다.
```

4.6 퍼셉트론 프로그래밍

4.6.1 필기 숫자 데이터 인식

- 〉〉 실습 4-2. 필기 숫자 데이터에 퍼셉트론 적용 vs. 실습 3-5
  - 퍼셉트론 정확률:90.82%
  - SVM 정확률 : 99.30%

➔ 퍼셉트론은 1950년에 개발된 원시적인 신경망이고 SVM은 1990년대에 다층 퍼셉트론을 앞지른 발전된 모델이기 때문에 SVM 성능이 더욱 좋음

4.4 퍼셉트론 학습 알고리즘

퍼셉트론을 학습하려면 손실 함수loss function J를 설계해야 하며, 손실 함수의 값을 낮추는 방향을 찾는 방법을 고안해야 함

4.4.1 손실 함수 설계

- 손실 함수 J가 만족해야 할 조건
  - (1) W가 훈련 집합에 있는 샘플을 모두 맞히면, 즉 정확률 100%면 J(W)=0 임
  - (2) W가 틀리는 샘플이 많을 수록 J(w)의 값이 큼

4.4 퍼셉트론 학습 알고리즘

4.4.1 손실 함수 설계 : 직관적으로 이해하기 쉬운 손실 함수 예시

$$J(\mathbf{w}) = \sum_{\mathbf{x}\in I} -y\left(\mathbf{w}\mathbf{x}^{\mathrm{T}}\right) \qquad (4.5)$$

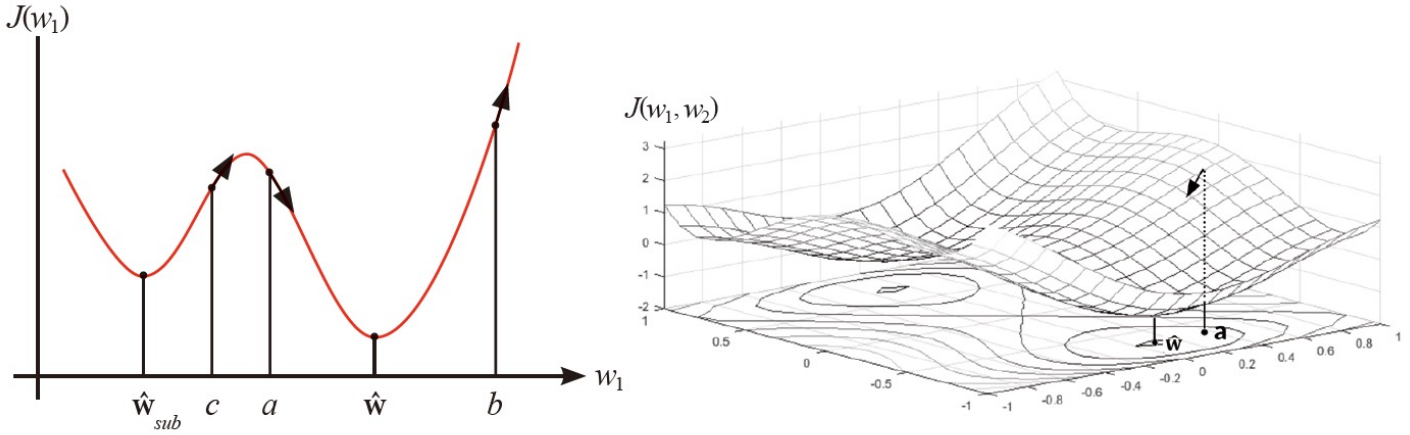
- I 는 **w** 가 틀리는 샘플의 집합이며, **x** 가 틀린 샘플이라고 가정
- X가 +1 부류에 속하면, 즉 y=1이면 퍼셉트론의 출력 wx^T는 음수임
  - 레이블의 부호(y)와 퍼셉트론이 예측한 결과(wx^T)의 부호가 달라야 틀린 샘플
  - 따라서 두 항을 곱한 값에 음수를 취한 $-y(wx^T)$은 양수가 됨
- 반대로 X가 −1부류에 속하면, 즉 y=−1이면 퍼셉트론의 출력 wx^T는 양수임
  - 따라서 두 항을 곱한 값에 음수를 취한 $-y(wx^T)$은 양수가 됨

➔ 식(4.5)는 손실 함수 두 가지 조건을 모두 만족함

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 경사 하강법gradient descent의 원리
  - 편의상 매개변수가 한 개인 경우와 두 개인 경우를 가지고 설명
  - 학습 알고리즘은 J의 최저점 $\hat{\mathbf{w}}$을 찾아야 함



(a) 매개변수가 1개인 경우: $\mathbf{w}=(w_1)$ (b) 매개변수가 2개인 경우: $\mathbf{w}=(w_1, w_2)$

그림 4-5 경사 하강법

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 경사 하강법gradient descent 을 이용한 학습 규칙 유도

- 매개 변수가 하나인 경우
  - 경사 하강법은 미분을 이용하여 최적해를 찾는 기법
  - 미분값 $\partial J/(\partial w_1)$의 반대 방향이 최적해에 접근하는 방향이므로 현재 w1에 $-\partial J/(\partial w_1)$를 더하면 최적해에 가까워짐 식 (4.6)의 학습 규칙
  - 방향은 알지만 얼만큼 가야하는지에 대한 정보가 없기 때문에 학습률 ρ를 곱하여 조금씩 이동(ρ는 하이퍼 매개변수로서 보통 0.001이나 0.0001처럼 작은 값 사용)

$$w_1 = w_1 + \rho\left(-\frac{\partial J}{\partial w_1}\right) \qquad (4.6)$$

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 경사 하강법gradient descent 을 이용한 학습 규칙 유도

- 매개 변수가 여럿인 경우
  - 편미분으로 구한 그레이디언트를 사용 (매개변수 별로 독립적으로 미분)

$$\left.\begin{aligned} \mathbf{w} &= \mathbf{w} + \rho\left(-\nabla \mathbf{w}\right) \\ \nabla \mathbf{w} &= \left(\frac{\partial J}{\partial w_0}, \frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2}, \cdots, \frac{\partial J}{\partial w_d}\right) \end{aligned}\right\} \qquad (4.7)$$

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 매개 변수가 여럿인 경우
  - 퍼셉트론에 식(4.7) 적용
  - 식 (4.5)를 매개변수 w_i로 편미분하면,

$$\frac{\partial J}{\partial w_i} = \sum_{\mathbf{x} \in I} \frac{\partial \left(-y\left(w_0 + w_1 x_1 + \cdots + w_i x_i + \cdots + w_d x_d\right)\right)}{\partial w_i} = \sum_{\mathbf{x} \in I} -yx_i$$

  - 정리하면,

$$\frac{\partial J}{\partial w_i} = \sum_{\mathbf{x} \in I} -yx_i, \ i = 0, 1, 2, \cdots, d \qquad (4.8)$$

  - 식 (4.8)을 식 (4.7)에 대입하면,

$$\text{퍼셉트론 학습 규칙: } w_i = w_i + \rho \sum_{\mathbf{x} \in I} yx_i, \ i = 0, 1, 2, \cdots, d \qquad (4.9)$$

  - 식 (4.9)를 행렬 형태로 쓰면, 퍼셉트론 학습 알고리즘이 사용하는 핵심 공식

$$\text{퍼셉트론의 학습 규칙(행렬 표기): } \mathbf{w} = \mathbf{w} + \rho \sum_{\mathbf{x} \in I} y\mathbf{x} \qquad (4.10)$$

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 퍼셉트론의 학습 알고리즘
  - 식 (4.10)을 [알고리즘 4-3]에 적용하면 [알고리즘 4-4]
  - 03~08행을 수행하여 훈련 집합에 있는 샘플 전체를 한번 처리하는 일을 세대epoch라 부름

[알고리즘 4-4] 퍼셉트론의 학습

입력: 훈련 집합(n은 샘플의 개수)

출력: 최적의 매개변수 $\hat{\mathbf{w}}$

```
01. 난수로 매개변수 벡터 w를 초기화한다. # w는 퍼셉트론 가중치
02. while (true)
03.     I=[ ]                          # 공집합
04.     for j=1 to n
05.         o=τ(wx_j^T)                # 식 (4.4)를 적용해 인식 수행
06.         if(o≠y_j) I=I∪x_j          # 틀린 샘플을 I에 추가
07.     if(I=공집합) break             # 모든 샘플을 맞히면(손실 함수가 0) 루프를 빠져 나감
08.     w=w+ρΣ_{x∈I} yx                # 식 (4.10)을 적용해 매개변수 갱신
09. ŵ=w
```

  - [알고리즘 4-4]는 데이터가 선형 분리 불가능한 경우 무한 루프
    - 여러 세대에 걸쳐 I의 크기가 줄지 않으면 수렴했다고 판단하고 루프를 빠져나오게 수정

4.4 퍼셉트론 학습 알고리즘

4.4.2 학습 알고리즘 설계

- 퍼셉트론의 학습 알고리즘을 개념적으로 설명
  - 알고리즘은 전방 계산(05행) 오차 계산(06행) 후방 가중치 갱신을 반복(08행)
  - 딥러닝을 포함하여 신경망 학습 알고리즘은 모두 이 절차를 따름(딥러닝은 많은 층을 거쳐 전방 계산과 후방 가중치 갱신을 수행)

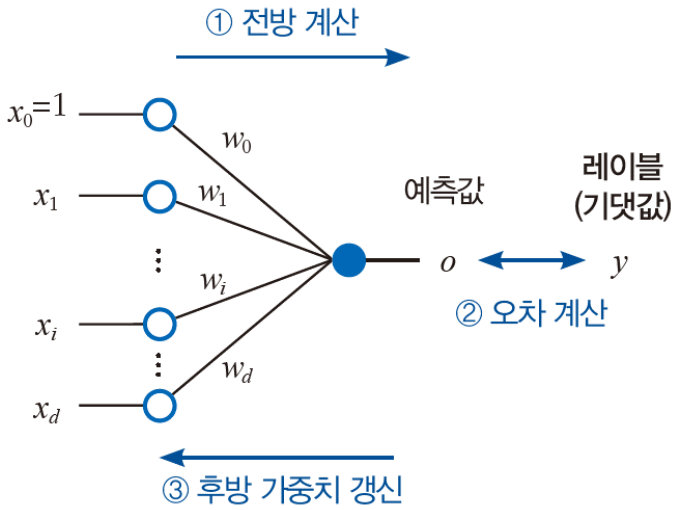


그림 4-6 퍼셉트론 학습 알고리즘의 절차

4.5 현대 기계 학습으로 확장

- 현대 기계 학습에서 가장 강력함 모델인 딥러닝도 퍼셉트론이 복잡해진 것
- 기본 원리는 퍼셉트론과 딥러닝 모두 동일함

4.5 현대 기계 학습으로 확장

4.5.1 현대적인 기계 학습의 복잡도

- 최근 관심 많은 문제
  - 분류 classification

    지정된 몇가지 부류로 구분하는 문제

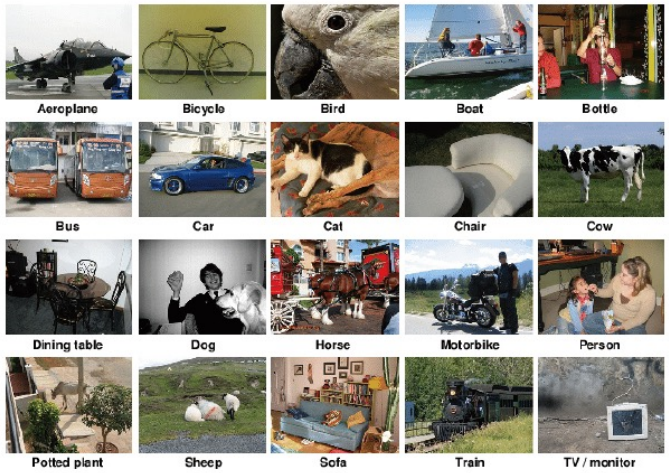
  - 물체 검출 object detection

    : 물체 위치를 바운딩 박스로 찾아내는 문제

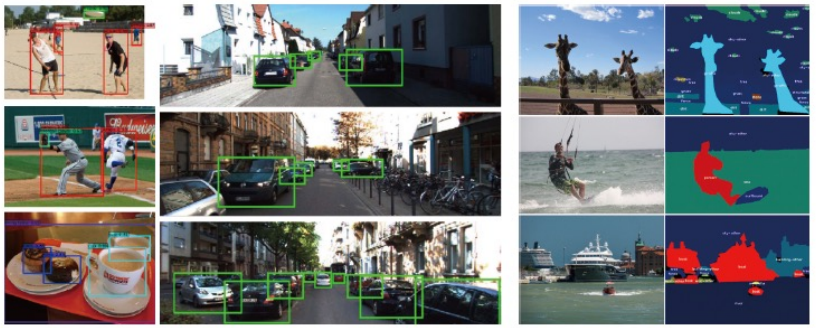
  - 물체 분할 object segmentation

    : 화소 수준으로 물체를 찾아내는 문제

    (추적 대상 물체를 형광색으로 칠하는 응용)



(a) 분류 문제



(b) 검출 문제 (c) 분할 문제

그림 4-7 세 종류의 문제와 데이터셋

4.5 현대 기계 학습으로 확장

4.5.1 현대적인 기계 학습의 복잡도

- 적절한 손실 함수 필요
  - 분류 : 참 값과 예측 값의 차이를 손실 함수에 반영
  - 물체 검출 : 참 바운딩 박스와 예측한 바운딩 박스의 겹침 정도를 손실 함수에 반영
  - 물체 분할 : 화서 별로 레이블이 지정한 물체에 해당하는지 따지는 손실 함수

- 실용적인 신경망의 방대한 매개변수 개수
  - 예) 딥러닝 모델 ResNet50은 50개의 층을 가지며 매개변수(W)는 2천300만개 이상
    - 다행히 1~2차원에서 개발한 공식이 고차원(수십~수천만)에 그대로 적용
  - 매개변수가 많아지면 과잉 적합 가능성이 커지므로, 과잉 적합을 방지하기 위해 더 많은 데이터 사용 또는 더 정교한 규제 기법 적용
  - 계산 시간이 많이 걸리므로 병렬 처리를 위해 GPU 필요

4.5 현대 기계 학습으로 확장

4.5.2 스토케스틱 경사 하강법

- 경사 하강법
  - 자연과학과 공학에서 오랫동안 사용해온 최적화 방법([그림 4-5])
  - 예) 항공공학자
  - 날개를 설계할 때 두께, 폭, 길이, 곡률 등을 매개변수로 설정한 다음 유체 역학 이론을 기반으로 손실 함수 정의
  - 손실 함수는 매개변수 변화에 따른 연료 소비량을 측정
  - 경사 하강법으로 최적해를 구한 다음 날개 제작

4.5 현대 기계 학습으로 확장

4.5.2 스토케스틱 경사 하강법

- 기계 학습의 경사 하강법
  - 여러 측면에서 표준 경사 하강법과 다름
    - 잡음이 섞인 데이터가 개입
    - 매개변수가 방대
    - 일반화 능력 필요

➔ 이런 특성은 최적해를 찾는 일을 어렵게 만듬. 정확률이 등락을 거듭하며 수렴하지 않는다거나 훈련 집합에 대해 높은 성능을 얻었는데 테스트 집합에 대해 형편 없는 성능 등의 문제

4.5 현대 기계 학습으로 확장

4.5.2 스토케스틱 경사 하강법

- 기계 학습은 스토캐스틱 경사 하강법으로 확장하여 사용
- 배치 모드 vs. 패턴 모드
  - [알고리즘 4-4]는 배치 모드: 틀린 샘플을 모은 다음 한꺼번에 매개변수 갱신
  - 패턴 모드는 패턴 별로 매개변수 갱신(세대를 시작할 때 샘플을 뒤섞어 랜덤 샘플링 효과 제공)
- 딥러닝은 주로 미니 배치 사용(패턴 모드와 배치 모드의 중간)
  - 훈련 집합을 일정한 크기의 부분 집합으로 나눈 다음 부분 집합 별로 처리
  - 부분 집합으로 나눌 때 랜덤 샘플링을 적용하기 때문에 스토캐스틱 경사 하강법(SGD, stochastic gradient descent)이라 부름