인공지능과 딥러닝 기초

2023.10.31 ~ 2023.11.01

이 강의 자료는 <파이썬으로 만드는 인공지능>(오일석.이진선 저, 한빛아카데미, 2021)의 내용을 인공지능 심화 교육에 맞게 발췌하고 정리한 것입니다.
전체 내용은 http://cv.jbnu.ac.kr/에서 받아볼 수 있습니다.
강의 동영상은 https://bit.ly/2VJ1sgY에 있습니다.

인공지능의 출현

- 에이다의 통찰
  - … 해석 엔진은 숫자 이외 것도 처리할 수 있을 것이다. … 예를 들어 화음과 음조를 해석 엔진의 표기에 맞출 수 있다면 해석 엔진은 꽤 복잡한 곡을 작곡할 수도 있다. [Ada1843]

- 180여년이 지난 현재 인공지능
  - 알파고가 이세돌을 이김
  - 자율주행차가 도로를 질주
  - 작곡하는 마젠타 프로젝트 [http://magenta.tensorflow.org]-[Try the Demos]-[Magenta.js]

1.4.1 시장을 파고드는 인공지능 제품

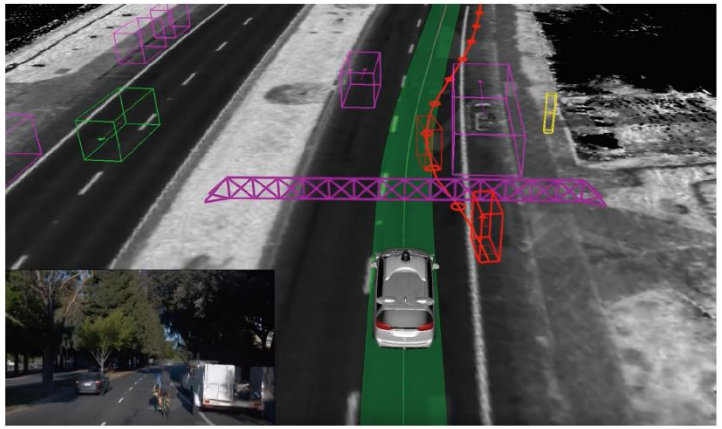


그림 1-3 자율주행

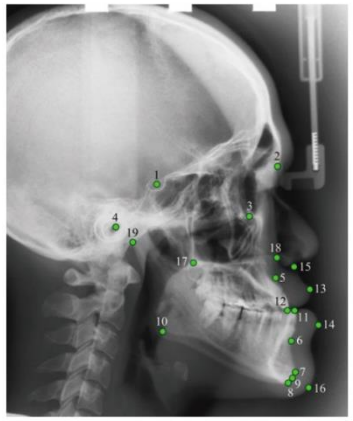


그림 1-4 인공지능 의료(랜드마크 검출)

그림 1-5 인공지능 예술

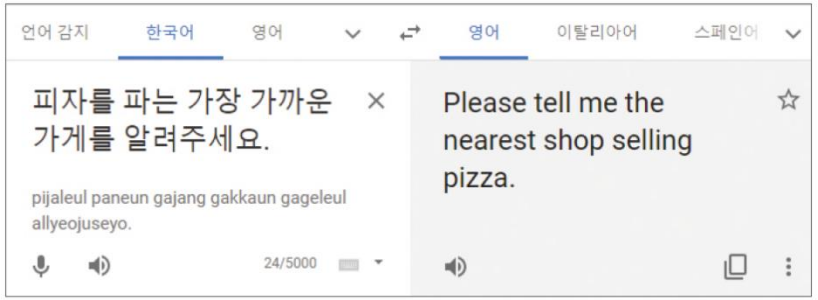
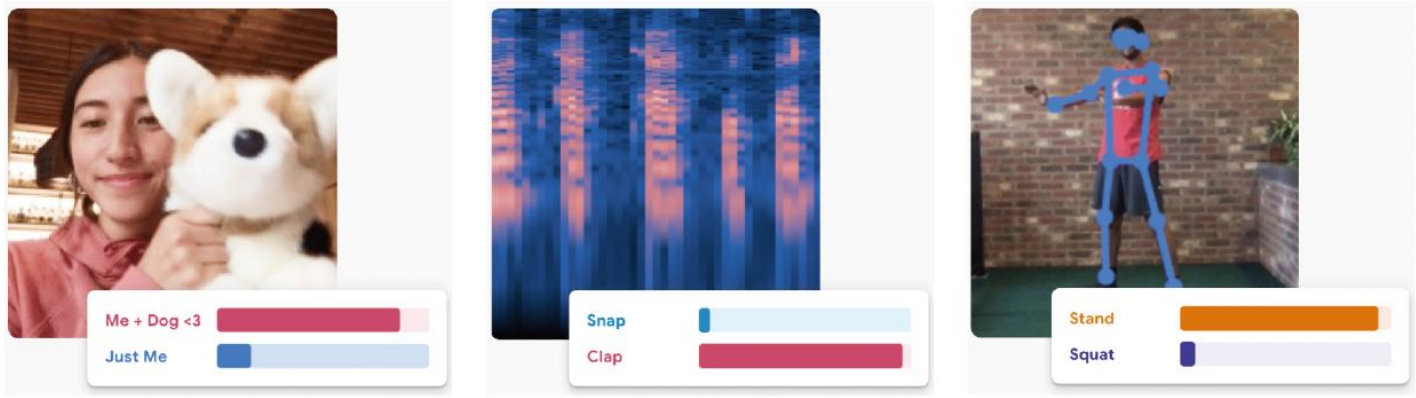


그림 1-6 언어 번역

그림 1-7 자율학습하는 인공지능 게임

1.4.2 대중 속으로 파고드는 DIY 인공지능

- 티처블 머신teachable machine



(a) 물체 인식 (b) 음성 인식 (c) 동작 인식

그림 1-8 티처블 머신으로 제작한 인공지능 응용

1.4.2 대중 속으로 파고드는 DIY 인공지능

- 티처블 머신teachable machine

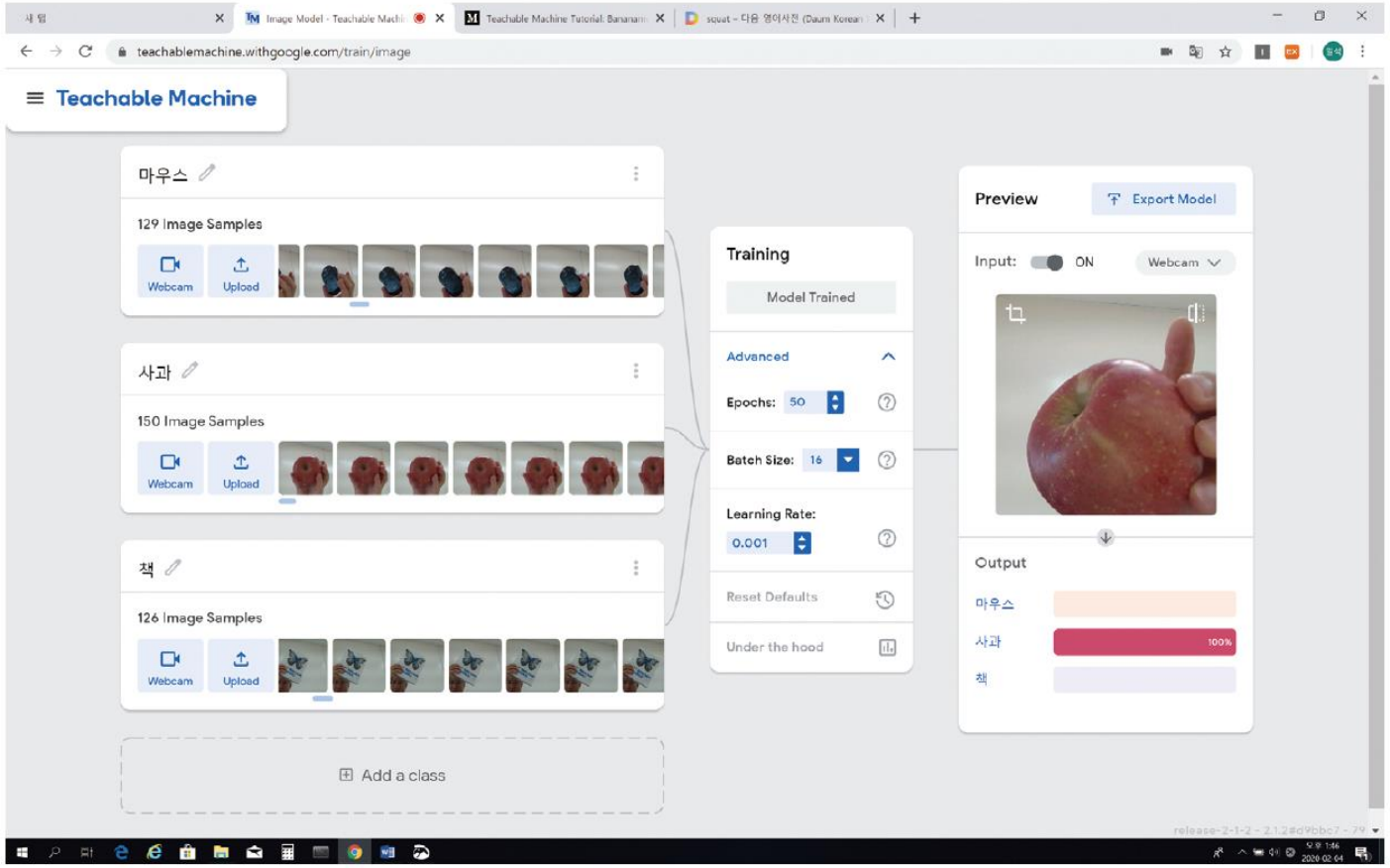


그림 1-9 티처블 머신의 인식 프로그램 제작 과정(데이터 수집 → 학습 → 예측)

1.5.1 지배적인 공학적 관점

- 인공지능 접근 방법
  - 과학적: 인간의 지능을 충분히 연구한 다음 그 원리를 충실히 모방하는 지능 기계 제작
  - 공학적: 쓸만한 지능 기계를 만들 수 있다면 굳이 인간의 지능 원리를 따르지 않아도 됨
    (비행기 날개는 새의 날개를 그대로 모방하지 않는다)

→ 현재는 공학적 접근방법이 지배적임

1.5.2 규칙 기반 방법론 vs. 기계학습 방법론

규칙 기반 방법론

- 사람이 사용하는 규칙을 수집하여 프로그래밍
- 예) 필기 숫자 인식 프로그램
  - 숫자 3은 "왼쪽에서 보면 위와 아래에 터진 골이 있고, 오른쪽에서 보면 둥근 원호가 중간에서 만나고"와 같은 규칙을 수집
- 한계 노출: 다음과 같이 규칙 위반하는 샘플이 꾸준히 발생

기계학습 방법론

- 인공지능 초반에는 규칙 기반이 대세였으나 1990년부터 기계학습으로 주도권이 이동함
- 충분한 데이터를 수집한 다음 기계학습 모델을 학습하는 방법(데이터-주도 패러다임)

예) 필기 숫자 인식을 위한 MNIST 데이터셋

1.5.3 파이썬 프로그래밍

C와 파이썬

- 파이썬은 벡터와 행렬 처리를 코딩하는데 편리한 언어

[C 코드]

```
int a[5]={2,3,1,5,4}
int b[5]={1,2,6,5,7}
int c[5];
for(int i=0; i<5; i++)
   c[i]=a[i]+b[i];
```

[파이썬 코드]

```
import numpy as np
a=np.array([2,3,1,5,4])
b=np.array([1,2,6,5,7])
c=a+b
```

- 기계 학습은 벡터와 행렬 처리를 많이 수행하므로 파이썬을 주로 사용
- 핵심 라이브러리는 효율성때문에 C로 코딩 되어 있음
  (파이썬은 이들 라이브러리를 호출해 사용하는 인터페이스 언어로 사용됨)

2.7.1 인공지능 개발에 많이 쓰는 라이브러리

기초 라이브러리

- 넘파이(Numpy): 다차원 배열 지원(부록 A)
- 맷플롯립(Matplotlib): 데이터 시각화(부록 B)

> TIP 예제와 함께 PDF 파일로 제공되는 부록 A에서 간단하게 넘파이 사용법을 소개하니 넘파이에 익숙하지 않다면 3장으로 넘어가기 전에 꼭 공부하기 바란다.

> TIP PDF 파일로 제공되는 부록 B에서 간단하게 맷플롯립 사용법을 소개한다. 맷플롯립에 익숙하지 않다면 3장으로 넘어가기 전에 꼭 공부하기 바란다.

인공지능 라이브러리

- 사이킷런(Scikit-learn): 고전적인 기계 학습 지원
- 텐서플로(TensorFlow): 딥러닝 지원(이 책이 사용하는 라이브러리)
- 케라스(Keras): 텐서플로를 한 단계 추상화한 라이브러리(이 책이 사용하는 라이브러리)
- 파이토치(PyTorch): 딥러닝 라이브러리

기계학습과 인식

- 사람은 끊임없이 주위 환경을 인식
  - 타인이 말한 소리, 얼굴, 감정 등을 인식
  - 생존과 자기 발전에 필수

- 주위 환경을 인식하는 인공지능
  - 자율 주행차, 음성인식 챗봇, 주문 받는 로봇 등
  - 이 장에서는 인식 프로그램을 만드는데 필요한 기계 학습의 기초 지식을 공부

- 인공지능, 기계 학습, 신경망, 딥러닝의 관계

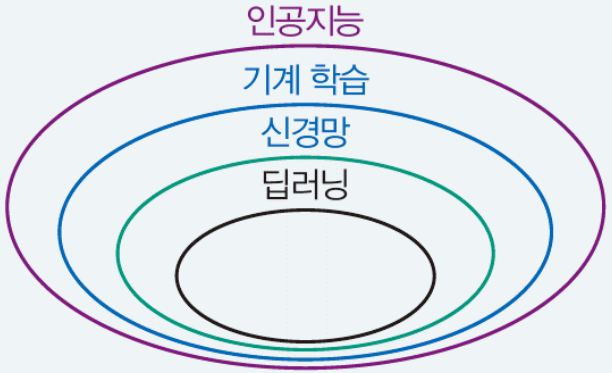


그림 3-1 인공지능, 기계 학습, 신경망, 딥러닝의 관계

3.1 기계학습 기초

- 기계학습에서 데이터의 중요성
  - 에너지를 만드는 연료에 해당
  - 데이터가 없으면 기계학습 적용이 불가능

- 가장 단순한 iris 데이터로 시작해보자.

3.1.1 데이터셋 읽기

- 사이킷런(scikit-learn) 라이브러리 설치
  - pip install scikit-learn 명령어로 라이브러리 설치

TIP https://scikit-learn.org/dev/_downloads/scikit-learn-docs.pdf에 접속하면 가장 최신 버전의 사이킷 런 사용 설명서를 무료로 다운로드할 수 있다. 무려 2,500여 쪽에 달하는 방대한 문서다. 그렇다고 겁먹을 필요는 없다. 필요한 부분을 선택적으로 참조하면 된다.

3.1.1 iris 데이터셋 읽기

[프로그램 3-1(a)]: iris 데이터셋 읽기

프로그램 3-1(a) iris 데이터셋 읽기

```
01  from sklearn import datasets
02
03  d=datasets.load_iris()     # iris 데이터셋을 읽고
04  print(d.DESCR)             # 내용을 출력
```

- 01행: sklearn 모듈의 datasets 클래스를 불러옴
- 03행: load_iris 함수를 호출해 iris 데이터셋을 읽어 객체 d에 저장
- 04행: 객체 d의 DESCR 변수를 출력

기계 학습의 용어

- 샘플로 구성되는 데이터셋
- 특징으로 구성되는 특징 벡터(feature vector)
- 부류(class)

TIP 기계 학습이 사용하는 데이터는 여러 개의 샘플을 담고 있어서 데이터셋(data set)이라 부르기도 한다. 이 책에서는 데이터와 데이터셋을 엄밀히 구분하지 않고 함께 사용하는데, 데이터셋은 iris처럼 특정한 데이터를 가리킬 때 주로 사용한다.

3.1.1 iris 데이터셋 읽기

```
Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

:Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988
...
```

150개의 샘플

네 개의 특징(feature)

세 개의 부류

그림 3-2 iris의 세 가지 품종(왼쪽부터 Setosa, Versicolor, Virginica)

3.1.1 iris 데이터셋 읽기

- iris의 내용 살피기

프로그램 3-1(b) iris의 내용 살펴보기

```
05  for i in range(0,len(d.data)):          # 샘플을 순서대로 출력
06      print(i+1,d.data[i],d.target[i])
```

```
1 [5.1 3.5 1.4 0.2] 0
2 [4.9 3.  1.4 0.2] 0
3 [4.7 3.2 1.3 0.2] 0
4 [4.6 3.1 1.5 0.2] 0
...
51 [7.  3.2 4.7 1.4] 1
52 [6.4 3.2 4.5 1.5] 1
53 [6.9 3.1 4.9 1.5] 1
54 [5.5 2.3 4.  1.3] 1
...
101 [6.3 3.3 6.  2.5] 2
102 [5.8 2.7 5.1 1.9] 2
103 [7.1 3.  5.9 2.1] 2
104 [6.3 2.9 5.6 1.8] 2
...
```

d.target(레이블)

d.data(특징 벡터)

3.1.1 기계 학습에서 데이터셋의 표현

- 샘플을 특징 벡터와 레이블로 표현
  - 특징 벡터는 **x**로 표기(d는 특징의 개수로서 특징 벡터의 차원이라 부름)

$$\text{특징 벡터: } \mathbf{x}=(x_1,x_2,\cdots,x_d) \qquad (3.1)$$

  - 레이블은 0,1,2,...,c-1의 값 또는 1,2,...,c-1,c의 값 또는 원핫 코드
    - 원핫 코드는 한 요소만 1인 이진열
      - 예) Setosa는 (1,0,0), Versicolor는 (0,1,0), Virginica는 (0,0,1)로 표현

	특징 벡터 $\mathbf{x}=(x_1, x_2,\cdots, x_d)$	레이블(참값) y
샘플 1:	(5.1, 3.5, 1.4, 0.2)	0
샘플 2:	(4.9, 3.0, 1.4, 0.2)	0
...	...	...
샘플 51:	(7.0, 3.2, 4.7, 1.4)	1
샘플 52:	(6.4, 3.2, 4.5, 1.5)	1
...	...	...
샘플 101:	(6.3, 3.3, 6.0, 2.5)	2
샘플 102:	(5.8, 2.7, 5.1, 1.9)	2
...	...	...
샘플 n:	(5.9, 3.0, 5.1, 1.8)	2

iris 데이터셋 (n=150, d=4)

그림 3-3 일반적인 데이터셋 표현 방법(iris 데이터셋 예시)

3.1.2 기계 학습 적용: 모델링과 예측

- [프로그램 3-1(c)]
  - SVM(support vector machine)이라는 기계 학습 모델을 사용

프로그램 3-1(c) iris에 기계 학습 적용: 모델링과 예측

```
07  from sklearn import svm
08
09  s=svm.SVC(gamma=0.1,C=10)                  # svm 분류 모델 SVC 객체 생성하고
10  s.fit(d.data,d.target)                     # iris 데이터로 학습
11
12  new_d=[[6.4,3.2,6.0,2.5],[7.1,3.1,4.7,1.35]]   # 101번째와 51번째 샘플을 변형하여
                                                    새로운 데이터 생성
13  res=s.predict(new_d)
14  print("새로운 2개 샘플의 부류는", res)
```

하이퍼 매개변수

훈련 집합

테스트 집합

```
새로운 2개 샘플의 부류는 [2 1]
```

  - 09행: SVM의 분류기 모델 SVC 클래스의 객체를 생성하여 s에 저장
  - 10행: 객체 s의 fit 함수는 훈련 집합을 가지고 학습을 수행(매개변수로 특징 벡터 iris.data와 레이블 iris,target을 설정)
  - 13행: 객체 s의 predict 함수는 테스트 집합을 가지고 예측 수행

3.1.2 기계 학습 적용: 모델링과 예측

훈련 집합과 테스트 집합

- 훈련 집합: 기계 학습 모델을 학습하는데 쓰는 데이터로서 특징 벡터와 레이블 정보를 모두 제공
- 테스트 집합: 학습을 마친 모델의 성능을 측정하는데 쓰는 데이터로서 예측할 때는 특징 벡터 정보만 제공하고, 예측 결과를 가지고 정확률을 측정할 때 레이블 정보를 사용

NOTE 하이퍼 매개변수 설정

하이퍼 매개변수hyper parameter란 모델의 동작을 제어하는 데 쓰는 변수이다. 모델의 학습을 시작하기 전에 설정해야 하는데, 적절한 값으로 설정해야 좋은 성능을 얻을 수 있다. 최적의 하이퍼 매개변수 값을 자동으로 설정하는 일을 하이퍼 매개변수 최적화(hyper parameter optimization)라 하는데, 이것은 기계 학습의 중요한 주제 중 하나다. 하이퍼 매개변수 최적화는 4.10절에서 다룬다.

3.2.1 인공지능 설계 사례: 과일 등급을 분류하는 기계

- 사과를 상중하의 세 부류로 분류하는 인공지능 기계의 설계

1. 데이터 확보
   - 상중하 비율이 비슷하게 수천 개의 사과 수집(데이터 편향$_{\text{data bias}}$을 방지하기 위해 여러 농장에서 수집)
   - 카메라로 촬영하여 파일에 저장
2. 특징 벡터와 레이블 준비
   - 어떤 특징을 사용할까? 예) 사과의 크기, 색깔, 표면의 균일도는 분별력이 높은 특징
   - 컴퓨터 비전 기술로 특징 추출 프로그램 작성. 특징 추출하여 apple.data 파일에 저장
   - 사과 분류 전문가를 고용하여 레이블링. apple.target 파일에 저장
3. 학습하는 과정을 프로그래밍(훈련 데이터 사용)

```
from sklearn import svm
s=svm.SVC(gamma=0.1,C=10)
s.fit(apple.data, apple.target)  # apple 데이터로 모델링
```

4. 예측 과정을 프로그래밍(새로 수집한 테스트 데이터 사용)

```
s.predict(x)  # 새로운 사과에서 추출한 특징 벡터 x를 예측
```

3.2.2 규칙 기반 vs. 고전적 기계 학습 vs. 딥러닝

규칙 기반 방법

- 분류하는 규칙을 사람이 구현.
  예)"꽃잎의 길이가 a보다 크고, 꽃잎의 너비가 b보다 작으면 Setosa"라는 규칙에서 a와 b를 사람이 결정해 줌
- 큰 데이터셋에서는 불가능하고, 데이터가 바뀌면 처음부터 새로 작업해야 하는 비효율성

기계 학습 방법

- 특징 벡터를 추출하고 레이블을 붙이는 과정은 규칙 기반과 동일(수작업 특징$_{\text{hand-crafted feature}}$)
- 규칙 만드는 일은 기계학습 모델을 이용하여 자동으로 수행(이 책의 3~4장)

딥러닝 방법

- 레이블을 붙이는 과정은 기계 학습과 동일
- 특징 벡터를 학습이 자동으로 알아냄. 특징 학습$_{\text{feature learning}}$ 또는 표현 학습$_{\text{representation learning}}$을 한다고 말함
- 장점
  - 특징 추출과 분류를 동시에 최적화하므로 뛰어난 성능 보장
  - 인공지능 제품 제작이 빠름
- 이 책의 5장 이후

3.3 데이터에 대한 이해

- 이 절은
  - 특징 공간에서 데이터 분포를 살펴보고
  - sklearn 라이브러리가 제공하는 여러 가지 데이터셋을 소개함

3.3.1 특징 공간에서 데이터 분포

iris 데이터

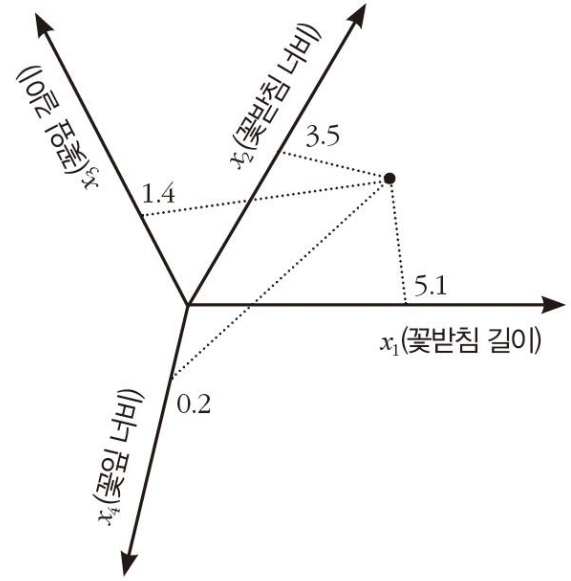
- 특징이 4개이므로 4차원 특징 공간을 형성
- 150개 샘플 각각은 4차원 특징 공간의 한 점
- [프로그램 3-2]는 차원을 하나 제외하고 3차원 공간에 데이터 분포를 그림

프로그램 3-2 iris 데이터의 분포를 특징 공간에 그리기

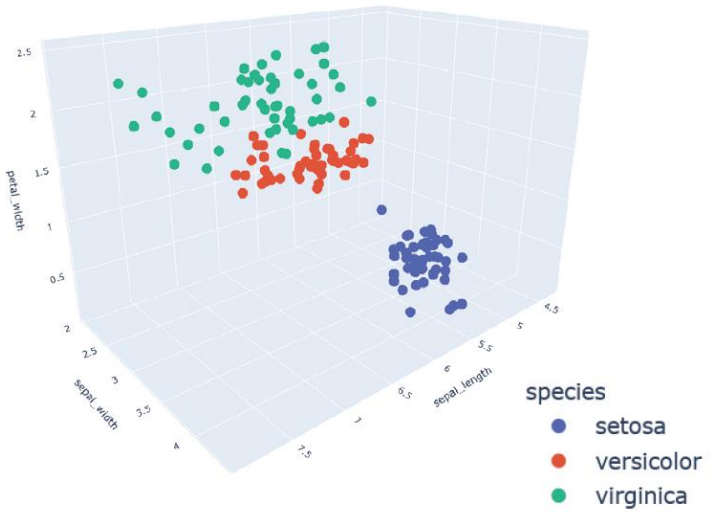
```
01  import plotly.express as px
02
03  df = px.data.iris()
04  fig = px.scatter_3d(df, x='sepal_length', y='sepal_width', z='petal_width',
    color='species')        # petal_length를 제외하여 3차원 공간 구성
05  fig.show(renderer="browser")
```

3.3.1 특징 공간에서 데이터 분포

- 특징 공간에서 데이터 분포 관찰([프로그램 3-2]의 실행 결과인 [그림 3-5(b)])
  - petal width(수직 축)에 대해 Setosa는 아래쪽, Virginica는 위쪽에 분포([프로그램 3-1(b)]의 실행 결과와 일치)
    → petal width 특징은 분별력$_{\text{discriminating power}}$이 뛰어남
  - sepal width 축은 세 부류가 많이 겹쳐서 분별력이 낮음
  - 전체적으로 보면, 세 부류가 3차원 공간에서 서로 다른 영역을 차지하는데 몇 개 샘플은 겹쳐 나타남



(a) 4차원 특징 공간(가상의 그림)



(b) 꽃잎 길이 축을 제외한 3차원 특징 공간

그림 3-5 iris 데이터를 특징 공간에 그리기

3.3.1 특징 공간에서 데이터 분포

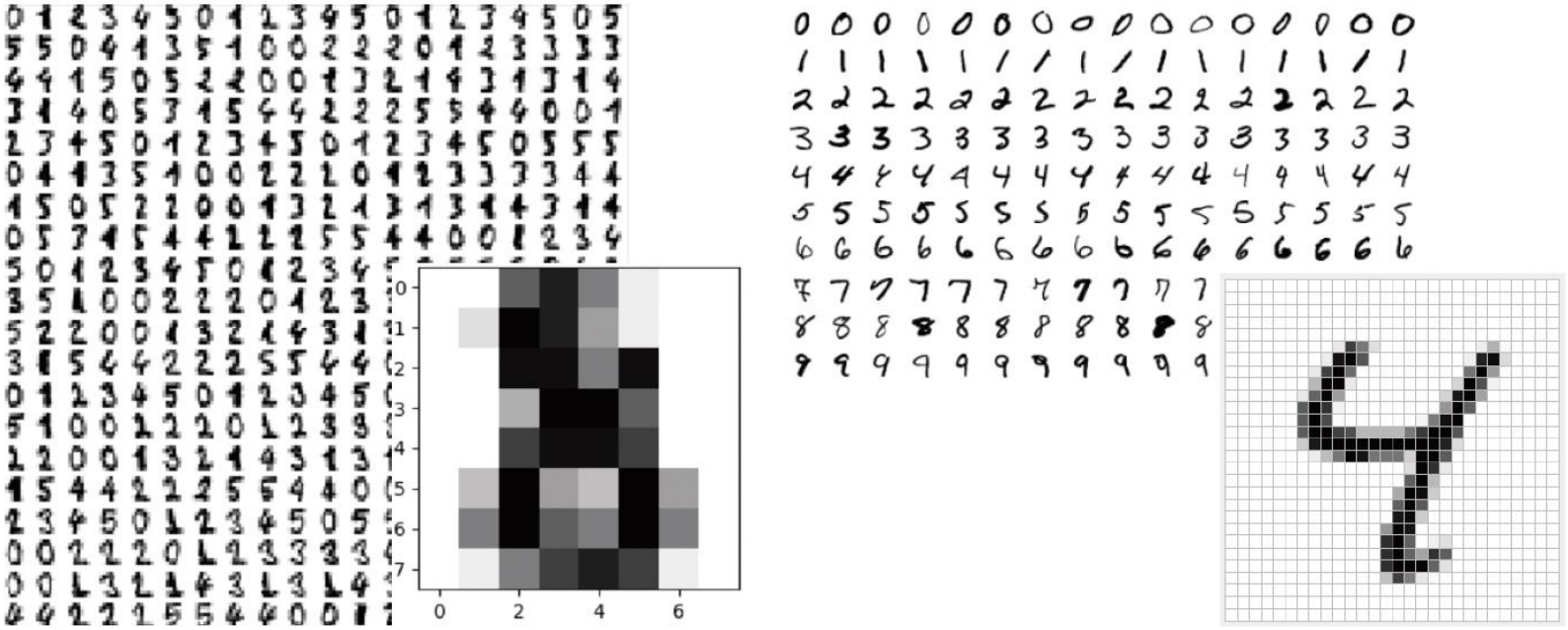
NOTE 다차원 특징 공간

종이에 그릴 수 있는 공간은 3차원으로 제한되지만, 수학은 아주 높은 차원까지 다룰 수 있다. 예를 들어 2차원 상의 두 점 $\mathbf{x}=(x_1,x_2)$와 $\mathbf{y}=(y_1,y_2)$의 거리를 $d(\mathbf{x},\mathbf{y})=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2}$으로 계산할 수 있는데, 4차원 상의 두 점 $\mathbf{x}=(x_1,x_2,x_3,x_4)$와 $\mathbf{y}=(y_1,y_2,y_3,y_4)$의 거리는 $d(\mathbf{x},\mathbf{y})=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+(x_3-y_3)^2+(x_4-y_4)^2}$로 계산할 수 있다.

일반적으로 d차원 상의 두 점의 거리는 $d(\mathbf{x},\mathbf{y})=\sqrt{\sum_{i=1}^{d}(x_i-y_i)^2}$ 로 계산한다. 기계 학습에서는 d=수백~수만에 달하는 매우 고차원 특징 공간의 데이터를 주로 다룬다.

3.3.2 영상 데이터 사례: 필기 숫자

- 두 가지 필기 숫자 데이터셋
  - sklearn 데이터셋: 8*8 맵(64개 화소), 1797개 샘플, [0,16] 명암값
  - MNIST 데이터셋: 28*28맵(784개 화소), 7만개 샘플, [0,255] 명암값



(a) sklearn에서 제공하는 데이터셋 (b) MNIST 데이터셋

그림 3-6 필기 숫자 데이터셋

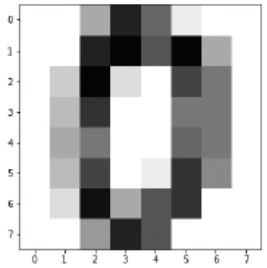
3.3.2 영상 데이터 사례: 필기 숫자

- [프로그램 3-3(a)]
  - matplotlib 라이브러리를 이용한 샘플 디스플레이와 샘플 내용(화소값) 출력

프로그램 3-3(a) 필기 숫자 데이터

```
01  from sklearn import datasets
02  import matplotlib.pyplot as plt
03
04  digit=datasets.load_digits()
05
06  plt.figure(figsize=(5,5))
07  plt.imshow(digit.images[0],cmap=plt.cm.gray_r,interpolation='nearest')
                                                        # 0번 샘플을 그림
08  plt.show()
09  print(digit.data[0])                                # 0번 샘플의 화솟값을 출력
10  print("이 숫자는 ",digit.target[0],"입니다.")
```

3.3.2 영상 데이터 사례: 필기 숫자



```
[ 0.  0.  5. 13.  9.  1.  0.  0.  0.  0. 13. 15. 10. 15.  5.  0.  0.  3.
 15.  2.  0. 11.  8.  0.  0.  4. 12.  0.  0.  8.  8.  0.  0.  5.  8.  0.
  0.  9.  8.  0.  0.  4. 11.  0.  1. 12.  7.  0.  0.  2. 14.  5. 10. 12.
  0.  0.  0.  0.  6. 13. 10.  0.  0.  0.]

이 숫자는 0입니다.
```

NOTE matplotlib을 이용한 시각화

파이썬에서 matplotlib 라이브러리는 시각화에 가장 널리 쓰인다. 인공지능은 학습 과정이나 예측 결과를 시각화하는 데 matplotlib을 자주 사용한다. matplotlib 사용이 처음이라면 부록 B를 공부해 기초를 먼저 다진다. matplotlib의 공식 사이트에서 제공하는 튜토리얼 문서를 공부하는 것도 효과적인 방법이다. [표 2-1]에서 제시한 https://matplotlib.org/users에 접속해 [Tutorials] 메뉴를 선택한다. 튜토리얼은 Introductory, Intermediate, Advanced로 나뉘어 있으니 최소한 Introductory 코스를 숙지하고 넘어간다.

3.3.4 텍스트 데이터 사례: 20newsgroups

- 20newsgroups 데이터셋
  - 웹에서 수집한 문서를 20개 부류로 구분. 텍스트로 구성되어 샘플의 길이가 다름
  - 시계열 데이터(단어가 나타나는 순서가 중요)

프로그램 3-3(c) 20newsgroups 데이터셋

```
21  news=datasets.fetch_20newsgroups(subset='train')  # 데이터셋 읽기
22  print("*****₩n",news.data[0],"₩n*****")           # 0번 샘플 출력
23  print("이 문서의 부류는 <",news.target_names[news.target[0]],"> 입니다.")
```

```
*****
From: lerxst@wam.umd.edu (where's my thing)
Subject: WHAT car is this!?
Nntp-Posting-Host: rac3.wam.umd.edu
Organization: University of Maryland, College Park
Lines: 15

I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
…
*****

이 문서의 부류는 < rec.autos >입니다.
```

3.5 필기 숫자 인식

- 필기 숫자 데이터셋을 가지고 프로그래밍 연습
  - 데이터 수집 – 특징 추출 – 모델링 – 예측 단계로 구성
  - 특징 추출을 위한 코드 작성
  - sklearn이 제공하는 fit 함수로 모델링(학습)
  - predict 함수로 예측

3.5.1 화소 값을 특징으로 사용

- 화소 각각을 특징으로 간주
  - sklearn의 필기 숫자는 8*8 맵으로 표현되므로 64차원 특징 벡터
  - 2차원 구조를 1차원 구조로 변환
  - 예) [프로그램 3-3(a)]의 샘플

```
x=(0,0,5,13,9,1,0,0,0,0,13,15,10,15,5,0,0,3,15,2,0,11,8,0,0,4,12,0,0,8,8,
0,0,5,8,0,0,9,8,0,0,4,11,0,1,12,7,0,0,2,14,5,10,12,0,0,0,0,6,13,10,0,0,0)
```

3.5.1 화소 값을 특징으로 사용

[프로그램 3-1]과 매우 비슷함

- [프로그램 3-4]
  - 07~08행: SVC로 학습 수행
    (특징 벡터 digit.data, 레이블 digit.target 사용)
  - 11~14행: 맨 앞의 세 개 샘플을 테스트 집합으로 간주하고 예측을 해봄
  - 17~20행: 훈련 집합을 테스트 집합으로 간주하고 정확률을 측정

프로그램 3-4 필기 숫자 인식 – 각 화소를 특징으로 간주하여 64차원 특징 벡터 사용

```
01  from sklearn import datasets
02  from sklearn import svm
03
04  digit=datasets.load_digits()
05
06  # svm의 분류기 모델 SC를 학습
07  s=svm.SVC(gamma=0.1,C=10)
08  s.fit(digit.data,digit.target) # digit 데이터로 모델링
09
10  # 훈련 집합의 앞에 있는 샘플 3개를 새로운 샘플로 간주하고 인식해봄
11  new_d=[digit.data[0],digit.data[1],digit.data[2]]
12  res=s.predict(new_d)
13  print("예측값은", res)
14  print("참값은", digit.target[0],digit.target[1],digit.target[2])
15
16  # 훈련 집합을 테스트 집합으로 간주하여 인식해보고 정확률을 측정
17  res=s.predict(digit.data)
18  correct=[i for i in range(len(res)) if res[i]==digit.target[i]]
19  accuracy=len(correct)/len(res)
20  print("화소 특징을 사용했을 때 정확률=",accuracy*100, "%")
```

```
예측값은 [0 1 2]
참값은 0 1 2

화소 특징을 사용했을 때 정확률=100.0%
```

3.6 성능 측정

- 객관적인 성능 측정의 중요성
  - 모델 선택할 때 중요
  - 현장 설치 여부 결정할 때 중요
- 일반화generalization 능력
  - 학습에 사용하지 않았던 새로운 데이터에 대한 성능
  - 가장 확실한 방법은 실제 현장에 설치하고 성능 측정 → 비용 때문에 실제 적용 어려움
  - 주어진 데이터를 분할하여 사용하는 지혜 필요

3.6.1 혼동 행렬과 성능 측정 기준

혼동 행렬confusion matrix

- 부류 별로 옳은 분류와 틀린 분류의 개수를 기록한 행렬
  - n_{ij}는 모델이 i라고 예측했는데 실제 부류는 j인 샘플의 개수

		참값(그라운드 트루스)					
		부류 1	부류 2	…	부류 j	…	부류 c
예측한 부류	부류 1	n_{11}	n_{12}		n_{1j}		n_{1c}
	부류 2	n_{21}	n_{22}		n_{2j}		n_{2c}
	…						
	부류 i	n_{i1}	n_{i2}		n_{ij}		n_{ic}
	…						
	부류 c	n_{c1}	n_{c2}		n_{cj}		n_{cc}

(a) 부류가 c개인 경우

		그라운드 트루스	
		긍정	부정
예측값	긍정	TP	FP
	부정	FN	TN

(b) 부류가 2개인 경우

그림 3-10 혼동 행렬

- 이진 분류에서 긍정positive과 부정negative
  - 검출하고자 하는 것이 긍정(환자가 긍정이고 정상인이 부정, 불량품이 긍정이고 정상이 부정)
- 참 긍정(TP), 거짓 부정(FN), 거짓 긍정(FP), 참 부정(TN)의 네 경우

3.6.1 혼동 행렬과 성능 측정 기준

널리 쓰이는 성능 측정 기준

- 정확률$_{accuracy}$
  - 부류가 불균형일 때 성능을 제대로 반영하지 못함

$$정확률 = \frac{맞힌\ 샘플\ 수}{전체\ 샘플\ 수} = \frac{대각선\ 샘플\ 수}{전체\ 샘플\ 수} \quad (3.2)$$

- 특이도$_{specificity}$와 민감도$_{sensitivity}$ (의료에서 주로 사용)

$$특이도 = \frac{TN}{TN+FP},\ 민감도 = \frac{TP}{TP+FN} \quad (3.3)$$

- 정밀도$_{precision}$와 재현률$_{recall}$ (정보검색에서 주로 사용)

$$정밀도 = \frac{TP}{TP+FP},\ 재현율 = \frac{TP}{TP+FN} \quad (3.4)$$

3.6.2 훈련/검증/테스트 집합으로 쪼개기

- 주어진 데이터를 적절한 비율로 훈련, 검증, 테스트 집합으로 나누어 씀
  - 모델 선택 포함: 훈련/검증/테스트 집합으로 나눔
  - 모델 선택 제외: 훈련/테스트 집합으로 나눔

그림 3-11 훈련/검증/테스트 집합으로 쪼개기

3.6.2 훈련/검증/테스트 집합으로 쪼개기

- [프로그램 3-5]는 모델 선택 제외
  - 08행: train_test_split 함수로 훈련 60%, 테스트 40%로 랜덤 분할
  - 12행: 훈련 집합 x_train, y_train을 fit 함수에 주어 학습 수행
  - 14행: 테스트 집합의 특징 벡터 x_test를 predict 함수에 주어 예측 수행
  - 17~20행: 테스트 집합의 레이블 y_test를 가지고 혼동 행렬 계산

프로그램 3-5 필기 숫자 인식 – 훈련 집합으로 학습하고 테스트 집합으로 성능 측정

```
01  from sklearn import datasets
02  from sklearn import svm
03  from sklearn.model_selection import train_test_split
04  import numpy as np
05
06  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
07  digit=datasets.load_digits()
08  x_train,x_test,y_train,y_test=train_test_split(digit.data,digit.target,train_size=0.6)
09
```

3.6.2 훈련/검증/테스트 집합으로 쪼개기

```
10  # svm의 분류 모델 SVC를 학습
11  s=svm.SVC(gamma=0.001)
12  s.fit(x_train,y_train)
13
14  res=s.predict(x_test)
15
16  # 혼동 행렬 구함
17  conf=np.zeros((10,10))
18  for i in range(len(res)):
19      conf[res[i]][y_test[i]]+=1
20  print(conf)
21
22  # 정확률 측정하고 출력
23  no_correct=0
24  for i in range(10):
25      no_correct+=conf[i][i]
26  accuracy=no_correct/len(res)
27  print("테스트 집합에 대한 정확률은", accuracy*100, "%입니다.")
```

예) 부류 3에 속하는 75개 샘플 중 73개를 3, 1개를 2, 1개를 7로 인식

```
[[76.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 78.  0.  0.  0.  0.  0.  0.  3.  0.]
 [ 0.  0. 66.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 73.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 63.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0. 70.  0.  0.  0.  2.]
 [ 0.  0.  0.  0.  0.  0. 77.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0. 77.  0.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0. 74.  0.]
 [ 0.  0.  0.  0.  0.  1.  0.  0.  0. 56.]]

테스트 집합에 대한 정확률은 98.74826147426981%입니다.
```

3.6.2 훈련/검증/테스트 집합으로 쪼개기

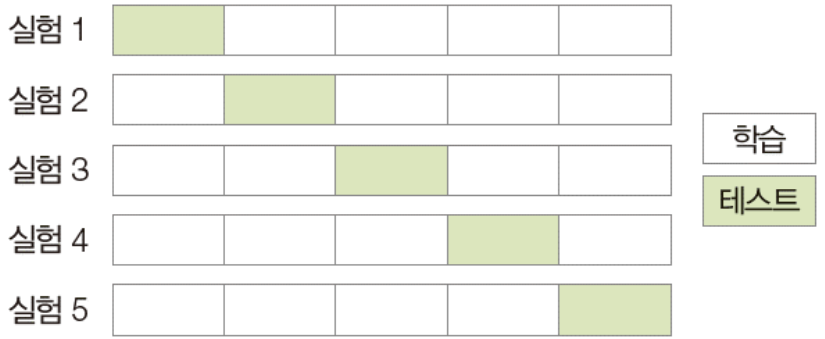
NOTE 난수를 사용하기 때문에 실행할 때마다 다른 결과가 나오는 프로그램

[프로그램 3-5]는 실행할 때마다 출력이 다르게 나온다. 08행의 train_test_split 함수가 난수를 사용해 데이터를 분할하기 때문이다. 앞으로 등장하는 프로그램에서도 난수를 사용하는 경우가 있는데 마찬가지로 실행할 때마다 다른 결과를 얻게 된다. 동일한 실행 결과를 얻으려면 08행 이전에 np.random.seed(0)을 추가하면 된다. 매개변수 0은 다른 값을 사용해도 된다. 어떤 값이든 고정시키면 매번 같은 난수 열이 생성된다.

3.6.3 교차 검증

- 훈련/테스트 집합 나누기의 한계
  - 우연히 높은 정확률 또는 우연히 낮은 정확률 발생 가능성
- k-겹 교차 검증k-fold cross validation
  - 훈련 집합을 k개의 부분집합으로 나누어 사용. 한 개를 남겨두고 k-1개로 학습한 다음 남겨둔 것으로 성능 측정. k개의 성능을 평균하여 신뢰도 높임

(a) 모델 선택 포함



(b) 모델 선택 제외

그림 3-12 k-겹 교차 검증(k=5인 경우)

3.6.3 교차 검증

- [프로그램 3-6]은 digit 데이터에 교차 검증 적용(모델 선택 제외)
  - cross_val_score 함수가 교차 검증 수행해줌(cv=5는 5-겹 교차 검증하라는 뜻)

프로그램 3-6 필기 숫자 인식 – 교차 검증으로 성능 측정

```
01  from sklearn import datasets
02  from sklearn import svm
03  from sklearn.model_selection import cross_val_score
04  import numpy as np
05
06  digit=datasets.load_digits()
07  s=svm.SVC(gamma=0.001)
08  accuracies=cross_val_score(s,digit.data,digit.target,cv=5) # 5-겹 교차 검증
09
10  print(accuracies)
11  print("정확률(평균)=%0.3f, 표준편차=%0.3f"%(accuracies.mean()*100,accuracies.std()))
```

```
[0.97527473 0.95027624 0.98328691 0.99159664 0.95774648]
정확률(평균)=97.164, 표준편차=0.015
```

  - 실행 결과 정확률이 들쭉날쭉. 한번만 시도하는 [프로그램 3-5]의 위험성을 잘 보여줌
  - k를 크게 하면 신뢰도 높아지지만 실행 시간이 더 걸림

신경망 기초

생물 신경망에서 인공 신경망 태동

- 수영 선수의 지식이 팔에 저장되는지 뇌에 저장되는지가 논란거리인 시대가 있었으나, 1900년대 뇌 과학과 신경 과학이 비약적으로 발전되면서 뇌에 저장된다는 사실 밝혀짐
- 1900년대 중반에 뇌의 정보처리 과정을 수학적으로 모델링하려는 연구 그룹 등장
- 인공 신경망ANN(artificial neural network)의 태동 또는 인공지능의 태동

컴퓨터의 등장으로 실제 구현 시도

- 퍼셉트론은 가장 성공한 인공 신경망 모델
- 퍼셉트론은 발전을 거듭해 현재 딥러닝으로 이어짐
- 이 장은 퍼셉트론과 다층 퍼셉트론을 다룸(이들은 얕은 신경망). 5장부터 딥러닝(깊은 신경망)을 다룸

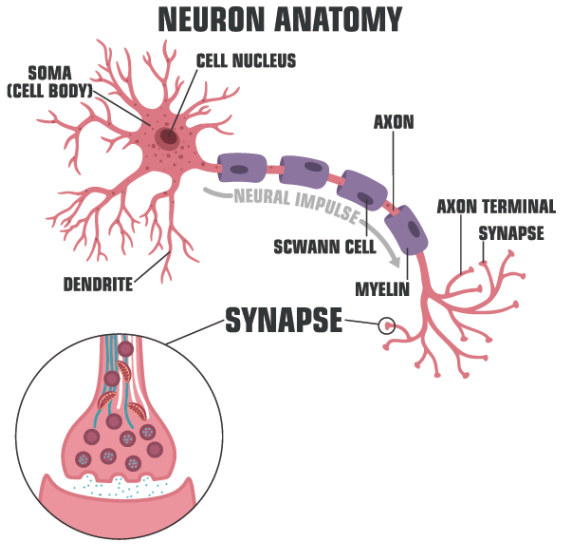
4.1 인공 신경망의 태동

- 생물 신경망을 간략히 소개한 후, 인공 신경망 설명
- 인공 신경망은 생물 신경망에서 영감을 얻었지만 실제 구현은 다름
  - 컴퓨터의 작동 원리가 생물의 작동 원리와 근본적으로 다르기 때문

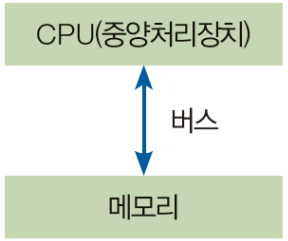
4.1.1 생물 신경망

사람 뇌와 컴퓨터

- 뉴런neuron은 뇌의 정보처리 단위로서
  연산을 수행하는 세포체soma 또는 cell body,
  처리한 정보를 다른 뉴런에 전달하는 축삭axon,
  다른 뉴런으로부터 정보를 받는 수상돌기dendrite로 구성
- 사람 뇌는 10^{11}개 가량의 뉴런, 뉴런마다 1000개 가량의 연결 → 고도의 병렬 처리기
- 반면에 폰 노이만 컴퓨터는 아주 빠른 순차 명령어 처리기



(a) 뇌의 정보처리 단위인 뉴런



(b) 폰 노이만 컴퓨터의 구조

그림 4-1 생물의 정보처리와 컴퓨터의 정보처리의 차이

4.1.2 인공 신경망의 발상과 전개

간략한 인공 신경망 역사

- 1943년 맥컬록과 피츠의 계산 모형
- 1949년 헤브의 학습 알고리즘
- 1958년 로젠블랫의 퍼셉트론
- 위드로와 호프의 아달린과 마달린
- 1960년대의 퍼셉트론에 대한 과대한 기대
- 1969년 민스키와 페퍼트의 『Perceptrons』는 퍼셉트론의 한계 지적. XOR 문제도 해결 못하는 선형 분류기에 불과함 입증 → 신경망 연구 퇴조. 인공지능 겨울에 일조
- 1986년 루멜하트의 『Parallel Distributed Processing』은 은닉층을 가진 다층 퍼셉트론 제안 → 신경망 부활
- 1990년대 SVM에 밀리는 형국
- 2000년대 딥러닝으로 인해 신경망이 인공지능의 주류로 자리매김

4.2 퍼셉트론의 원리

퍼셉트론

- 현재 기준으로 매우 낡은 기술이지만 신경망 공부에서 중요
  - 퍼셉트론은 다층 퍼셉트론과 딥러닝의 핵심 구성 요소이기 때문
- 퍼셉트론은 단순한 모델이기 때문에 기계 학습의 용어와 원리를 설명하는데 적합

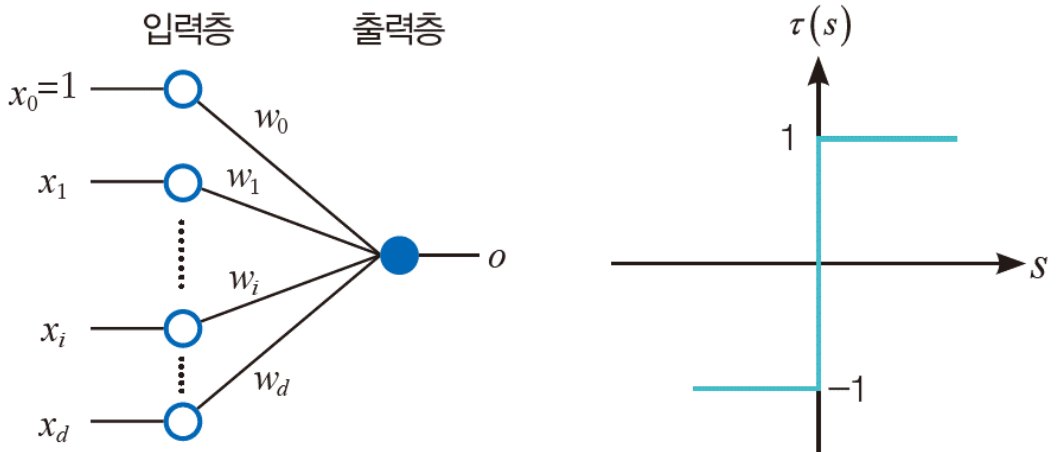
4.2.1 퍼셉트론의 구조와 연산

퍼셉트론의 구조

- 입력층과 출력층으로 구성(출력층은 한 개의 노드)
- 입력층은 d+1개의 노드(d는 특징 벡터의 차원). 예) iris는 d=4, digit는 d=64

$$\text{특징 벡터: } \mathbf{x}=(x_1, x_2, \cdots, x_d) \qquad (4.1)$$

- i번째 입력 노드와 출력 노드는 가중치 w_i를 가진 에지로 연결됨



(a) 퍼셉트론의 구조 (b) 계단 함수를 활성 함수로 사용

그림 4-2 퍼셉트론의 구조와 동작

4.2.1 퍼셉트론의 구조와 연산

퍼셉트론의 연산

- i번째 에지는 x_i와 w_i를 곱해 출력 노드로 전달
- 0번째 입력 노드 x_0은 1인 바이어스 노드
- 출력 노드는 d+1개의 곱셈 결과를 모두 더한 s를 계산하고 활성 함수$_{\text{activation function}}$ 적용

$$\left.\begin{aligned} o = \tau(s) = \tau\left(\sum_{i=1}^{d} w_i x_i + w_0\right) \\ \text{이때 } \tau(s) = \begin{cases} +1, s > 0 \\ -1, s \le 0 \end{cases} \end{aligned}\right\} \quad (4.2)$$

- 활성 함수
  - 뉴런을 활성화하는 과정을 모방
  - 퍼셉트론은 활성 함수로 계단 함수 사용(s가 0보다 크면 1, 그렇지 않으면 -1 출력)
  - 따라서 퍼셉트론은 특징 벡터를 1 또는 -1로 변환하는 장치, 즉 이진 분류기

4.2.2 퍼셉트론으로 인식하기

- 퍼셉트론은 이진 분류기binary classifier
  - 이진 분류 문제의 예)
    - 생산 라인에서 나오는 제품을 우량과 불량으로 구분
    - 병원에 온 사람을 정상인과 환자로 구분
    - 내가 등장하는 사진을 구별해 냄

- [예제 4-1] 퍼셉트론의 인식 능력
  - 제품을 크기와 색상을 나타내는 두 개의 특징으로 표현한다고 가정(d=2). 특징 값은 0 또는 1이라 가정

$$\mathbf{x}=(\text{크기},\text{색상})=(x_1,x_2)$$

  - 생산 라인에서 제품 4개를 수집하여 관찰한 결과 다음 데이터를 확보([그림 4-3(a)])

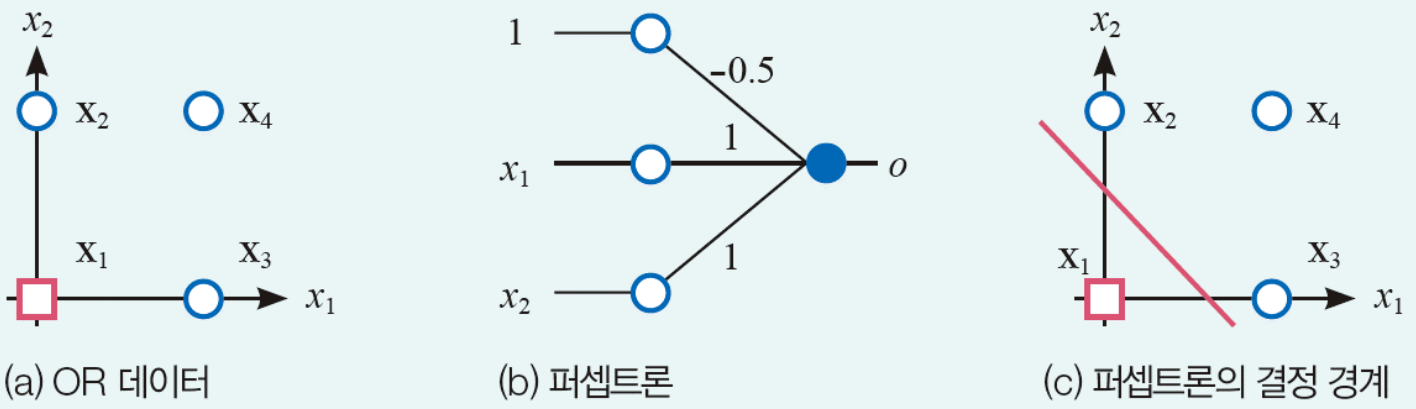
$$\mathbf{x}_1=(0,0),\quad \mathbf{x}_2=(0,1),\quad \mathbf{x}_3=(1,0),\quad \mathbf{x}_4=(1,1)$$ ← 특징 벡터

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

4.2.2 퍼셉트론으로 인식하기

- [그림 4-3(b)]는 데이터를 인식하는 퍼셉트론
  - $\mathbf{x}_2$=(0,1) 샘플을 예측해보면,

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



(a) OR 데이터 (b) 퍼셉트론 (c) 퍼셉트론의 결정 경계

그림 4-3 퍼셉트론의 인식 능력(OR 분류 문제)

NOTE 논리 연산을 본뜬 OR 데이터와 XOR 데이터

[그림 4-3(a)]의 데이터를 분류하는 문제를 OR 분류라고 한다. 1을 참, 0을 거짓으로 간주하면 이진 논리의 OR 연산에 해당하기 때문이다. 단순하여 설명하기 쉽기 때문에 초심자에게 신경망의 동작을 설명할 때 종종 사용한다. 레이블을 y_1=−1, y_2=−1, y_3=−1, y_4=1로 설정하면 AND 데이터가 되고, y_1=−1, y_2=1, y_3=1, y_4=−1로 설정하면 XOR 데이터가 된다.

4.2.2 퍼셉트론으로 인식하기

- 퍼셉트론을 수학적으로 해석하면([그림 4-3(c)])
  - 가중치 w_0=-0.5, w_1=1, w_2=1을 식 (4.2)에 대입하면,

    $$o=w_1*x_1+w_2*x_2+w_0=x_1+x_2-0.5$$

  - o=0으로 설정하면 직선의 방정식을 얻음 → 특징 공간을 분할하는 결정 경계([그림 4-3(c)])

    $$x_1+x_2-0.5=0 \text{ 또는 } x_2=-x_1+0.5$$

  - 퍼셉트론은 특징 공간을 두 부분 공간으로 분할하는 이진 분류기

- 퍼셉트론은 선형으로 국한됨

4.3 사람의 학습과 신경망의 학습

퍼셉트론의 학습

- [그림 4-3]에서는 데이터와 함께 데이터를 인식하는 퍼셉트론(가중치)이 주어짐.
  즉 데이터로 학습을 마친 퍼셉트론이 주어짐
- 실제 상황에서는 데이터만 주어지므로, 학습 알고리즘으로 가중치($w_0, w_2, \ldots, w_d$)를 알아내야 함
  - OR 데이터의 경우 2차원 특징 벡터이고 샘플이 4개 뿐이라 연필을 가지고 쉽게 가중치를 알아낼 수 있음
  - sklearn의 필기 숫자 데이터는 64차원 특징 벡터이고 1,797개 샘플을 가지므로 학습 알고리즘 없이 불가능

4.3.1 사람의 학습 알고리즘

- 사람이 수영을 학습하는 과정을 알고리즘 형식으로 기술하면,

[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.
```

- 좀 더 수학적으로 기술하면,

[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(w)를 계산한다.
04.       if (J(w)가 만족스러움) break
05.       더 나은 방향 Δw를 계산한다.
06.       w=w+Δw
07. w를 기억한다.
```

4.3.2 신경망의 학습 알고리즘

- 조금씩 나은 방향을 찾아 개선해 나가는 절차는 사람의 학습과 같음
  - 신경망은 철저히 수학에 의존한다는 점에서 사람과 다름
    - 신경망의 학습은 최적화 문제. 최적화 대상은 신경망의 가중치(매개변수)

[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.
```

  - 구현에 필요한 사항
    - 1행의 초기화는 어떻게? 4행의 멈춤 조건은 어떻게?
    - 3행의 손실 함수 정의(보통 오류의 양을 사용)
    - 5행에서 Δw를 어떻게 구하나?(미분을 사용)

4.4 퍼셉트론 학습 알고리즘

- 학습 알고리즘 고안
  - 손실 함수 J를 설계([알고리즘 4-3]의 03행)
  - 손실 함수의 값을 낮추는 방향을 찾는 방법([알고리즘 4-3]의 05행)

4.4.1 손실 함수 설계

- 손실 함수 J가 만족해야 할 조건

① **w**가 훈련 집합에 있는 샘플을 모두 맞히면, 즉 정확률이 100%이면 $J(\mathbf{w})$는 0이다.

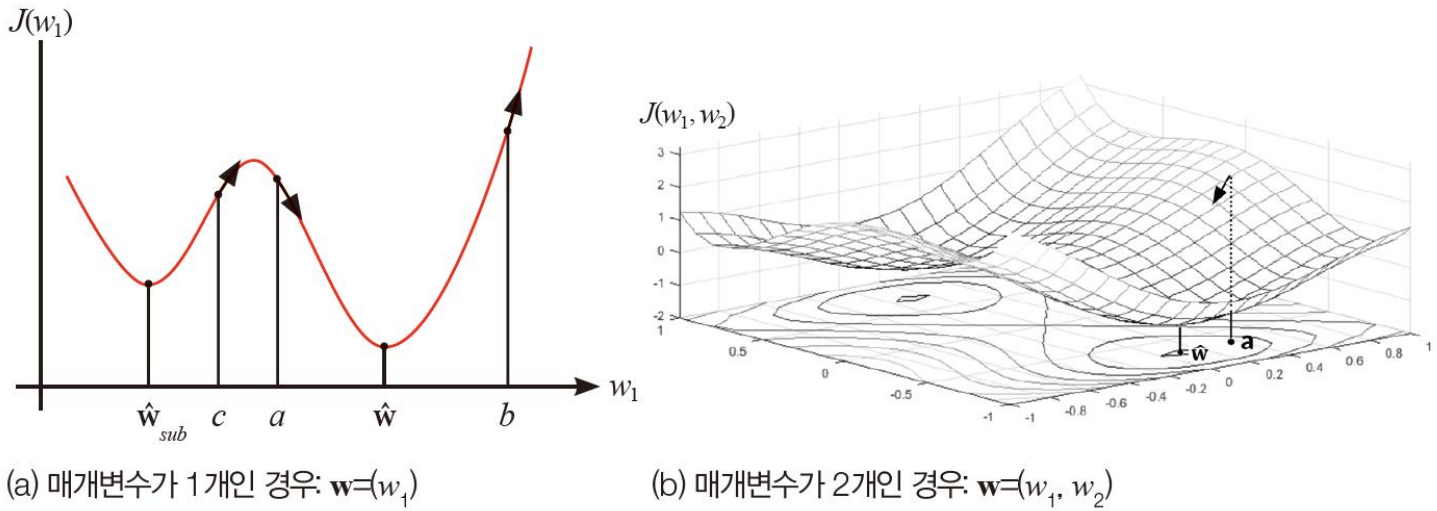
② **w**가 틀리는 샘플이 많을수록 $J(\mathbf{w})$의 값이 크다.

  - 조건을 만족하는 함수는 여럿. 식 (4.5)는 그 중 하나로서 직관적으로 이해하기 쉬움
    - I는 **w**가 틀리는 샘플
    - **x**가 +1 부류라면(즉 y=1이라면), 퍼셉트론의 출력 $\mathbf{wx}^T$는 음수. $-y(\mathbf{wx}^T)$는 양수
    - **x**가 -1 부류라면(즉 y=-1이라면), 퍼셉트론의 출력 $\mathbf{wx}^T$는 양수. $-y(\mathbf{wx}^T)$는 양수
    - 결국 틀린 샘플 **x**는 손실 함수의 값을 증가시킴(틀린 샘플이 많을수록 손실 함수 값은 커짐)

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

4.4.2 학습 알고리즘 설계

- 경사 하강법gradient descent의 원리([그림 4-5]는 가상의 손실 함수)
  - 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

4.4.2 학습 알고리즘 설계

- 학습 규칙 유도([그림 4-5(a)]는 매개변수가 하나인 경우)
  - 경사 하강법은 미분을 이용하여 최적해를 찾는 기법
  - 미분값 $\frac{\partial J}{\partial w_1}$의 반대 방향이 최적해에 접근하는 방향이므로 현재 w_1에 $-\frac{\partial J}{\partial w_1}$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)

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

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

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

4.4.2 학습 알고리즘 설계

- 퍼셉트론에 식 (4.7) 적용
  - 식 (4.5)를 매개변수 w_i로 편미분하면,

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

  - 정리하면,

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

  - 식 (4.8)을 식 (4.7)에 대입하면,

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

퍼셉트론 학습 알고리즘이 사용하는 핵심 공식

  - 식 (4.9)를 행렬 형태로 쓰면,

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

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 세대$_{epoch}$라 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

4.4.2 학습 알고리즘 설계

- [그림 4-6]은 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)

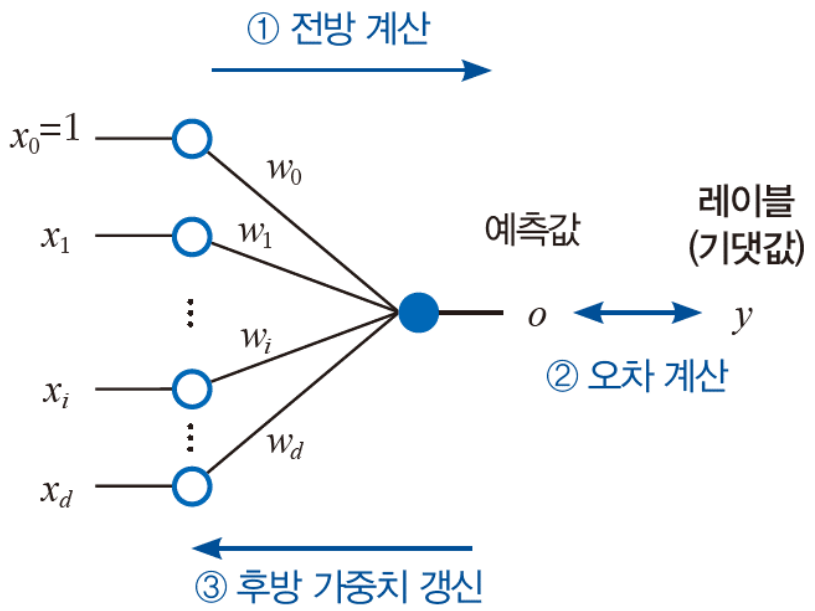


그림 4-6 퍼셉트론 학습 알고리즘의 절차

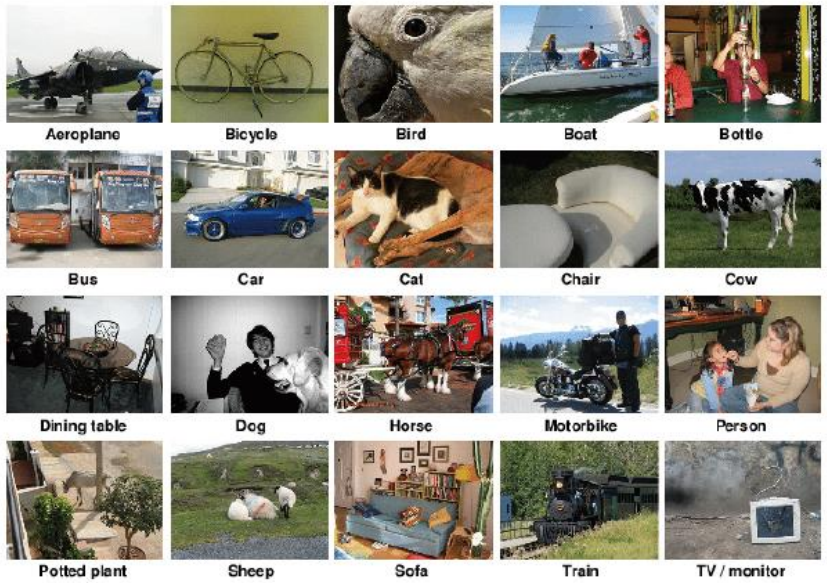
4.5 현대 기계 학습으로 확장

- 앞 절은
  - 이 책에서 수학이 가장 많이 등장하는 절(앞으로는 수학이 적게 등장)
  - 단순한 퍼셉트론을 이용하여 기계 학습의 원리를 설명하고 딥러닝으로 도약할 발판을 마련할 목적이었음

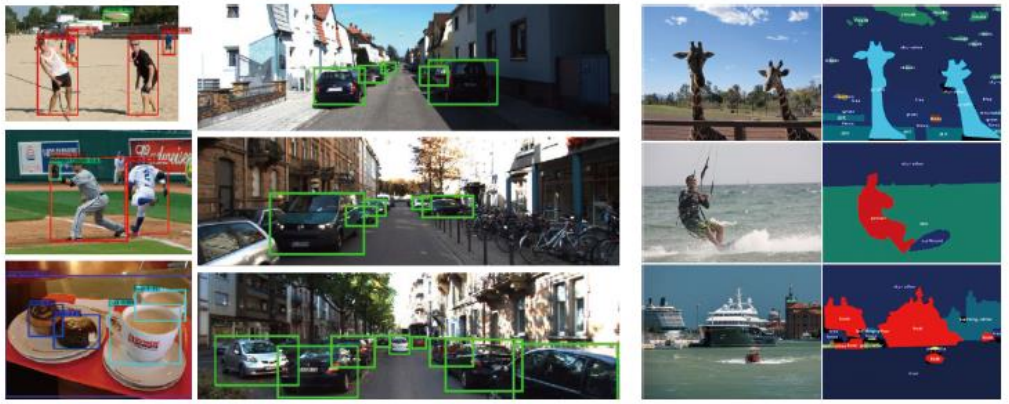
- 이 절은
  - 앞 절에서 공부한 퍼셉트론 원리를 바탕으로 현대 기계 학습으로 개념 확장

4.5.1 현대적인 기계학습의 복잡도

- [그림 4-7]의 세 가지 문제
  - 분류$_{\text{classification}}$: 지정된 몇 가지 부류로 구분하는 문제
  - 물체 검출$_{\text{object detection}}$: 물체 위치를 바운딩 박스로 찾아내는 문제
    (사진 촬영에서 얼굴 초점 응용)
  - 물체 분할$_{\text{object segmentation}}$: 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

4.5.1 현대적인 기계학습의 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: 화소 별로 레이블이 지정한 물체에 해당하는지 따지는 손실 함수

- 실용적인 신경망의 매개변수 개수는 방대
  - 예) 딥러닝 모델 ResNet-50은 50개의 층을 가지며 매개변수는 2천300만개 이상
  - 다행히 1~2차원에서 개발한 공식이 고차원에 그대로 적용
  - 단지 과잉 적합을 방지하기 위해 더 많은 데이터 사용 또는 더 정교한 규제 기법 적용. 속도 향상을 위한 GPU 사용

4.5.2 스토캐스틱 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

기계 학습의 경사 하강법

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

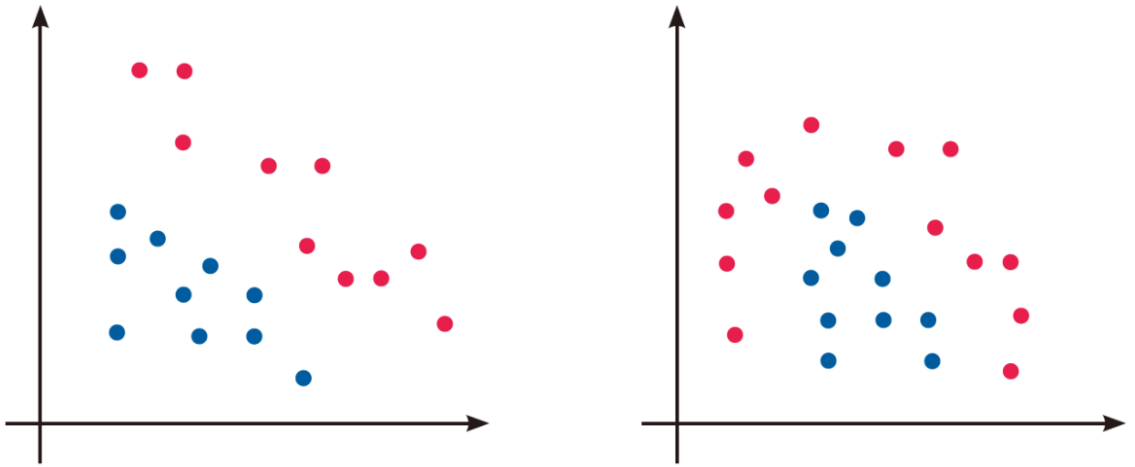
  ← 이런 특성은 최적해를 찾는 일을 어렵게 만듦. 정확률이 등락을 거듭하며 수렴하지 않는다거나 훈련 집합에 대해 높은 성능을 얻었는데 테스트 집합에 대해 형편 없는 성능 등의 문제

4.5.2 스토캐스틱 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

4.7 다층 퍼셉트론

- 퍼셉트론은 선형이라는 한계
  - [그림 4-9]의 선형 분리 불가능한 데이터에서는 높은 오류율([프로그램 4-2]의 6.12% 오류율의 원인)



(a) 선형 분리 가능　　(b) 선형 분리 불가능

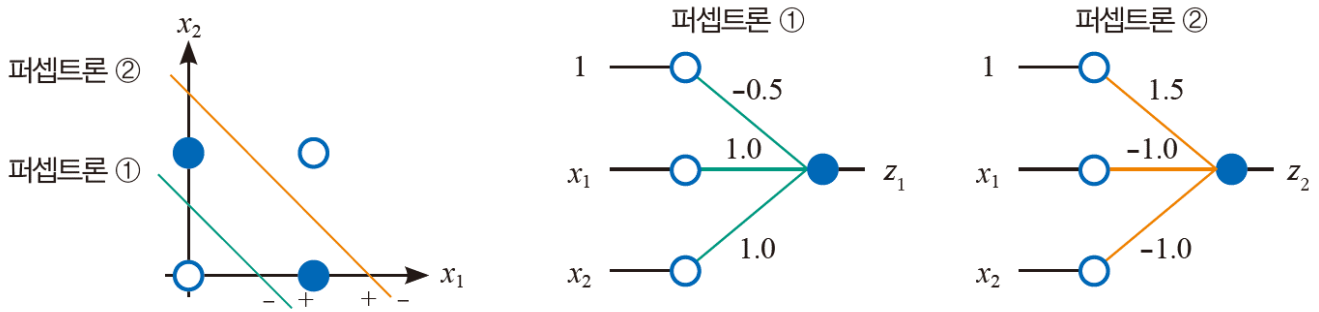
그림 4-9 데이터의 선형 분리 가능성

  - XOR에서는 25% 오류율([그림 4-10(a)])

- 이 절에서는 은닉층을 추가한 다층 퍼셉트론으로 비선형으로 확장

4.7.1 특징 공간 변환

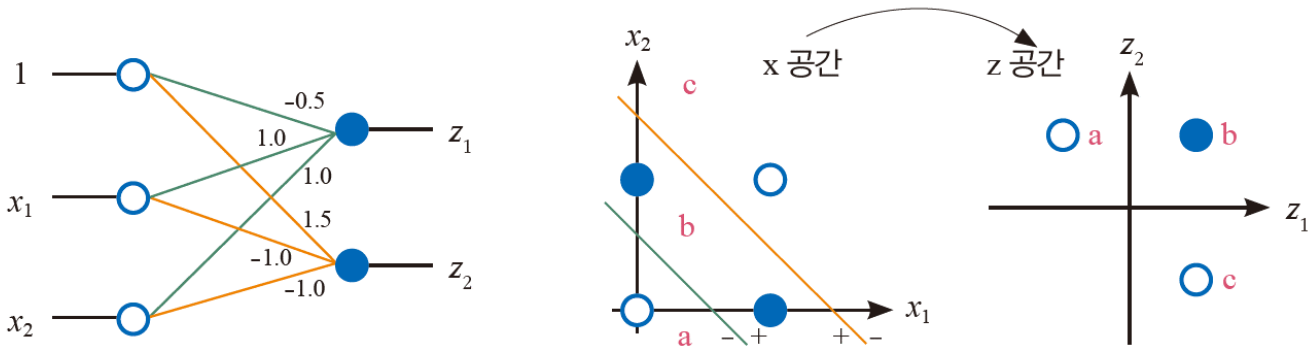
- 퍼셉트론 두 개로 특징 공간을 세 개의 부분 공간으로 나눌 수 있음



(a) 퍼셉트론 2개를 이용해 부분공간 3개로 분할 (b) 2개의 퍼셉트론

그림 4-10 퍼셉트론 2개로 XOR 데이터를 인식하는 길을 찾음

- 두 퍼셉트론을 병렬로 결합하면 (x_1,x_2) 공간을 (z_1,z_2) 공간으로 변환

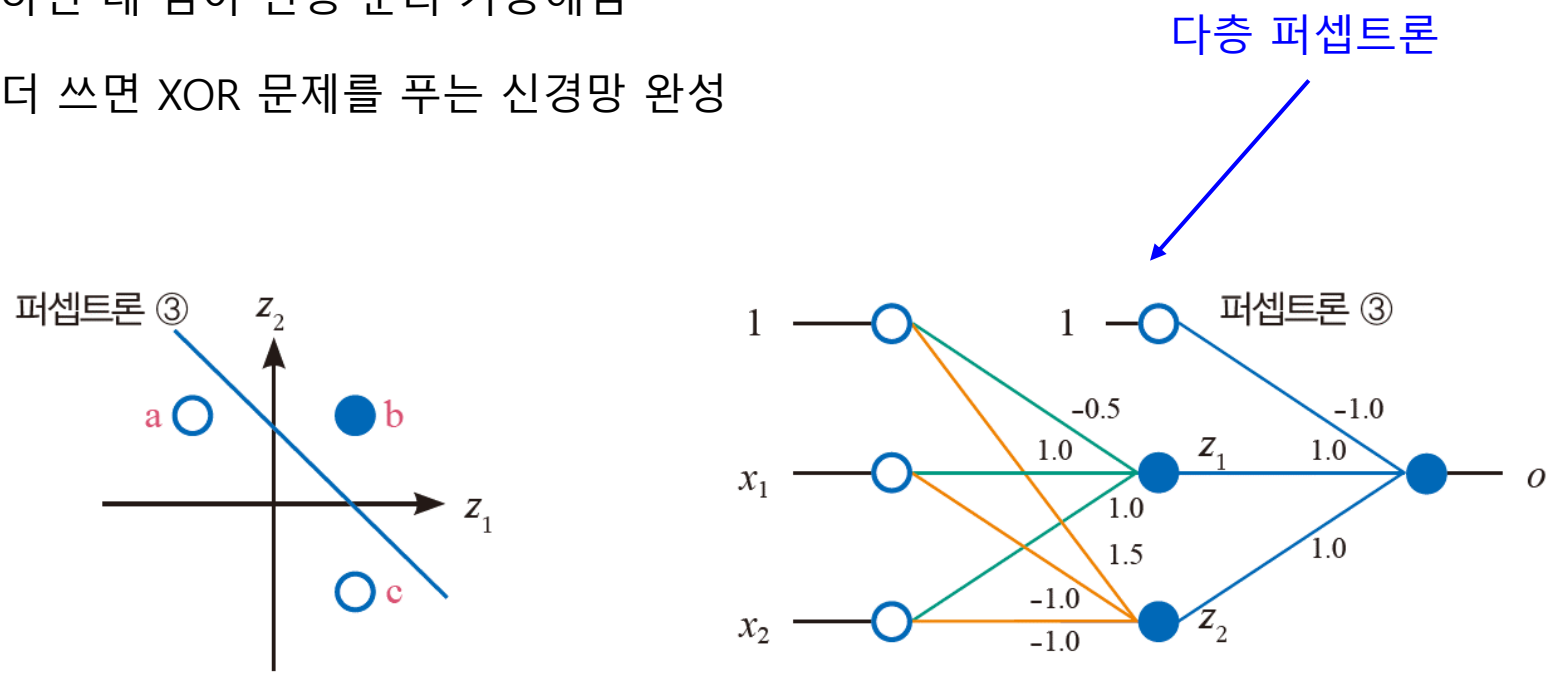


(a) 공간 변환하는 퍼셉트론 2개의 병렬 결합 (b) 특징 공간 변환(세 부분공간을 세 점으로 매핑)

그림 4-11 퍼셉트론 2개를 병렬 결합해 특징 공간을 변환

4.7.1 특징 공간 변환

- 새로운 공간 (z_1,z_2)의 흥미로운 특성
  - 선형 분리 불가능하던 네 점이 선형 분리 가능해짐
  - 퍼셉트론을 하나 더 쓰면 XOR 문제를 푸는 신경망 완성



(a) 공간 분할하는 세 번째 퍼셉트론

(b) 세 번째 퍼셉트론을 순차적으로 덧붙임

그림 4-12 다층 퍼셉트론

4.7.1 특징 공간 변환

- [예제 4-2] XOR를 푸는 다층 퍼셉트론

[그림 4-12(b)]의 다층 퍼셉트론에 XOR 데이터의 샘플 4개를 차례로 입력하고 제대로 인식하는지 확인해보자. 아래 표는 샘플 4개를 인식하는 과정을 보여준다. 100% 옳게 분류하는 것을 확인할 수 있다.

원래 특징 공간(x_1, x_2)	은닉 특징 공간(z_1, z_2)	출력 o	레이블 y
(0,0)	(–1,1)	–1	–1
(0,1)	(1,1)	1	1
(1,0)	(1,1)	1	1
(1,1)	(1,–1)	–1	–1

4.7.1 특징 공간 변환

- 신경망을 공간 변환기로 볼 수 있음
  - 원래 특징 공간을 임시 공간으로 변환하고, 임시 공간을 레이블 공간으로 변환하는 두 단계 처리
  - 임시 공간에 해당하는 층을 은닉층$_{\text{hidden layer}}$이라 부름. 임시 공간을 은닉 공간$_{\text{hidden space}}$ 또는 잠복 공간$_{\text{latent space}}$이라 부름
  - 새로운 공간은 이전 공간보다 분류에 더 유리하도록 학습됨
  - 자연 영상이나 자연어처럼 복잡한 데이터는 대여섯 은닉층 또는 수십~수백 개의 은닉층 → 이런 깊은 신경망의 학습을 딥러닝이라 부름

4.7.2 다층 퍼셉트론의 구조

- 입력층, 은닉층, 출력층으로 구성된 다층 퍼셉트론
  - 층을 연결하는 가중치 뭉치가 두 개 있어 3층이 아니라 2층 신경망으로 간주함
  - 데이터에 따라 입력층과 출력층의 노드 개수 확정
    - 예) iris에서는 5개의 입력 노드와 3개의 출력 노드
  - 은닉층의 노드 개수는 하이퍼 매개변수
    - 은닉 노드가 많으면 신경망 용량이 커지지만 과잉 적합 가능성 높아짐

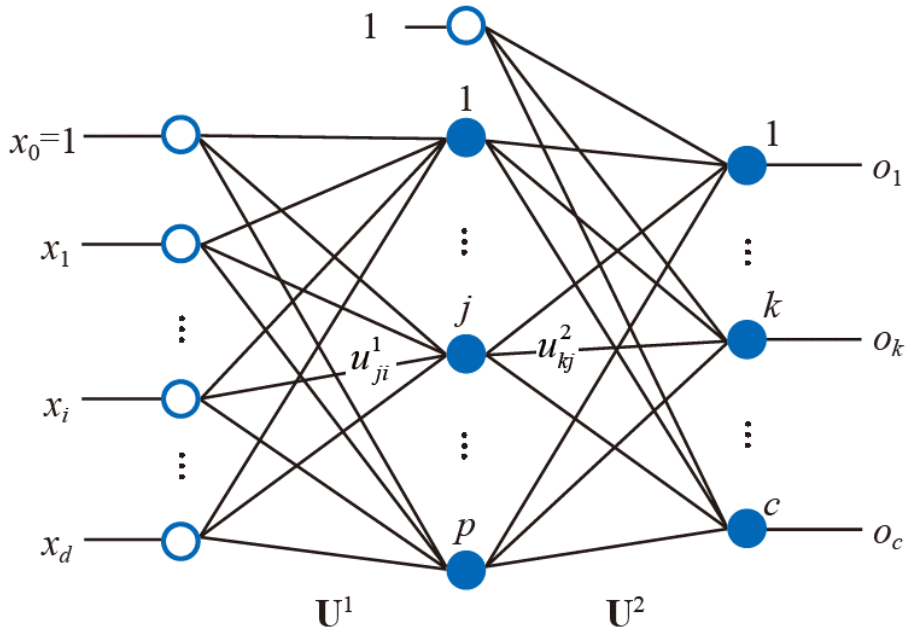


그림 4-13 다층 퍼셉트론의 구조

4.7.2 다층 퍼셉트론의 구조

- 가중치 행렬 $\mathbf{U}^1$과 $\mathbf{U}^2$
  - u_{ji}^1은 i번째 입력 노드와 j번째 은닉 노드를 연결하는 가중치
  - u_{kj}^2는 j번째 은닉 노드와 k번째 출력 노드를 연결하는 가중치

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix}, \quad \mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} \tag{4.11}$$

4.7.3 다층 퍼셉트론의 동작

- j번째 은닉 노드의 동작([그림 4-13])
  - $\mathbf{u}_j^1$은 $\mathbf{U}^1$의 j번째 행
  - 은닉층 때문에 첨자가 많아져 복잡해 보이지만 본질적으로 퍼셉트론의 식 (4.2)와 동일

$$
\begin{aligned}
&j\text{번째 은닉 노드의 연산:}\\
&z_j = \tau_1\left(s_j\right),\ j=1,2,\cdots,p\\
&\text{이때 } s_j = \mathbf{u}_j^1\mathbf{x}^{\mathrm{T}}\text{이고 } \mathbf{u}_j^1 = \left(u_{j0}^1, u_{j1}^1, \cdots, u_{jd}^1\right),\ \mathbf{x}=(1,\ x_1,\ x_2,\cdots x_d)
\end{aligned}
\qquad (4.12)
$$

- k번째 출력 노드의 동작
  - $\mathbf{u}_k^2$는 $\mathbf{U}^2$의 k번째 행

$$
\begin{aligned}
&k\text{번째 은닉 노드의 연산:}\\
&o_k = \tau_2\left(s_k\right),\ k=1,2,\cdots,c\\
&\text{이때 } s_k = \mathbf{u}_k^2\mathbf{z}^{\mathrm{T}}\text{이고 } \mathbf{u}_k^2 = \left(u_{k0}^2, u_{k1}^2, \cdots, u_{kp}^2\right),\ \mathbf{z} = \left(1, z_1, z_2, \cdots, z_p\right)
\end{aligned}
\qquad (4.13)
$$

4.7.3 다층 퍼셉트론의 동작

[예제 4-3] 다층 퍼셉트론의 동작

- [그림 4-12(b)] 다층 퍼셉트론의 가중치 행렬(d=2, p=2, c=1)

$$\mathbf{U}^1 = \begin{pmatrix} -0.5 & 1.0 & 1.0 \\ 1.5 & -1.0 & -1.0 \end{pmatrix}, \mathbf{U}^2 = (-1 \quad 1.0 \quad 1.0)$$

- 샘플을 하나씩 처리하는 식 (4.14) 적용
  - **x**=(0,1) 샘플을 처리(계단 함수 사용, 바이어스 항을 추가해 (1,0,1) 형식으로 입력)

$$\begin{aligned} \mathbf{O} &= \tau_2\left((-1 \quad 1.0 \quad 1.0)\,\tau_1\left(\begin{pmatrix} -0.5 & 1.0 & 1.0 \\ 1.5 & -1.0 & -1.0 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\right)\right) \\ &= \tau_2\left((-1 \quad 1.0 \quad 1.0)\,\tau_1\left(\begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}\right)\right) \\ &= \tau_2\left((-1 \quad 1.0 \quad 1.0)\begin{pmatrix} 1 \\ 1.0 \\ 1.0 \end{pmatrix}\right) \\ &= \tau_2((1)) = 1 \end{aligned}$$

← 레이블과 같으므로 맞힘

4.7.3 다층 퍼셉트론의 동작

- [예제 4-3] 다층 퍼셉트론의 동작(...앞에서 계속)
  - 데이터를 한꺼번에 처리하는 식 (4.16) 적용

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

$$\mathbf{O} = \tau_2\left(\begin{pmatrix} -1 & 1.0 & 1.0 \end{pmatrix}\tau_1\left(\begin{pmatrix} -0.5 & 1.0 & 1.0 \\ 1.5 & -1.0 & -1.0 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}\right)\right)$$

$$= \tau_2\left(\begin{pmatrix} -1 & 1.0 & 1.0 \end{pmatrix}\tau_1\left(\begin{pmatrix} -0.5 & 0.5 & 0.5 & 1.5 \\ 1.5 & 0.5 & 0.5 & -0.5 \end{pmatrix}\right)\right)$$

$$= \tau_2\left(\begin{pmatrix} -1 & 1.0 & 1.0 \end{pmatrix}\begin{pmatrix} 1 & 1 & 1 & 1 \\ -1.0 & 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 & -1.0 \end{pmatrix}\right)$$

$$= \tau_2\left(\begin{pmatrix} -1 & 1 & 1 & -1 \end{pmatrix}\right) = \begin{pmatrix} -1 & 1 & 1 & -1 \end{pmatrix}$$

네 개 샘플의 레이블과 같으므로 모두 맞힘

4.7.3 다층 퍼셉트론의 동작

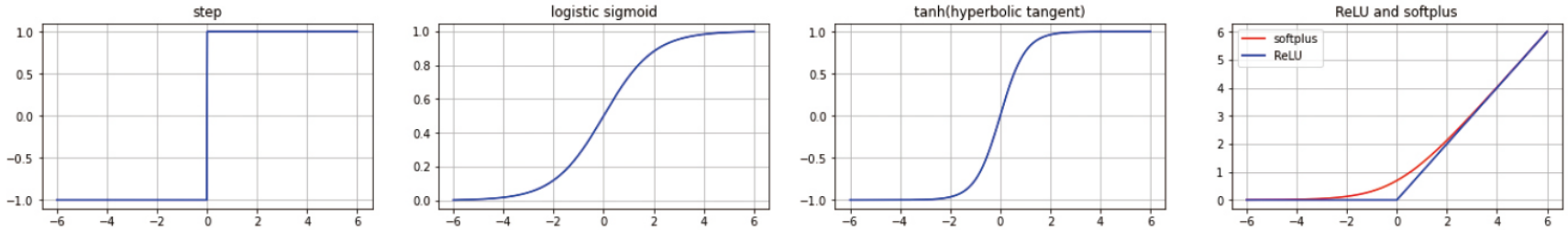
- 출력 벡터 **O**의 모양
  - [예제 4-3]은 출력 노드가 하나이고 샘플이 4개이므로 **O**는 1*4 행렬
  - 일반적으로 출력 노드가 c이고(c는 부류 개수) 샘플이 n개 이면 **O**는 c*n 행렬
  - 열은 샘플을 나타내며, 값이 가장 큰 인덱스를 최종 분류 결과로 출력
  - 예) 숫자 인식

$$\mathbf{O} = \begin{pmatrix} 0.1 & 0 & 0.9 & 0 & & 0.78 \\ 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & & 0 \\ 0.001 & 0 & 0 & 0.01 & & 0 \\ 0.899 & 0 & 0 & 0 & \cdots & 0.02 \\ 0 & 0 & 0 & 0.99 & & 0 \\ 0 & 0.22 & 0.1 & 0 & & 0.2 \\ 0 & 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & 0 & & 0 \\ 0 & 0.78 & 0 & 0 & & 0 \end{pmatrix} \rightarrow \text{최종 분류 결과} = (4 \quad 9 \quad 0 \quad 5 \quad \cdots \quad 0)$$

4.7.4 활성 함수

다양한 활성 함수

- 퍼셉트론은 계단 함수를 사용하는데, 예측 결과를 0~1 사이의 확률로 표현해야 하는 경우에 계단 함수는 부적절함
- 따라서 다층 퍼셉트론은 시그모이드, 딥러닝은 ReLU와 softmax를 주로 사용



(a) 계단 함수 (b) 로지스틱 시그모이드 (c) 하이퍼볼릭 탄젠트 시그모이드 (d) 소프트플러스와 ReLU

그림 4-14 신경망이 사용하는 다양한 활성 함수

4.7.4 활성 함수

- 다양한 활성 함수

표 4-1 신경망에서 사용하는 여러 가지 활성 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 (tanh) 시그모이드	$\tau(s)=\frac{2}{1+e^{-as}}-1$	$\tau'(s)=\frac{a}{2}(1-\tau(s)^2)$	(−1,1)
소프트플러스	$\tau(s)=\log_e(1+e^s)$	$\tau'(s)=\frac{1}{1+e^{-s}}$	(0,∞)
렉티파이어(ReLU)	$\tau(s)=\max(0,s)$	$\tau'(s)=\begin{cases}0 & s<0\\ 1 & s>0\\ \text{불가} & s=0\end{cases}$	(0,∞)

- 출력층에서 주로 사용하는 소프트맥스 함수(확률로 간주할 수 있음)

$$o_k=\frac{e^{s_k}}{\sum_{i=1,c}e^{s_i}} \qquad (4.17)$$

← $o_1+o_2+\ldots+o_c=1.0$을 만족함

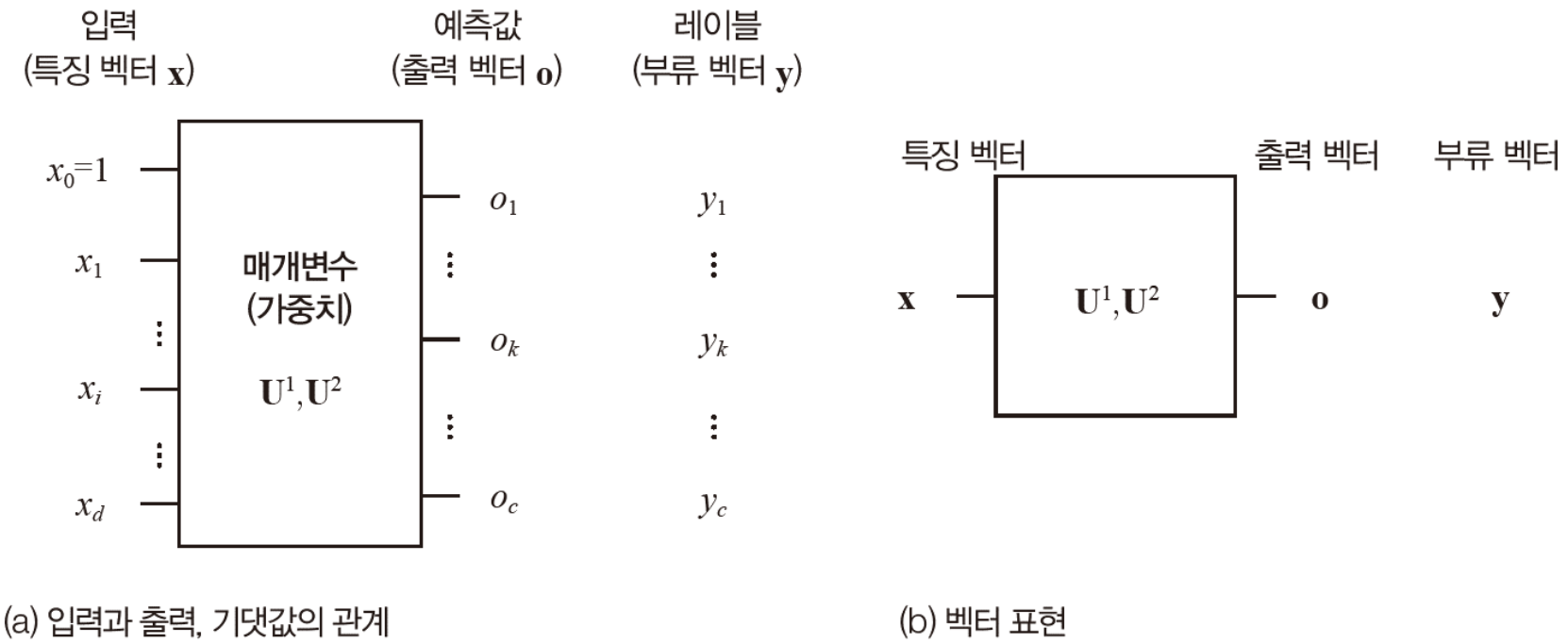
4.8 오류 역전파 알고리즘

- 다층 퍼셉트론은 오류 역전파 알고리즘으로 학습함
  - 은닉층이 있고 활성 함수가 시그모이드이므로 퍼셉트론 학습 알고리즘보다 복잡하지만 기본 원리는 같음

4.8.1 손실 함수 설계

다층 퍼셉트론의 블록 다이어그램

- 신경망의 출력 벡터(예측값) **o**가 부류 벡터(참값) **y**와 같을수록 매개변수 $\mathbf{U}^1$과 $\mathbf{U}^2$는 데이터를 잘 인식. 다르면 **o**와 **y**가 가까워지도록 $\mathbf{U}^1$과 $\mathbf{U}^2$를 갱신해야 함. 오차가 클수록 갱신하는 양이 큼



(a) 입력과 출력, 기댓값의 관계

(b) 벡터 표현

그림 4-15 다층 퍼셉트론의 블록 다이어그램

4.8.1 손실 함수 설계

- 샘플 하나의 오차를 측정하는 손실 함수
  - 보통 **y**는 원핫 코드로 표현. 예) 숫자 0이면 **y**=(1,0,0,…,0), 2면 **y**=(0,0,1,0,…,0)

$$J\left(\mathbf{U}^1,\mathbf{U}^2\right)=\sqrt{\left(y_1-o_1\right)^2+\left(y_2-o_2\right)^2+\cdots+\left(y_c-o_c\right)^2}=\|\mathbf{y}-\mathbf{o}\| \qquad (4.18)$$

- 식 (4.18)을 확장
  - 계산 편의를 위해 제곱을 하여 제곱근 제거
  - 미니 배치 단위로 처리(샘플의 오차를 평균)

평균제곱오차(MSE, mean squared error)

$$J\left(\mathbf{U}^1,\mathbf{U}^2\right)=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\|\mathbf{y}-\mathbf{o}\|^2$$

$$=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\left\|\mathbf{y}-\tau_2\left(\mathbf{U}^2\tau_1\left(\mathbf{U}^1\mathbf{x}^T\right)\right)\right\|^2 \qquad (4.19)$$

4.8.2 학습 알고리즘

가중치 갱신 규칙

- 퍼셉트론과 비슷
- $\frac{\partial J}{\partial u_{ji}^1}$와 $\frac{\partial J}{\partial u_{kj}^2}$를 구하는 과정은 생략

$$\mathbf{U}^2 = \mathbf{U}^2 + \rho\left(-\nabla \mathbf{U}^2\right) \qquad \mathbf{U}^1 = \mathbf{U}^1 + \rho\left(-\nabla \mathbf{U}^1\right)$$

$$\nabla \mathbf{U}^2 = \begin{pmatrix} \frac{\partial J}{\partial u_{10}^2} & \frac{\partial J}{\partial u_{11}^2} & \cdots & \frac{\partial J}{\partial u_{1p}^2} \\ \frac{\partial J}{\partial u_{20}^2} & \frac{\partial J}{\partial u_{21}^2} & \cdots & \frac{\partial J}{\partial u_{2p}^2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial u_{c0}^2} & \frac{\partial J}{\partial u_{c1}^2} & \cdots & \frac{\partial J}{\partial u_{cp}^2} \end{pmatrix} \qquad \nabla \mathbf{U}^1 = \begin{pmatrix} \frac{\partial J}{\partial u_{10}^1} & \frac{\partial J}{\partial u_{11}^1} & \cdots & \frac{\partial J}{\partial u_{1d}^1} \\ \frac{\partial J}{\partial u_{20}^1} & \frac{\partial J}{\partial u_{21}^1} & \cdots & \frac{\partial J}{\partial u_{2d}^1} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial u_{p0}^1} & \frac{\partial J}{\partial u_{p1}^1} & \cdots & \frac{\partial J}{\partial u_{pd}^1} \end{pmatrix} \tag{4.20}$$

TIP 미분값을 구하는 수식 유도는 이 책의 범위를 넘으므로 생략한다. 관심있는 독자는 『기계 학습(오일석, 한빛아카데미)』의 3.4절을 참조한다.

4.8.2 학습 알고리즘

- 오류 역전파error-backpropagation 학습 알고리즘
  - 출력층에서 시작하여 역방향으로 오류를 전파한다는 뜻에서 오류 역전파라 부름

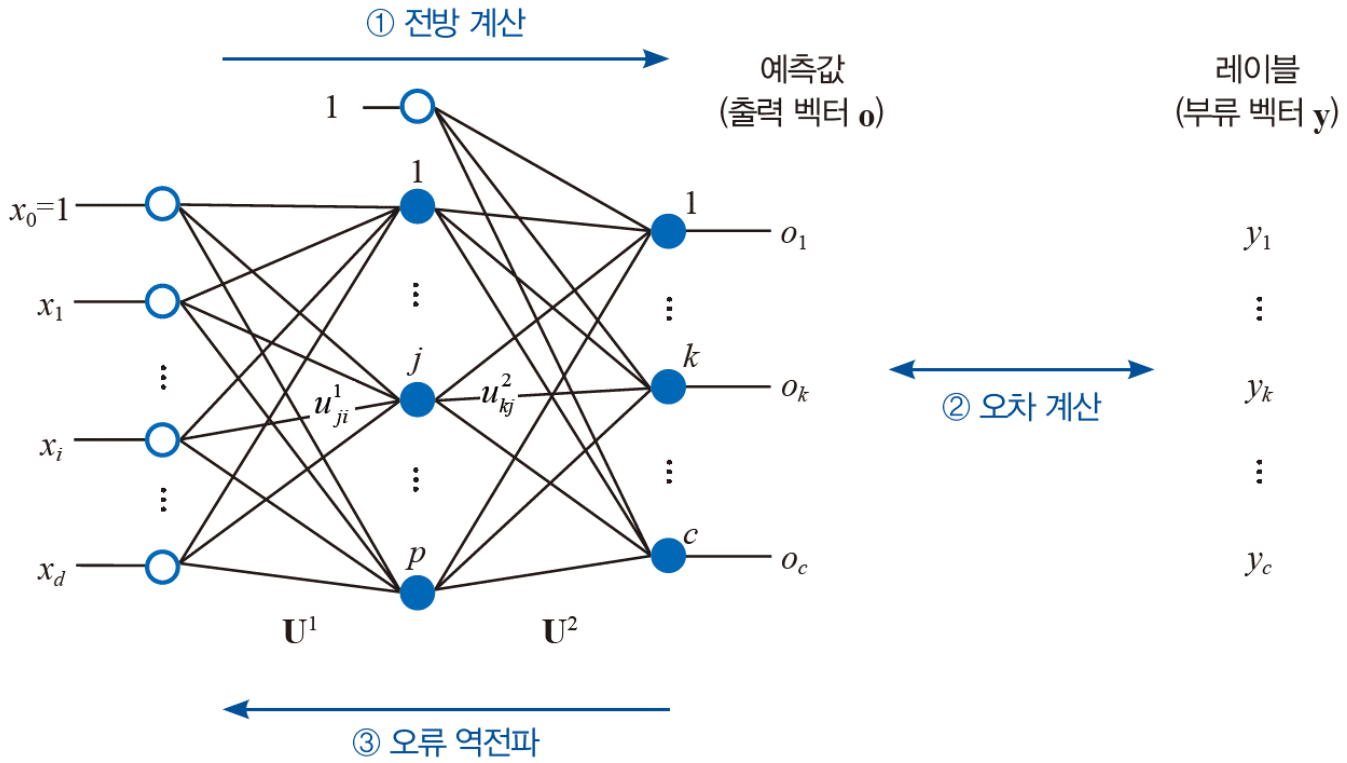


그림 4-16 오류 역전파 학습 알고리즘

4.9 다층 퍼셉트론 프로그래밍

- 다층 퍼셉트론
  - 1980~1990년대 실용적인 시스템을 구현하는데 널리 사용
  - 현재도 사용됨

4.9.1 sklearn의 필기 숫자 데이터셋

- [프로그램 4-3]

프로그램 4-3 sklearn 필기 숫자 데이터에 다층 퍼셉트론 적용

```
01  from sklearn import datasets
02  from sklearn.neural_network import MLPClassifier
03  from sklearn.model_selection import train_test_split
04  import numpy as np
05
06  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
07  digit=datasets.load_digits()
08  x_train,x_test,y_train,y_test=train_test_split(digit.data,digit.target,train_
    size=0.6)
09
10  # MLP 분류기 모델을 학습
11  mlp=MLPClassifier(hidden_layer_sizes=(100),learning_rate_init=0.001,batch_
    size=32,max_iter=300,solver='sgd',verbose=True)
12  mlp.fit(x_train,y_train)
13
14  res=mlp.predict(x_test) # 테스트 집합으로 예측
15
16  # 혼동 행렬
17  conf=np.zeros((10,10))
18  for i in range(len(res)):
```

4.9.1 sklearn의 필기 숫자 데이터셋

```
19      conf[res[i]][y_test[i]]+=1
20  print(conf)
21
22  # 정확률 계산
23  no_correct=0
24  for i in range(10):
25      no_correct+=conf[i][i]
26  accuracy=no_correct/len(res)
27  print("테스트 집합에 대한 정확률은 ", accuracy*100, "%입니다.")
```

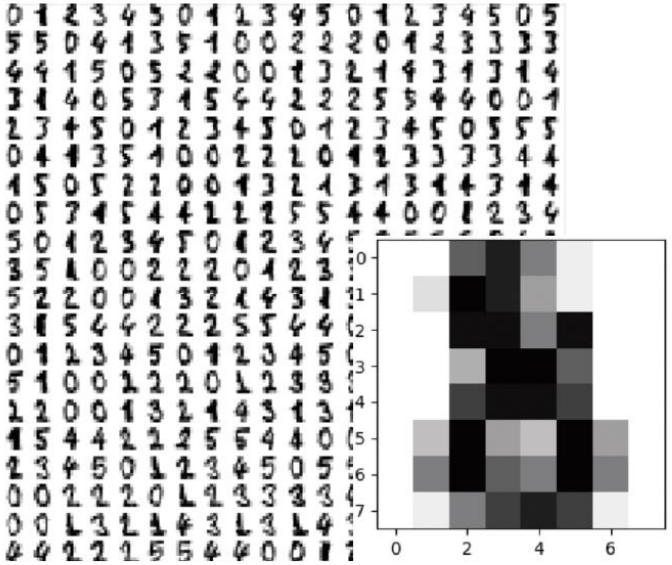
4.9.1 sklearn의 필기 숫자 데이터셋

- 실행 결과
  - 정확률 97.2%로서 [프로그램 3-6]의 SVM(98.7%)보다 열등하고 [프로그램 4-2]의 퍼셉트론(93.8%)보다 우수

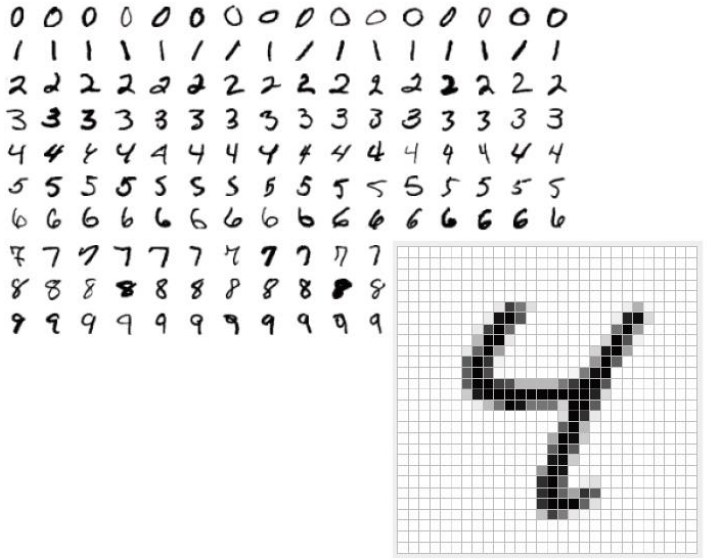
```
Iteration 1, loss = 1.93427517
Iteration 2, loss = 0.32451464
Iteration 3, loss = 0.20142691
Iteration 4, loss = 0.14313824
...
Iteration 40, loss = 0.01163957
Iteration 41, loss = 0.01127886
...
Iteration 80, loss = 0.00538125
Iteration 81, loss = 0.00538663
...
Iteration 107, loss = 0.00405861
Iteration 108, loss = 0.00402524
Iteration 109, loss = 0.00397349
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.

[[73.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 69.  1.  0.  1.  0.  1.  0.  1.  0.]
 [ 0.  0. 70.  1.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. 67.  0.  0.  0.  0.  1.  3.]
 [ 1.  0.  0.  0. 77.  0.  1.  0.  0.  0.]
 [ 1.  0.  0.  0.  0. 64.  1.  0.  0.  1.]
 [ 0.  0.  0.  0.  0.  0. 67.  0.  0.  0.]
 [ 0.  0.  0.  1.  0.  1.  0. 83.  0.  0.]
 [ 0.  2.  0.  0.  0.  0.  0.  1. 58.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0.  0. 71.]]

테스트 집합에 대한 정확률은 97.2183588317107%입니다.
```

4.9.2 MNIST 데이터셋으로 확장하기

- MNIST 필기 숫자 데이터셋([그림 3-6(b)])
  - 훈련 집합 60000자 + 테스트 집합 10000자
  - 샘플은 28*28 맵으로 표현



(a) sklearn에서 제공하는 데이터셋



(b) MNIST 데이터셋

그림 3-6 필기 숫자 데이터셋

4.9.2 MNIST 데이터셋으로 확장하기

- [프로그램 4-4]
  - [프로그램 4-3]과 매우 유사

프로그램 4-4 MNIST 데이터셋을 다층 퍼셉트론으로 인식

```
01  from sklearn.datasets import fetch_openml
02  from sklearn.neural_network import MLPClassifier
03  import matplotlib.pyplot as plt
04  import numpy as np
05
06  # MNIST 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
07  mnist=fetch_openml('mnist_784')
08  mnist.data=mnist.data/255.0
09  x_train=mnist.data[:60000]; x_test=mnist.data[60000:]
10  y_train=np.int16(mnist.target[:60000]); y_test=np.int16(mnist.target[60000:])
11
12  # MLP 분류기 모델을 학습
13  mlp=MLPClassifier(hidden_layer_sizes=(100),learning_rate_init=0.001,batch_
    size=512,max_iter=300,solver='adam',verbose=True)
14  mlp.fit(x_train,y_train)
15
16  # 테스트 집합으로 예측
17  res=mlp.predict(x_test)
18
19  # 혼동 행렬
20  conf=np.zeros((10,10),dtype=np.int16)
21  for i in range(len(res)):
22      conf[res[i]][y_test[i]]+=1
23  print(conf)
24
25  # 정확률 계산
26  no_correct=0
27  for i in range(10):
28     no_correct+=conf[i][i]
29  accuracy=no_correct/len(res)
30  print("테스트 집합에 대한 정확률은", accuracy*100, "%입니다.")
```

8행: [0,255] 범위를 [0,1] 범위로 변환

09~10행: 앞 6만 자를 훈련,
뒤 1만 자를 테스트로 분할

4.9.2 MNIST 데이터셋으로 확장하기

- 실행 결과 97.78% 정확률을 얻음

```
Iteration 1, loss = 0.61803286
Iteration 2, loss = 0.26101832
Iteration 3, loss = 0.20624874
Iteration 4, loss = 0.17280300
...
Iteration 82, loss = 0.00074115
Iteration 83, loss = 0.00072676
Iteration 84, loss = 0.00067264
Iteration 85, loss = 0.00065032
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.

[[970    0    4    0    0    1    4    0    4    1]
 [  0 1124    2    0    0    0    2    4    1    3]
 [  1    3 1002    6    3    0    1   10    4    0]
 [  0    1    6  988    2   10    1    1    6    5]
 [  1    0    2    2  964    2    4    3    7    9]
 [  1    1    0    3    0  863    3    0    4    3]
 [  2    2    2    0    2    7  942    0    0    1]
 [  1    1    5    5    3    2    0 1003    3    6]
 [  3    3    8    3    1    4    1    3  942    1]
 [  1    0    1    3    7    3    0    4    3  980]]

테스트 집합에 대한 정확률은 97.78%입니다.
```

4.10 하이퍼 매개변수 최적화

하이퍼 매개변수

- 모델의 구조와 모델의 학습 과정을 제어하는 역할. 예) 은닉 노드 개수, 식 (4.20)의 학습률, 미니 배치 크기 등
- [프로그램 4-3]의 경우 은닉 노드는 100개, 학습률은 0.001 ← 어떻게 결정했나?
- 이 절에서는 체계적으로 최적의 하이퍼 매개변수 값을 결정하는 방법을 공부

4.10.1 하이퍼 매개변수 살피기

- [프로그램 4-3] 11행의 MLPClassifier 함수는 6개의 매개변수를 가짐

```
mlp=MLPClassifier(hidden_layer_sizes=(100),learning_rate_init=0.001, batch_
size=32,max_iter=300,solver='sgd',verbose=True)
```

  - 앞의 5개는 하이퍼 매개변수
    - hidden_layer_sizes=(100): 100개 노드를 가진 은닉층 한 개를 둠(100개와 80개 노드를 가진 은닉층 두 개를 설정하려면 hidden_layer_sizes=(100,80)으로 함)
    - learning_rate_init=0.001: 식 (4.20)의 학습률 ρ를 0.001로 설정
    - batch_size=32: 미니 배치 크기를 32로 설정
    - max_iter=300: 최대 세대 수를 300으로 설정
    - solver='sgd': 최적화 알고리즘으로 스토캐스틱 경사 알고리즘을 사용

  - MLPClassifier 함수의 매개변수는 6개가 전부인가?

4.10.1 하이퍼 매개변수 살피기

함수의 API

- 예) MLPClassifier는 23개의 매개변수를 가짐
  - 파란색은 11행이 사용한 것들. 나머지는 기본값을 사용
    - 예) activation='relu'가 기본값이므로 활성 함수로 ReLU 사용
    - 예) shuffle=True가 기본값이므로 세대를 시작할 때마다 훈련 집합의 샘플 순서를 섞음

```
class sklearn.neural_network.MLPClassifier(hidden_layer_sizes=(100, ),
activation='relu', solver='adam', alpha=0.0001, batch_size='auto',
learning_rate='constant', learning_rate_init=0.001, power_t=0.5, max_
iter=200, shuffle=True, random_state=None, tol=0.0001, verbose=False, warm_
start=False, momentum=0.9, nesterovs_momentum=True, early_stopping=False,
validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08, n_iter_
no_change=10, max_fun=15000)
```

  - 11행에서 max_iter=300으로 설정했는데, 109 세대에서 학습을 멈춤
    - tol=0.0001과 n_iter_no_change=10 때문(10세대 동안 손실 함수 감소량이 0.0001 이하이면 멈추라는 뜻)

4.10.1 하이퍼 매개변수 살피기

- 하이퍼 매개변수 설정 가이드라인
  - 논문이나 공식 문서를 따라 함
  - 라이브러리 함수가 제공하는 기본값 사용
  - 중요한 하이퍼 매개변수를 골라 최적화

4.10.2 단일 하이퍼 매개변수 최적화: validation_curve 함수 이용

- [프로그램 4-5]는 은닉 노드 개수를 최적화

프로그램 4-5 validation_curve 함수로 최적의 은닉 노드 개수 찾기

```
01  from sklearn import datasets
02  from sklearn.neural_network import MLPClassifier
03  from sklearn.model_selection import train_test_split,validation_curve
04  import numpy as np
05  import matplotlib.pyplot as plt
06  import time
07
08  # 데이터셋을 읽고 훈련 집합과 테스트 집합으로 분할
09  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
```

훈련:테스트를 6:4로 분할

4.10.2 단일 하이퍼 매개변수 최적화: validation_curve 함수 이용

```
12  # 다층 퍼셉트론을 교차 검증으로 성능 평가 (소요 시간 측정 포함)
13  start=time.time()    # 시작 시각
14  mlp=MLPClassifier(learning_rate_init=0.001,batch_size=32,max_iter=300,solver='sgd')
15  prange=range(50,1001,50)
16  train_score,test_score=validation_curve(mlp,x_train,y_train,param_name="hidden_
    layer_sizes",param_range=prange,cv=10,scoring="accuracy",n_jobs=4)
17  end=time.time()      # 끝난 시각
18  print("하이퍼 매개변수 최적화에 걸린 시간은",end-start,"초입니다.")
19
20  # 교차 검증 결과의 평균과 분산 구하기
21  train_mean = np.mean(train_score,axis=1)
22  train_std = np.std(train_score,axis=1)
23  test_mean = np.mean(test_score,axis=1)
24  test_std = np.std(test_score,axis=1)
25
26  # 성능 그래프 그리기
27  plt.plot(prange,train_mean,label="Train score",color="r")
28  plt.plot(prange,test_mean,label="Test score",color="b")
29  plt.fill_between(prange,train_mean-train_std,train_mean+train_std,alpha=0.2,color="r")
30  plt.fill_between(prange,test_mean-test_std,test_mean+test_std,alpha=0.2,color="b")
31  plt.legend(loc="best")
32  plt.title("Validation Curve with MLP")
33  plt.xlabel("Number of hidden nodes"); plt.ylabel("Accuracy")
34  plt.ylim(0.9,1.01)
35  plt.grid(axis='both')
36  plt.show()
```

50에서 시작하여 50씩 증가시키면서 1000까지 조사

코어 4개를 사용하여 병렬 처리

10-겹 교차 검증으로 성능 측정

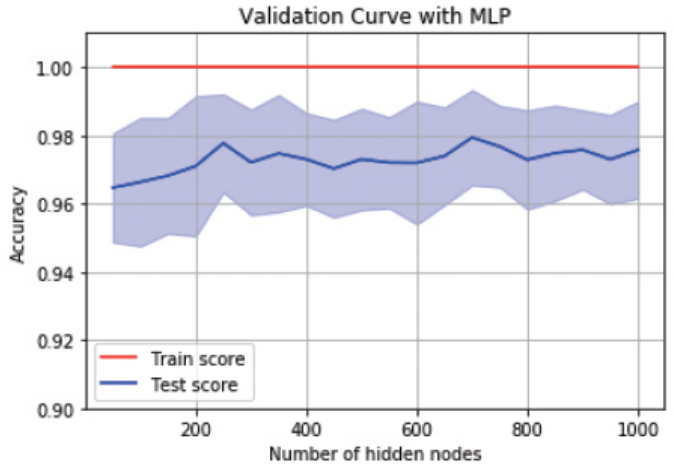
4.10.2 단일 하이퍼 매개변수 최적화: validation_curve 함수 이용

```
37
38  best_number_nodes=prange[np.argmax(test_mean)]     # 최적의 은닉 노드 개수
39  print("\n최적의 은닉층의 노드 개수는",best_number_nodes,"개입니다.\n")
40
41  # 최적의 은닉 노드 개수로 모델링
42  mlp_test=MLPClassifier(hidden_layer_sizes=(best_number_nodes),learning_rate_
    init=0.001,batch_size=32,max_iter=300,solver='sgd')
43  mlp_test.fit(x_train,y_train)
44
45  # 테스트 집합으로 예측
46  res=mlp_test.predict(x_test)
47
48  # 혼동 행렬
49  conf=np.zeros((10,10))
50  for i in range(len(res)):
51      conf[res[i]][y_test[i]]+=1
52  print(conf)
53
54  # 정확률 계산
55  no_correct=0
56  for i in range(10):
57      no_correct+=conf[i][i]
58  accuracy=no_correct/len(res)
59  print("테스트 집합에 대한 정확률은", accuracy*100, "%입니다.")
```

4.10.2 단일 하이퍼 매개변수 최적화: validation_curve 함수 이용

- 실행 결과

하이퍼 매개변수 최적화에 걸린 시간은 433.8679416179657초입니다.



최적의 은닉층의 노드 개수는 700개입니다.

```
[[65.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. 71.  0.  0.  0.  1.  0.  0.  3.  1.]
 [ 0.  0. 67.  0.  0.  1.  0.  0.  0.  0.]
 [ 0.  0.  0. 64.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. 78.  0.  0.  0.  0.  0.]
 [ 2.  0.  0.  0.  0. 80.  0.  0.  2.  0.]
 [ 0.  0.  0.  0.  0.  1. 78.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0. 64.  0.  0.]
 [ 0.  0.  0.  1.  0.  0.  0.  0. 74.  1.]
 [ 0.  0.  0.  0.  0.  0.  0.  1.  0. 64.]]
```

테스트 집합에 대한 정확률은 98.0528511821975%입니다.

5.1 딥러닝의 등장

- 1980년대의 깊은 신경망
  - 구조적으로는 쉬운 개념
    - 다층 퍼셉트론에 은닉층을 많이 두면 깊은 신경망
  - 하지만 학습이 잘 안됨
    - 그레이디언트 소멸 문제
    - 작은 데이터셋 문제(추정할 매개변수가 많아지는데 데이터는 적어 과잉 적합 발생)
    - 과다한 계산 시간(비싼 슈퍼컴퓨터)

5.1.1 딥러닝의 기술 혁신

- 딥러닝은 새로 창안된 이론이나 원리는 빈약
  - 신경망의 구조와 동작, 학습 알고리즘의 기본 원리는 거의 그대로
- 딥러닝의 기술 혁신 요인
  - 값싼 GPU 등장
    - 10~100배의 속도 향상으로 학습 시간 단축
  - 데이터셋 커짐
    - 인터넷을 통한 데이터 수집과 레이블링(예, 1400만장을 담은 ImageNet)
  - 학습 알고리즘의 발전
    - ReLU 활성 함수
    - 규제 기법(가중치 감쇠, 드롭아웃, 조기 멈춤, 데이터 증대, 앙상블 등)
    - 다양한 손실 함수와 옵티마이저 개발

5.1.1 딥러닝의 기술 혁신

- 딥러닝으로 인한 인공지능의 획기적 발전
  - 2010년대에 딥러닝의 성공 사례 발표(예, AlexNet)
  - 고전적인 기계 학습을 사용하던 연구 그룹이 딥러닝으로 전환
  - 낮은 성능 때문에 대학 실험실에 머물던 프로토타입 시스템에 획기적인 성능 향상
  - 뛰어난 인공지능 제품이 시장에 속속 등장하여 '인공지능 붐' 조성
  - 딥러닝은 인공지능을 구현하는 핵심 기술로 자리잡음

5.1.1 딥러닝의 기술 혁신

- 학술적인 측면의 혁신 사례
  - 컨볼루션 신경망이 딥러닝의 가능성을 엶
    - 작은 크기의 컨볼루션 마스크를 사용하여 우수한 특징을 추출
    - 1990년대 르쿤은 필기 숫자에서 획기적 성능 향상(수표 자동인식 시스템)
  - AlexNet은 컨볼루션 신경망으로 자연 영상 인식이 가능하다는 사실을 보여줌
    - 2012년 ILSVRC 대회에서 15.3% 오류율이라는 당시 경이로운 성능으로 우승 차지
    - 이후 컴퓨터 비전 연구는 고전적인 기계학습에서 딥러닝으로 대전환

그림 5-2 자연 영상의 심한 변화 예(ImageNet의 고양이 부류 사진)

  - 음성 인식에서 혁신
    - 힌튼 교수는 딥러닝을 적용하여 오류율을 단숨에 20%만큼 줄임
    - "10년 걸릴 일을 단번에 이루었다. … 10개의 기술 혁신이 한꺼번에 일어났다"라고 자평

5.1.1 딥러닝의 기술 혁신

NOTE ImageNet과 ILSVRC 대회

자연 영상을 분류하는 문제는 ImageNet이라는 데이터베이스가 만들어진 이후에 다시 주목받기 시작했다. ImageNet은 WordNet의 계층적 단어 분류 체계에 따라 부류를 정하고 약 2만 부류에 대해 각각 500~1,000장의 영상을 인터넷에서 수집해 구축하였다[Deng2009]. ILSVRC는 ImageNet에서 1,000개의 부류를 뽑아 분류 문제를 푸는 대회이다[Russakovsky2015]. 총 120만 장의 훈련 집합, 5만 장의 검증 집합, 15만 장의 테스트 집합이 주어진다. [그림 5-2]는 1,000개의 부류 중 'cat' 부류의 예제 영상인데, 같은 부류 안에서 변화가 아주 심하다는 것을 확인할 수 있다. 만일 신경망이 cat 부류에 속하는 영상을 보고 'cat(0.9), dog(0.1), …'이라고 출력하면 1순위로 맞힌 것으로 간주한다. 괄호 속 숫자는 해당 부류에 속할 확률이다. 그리고 만약 'dog(0.5), bear(0.3), swing(0.1), cat(0.09), abacus(0.02), Great white shark(0.01), …'이라고 출력하면 정답 부류가 5순위 안에 있으므로 5순위로 맞힌 것으로 간주한다. ILSVRC는 1순위 오류율과 5순위 오류율 두 가지로 성능을 측정한다. 2012년에 AlexNet은 5순위 오류율 15.3%를 달성했다. 그리고 2015년에는 마이크로소프트 팀에서 지름길 연결이라는 아이디어를 구현한 ResNet이 3.5%의 5순위 오류율로 우승했다.

5.1.2 딥러닝 소프트웨어

대표적인 딥러닝 소프트웨어

- 현재는 텐서플로와 파이토치가 대세
- 대략 텐서플로는 기업, 파이토치는 대학 연구자들이 많이 사용

표 5-1 딥러닝 소프트웨어

이름	개발 그룹	최초 공개일	작성 언어	인터페이스 언어	전이학습 지원	철저한 관리
씨아노(Theano)	몬트리올 대학교	2007년	파이썬	파이썬	O	X
카페(Caffe)	UC버클리	2013년	C++	파이썬, 매트랩, C++	O	X
텐서플로 (TensorFlow)	구글 브레인	2015년	C++, 파이썬, CUDA	파이썬, C++, 자바, 자바스크립트, R, Julia, Swift, Go	O	O
케라스(Keras)	프랑소와 숄레 (François Chollet)	2015년	파이썬	파이썬, R	O	O
파이토치(PyTorch)	페이스북	2016년	C++, 파이썬, CUDA	파이썬, C++	O	O

TIP 딥러닝 라이브러리를 보다 폭넓게 살펴보려면 위키피디어에서 'comparison of deep-learning software'를 검색한다.

5.2.2 텐서 이해하기

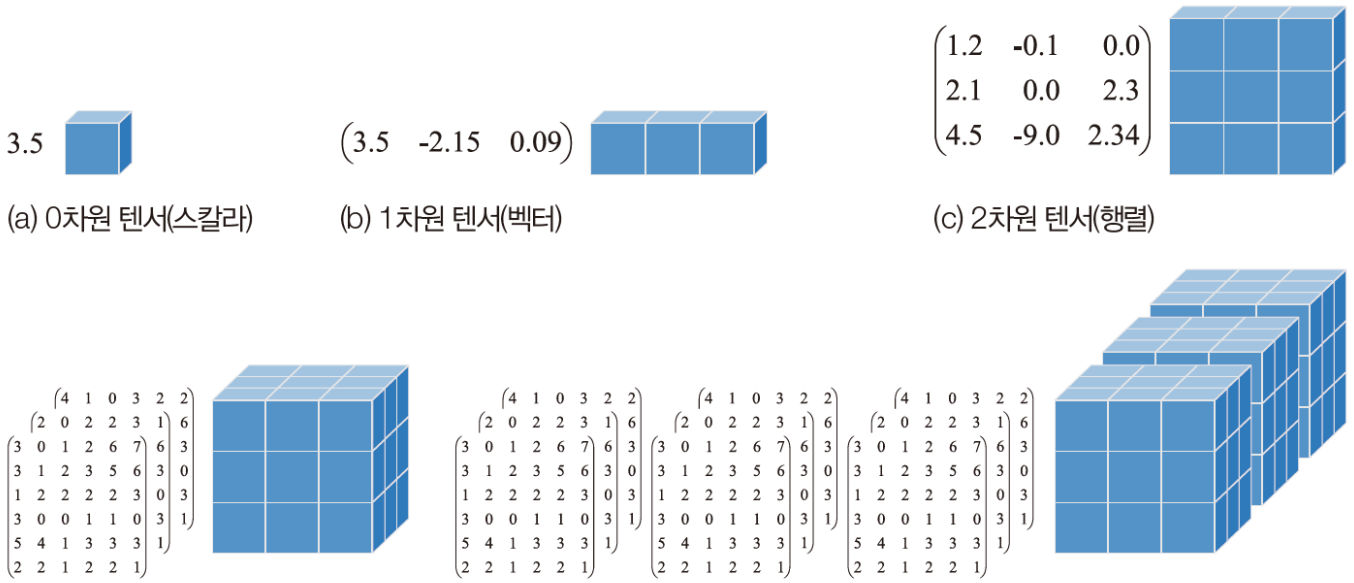
딥러닝에서 텐서

- 다차원 배열을 텐서라 부름
  - 데이터를 텐서로 표현
  - 신경망의 가중치(매개변수)를 텐서로 표현
- 넘파이는 ndarray 클래스, 텐서플로는 Tensor 클래스로 표현. 둘은 호환됨

TIP 특징 벡터의 차원과 텐서의 차원을 구별해야 한다. 예를 들어 iris 데이터의 경우 샘플 하나를 (꽃잎의 길이, 꽃잎의 너비, 꽃받침의 길이, 꽃받침의 너비)의 특징 벡터로 표현한다. 이 특징 벡터는 요소가 4개이므로 4차원이라고 말한다. 하지만 이 특징 벡터는 한 방향으로 길쭉한 [그림 5-4(b)]의 1차원 모양 텐서다. 엄밀하게는 1차원 구조의 텐서(tensor with 1-dimensional shape)라고 해야 하지만 줄여서 1차원 텐서라 부른다.

5.2.2 텐서 이해하기

- 0~4차원 구조의 텐서의 예
  - 1차원: iris 샘플 하나
  - 2차원: iris 샘플 여러 개, 명암 영상 한 장
  - 3차원: 명암 영상 여러 장, 컬러 영상 한 장
  - 4차원: 컬러 영상 여러 장, 컬러 동영상 하나
  - 5차원: 컬러 동영상 여러 개



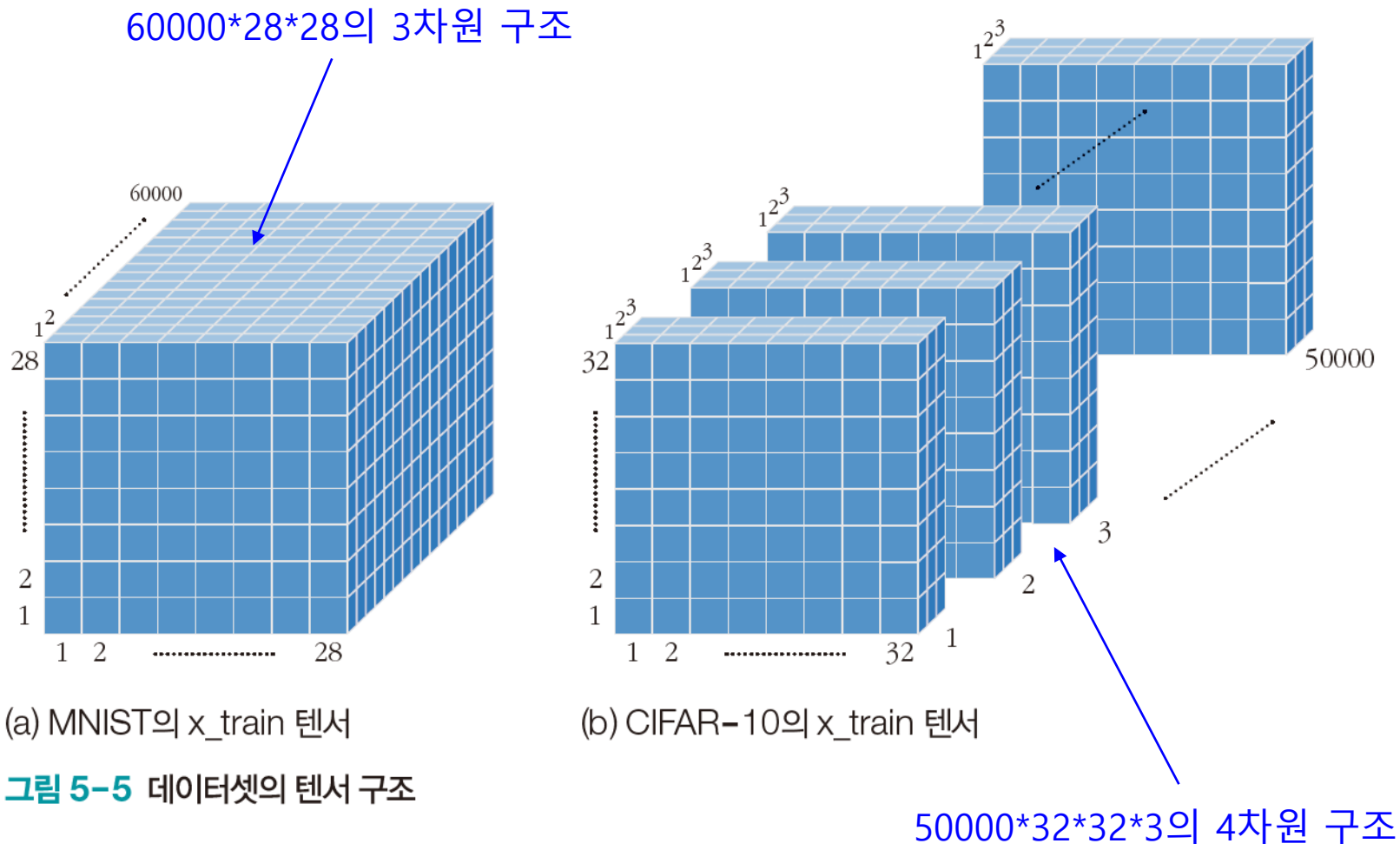
(a) 0차원 텐서(스칼라) (b) 1차원 텐서(벡터) (c) 2차원 텐서(행렬)

(d) 3차원 텐서 (e) 4차원 텐서

그림 5-4 텐서의 구조

5.2.2 텐서 이해하기

- 텐서플로가 제공하는 데이터셋의 텐서 구조



(a) MNIST의 x_train 텐서

(b) CIFAR-10의 x_train 텐서

그림 5-5 데이터셋의 텐서 구조

5.5 깊은 다층 퍼셉트론

- 다층 퍼셉트론에 은닉층을 더 많이 추가하면 깊은 다층 퍼셉트론
  - 깊은 다층 퍼셉트론은 가장 쉽게 생각할 수 있는 딥러닝 모델

5.5.1 구조와 동작

- 깊은 다층 퍼셉트론DMLP(deep MLP)의 구조
  - L-1개의 은닉층이 있는 L층 신경망. 입력층에 d+1개의 노드, 출력층에 c개의 노드. i번째 은닉층에 n_i개의 노드 (n_i는 하이퍼 매개변수)
  - 인접한 층은 완전 연결, 즉 FC(fully-connected) 구조. 아주 많은 가중치: 예) n_i=500이고 L=5라면, MNIST데이터에서 (784+1)*500+(500+1)*500*3+(500+1)*10=1,149,010개의 가중치

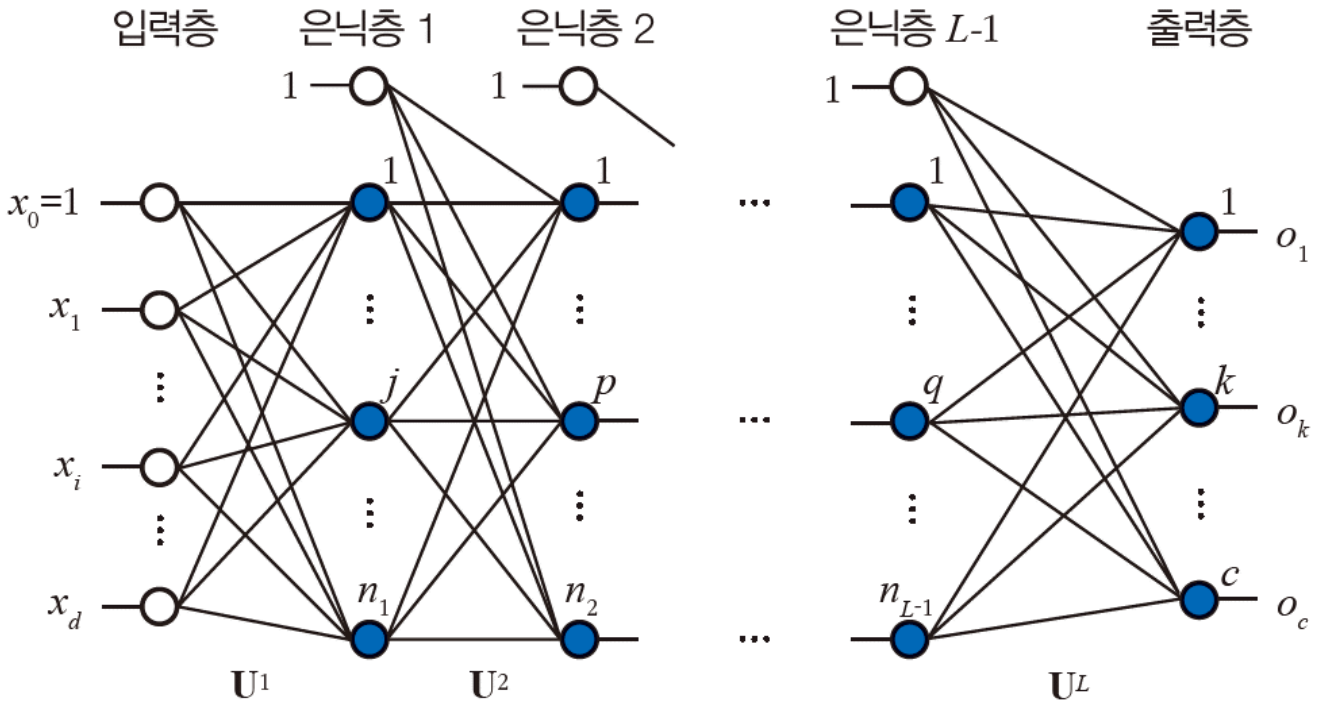


그림 5-8 깊은 다층 퍼셉트론의 구조

5.5.1 구조와 동작

깊은 다층 퍼셉트론의 동작

- 식 (5.1)은 l-1번째 층과 l번째 층을 연결하는 가중치 행렬
  - u_{ji}^{l}은 l-1번째 층의 i번째 노드와 l번째 층의 j번째 노드를 연결하는 가중치

$$\mathbf{U}^{l}=\begin{pmatrix} u_{10}^{l} & u_{11}^{l} & \cdots & u_{1n_{l-1}}^{l} \\ u_{20}^{l} & u_{21}^{l} & \cdots & u_{2n_{l-1}}^{l} \\ \vdots & \vdots & \ddots & \vdots \\ u_{n_l 0}^{l} & u_{n_l 1}^{l} & \cdots & u_{n_l n_{l-1}}^{l} \end{pmatrix}, \quad l=1,2,\cdots,L \qquad (5.1)$$

- 입력층으로 들어오는 특징 벡터

$$\mathbf{z}^{0}=\left(z_0^{0},z_1^{0},\cdots,z_{n_0}^{0}\right)=\left(1,x_1,x_2,\cdots,x_d\right) \qquad (5.2)$$

5.5.1 구조와 동작

- 깊은 다층 퍼셉트론의 동작
  - l번째 층의 j번째 노드가 수행하는 연산

l번째 은닉층의 j번째 노드의 연산:

$$z_j^l = \tau_l\left(s_j^l\right)$$

이때 $s_j^l = \mathbf{u}_j^l \mathbf{z}^{l-1}$이고 $\mathbf{u}_j^1 = \left(u_{j0}^l, u_{j1}^l, \cdots, u_{jn_{l-1}}^l\right), \mathbf{z}^{l-1} = \left(1, z_1^{l-1}, z_2^{l-1}, \cdots, z_{n_{l-1}}^{l-1}\right)^{\mathrm{T}}$ (5.3)

  - l번째 층의 연산을 행렬 표현으로 쓰면

l번째 층의 연산: $\mathbf{z}^l = \tau_l(\mathbf{U}^l \mathbf{z}^{l-1}),\ l = 1, 2, \cdots, L$ (5.4)

- 훈련 집합 전체에 대한 연산

$$\mathbf{O} = \tau_L\left(\cdots\cdots \tau_2\left(\mathbf{U}^2 \tau_1\left(\mathbf{U}^1 \mathbf{X}^{\mathrm{T}}\right)\right)\right) \quad (5.5)$$

  - 1,2,3,…,L-1층의 활성 함수는 주로 ReLU, L층(출력층)은 softmax 사용

5.5.2 오류 역전파 알고리즘

- 다층 퍼셉트론(4.8절)의 학습 알고리즘을 조금 확장
  - 식 (5.6)은 손실 함수

$$J\left(\mathbf{U}^1,\mathbf{U}^2,\cdots,\mathbf{U}^L\right)=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\left\|\mathbf{y}-\mathbf{o}\right\|^2$$
$$=\frac{1}{|M|}\sum_{\mathbf{x}\in M}\left\|\mathbf{y}-\tau_L\left(\cdots\tau_2\left(\mathbf{U}^2\tau_1\left(\mathbf{U}^1\mathbf{x}^T\right)\right)\right)\right\|^2 \qquad (5.6)$$

  - 식 (5.7)은 가중치 갱신 규칙

$$\mathbf{U}^l=\mathbf{U}^l+\rho\left(-\nabla\mathbf{U}^l\right),\ l=L,L-1,\cdots,1 \qquad (5.7)$$

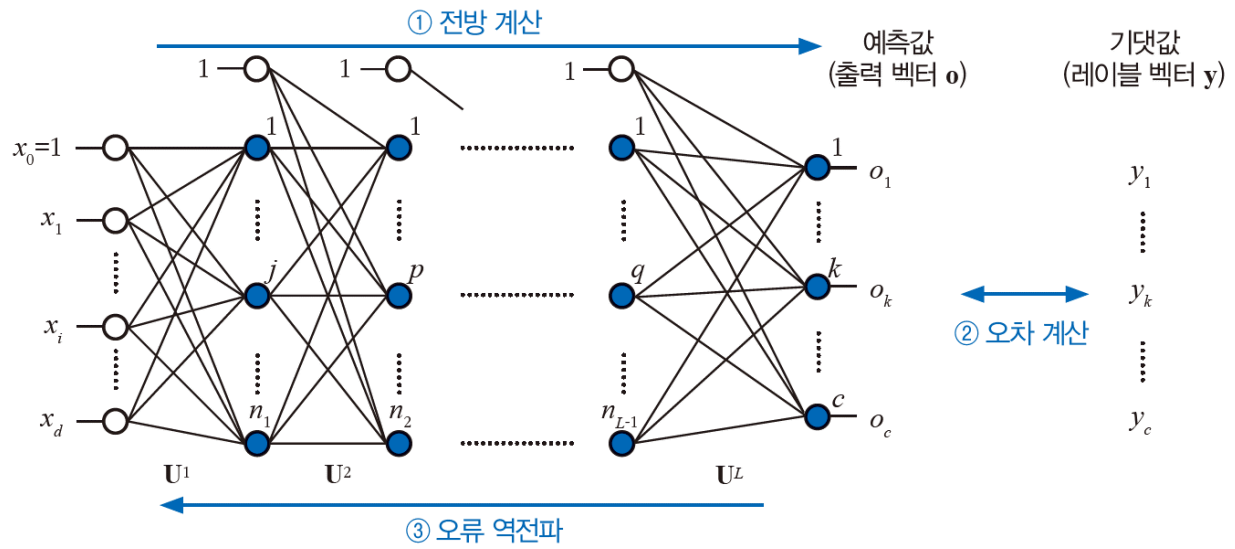


그림 5-9 깊은 다층 퍼셉트론이 사용하는 오류 역전파 알고리즘

5.6 딥러닝의 학습 전략

- 깊은 다층 퍼셉트론의 학습 알고리즘인 식 (5.7)은 수학적으로 아주 깔끔
  - 코딩도 깔끔
- 하지만 층이 깊어지면 현실적인 문제 발생
  - 이 장에서는 대표적인 두가지 문제 제시하고 해결 전략 설명
    - 그레이디언트 소멸 문제
    - 과잉 적합 문제

5.6.1 그레이디언트 소멸 문제와 해결책

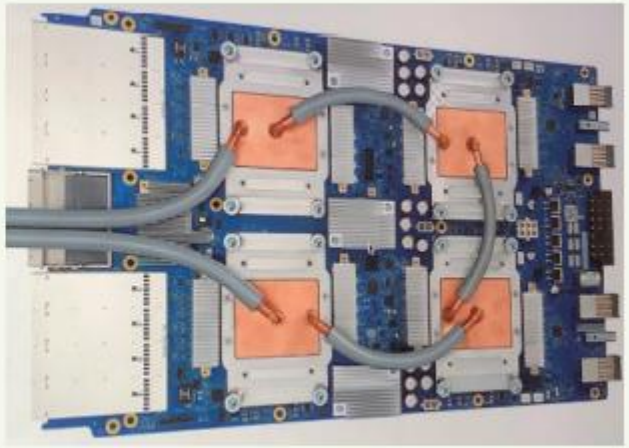
- 미분의 연쇄 법칙chain rule에 따르면,
  - I번째 층의 그레이디언트는 오른쪽에 있는 I+1번째 층의 그레이디언트에 자신 층에서 발생한 그레이디언트를 곱하여 구함
  - 따라서 그레이디언트가 0.001처럼 작은 경우 왼쪽으로 진행하면서 점점 작아짐
  - 왼쪽으로 갈수록 가중치 갱신이 느려져서 전체 신경망의 학습이 매우 느린 현상이 발생
- 병렬 처리로 해결
  - GPU 사용 또는 colab에서 tpu 설정

그림 5-10 코랩에서 GPU 또는 TPU 사용하기

5.6.1 그레이디언트 소멸 문제와 해결책

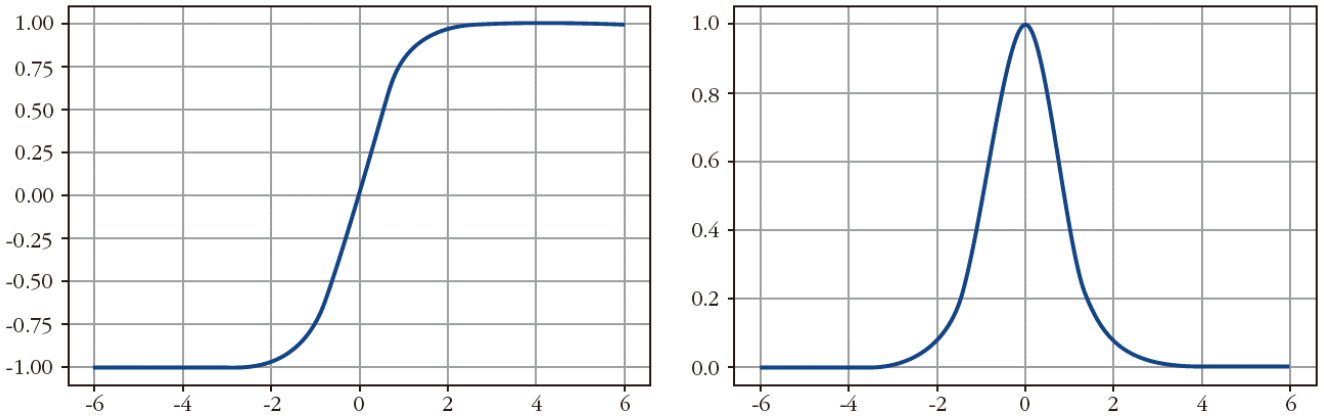
NOTE TPU

오른쪽 사진은 TPU(Tensor Processing Unit)이다. TPU는 구글이 신경망 학습을 빠르게 할 목적으로 개발한 기계 학습 전용 병렬 처리 기계이다. GPU는 원래 그래픽 가속기로 개발되었다는 점에서 차이가 있다. TPU는 그래픽 처리 측면에서는 GPU보다 열등하지만 기계 학습 측면에서는 GPU보다 뛰어나다. TPU는 텐서플로 소프트웨어에 맞추어 개발되었기 때문에 텐서플로로 개발하는 사람에겐 희소식이다. 구글에서는 알파고를 TPU로 학습했다고 밝혔다. 아직 정식 시판은 되지 않았으나, https://coral.ai에서 코랄 보드라는 초기 제품을 10~20만 원에 구입할 수 있다.

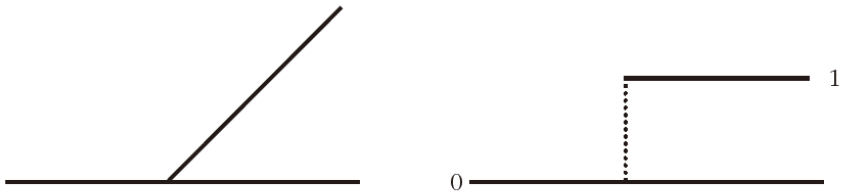


5.6.1 그레이디언트 소멸 문제와 해결책

- ReLU 함수를 사용하여 해결
  - Tanh(s) 시그모이드 함수의 문제점
    - s가 클 때 그레이디언트가 0에 가까워짐(s=8이면 그레이디언트 값은 0.0000004501)
  - ReLU는 s가 음수일 때 그레이디언트는 0, 양수일 때 1



(a) tanh 시그모이드와 그레이디언트



(b) ReLU와 그레이디언트

그림 5-11 tanh 시그모이드와 ReLU의 그레이디언트 특성 비교

5.6.2 과잉 적합과 과잉 적합 회피 전략

- [그림 5-12]는 과소 적합과 과잉 적합을 설명
  - x는 특징이고 y는 레이블인 회귀 문제로 설명
  - 모델로 1차 다항식을 사용하면 과소 적합$_{\text{under fitting}}$(데이터에 비해 모델 용량이 작은 상황)
  - 용량이 가장 큰 12차 다항식은 훈련 집합에 대해 가장 적은 오류

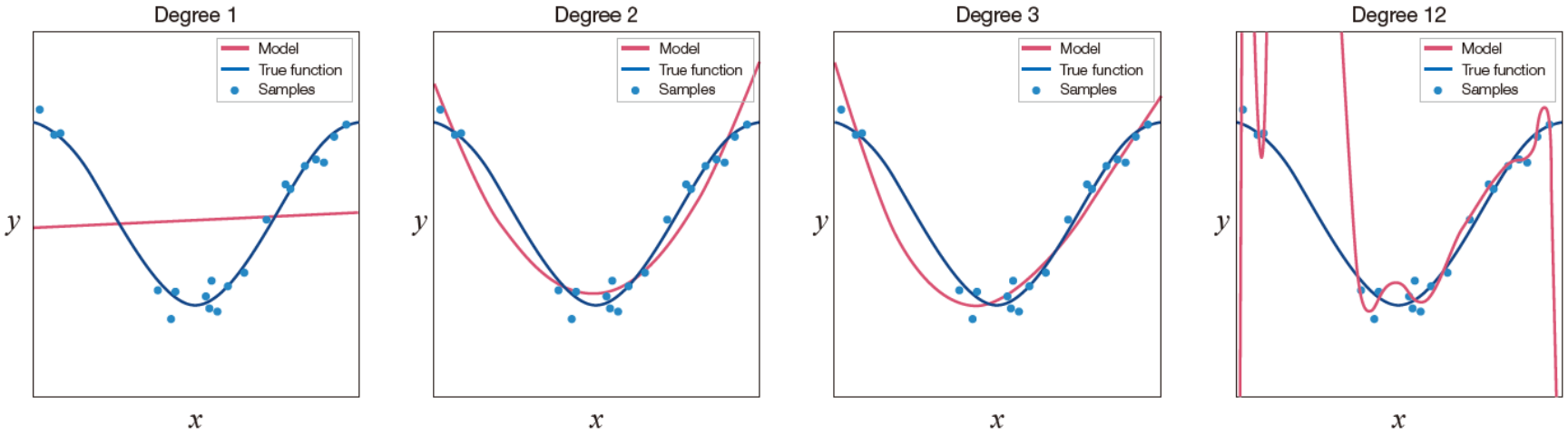


그림 5-12 과소 적합과 과잉 적합

5.6.2 과잉 적합과 과잉 적합 회피 전략

- 12차 다항식 모델은 일반화 능력이 떨어짐([그림 5-13])
  - 예를 들어, x_0에서 부정확한 예측
  - 데이터의 복잡도에 비해 너무 큰 용량의 모델을 사용한 탓 ← 과잉 적합$_{\text{over fitting}}$ 현상

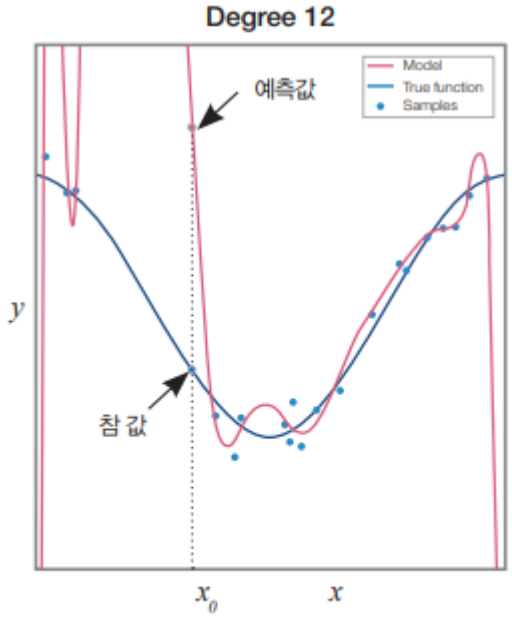
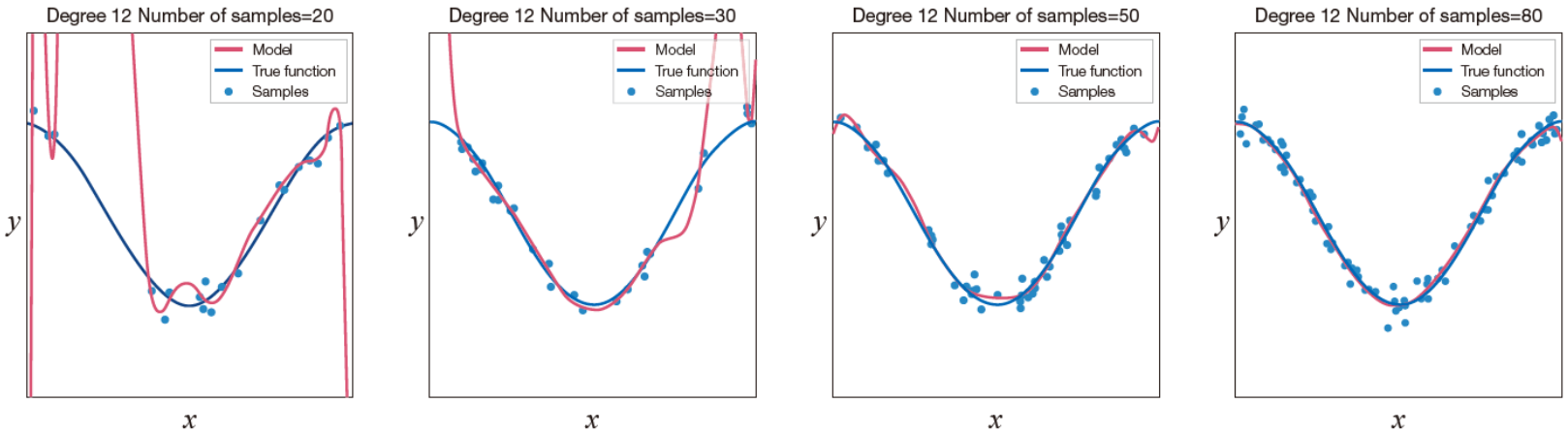


그림 5-13 과잉 적합에 따른 부정확한 예측

5.6.2 과잉 적합과 과잉 적합 회피 전략

딥러닝의 과잉 적합 회피 전략

- 데이터 양을 늘림. 데이터 양을 늘릴 수 없는 상황에서는 훈련 샘플을 변형하여 인위적으로 늘리는 데이터 증대data augmentation 사용



(a) 훈련 집합 크기: 20 (b) 훈련 집합 크기: 30 (c) 훈련 집합 크기: 50 (d) 훈련 집합 크기: 80

그림 5-14 데이터 양을 늘려 과잉 적합을 해소

- 규제 기법 적용
  - 데이터 증대, 가중치 감쇠, 드롭아웃, 앙상블 등

5.6.2 과잉 적합과 과잉 적합 회피 전략

NOTE 분류와 회귀 문제

분류classification 문제는 레이블이 이산인 경우이다. MNIST 숫자 인식, fashion MNIST 인식, iris 인식 등은 모두 분류 문제이다. 레이블이 연속인 경우를 분류와 구별해 회귀regression 문제라 부른다. 비트코인 가격을 예측하는 문제는 가격이 연속된 값을 가지므로 회귀이다.

예전 통계학자들은 회귀 문제를 푸는 일반화 선형 모델generalized linear model을 개발했는데, 이 모델은 나중에 분류 문제를 풀 수 있는 로지스틱 회귀logistic regression 모델로 개선되었다. 기계 학습 연구자는 주로 분류 문제를 푸는 알고리즘을 개발한다. 3장에서 공부한 SVM, 지금 공부하고 있는 신경망은 모두 분류 문제를 푸는 용도로 개발되었다. 회귀를 푸는 일반화 선형 모델을 분류 문제를 푸는 로지스틱 회귀 모델로 개조해 사용하듯이, 분류 문제를 푸는 SVM을 회귀 문제를 푸는 SVM으로 개조할 수 있다.

sklearn 라이브러리에서는 분류 문제를 푸는 SVM을 SVC, 회귀 문제를 푸는 SVM을 SVR이라는 함수로 구분해 제공한다.

5.7 딥러닝이 사용하는 손실 함수

시험 점수의 역할

- 점수가 낮은 학생에게 F학점 또는 낙방과 같은 벌점을 부여하면 자신을 성찰하고 더 열심히 공부할 동기 부여
- 점수가 낮거나 높거나 비슷한 벌점을 받으면 공정성이 깨지고 공부 의욕을 꺾음

신경망 학습도 비슷

- 신경망 가중치가 학생, 손실 함수가 시험 점수에 해당

5.7.1 평균제곱오차

- 샘플 하나의 오류
  - 레이블 **y**와 신경망이 예측한 값 **o**의 차이

$$e = \| \mathbf{y} - \mathbf{o} \|^2 \qquad (5.8)$$

- 평균제곱오차MSE(mean-squared error)
  - 통계학에서 오랫동안 사용해온 식을 기계 학습이 빌려다 쓰는 셈

$$J\left(\mathbf{U}^1, \mathbf{U}^2, \cdots, \mathbf{U}^L\right) = \frac{1}{|M|} \sum_{\mathbf{x} \in M} \|\mathbf{y} - \mathbf{o}\|^2$$
$$= \frac{1}{|M|} \sum_{\mathbf{x} \in M} \left\| \mathbf{y} - \tau_L\left(\cdots \tau_2\left(\mathbf{U}^2 \tau_1\left(\mathbf{U}^1 \mathbf{x}^T\right)\right)\right)\right\|^2 \qquad (5.6)$$

- 평균제곱오차의 문제점
  - 교정에 사용하는 값, 즉 그레이디언트가 벌점에 해당. 오차 e가 더 큰데 그레이디언트가 더 작은 상황이 발생(공부를 못하는 학생이 더 높은 점수를 받는 상황에 비유)
  - 학습이 느려지거나 학습이 안되는 상황을 초래할 가능성

5.7.2 교차 엔트로피

TIP 엔트로피는 섀넌이 제안한 정보 이론에서 유래한다. 섀넌에 대해서는 3장의 '재미있는 인공지능 이야기'를 참조한다.

엔트로피 entropy

- 확률 분포의 무작위성(불확실성)을 측정하는 함수 (식 (5.9))
  - 공정한 주사위의 엔트로피는 찌그러진 주사위보다 높음(예, 1의 면적이 더 넓은 찌그러진 주사위는 불확실성이 낮아짐)
  - 예를 들어, 공정한 주사위의 엔트로피

$$-\left(\frac{1}{6}\log\frac{1}{6}+\cdots+\frac{1}{6}\log\frac{1}{6}\right)=1.7918$$

$$H(x)=-\Sigma_{i=1,k}P(e_i)\log P(e_i) \qquad (5.9)$$

교차 엔트로피 cross entropy

- 두 확률 분포 P와 Q가 다른 정도를 측정하는 함수(식 (5.10))

$$H(P,Q)=-\sum_{i=1,k}P(e_i)\log Q(e_i) \qquad (5.10)$$

- 예를 들어,
  - 공정한 주사위 P와 Q의 교차 엔트로피 $-\left(\frac{1}{6}\log\frac{1}{6}+\ldots+\frac{1}{6}\log\frac{1}{6}\right)=1.7918$
  - 공정한 주사위 P와 찌그러진 주사위 Q(1이 ½, 나머지는 1/10확률)의 교차 엔트로피

$$-\left(\frac{1}{6}\log\frac{1}{2}+\frac{1}{6}\log\frac{1}{10}+\cdots+\frac{1}{6}\log\frac{1}{10}\right)=2.0343$$

5.7.2 교차 엔트로피

- 교차 엔트로피 손실 함수(식 (5.11))
  - 교차 엔트로피는 평균제곱오차의 불공정성 문제를 해결해 줌
  - 딥러닝은 주로 교차 엔트로피를 사용

$$\text{교차 엔트로피 손실 함수: } e = -\sum_{i=1,c} y_i \log o_i \qquad (5.11)$$

[예제 5-1] 교차 엔트로피의 합리성 확인

숫자 인식에서 부류 3에 속하는 샘플의 경우 $\mathbf{y}=(0,0,0,1,0,\cdots,0)$이다. 신경망의 출력이 $\mathbf{o}=(0.1,0,0,0.9,0,\cdots,0)$이라 하자. 부류 3에 해당하는 곳이 0.9로서 가장 크므로 신경망이 샘플을 맞힌 경우이다. 식 (5.11)을 계산하면 $e=-(0\times\log(0.1)+0\times\log(0)+0\times\log(0)+1\times\log(0.9)+\cdots+0\times\log(0))=0.1054$가 된다.

이제 신경망이 $\mathbf{o}=(0,0.9,0,0.1,0,\cdots,0)$을 출력했다고 가정하자. 부류 1에 해당하는 곳이 가장 큰 값을 갖기 때문에 신경망이 틀린 경우이다. 이 경우 $e=-(0\times\log(0)+0\times\log(0.9)+0\times\log(0)+1\times\log(0.1)+\cdots+0\times\log(0))=2.3026$이 된다. 후자의 틀린 경우에서 손실 함수 값이 훨씬 큰 사실을 확인할 수 있다.

5.8 딥러닝이 사용하는 옵티마이저

- 손실 함수의 최저점을 찾아주는 옵티마이저
  - 표준에 해당하는 SGD 옵티마이저([그림 4-5])를 개선하는 두 가지 아이디어
    - 모멘텀momentum
    - 적응적 학습률adaptive learning rate

5.8.1 모멘텀을 적용한 옵티마이저

- 물리에서 모멘텀
  - 이전 운동량을 현재에 반영(관성과 관련)
  - 옵티마이저에 적용하면 뚜렷한 성능 향상
- 모멘텀의 원리
  - 모멘텀에서는 이전 방향 정보 **v**를 같이 고려(α는 [0,1]사이에서 조절)
    - α=0는 고전적 SGD, α가 1에 가까울수록 이전 정보에 큰 가중치 부여
    - 보통 α=0.5, 0.9를 사용

$$\text{고전적 SGD: } \mathbf{w} = \mathbf{w} - \rho\frac{\partial J}{\partial \mathbf{w}} \rightarrow \text{모멘텀을 적용한 SGD: } \left.\begin{aligned} \mathbf{v} &= \alpha\mathbf{v} - \rho\frac{\partial J}{\partial \mathbf{w}} \\ \mathbf{w} &= \mathbf{w} + \mathbf{v} \end{aligned}\right\} \quad (5.12)$$

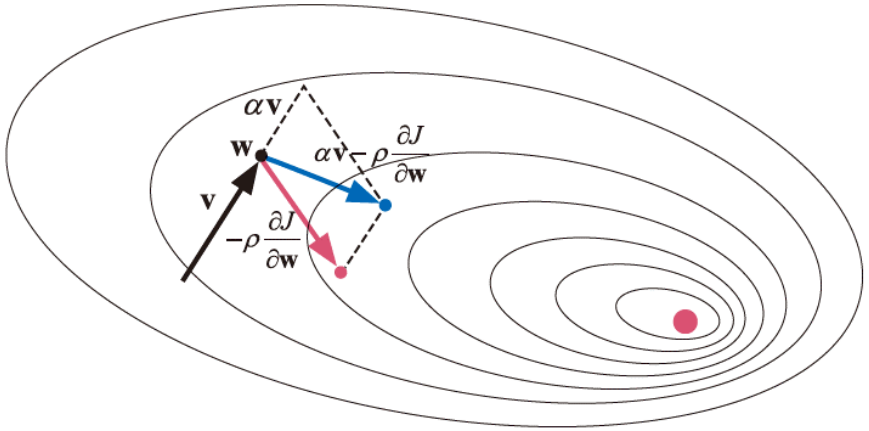
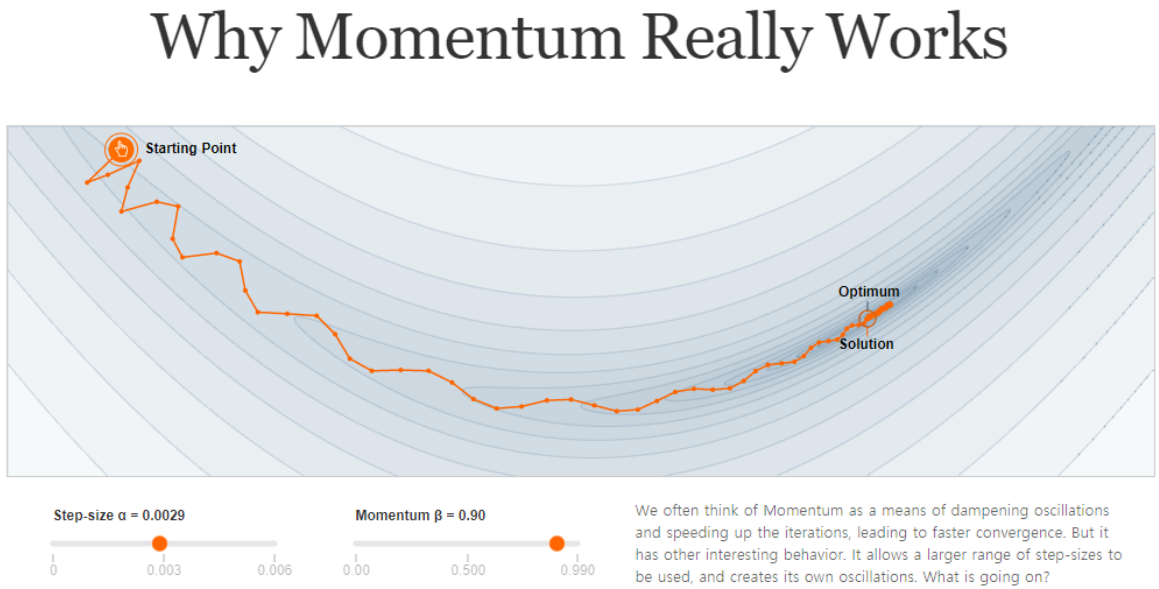


그림 5-15 모멘텀의 원리와 효과

5.8.1 모멘텀을 적용한 옵티마이저

- 네스테로프 모멘텀
  - 현재 점 **w**에서 미분하는 대신, 이전 정보 $\alpha\mathbf{v}$를 이용하여 다음에 이동할 곳 $\hat{\mathbf{w}}$을 예측하고 그곳에서 그레이디언트를 계산
- 모멘텀 효과를 시각화하는 사이트

TIP https://distill.pub/2017/momentum에 접속하면 모멘텀 계수 α를 변경해봄으로써 최저점 탐색 과정과 결과가 어떻게 달라지는지 애니메이션으로 확인할 수 있다.



5.8.1 모멘텀을 적용한 옵티마이저

텐서플로에서 모멘텀 적용

- 기본값은 모멘텀 적용 않고 네스테로프 적용 안함

[SGD 옵티마이저의 API]

```
tensorflow.keras.optimizers.SGD(learning_rate=0.01,momentum=0.0,nesterov
                                =False,name='SGD',**kwargs)
```

- 만일 학습률 0.0001, 모멘텀 0.9, 네스테로프 적용하려면
  - tensorflow.keras. optimizer.SGD(learning_rate=0.0001, momentum=0.9, nesterov=True)로 호출

5.8.2 적응적 학습률을 적용한 옵티마이저

- 식 (5.12)의 학습률
  - 그레이디언트는 최저점의 방향은 알려주지만 이동량에 대한 정보는 없기 때문에 작은 학습률을 곱해 조금씩 보수적으로 이동
  - 학습률이 너무 작으면 학습에 많은 시간 소요. 너무 크면 진동 가능성

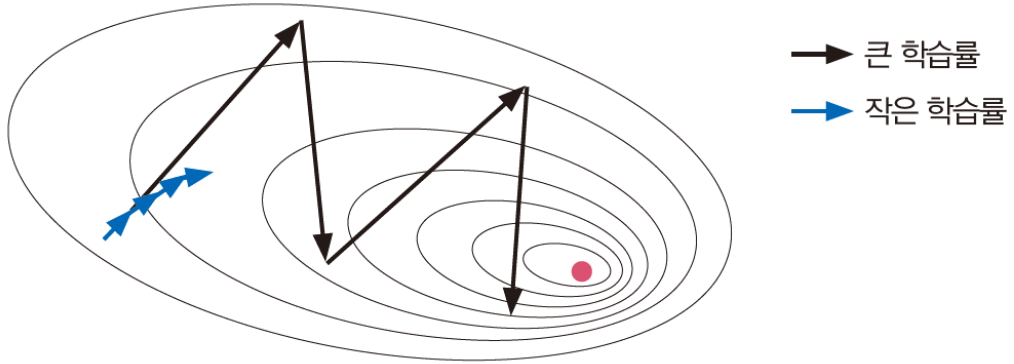


그림 5-16 학습률에 따른 수렴 특성

- 적응적 학습률
  - 상황에 맞게 학습률을 조절하는 방법

• Adagrad: 이전 그레이디언트를 누적한 정보를 이용하여 학습률을 적응적으로 설정하는 기법

• RMSprop: 이전 그레이디언트를 누적할 때 오래된 것의 영향을 줄이는 정책을 사용하여 AdaGrad를 개선한 기법

• Adam: RMSProp에 식 (5.12)의 모멘텀을 적용하여 RMSprop을 개선한 기법

5.9 좋은 프로그래밍 스킬

■ 프로그래밍 스킬의 중요성

- 프로그래밍을 못하면 아무런 인공지능 프로그램도 만들 수 없음
- 프로그래밍이 미숙하면 좋은 아이디어보다 디버깅에 에너지 소진
- 프로그래밍은 인공지능에서 충분조건은 아니지만 핵심 필요조건

1. 모듈화하라.

- [프로그램 5-11]의 build_model 함수가 좋은 사례
- 26~27행의 batch_siz와 n_epoch 상수(상수로 정의해 놓고 여러 군데에서 활용)

2. 언어의 좋은 특성을 최대한 활용하라.

- [프로그램 5-11]의 60행 사례(②의 두 행을 간결하게 ①의 한 행으로 코딩)

```
print("SGD 정확률은",dmlp_sgd.evaluate(x_test,y_test,verbose=0)[1]*100)
```

①

```
res_sgd=dmlp_mse.evaluate(x_test,y_test,verbose=0)
print("평균제곱오차의 정확률은",res_sgd[1]*100)
```

②

5.9 좋은 프로그래밍 스킬

3. 점증적으로 코딩하라.
   - 한번에 한가지 기능을 추가하고 옳게 작동하는지 확인하는 일을 반복
   - [프로그램 5-11]의 그래프 그리는 68~82행

4. 디자인 패턴을 몸에 배게 하라.
   - 다른 프로그램과 공유하는 디자인 패턴에 대한 눈썰미

5. 도구에 한없이 익숙해져라.
   - Colab, VS Code, Pycharm, 스파이더 사용법에 익숙
   - 라이브러리 사용에 익숙

6. 기초에 충실하라.
   - 파이썬의 기초 자료구조인 리스트, 튜플, 딕셔너리
   - 중요한 라이브러리인 numpy
   - 기계 학습의 기초 이론 등