

영상처리 실제

Image Processing Practice

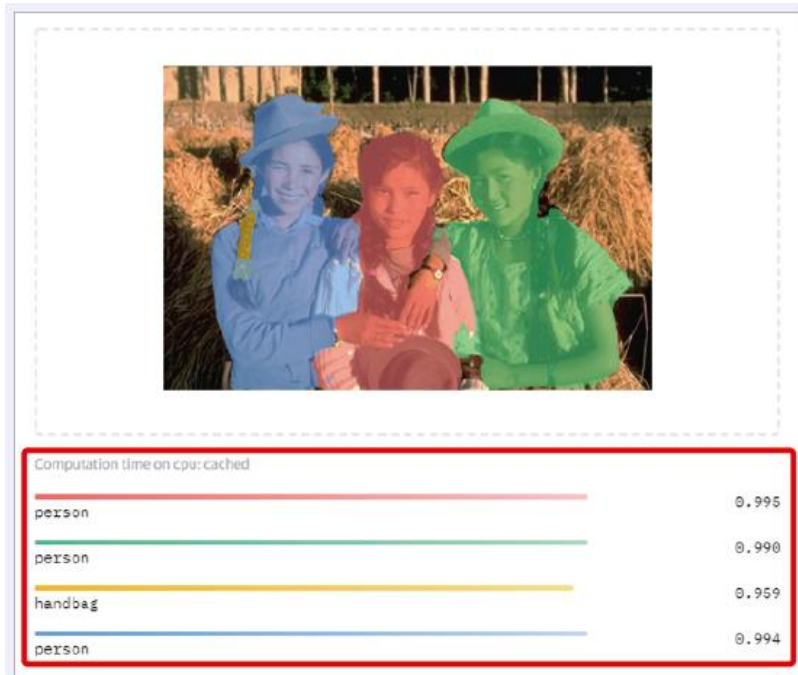
신재혁

naezang@cbnu.ac.kr

7장 딥러닝 비전

Preview

■ 컴퓨터 비전의 오랜 난제와 딥러닝의 혁신



(a) 의미 분할 문제



(b) 영상 설명하기 문제

그림 7-1 딥러닝이 일으킨 혁신

<https://huggingface.co/facebook/detr-resnet-50-panoptic>
https://huggingface.co/spaces/akhaliq/CLIP_prefix_captioning

기계학습과 비전

7.1 방법론의 대전환

■ 규칙 기반 방법론과 딥러닝 방법론

- 규칙 기반은 사람이 영상을 보고 규칙을 도출하고 프로그래밍
 - 분명한 한계. 예) [그림 5-9]의 스케일 공간에서 가우시안보다 좋은 필터는 없을까? 데이터에 따라 최적 필터를 설계해야 하지 않을까?
- 딥러닝은 데이터 기반 학습을 통해 문제 해결
 - 예) 최적 필터를 학습으로 알아냄
 - 양질의 데이터가 매우 중요

■ 둘 다 이해하고 다룰 수 있을 때 컴퓨터 비전 전문성 완성

7.1 방법론의 대전환

■ 딥러닝으로 대전환함으로써 혁신을 이룸

- 신경망은 기계학습의 일종
- 딥러닝은 층을 깊게 쌓은 깊은 신경망을 이용

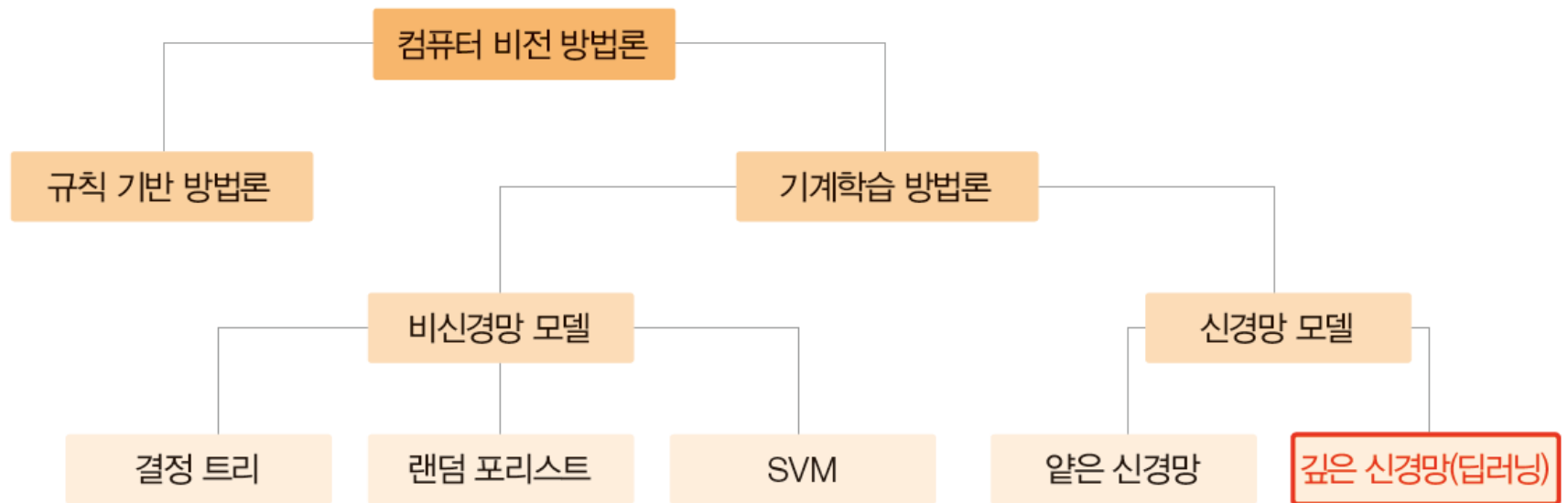


그림 7-2 컴퓨터 비전 방법론의 열개

7.2 기계학습 기초

■ 기계학습 machine learning의 단순한 예

- 기름을 더 분사하면 보일러 온도가 올라간다는 단순한 사례
- 기름 분사량을 x , 온도를 y 로 두었을 때 x 와 y 의 관계를 표현하는 함수
- 이 함수를 알면 분사량에 따른 온도 예측 가능

■ 기계학습

- 함수를 **모델**이라 부름
- 수집한 데이터로 방정식을 풀어 함수를 알아내는 일을 **학습**이라 부름
- 학습된 모델로 특정 분사량에 따른 온도를 계산하는 일을 **예측**이라 부름

$$\text{기계학습 모델: } y=f(x) \quad (7.1)$$

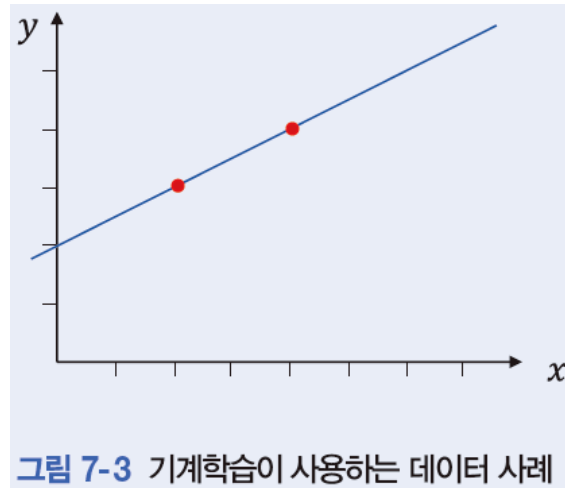
7.2 기계학습 기초

■ [예시 7-1] 기름 분사량에 따라 온도를 예측하는 모델

1. 데이터 수집

$$\text{훈련 집합} \begin{cases} x^1 = 2, y^1 = 3 \\ x^2 = 4, y^2 = 4 \end{cases}$$

이 책은 i 번째 샘플을 x^i 로 표기함



2. 모델 선택: 선형이라고 가정하면,

$$\text{모델: } y = f(x) = u_1 x + u_0$$

3. 학습: 방정식을 풀면,

가중치(매개변수) 집합 $\mathbf{w} = (u_0 \ u_1)$

$$u_1 = 0.5 \text{와 } u_0 = 2$$

4. 예측: 훈련 집합에 없던 새로운 샘플 $x=5$ 를 입력하면,

$$\text{예측: } y = f(5) = 0.5 \times 5 + 2 = 4.5$$

7.2 기계학습 기초

■ 기계학습의 네 단계

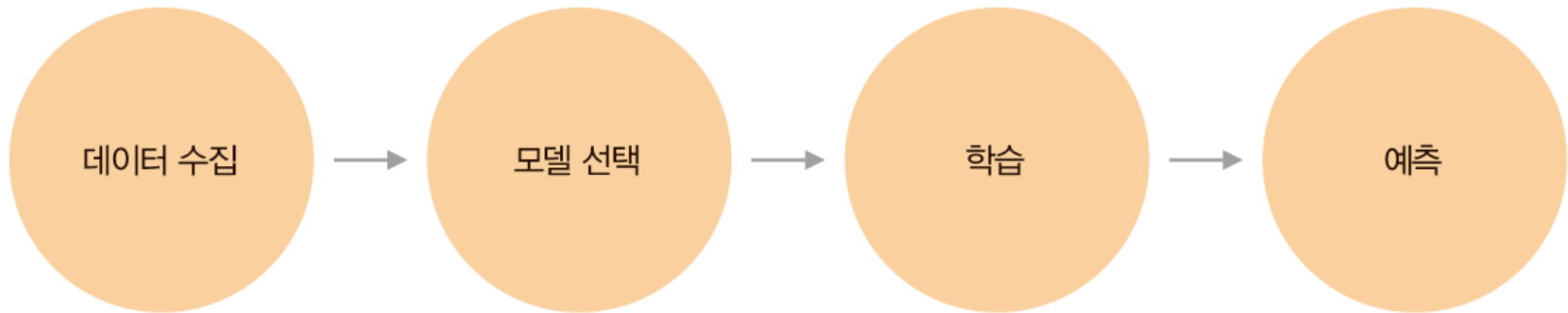


그림 7-4 전형적인 기계학습 과정

7.2 기계학습 기초

■ 1단계: 데이터 수집

- 모델의 입력은 특징 벡터_{feature vector} 출력은 참값_{GT; ground truth} (또는 레이블_{label}이라 부름)

$$\text{특징 벡터: } \mathbf{x} = (x_1, x_2, \dots, x_d) \quad (7.2)$$

$$\text{데이터셋: } D = \{(\mathbf{x}^1, y^1), (\mathbf{x}^2, y^2), \dots, (\mathbf{x}^n, y^n)\} \quad (7.3)$$

n은 샘플의 개수(데이터셋 크기)

↑ 특징 벡터 ↘ 레이블(참값)

- 회귀_{regression}와 분류_{classification} 문제
 - 회귀는 레이블이 연속 값(온도, 판매량 등): [예시 7-1]은 회귀 문제
 - 분류는 레이블이 이산 값(숫자 분류, 물체 분류, 성별 등)

7.2 기계학습 기초

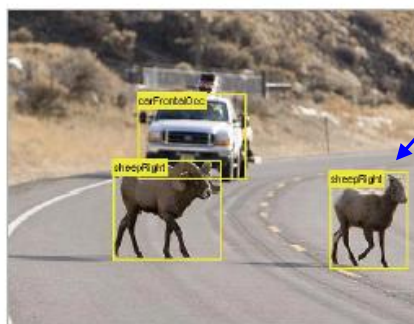
■ 1단계: 데이터 수집(...계속)

- 검출, 분할, 추적 등의 다양한 문제
- 데이터 수집은 많은 비용 소요
- 다행히 공개 데이터 많음

예: aihub 사이트(<https://www.aihub.or.kr/>)



(a) 분류: Boat(왼쪽)와 Chair(오른쪽)



(b) 검출

박스 집합이 레이블



(c) 분할



화소 집합이 레이블

7.2 기계학습 기초

■ 2단계: 모델 선택 model selection

- 선형 모델과 비선형 모델
- 예) [예시 7-1]에서 2차 함수를 모델로 사용하면 가중치(매개변수)가 3개로 확대
$$y = u_2 x^2 + u_1 x + u_0$$
 - 가중치(매개변수) 집합 $\mathbf{W} = (u_0 \ u_1 \ u_2)$
- 딥러닝은 가중치가 수만~수조 개

7.2 기계학습 기초

■ 3단계: 학습_{learning}

- 훈련 집합에 있는 샘플을 최소 오류로 맞추는 최적의 가중치 값을 알아내는 작업
- [예시 7-1]처럼 단순한 경우, 방정식 풀어 해결하는 분석적 방법_{analytical method} 사용
- 기계학습은 오류를 조금씩 줄이는 과정을 반복하는 수치적 방법_{numerical method} 사용
- 모델이 범하는 오류를 측정하는 손실 함수_{loss function} 필요

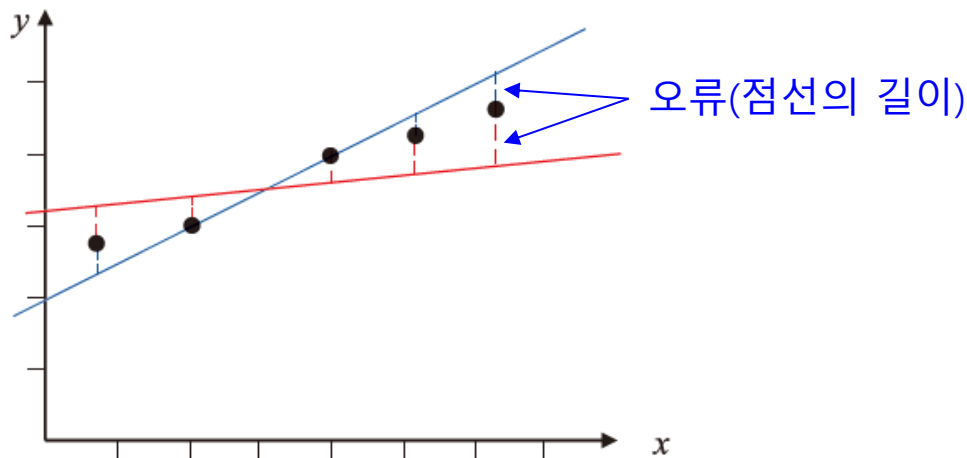


그림 7-6 모델이 범하는 오류를 측정하는 손실 함수

7.2 기계학습 기초

■ 3단계: 학습 (...계속)

- 평균제곱오차 MSE: mean squared error 는 가장 널리 쓰이는 손실 함수
 - 참값과 예측 값의 차이를 제곱하고 평균을 계산

$$\text{평균제곱오차: } J(\mathbf{W}) = \frac{1}{n} \sum_{i=1, n} \left(f(\mathbf{x}^i) - y^i \right)^2 \quad (7.4)$$

- 손실 함수가 최소가 되는 점을 알아내는 최적화 알고리즘을 옵티마이저 optimizer라 부름
 - [그림 7-6]에서는 파란색 모델이 최적에 더 가까움

7.2 기계학습 기초

■ 4단계: 예측_{prediction} (또는 추론_{inference} 라고 부름)

- 학습을 마친 모델에 (학습에 사용하지 않았던) 새로운 특징 벡터를 입력하고 출력을 구하는 과정
- 예측을 통해 모델의 성능을 측정
- 일반화 능력 중요
 - 훈련 집합과 테스트 집합으로 나누고 테스트 집합으로 성능 측정
 - 또는 성능에 대한 신뢰를 높이려고 k-겹 교차 검증_{k-fold cross validation} 사용

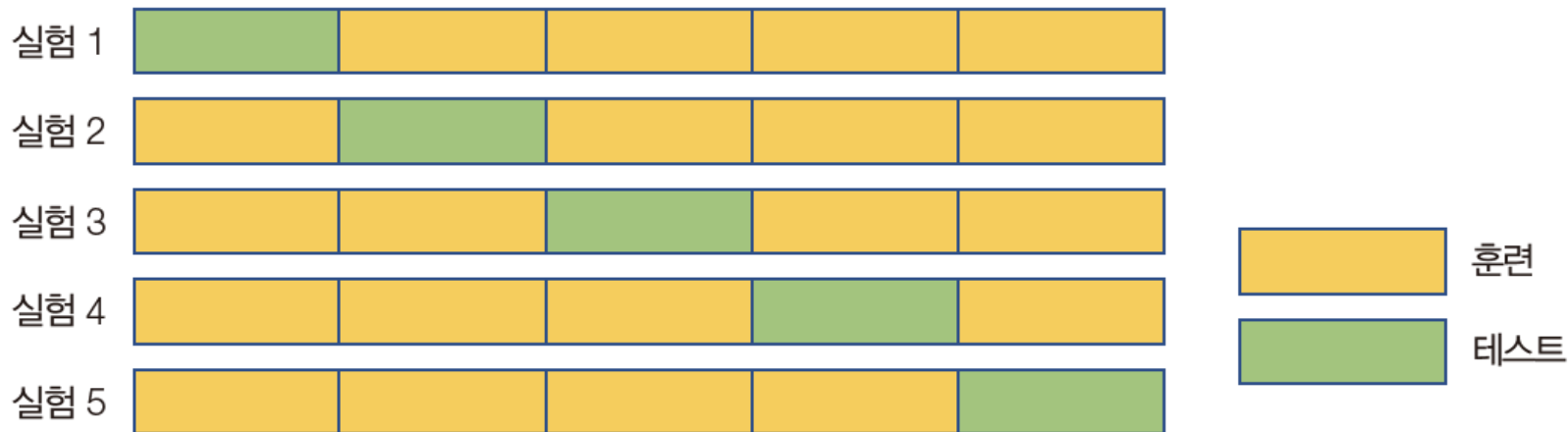


그림 7-7 k -겹 교차 검증($k=5$)

딥러닝과 비전

7.3 딥러닝 소프트웨어 맛보기

■ 기계학습 라이브러리

- sklearn은 딥러닝 이전의 고전적 기계학습 모델 지원(mlp, svm, decision tree, random forest 등)
- 텐서플로_{TensorFlow}와 파이토치_{PyTorch}는 딥러닝 지원

7.3 딥러닝 소프트웨어 맛보기

■ 텐서플로 소개

- 2015년 최초 공개
- 솔레는 텐서플로 위에서 동작하는 케라스를 개발하여 공개
- 2019년에 버전2를 공개: 버전2에서는 tensorflow 객체가 numpy와 호환되고 케라스와 한 몸이 됨(텐서플로 설치하면 케라스 따라 설치됨)

7.3 딥러닝 소프트웨어 맛보기

■ 데이터와 텐서

- [프로그램 7-1]은 텐서플로가 제공하는 데이터셋 확인
 - MNIST 필기 숫자 데이터셋
 - CIFAR-10 자연영상 데이터셋
- 딥러닝에서는 다차원 배열을 텐서라 부름

프로그램 7-1

텐서플로로 데이터 확인하기

```
01 import tensorflow as tf
02 import tensorflow.keras.datasets as ds
03 import matplotlib.pyplot as plt
04
05 (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
06 print(x_train.shape,y_train.shape,x_test.shape,y_test.shape) ①
07 plt.figure(figsize=(24,3))
08 plt.suptitle('MNIST',fontsize=30)
09 for i in range(10):
10     plt.subplot(1,10,i+1)
11     plt.imshow(x_train[i],cmap='gray') ②
12     plt.xticks([]); plt.yticks([])
13     plt.title(str(y_train[i]),fontsize=30)
```

7.3 딥러닝 소프트웨어 맛보기

```
15 (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
16 print(x_train.shape,y_train.shape,x_test.shape,y_test.shape) ③
17 class_names=['airplane','car','bird','cat','deer','dog','frog','horse','ship',
18              'truck']
19 plt.figure(figsize=(24,3))
20 plt.suptitle('CIFAR-10',fontsize=30)
21 for i in range(10):
22     plt.subplot(1,10,i+1)
23     plt.imshow(x_train[i]) ④
24     plt.xticks([]); plt.yticks([])
25     plt.title(class_names[y_train[i,0]],fontsize=30)
```

7.3 딥러닝 소프트웨어 맛보기

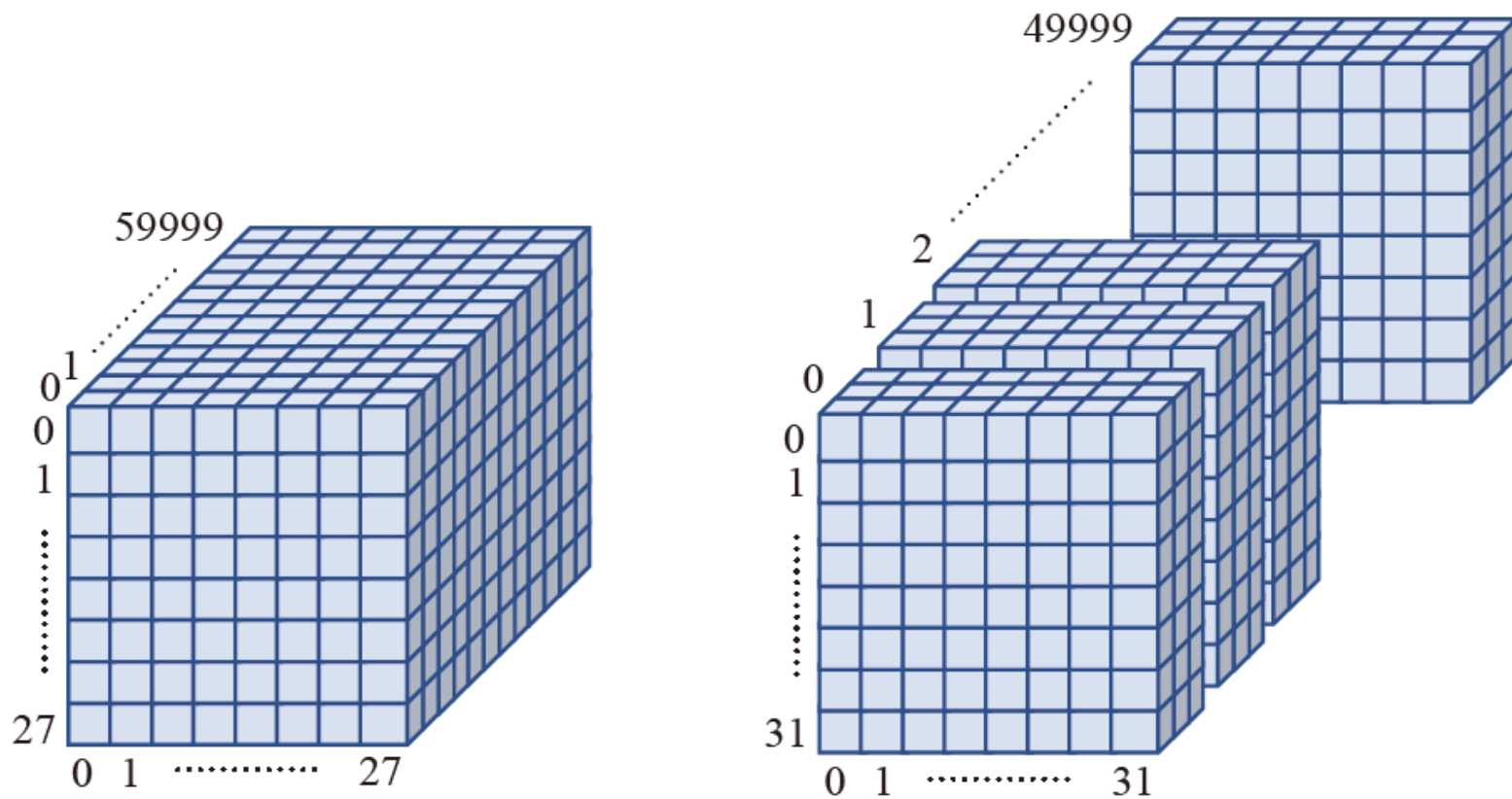
훈련 집합에 28*28 영상이 60000개

테스트 집합에 28*28 영상이 10000개



TIP MNIST의 y_train은 (60000,)인데, CIFAR-10의 y_train은 (50000,1)이다. MNIST의 y_train은 [5,0,4,...]의 리스트로 저장되어 있고, CIFAR-10의 y_train은 [[6],[9],[9],...]의 리스트로 저장되어 있어 다르다.

7.3 딥러닝 소프트웨어 맛보기



(a) MNIST의 x_train

(b) CIFAR-10의 x_train

그림 7-8 데이터셋의 텐서 구조

인공신경망

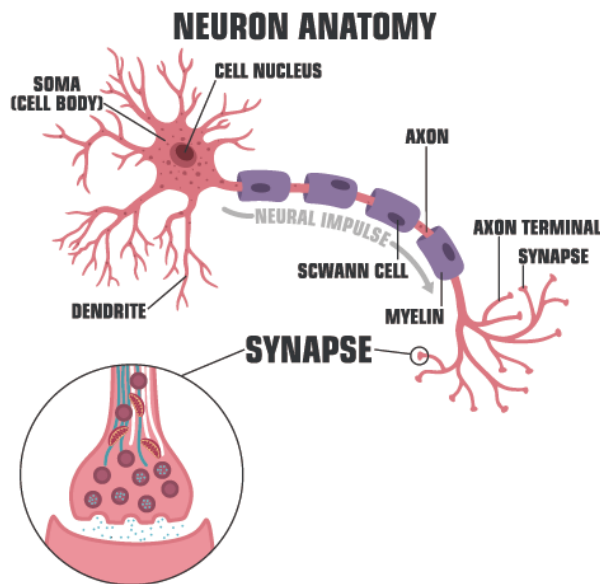
7.4 인공 신경망의 태동

- **신경망은 기계학습에서 가장 성공한 모델([그림 7-2])**
 - 이 절은 신경망의 간략한 역사와 초창기 모델인 퍼셉트론 소개

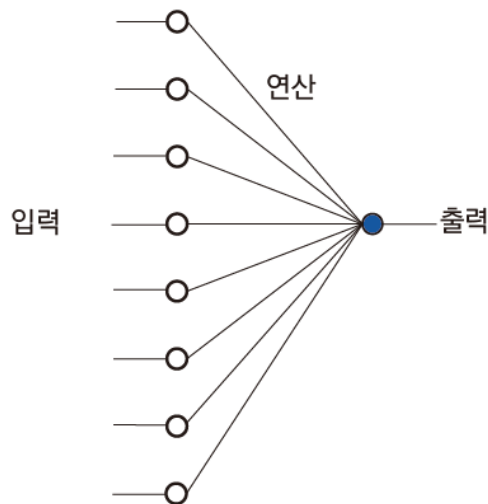
7.4.1 신경망의 간략 역사

■ 사람 뇌와 컴퓨터

- 뉴런^{neuron}은 뇌의 정보처리 단위. 연산을 수행하는 세포체^{soma 또는 cell body} 처리한 정보를 다른 뉴런에 전달하는 축삭^{axon}, 다른 뉴런으로부터 정보를 받는 수상돌기^{dendrite}로 구성
- 사람 뇌는 10^{11} 개 가량의 뉴런, 뉴런마다 1000개 가량 연결 → 고도의 병렬 처리기
- 반면에 폰 노이만 컴퓨터는 아주 빠른 순차 명령어 처리기



(a) 뇌의 정보 처리 단위인 뉴런



(b) 뉴런을 모방한 퍼셉트론

7.4.1 신경망의 간략 역사

■ 간략한 인공 신경망 역사

- 1943년 맥컬록과 피츠의 계산 모형
- 1949년 헤브의 학습 알고리즘
- 1958년 로젠블랫의 퍼셉트론
- 위드로와 호프의 아달린과 마달린
- 1960년대 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

7.4.2 퍼셉트론

■ 퍼셉트론

- 1958년에 등장한 원시적인 모델
- 딥러닝 이론의 토대이고 핵심 부품으로 사용되므로 딥러닝을 이해하는데 지름길

7.4.2 퍼셉트론

■ 구조와 연산

s 는 로짓(logit) τ 는 활성화 함수(activation function)

$$o = \tau(s) = \tau\left(\sum_{i=0,d} u_i x_i\right)$$
$$\tau(s) = \begin{cases} +1, s > 0 \\ -1, s \leq 0 \end{cases} \quad (7.5)$$

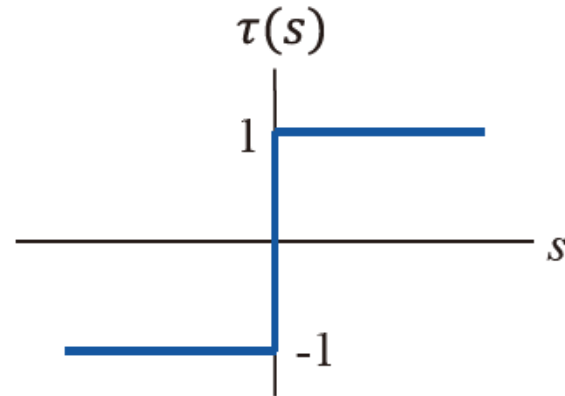
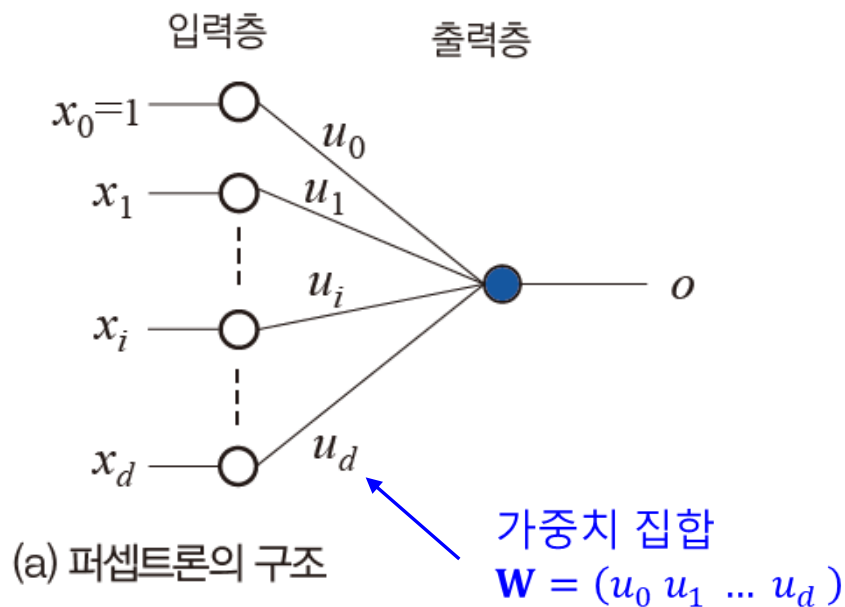


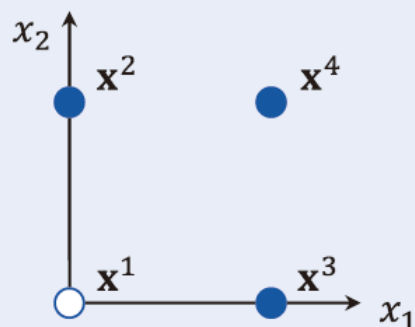
그림 7-10 퍼셉트론의 구조와 연산

7.4.2 퍼셉트론

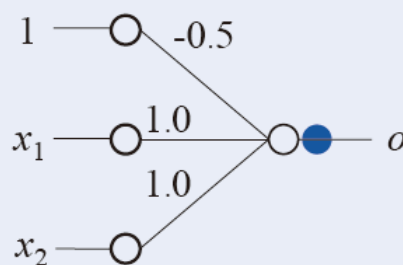
[예시 7-2] 퍼셉트론으로 OR 분류 문제 풀기

OR 데이터셋은 아주 단순하여 퍼셉트론을 설명하는 데 자주 쓰인다. [그림 7-11]은 4개 샘플로 구성된 OR 데이터셋을 보여준다. $\mathbf{x}^1 \sim \mathbf{x}^4$ 는 특징 벡터고 $y^1 \sim y^4$ 는 참값이다. 식 (7.2)와 식 (7.3)에 따르면 $d=2$, $n=4$ 다. 속이 찬 샘플은 참값이 1이고 속이 빈 샘플은 -1이다.

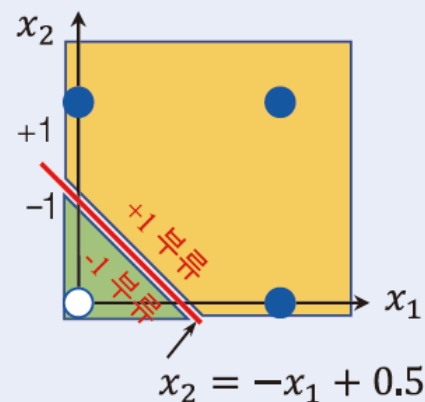
$$\begin{array}{cccc} \mathbf{x}^1 = (0,0) & \mathbf{x}^2 = (0,1) & \mathbf{x}^3 = (1,0) & \mathbf{x}^4 = (1,1) \\ y^1 = -1 & y^2 = 1 & y^3 = 1 & y^4 = 1 \end{array}$$



(a) OR 데이터셋



(b) OR 데이터셋을 분류하는 퍼셉트론



(c) 퍼셉트론이 형성하는 결정 경계

그림 7-11 OR 분류 문제를 해결하는 퍼셉트론

7.4.2 퍼셉트론

[그림 7-11(b)]는 OR 데이터셋을 분류하는 퍼셉트론이다. 가중치는 $u_0 = -0.5$, $u_1 = 1$, $u_2 = 1$ 이다. 데이터셋을 식 (7.5)에 입력하여 옳게 분류하는지 확인해보자. \mathbf{x}^2 를 식 (7.5)에 입력하면 아래와 같이 1을 얻는데, \mathbf{x}^2 의 참값이 $y^2 = 1$ 이므로 옳게 분류했다. 나머지 샘플 3개도 같은 방식으로 입력해보면 모두 옳게 분류한다.

$$o = \tau(-0.5 \times 1 + 1 \times 0 + 1 \times 1) = \tau(0.5) = 1$$

7.4.2 퍼셉트론

■ 퍼셉트론은 이진 분류기

- [그림 7-11(b)]의 가중치를 식 (7.5)에 대입하면 아래 식

$$o = u_2x_2 + u_1x_1 + u_0x_0 = x_2 + x_1 - 0.5$$

- $o = 0$ 으로 두고 식을 정리하면 아래 결정 직선을 얻음

$$x_2 + x_1 - 0.5 = 0$$

$$x_2 = -x_1 + 0.5 \quad \longleftarrow \text{[그림 7-11(c)]의 빨간 선분}$$

- 퍼셉트론은 특징 공간을 두 부분 공간으로 나누는 이진 분류기_{binary classifier}

■ 바이어스의 역할

- 바이어스가 없으면 결정 직선이 항상 원점을 지나므로 제대로 분류할 수 없음

7.4.2 퍼셉트론

■ 행렬 표기

- 샘플 하나 처리하는 상황
 - \mathbf{x} 는 바이어스를 추가한 $(d+1)$ 차원 벡터

$$o = \tau(\mathbf{u}\mathbf{x}^T) \quad (7.6)$$

- 예) 두번째 샘플 \mathbf{x}^2 에 대한 행렬 계산

$$o = \tau(\mathbf{u}\mathbf{x}^{2T}) = \tau\left(\begin{pmatrix} -0.5 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}\right) = \tau(-0.5 \times 1 + 1 \times 0 + 1 \times 1) = \tau(0.5) = 1$$

- 훈련 집합 전체를 한꺼번에 처리하는 상황

$$\mathbf{O} = \tau(\mathbf{u}\mathbf{X}^T) \quad (7.7)$$



\mathbf{x} 는 특징 벡터가 행에 배치된 설계 행렬(design matrix)

7.4.2 퍼셉트론

[예시 7-3] OR 데이터셋을 행렬 표기로 처리

식 (7.7)의 행렬 표기를 따르면 OR 데이터셋은 아래와 같다.

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \mathbf{x}^3 \\ \mathbf{x}^4 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{pmatrix}$$

바이어스 값을 추가하고 전치를 적용한 \mathbf{X}^T 는 아래와 같다. 바이어스를 파란색으로 표시해 쉽게 확인할 수 있게 하였다.

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

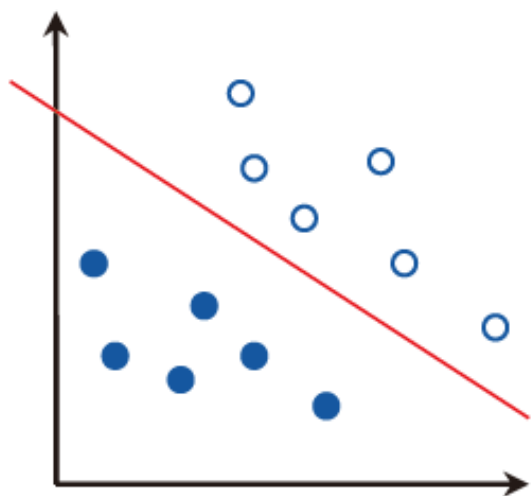
\mathbf{X}^T 를 식 (7.7)에 대입하면 아래와 같다. 퍼셉트론이 예측한 결과를 담은 행렬 \mathbf{O} 를 참값과 비교하면 같다. 따라서 [그림 7-11(b)]의 퍼셉트론은 OR 데이터를 100% 정확률로 분류한다고 말할 수 있다.

$$\mathbf{O} = \tau(\mathbf{uX}^T) = \tau \left(\begin{pmatrix} -0.5 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \right) = \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix}$$

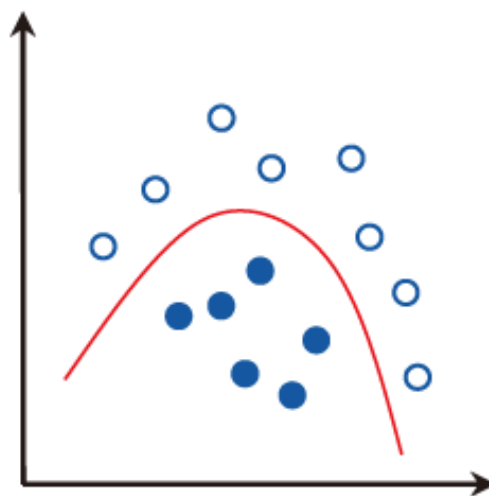
7.5 깊은 다층 퍼셉트론

■ 퍼셉트론은 선형 분류기

- 선형 분리 가능한 데이터만 높은 성능
- 현실에서는 대부분 비선형(선형 분리 불가능) 데이터



(a) 선형 분리 가능한 데이터셋



(b) 선형 분리 불가능한 데이터셋

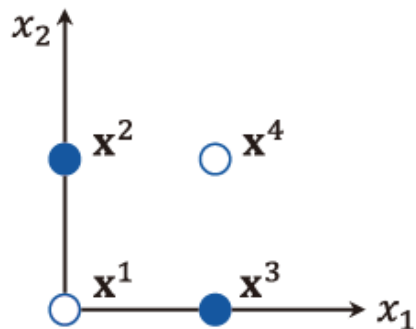
그림 7-12 선형 분리 가능과 불가능

■ 루멜하트는 다층 퍼셉트론으로 비선형 분류 문제 해결

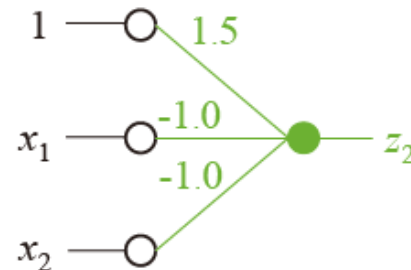
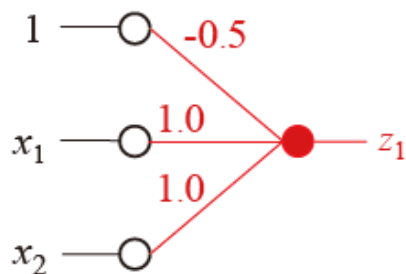
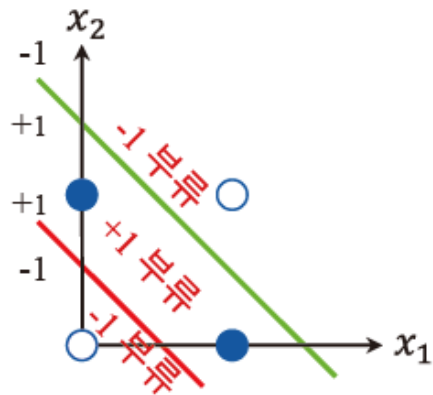
7.5.1 다층 퍼셉트론

■ 퍼셉트론을 여러 개 조합하여 다층 퍼셉트론 구성

- XOR는 선형 분리 불가능한 데이터인데, 다층 퍼셉트론으로 해결 가능



(a) XOR 데이터셋



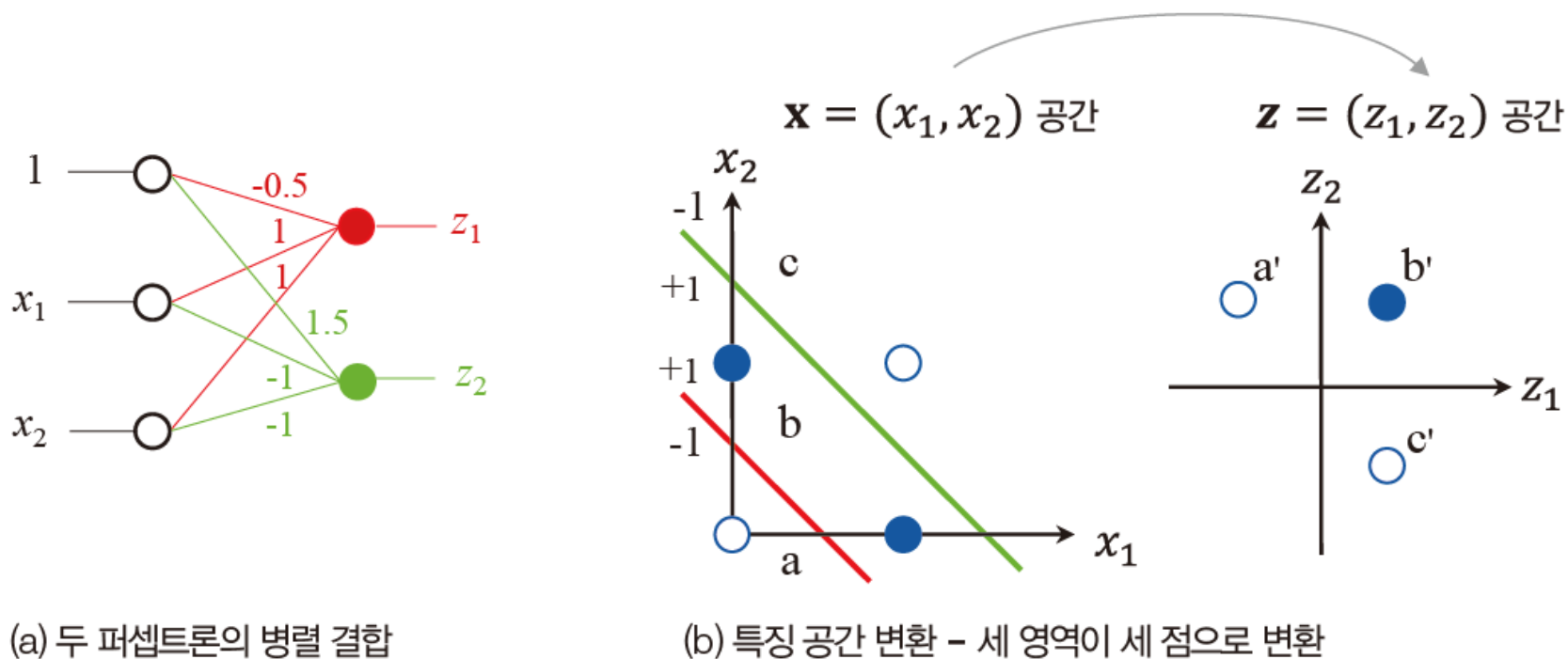
(b) 퍼셉트론을 2개 사용하여 특징 공간을 3개 영역으로 분할

그림 7-13 퍼셉트론을 여러 개 사용하여 XOR 분류 문제 해결

7.5.1 다층 퍼셉트론

■ 두 개의 퍼셉트론으로 특징 공간을 변환

- 원래 특징 공간 (x_1, x_2) 를 더 유리한 특징 공간 (z_1, z_2) 로 변환
- 새로운 특징 공간은 선형 분리 가능해짐
- 새로운 특징 공간을 은닉 공간_{hidden space} 또는 잠복 공간_{latent space}라고 부름



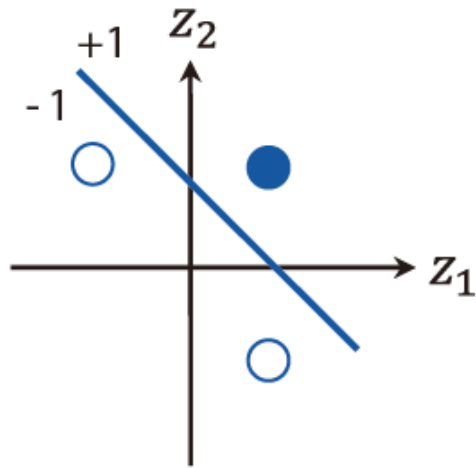
(a) 두 퍼셉트론의 병렬 결합

(b) 특징 공간 변환 - 세 영역이 세 점으로 변환

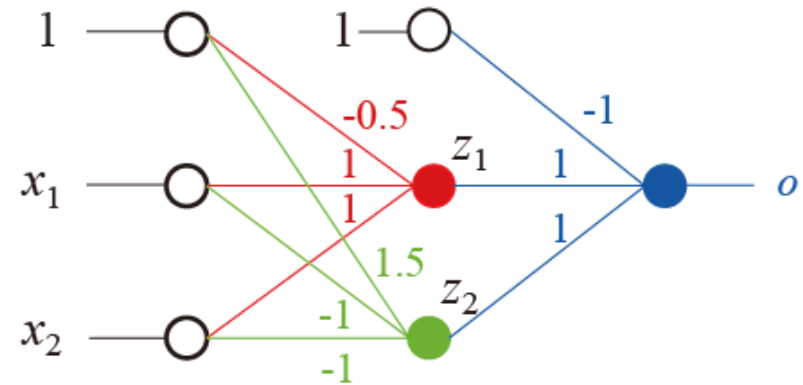
그림 7-14 특징 공간의 변환

7.5.1 다층 퍼셉트론

■ 변환된 공간에서 세번째 퍼셉트론을 사용하여 분류 수행



(a) z 공간을 분할하는 세 번째 퍼셉트론



(b) 세 번째 퍼셉트론의 순차 결합

그림 7-15 다층 퍼셉트론

7.5.1 다층 퍼셉트론

표 7-1 [그림 7-15(b)]의 다층 퍼셉트론을 이용한 XOR 데이터 인식

| 특징 공간 x | 은닉 공간 z | 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 |

7.5.1 다층 퍼셉트론

■ 다층 퍼셉트론의 구조

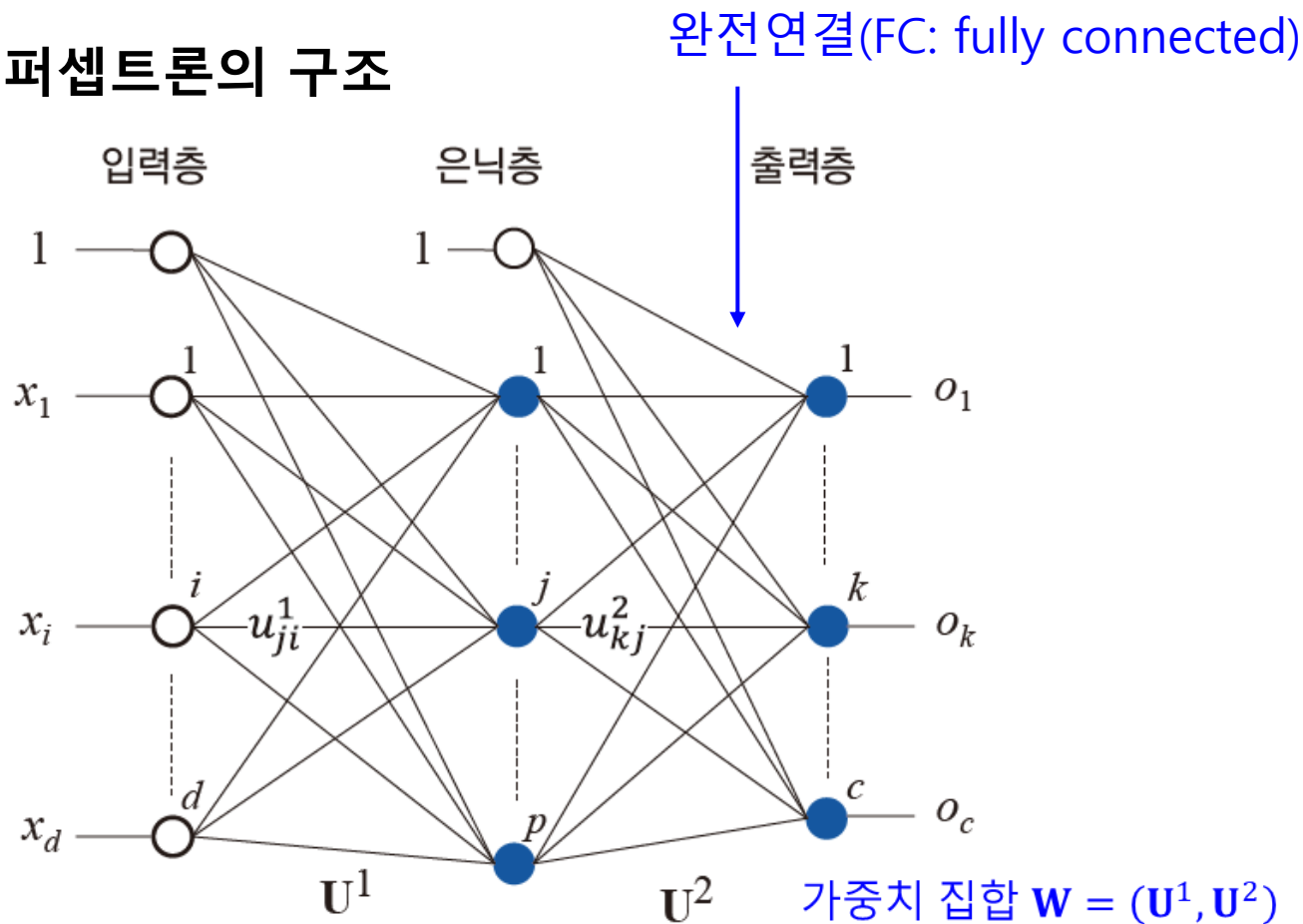


그림 7-16 다층 퍼셉트론의 일반적인 구조

은닉층과 출력층을 연결하는 가중치 행렬
입력층과 은닉층을 연결하는 가중치 행렬

7.5.1 다층 퍼셉트론

■ 가중치

- 은닉층의 j 번째 노드와 입력층의 i 번째 노드를 연결하는 에지 가중치를 u_{ji}^1 로 표기
- 출력층의 k 번째 노드와 은닉층의 j 번째 노드를 연결하는 에지 가중치를 u_{kj}^2 로 표기

$$\mathbf{U}^1 = \begin{pmatrix} u_{10}^1 & u_{11}^1 & \cdots & u_{1d}^1 \\ u_{20}^1 & u_{21}^1 & \cdots & u_{2d}^1 \\ \vdots & \vdots & \ddots & \vdots \\ u_{p0}^1 & u_{p1}^1 & \cdots & u_{pd}^1 \end{pmatrix} \quad \mathbf{U}^2 = \begin{pmatrix} u_{10}^2 & u_{11}^2 & \cdots & u_{1p}^2 \\ u_{20}^2 & u_{21}^2 & \cdots & u_{2p}^2 \\ \vdots & \vdots & \ddots & \vdots \\ u_{c0}^2 & u_{c1}^2 & \cdots & u_{cp}^2 \end{pmatrix} \quad (7.8)$$

■ 데이터에 따라 신경망 구조가 결정됨

- 특징 벡터의 차원이 d 이고 부류의 개수가 c 이면 입력층의 노드 개수는 $d + 1$ 이고 출력층의 노드 개수는 c
 - 예) MNIST 경우, 입력층은 785개($28 \times 28 + 1$), 출력층은 10개 노드를 가짐
- 은닉층의 노드 개수 p 는 하이퍼 매개변수(사용자가 지정해야 함)

7.5.1 다층 퍼셉트론

■ 전방 계산(노드 한 개를 위한 식)

- 신경망 구조가 복잡해져서 표기가 복잡해졌을 뿐 퍼셉트론의 식 (7.5)가 같음

$$j\text{번째 은닉 노드의 연산: } z_j = \tau\left(\sum_{i=0,d} u_{ji}^1 x_i\right) \quad (7.9)$$

- 식 (7.9)의 은닉 노드 계산을 행렬로 표기하면

$$j\text{번째 은닉 노드의 연산(행렬 표기): } z_j = \tau\left(\mathbf{u}_j^1 \mathbf{x}^T\right) \quad (7.10)$$

- \mathbf{u}_j^1 은 \mathbf{U}^1 의 j 번째 행 (즉 $\mathbf{u}_j^1 = (u_{j0}^1 \ u_{j1}^1 \ \cdots \ u_{jd}^1)$)

- 출력 노드 계산을 행렬로 표기하면

$$k\text{번째 출력 노드의 연산(행렬 표기): } o_k = \tau\left(\mathbf{u}_k^2 \mathbf{z}\right) \quad (7.11)$$

- \mathbf{u}_k^2 은 \mathbf{U}^2 의 k 번째 행 (즉 $\mathbf{u}_k^2 = (u_{k0}^2 \ u_{k1}^2 \ \cdots \ u_{kp}^2)$)

7.5.1 다층 퍼셉트론

■ 전방 계산(층의 모든 노드를 포함한 식. 샘플 한 개를 위한 식)

$$\begin{aligned} \text{은닉층의 연산: } \mathbf{z} &= \tau_1(\mathbf{U}^1 \mathbf{x}^T) \\ \text{출력층의 연산: } \mathbf{o} &= \tau_2(\mathbf{U}^2 \mathbf{z}) \end{aligned} \quad (7.12)$$

- 두 층의 연산을 결합하여 복합 함수로 쓰면

$$\text{다층 퍼셉트론의 전방 계산: } \mathbf{o} = \tau_2(\mathbf{U}^2 \tau_1(\mathbf{U}^1 \mathbf{x}^T)) \quad (7.13)$$

■ 전방 계산(모든 샘플을 담은 설계 행렬을 위한 식)

$$\text{데이터셋 전체에 대한 다층 퍼셉트론의 전방 계산: } \mathbf{O} = \tau_2(\mathbf{U}^2 \tau_1(\mathbf{U}^1 \mathbf{X}^T))$$

$$\text{이때, 설계 행렬 } \mathbf{X} = \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^n \end{pmatrix} \quad (7.14)$$

7.5.1 다층 퍼셉트론

[예시 7-4] 다층 퍼셉트론의 연산

XOR 데이터셋을 인식하는 [그림 7-15(b)]의 신경망의 연산을 살펴보자. 이 신경망은 $d=2$, $p=2$, $c=1$ 이다. 가중치 행렬은 다음과 같다.

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

샘플을 하나씩 처리하는 식 (7.13)을 $\mathbf{x}=(0,1)$ 샘플에 적용해보자. 계산을 편하게 하려고 활성 함수로 계단 함수를 사용한다. \mathbf{x} 에 바이어스를 추가한 $\mathbf{x}=(1,0,1)$ 을 신경망에 입력하고 계산하는 과정은 아래와 같다. 활성 함수 τ_1 은 활성 함수를 적용할 뿐 아니라 계산 결과에 은닉층의 바이어스를 추가하는 역할까지 한다고 가정한다. 나머지 3개 샘플에 대한 계산은 연습문제로 남겨둔다.

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

7.5.1 다층 퍼셉트론

데이터셋을 한꺼번에 처리하는 식 (7.14)를 적용하면 아래와 같다.

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

7.5.2 딥러닝의 서막: 깊은 다층 퍼셉트론

■ 깊은 다층 퍼셉트론의 전방 계산

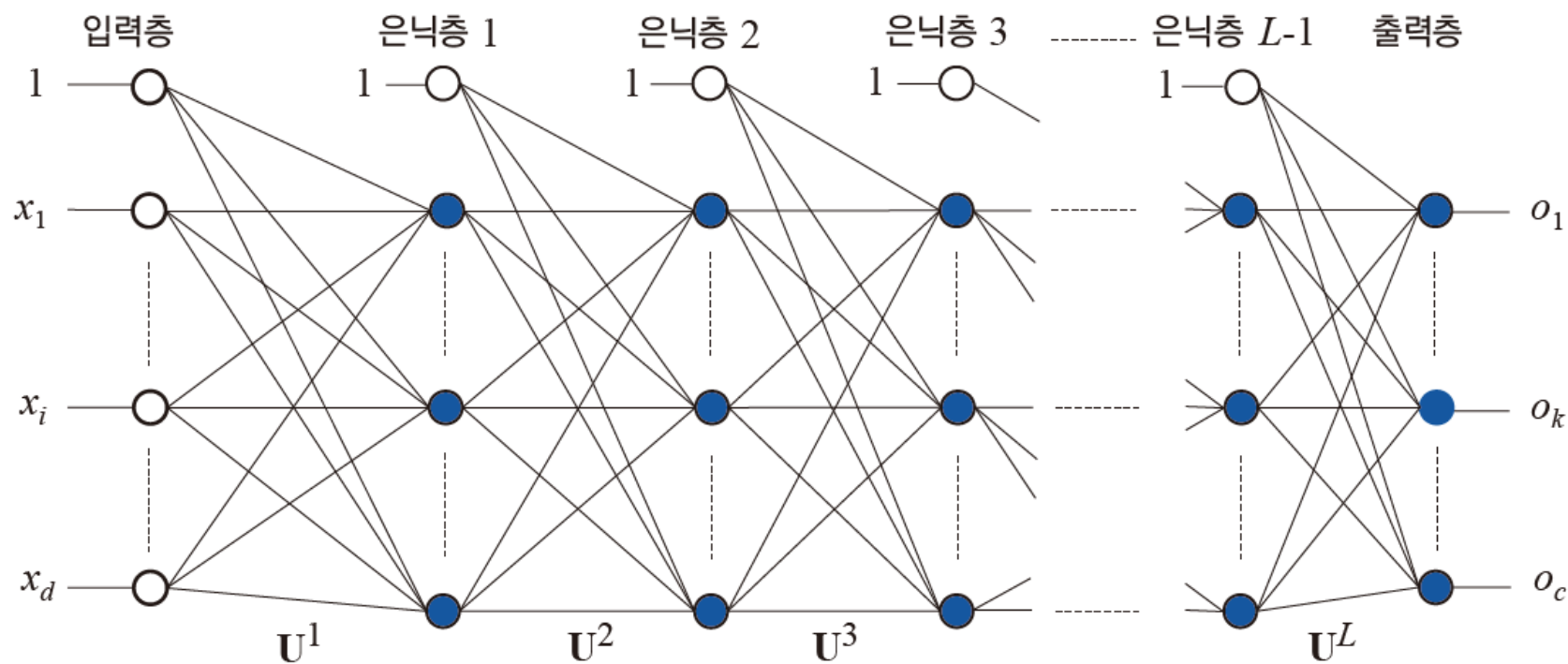


그림 7-17 깊은 다층 퍼셉트론(L 층 신경망)

가중치 집합 $\mathbf{W} = (U^1, U^2, \dots, U^L)$

7.5.2 딥러닝의 서막: 깊은 다층 퍼셉트론

■ l 번째 층의 가중치 행렬

- l 번째 층의 j 번째 노드와 $l-1$ 번째 층의 i 번째 노드를 연결하는 가중치를 u_{ji}^l 로 표기

$$\mathbf{U}^l = \begin{pmatrix} u_{10}^l & u_{11}^l & \cdots & u_{1n_{l-1}}^l \\ u_{20}^l & u_{21}^l & \cdots & u_{2n_{l-1}}^l \\ \vdots & \vdots & \ddots & \vdots \\ u_{n_l 0}^l & u_{n_l 1}^l & \cdots & u_{n_l n_{l-1}}^l \end{pmatrix}, \quad l=1,2,\cdots,L \quad (7.15)$$

7.5.2 딥러닝의 서막: 깊은 다층 퍼셉트론

- 전방 계산(샘플 하나를 위한 l 번째 층의 연산)

$$l\text{번째 층의 연산: } \mathbf{z}^l = \tau_l(\mathbf{U}^l \mathbf{z}^{l-1}), \quad l = 1, 2, \dots, L \quad (7.16)$$

- 전방 계산(샘플 하나를 위한 신경망 전체 연산)

$$\text{한 샘플의 전방 계산: } \mathbf{o} = \tau_L \left(\dots \tau_3 \left(\mathbf{U}^3 \tau_2 \left(\mathbf{U}^2 \tau_1 \left(\mathbf{U}^1 \mathbf{x}^T \right) \right) \right) \right) \quad (7.17)$$

- 전방 계산(설계 행렬을 위한 신경망 전체 연산)

$$\text{데이터셋 전체에 대한 전방 계산: } \mathbf{O} = \tau_L \left(\dots \tau_3 \left(\mathbf{U}^3 \tau_2 \left(\mathbf{U}^2 \tau_1 \left(\mathbf{U}^1 \mathbf{X}^T \right) \right) \right) \right) \quad (7.18)$$

7.5.2 딥러닝의 서막: 깊은 다층 퍼셉트론

■ 신경망 출력을 부류 정보로 해석

- 이 단계를 예측_{prediction} 또는 추론_{inference}이라 부름

$$\text{최종 부류: } \hat{k} = \operatorname{argmax}_k o_k \quad (7.19)$$

- 출력층은 보통 softmax 활성 함수 사용

7.5.3 활성 함수

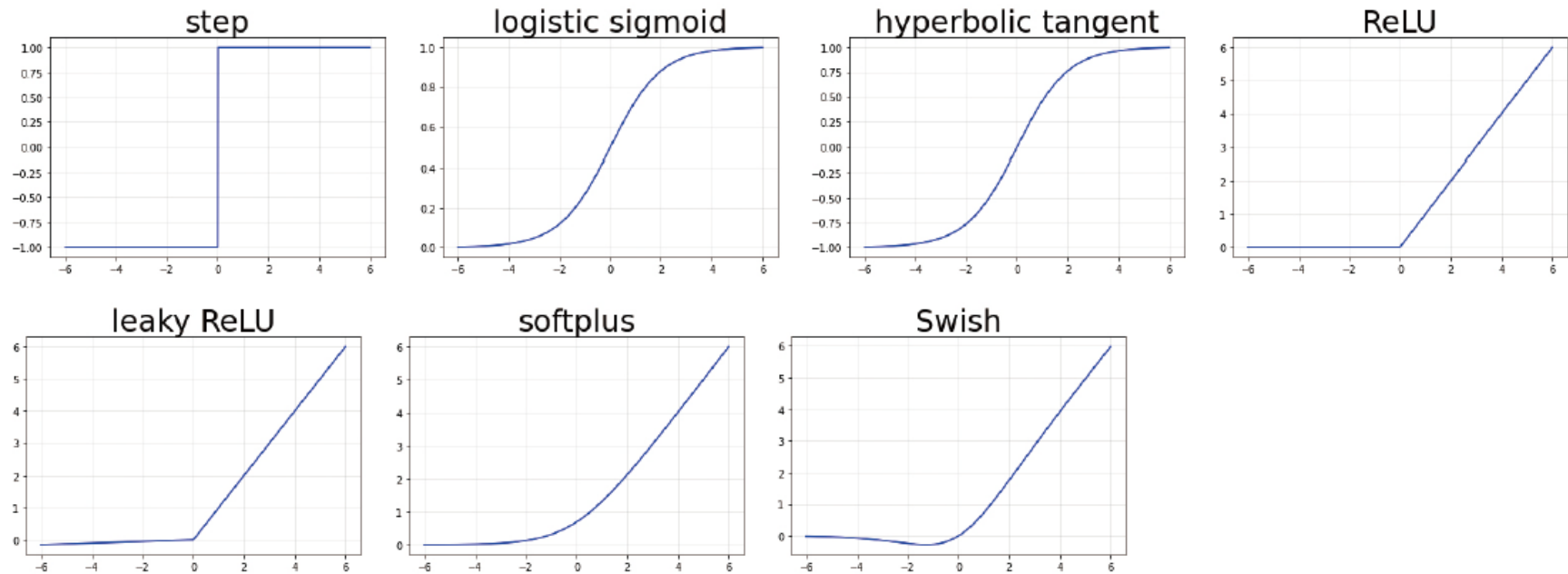


그림 7-18 신경망이 사용하는 다양한 활성 함수

표 7-2 다양한 활성 함수 $\tau(s)$

| step | logistic sigmoid | hyperbolic tangent | ReLU | leaky LeRU | softplus | Swish |
|--|------------------------------|----------------------------------|--------------|---|---------------------------------------|------------------------------|
| $\begin{cases} 1 & s > 0 \\ -1 & s \leq 0 \end{cases}$ | $\frac{1}{1 + e^{-\beta s}}$ | $\frac{2}{1 + e^{-\beta s}} - 1$ | $\max(0, s)$ | $\begin{cases} s & s > 0 \\ 0.01s & s \leq 0 \end{cases}$ | $\frac{\log(1 + e^{\beta s})}{\beta}$ | $\frac{s}{1 + e^{-\beta s}}$ |

7.5.3 활성 함수

■ 출력층이 주로 사용하는 softmax

- 로짓 벡터 $\mathbf{s} = (s_1, s_2, \dots, s_c)$ 를 부류 확률 벡터 $\mathbf{o} = (o_1, o_2, \dots, o_c)$ 로 변환
- $o_1 + o_2 + \dots + o_c = 1$ 을 만족

$$o_k = \frac{e^{s_k}}{\sum_{i=1,c} e^{s_i}} \quad (7.20)$$

7.6 학습 알고리즘

■ 학습은

- 데이터셋 $D = \{\mathbf{X}, \mathbf{Y}\}$ 를 가장 잘 맞추는 가중치 $\mathbf{W} = (\mathbf{U}^1, \mathbf{U}^2, \dots, \mathbf{U}^L)$ 을 추정하는 문제

7.6.1 발상

■ 훈련 데이터의 표현(MNIST와 CIFAR-10 데이터셋으로 설명)

- 특징 벡터 \mathbf{x}^i 는 1차원 구조로 펼쳐서 표현
- 레이블 \mathbf{y}^i 는 원핫 코드로 표현

$$\mathbf{x}^i = \text{[Handwritten digit 5 image]} = (x_1, x_2, \dots, x_{784})$$

$$\mathbf{y}^i = (0, 0, 0, 0, 0, 1, 0, 0, 0, 0)$$

(a) MNIST

$$\mathbf{x}^i = \text{[Ship image]} = (x_1, x_2, \dots, x_{3072})$$

$$\mathbf{y}^i = (0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0)$$

(b) CIFAR-10

그림 7-19 학습에 사용되는 데이터 샘플의 벡터 표현

7.6.1 발상

■ 학습 알고리즘이 알아내야 하는 가중치

- 예) MNIST를 위한 신경망
 - 은닉층 노드가 128개라면, [그림 7-16]의 신경망은 $(784+1)*128+(128+1)*10 = 101,770$ 개의 가중치를 가짐
- 60,000개의 샘플로 높은 정확률로 맞히는 10만개 이상의 가중치를 알아내는 최적화 문제

■ 학습 전략

- 처음에는 가중치에 대한 실마리가 없기 때문에 난수 설정하고 출발
- 손실 함수 값을 줄이는 방향으로 가중치 갱신을 반복
- 충분히 높은 정확률을 달성하거나 더 이상 개선이 불가능하다고 여겨질 때 멈춤

7.6.1 발상

■ 알고리즘 형식을 갖춰 쓰면

[알고리즘 7-1] 신경망 학습 알고리즘

입력: 훈련 데이터 $D=\{\mathbf{X}, \mathbf{Y}\}$

출력: 최적의 가중치 $\hat{\mathbf{W}}$

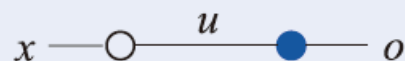
1. 가중치 행렬 \mathbf{W} 를 난수로 초기화한다. 예) 식 (7.4)의 MSE
2. while (True)
 ↓
3. \mathbf{W} 로 전방 계산을 수행하고 손실 함수 $J(\mathbf{W})$ 를 계산한다.
4. if ($J(\mathbf{W})$ 가 만족스럽거나 여러 번 반복에서 더 이상 개선이 없음) break
5. $J(\mathbf{W})$ 를 낮추는 방향 $\Delta\mathbf{W}$ 를 계산한다.
6. $\mathbf{W}=\mathbf{W}+\Delta\mathbf{W}$ ↑
7. $\hat{\mathbf{W}}=\mathbf{W}$ 미분으로 알아냄

7.6.2 스토캐스틱 경사하강법

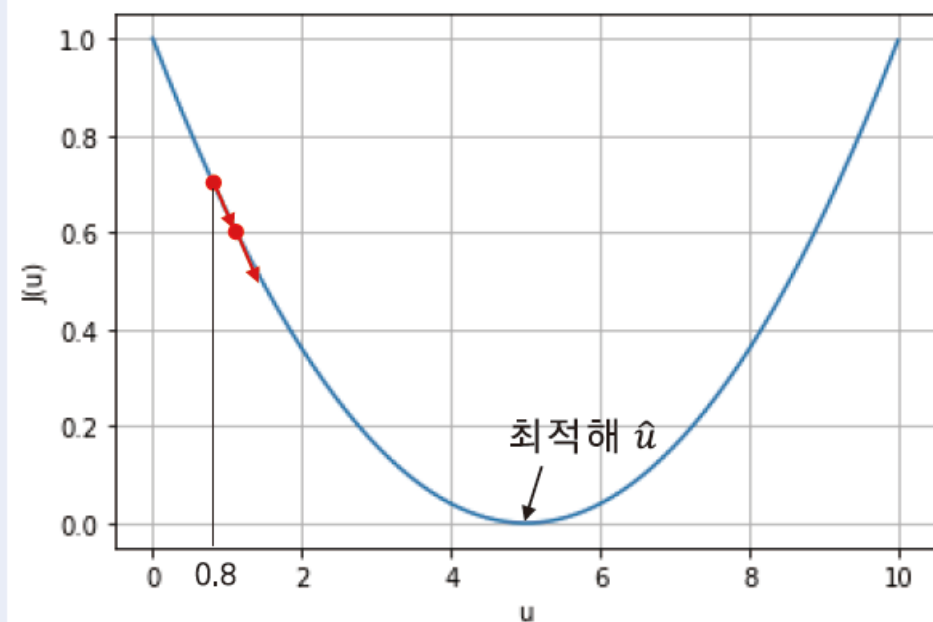
■ [알고리즘 7-1]의 5행을 어떻게 구현할 것인가?

- 손실 함수 J 를 줄이는 방향 $\Delta \mathbf{W}$ 는 J 를 가중치 \mathbf{u}^i 로 미분한 값 $\frac{\partial J}{\partial \mathbf{u}^i}$ 의 음수 \leftarrow 경사하강법의 원리

■ [예시 7-5] 가중치가 하나뿐인 단순한 신경망의 학습



$$D = \{x = 0.2, y = 1\}$$



(a) 신경망과 훈련 집합

(b) 손실 함수 J

그림 7-20 가장 단순한 상황의 경사하강법

7.6.2 스토캐스틱 경사하강법

■ 신경망 연산: $o = ux$

손실 함수: $J(u) = (y - ux)^2 = x^2u^2 - 2yxu + y^2$

J 의 도함수: $\frac{dJ}{du} = 2x^2u - 2yx$

- [알고리즘 7-1]의 1행에서 난수 생성으로 $u = 0.8$ 이라고 가정
- 5행에서 $\frac{dJ}{du} = 2(0.2)^2(0.8) - 2(1)(0.2) = -0.336$. 따라서 $\Delta u = -\frac{dJ}{du} = 0.336$
- 6행에서 $u = u + \Delta u = u + \left(-\frac{dJ}{du}\right) = 0.8 + 0.336 = 1.136$
- 5행에서 $\frac{dJ}{du} = 2(0.2)^2(1.136) - 2(1)(0.2) = -0.3091$. 따라서 $\Delta u = -\frac{dJ}{du} = 0.336$
- 6행에서 $u = 1.136 + 0.3091 = 1.4451$
- 이런 과정을 반복하여 최적해인 $\hat{u} = 5$ 에 점점 가까워짐

7.6.2 스토캐스틱 경사하강법

■ [예시 7-5]를 일반화하면

$$u = u - \rho \frac{dJ}{du} \quad (7.21)$$

- ρ 는 학습률_{learning rate} (중요한 하이퍼 매개변수)
- [예시 7-5]는 $\rho=1$ 을 사용한 셈. 이렇게 큰 값은 진자 현상 위험이 있어 0.01, 0.001, 0.0001같이 작은 값 사용하여 보수적으로 학습

■ 스토캐스틱 경사하강법 SGD: Stochastic Gradient Descent **으로 확장**

- 첫째, 신경망의 가중치는 층을 구성하고 수만~수억 개라는 사실을 반영. 또한 맨 마지막 L 층에서 시작하여 왼쪽으로 진행하면서 가중치를 갱신 \leftarrow 역전파

$$\mathbf{U}^l = \mathbf{U}^l - \rho \frac{\partial J}{\partial \mathbf{U}^l}, \quad l = L, L-1, \dots, 2, 1 \quad (7.22)$$

7.6.2 스토캐스틱 경사하강법

■ ■ 둘째, 식 (7.22)를 적용할 데이터의 단위

- 한쪽 극단은 샘플마다 식 (7.22) 적용. 다른 극단은 모든 샘플의 미분값의 평균을 구하고 식 (7.22)를 한번 적용 \leftarrow 모든 샘플을 처리하는 한 사이클을 세대_{epoch}라 부름
- 딥러닝에서는 둘의 중간인 미니 배치를 주로 사용. 훈련 집합을 미니 배치로 분할하고 미니 배치 단위로 식 (7.22)를 적용. 식 (7.22)를 $\left\lceil \frac{n}{|M|} \right\rceil$ 번 반복하여 한 세대를 처리(M 은 미니 배치, n 은 훈련 집합의 크기)
- 미니 배치를 위한 평균제곱오차

$$J(\mathbf{W}) = \frac{1}{|M|} \sum_{\mathbf{x} \in M} \|\mathbf{y} - \mathbf{o}\|_2^2 \quad (7.23)$$

■ 옵티마이저

- SGD는 가장 오랫동안 애용해온 최적화 방법
- 코메터가 저으며 하스루 이요체 개서하 Adam, AdaGrad, RMSprop 드자(오자)

7.6.2 스토캐스틱 경사하강법

■ 역전파

- [그림 7-21]의 깊은 다층 퍼셉트론 가중치 집합은 $\mathbf{W} = \{\mathbf{U}^1, \mathbf{U}^2, \dots, \mathbf{U}^L\}$ 의 계층 구조
- 계층 구조에 따라 식 (7.18)은 복합 함수로 전방 계산
 - 맨 뒤에 있는 L 층의 가중치 \mathbf{U}^L 은 출력 벡터 \mathbf{o} 에만 영향
 - $L - 1$ 층의 \mathbf{U}^{L-1} 은 L 층 노드의 값에 영향. L 층 노드의 값은 출력 벡터에 영향
 - 일반적으로 l 층의 \mathbf{U}^l 은 $l + 1, l + 2, \dots$ 층에 순차적으로 영향을 미치고 출력 벡터에 영향
- 연쇄 법칙_{chain rule}을 적용하면, l 층의 그레이디언트는 $l + 1$ 층의 그레이디언트 $\frac{\partial J}{\partial \mathbf{U}^{i+1}}$ 에 두 층 사이의 그레이디언트 $\frac{\partial \mathbf{U}^{i+1}}{\partial \mathbf{U}^i}$ 을 곱하여 구할 수 있음
- [그림 7-21]은 L 층에서 시작해 역으로 $L - 1, L - 2, \dots, 1$ 층으로 진행하며 그레이디언트 $\frac{\partial J}{\partial \mathbf{U}^i}$ 계산하고 가중치 갱신 수행하는 과정 \leftarrow 역전파_{backpropagation} 알고리즘

7.6.2 스토캐스틱 경사하강법

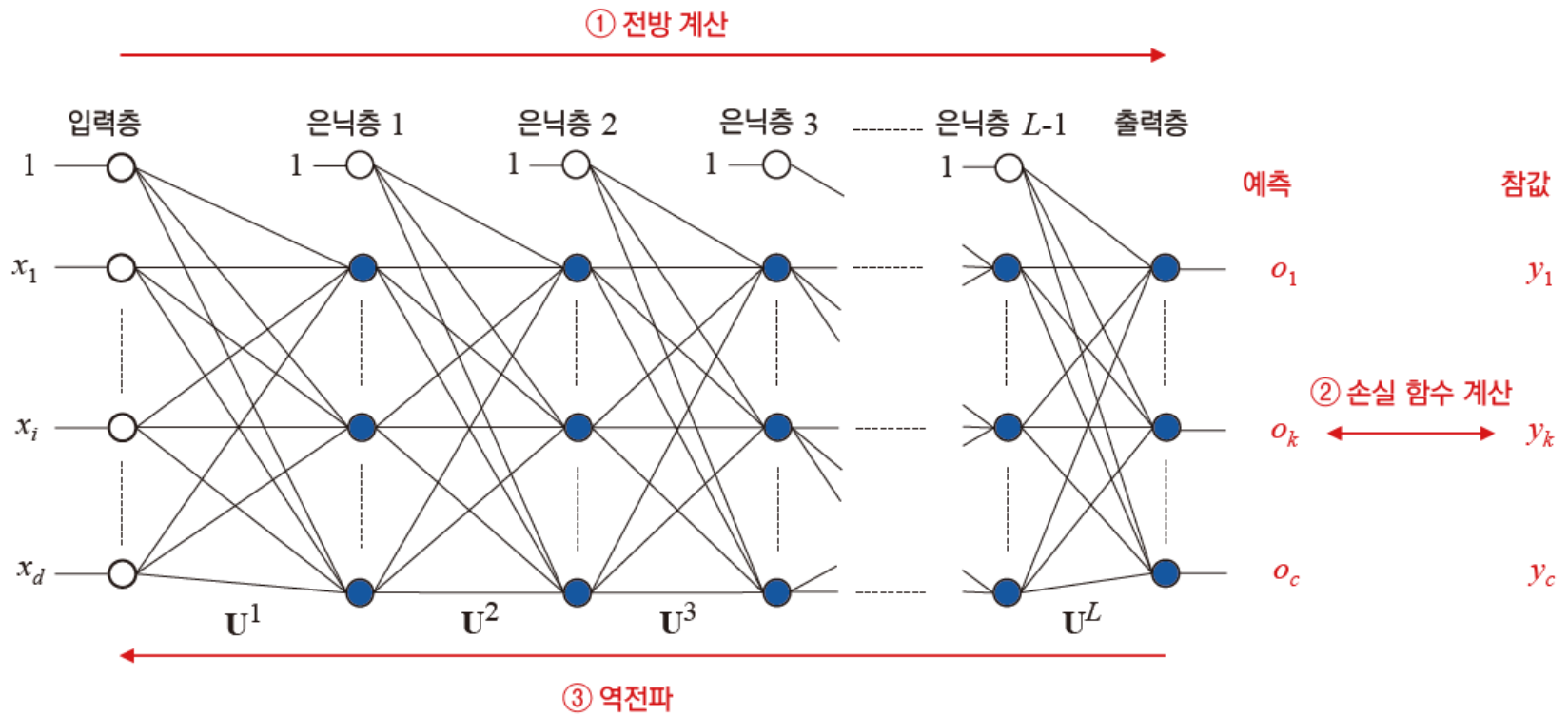


그림 7-21 신경망 학습 알고리즘

7.6.2 스톡캐스틱 경사하강법

[알고리즘 7-2] 스토캐스틱 경사하강법

입력: 훈련 데이터 $D=\{\mathbf{X}, \mathbf{Y}\}$, 세대 수 e , 학습률 ρ

출력: 최적의 가중치 $\hat{\mathbf{w}}$

1. 가중치 행렬 \mathbf{W} 를 난수로 초기화한다.
2. for $i=1,2,\dots,e$
3. $D'=D$
4. while D' 가 공집합이 아님
5. D' 에서 미니 배치 M 을 랜덤 선택하고 선택된 샘플을 D' 에서 제거한다.
6. M 으로 전방 계산을 수행하고 손실 함수 $J(\mathbf{W})$ 를 계산한다.
7. $\frac{\partial J}{\partial \mathbf{U}^{L+1}} = 1$ // 초깃값을 1로 두고 시작
8. for $l=L,L-1,\dots,2,1$
9. $\frac{\partial J}{\partial \mathbf{U}^{l+1}}$ 를 이용하여 $\frac{\partial J}{\partial \mathbf{U}^l}$ 을 계산한다. // 역전파
10. $\mathbf{U}^l = \mathbf{U}^l - \rho \frac{\partial J}{\partial \mathbf{U}^l}$ // 식 (7.22)
11. $\hat{\mathbf{W}} = \mathbf{W}$

7.7 다층 퍼셉트론 구현하기

■ 다층 퍼셉트론과 깊은 다층 퍼셉트론을 실습

- MNIST와 CIFAR-10 데이터셋 사용

7.7.1 필기 숫자 인식

프로그램 7-2

다층 퍼셉트론으로 MNIST 인식하기(SGD 옵티마이저)

```
01 import numpy as np
02 import tensorflow as tf
03 import tensorflow.keras.datasets as ds
04
05 from tensorflow.keras.models import Sequential
06 from tensorflow.keras.layers import Dense
07 from tensorflow.keras.optimizers import SGD
08
09 (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10 x_train=x_train.reshape(60000,784) [0,1]로 정규화
11 x_test=x_test.reshape(10000,784)
12 x_train=x_train.astype(np.float32)/255.0
13 x_test=x_test.astype(np.float32)/255.0
14 y_train=tf.keras.utils.to_categorical(y_train,10)
15 y_test=tf.keras.utils.to_categorical(y_test,10)
16
```

1차원 구조로 변환

데이터 준비

원한 코드로 변환

7.7.1 필기 숫자 인식

은닉층 노드 개수 512, 출력층 노드 개수 10

은닉층 활성화 함수 tanh, 출력층 활성화 함수 softmax

```
17 mlp=Sequential()
18 mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))
19 mlp.add(Dense(units=10,activation='softmax'))
20
21 mlp.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),
22             metrics=['accuracy'])
23
24 res=mlp.evaluate(x_test,y_test,verbose=0)
25 print('정확률=',res[1]*100) ②
```

입력층 모양

모델 선택
(신경망
구조 설계)

MSE 손실 함수

SGD 옵티마이저

학습률

학습

예측(성능 측정)

훈련 집합

미니 배치 크기

세대 수

7.7.1 필기 숫자 인식

Epoch 1/50 ①

469/469 - 1s - loss: 0.0876 - accuracy: 0.2390 - val_loss: 0.0842 - val_accuracy: 0.3410 - 1s/epoch - 3ms/step

Epoch 2/50

469/469 - 1s - loss: 0.0805 - accuracy: 0.4025 - val_loss: 0.0764 - val_accuracy: 0.4418 - 1s/epoch - 2ms/step

...

Epoch 49/50

469/469 - 1s - loss: 0.0189 - accuracy: 0.8870 - val_loss: 0.0179 - val_accuracy: 0.8941 - 1s/epoch - 2ms/step

Epoch 50/50

469/469 - 1s - loss: 0.0187 - accuracy: 0.8874 - val_loss: 0.0178 - val_accuracy: 0.8946 - 1s/epoch - 2ms/step

정확률= 89.4599974155426 ②

7.7.1 필기 숫자 인식

■ 텐서플로 프로그래밍 기초

- 아래 세 모듈은 텐서플로 프로그램의 기초
- **models 모듈** Sequential과 functional API의 두 모델을 제공한다. 다층 퍼셉트론처럼 왼쪽에서 오른쪽으로 계산이 한 줄기로 흐르는 경우에 Sequential을 쓴다.
- **layers 모듈** 여러 가지 층을 제공한다. 다층 퍼셉트론을 구성하는 완전연결층은 Dense 클래스로 쌓는다. 다른 종류의 층은 8장에서 소개한다.
- **optimizers 모듈** 학습 알고리즘이 사용하는 옵티마이저 함수를 제공한다. SGD, Adam, AdaGrad, RMSprop 등이 있다. 옵티마이저의 원리에 대해서는 8.6.2항에서 설명한다.

7.7.1 필기 숫자 인식

■ Adam 옵티마이저를 사용하여 성능 향상

- 21행에서 optimizer 인수를 Adam으로 바꿈

프로그램 7-3

다층 퍼셉트론으로 MNIST 인식하기(Adam 옵티마이저)

07행과 21을 제외한 다른 부분은 [프로그램 7-2]와 같음

```
07  from tensorflow.keras.optimizers import Adam

21  mlp.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics
    =['accuracy'])
```

7.7.1 필기 숫자 인식

빠른 수렴

```
Epoch 1/50
469/469 - 1s - loss: 0.0148 - accuracy: 0.9028 - val_loss: 0.0109 - val_accuracy:
0.9284 - 1s/epoch - 3ms/step
Epoch 2/50
469/469 - 1s - loss: 0.0089 - accuracy: 0.9428 - val_loss: 0.0073 - val_accuracy:
0.9523 - 1s/epoch - 2ms/step
...
Epoch 49/50
469/469 - 1s - loss: 2.4108e-04 - accuracy: 0.9987 - val_loss: 0.0029 - val_accuracy:
0.9822 - 1s/epoch - 2ms/step
Epoch 50/50
469/469 - 1s - loss: 2.8139e-04 - accuracy: 0.9984 - val_loss: 0.0029 - val_accuracy:
0.9819 - 1s/epoch - 2ms/step
정확률= 98.18999767303467
```

SGD에 비해 무려 8.73% 향상

7.7.2 성능 시각화

■ SGD와 Adam 성능을 그래프로 비교

프로그램 7-4

다층 퍼셉트론으로 MNIST 인식하기(SGD와 Adam의 성능 그래프 비교)

```
01 import numpy as np
02 import tensorflow as tf
03 import tensorflow.keras.datasets as ds
04
05 from tensorflow.keras.models import Sequential
06 from tensorflow.keras.layers import Dense
07 from tensorflow.keras.optimizers import SGD, Adam
08
09 (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10 x_train=x_train.reshape(60000,784)
11 x_test=x_test.reshape(10000,784)
12 x_train=x_train.astype(np.float32)/255.0
13 x_test=x_test.astype(np.float32)/255.0
14 y_train=tf.keras.utils.to_categorical(y_train,10)
15 y_test=tf.keras.utils.to_categorical(y_test,10)
16
17 mlp_sgd=Sequential()
18 mlp_sgd.add(Dense(units=512,activation='tanh',input_shape=(784,)))
19 mlp_sgd.add(Dense(units=10,activation='softmax'))
```

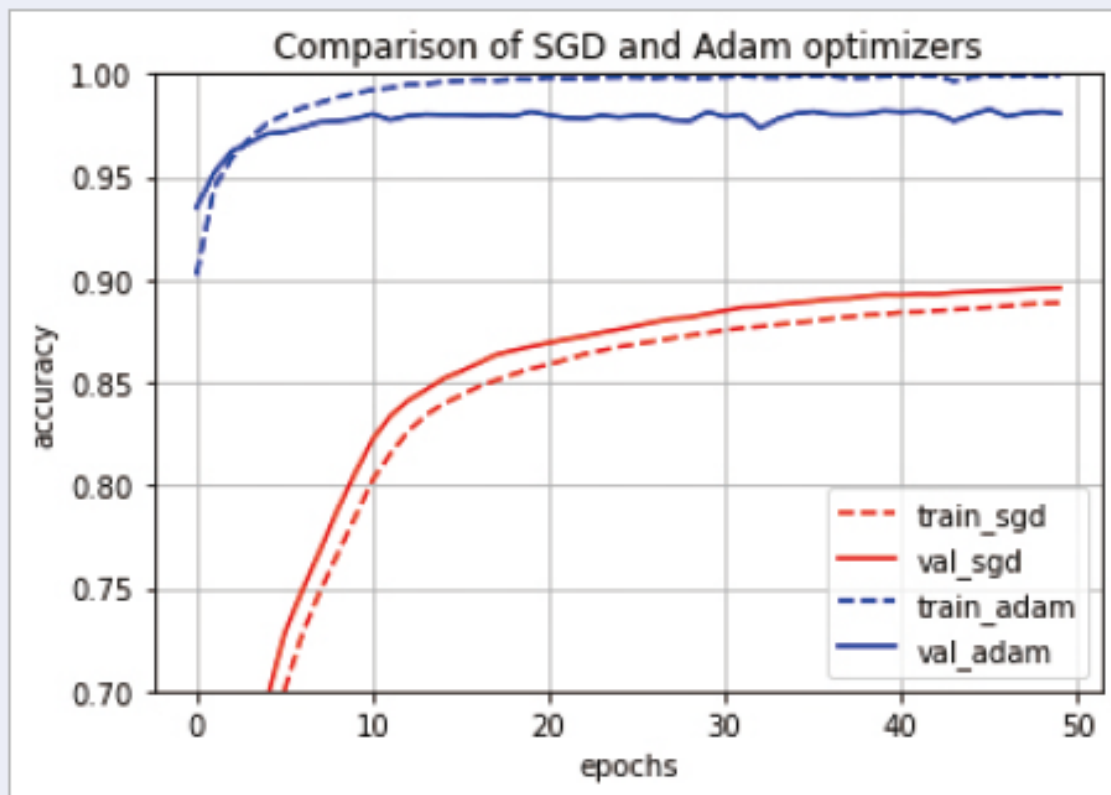
7.7.2 성능 시각화

```
20
21 mlp_sgd.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
22 hist_sgd=mlp_sgd.fit(x_train,y_train,batch_size=128,epochs=50,validation_
    data=(x_test,y_test),verbose=2)
23 print('SGD 정확률=',mlp_sgd.evaluate(x_test,y_test,verbose=0)[1]*100)
24
25 mlp_adam=Sequential()
26 mlp_adam.add(Dense(units=512,activation='tanh',input_shape=(784,)))
27 mlp_adam.add(Dense(units=10,activation='softmax'))
28
29 mlp_adam.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics
    =['accuracy'])
30 hist_adam=mlp_adam.fit(x_train,y_train,batch_size=128,epochs=50,validation_
    data=(x_test,y_test),verbose=2)
31 print('Adam 정확률=',mlp_adam.evaluate(x_test,y_test,verbose=0)[1]*100)
32
33 import matplotlib.pyplot as plt
34
35 plt.plot(hist_sgd.history['accuracy'],'r--')
36 plt.plot(hist_sgd.history['val_accuracy'],'r')
37 plt.plot(hist_adam.history['accuracy'],'b--')
38 plt.plot(hist_adam.history['val_accuracy'],'b')
39 plt.title('Comparison of SGD and Adam optimizers')
40 plt.ylim((0.7,1.0))
41 plt.xlabel('epochs')
42 plt.ylabel('accuracy')
43 plt.legend(['train_sgd','val_sgd','train_adam','val_adam'])
44 plt.grid()
45 plt.show()
```

7.7.2 성능 시각화

SGD 정확률= 89.60000276565552

Adam 정확률= 98.07999730110168



7.7.3 하이퍼 매개변수 다루기

■ 하이퍼 매개변수

- 신경망의 구조 또는 학습과 관련하여 사용자가 설정해야 하는 매개변수
- 아주 많은 하이퍼 매개변수가 있음
- 예) 층 수, 노드 수, 활성화 함수, 옵티마이저, 학습률, 미니배치 크기, 세대 수, 손실 함수 ...

■ 예) Dense 층과 Adam 옵티마이저의 API

좋은 값을 기본값으로 설정해 둬

```
tf.keras.layers.Dense(units, activation=None, use_bias=True, kernel_initializer="glorot_uniform", bias_initializer="zeros", kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None, **kwargs)
```

```
tf.keras.optimizers.Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, amsgrad=False, name="Adam", **kwargs)
```


7.7.3 하이퍼 매개변수 다루기

■ 하이퍼 매개변수 설정에 만능은 없음. 몇 가지 유용한 요령

- 텐서플로가 제공하는 기본값 사용
- 신뢰할 수 있는 논문이나 웹 사이트가 제공하는 권고 사항 따름
- 중요한 하이퍼 매개변수 1~3개에 대해 성능 실험을 통해 스스로 설정

7.7.3 하이퍼 매개변수 다루기


■ 깊은 다층 퍼셉트론으로 MNIST 인식

프로그램 7-5

깊은 다층 퍼셉트론으로 MNIST 인식하기

```
01 import numpy as np
02 import tensorflow as tf
03 import tensorflow.keras.datasets as ds
04
05 from tensorflow.keras.models import Sequential
06 from tensorflow.keras.layers import Dense
07 from tensorflow.keras.optimizers import Adam
08
09 (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10 x_train=x_train.reshape(60000,784)
11 x_test=x_test.reshape(10000,784)
12 x_train=x_train.astype(np.float32)/255.0
13 x_test=x_test.astype(np.float32)/255.0
14 y_train=tf.keras.utils.to_categorical(y_train,10)
15 y_test=tf.keras.utils.to_categorical(y_test,10)
16
17 dmlp=Sequential()
18 dmlp.add(Dense(units=1024,activation='relu',input_shape=(784,)))
19 dmlp.add(Dense(units=512,activation='relu'))
20 dmlp.add(Dense(units=512,activation='relu'))
21 dmlp.add(Dense(units=10,activation='softmax'))
```

신경망의 층을 4개로 늘림



7.7.3 하이퍼 매개변수 다루기

학습률을
0.0001로 설정

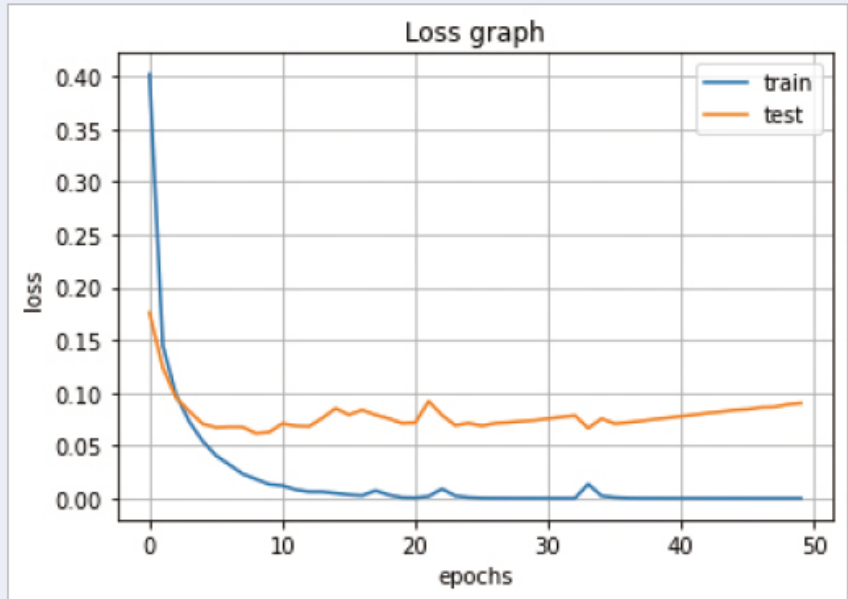
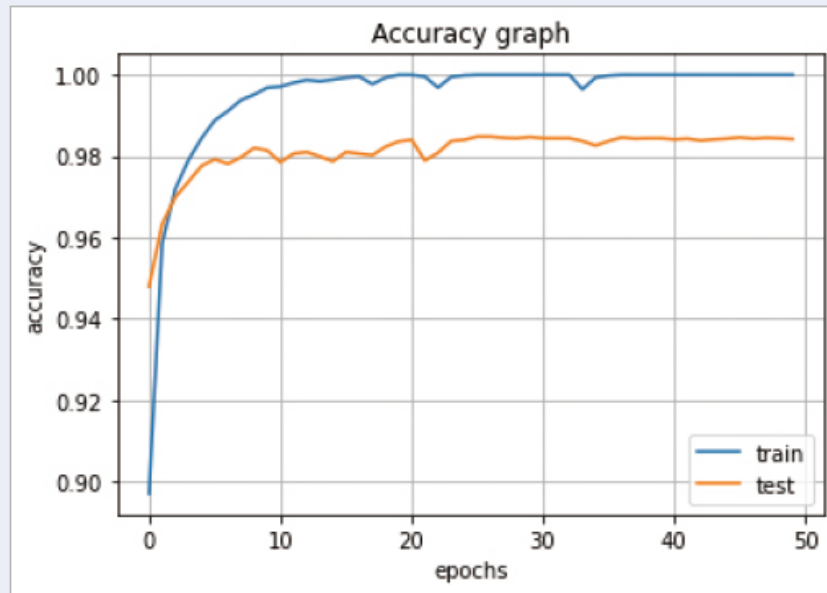
```
22
23 dmlp.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate
    =0.0001),metrics=['accuracy'])
24 hist=dmlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_
    test,y_test),verbose=2)
25 print('정확률=', dmlp.evaluate(x_test,y_test,verbose=0)[1]*100)
26
27 dmlp.save('dmlp_trained.h5')
28
29 import matplotlib.pyplot as plt
30
31 plt.plot(hist.history['accuracy'])
32 plt.plot(hist.history['val_accuracy'])
33 plt.title('Accuracy graph')
34 plt.xlabel('epochs')
35 plt.ylabel('accuracy')
36 plt.legend(['train','test'])
37 plt.grid()
38 plt.show()
39
40 plt.plot(hist.history['loss'])
41 plt.plot(hist.history['val_loss'])
42 plt.title('Loss graph')
43 plt.xlabel('epochs')
44 plt.ylabel('loss')
45 plt.legend(['train','test'])
46 plt.grid()
47 plt.show()
```

교차 엔트로피 손실 함수 사용

비전 에이전트 제작을 위해
학습된 모델을 저장

7.7.3 하이퍼 매개변수 다루기

정확률 = 98.42000007629395



7.7.3 하이퍼 매개변수 다루기

■ 하이퍼 매개변수 설정 체험

파란 박스는 신경망 구조
빨간 박스는 학습에 관련된 하이퍼 매개변수

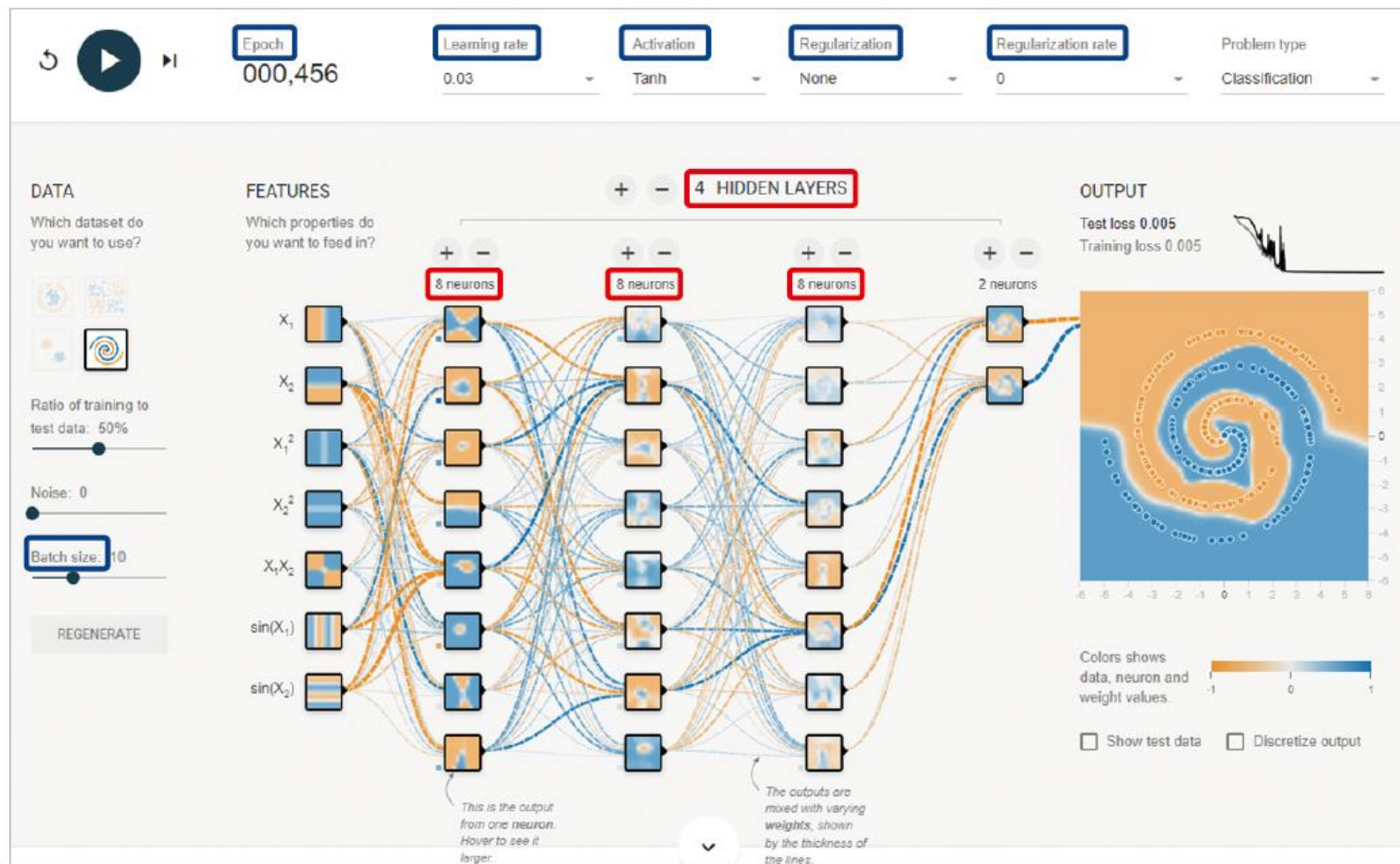
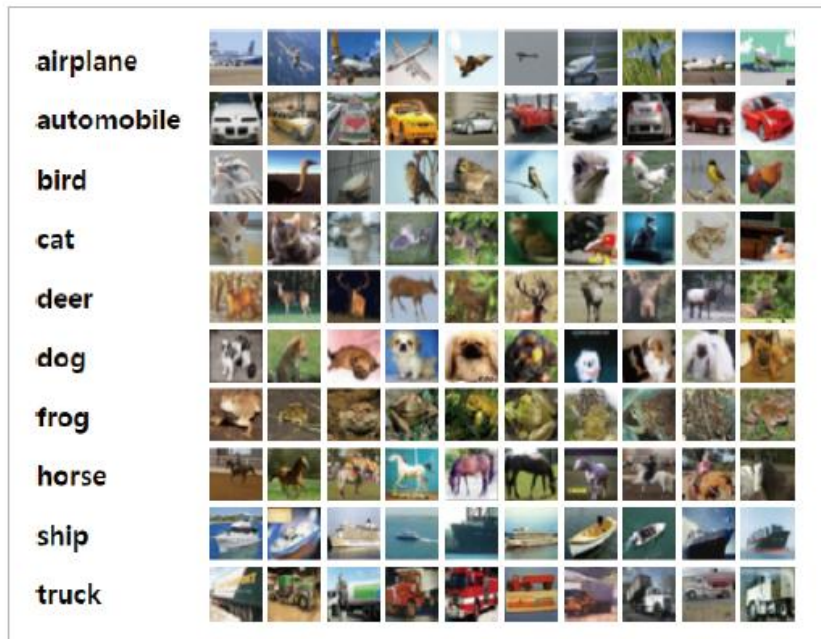


그림 7-22 하이퍼 매개변수 체험 사이트 <https://playground.tensorflow.org/>

7.7.4 자연 영상 인식

■ CIFAR-10과 CIFAR-100(장난감 자연영상 데이터셋)



(a) CIFAR-10

| Superclass | Classes |
|--------------------------------|---|
| aquatic mammals | beaver, dolphin, otter, seal, whale |
| fish | aquarium fish, flatfish, ray, shark, trout |
| flowers | orchids, poppies, roses, sunflowers, tulips |
| food containers | bottles, bowls, cans, cups, plates |
| fruit and vegetables | apples, mushrooms, oranges, pears, sweet peppers |
| household electrical devices | clock, computer keyboard, lamp, telephone, television |
| household furniture | bed, chair, couch, table, wardrobe |
| insects | bee, beetle, butterfly, caterpillar, cockroach |
| large carnivores | bear, leopard, lion, tiger, wolf |
| large man-made outdoor things | bridge, castle, house, road, skyscraper |
| large natural outdoor scenes | cloud, forest, mountain, plain, sea |
| large omnivores and herbivores | camel, cattle, chimpanzee, elephant, kangaroo |
| medium-sized mammals | fox, porcupine, possum, raccoon, skunk |
| non-insect invertebrates | crab, lobster, snail, spider, worm |
| people | baby, boy, girl, man, woman |
| reptiles | crocodile, dinosaur, lizard, snake, turtle |
| small mammals | hamster, mouse, rabbit, shrew, squirrel |
| trees | maple, oak, palm, pine, willow |
| vehicles 1 | bicycle, bus, motorcycle, pickup truck, train |
| vehicles 2 | lawn-mower, rocket, streetcar, tank, tractor |

(b) CIFAR-100

그림 7-23 인식 실험에 사용할 자연 영상 데이터셋

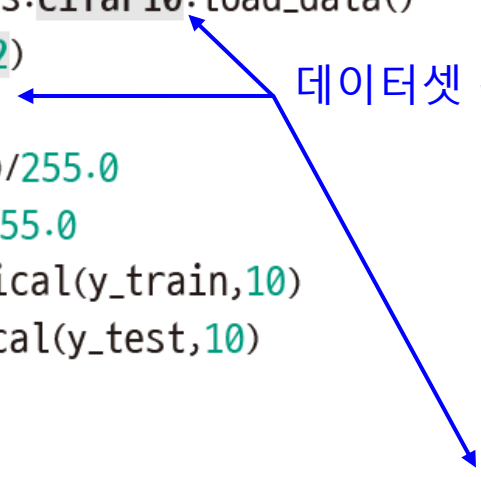
7.7.4 자연 영상 인식

프로그램 7-6

깊은 다층 퍼셉트론으로 CIFAR-10 인식하기

```
01~07행은 [프로그램 7-5]와 같음
08 ... ..
09 (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
10 x_train=x_train.reshape(50000,3072)
11 x_test=x_test.reshape(10000,3072)
12 x_train=x_train.astype(np.float32)/255.0
13 x_test=x_test.astype(np.float32)/255.0
14 y_train=tf.keras.utils.to_categorical(y_train,10)
15 y_test=tf.keras.utils.to_categorical(y_test,10)
16
17 dmlp=Sequential()
18 dmlp.add(Dense(units=1024,activation='relu',input_shape=(3072,)))
19 dmlp.add(Dense(units=512,activation='relu'))
20 dmlp.add(Dense(units=512,activation='relu'))
21 dmlp.add(Dense(units=10,activation='softmax'))
...
23~47행은 [프로그램 7-5]와 같으며 27행의 dmlp.save('dmlp_trained.h5')는 삭제
```

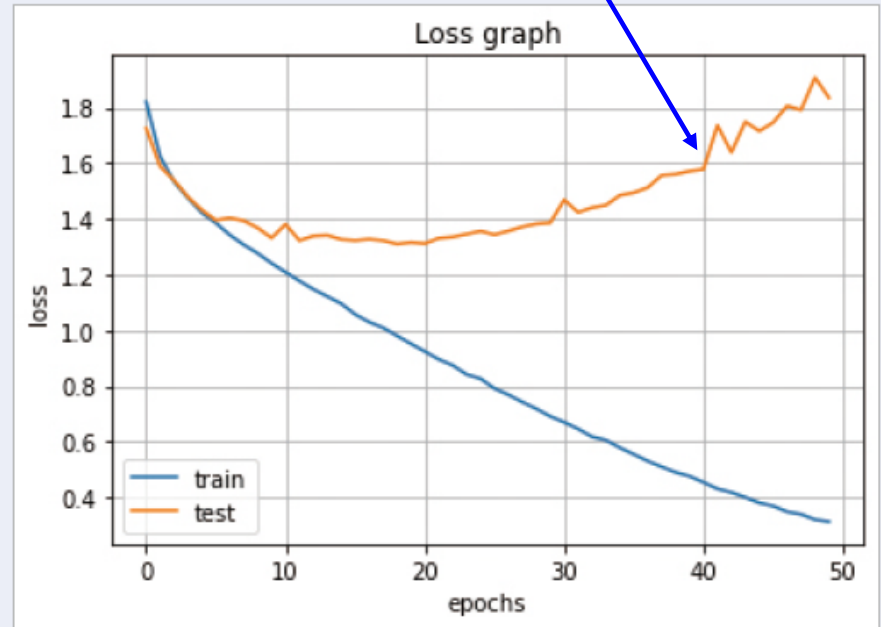
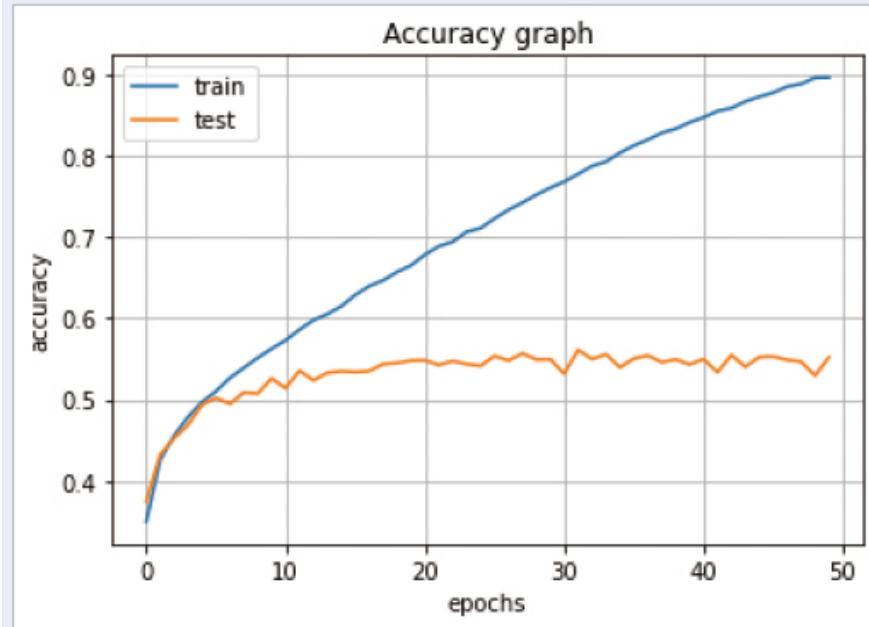
데이터셋 관련 부분만 수정



7.7.4 자연 영상 인식

과잉 적합(8.6.3절에서 설명)

정확률= 55.15999794006348




7.8 비전 에이전트: 우편번호 인식기 v.1

프로그램 7-7

우편번호 인식기 v.1(DMLP 버전) 구현하기

```
01 import numpy as np
02 import tensorflow as tf
03 import cv2 as cv
04 import matplotlib.pyplot as plt
05 import winsound
06
07 model=tf.keras.models.load_model('dmlp_trained.h5')
08
09 def reset():
10     global img
11
12     img=np.ones((200,520,3),dtype=np.uint8)*255
13     for i in range(5):
14         cv.rectangle(img,(10+i*100,50),(10+(i+1)*100,150),(0,0,255))
15         cv.putText(img,'e:erase s:show r:recognition q:quit',(10,40),cv.FONT_
HERSHEY_SIMPLEX,0.8,(255,0,0),1)
16
17 def grab_numerals():
18     numerals=[]
```

[프로그램 7-5]에서
저장해 둔 모델 파일 읽어옴



7.8 비전 에이전트: 우편번호 인식기 v.1

```
19     for i in range(5):
20         roi=img[51:149,11+i*100:9+(i+1)*100,0]
21         roi=cv.resize(roi,(28,28),interpolation=cv.INTER_CUBIC)
22         numerals.append(roi)
23     numerals=np.array(numerals)
24     return numerals
25
26 def show():
27     numerals=grab_numerals()
28     plt.figure(figsize=(25,5))
29     for i in range(5):
30         plt.subplot(1,5,i+1)
31         plt.imshow(numerals[i],cmap='gray')
32         plt.xticks([]); plt.yticks([])
33     plt.show()
34
35 def recognition():
36     numerals=grab_numerals()
37     numerals=numerals.reshape(5,784)
38     numerals=numerals.astype(np.float32)/255.0
39     res=model.predict(numerals) # 신경망 모델로 예측
40     class_id=np.argmax(res,axis=1)
41     for i in range(5):
42         cv.putText(img,str(class_id[i]),(50+i*100,180),cv.FONT_HERSHEY_
43                     SIMPLEX,1,(255,0,0),1)
44     winsound.Beep(1000,500)
```

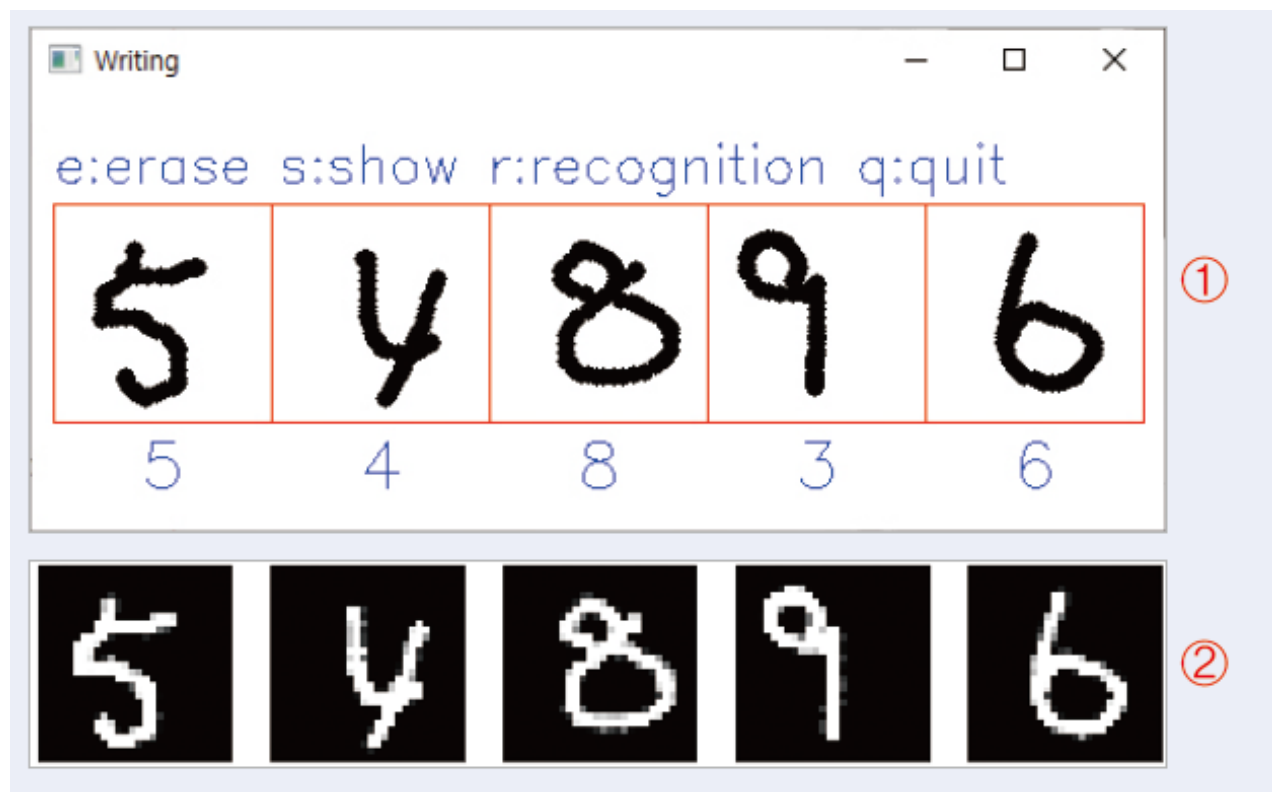
7.8 비전 에이전트: 우편번호 인식기 v.1

```
44
45 BrushSiz=4
46 LColor=(0,0,0)
47
48 def writing(event,x,y,flags,param):
49     if event==cv.EVENT_LBUTTONDOWN:
50         cv.circle(img,(x,y),BrushSiz,LColor,-1)
51     elif event==cv.EVENT_MOUSEMOVE and flags==cv.EVENT_FLAG_LBUTTON:
52         cv.circle(img,(x,y),BrushSiz,LColor,-1)
53
54 reset()
55 cv.namedWindow('Writing')
56 cv.setMouseCallback('Writing',writing)
57
```

7.8 비전 에이전트: 우편번호 인식기 v.1

```
58 while(True):
59     cv.imshow('Writing',img)
60     key=cv.waitKey(1)
61     if key==ord('e'):
62         reset()
63     elif key==ord('s'):
64         show()
65     elif key==ord('r'):
66         recognition()
67     elif key==ord('q'):
68         break
69
70 cv.destroyAllWindows()
```

7.8 비전 에이전트: 우편번호 인식기 v.1



상당히 많이 틀리는데, 미국 사람이 종이에 쓴 숫자를 스캐너로 수집한 MNIST와 한국인이 화면에 마우스로 쓴 패턴이 다른 탓.
8장 컨볼루션 신경망은 획기적으로 성능 향상. 8장에서 버전 2 작성