3주차 chap07

홀수번 문제풀이

chap7 딥러닝 비전

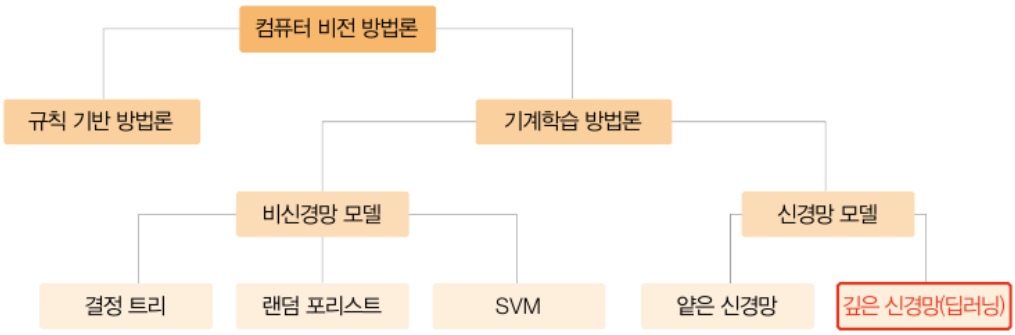
7.1 방법론의 대전환

고전적 컴퓨터 비전 "규칙기반" 의 한계

사람이 면밀히 데이터를 관찰해 정교하게 알고리즘 개발 및 개선 => 수작업 특징 / 수작업 알고리즘
수작업 알고리즘 결과 => 규칙으로 표현

- 이런 규칙기반 알고리즘은 사람의 노력으로 일정 수준 달성하나, 그 이상을 돌파하기 어려움
  ex. 더 좋은 가우시안 필터는 없을까? 데이터에 따라 최적의 필터를 설계해야 하지 않을까?

↔ 현대적 컴퓨터 비전 "데이터 중심 딥러닝"(=기계학습) 은 위의 한계점 보완



7.2 기계학습의 기초

$$\text{기계학습 모델: } y=f(x)$$

- 모델 : 기계학습에서 함수
- 학습 : 수집한 데이터로 방정식을 풀어 함수(모델)을 알아내는 일

<예측에 쓸 모델 얻는 방법>

1. 데이터 수집
- 훈련집합 : 수집한 샘플의 모음 (보통, 샘플의 개수 : n)
2. 모델 선택
   선형과 비선형의 다양한 함수 관계 적용 가능
3. 학습
   훈련 집합에 있는 n개의 샘플을 모델에 대입한 방정식 풀이

-> 가중치 값 도출
-> 데이터 학습된 모델 확인

4. 예측
훈련 집합에 없는 새로운 샘플을 대입해 예측
ex) x=X 대입 -> 모델 기반 y 예측

STEP1 : 데이터 수집

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

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

- 입력 : x 특징벡터 feature vector
  - d개의 특징으로 구성되는 벡터
  -
- 출력 : y 참값 GT: Ground Truh or 레이블 label
  - 학습된 모델로 x에 대해 y를 계산하는 예측 문제로 y 연속된 값 => 회귀 regression
  - 입력영상에서 물체 부류를 알아내는 문제로 y 이산값 => 분류 classification
- 데이터셋 D : 특징 벡터(x)마다 참값(y)이 붙어 있는 n개 샘플로 구성

데이터 수집 시, 비용이 많이 듦
분할의 경우 물체가 차지한 영역을 화소 수준으로 지정해야 하기에 노동 집약적

STEP2 : 모델 선택

입력가 출력의 관계가 더 복잡할수록 비선형 모델 사용
학습이 추정할 가중치가 핵심 포인트

딥러닝은 수십 개 층으로 구성되어 각 층마다 가중치 존재
가중치가 수만~수억 개인 큰 용량의 모델 사용
가중치를 매개변수라고 부르기도 함

W = {U^1, U^2, U^3,... , U^L}

- W : 가중치 행렬
- U : 각 층의 가중치 벡터
- L : 층의 개수

STEP3 : 학습

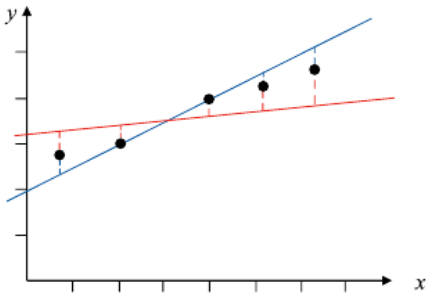
훈련 집합에 있는 샘플을 **최소 오류로 맞히는 최적의 가중치 값 도출**하는 과정

- 분석적 방법 : 모델 함수에 데이터셋을 대입해 만든 방정식을 풀어 가중치 최적값 도출
- 수치적 방법 : 분석적 방법에서 발생하는 잡음, 모델 분포와 데이터 분포의 불일치 등으로 발생하는 오류를 줄이는 과정을 반복

평균제곱오차MSE
최적화 알고리즘 (옵티마이저 optimizer) : 평균제곱오차를 최소로하는 가중치의 최적값 도출
** 최적의 가중치 찾는 방법 = 오류를 얼마나 덜 범하는지 측정하는 함수 필요
=> 손실함수 loss function

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

- 0에 가까울수록 좋음



- 검은점 : y값 <-> 직선상의 점 : 모델의 예측값
- 점선 : 오류, 예측값 - y

=> 모든 점선 길이의 제곱의 평균을 구하면 평균제곱오차
빨간색, 파란색 모델 중 파란색 모델의 평균제공오차가 더 작아 파란색 모델이 최적

STEP4 : 예측

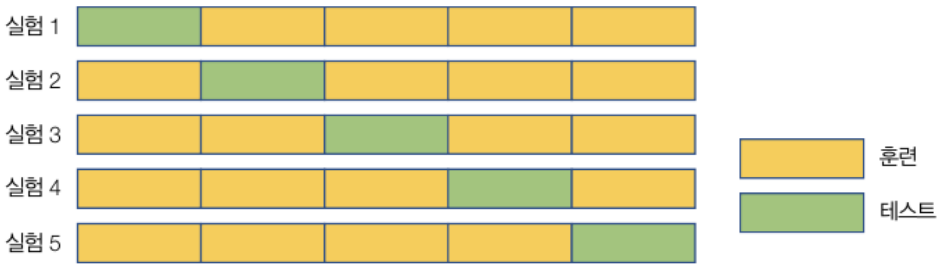
예측 : 학습을 마친 모델, 가중치의 최적값을 가진 모델에 학습 시 사용하지 않던 새로운 특징 벡터를 입력하고 출력을 구하는 과정

- 출력 구함
- 모델 성능평가

모델 성능 평가 방법

1. 바로 현장에 설치해 정확률 측정
   -> 돈 많이 듦
2. 데이터셋을 일정비율로 분할해 일부는 훈련 집합에 넣어 학습 사용, 나머지는 테스트 집합에 넣어 예측 사용
   -> 객관성 보장 but 데이터 셋 크기가 충분히 크지 않은 상태에서는 좋지 않을 수도.
   ex. `train_test_split(X, y, test_size = 0.4)`
   데이터를 train, test 로 60:40으로 분할

ex. K-fold 교차검증



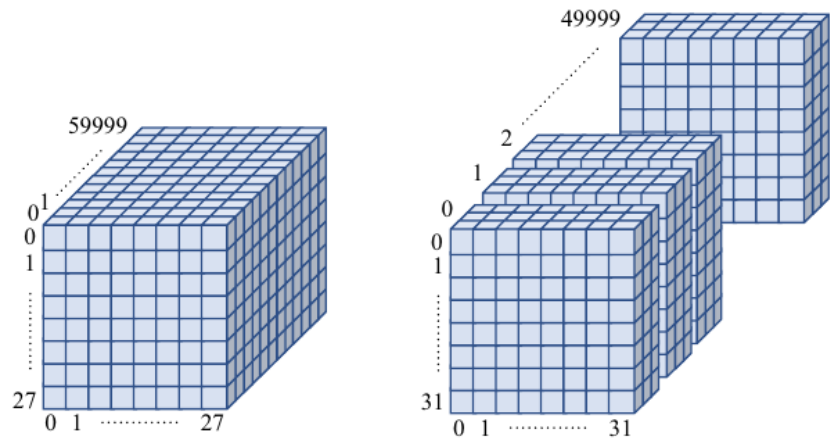

```
1. 데이터 k개 분할
2. 성능실험 k번해서 평균을 취함
3. i = 1, 2, .., k 로 바꾸며 i번째 실험에서 i번 부분집합을 test 데이터셋으로 사용, 나머지 k-1개 부분 집합은 train 데이터 셋으로 사용
```

7.3 딥러닝 소프트웨어 맛보기

tensorflow

- 기계학습 구현을 위한 소프트웨어 도구
- tensor : 딥러닝에서 다차원배열을 이르는 말

- tesorflow 내에서 Keras 명령어로 딥러닝 개발
- https://www.tensorflow.org/datasets?hl=ko



(a) MNIST의 x_train (b) CIFAR-10의 x_train

그림 7-8 데이터셋의 텐서 구조

(a). MNIST : 70,000개의 필기 숫자 샘플 존재 (train : 60,000 / test : 10,000 집합으로 분할)

특징 벡터(x)는 숫자를 28 * 28 맵으로 표현
참값(y)는 숫자 부류를 나타내기 위해 0~9 사이의 값 보유
ex. x_train : 60000x28x28 으로 28x28 맵이 60,000장 쌓여 있는 3차원 구조

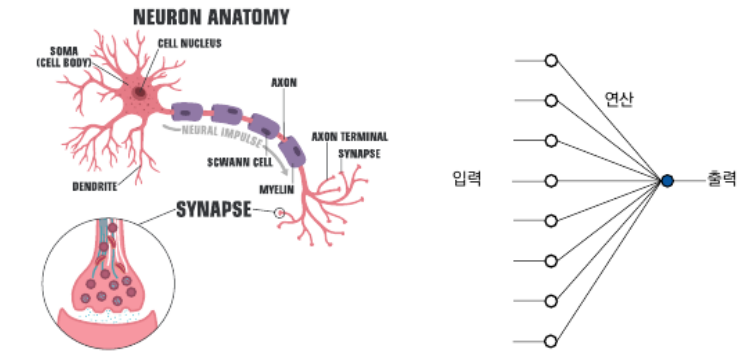
(b). CIFAR-10 : 60,000개의 필기 숫자 샘플 존재 (train : 50,000 / test : 10,000 집합으로 분할)

특징 벡터(x)는 숫자를 32 * 32 * 3 맵으로 표현, RGB를 위해 32x32맵이 3장 있는 구조
참값(y)는 숫자 부류를 나타내기 위해 0~9 사이의 값 보유

7.4 인공신경망의 태동

인공 신경망 : 뉴런의 정보 처리 과정을 수학적으로 모델링한 신경망

- 사람 뇌는 단순한 연산 장치가 엄청나게 밀집된 형태로 연결된 병렬 처리기



(a) 뇌의 정보 처리 단위인 뉴런 (b) 뉴런을 모방한 퍼셉트론

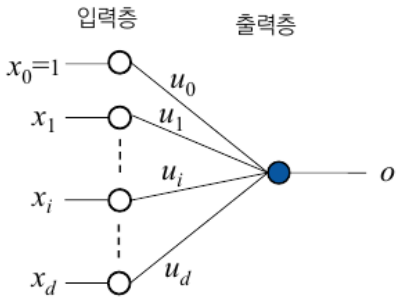
그림 7-9 생물 신경망과 인공 신경망

퍼셉트론

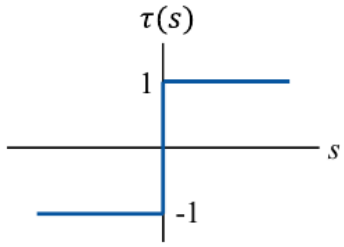
- 인공신경망의 구성 요소로 다수의 값을 특징벡터(x)를 입력 받아 연산을 수행하고 결과(y)를 출력층으로 내보냄
- Perceptron은 perception과 neuron의 합성어이며 인공 뉴런

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

활성함수logit는 계단함수를 사용함

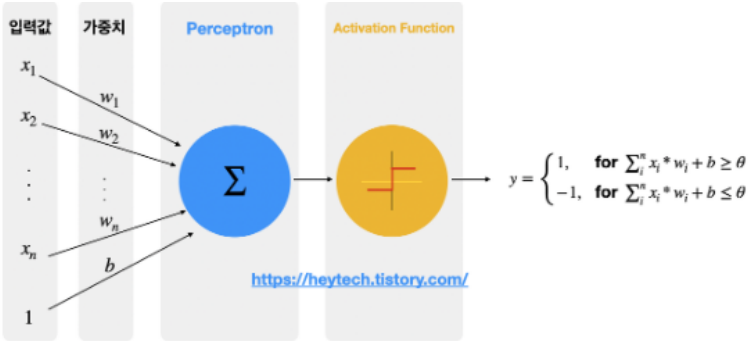


(a) 퍼셉트론의 구조

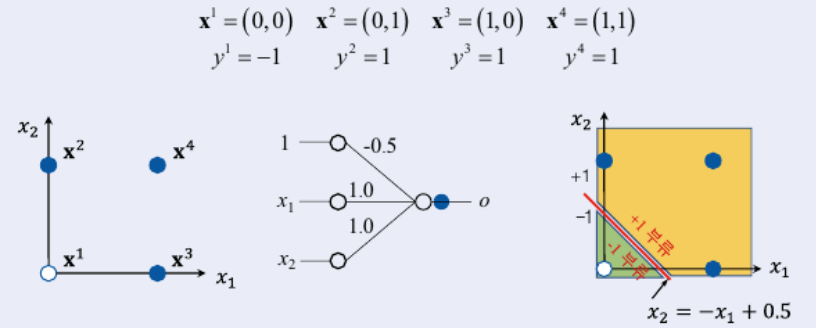


(b) 퍼셉트론이 사용하는 활성 함수

- 입력층 : d+1개의 노드
  - 특징값을 통과시키는 일만 수행해 속이 빈 원
  - d차원의 특징 벡터를 받기 위한 d개 노드 + 1개의 바이어스 노드
  - 바이어스 노드의 역할은 딥러닝 모델의 최적화 요소이며 x0=1로 설정
  - 가중합의 크기는 편향의 크기로 조절할 수 있기에, 편향이 퍼셉트론의 출력값 y를 결정짓는 중요 변수
- 가중치 u : 입력 노드와 출력 노드를 연결하는 에지에 위치
- s : 입력 노드값 x와 가중치 u를 곱해 얻은 바이어스 포함한 d+1개의 곱셈결과를 합함
  - 활성함수에 통과시켜 얻은 값 출력
- 출력층 : 1개 노드
  - 연산을 수행하기 때문에 속이 찬 파란원



OR 분류기



참 분류인지 확인가능
u0 = -0.5, u1=1, u2=1의 가중치를 가질 때, x2=(0,1)입력시 예측값이 1로 y2=1이므로 옳게 분류됐음
BUT. 이런 상황에서 바이어스가 없다면 직선이 항상 원줌을 지남
-> 데이터를 제대로 분류하기 어려움
-> 특징 공간을 두 부분 공간으로 나누는 분류기

행렬표기
표현 방식만 다르고 계산 방법은 동일

$$\mathbf{O} = \tau\left(\mathbf{u}\mathbf{X}^{\mathrm{T}}\right) \qquad (7.7)$$

X design matrix(설계행렬) : 특징벡터에서 데이터셋에 있는 n개의 샘플 적용한 행렬

- 특징벡터 x가 행에 배치된 행렬
- 특징벡터란 d차원 특징벡터에 바이어스값 추가해 (d+1)차원으로 확장한 벡터

[예시 7-3] OR 데이터셋을 행렬 표기로 처리

식 (7.7)의 행렬 표기를 따르면 OR 데이터셋은 아래와 같다.

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

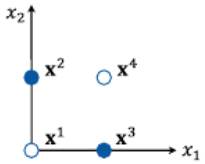
바이어스 값을 추가하고 전치를 적용한 $\mathbf{X}^T$는 아래와 같다. 바이어스를 파란색으로 표시해 쉽게 확인할 수 있게 하였다.

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

$\mathbf{X}^T$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.

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

7.5 깊은 다층 퍼셉트론



(a) XOR 데이터셋

퍼셉트론의 한계 : 선형 분리가 가능한 데이터셋만 100% 정확률로 분류 가능
-> XOR문제와 함께 해결 방안으로 은닉층을 추가한 다층 퍼센트론 제시

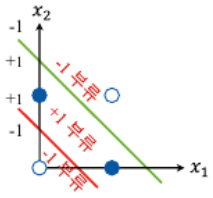
다층퍼셉트론 적용 순서

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

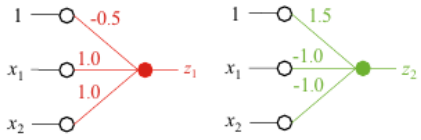
STEP1

[특징공간 x] 퍼셉트론 2개 사용해 특징공간 2+1=3개의 영역으로 분할 가능



STEP2

[특징공간 x] 데이터를 두 퍼셉트론에 적용해 출력 z1, z2로 지정
(z1,z2) 모두 +1이면 +1 분류, 그 외의 경우 모두 -1 부류로 분류

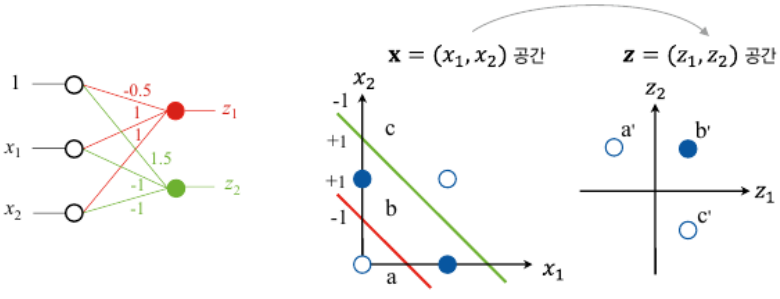


STEP3

[은닉공간 z] 기존 특징공간 x를 새로운 특징공간 z로 변환가능

-> 새로운 z공간에서 세 점으로 변함

2번에서 진행한 두개의 퍼셉트론 출력으로 4개의 샘플 z로 표현 가능

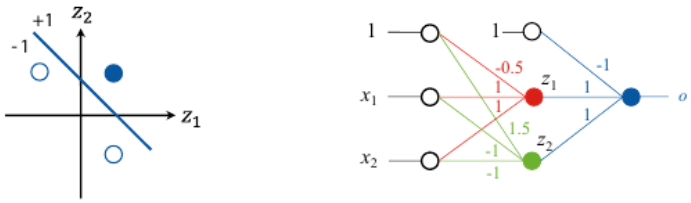


(a) 두 퍼셉트론의 병렬 결합 (b) 특징 공간 변환 – 세 영역이 세 점으로 변환

그림 7-14 특징 공간의 변환

STEP4

[출력 o, 레이블 y] z공간에서 분류를 수행하는 퍼셉트론을 하나 더 사용하면 XOR 해결



(a) z 공간을 분할하는 세 번째 퍼셉트론 (b) 세 번째 퍼셉트론의 순차 결합

그림 7-15 다층 퍼셉트론

위의 스탭들을 거쳐 만들어진 다층퍼셉트론의 일반적인 구조

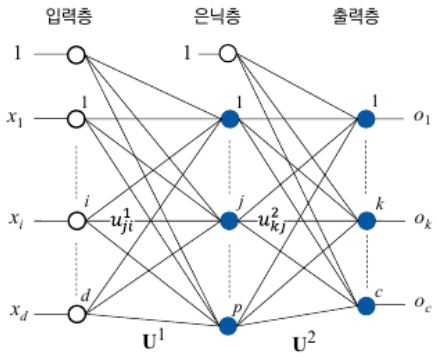


그림 7-16 다층 퍼셉트론의 일반적인 구조

- 입력층은 특징을 통과시키는 일만 하여 연산은 수행하지 않아 층으로 count X
- 은닉층 : 입력층과 출력층 사이에 새로 생긴 층
  - 은닉공간 : 은닉층이 형성하는 새로운 특징공간
- 완전 연결 구조 : 이웃한 두 층에 있는 노드의 모든 쌍에 에지 존재
- 연산이 일어나는 층을 강조하기 위해 노드 파란색 표시
- 특징 벡터의 차원이 d, 부류의 개수가 c인 데이터가 있다면

  입력층의 노드개수 *d+1*, 출력층의 노드 개수 *c*

  은닉층의 노드 개수 p는 사용자 지정

  → 사용자가 지정해줘야 하는 매개변수를 하이퍼 매개변수

  → 이를 잘 설정해야 신경망이 높은 성능

가중치행렬 : 이웃한 층을 연결하는 가중치

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

- U^1 : 입력층과 은닉층을 연결하는 가중치 행렬

  은닉층의 j번째 노드와 입력층의 i번째 노드를 연결하는 에지 가중치 (u_ji)^1
- U^2 : 은닉층과 출력층을 연결하는 가중치 행렬

  출력층의 k번째 노드와 은닉층의 j번째 노드를 연결하는 에지 가중치 (u_kj)^2

전방계산

전방계산 : 특징벡터 x가 입력층으로 들어가 은닉층과 출력층을 거치며 순차적으로 연산 수행 과정

은닉층 연산

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

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

- z_j : 은닉 노드의 출력으로 그 다음 층인 출력층의 입력으로 사용
- 활성화 함수 입력을 **로짓**
- (u_j)^1 : U^1의 j 번째 행 → j번째 은닉 노드에서 연결된 d+1개의 가중치

$$u_j^1 = (u_{(}j0)^1, u_{(}j1)^1, ..., u_{(}jd)^1)$$

출력층 연산

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

- k번째 출력노드는 은닉층의 연산 결과 z를 입력받음
- (u_k)^2 : U^2의 k 번째 행 → k번째 출력 노드에서 연결된 p+1개의 가중치

$$u_k^2 = (u_{(}k0)^2, u_{(}k1)^2, ..., u_{(}kp)^2)$$

위의 연산과정을 층에 있는 모든 노드의 연산으로 확대

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

활성화함수가 다름

(7.12) 를 식변형해 **(7.13)**

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

위는 하나의 샘플 x를 처리하는 경우이며

아래는 데이터셋 전체를 한꺼번에 처리하는 경우

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

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

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

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

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_1은 활성 함수를 적용할 뿐 아니라 계산 결과에 은닉층의 바이어스를 추가하는 역할까지 한다고 가정한다. 나머지 3개 샘플에 대한 계산은 연습문제로 남겨둔다.

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

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

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

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

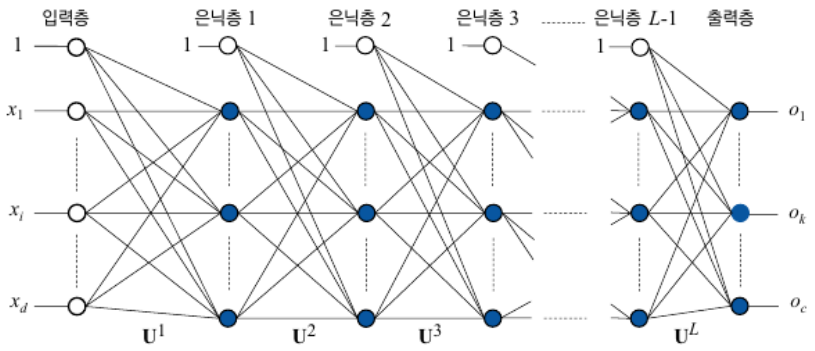
$$=\tau_2\left(\begin{pmatrix}-1 & 1 & 1\end{pmatrix}\tau_1\left(\begin{pmatrix}-0.5 & 0.5 & 0.5 & 1.5\\ 1.5 & 0.5 & 0.5 & -0.5\end{pmatrix}\right)\right)$$

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

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

다층퍼셉트론의 대체제로 얕은 신경망 사용

깊은 다층 퍼셉트론



은닉층 각각의 가중치 행렬은 U^1, U^2, … ,U^L

U^L인 I번째 층의 **가중치 행렬**

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

*l*번째 층의 *j*번째 노드와 *l-1*번째 층의 *i*번째 노드 연결하는 가중치 *(u_ji)^l*

층번호는 입력층을 0번째 층, 은닉층 l을 l번째 층, 출력층을 L번째 층

연산 방법

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

특징 벡터 *x*가 입력층으로 들어와 은닉층1, 은닉층2, …, 출력층의 순서대로 연산

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

해석

전방 계산 수행이후, 신경망의 최종 출력$o = (o_1, o_2, ..., o_c)$ 을 부류 정보로 해석함

- c : 출력노드 개수 == 부류의 수

$$\text{최종 부류: } \hat{k} = \underset{k}{\operatorname{argmax}}\, o_k \qquad (7.19)$$

몇번째 값이 최댓값인지가 핵심 ..?

o =(0.0 0.1 0.8 … 0.1)로 c=10일때 세번째 값이 최대이므로 0,1,2..,9 순서로 부류 번호 부여시 숫자 2로 인식

활성함수

앞서 계단 함수를 사용해 출력이 -1 or 1

→ 한계 : 이것으로 모든걸 표현하기 어려움

→ 해결방안 : 계단함수를 닮았지만 매끄럽게 변하는 시그모이드 함수 적용

s : 활성함수의 입력으로, 로짓값

b (베타) : 함수 모양 제어

ex.

- logistic sigmoid의 경우, 베타가 크면 0 근방에서 가팔라져 계단 함수 모양
- Swish의 경우, 베타가 작으면 ReLU

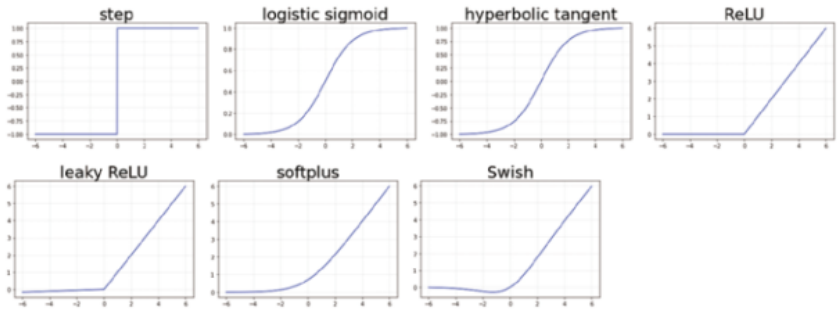


그림 7-18 신경망이 사용하는 다양한 활성 함수

표 7-2 다양한 활성 함수 $\tau(s)$

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

- logistic sigmoid : 0 근방에서 급격히 변하지만 0에서 멀어지면 변화량 아주 작음

  0에서 먼 곳의 미분값은 0
- hyperbolic tangent : 0 근방에서 급격히 변하지만 0에서 멀어지면 변화량 아주 작음

  0에서 먼 곳의 미분값은 0
- ReLU

  양수 구간이면 어느 곳에서나 미분값이 1이기에 그레디언트 소멸 현상을 누그러뜨려 주로 깊은 딥러닝에서 주로 사용

  → 출력층에서 입력층으로 진행하며 미분값을 이용해 가중치 갱신하는 방식으로 진행
- softmax : 신경망의 출력층으로 주로 사용

  모든 노드의 값을 합하면 1이 되는 성질$o_1 + o_2 + ... + o_c = 1$ 을 만족해 확률로 간주할 수 있어 신경망의 최종 출력으로 적합

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

7.6 학습 알고리즘

→ 가중치 설정도 **학습 알고리즘** 사용

학습 알고리즘은 샘플로 높은 정확률로 가중치를 알아내는 최적화 문제 해결해야 함

[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.
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. $\hat{\mathbf{W}}=\mathbf{W}$

1. 가중치를 어떻게 설정해야 높은 정확률로 얻을 수 있는 지에 대한 아무런 실마리 없기에 **난수설정**
2. 2-6과정으로 충분히 높은 정확률을 달성 or

   더 이상 개선이 불가능하다고 여겨질 때까지 정확률을 향상하는 방향으로 가중치 갱신 반복

   - 3행

     손실함수 J(W)로는 신경망의 예측과 참값의 차이인 오류를 평균한 평균제곱오차 사용
   - 4행

     수렴조건 확인
   - 5행

     불만족 시, **오류를 줄이**는 방향인 *델타 W* 계산 → *델타 W*는 미분으로 계산
   - 6행

     현재 가중치 W에 개선 방향 델타 W를 더해 **가중치 갱신**

스토캐스틱 경사하강법

위에서 설명한 알고리즘에서 5행, 6행이 손실함수를 줄이도록 가중치 갱신하는 것이 가장 중요

경사하강법 : 미분값을 보고 함수가 낮아지는 방향을 찾아 이동하는 일을 반복하여 최저점에 도달하는 알고리즘

예시 7-5 : 단순한 신경망의 학습

전제 조건

- 층에 노드가 하나뿐인 신경망

$$\text{신경망 연산: } o=ux$$

- 훈련집한은 샘플 하나뿐, 활성화 함수는 s가 들어가면 s가 나오는 항등함수
- 신경망의 연산과 손실함수인 평균제곱오차 J

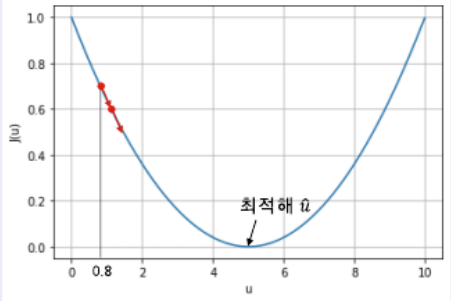
$$\text{손실 함수: } J(u)=(y-ux)^2=x^2u^2-2yxu+y^2$$

- J의 도함수는 J가 최소인 u를 구하는 문제이기에 J를 u로 미분한 것 구함

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

1행에서 난수 생성해 u = 0.8(1행)이면, 점 D=(0.2, 1)일때 적용 → 5행

$$\frac{dJ}{du} = 2(0.2)^2(0.8) - 2(1)(0.2) = -0.336$$



(b) 손실 함수 J

미분값이 음수라는 사실은 u가 증가하면 J가 감소한다는 의미로 기울기가 더 낮은 쪽으로 이동 (6)

$$u = u + \Delta u = u + \left(-\frac{dJ}{du}\right) = 0.8 + 0.336 = 1.136$$

새로운 점에서 미분값 다시 계산하기

$$\frac{dJ}{du} = 2(0.2)^2(1.136) - 2(1)(0.2) = -0.309$$

위의 과정을 반복하면 u햇=5에 더 가까워짐

$$u = 1.136 + 0.3091 = 1.4451$$

위의 과정을 일반화

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

미분의 이유로는 최적해에 다가갈 방향알려줌, but 이동량은 몰라…

→ 학습률 p를 사용해서 작은 값으로 설정해 조금씩 이동하는 보수적인 전략 사용

딥러닝에 이를 적용하기 위해 !

1. 신경망의 가중치는 층을 구성하고 수만~수억 개라는 사실 반영

   가중치가 여러개일때 아래와 같이 확장

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

   L층에서 시작해 왼쪽으로 진행하며 가중치 갱신

   역방향으로 진행하면서 미분값인 그래디언트를 계산하는 역전파 사용

2. 데이터 단위와 관련

데이터셋에 n개의 샘플이 있다면 n번의 미분과 가중치 갱신 발생

→ 모든 샘플을 처리하는 한 사이클을 세대*epoch*

한쪽극단은 샘플마다 역전파 방식 적용,, 다른 극단은 모든 샘플의 미분값의 평균을 구하고 역전파 방식 적용해 한세대 마치는 배치방식 적용

배치방식

미니배치 M을 적절한 크기로 설정하고 미니배치 단위로 역전파 방식 적용

위의 방식을 n/M 번 반복해 한 세대 처리

미니 배치 방식에서 평균제곱오차

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

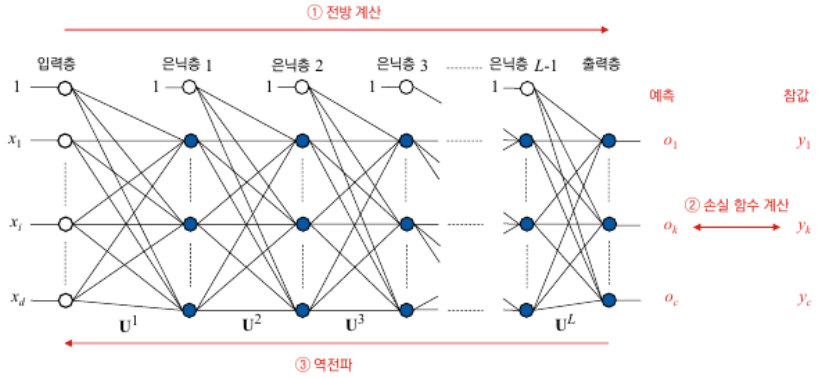


그림 7-21 신경망 학습 알고리즘

7.7 다층 퍼셉트론 구현하기

```
#7.2 다층 퍼셉트론으로 MNIST 인식하기(SGD 옵티마이저)
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

# models, layers, optimizers 모듈에서 필요한 클래스 소환
from tensorflow.keras.models import Sequential # 왼쪽에서 오른쪽으로 계산이 한 줄기로 흐르는 경우 사용
from tensorflow.keras.layers import Dense # 완전연결층
from tensorflow.keras.optimizers import SGD # 옵티마이저 함에서 SGD

# 데이터 준비 - 수집단계
(x_train,y_train),(x_test,y_test)=ds.mnist.load_data() # 데이터 로드해 훈련, 테스트
x_train=x_train.reshape(60000,784) # 1차원 맵으로 변경
x_test=x_test.reshape(10000,784) # 1차원 맵으로 변경
x_train=x_train.astype(np.float32)/255.0 # 실수 연산이 가능하도록 데이터 타입 변환
x_test=x_test.astype(np.float32)/255.0 # 실수 연산이 가능하도록 데이터 타입 변환
y_train=tf.keras.utils.to_categorical(y_train,10) # 데이터 출력 결과 onehot코드로 변환
y_test=tf.keras.utils.to_categorical(y_test,10) # 데이터 출력 결과 onehot코드로 변환

# 모델선택(신경망 구조 설계) : 다층퍼셉트론 구축
mlp=Sequential() # mlp 객체 생성
## add로 층을 쌓고 dense는 완전연결층
mlp.add(Dense(units=512,activation='tanh',input_shape=(784,))) # 입력층, 은닉층
mlp.add(Dense(units=10,activation='softmax')) # 출력층, 텐서플로가 이전 층의 노드개수를 알아 input_shape 생략략

# 학습
mlp.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
```

```
mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2) # 실제 학습 실행

# 예측 - 성능 측정
res=mlp.evaluate(x_test,y_test,verbose=0) # 성능 측정 - test 집합으로 실행
print('정확률=',res[1]*100) # 정확률*100으로 퍼센트로 출력
```

`mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))`

- units = 512 : 은닉층에 512개의 노드 배치
- activation = 'tanh'는 은닉층 활성함수

`mlp.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])`

- 손실함수로 MSE사용
- 옵티마이저로 SGD(스토캐스틱 경사 하강법) 사용하며, 기본값 0.01
- 학습 도중 정확률을 기준으로 성능 측정

`mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)`

- `validation_data=(x_test,y_test)` 학습 도중에 x_test, y_test를 가지고 성능 측정
- `verbose=2` 학습 도중에 세대마다 성능 출력

▼ 결과

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/50
469/469 - 16s - loss: 0.0900 - accuracy: 0.1820 - val_loss: 0.0870 - val_accuracy: 0.2746 - 16s/epoch - 34ms/step
Epoch 2/50
469/469 - 7s - loss: 0.0840 - accuracy: 0.3421 - val_loss: 0.0803 - val_accuracy: 0.4110 - 7s/epoch - 15ms/step
Epoch 3/50
469/469 - 5s - loss: 0.0770 - accuracy: 0.4463 - val_loss: 0.0729 - val_accuracy: 0.4828 - 5s/epoch - 11ms/step
Epoch 4/50
469/469 - 4s - loss: 0.0700 - accuracy: 0.5245 - val_loss: 0.0661 - val_accuracy: 0.5710 - 4s/epoch - 9ms/step
Epoch 5/50
469/469 - 4s - loss: 0.0635 - accuracy: 0.6041 - val_loss: 0.0598 - val_accuracy: 0.6499 - 4s/epoch - 9ms/step
Epoch 6/50
469/469 - 5s - loss: 0.0575 - accuracy: 0.6754 - val_loss: 0.0539 - val_accuracy: 0.7112 - 5s/epoch - 11ms/step
Epoch 7/50
469/469 - 4s - loss: 0.0521 - accuracy: 0.7200 - val_loss: 0.0489 - val_accuracy: 0.7391 - 4s/epoch - 9ms/step
Epoch 8/50
469/469 - 5s - loss: 0.0477 - accuracy: 0.7411 - val_loss: 0.0449 - val_accuracy: 0.7569 - 5s/epoch - 10ms/step
Epoch 9/50
469/469 - 6s - loss: 0.0442 - accuracy: 0.7556 - val_loss: 0.0417 - val_accuracy: 0.7733 - 6s/epoch - 12ms/step
Epoch 10/50
469/469 - 4s - loss: 0.0413 - accuracy: 0.7700 - val_loss: 0.0390 - val_accuracy: 0.7892 - 4s/epoch - 9ms/step
Epoch 11/50
469/469 - 4s - loss: 0.0388 - accuracy: 0.7858 - val_loss: 0.0367 - val_accuracy: 0.8063 - 4s/epoch - 9ms/step
Epoch 12/50
469/469 - 6s - loss: 0.0367 - accuracy: 0.8013 - val_loss: 0.0346 - val_accuracy: 0.8206 - 6s/epoch - 12ms/step
Epoch 13/50
469/469 - 4s - loss: 0.0348 - accuracy: 0.8138 - val_loss: 0.0329 - val_accuracy: 0.8326 - 4s/epoch - 9ms/step
Epoch 14/50
469/469 - 5s - loss: 0.0332 - accuracy: 0.8246 - val_loss: 0.0313 - val_accuracy: 0.8403 - 5s/epoch - 10ms/step
Epoch 15/50
```

```
469/469 - 5s - loss: 0.0190 - accuracy: 0.8884 - val_loss: 0.0179 - val_accuracy: 0.8945 - 5s/epoch - 12ms/step
Epoch 48/50
469/469 - 5s - loss: 0.0188 - accuracy: 0.8891 - val_loss: 0.0178 - val_accuracy: 0.8948 - 5s/epoch - 11ms/step
Epoch 49/50
469/469 - 6s - loss: 0.0187 - accuracy: 0.8895 - val_loss: 0.0177 - val_accuracy: 0.8952 - 6s/epoch - 13ms/step
Epoch 50/50
469/469 - 4s - loss: 0.0186 - accuracy: 0.8900 - val_loss: 0.0176 - val_accuracy: 0.8953 - 4s/epoch - 8ms/step
정확률= 89.52999711036682
```

매 에포크마다 test, train에 대한 loss, accuracy 출력

결과로는 100개의 샘플중 89개 가량 맞춤

```
# 다층퍼셉트론으로 MNIST 인식하기 (ADAM 옵티마이저)

import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam # 앞서 SGD에서 ADAM으로 변경

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

mlp=Sequential()
mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp.add(Dense(units=10,activation='softmax'))

mlp.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics=['accuracy']) # 학습률learning_rate값은 디폴트
mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)

res=mlp.evaluate(x_test,y_test,verbose=0)
print('정확률=',res[1]*100)
```

▼ 결과

```
Epoch 48/50
469/469 - 5s - loss: 2.6190e-04 - accuracy: 0.9985 - val_loss: 0.0031 - val_accurac
Epoch 49/50
469/469 - 6s - loss: 2.8277e-04 - accuracy: 0.9984 - val_loss: 0.0030 - val_accurac
Epoch 50/50
469/469 - 6s - loss: 2.4428e-04 - accuracy: 0.9987 - val_loss: 0.0029 - val_accurac
정확률= 98.18999767303467
```

SGD : 89.46 / ADAM 98.19 로 정확률 향상

→ 옵티마이저는 학습과 관련된 중요한 하이퍼 매개변수

```
#7.4 다층 퍼셉트론으로 MNIST 인식하기 - SGD와 ADAM의 성능 그래프 비교
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import SGD, Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

mlp_sgd=Sequential()
mlp_sgd.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp_sgd.add(Dense(units=10,activation='softmax'))

# SGD
mlp_sgd.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
## 히스토그램으로 파악
hist_sgd=mlp_sgd.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
```

```
print('SGD 정확률=',mlp_sgd.evaluate(x_test,y_test,verbose=0)[1]*100)

mlp_adam=Sequential()
mlp_adam.add(Dense(units=512,activation='tanh',input_shape=(784,)))
mlp_adam.add(Dense(units=10,activation='softmax'))

# Adam
mlp_adam.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics=['accuracy'])
## 히스토그램으로 파악
hist_adam=mlp_adam.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
print('Adam 정확률=',mlp_adam.evaluate(x_test,y_test,verbose=0)[1]*100)

import matplotlib.pyplot as plt

# 하나의 그래프 그려 성능 비교
plt.plot(hist_sgd.history['accuracy'],'r--')
plt.plot(hist_sgd.history['val_accuracy'],'r')
plt.plot(hist_adam.history['accuracy'],'b--')
plt.plot(hist_adam.history['val_accuracy'],'b')
plt.title('Comparison of SGD and Adam optimizers')
plt.ylim((0.7,1.0))
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train_sgd','val_sgd','train_adam','val_adam'])
plt.grid()
plt.show()
```

▼ 결과

학습 곡선 : Adam이 SGD 능가

실행할 때마다 다른 결과가 나오는데 신경망의 학습은 가중치를 초기화할 때 난수를 사용했기 때문

하이퍼 매개변수

하이퍼 매개변수를 잘 설정해야 신경망이 높은 성능 발휘

→ 하이퍼 매개변수의 최적값을 찾는 일을 **하이퍼 매개변수 최적화**

좋은 하이퍼 매개변숫값을 찾는 일은 시간이 많이 걸림

하이퍼 매개변수 설정에 대한 요령

1. 텐서플로가 제공하는 기본값 사용

   함수 호출에서 해당 매개변수 생략
2. 신뢰할 수 있는 논문이나 문서 또는 웹 사이트에서 제공하는 권고사항 따르기
3. 중요한 하이퍼 매개변수 1~3개에 대해 성능 실험을 통해 최적값 스스로 설정

```
# 깊은 다층 퍼셉트론으로 MNIST 인식
import numpy as np
import tensorflow as tf
import tensorflow.keras.datasets as ds

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam

(x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
x_train=x_train.reshape(60000,784)
x_test=x_test.reshape(10000,784)
x_train=x_train.astype(np.float32)/255.0
x_test=x_test.astype(np.float32)/255.0
y_train=tf.keras.utils.to_categorical(y_train,10)
y_test=tf.keras.utils.to_categorical(y_test,10)

# 모델선택(신경망 구조 설계) : 다층퍼셉트론 구축
```

```
dmlp=Sequential() # dmlp 객체 생성
## 4개의 층 쌓기
dmlp.add(Dense(units=1024,activation='relu',input_shape=(784,))) # 은닉층1
dmlp.add(Dense(units=512,activation='relu')) # 은닉층2
dmlp.add(Dense(units=512,activation='relu')) # 은닉층33
dmlp.add(Dense(units=10,activation='softmax'))

## categorical_crossentropy : 손실 함수로 교차 엔트로피 사용
### 교차엔트로피 : 분류문제에서 평균제곱오차보다 더 좋은 성능
## learning rate : 학습률 0.0001로 천천히 학습
dmlp.compile(loss='categorical_crossentropy',optimizer=Adam(learning_rate=0.0001),metrics=['accuracy'])
hist=dmlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_data=(x_test,y_test),verbose=2)
print('정확률=', dmlp.evaluate(x_test,y_test,verbose=0)[1]*100)

# 학습을 마친 신경망의 구조정보와 가중치 값을 지정한 파일에 저장
# -> 필요할때 load_model로 불러다 사용
dmlp.save('dmlp_trained.h5')

import matplotlib.pyplot as plt

# 시각화 - 정확률
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('Accuracy graph')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend(['train','test'])
plt.grid()
plt.show()

# 시각화 - loss 함수 그래프
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('Loss graph')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend(['train','test'])
plt.grid()
plt.show()
```

▼ 결과

과잉 적합

과잉적합 : 학습 알고리즘이 훈련 집합을 과하게 맞춰 일반화 능력을 상실하는 현상 발생

ex. CIFAR-10에서 훈련 집합에 대해 90% 정도 달성,

test 집합에 대해 55% 달성해 차이가 심해 과잉적합 발생

→ 모델의 용량은 충분히 크게 설계하되, 이런 문제를 방지하기 위해 다양한 규제 기법 적용

⇒ 드롭아웃, 데이터 증강, 배치 정규화 등 적용

7.8 [비전에이전트] 우편번호 인식기 v.1

```
# 7-7 우편번호 인식기 c.1 구현
import numpy as np
import tensorflow as tf
import cv2 as cv
import matplotlib.pyplot as plt
import winsound

model=tf.keras.models.load_model('dmlp_trained.h5') # 앞서 저장되어 있어서 실행되는 것

# e : erase 명령어 처리하는 함수
def reset():
    global img

    img=np.ones((200,520,3),dtype=np.uint8)*255 # img 영상생성, 3채널의 컬러영상
    for i in range(5): # 빨간색 박스
        cv.rectangle(img,(10+i*100,50),(10+(i+1)*100,150),(0,0,255))
    cv.putText(img,'e:erase s:show r:recognition q:quit',(10,40),cv.FONT_HERSHEY_SIMPLEX,0.8,(255,0,0),1) # 명령어를 나타내는 글씨 쓰기

# 다섯개 숫자 떼어내기
```

```
def grab_numerals():
    numerals=[]
    for i in range(5):
        roi=img[51:149,11+i*100:9+(i+1)*100,0]
        roi=255-cv.resize(roi,(28,28),interpolation=cv.INTER_CUBIC) # 이미지에서 숫제 분리해 리스트 추가가
        numerals.append(roi)
    numerals=np.array(numerals)
    return numerals

# s : show 명령어 처리하는 함수
def show():
    numerals=grab_numerals()
    plt.figure(figsize=(25,5))
    for i in range(5):
        plt.subplot(1,5,i+1)
        plt.imshow(numerals[i],cmap='gray')
        plt.xticks([]); plt.yticks([])
    plt.show()

# r : recognition 명령어 처리하는 함수
def recognition():
    numerals=grab_numerals()
    numerals=numerals.reshape(5,784)
    numerals=numerals.astype(np.float32)/255.0
    res=model.predict(numerals) # 신경망 모델로 예측
    class_id=np.argmax(res,axis=1)
    for i in range(5):
        cv.putText(img,str(class_id[i]),(50+i*100,180),cv.FONT_HERSHEY_SIMPLEX,1,(255,0,0),1)
    winsound.Beep(1000,500)

BrushSiz=4
LColor=(0,0,0)

# 마우스 콜백함수
def writing(event,x,y,flags,param):
    if event==cv.EVENT_LBUTTONDOWN:
        cv.circle(img,(x,y),BrushSiz,LColor,-1)
    elif event==cv.EVENT_MOUSEMOVE and flags==cv.EVENT_FLAG_LBUTTON:
        cv.circle(img,(x,y),BrushSiz,LColor,-1)

# Main
reset()
cv.namedWindow('Writing') # 윈도우 생성
cv.setMouseCallback('Writing',writing) # 윈도우 콜백함수, writing 함수 등록

# 무한 방복하며 사용자와 인터페이스
while(True):
    cv.imshow('Writing',img) # 변경된 내용 윈도우에 반영
    key=cv.waitKey(1)
    # 키에 따라 함수 호출
    if key==ord('e'):
        reset()
    elif key==ord('s'):
        show()
    elif key==ord('r'):
        recognition()
    elif key==ord('q'): # 루프 빠져나오기 - 윈도우 닫음
        break

cv.destroyAllWindows()
```