Contents

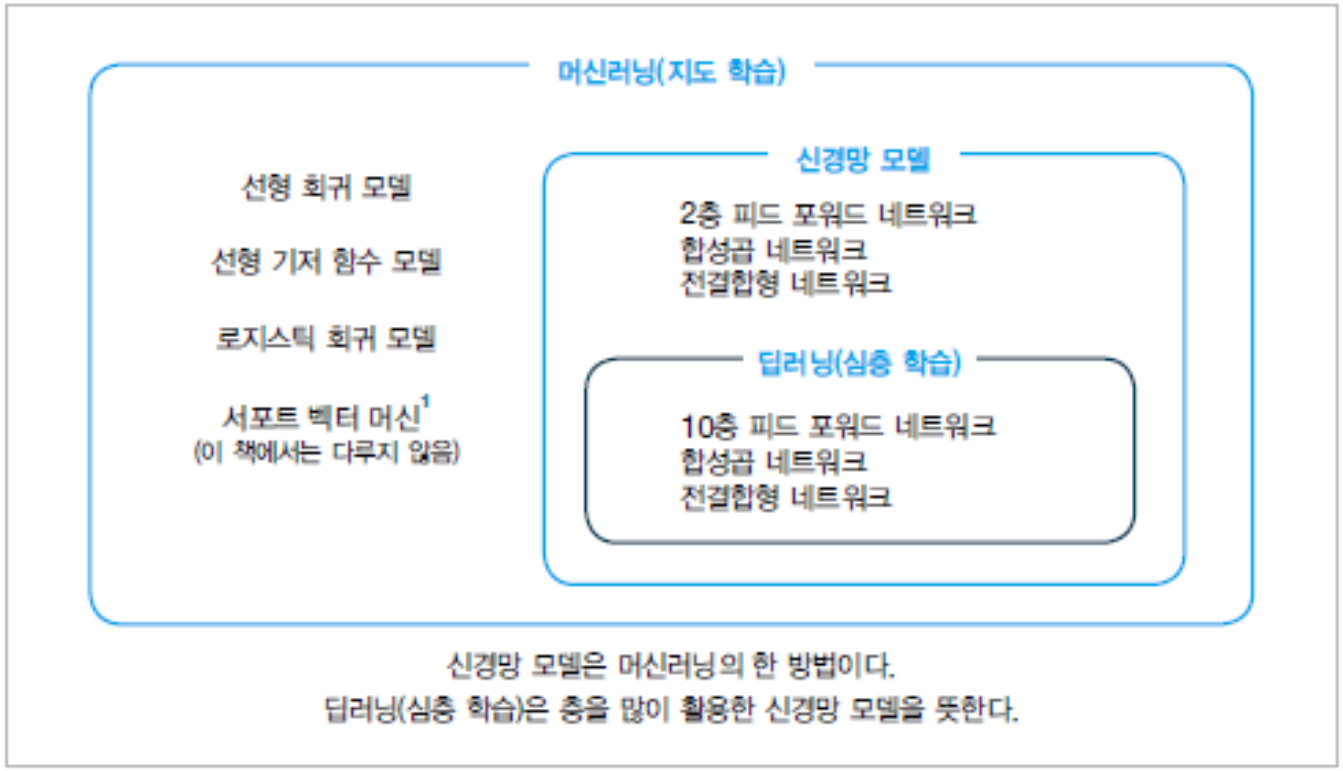
Contents

CHAPTER 07 신경망·딥러닝

분류 문제를 풀기 위한 신경망(딥러닝)을 설명.

SECTION.01 뉴런 모델

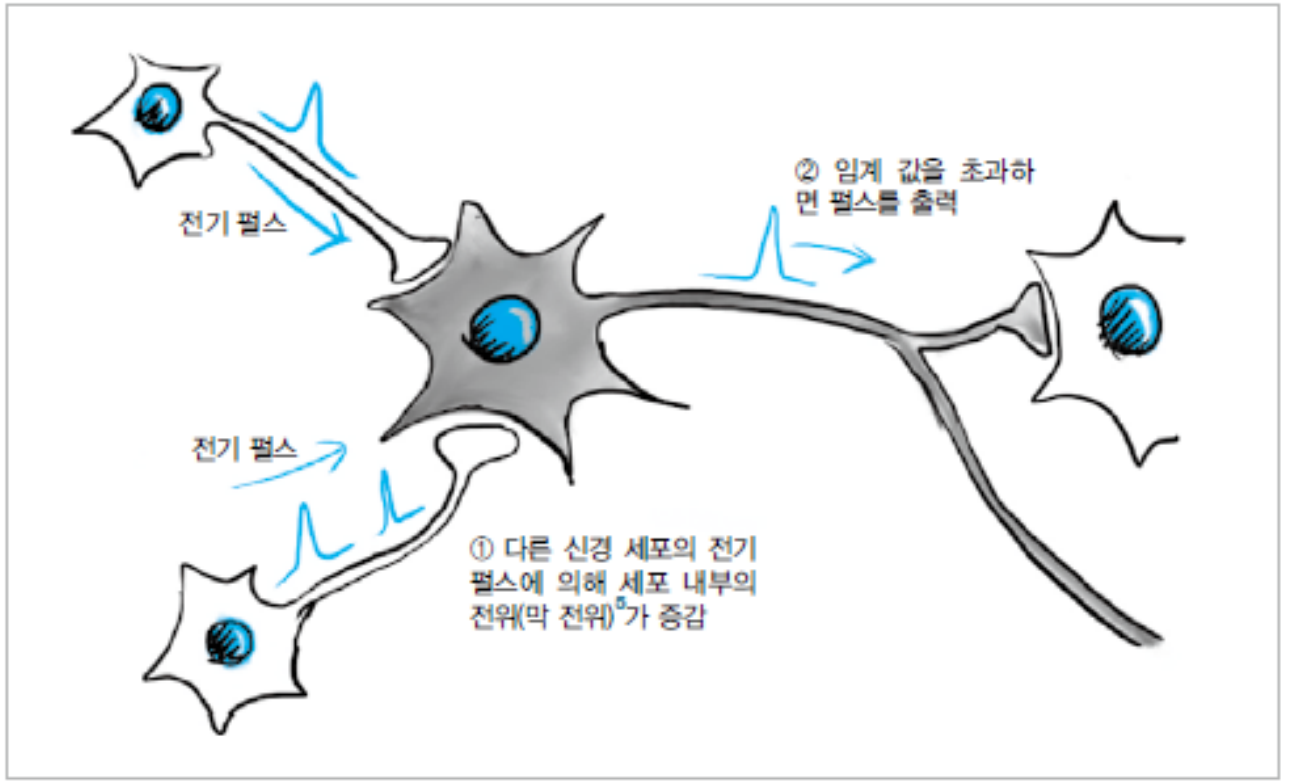
- 머신러닝(지도 학습)의 분류
- 신경망 모델은 '뉴런 모델' 단위로 구축됨.
- 실제 뇌의 것을 '신경 세포'로, 수학적 모델을 '뉴런'이라고 규정.



SECTION.01 뉴런 모델

1.1 신경 세포

- 신경 세포의 신호 전달 메커니즘.



SECTION.01 1차원 입력 2클래스 분류

1.2 뉴런 모델

- 신경 세포의 움직임을 단순화한 수학적 모델인 뉴런 모델.

그림 7-3 뉴런모델

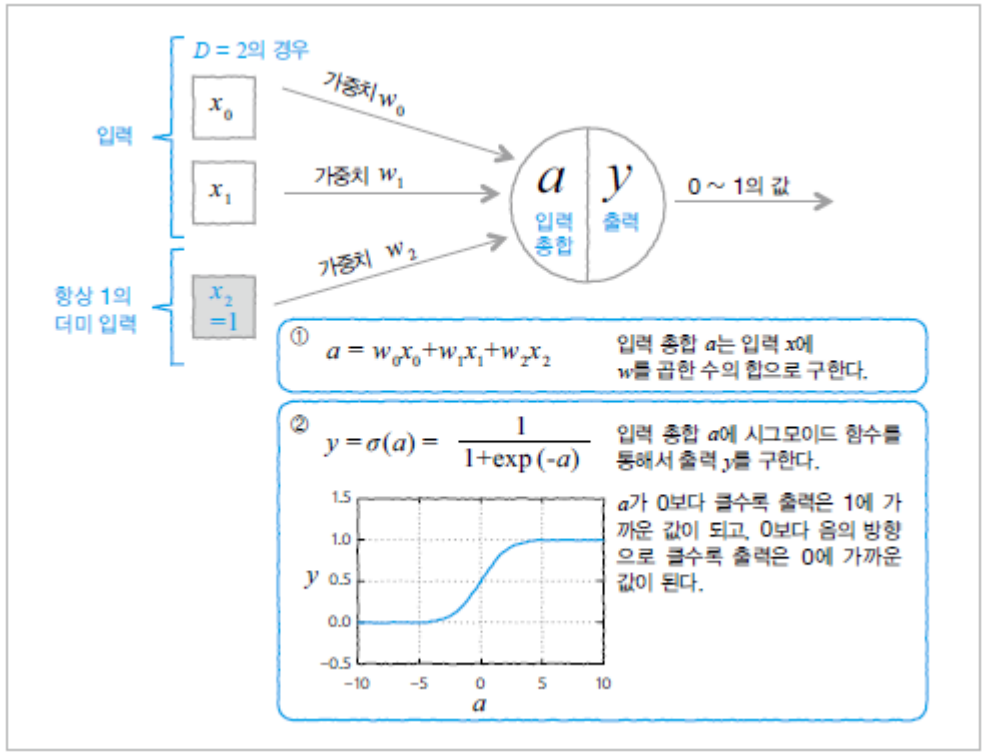
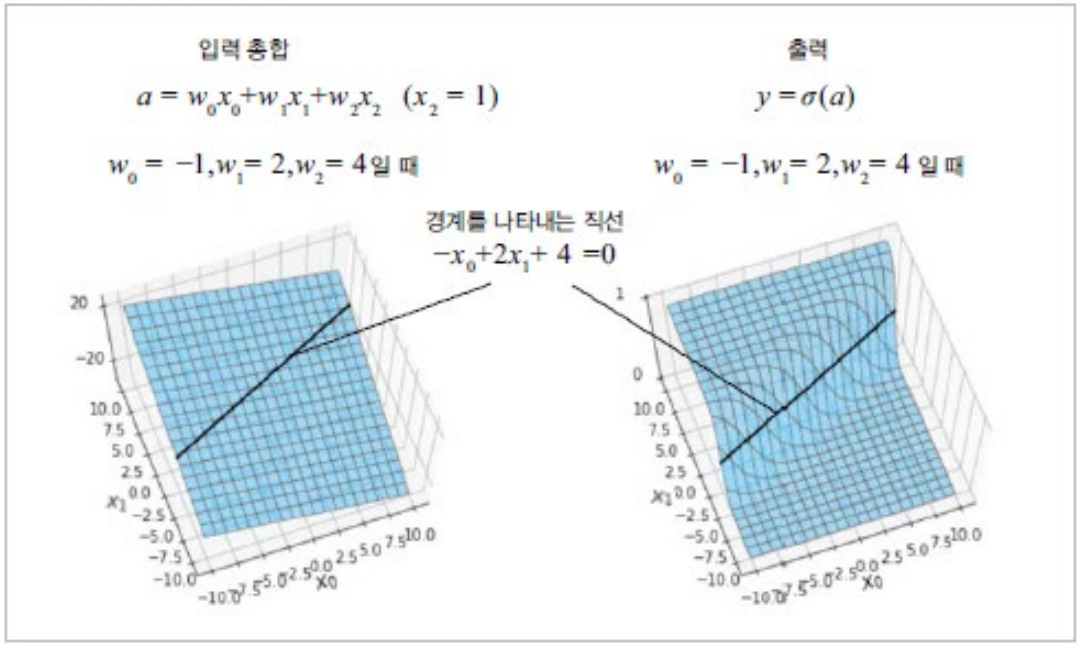


그림 7-4 뉴런 모델의 입출력 관계



SECTION.02 신경망 모델

2.1 2층 피드 포워드 신경망

- 뉴런의 집합체 모델을 신경망 모델 (또는 단순히 신경망)이라고 함.
- 한 방향으로만 흐르는 '피드 포워드 신경망'

그림 7-5 2층 피드 포워드 신경망 모델

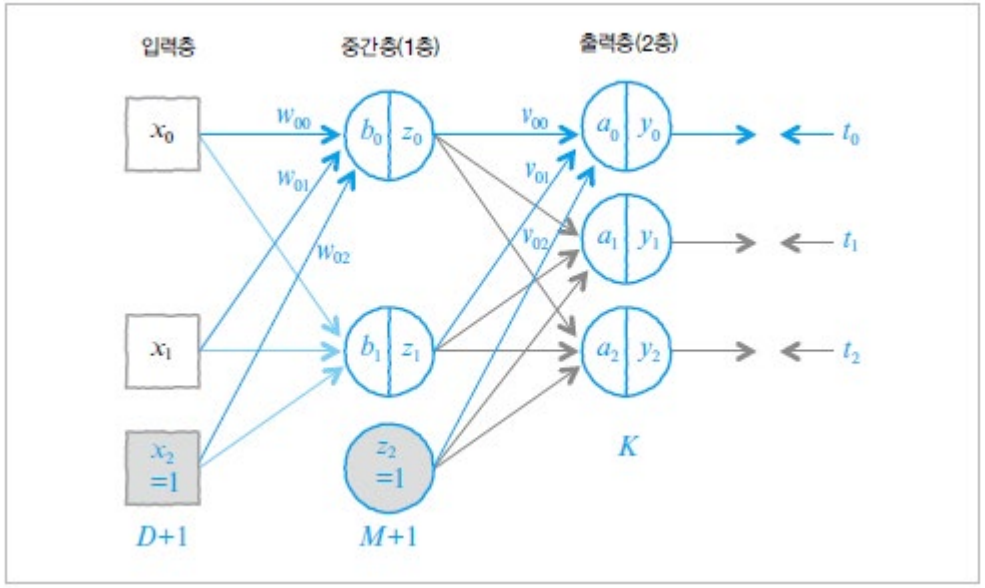
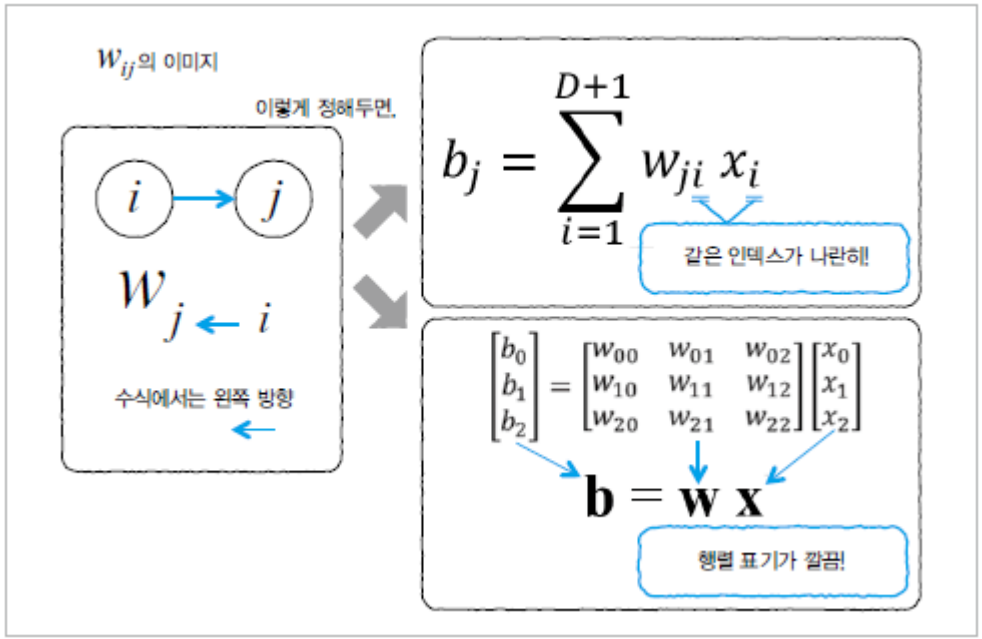


그림 7-6 결합의 인덱스



SECTION.02 신경망 모델

- 일반적으로 입력 차원을 D, 중간층 뉴런의 수를 M, 출력 차원을 K로 한 경우, 네트워크는 다음과 같이 정의됨.

- 중간층의 입력 총합: $b_j = \sum_{i=0}^{D} w_{ji} x_i$
- 중간층의 출력: $z_j = h(b_j)$
- 출력층의 입력 총합: $a_k = \sum_{j=0}^{M} v_{kj} z_j$
- 출력층의 출력: $y_k = \frac{\exp(a_k)}{\sum_{l=0}^{K-1} \exp(a_l)} = \frac{\exp(a_k)}{u}$

2.2 2층 피드 포워드 신경망의 구현

- 파이썬으로 구현하여 동작을 확인.

In

```
#-- 리스트 7-1-(1)
import numpy as np
# 데이터 생성 ------------------------------
np.random.seed(seed=1) # 난수를 고정
N = 200 # 데이터의 수
K = 3 # 분포의 수
T = np.zeros((N, 3), dtype=np.uint8)
X = np.zeros((N, 2))
X_range0 = [-3, 3] # X0의 범위, 표시용
X_range1 = [-3, 3] # X1의 범위, 표시용
Mu = np.array([[-.5, -.5], [.5, 1.0], [1, -.5]]) # 분포의 중심
Sig = np.array([[.7, .7], [.8, .3], [.3, .8]]) # 분포의 분산
Pi = np.array([0.4, 0.8, 1]) # 각 분포에 대한 비율
for n in range(N):
    wk = np.random.rand()
    for k in range(K):
        if wk < Pi[k]:
            T[n, k] = 1
            break
    for k in range(2):
        X[n, k] = np.random.randn() * Sig[T[n, :] == 1, k] + \
        Mu[T[n, :] == 1, k]
```

In

```
#-- 리스트 7-1-(2)
# -------- 2 분류 데이터를 테스트 훈련 데이터로 분할
TestRatio = 0.5
X_n_training = int(N * TestRatio)
X_train = X[:X_n_training, :]
X_test = X[X_n_training:, :]
T_train = T[:X_n_training, :]
T_test = T[X_n_training:, :]


# -------- 데이터를 'class_data.npz'에 저장
np.savez('class_data.npz', X_train=X_train, T_train=T_train,
         X_test=X_test, T_test=T_test,
         X_range0=X_range0, X_range1=X_range1)
```

SECTION.02 신경망 모델

• 3클래스 분류 문제의 인공 데이터

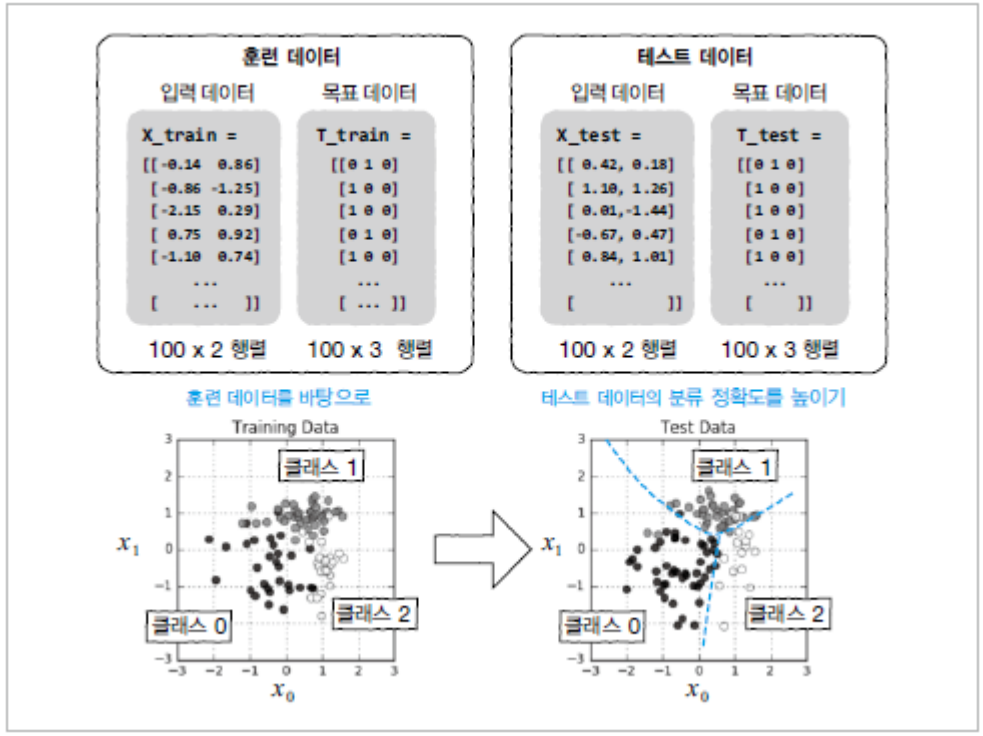
In

```
#-- 리스트 7-1-(3)
import matplotlib.pyplot as plt
%matplotlib inline


# 데이터를 그리기 ------------------------------
def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                 linestyle='none',
                 marker='o', markeredgecolor='black',
                 color=c[i], alpha=0.8)
    plt.grid(True)


# 메인 ----------------------------------------
plt.figure(1, figsize=(8, 3.7))
plt.subplot(1, 2, 1)
Show_data(X_train, T_train)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Training Data')
plt.subplot(1, 2, 2)
Show_data(X_test, T_test)
plt.xlim(X_range0)
plt.ylim(X_range1)
plt.title('Test Data')
plt.show()
```

Out

```
# 실행 결과는 [그림 7-7]의 아래를 참조
```

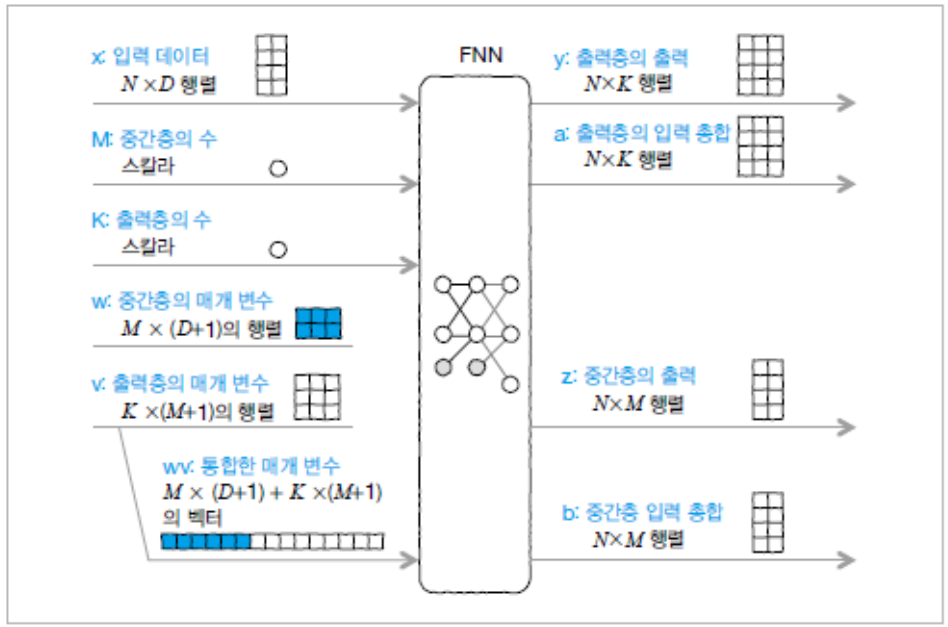
그림 7-7 3클래스 분류 문제의 인공 데이터



SECTION.02 신경망 모델

- 피드 포워드 네트워크 모델의 함수 FNN의 인수와 출력
- 네트워크 프로그램의 코드.

그림 7-8 피드 포워드 네트워크 모델의 함수 FNN의 인수와 출력



In

```
# 리스트 7-1-(4)
# 시그모이드 함수 ------------------------
def Sigmoid(x):
    y = 1 / (1 + np.exp(-x))
    return y


# 네트워크 ------------------------
def FNN(wv, M, K, x):
    N, D = x.shape # 입력 차원
    w = wv[:M * (D + 1)] # 중간층 뉴런의 가중치
    w = w.reshape(M, (D + 1))
    v = wv[M * (D + 1):] # 출력층 뉴런의 가중치
    v = v.reshape((K, M + 1))
    b = np.zeros((N, M + 1)) # 중간층 뉴런의 입력 총합
    z = np.zeros((N, M + 1)) # 중간층 뉴런의 출력
    a = np.zeros((N, K)) # 출력층 뉴런의 입력 총합
    y = np.zeros((N, K)) # 출력층 뉴런의 출력
    for n in range(N):
        # 중간층의 계산
        for m in range(M):
            b[n, m] = np.dot(w[m, :], np.r_[x[n, :], 1]) # (A)
            z[n, m] = Sigmoid(b[n, m])
        # 출력층의 계산
        z[n, M] = 1 # 더미 뉴런
        wkz = 0
        for k in range(K):
            a[n, k] = np.dot(v[k, :], z[n, :])
            wkz = wkz + np.exp(a[n, k])
        for k in range(K):
            y[n, k] = np.exp(a[n, k]) / wkz
    return y, a, z, b


# test ---
WV = np.ones(15)
M = 2
K = 3
FNN(WV, M, K, X_train[:2, :])
```

Out

```
(array([[0.33333333,    0.33333333,  0.33333333],
        [0.33333333,    0.33333333,  0.33333333]]),
 array([[2.6971835 ,    2.6971835 ,  2.6971835 ],
        [1.49172649,    1.49172649,  1.49172649]]),
 array([[0.84859175,    0.84859175,  1        ],
        [0.24586324,    0.24586324,  1        ]]),
 array([[ 1.72359839,   1.72359839,  0.       ],
        [-1.12079826,  -1.12079826,  0.       ]]))
```

2.3 수치 미분법

- 평균 교차 엔트로피 오차를 다음의 CE_FNN 함수로 구현함.

$$E(\mathbf{w}, \mathbf{v}) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} t_{nk} \log(y_{nk})$$

[평균 교차 엔트로피 사용]

In

```
# 리스트 7-1-(5)
# 평균 교차 엔트로피 오차 ---------
def CE_FNN(wv, M, K, x, t):
    N, D = x.shape
    y, a, z, b = FNN(wv, M, K, x)
    ce = -np.dot(np.log(y.reshape(-1)), t.reshape(-1)) / N
    return ce


# test ---
WV = np.ones(15)
M = 2
K = 3
CE_FNN(WV, M, K, X_train[:2, :], T_train[:2, :])
```

Out

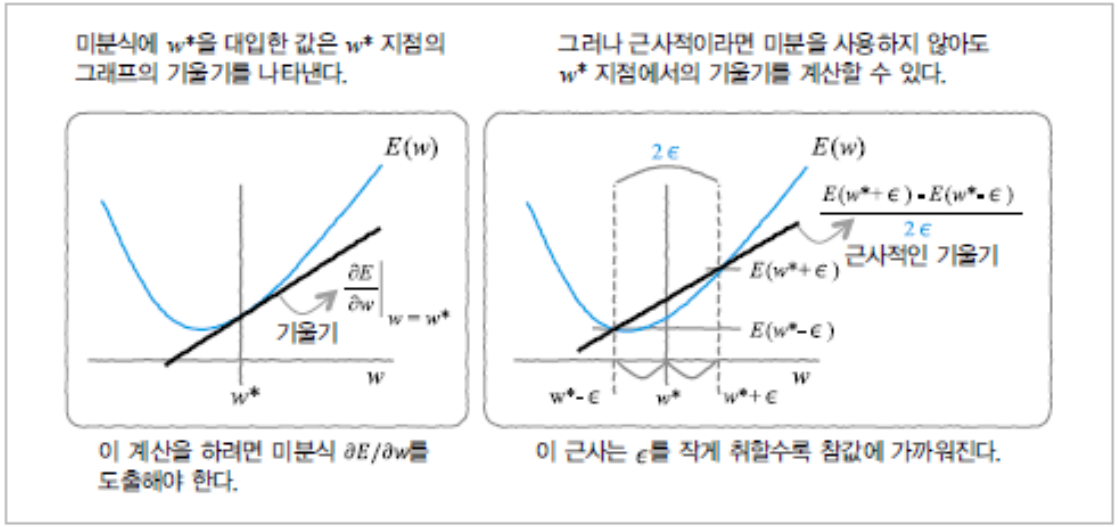
```
1.0986122886681098
```

SECTION.02 신경망 모델

- 경사 하강법을 적용하려면 오차 함수를 매개 변수로 편미분한 식이 필요.
- 이 미분계산을 성실하게 하지 않고도, 간단히 수치적 미분과 마찬가지의 값을 구할 수 있음.

$$\left.\frac{\partial E}{\partial w}\right|_{w^*} \cong \frac{E(w^*+\epsilon)-E(w^*-\epsilon)}{2\epsilon} \qquad \text{(식 7–19)}$$

그림 7-9 수치미분



SECTION.02 신경망 모델

- 비교적 짧은 코드로 2층 네트워크에서 각 매개 변수의 편미분 값을 구함.

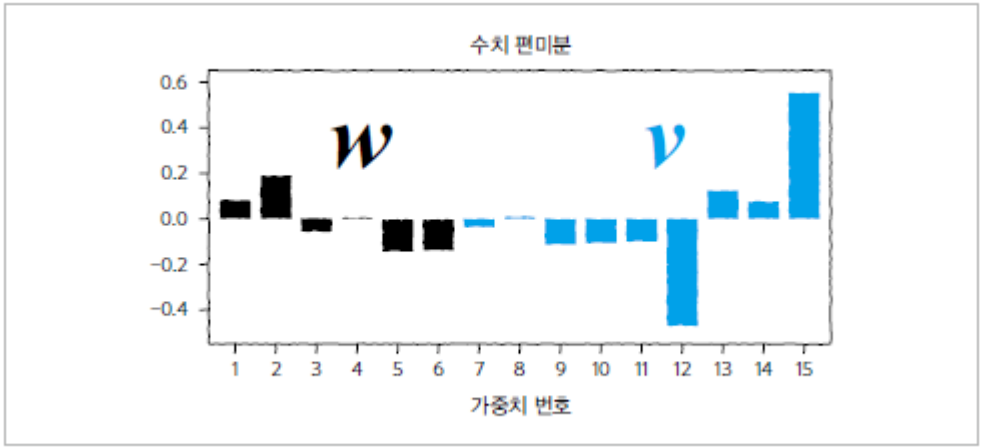
In

```
# 리스트 7-1-(6)
# - 수치 미분 ------------------
def dCE_FNN_num(wv, M, K, x, t):
    epsilon = 0.001
    dwv = np.zeros_like(wv)
    for iwv in range(len(wv)):
        wv_modified = wv.copy()
        wv_modified[iwv] = wv[iwv] - epsilon
        mse1 = CE_FNN(wv_modified, M, K, x, t)
        wv_modified[iwv] = wv[iwv] + epsilon
        mse2 = CE_FNN(wv_modified, M, K, x, t)
        dwv[iwv] = (mse2 - mse1) / (2 * epsilon)
    return dwv


#--dVW의 표시 ------------------
def Show_WV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M * 3 + 1), wv[:M * 3], align="center", color='black')
    plt.bar(range(M * 3 + 1, N + 1), wv[M * 3:],
            align="center", color='cornflowerblue')
    plt.xticks(range(1, N + 1))
    plt.xlim(0, N + 1)


#-test----
M = 2
K = 3
nWV = M * 3 + K * (M + 1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)
dWV = dCE_FNN_num(WV, M, K, X_train[:2, :], T_train[:2, :])
print(dWV)
plt.figure(1, figsize=(5, 3))
Show_WV(dWV, M)
plt.show()
```

Out

```
# 실행 결과는 [그림 7-10]을 참조
```

그림 7-10 수치 편미분



2.4 수치 미분법에 의한 경사 하강법

- 경사 하강법으로 함수명 Fit_FNN_num을 풀어봄.

```
Fit_FNN_num(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha)
```

- 가중치 매개 변수를 찾아봄.

In

```
# 리스트 7-1-(7)
import time


# 수치 미분을 사용한 경사 하강법 -------
def Fit_FNN_num(wv_init, M, K, x_train, t_train, x_test, t_test, n, alpha):
    wvt = wv_init
    err_train = np.zeros(n)
    err_test = np.zeros(n)
    wv_hist = np.zeros((n, len(wv_init)))
    epsilon = 0.001
    for i in range(n): # (A)
        wvt = wvt - alpha * dCE_FNN_num(wvt, M, K, x_train, t_train)
        err_train[i] = CE_FNN(wvt, M, K, x_train, t_train)
        err_test[i] = CE_FNN(wvt, M, K, x_test, t_test)
        wv_hist[i, :] = wvt
    return wvt, wv_hist, err_train, err_test


# 메인 ----------------------------
startTime = time.time()
M = 2
K = 3
np.random.seed(1)
WV_init = np.random.normal(0, 0.01, M * 3 + K * (M + 1))
N_step = 1000 # (B) 학습 단계
alpha = 0.5
WV, WV_hist, Err_train, Err_test = Fit_FNN_num(
    WV_init, M, K, X_train, T_train, X_test, T_test, N_step, alpha)
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

Out

```
Calculation time : 162.675 sec
```

SECTION.02 신경망 모델

- 훈련 데이터의 오차와 테스트 데이터의 오차를 계산

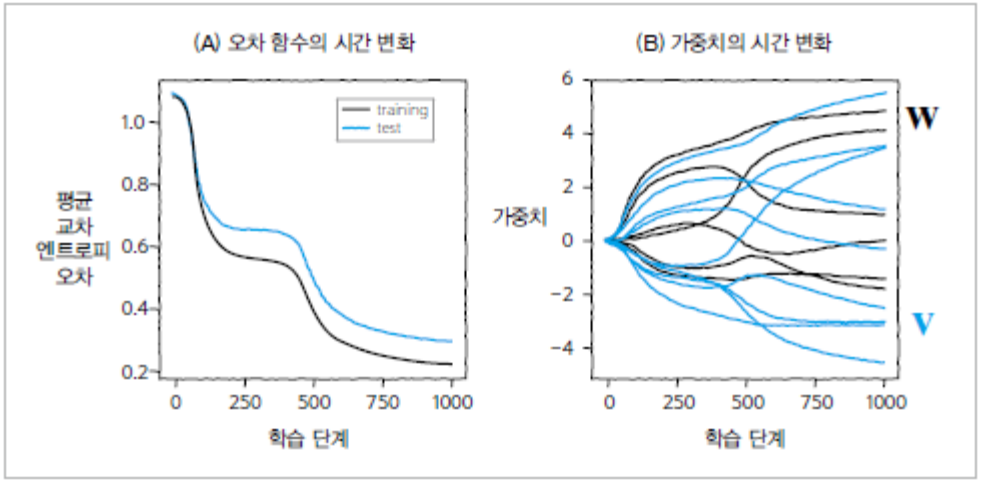
In

```
# 리스트 7-1-(8)
# 학습 오차의 표시 ---------------------------
plt.figure(1, figsize=(3, 3))
plt.plot(Err_train, 'black', label='training')
plt.plot(Err_test, 'cornflowerblue', label ='test')
plt.legend()
plt.show()
```

Out

```
# 실행 결과는 [그림 7-11]의 왼쪽 참조
```

그림 7-11 수치 미분을 사용한 경사 하강법의 실행 결과



SECTION.02 신경망 모델

- 가중치의 시간 변화 플롯.
- 안장점(saddle point)은 다변수 실함수의 변역에서, 어느 방향에서 보면 극대값이지만 다른 방향에서 보면 극솟값이 되는 점임.

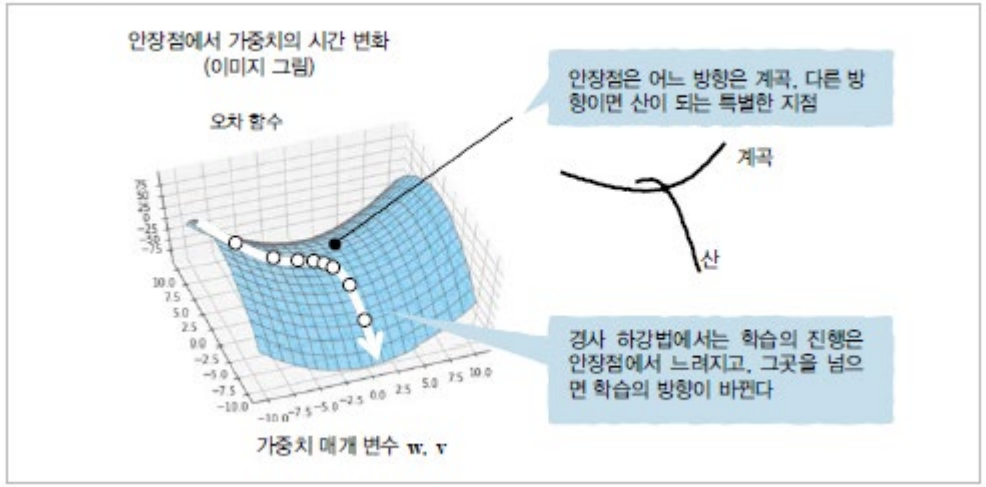
In

```
# 리스트 7-1-(9)
# 가중치의 시간 변화의 표시 --------------------------
plt.figure(1, figsize=(3, 3))
plt.plot(WV_hist[:, :M * 3], 'black')
plt.plot(WV_hist[:, M * 3:], 'cornflowerblue')
plt.show()
```

Out

```
# 실행 결과는 [그림 7-11]을 참조
```

그림 7-12 안장점



- 수치 미분법에 의한 경사 하강법에서 얻은 클래스 간의 경계선

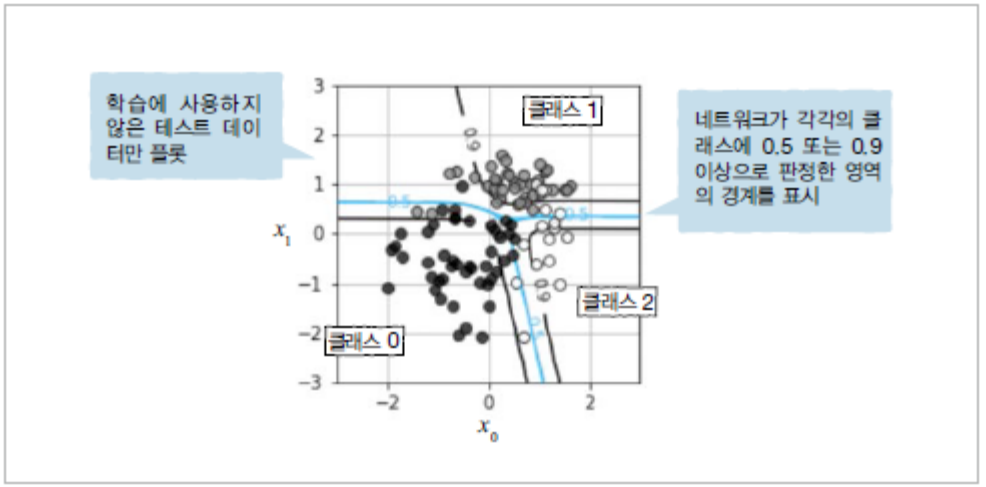
In

```
# 리스트 7-1-(10)
# 경계선 표시 함수 -------------------------
def show_FNN(wv, M, K):
    xn = 60 # 등고선 표시 해상도
    x0 = np.linspace(X_range0[0], X_range0[1], xn)
    x1 = np.linspace(X_range1[0], X_range1[1], xn)
    xx0, xx1 = np.meshgrid(x0, x1)
    x = np.c_[np.reshape(xx0, xn * xn, 1), np.reshape(xx1, xn * xn, 1)]
    y, a, z, b = FNN(wv, M, K, x)
    plt.figure(1, figsize=(4, 4))
    for ic in range(K):
        f = y[:, ic]
        f = f.reshape(xn, xn)
        f = f.T
        cont = plt.contour(xx0, xx1, f, levels=[0.8, 0.9],
                           colors=['cornflowerblue', 'black'])
        cont.clabel(fmt='%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)


# 경계선 표시 -------------------------
plt.figure(1, figsize=(3, 3))
Show_data(X_test, T_test)
show_FNN(WV, M, K)
plt.show()
```

Out

```
# 실행 결과는 [그림 7-13]을 참조
```

그림 7-13 수치 미분법에 의한 경사 하강법에서 얻은 클래스 간의 경계선



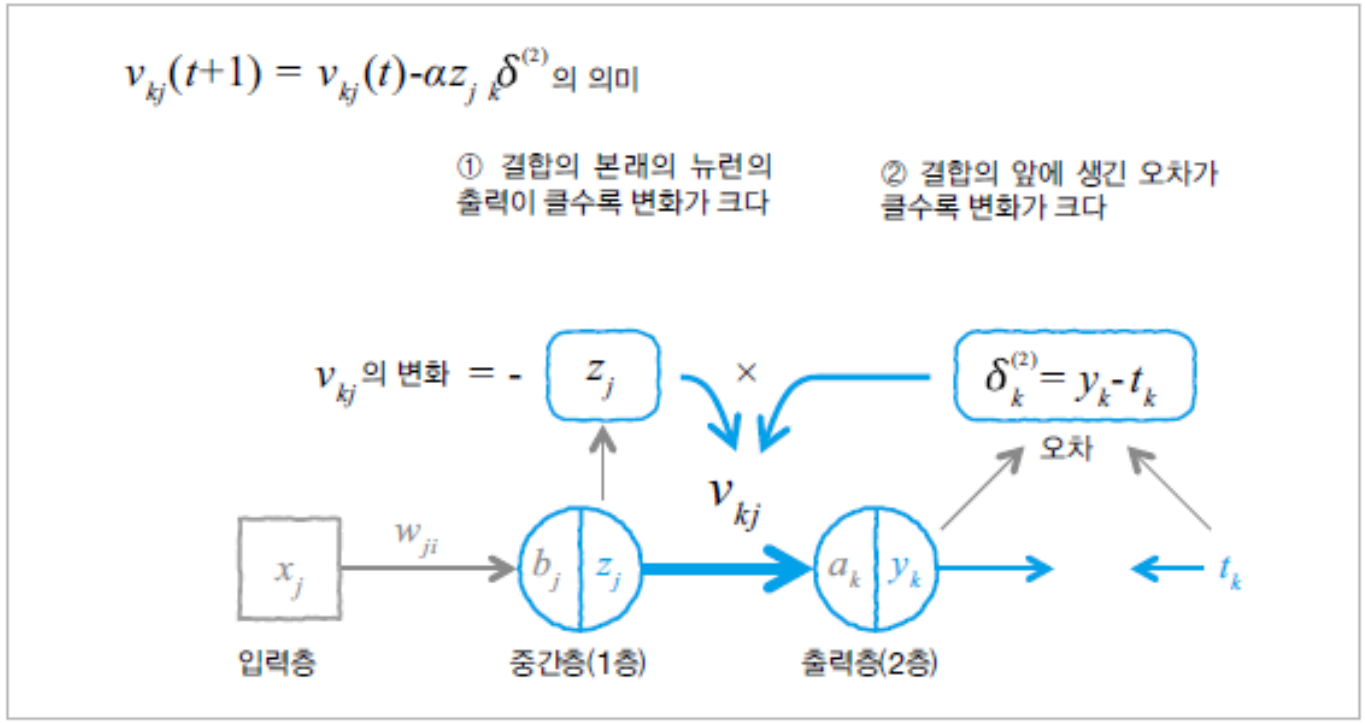
2.5 오차 역전파법

- 피드 포워드 신경망에 학습시키는 방법
- 오차 역전파법은 경사 하강법임.
- 경사 하강법을 피드 포워드 네트워크에 적용하면 오차 역전파법이 자연스럽게 도출됨.

공식	설명
$E(\mathbf{w},\mathbf{v}) = -\frac{1}{N}\sum_{n=0}^{N-1}\sum_{k=0}^{K-1} t_{nk}\ln(y_{nk})$ (식 7-21)	• 경사 하강법을 적용하기 위해 오차 함수를 매개 변수로 편미분함. • 오차 함수는 [식 7-18]의 평균 교차 엔트로피 오차를 생각함.
$E_n(\mathbf{w},\mathbf{v}) = -\sum_{k=0}^{K-1} t_k\ln(y_k)$ (식 7-22)	• 하나의 데이터 n에만 해당하는 상호 엔트로피 오차 을 [식 7-22]과 같이 정의함.
$E(\mathbf{w},\mathbf{v}) = \frac{1}{N}\sum_{n=0}^{N-1} E_n(\mathbf{w},\mathbf{v})$ (식 7-23)	• [식 7-21]을 [식 7-23]과 같이 나타냄
$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}}\frac{1}{N}\sum_{n=0}^{N-1} E_n = \frac{1}{N}\sum_{n=0}^{N-1}\frac{\partial E_n}{\partial w_{ji}}$ (식 7-24)	• 각 데이터 n에 대한 을 구하여 평균을 하면 본래 목적인 $\partial E_n/\partial v_{kj}$ 을 구할 수 있음

2.6 $\partial E_n/\partial v_{kj}$을 구하기

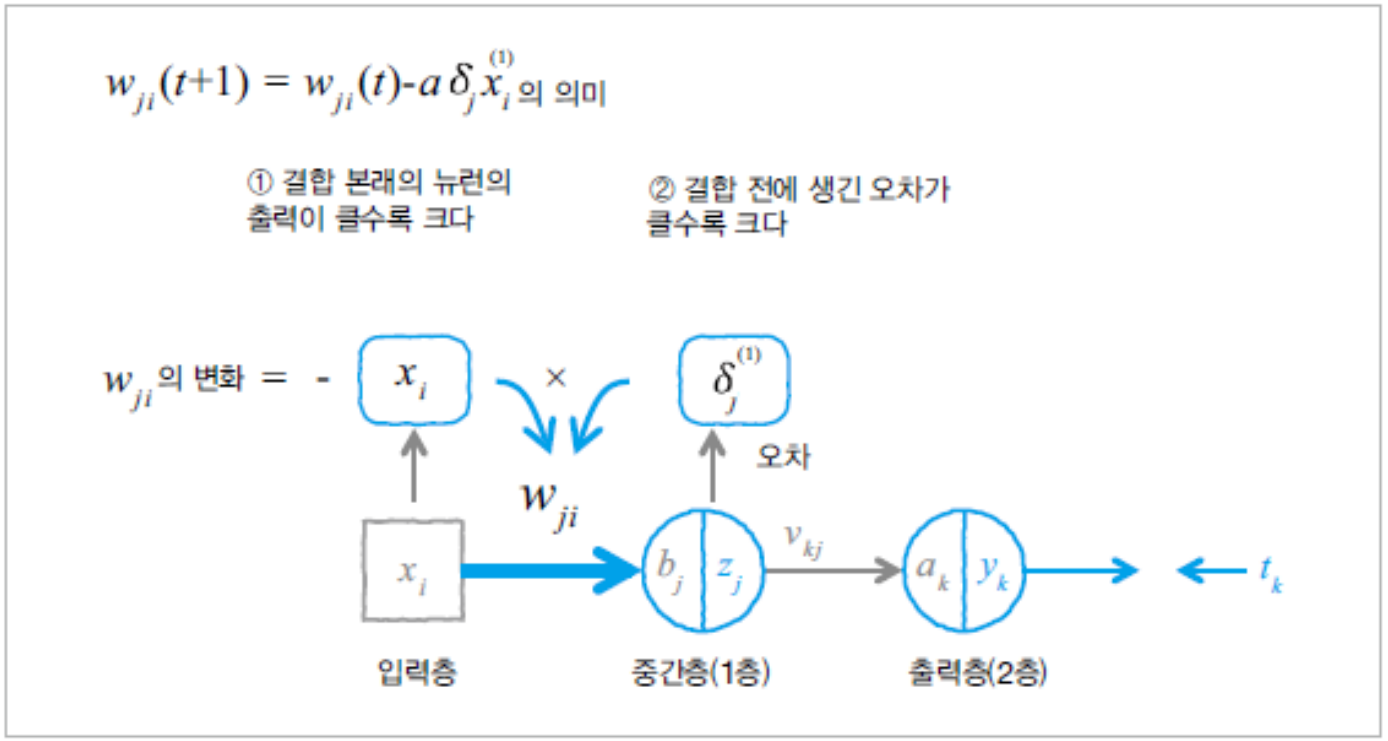
- v의 학습 법칙의 의미



2.7 $\partial E_n/\partial w_{ji}$를 구하기

- 입력층에서 1층의 가중치 매개 변수 의 학습 법칙을 도출함.

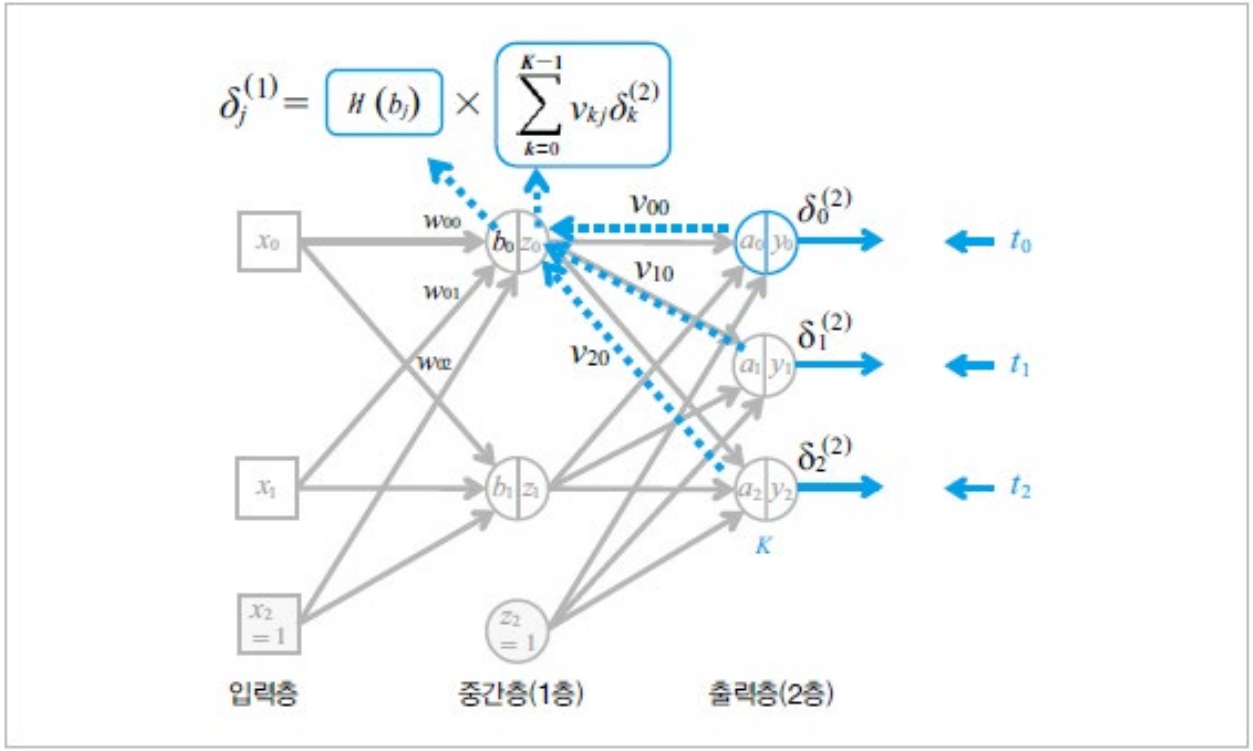
그림 7-15 w학습 법칙의 의미



SECTION.02 신경망 모델

- 오차 역전파법은 네트워크 계층이 더 늘어도 이 법칙을 사용하여 간단하게 가중치 매개 변수의 학습 법칙을 도출할 수 있음.
- 이 특성으로 피드 포워드 신경망의 경사 하강법이 오차 역전파법이라고 불림.

그림 7-16 오차의 역전파



SECTION.02 신경망 모델

- 네트워크 매개 변수의 갱신 방법

그림 7-17 오차 역전파법 ①

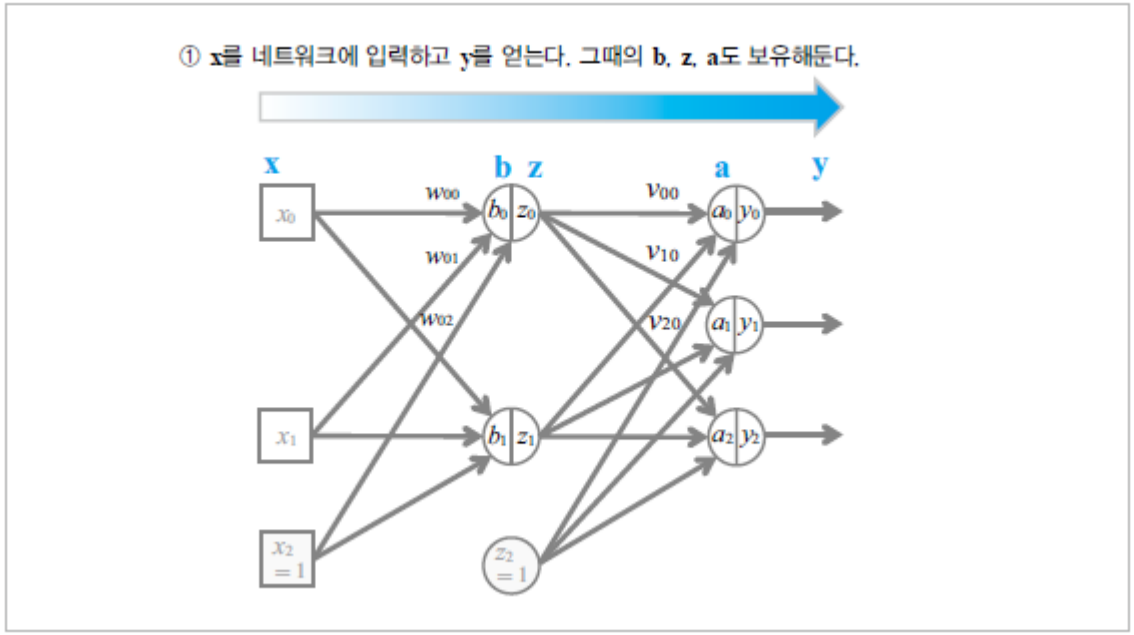
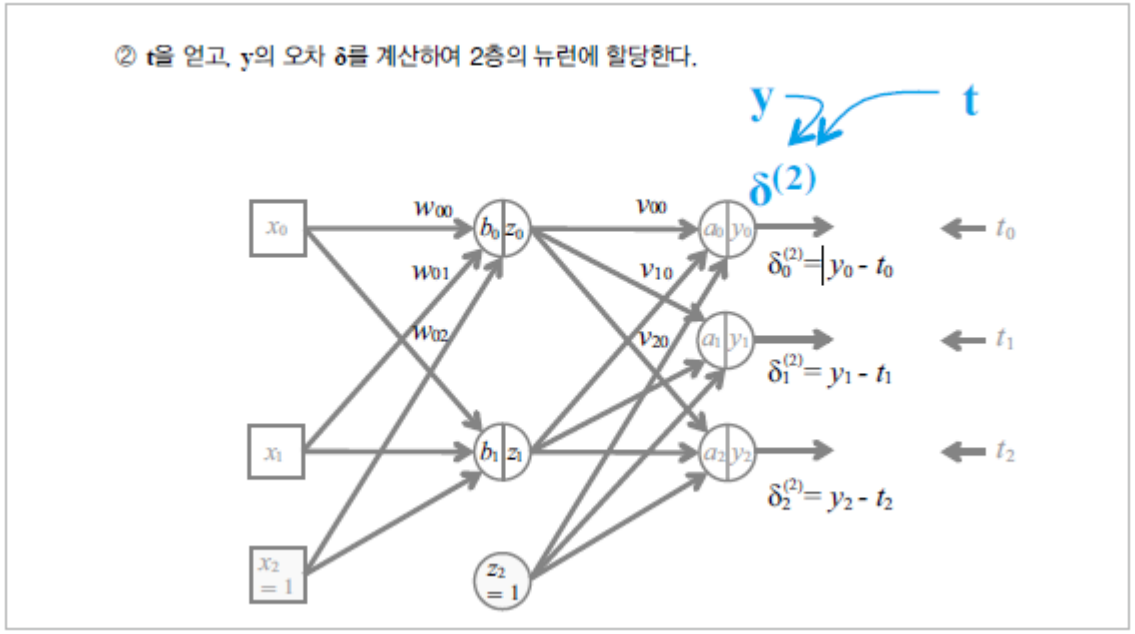


그림 7-18 오차 역전파법 ②



SECTION.02 신경망 모델

- 네트워크 매개 변수의 갱신 방법

그림 7-19 오차 역전파법 ③

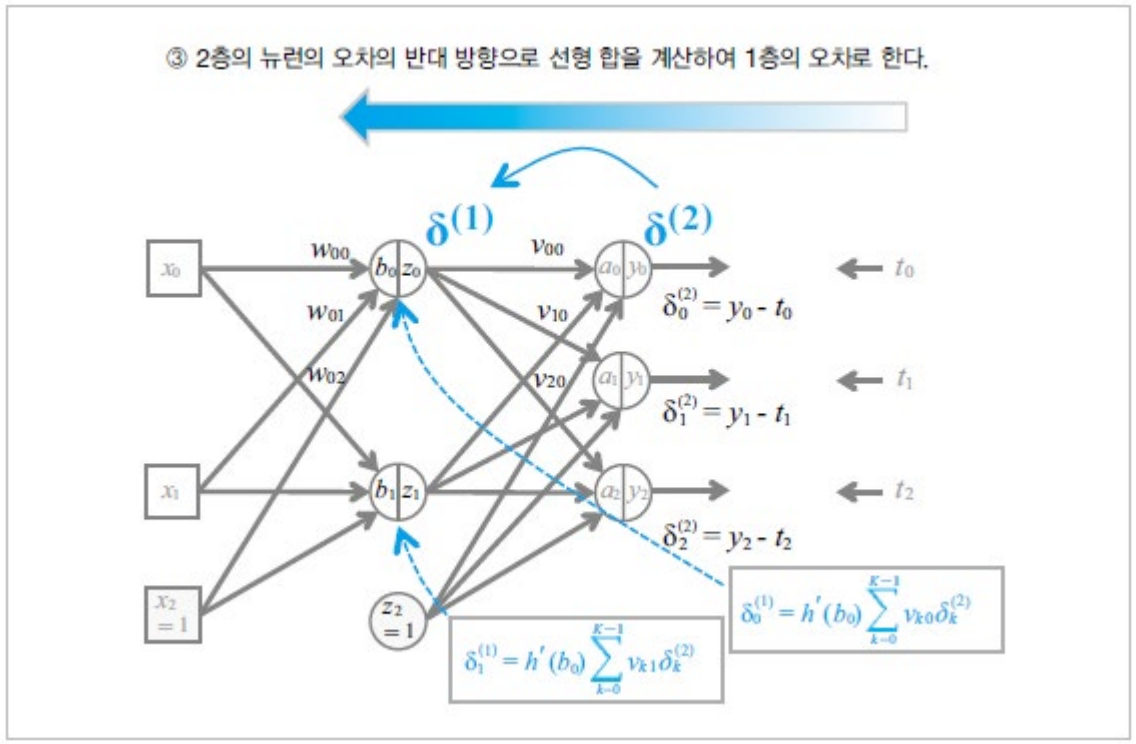
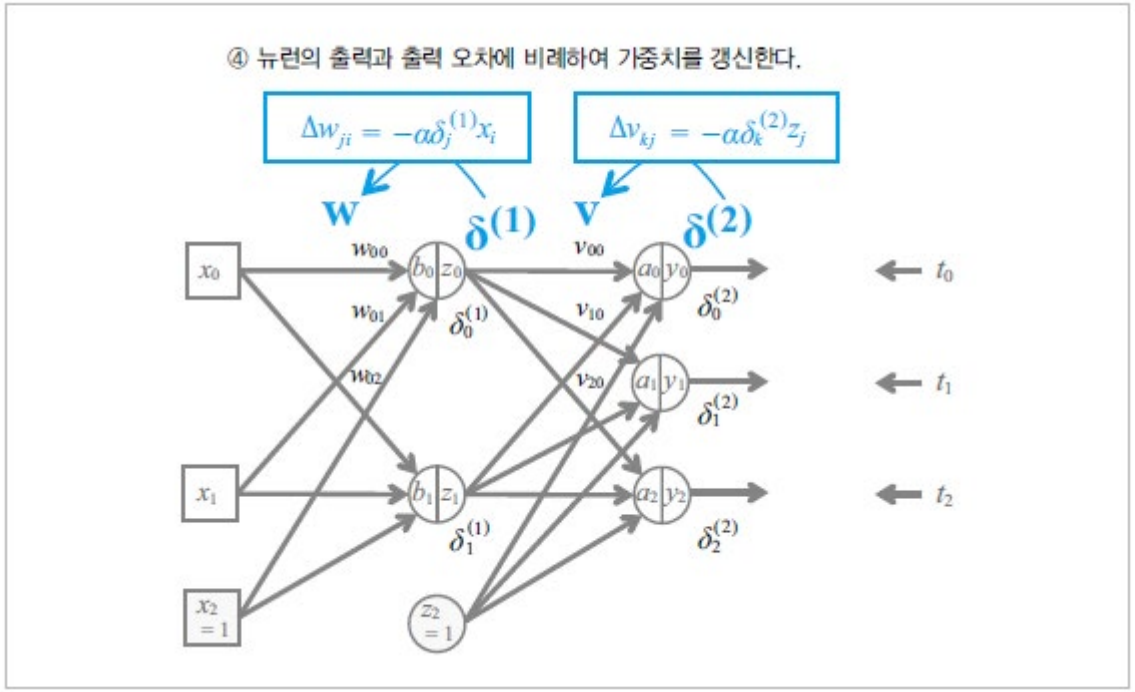


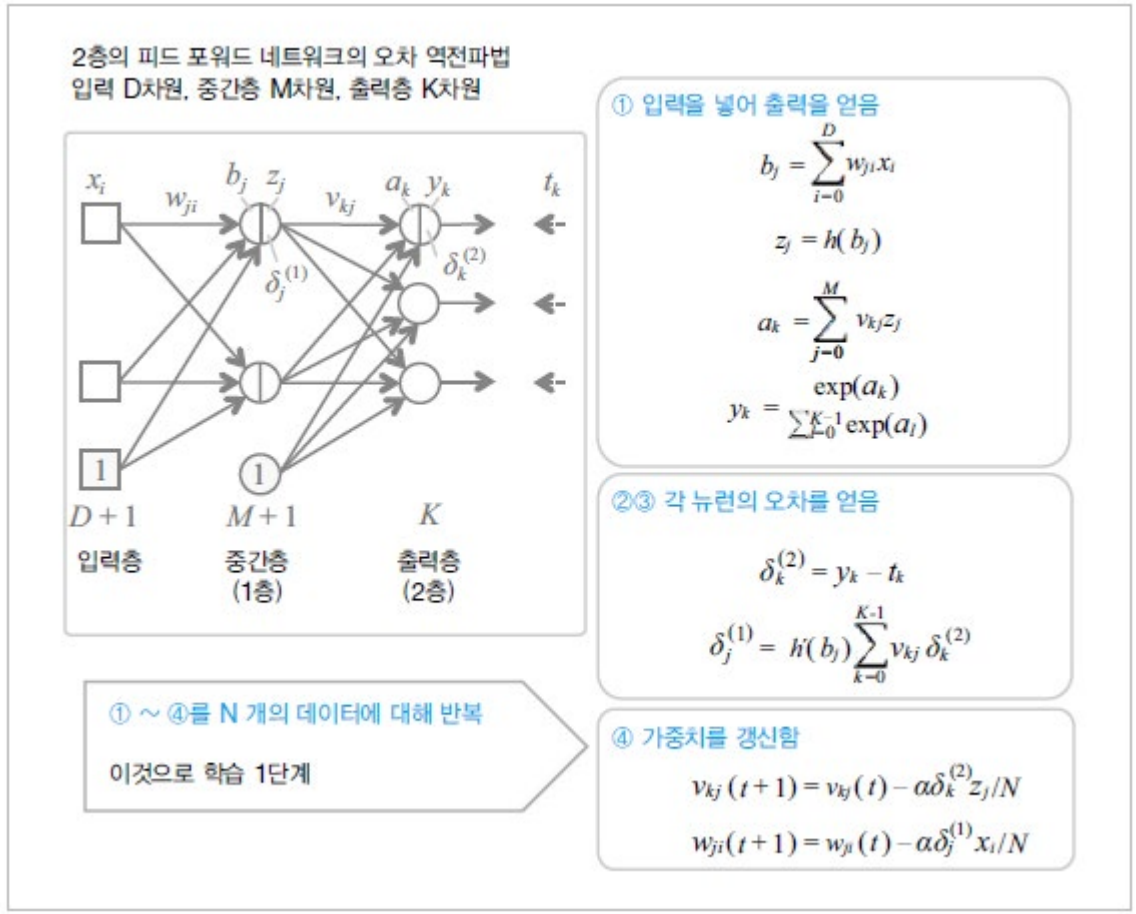
그림 7-20 오차 역전파법 ④



SECTION.02 신경망 모델

- 네트워크 매개 변수의 갱신 방법

그림 7-21 오차 역전파법 정리



2.8 오차 역전파법의 구현

- 경사 하강법, 즉 오차 역전파법으로 프로그램을 만듦.

In

```
# 리스트 7-1-(11)
# -- 해석적 미분 ----------------------------------
def dCE_FNN(wv, M, K, x, t):
    N, D = x.shape
    # wv을 w와 v로 되돌림
    w = wv[:M * (D + 1)]
    w = w.reshape(M, (D + 1))
    v = wv[M * (D + 1):]
    v = v.reshape((K, M + 1))
    # ① x를 입력하여 y를 얻음
    y, a, z, b = FNN(wv, M, K, x)
    # 출력 변수의 준비
    dwv = np.zeros_like(wv)
    dw = np.zeros((M, D + 1))
    dv = np.zeros((K, M + 1))
    delta1 = np.zeros(M) # 1층 오차
    delta2 = np.zeros(K) # 2층 오차(k = 0 부분은 사용하지 않음)
    for n in range(N): # (A)
        # ② 출력층의 오차를 구하기
        for k in range(K):
            delta2[k] = (y[n, k] - t[n, k])
        # ③ 중간층의 오차를 구하기
        for j in range(M):
            delta1[j] = z[n, j] * (1 - z[n, j]) * np.dot(v[:, j], delta2)
        # ④ v의 기울기 dv를 구하기
        for k in range(K):
            dv[k, :] = dv[k, :] + delta2[k] * z[n, :] / N
        # ④ w의 기울기 dw를 구하기
        for j in range(M):
            dw[j, :] = dw[j, :] + delta1[j] * np.r_[x[n, :], 1] / N
    # dw와 dv를 합체시킨 dwv로 만들기
    dwv = np.c_[dw.reshape((1, M * (D + 1))), \
                dv.reshape((1, K * (M + 1)))]
    dwv = dwv.reshape(-1)
    return dwv
```

```
#------Show VW
def Show_dWV(wv, M):
    N = wv.shape[0]
    plt.bar(range(1, M * 3 + 1), wv[:M * 3],
            align="center", color='black')
    plt.bar(range(M * 3 + 1, N + 1), wv[M * 3:],
            align="center", color='cornflowerblue')
    plt.xticks(range(1, N + 1))
    plt.xlim(0, N + 1)


#-- 동작 확인
M = 2
K = 3
N = 2
nWV = M * 3 + K * (M + 1)
np.random.seed(1)
WV = np.random.normal(0, 1, nWV)


dWV_ana = dCE_FNN(WV, M, K, X_train[:N, :], T_train[:N, :])
print("analytical dWV")
print(dWV_ana)


dWV_num = dCE_FNN_num(WV, M, K, X_train[:N, :], T_train[:N, :])
print("numerical dWV")
print(dWV_num)


plt.figure(1, figsize=(8, 3))
plt.subplots_adjust(wspace=0.5)
plt.subplot(1, 2, 1)
Show_dWV(dWV_ana, M)
plt.title('analitical')
plt.subplot(1, 2, 2)
Show_dWV(dWV_num, M)
plt.title('numerical')
plt.show()
```

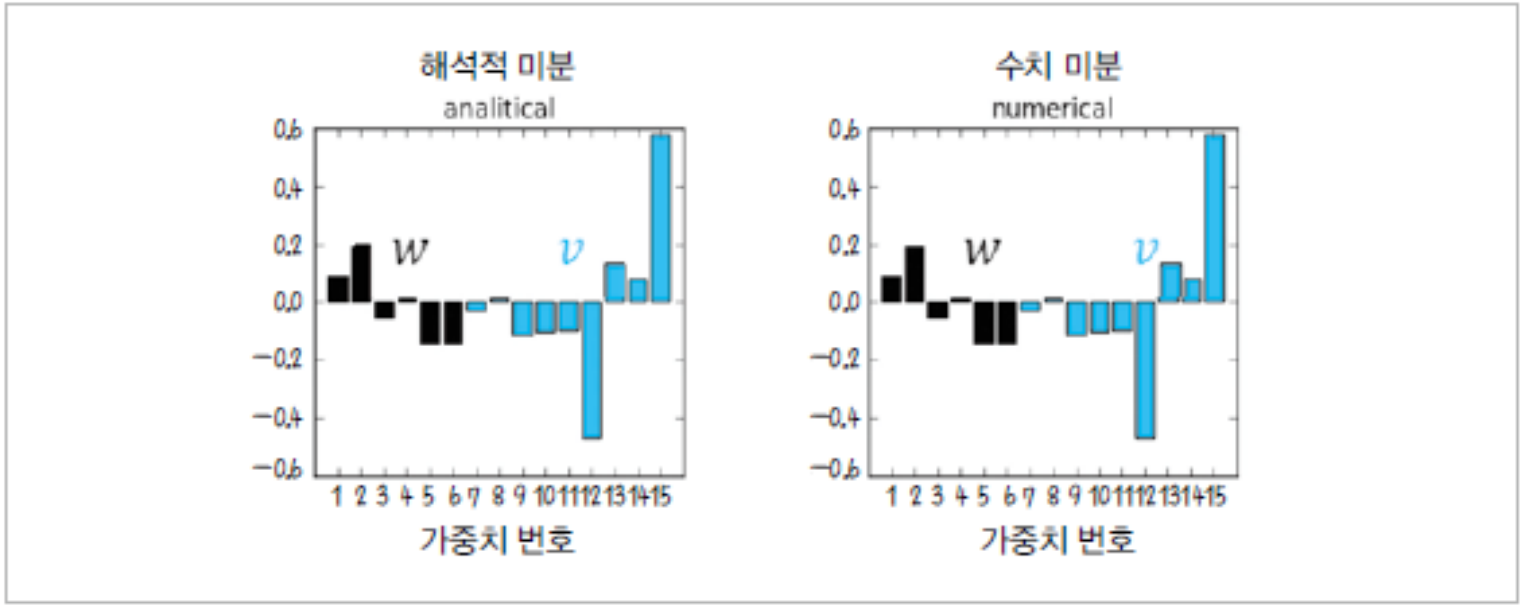
Out

```
# 실행 결과는 [그림 7-22]를 참조
```

SECTION.02 신경망 모델

- 분석 미분이 제대로 계산되어 있었던 것이 확인

그림 7-22 해석적 미분과 수치 미분



SECTION.02 신경망 모델

- 해석적 미분을 사용한 경사 하강법(오차 역전파법)의 실행 결과

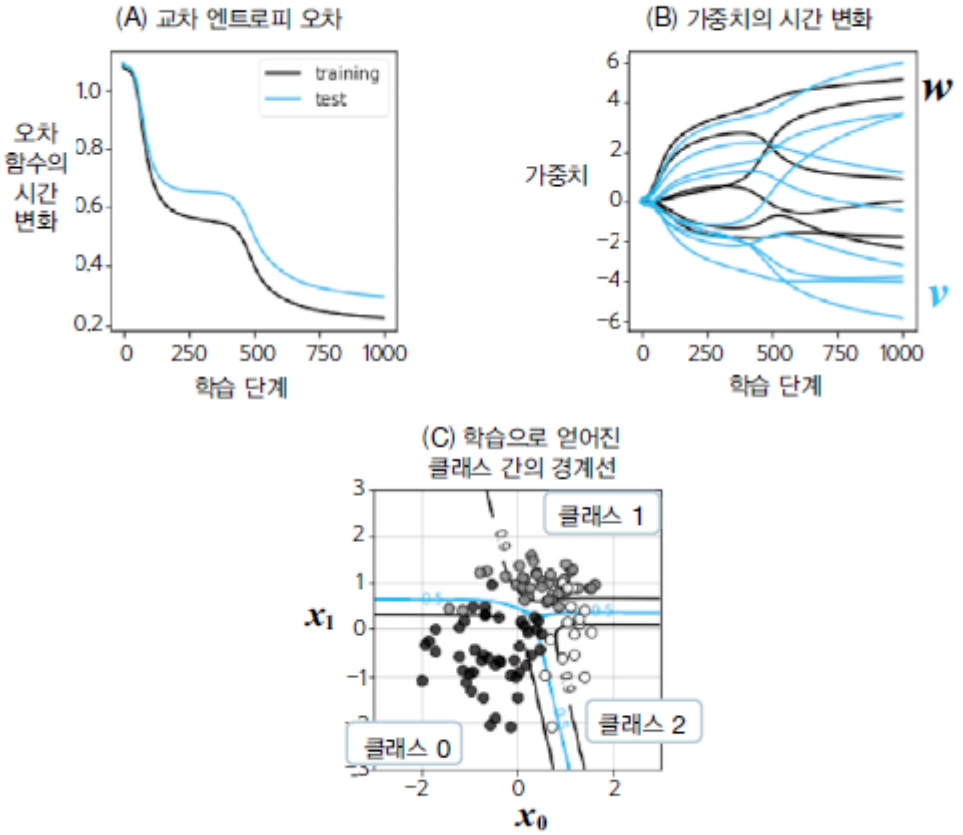
In

```
# 리스트 7-1-(13)
plt.figure(1, figsize=(12, 3))
plt.subplots_adjust(wspace=0.5)
# 학습 오차의 표시 ------------------------
plt.subplot(1, 3, 1)
plt.plot(Err_train, 'black', label='training')
plt.plot(Err_test, 'cornflowerblue', label='test')
plt.legend()
# 가중치의 시간 변화 표시 ------------------------
plt.subplot(1, 3, 2)
plt.plot(WV_hist[:, :M * 3], 'black')
plt.plot(WV_hist[:, M * 3:], 'cornflowerblue')
# 경계선 표시 ------------------------
plt.subplot(1, 3, 3)

Show_data(X_test, T_test)
M = 2
K = 3
show_FNN(WV, M, K)
plt.show()
```

Out

```
# 실행 결과는 [그림 7-23]을 참조
```

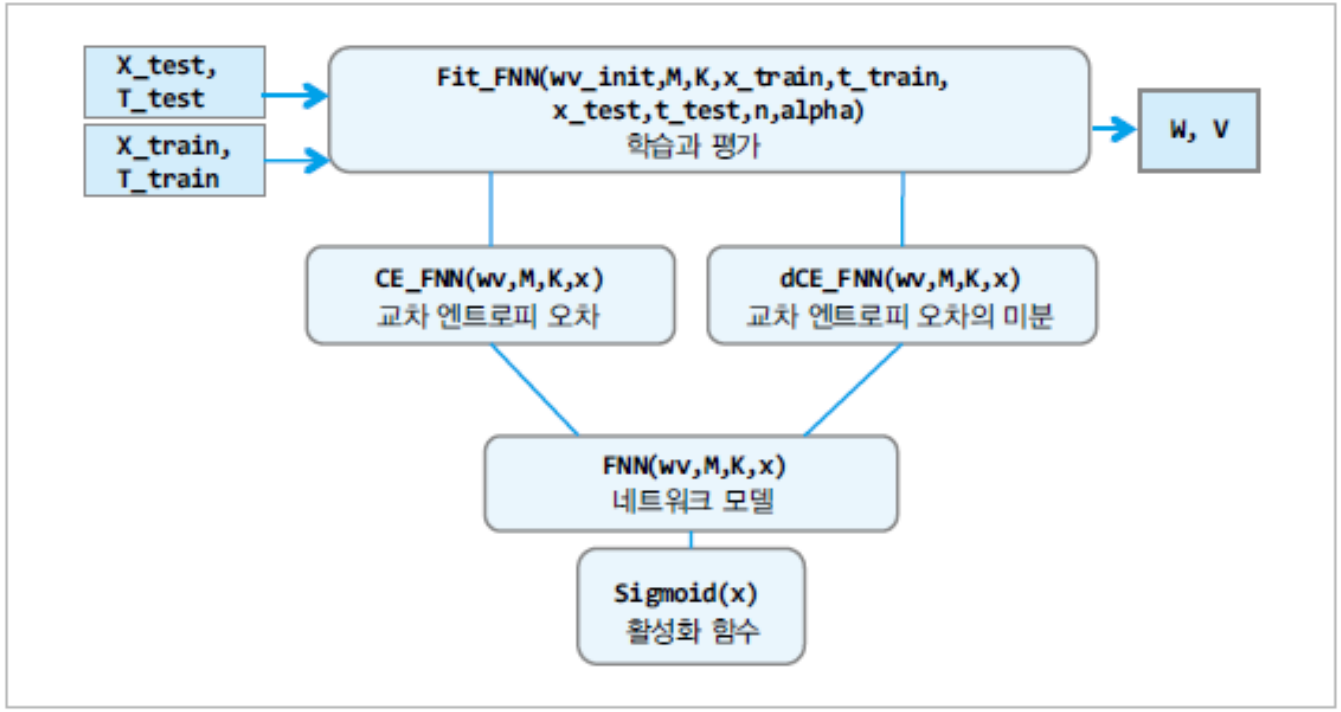
그림 7-23 해석적 미분을 사용한 경사 하강법(오차 역전파법)의 실행 결과



SECTION.02 신경망 모델

- 주요 프로그램의 관계도

그림 7-24 주요 프로그램의 관계도



2.9 학습 후 뉴런의 특성

- 오차 역전파법으로 얻은 가중치에 의한 입력 총합 · 뉴런 출력의 특성

In

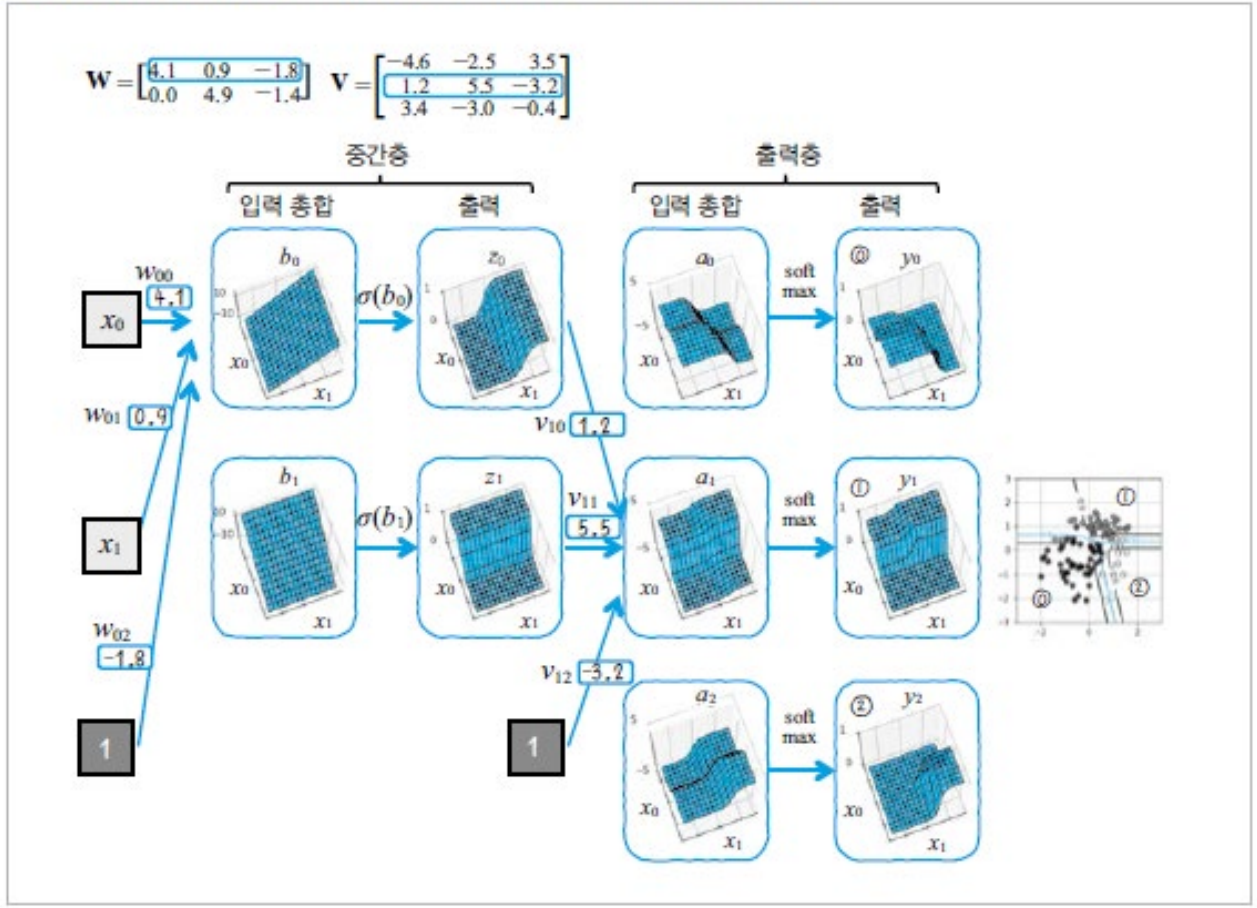
```
# 리스트 7-1-(14)
from mpl_toolkits.mplot3d import Axes3D

def show_activation3d(ax, v, v_ticks, title_str):
    f = v.copy()
    f = f.reshape(xn, xn)
    f = f.T
    ax.plot_surface(xx0, xx1, f, color='blue', edgecolor='black',
                    rstride=1, cstride=1, alpha=0.5)
    ax.view_init(70, -110)
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.set_zticks(v_ticks)
    ax.set_title(title_str, fontsize=18)


M = 2
K = 3
xn = 15 # 등고선 표시 해상도
x0 = np.linspace(X_range0[0], X_range0[1], xn)
x1 = np.linspace(X_range1[0], X_range1[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
x = np.c_[np.reshape(xx0, xn * xn, 1), np.reshape(xx1, xn * xn, 1)]
y, a, z, b = FNN(WV, M, K, x)

fig = plt.figure(1, figsize=(12, 9))
plt.subplots_adjust(left=0.075, bottom=0.05, right=0.95,
                    top=0.95, wspace=0.4, hspace=0.4)

for m in range(M):
    ax = fig.add_subplot(3, 4, 1 + m * 4, projection='3d')
    show_activation3d(ax, b[:, m], [-10, 10], '$b_{0:d}$'.format(m))
    ax = fig.add_subplot(3, 4, 2 + m * 4, projection='3d')
    show_activation3d(ax, z[:, m], [0, 1], '$z_{0:d}$'.format(m))


for k in range(K):
    ax = fig.add_subplot(3, 4, 3 + k * 4, projection='3d')
    show_activation3d(ax, a[:, k], [-5, 5], '$a_{0:d}$'.format(k))
    ax = fig.add_subplot(3, 4, 4 + k * 4, projection='3d')
    show_activation3d(ax, y[:, k], [0, 1], '$y_{0:d}$'.format(k))


plt.show()
```

Out

```
# 실행 결과는 [그림 7-25]를 참조
```

- 오차 역전파법으로 얻은 가중치에 의한 입력 총합 · 뉴런 출력의 특성

그림 7-25 오차 역전파법으로 얻은 가중치에 의한 입력 총합 · 뉴런 출력의 특성



SECTION.03 케라스로 신경망 모델 구현

3.1 2층 피드 포워드 신경망

- 케라스(Keras) 라이브러리를 사용하면 텐서플로를 쉽게 동작시킬 수 있음.
- 케라스에서 3 분류 문제를 푸는 2층 피드 포워드 네트워크를 만들어 움직여 봄.

- 필요한 라이브러리를 import하고, 저장된 데이터를 load 함.
- 데이터를 그림으로 그리는 함수를 재정의.

In

```
# 리스트 7-2-(1)
import numpy as np
import matplotlib.pyplot as plt
import time
np.random.seed(1) # (A)
import keras.optimizers # (B)
from keras.models import Sequential # (C)
from keras.layers.core import Dense, Activation # (D)


# 데이터 로드 ----------------------------
outfile = np.load('class_data.npz')
X_train = outfile['X_train']
T_train = outfile['T_train']
X_test = outfile['X_test']
T_test = outfile['T_test']
X_range0 = outfile['X_range0']
X_range1 = outfile['X_range1']
```

In

```
# 리스트 7-2-(2)
# 데이터를 그리기 ------------------------------
def Show_data(x, t):
    wk, n = t.shape
    c = [[0, 0, 0], [.5, .5, .5], [1, 1, 1]]
    for i in range(n):
        plt.plot(x[t[:, i] == 1, 0], x[t[:, i] == 1, 1],
                 linestyle='none', marker='o',
                 markeredgecolor='black',
                 color=c[i], alpha=0.8)
    plt.grid(True)
```

SECTION.03 케라스로 신경망 모델 구현

- 2층 피드백 신경망 모델을 만들고 학습 시킴.

In

```
# 리스트 7-2-(3)
# 난수 초기화
np.random.seed(1)


# --- Sequential 모델 작성
model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid',
            kernel_initializer='uniform')) # (A)
model.add(Dense(3,activation='softmax',
            kernel_initializer='uniform')) # (B)
sgd = keras.optimizers.SGD(lr=1, momentum=0.0,
                    decay=0.0, nesterov=False) # (C)
model.compile(optimizer=sgd, loss='categorical_crossentropy',
           metrics=['accuracy']) # (D)


# ---------- 학습
startTime = time.time()
history = model.fit(X_train, T_train, epochs=1000, batch_size=100,
                verbose=0, validation_data=(X_test, T_test)) # (E)


# ---------- 모델 평가
score = model.evaluate(X_test, T_test, verbose=0) # (F)
print('cross entropy {0:3.2f}, accuracy {1:3.2f}'\
     .format(score[0], score[1]))
calculation_time = time.time() - startTime
print("Calculation time:{0:.3f} sec".format(calculation_time))
```

Out

```
cross entropy 0.30, accuracy 0.88
Calculation time : 1.879 sec
```

3.3 케라스 사용의 흐름

- 자세한 내용은 케라스의 공식 홈페이지(https://keras.io/) 도움.
- 케라스에서 필요한 라이브러리를 import 함.

```
import keras.optimizers
from keras.models import Sequential
from keras.layers.core import Dense, Activation
```

- Sequential이라는 유형의 네트워크 모델로 model을 만듬.

```
model = Sequential()
```

- model에 중간층으로 Dense라는 전결합형의 층을 추가함.

```
model.add(Dense(2, input_dim=2, activation='sigmoid',
                kernel_initializer='uniform')) #(A)
```

SECTION.03 케라스로 신경망 모델 구현

• 출력층도 Dense()로 정의함.

```
model.add(Dense(3, activation='softmax',
                kernel_initializer='uniform')) #(B)
```

• 학습 방법의 설정을 keras.optimizers.SGD()에서 실시해, 그 내용을 sgd에 넣음.

```
sgd = keras.optimizers.SGD( lr=0.5, momentum=0.0,
            decay=0.0, nesterov=False) #(C)
```

• sgd을 model.compile()에 전달하여 학습 방법의 설정이 이루어짐.

```
model.compile(optimizer=sgd, loss='categorical_crossentropy',
              metrics=['accuracy']) #(D)
```

• 실제 학습은 model.fit()으로 실행함.

```
history = model.fit(X_train, T_train, batch_size=100, epochs=1000,
                    verbose=0, validation_data=(X_test, T_test)) #(E)
```

SECTION.03 케라스로 신경망 모델 구현

- model.evaluate()에서 최종 학습의 평가 값을 출력함.

```
# ---------- 모델 평가
score = model.evaluate(X_test, T_test, verbose=0) #(F)
print('loss {0:f}, acc {1:f}'.format(score[0], score[1]))
```

- 학습 과정과 그 결과를 그래프로 표시.

In

```
# 리스트 7-2-(4)
plt.figure(1, figsize = (12, 3))
plt.subplots_adjust(wspace=0.5)


# 학습 곡선 표시 ------------------------
plt.subplot(1, 3, 1)
plt.plot(history.history['loss'], 'black', label='training') # (A)
plt.plot(history.history['val_loss'], 'cornflowerblue', label='test') # (B)
plt.legend()


# 정확도 표시 ------------------------
plt.subplot(1, 3, 2)
plt.plot(history.history['acc'], 'black', label='training') # (C)
plt.plot(history.history['val_acc'], 'cornflowerblue', label='test') # (D)
plt.legend()
```

```
# 경계선 표시 ------------------------
plt.subplot(1, 3, 3)
Show_data(X_test, T_test)
xn = 60 # 등고선 표시 해상도
x0 = np.linspace(X_range0[0], X_range0[1], xn)
x1 = np.linspace(X_range1[0], X_range1[1], xn)
xx0, xx1 = np.meshgrid(x0, x1)
x = np.c_[np.reshape(xx0, xn * xn, 1), np.reshape(xx1, xn * xn, 1)]
y = model.predict(x) # (E)
K = 3
for ic in range(K):
    f = y[:, ic]
    f = f.reshape(xn, xn)
    f = f.T
    cont = plt.contour(xx0, xx1, f, levels=[0.5, 0.9], colors=[
        'cornflowerblue', 'black'])
    cont.clabel(fmt='%1.1f', fontsize=9)
    plt.xlim(X_range0)
    plt.ylim(X_range1)
plt.show()
```

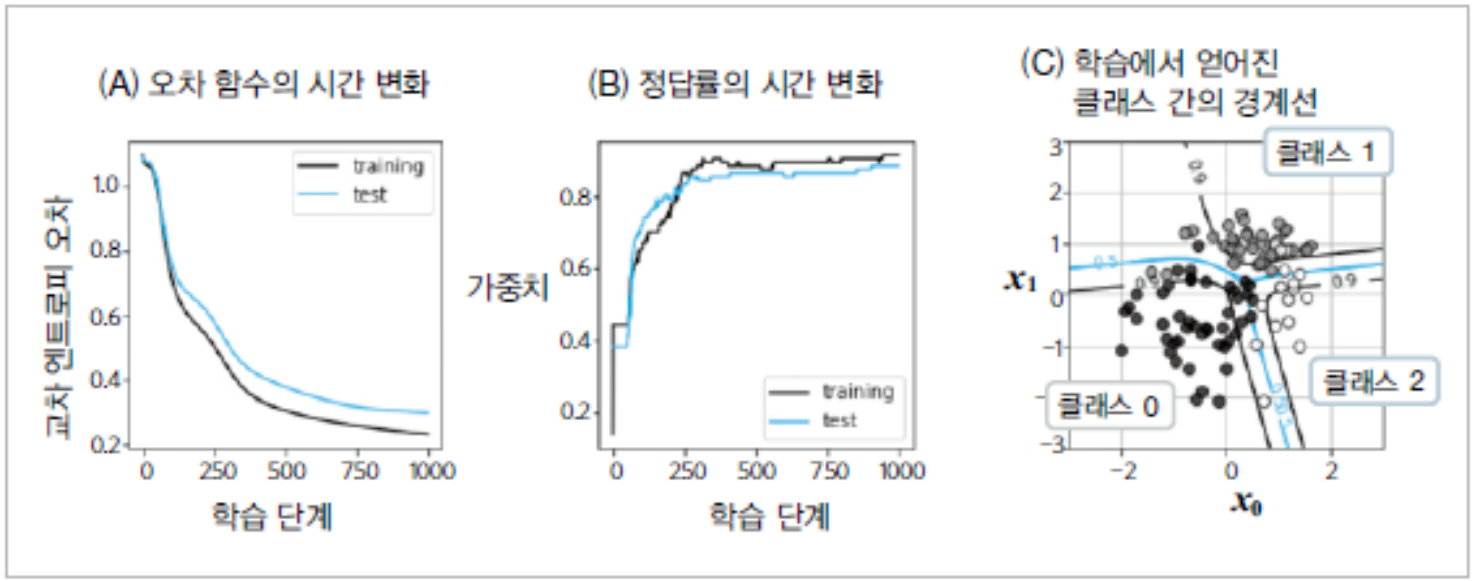
Out

```
# 실행 결과는 [그림 7-26]을 참조
```

SECTION.03 케라스로 신경망 모델 구현

- 케라스를 사용한 2층 피드 포워드 네트워크의 실행 결과

그림 7-26 케라스를 사용한 2층 피드 포워드 네트워크의 실행 결과



Contents

CHAPTER 08 신경망·딥러닝의 응용(필기체 숫자 인식)

필기 숫자 인식을 구현해 본다.

SECTION.01 MNIST 데이터베이스

- 실제로 체험해보기 위해 x_train에 저장된 처음 3개의 이미지를 표시함.

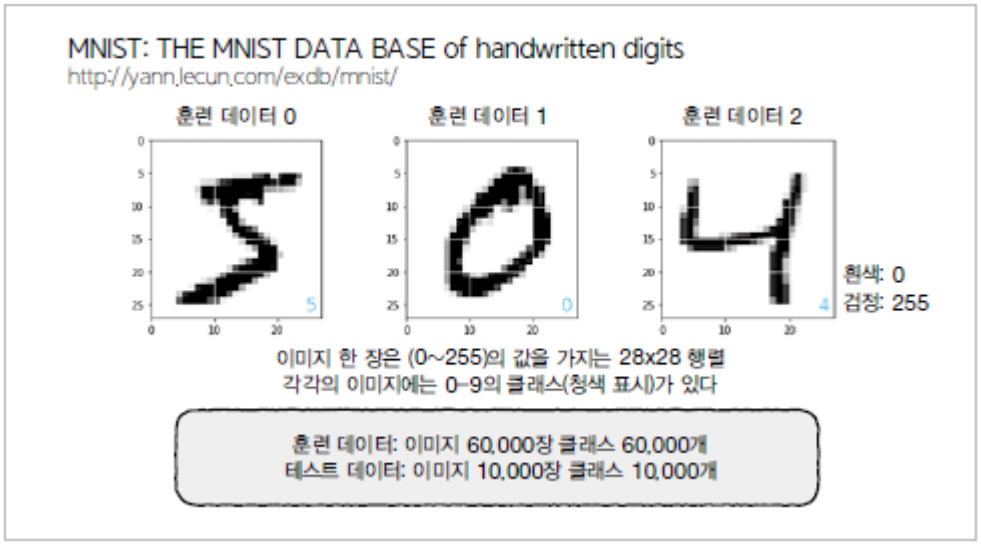
In

```
#-- 리스트 8-1-(2)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(1, figsize=(12, 3.2))
plt.subplots_adjust(wspace=0.5)
plt.gray()
for id in range(3):
    plt.subplot(1, 3, id + 1)
    img = x_train[id, :, :]
    plt.pcolor(255 - img)
    plt.text(24.5, 26, "%d" % y_train[id],
             color='cornflowerblue', fontsize=18)
    plt.xlim(0, 27)
    plt.ylim(27, 0)
    plt.grid('on', color='white')
plt.show()
```

Out

```
# 실행 결과는 [그림 8-1]을 참조
```

그림 8-1 MNIST 필기체 숫자 데이터베이스



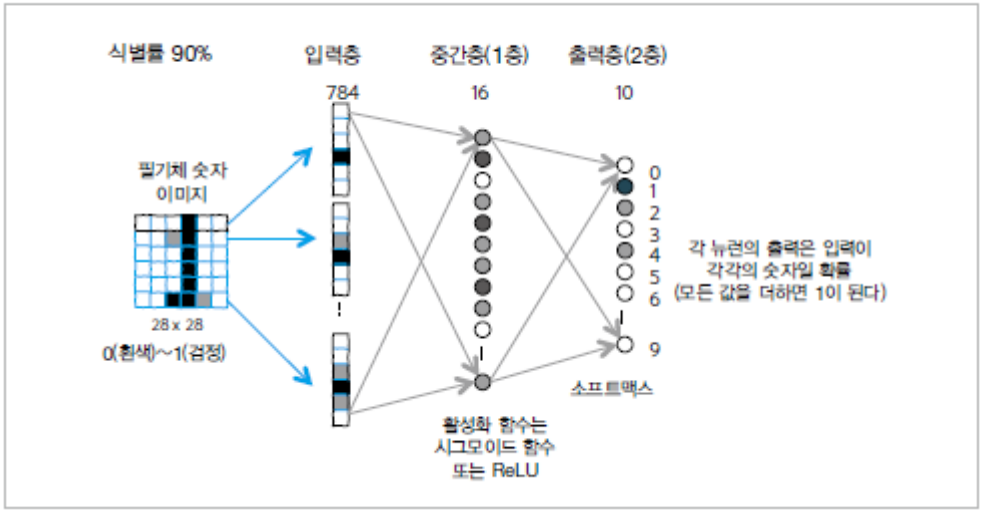
SECTION.02 2층 피드 포워드 네트워크 모델

- 2층 피드 포워드 네트워크 모델을 사용해 이 필기체 숫자의 클래스 분류 문제가 해결되는지 살펴봄.

In

```
#-- 리스트 8-1-(3)
from keras.utils import np_utils


x_train = x_train.reshape(60000, 784) # (A)
x_train = x_train.astype('float32') # (B)
x_train = x_train / 255 # (C)
num_classes = 10
y_train = np_utils.to_categorical(y_train, num_classes) # (D)

x_test = x_test.reshape(10000, 784)
x_test = x_test.astype('float32')
x_test = x_test / 255
y_test = np_utils.to_categorical(y_test, num_classes)
```

그림 8-2 필기체 숫자 인식을 위한 2층 피드 포워드 네트워크



SECTION.02 2층 피드 포워드 네트워크 모델

- 네트워크의 출력층은 10개의 숫자를 분류할 수 있도록 10개의 뉴런으로 하여, 각 뉴런의 출력값이 확률을 나타내도록 하기 위해 활성화 함수는 소프트맥스를 사용함.
- 입력과 출력을 연결하는 중간층은 16개로 하고, 활성화 함수는 시그모이드 함수로 함.

In

```
#-- 리스트 8-1-(4)
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam


model = Sequential() # (A)
model.add(Dense(16, input_dim=784, activation='sigmoid')) # (B)
model.add(Dense(10, activation='softmax')) # (C)
model.compile(loss='categorical_crossentropy',
optimizer=Adam(), metrics=['accuracy']) # (D)
```

In

```
#-- 리스트 8-1-(5)
import time


startTime = time.time()
history = model.fit(x_train, y_train, epochs=10, batch_size=1000,
                    verbose=1, validation_data=(x_test, y_test)) # (A)
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

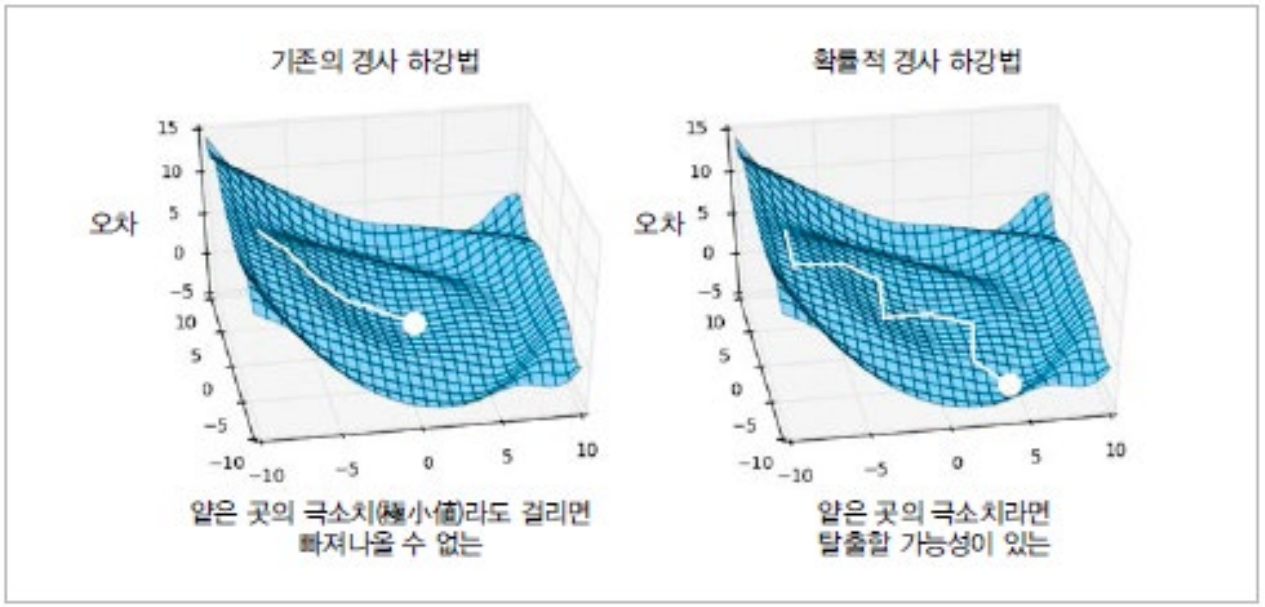
Out

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 0s - loss: 2.0609 - acc:
0.2892 - val_loss: 1.7853 - val_acc: 0.5011
Epoch 2/10
60000/60000 [==============================] - 0s - loss: 1.6047 - acc:
0.6524 - val_loss: 1.4361 - val_acc: 0.7675
(... 생략 ...)
Epoch 10/10
60000/60000 [==============================] - 0s - loss: 0.5539 - acc:
0.8892 - val_loss: 0.5282 - val_acc: 0.8951
Test loss: 0.528184900331
Test accuracy: 0.8951
Computation time:7.647 sec
```

SECTION.02 2층 피드 포워드 네트워크 모델

- 일부 데이터셋에서 계산된 기울기 방향은 전체 데이터셋에서 계산된 실제 기울기 방향과는 다름.
- 전체 오차를 최소화하는 방향으로 곧장 나아가는 것이 아니라, 노이즈의 영향을 받는 것처럼 휘청거리면서 서서히 오차가 낮은 방향으로 나아감.

그림 8-3 확률적 경사 하강법의 이미지



SECTION.02 2층 피드 포워드 네트워크 모델

• 오버 피팅이 일어나지 않았는지 확인하기 위해, 테스트 데이터 오차의 시간 변화를 살펴봄.

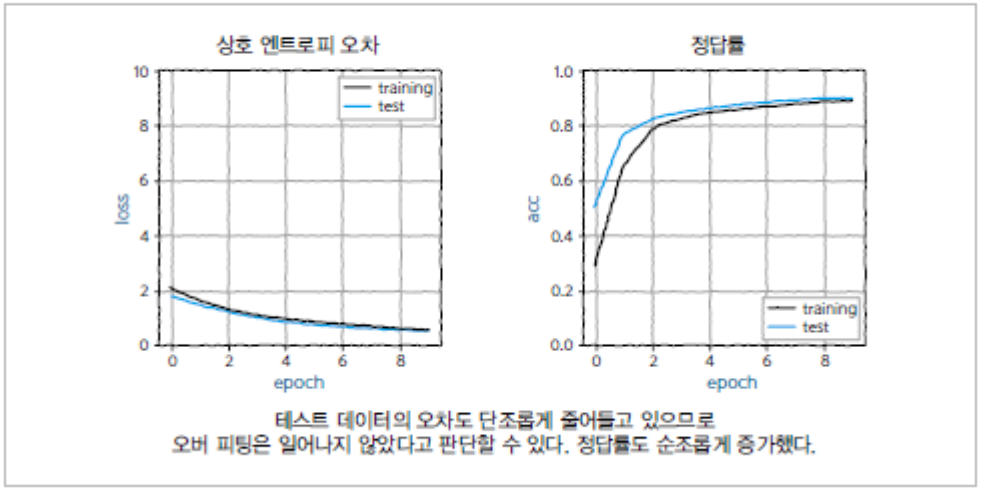
In

```
#-- 리스트 8-1-(6)
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline


plt.figure(1, figsize=(10, 4))
plt.subplots_adjust(wspace=0.5)


plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='training', color='black')
plt.plot(history.history['val_loss'], label='test',
         color='cornflowerblue')
plt.ylim(0, 10)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')


plt.subplot(1, 2, 2)
plt.plot(history.history['acc'], label='training', color='black')
plt.plot(history.history['val_acc'],label='test', color='cornflowerblue')
plt.ylim(0, 1)
plt.legend()
plt.grid()
plt.xlabel('epoch')
plt.ylabel('acc')
plt.show()
```

Out

```
# 실행 결과는 [그림 8-4]를 참조
```

그림 8-4 2층 피드 포워드 네트워크 모델의 오차와 정답률의 변화



테스트 데이터의 오차도 단조롭게 줄어들고 있으므로
오버 피팅은 일어나지 않았다고 판단할 수 있다. 정답률도 순조롭게 증가했다.

SECTION.02 2층 피드 포워드 네트워크 모델

- 실제 테스트 데이터를 입력했을 때 모델의 출력을 살펴봄.

In

```
#-- 리스트 8-1-(7)
def show_prediction():
    n_show = 96
    y = model.predict(x_test) # (A)
    plt.figure(2, figsize=(12, 8))
    plt.gray()
    for i in range(n_show):
        plt.subplot(8, 12, i + 1)
        x = x_test[i, :]
        x = x.reshape(28, 28)
        plt.pcolor(1 - x)
        wk = y[i, :]
        prediction = np.argmax(wk)
        plt.text(22, 25.5, "%d" % prediction, fontsize=12)
        if prediction != np.argmax(y_test[i, :]):
            plt.plot([0, 27], [1, 1], color='cornflowerblue', linewidth=5)
        plt.xlim(0, 27)
        plt.ylim(27, 0)
        plt.xticks([], "")
        plt.yticks([], "")

#-- 메인
show_prediction()
plt.show()
```

Out

```
# 실행 결과는 [그림 8-5]를 참조
```

그림 8-5 2층 피드 포워드 네트워크 모델의 테스트 데이터에 대한 출력 결과

오른쪽 아래의 숫자는 네트워크의 출력을 나타냄. 파란색 가로줄은 오답일 경우.

SECTION.03 ReLU 활성화 함수

- 활성화 함수는 전통적으로 시그모이드 함수가 사용, 최근에는 ReLU(Rectified Linear Unit)이 인기.

그림 8-6 ReLU 활성화 함수

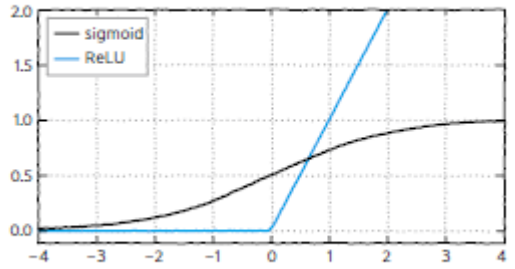
ReLU (Rectified Linear Unit)

$$h(x) = \begin{cases} x, & x > 0 \\ 0, & x \le 0 \end{cases}$$

0보다 크면 x를 그대로 출력
0 이하는 0을 출력

프로그래밍으로 쉽게 구현

```
def ReLU(0,x):
    return max(0,x)
```



- 네트워크 중간층의 활성화 함수를 ReLU로 바꾸어 실행.

In

```
#-- 리스트 8-1-(8)
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam


model = Sequential()
model.add(Dense(16, input_dim=784, activation='relu')) # (A)

model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(), metrics=['accuracy'])
```

```
startTime = time.time()
history = model.fit(x_train, y_train, batch_size=1000, epochs=10,
                    verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

Out

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 0s - loss: 1.5426 - acc:
0.5440 - val_loss: 0.8998 - val_acc: 0.8071
(... 중략 ...)
Epoch 10/10
60000/60000 [==============================] - 0s - loss: 0.2574 - acc:
0.9269 - val_loss: 0.2524 - val_acc: 0.9299
Test loss: 0.252516842544
Test accuracy: 0.9292
Computation time:7.497 sec
```

SECTION.03 ReLU 활성화 함수

- 정의된 show_prediction()을 실행하면 테스트 데이터 인식의 예를 볼 수 있음.

In
```
# 리스트 8-1-(9)
show_prediction()
plt.show()
```

Out
```
# 실행 결과는 [그림 8-7] 참조
```

그림 8-7 ReLU를 사용한 2층 피드 포워드 네트워크 모델의 출력 결과

오른쪽 아래의 숫자는 네트워크의 출력을 나타낸다. 파란색 가로줄은 오답의 경우.

SECTION.03 ReLU 활성화 함수

- 네트워크 모델의 중간층 가중치 매개 변수 획득
- 중간층 가중치 매개 변수를 실행함.

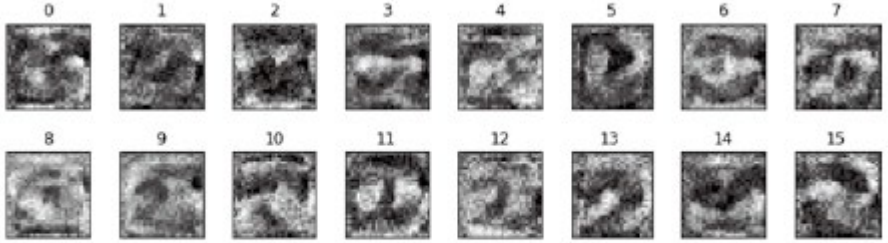
In

```
#-- 리스트 8-1-(10)
# 1층째의 가중치 시각화
w = model.layers[0].get_weights()[0]
plt.figure(1, figsize=(12, 3))
plt.gray()
plt.subplots_adjust(wspace=0.35, hspace=0.5)
for i in range(16):

    plt.subplot(2, 8, i + 1)
    w1 = w[:, i]
    w1 = w1.reshape(28, 28)
    plt.pcolor(-w1)
    plt.xlim(0, 27)
    plt.ylim(27, 0)
    plt.xticks([], "")
    plt.yticks([], "")
    plt.title("%d" % i)
plt.show()
```

Out

```
# 실행 결과는 [그림 8-8] 참조
```

그림 8-8 2층 피드 포워드 네트워크 모델에서 중간층 뉴런의 가중치

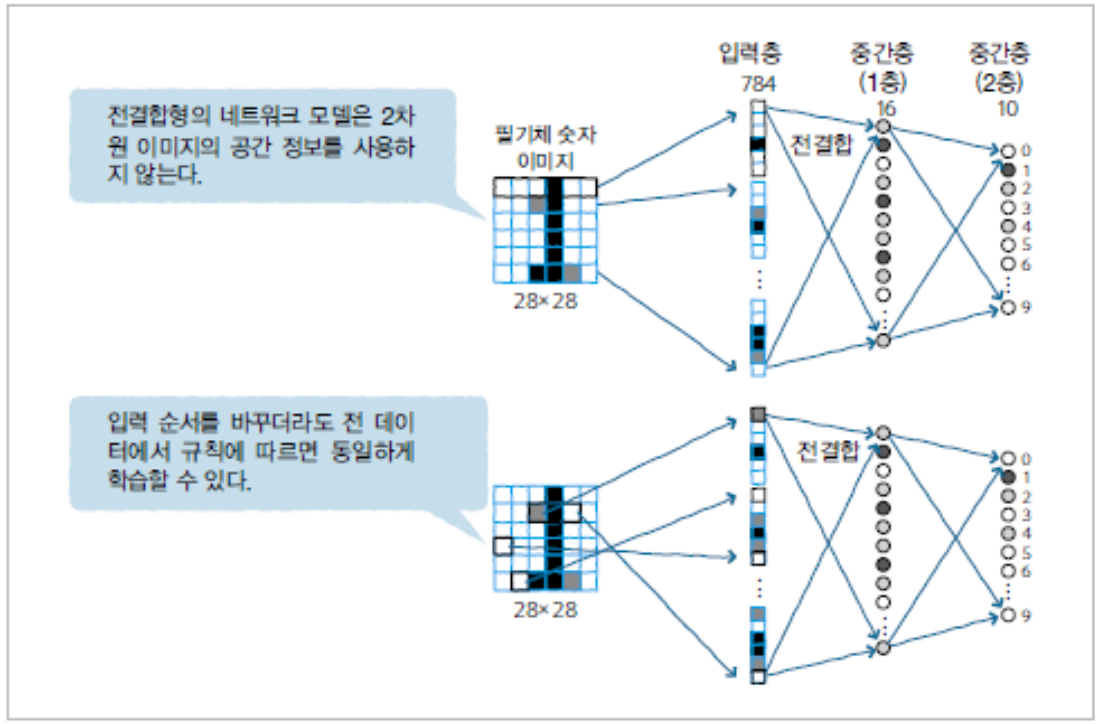


ReLU 네트워크가 학습 후 획득한 입력부터 중간층 뉴런까지의 가중치.
검은 부분이 양의 값을, 흰색 부분이 음의 값을 나타낸다.
검은 부분에 입력 이미지가 있으면 그 유닛은 활성화하고,
반대로 흰색 부분에 입력 이미지가 있는 유닛은 억제된다.

SECTION.03 ReLU 활성화 함수

- 2층 피드 포워드 네트워크는 공간 정보를 사용하지 않음.
- 네트워크의 구조가 전결합형이며 모든 입력 성분은 대등한 관계이기 때문.

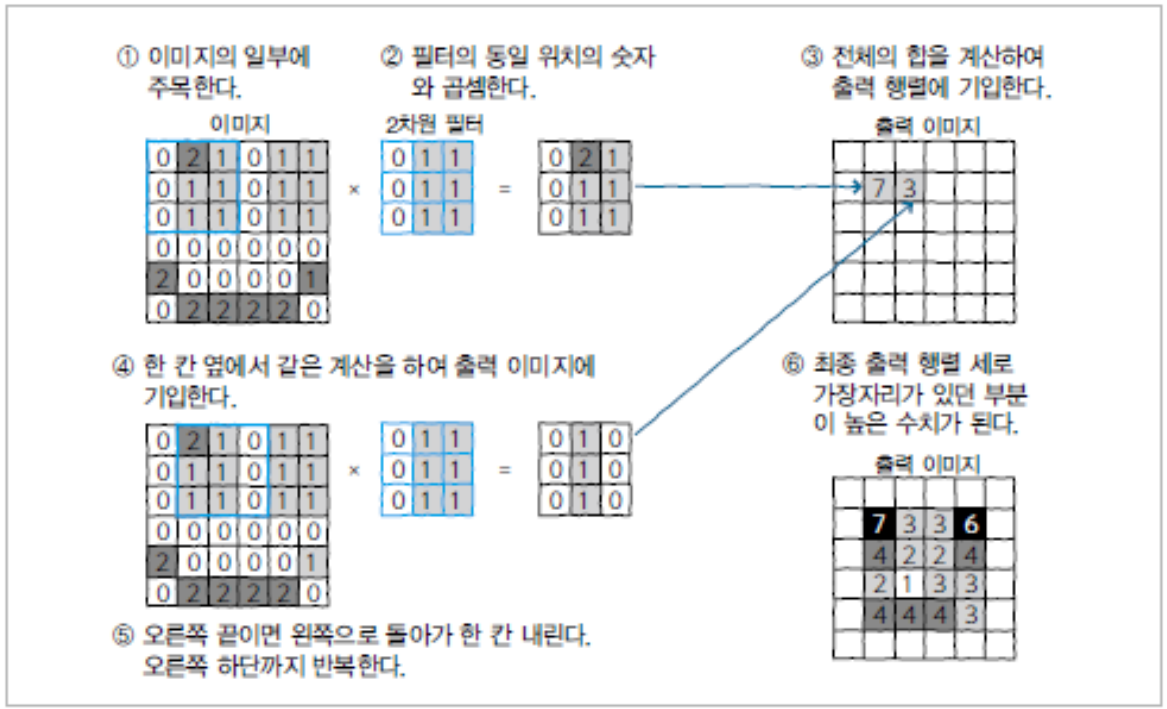
그림 8-9 2층 피드 포워드 네트워크는 공간 정보를 사용하지 않음



SECTION.04 공간 필터

- 공간 정보란 직선, 곡선, 원형이나 사각형 같은 모양을 나타내는 정보
- 이러한 형태를 골라 내는 방법으로 '공간 필터'라는 이미지 처리법이 있음.

그림 8-10 세로 엣지를 검출하는 2차원 필터, 합성곱 연산



SECTION.04 공간 필터

- 실제 필기체 숫자에 합성곱 연산을 해보기.
- MNIST 데이터를 읽는데, (data index)×28×28인 채로 사용함.

```
#-- 리스트 8-2-(1)
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
num_classes = 10
y_train = np_utils.to_categorical(y_train, num_classes)
y_test = np_utils.to_categorical(y_test, num_classes)
```

SECTION.04 공간 필터

- 가로 및 세로 엣지를 강조하는 2개의 필터를 훈련 데이터의 2번째인 '4'에 적용해 봄.

In

```
#-- 리스트 8-2-(2)
import matplotlib.pyplot as plt
%matplotlib inline

id_img = 2
myfil1 = np.array([[1, 1, 1],
                   [1, 1, 1],
                   [-2, -2, -2]], dtype=float) # (A)
myfil2 = np.array([[-2, 1, 1],
                   [-2, 1, 1],
                   [-2, 1, 1]], dtype=float) # (B)


x_img = x_train[id_img, :, :, 0]
img_h = 28
img_w = 28
x_img = x_img.reshape(img_h, img_w)
out_img1 = np.zeros_like(x_img)
out_img2 = np.zeros_like(x_img)

# 필터 처리
for ih in range(img_h - 3):
    for iw in range(img_w - 3):
        img_part = x_img[ih:ih + 3, iw:iw + 3]
        out_img1[ih + 1, iw + 1] = \
            np.dot(img_part.reshape(-1), myfil1.reshape(-1))
        out_img2[ih + 1, iw + 1] = \
            np.dot(img_part.reshape(-1), myfil2.reshape(-1))

#-- 표시
plt.figure(1, figsize=(12, 3.2))
plt.subplots_adjust(wspace=0.5)
plt.gray()
plt.subplot(1, 3, 1)
plt.pcolor(1 - x_img)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.subplot(1, 3, 2)
plt.pcolor(-out_img1)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.subplot(1, 3, 3)
plt.pcolor(-out_img2)
plt.xlim(-1, 29)
plt.ylim(29, -1)
plt.show()
```

Out

```
# 실행 결과는 [그림 8-11]을 참조
```

SECTION.04 공간 필터

- 2차원 필터를 필기체 숫자 데이터에 적용 실행 결과
- 필터를 적용하면 출력 이미지의 크기는 작아지며, 불편함도 생김. 대응책으로 패딩Padding 방법이 있음.

그림 8-11 2차원 필터를 필기체 숫자 데이터에 적용

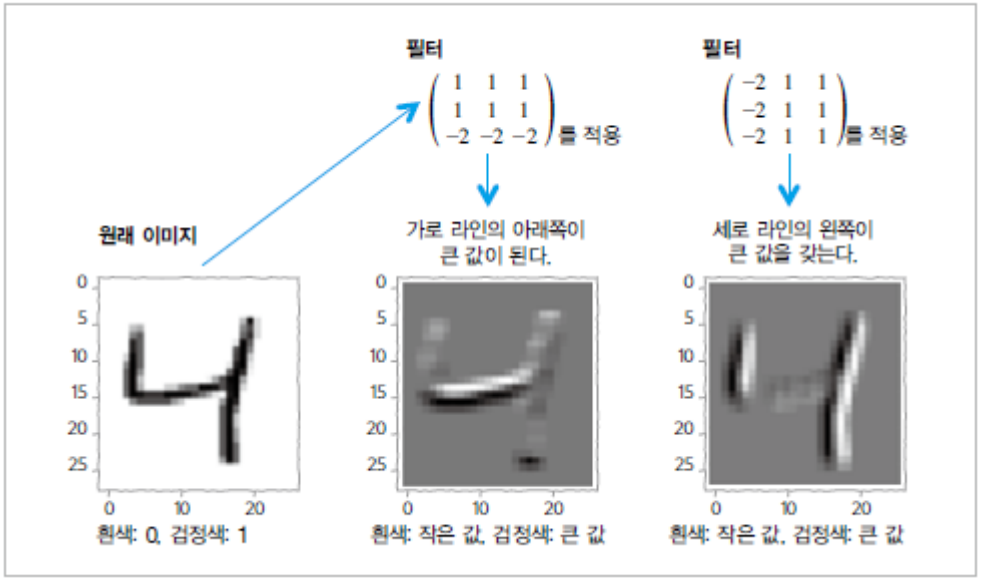
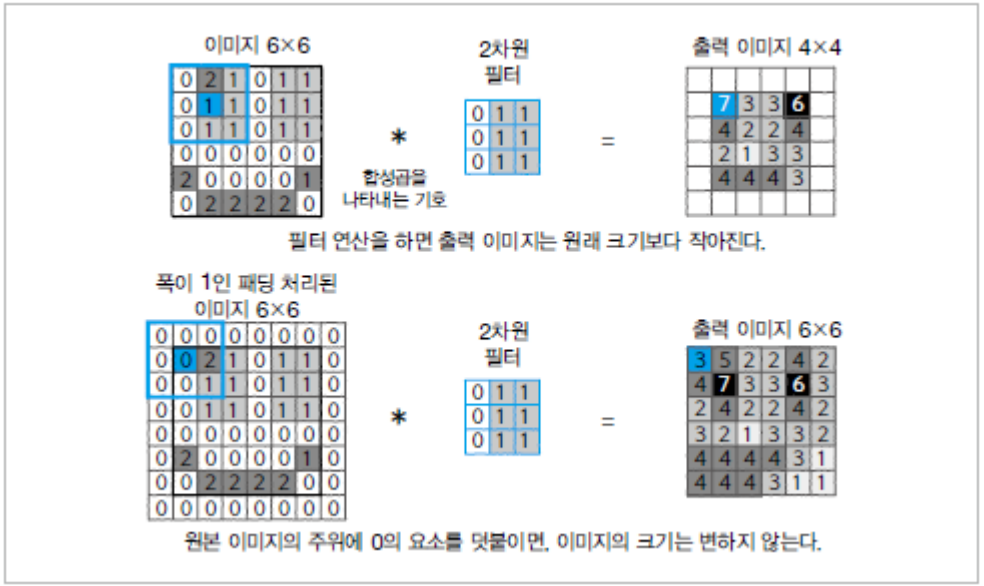


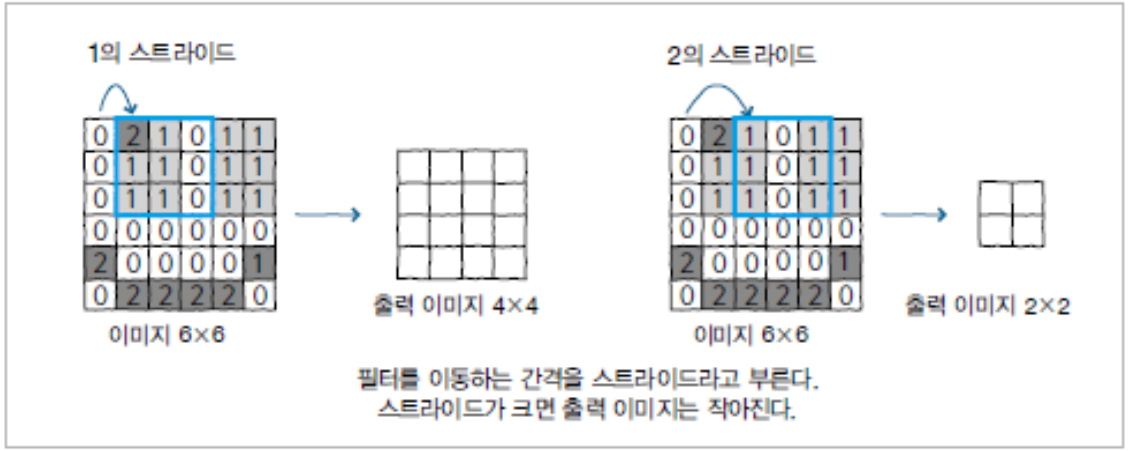
그림 8-12 패딩



SECTION.04 공간 필터

- 지금까지의 필터는 한 칸씩 이동했지만, 2칸이나 3칸 등 어떤 간격이든 이동할 수 있음.
- 이 간격을 스트라이드Stride라고 함.
- 스트라이드를 크게 하면 출력 이미지가 작아짐.
- 패딩과 스트라이드 값은 라이브러리로 합성곱 네트워크를 사용할 때 인수로 전달하게 됨.

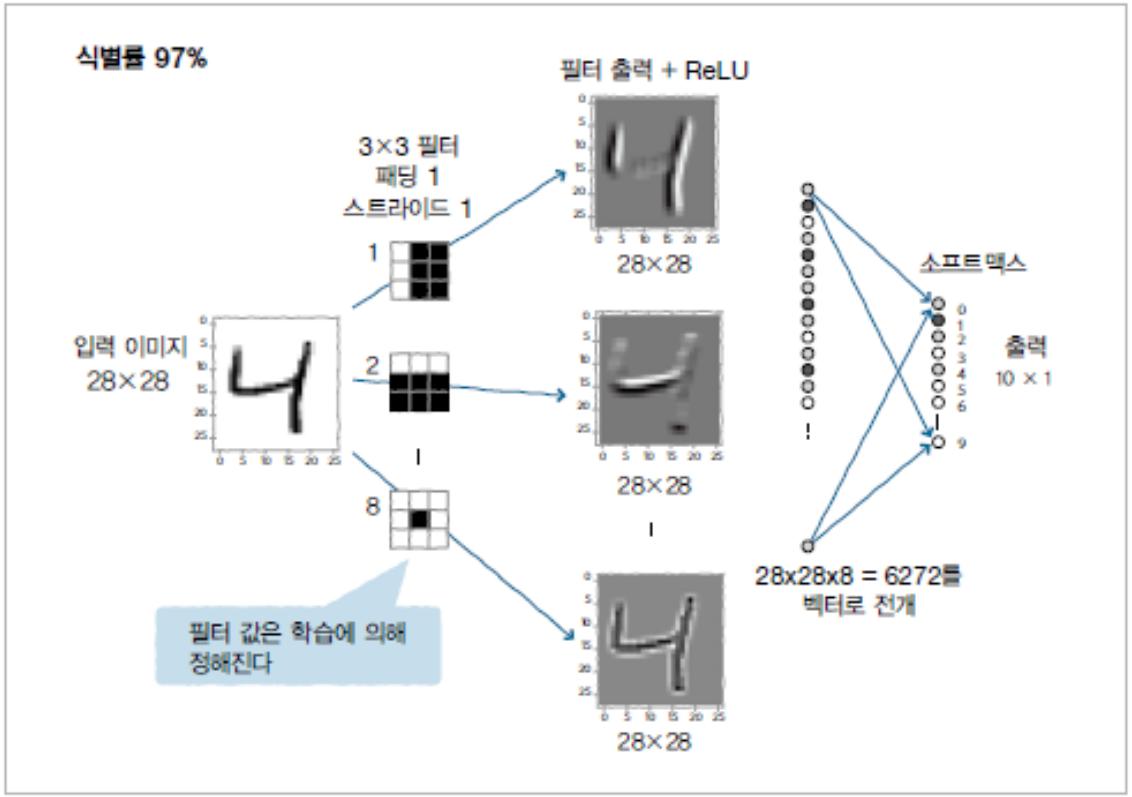
그림 8-13 스트라이드



SECTION.05 합성곱 신경망

- 필터를 사용한 신경망을 합성곱 신경망Convolution Neural Network: CNN이라고 함.

그림 8-14 2층 합성곱 신경망



SECTION.05 합성곱 신경망

- CNN을 케라스로 구현

In

```
#-- 리스트 8-2-(3)
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import Adam
import time


model = Sequential()
model.add(Conv2D(8, (3, 3), padding='same',
                 input_shape=(28, 28, 1), activation='relu')) # (A)
model.add(Flatten()) # (B)
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
startTime = time.time()
history = model.fit(x_train, y_train, batch_size=1000, epochs=20,
                    verbose=1, validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

Out

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 7s - loss: 0.7694 - acc:
0.8154 - val_loss: 0.3387 - val_acc: 0.9043
Epoch 2/20
60000/60000 [==============================] - 7s - loss: 0.3161 - acc:
0.9093 - val_loss: 0.2741 - val_acc: 0.9216
Epoch 3/20
(중략)
Test loss: 0.0957389078975
Test accuracy: 0.9707
Computation time:226.190 sec
```

SECTION.05 합성곱 신경망

- 2층 합성곱 네트워크의 학습에서 얻은 필터와 그 적용 이미지

In

```
# 리스트 8-2-(5)
plt.figure(1, figsize=(12, 2.5))
plt.gray()
plt.subplots_adjust(wspace=0.2, hspace=0.2)
plt.subplot(2, 9, 10)
id_img = 12
x_img = x_test[id_img, :, :, 0]
img_h = 28
img_w = 28
x_img = x_img.reshape(img_h, img_w)
plt.pcolor(-x_img)
plt.xlim(0, img_h)
plt.ylim(img_w, 0)
plt.xticks([], "")
plt.yticks([], "")
plt.title("Original")
w = model.layers[0].get_weights()[0] # (A)
max_w = np.max(w)
min_w = np.min(w)
for i in range(8):
    plt.subplot(2, 9, i + 2)
    w1 = w[:, :, 0, i]
    w1 = w1.reshape(3, 3)
    plt.pcolor(-w1, vmin=min_w, vmax=max_w)
    plt.xlim(0, 3)
    plt.ylim(3, 0)
    plt.xticks([], "")
    plt.yticks([], "")
    plt.title("%d" % i)
    plt.subplot(2, 9, i + 11)
    out_img = np.zeros_like(x_img)
    # 필터 처리
    for ih in range(img_h - 3):
        for iw in range(img_w - 3):
            img_part = x_img[ih:ih + 3, iw:iw + 3]
            out_img[ih + 1, iw + 1] = \
            np.dot(img_part.reshape(-1), w1.reshape(-1))
    plt.pcolor(-out_img)
    plt.xlim(0, img_w)
    plt.ylim(img_h, 0)
    plt.xticks([], "")
    plt.yticks([], "")
plt.show()
```

Out

```
# 실행 결과는 [그림 8-16]을 참조
```

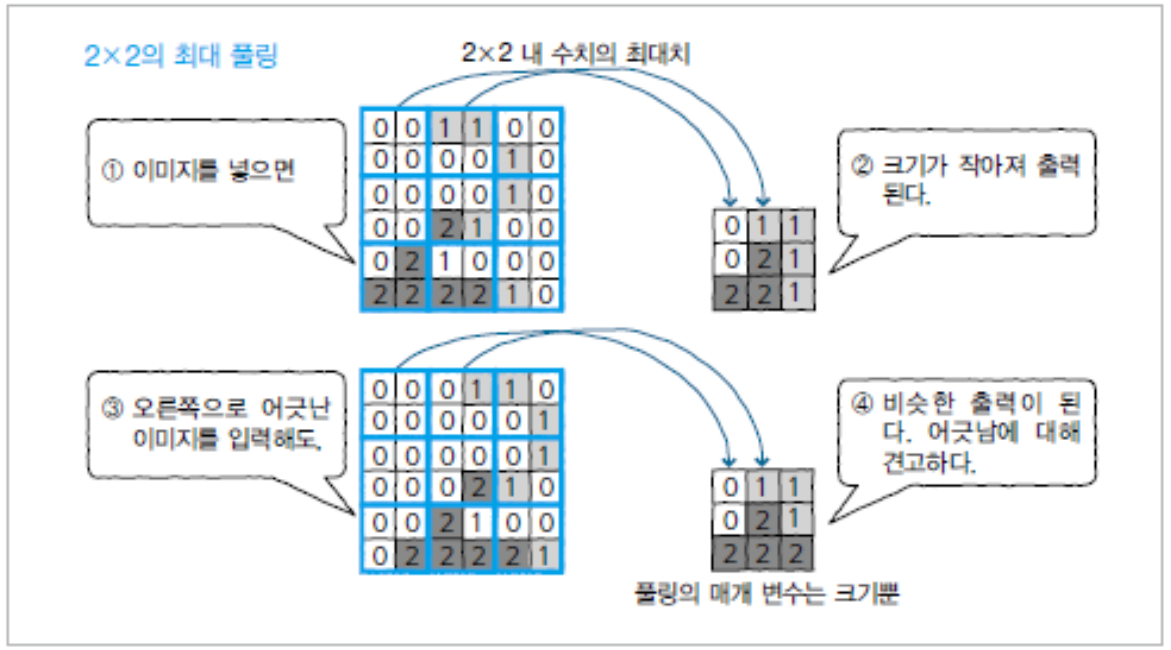
그림 8-16 2층 합성곱 네트워크의 학습에서 얻은 필터와 그 적용 이미지

'9'의 이미지에 각각의 필터를 적용한 이미지
(작은 값은 흰색, 큰 값은 검은색으로 나타남)

SECTION.06 풀링

- 풀링(pooling)은 합성곱층 데이터의 공간적 크기를 축소하는 데 사용함.
- 필기체 숫자 '2'가 1픽셀만 어긋난 이미지를 입력해도 각 배열의 수치는 완전히 달라지며, 네트워크에서는 완전히 다른 패턴으로 인식되어 버림. 문제를 해결하는 방법으로 '풀링 처리'가 있음.

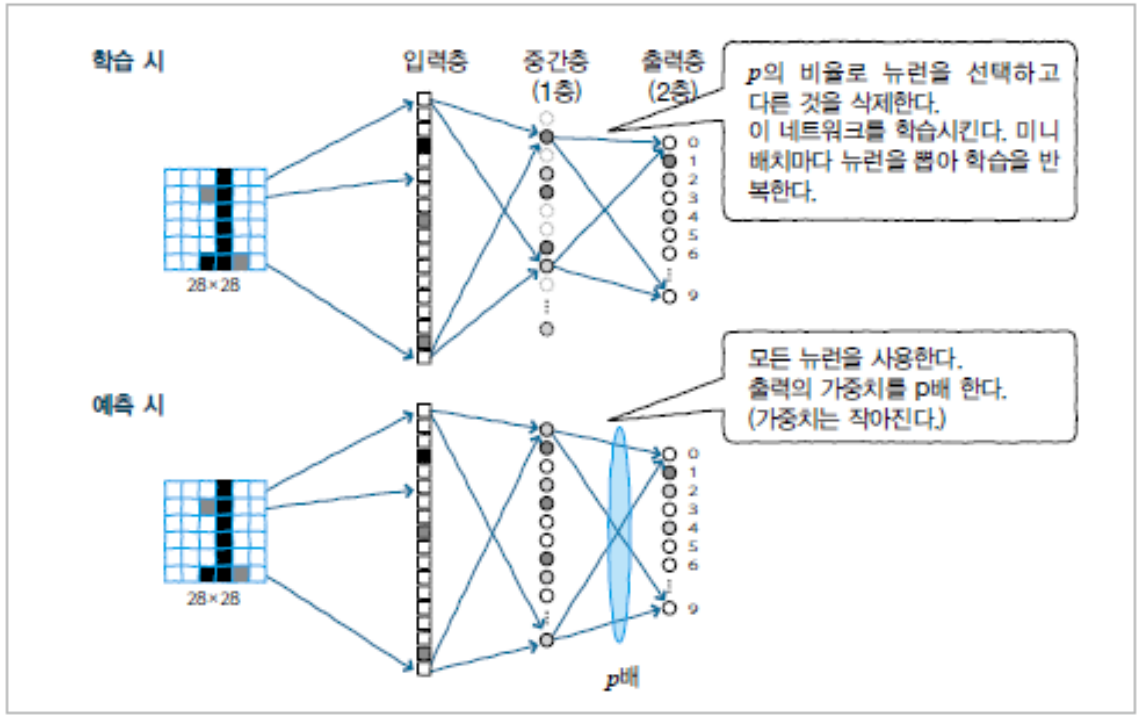
그림 8-17 풀링



SECTION.07 드롭아웃

- 드롭아웃은 네트워크의 학습을 개선하는 방법.
- 드롭아웃(dropout)은 신경망 전체를 다 학습시키지 않고 일부 노드만 무작위로 골라 학습시키는 기법.
- 드롭아웃은 여러 네트워크를 각각 학습시켜 예측 시에 네트워크를 평균화해 합치는 효과가 있음.

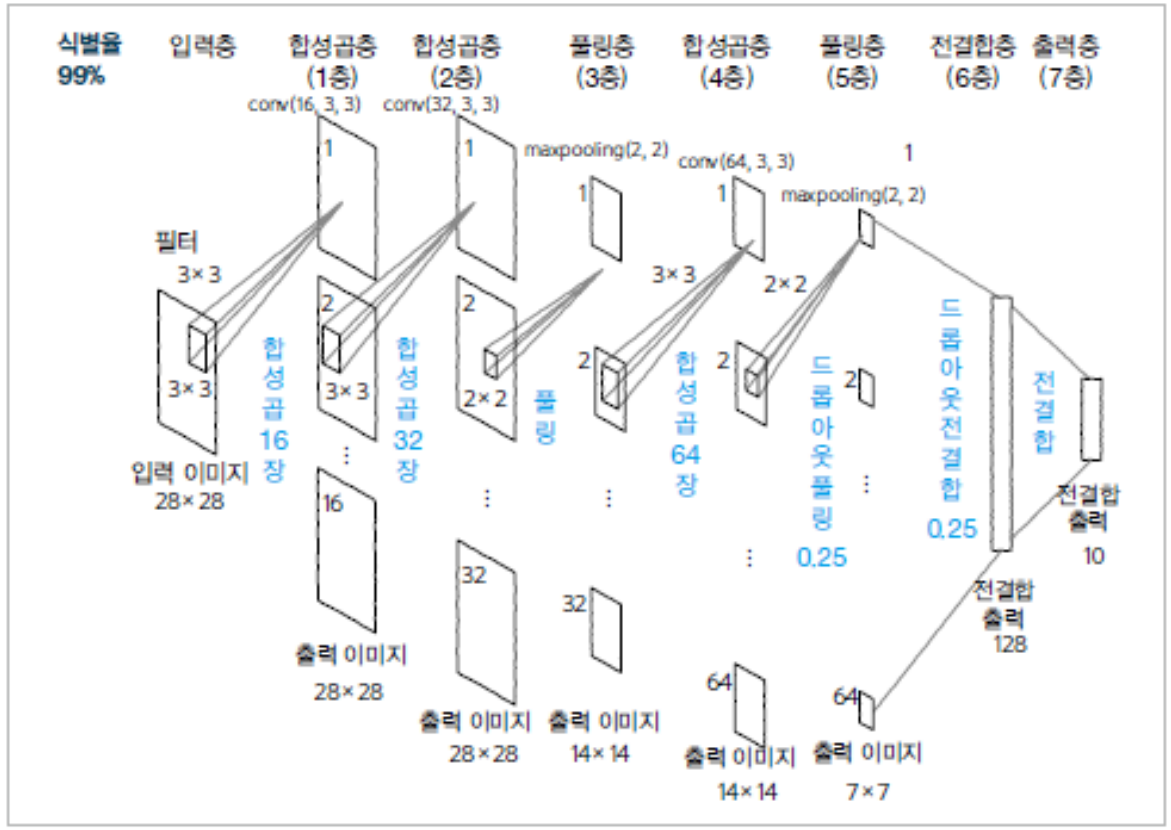
그림 8-18 드롭아웃



SECTION.08 MNIST 인식 네트워크 모델

- 합성곱 네트워크에 풀링과 드롭아웃을 도입하여 계층의 수를 늘리고, 모두를 갖추고 있는 네트워크를 마지막으로 구축해 봄.

그림 8-19 집대성한 네트워크



SECTION.08 MNIST 인식 네트워크 모델

- 집대성한 네트워크를 만들어 학습을 실시함.

In

```
# 리스트 8-2-(6)
import numpy as np
np.random.seed(1)
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import Adam
import time


model = Sequential()
model.add(Conv2D(16, (3, 3),
                 input_shape=(28, 28, 1), activation='relu'))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) # (A)
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2))) # (B)
model.add(Dropout(0.25)) # (C)
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25)) # (D)
model.add(Dense(num_classes, activation='softmax'))


model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])


startTime = time.time()


history = model.fit(x_train, y_train, batch_size=1000, epochs=20,
                    verbose=1, validation_data=(x_test, y_test))


score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
print("Computation time:{0:.3f} sec".format(time.time() - startTime))
```

Out

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 64s - loss: 0.6143 - acc:
0.8118 - val_loss: 0.1179 - val_acc: 0.9645
Epoch 2/20
(중략)
60000/60000 [==============================] - 64s - loss: 0.0161 - acc:
0.9945 - val_loss: 0.0210 - val_acc: 0.992


Test loss: 0.0208244939562
Test accuracy: 0.9931
Computation time:1877.519 sec
```

SECTION.08 MNIST 인식 네트워크 모델

- 집대성한 네트워크에서 테스트 데이터의 출력 결과

In

```
# 리스트 8-2-(7)
show_prediction()
plt.show()
```

Out

```
# 실행 결과는 [그림 8-20]을 참조
```

그림 8-20 집대성한 네트워크에서 테스트 데이터의 출력 결과

오른쪽 아래 숫자는 네트워크의 출력을 나타낸다. 이 테스트 데이터는 모두 정답.