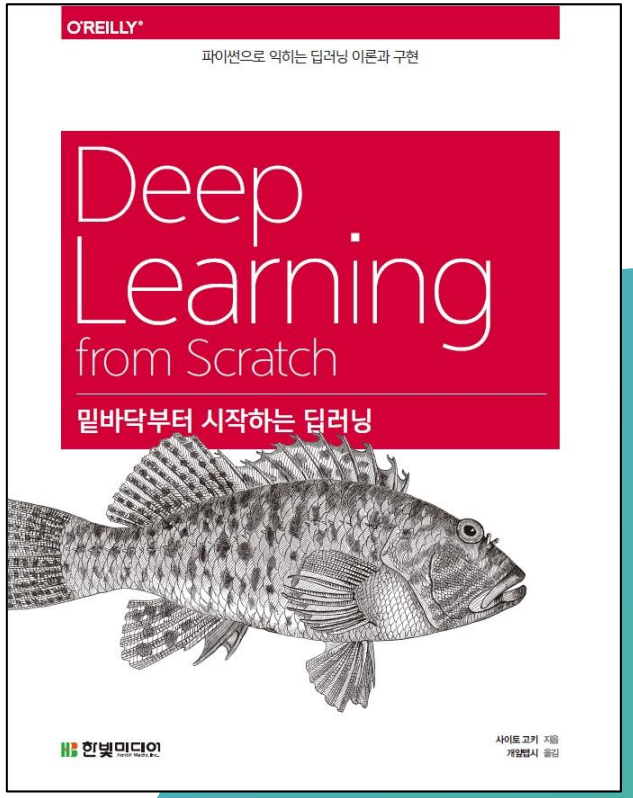
CHAPTER 5 오차역전파법(Error Back-Propagation Algorithm)

밑바닥부터 시작하는 딥러닝



이 책의 학습 목표

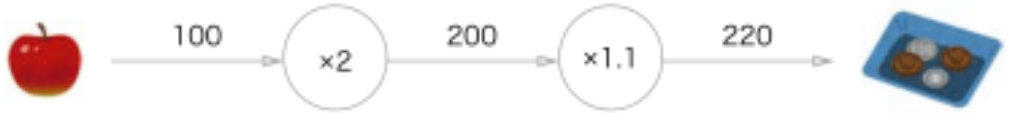
- CHAPTER 1 파이썬에 대해 간략하게 살펴보고 사용법 익히기
- CHAPTER 2 퍼셉트론에 대해 알아보고 퍼셉트론을 써서 간단한 문제를 풀어보기
- CHAPTER 3 신경망의 개요, 입력 데이터가 무엇인지 신경망이 식별하는 처리 과정 알아보기
- CHAPTER 4 손실 함수의 값을 가급적 작게 만드는 경사법에 대해 알아보기
- **CHAPTER 5 가중치 매개변수의 기울기를 효율적으로 계산하는 오차역전파법 배우기**
- CHAPTER 6 신경망(딥러닝) 학습의 효율과 정확도를 높이기
- CHAPTER 7 CNN의 메커니즘을 자세히 설명하고 파이썬으로 구현하기
- CHAPTER 8 딥러닝의 특징과 과제, 가능성, 오늘날의 첨단 딥러닝에 대해 알아보기

Contents

SECTION 05 오차역전파법

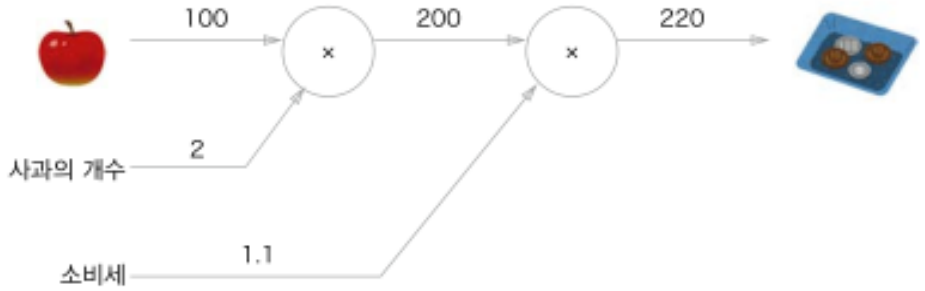
5.1.1 계산 그래프로 풀다

그림 5-1 계산 그래프로 풀어본 문제 1 의 답



문제1: 현빈군은 슈퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요. 단, 소비세 10%가 부과됩니다.

그림 5-2 계산 그래프로 풀어본 문제 1 의 답 : '사과의 개수'와 '소비세'를 변수로 취급해 원 밖에 표기

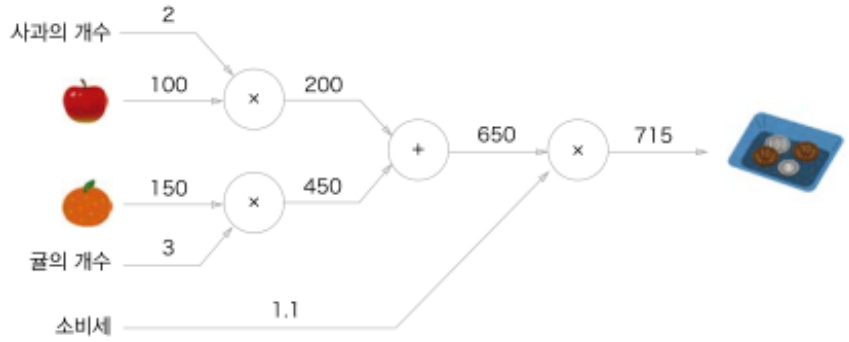


문제2: 현빈군은 슈퍼에서 사과(100원)를 2개, 귤(150원)을 3개 샀습니다. 이때 지불 금액을 구하세요. 단, 소비세 10%가 부과됩니다.

지금까지 살펴본 것처럼 계산 그래프를 이용한 문제풀이는 다음 흐름으로 진행한다.

1. 계산 그래프를 구성한다.
2. 그래프에서 계산을 왼쪽에서 오른쪽으로

그림 5-3 계산 그래프로 풀어본 문제 2 의 답

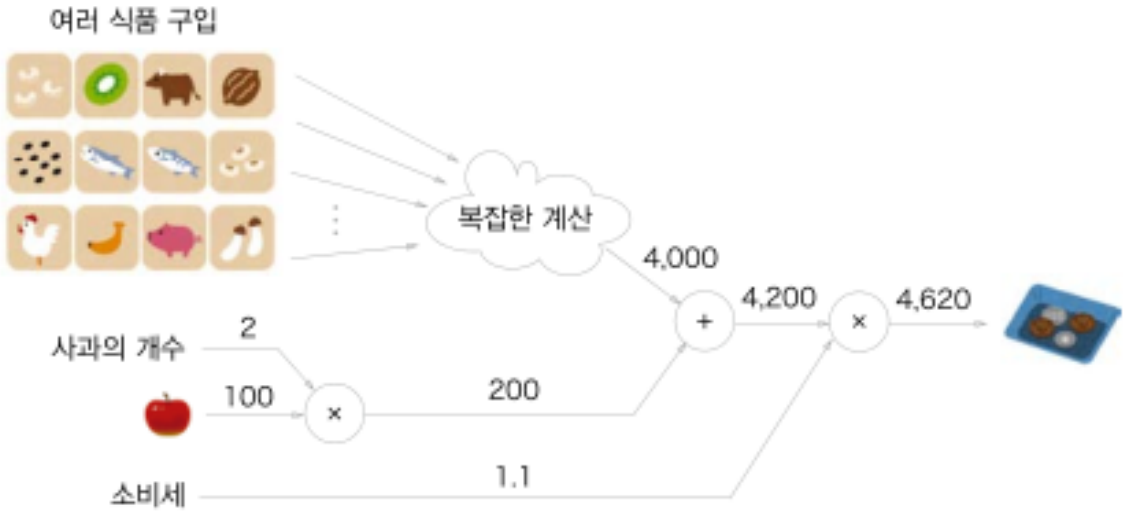


여기서 2 번째 '계산을 왼쪽에서 오른쪽으로 진행'하는 단계를 순전파 forward propagation 라고 한다.
순전파는 계산 그래프의 출발점부터 종착점으로의 전파이다.
순전파라는 이름이 있다면 반대 방향(그림에서 말하면 오른쪽에서 왼쪽)의 전파도 가능한데 그것을 역전파 backwardpropagation 라고 한다

SECTION 05 오차역전파법

5.1.2 국소적 계산

계산 그래프의 특징은 **국소적 계산**을 전파함으로써 최종 결과를 얻는다는 점에 있다.
국소적이란 '자신과 직접 관계된 작은 범위' 라는 뜻이다.



5.1.3 왜 계산 그래프로 푸는가?

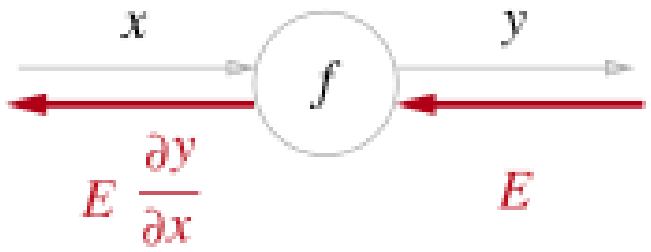
역전파를 통해 미분을 효율적으로 계산할 수 있다.

SECTION 05 오차역전파법

5.2 연쇄법칙(chain rule)

5.2.1 계산 그래프의 역전파

그림 5-6 계산 그래프의 역전파 : 순방향과는 반대 방향으로 국소적 미분을 곱한다.



5.2.2 연쇄법칙이란?

연쇄법칙을 설명하려면 우선 합성 함수 이야기부터 시작해야 한다. 합성 함수란 여러 함수로구성된 함수다

$$z = t^2$$
$$t = x + y \qquad \text{[식 5.1]}$$

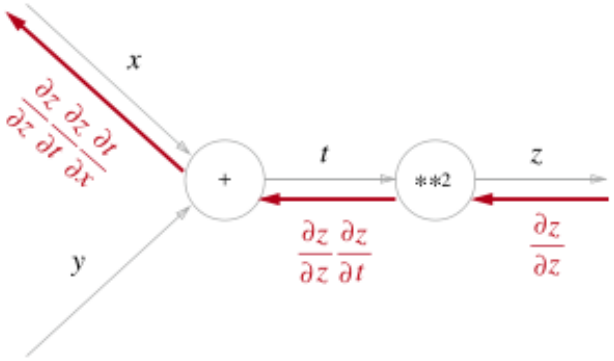
합성 함수의 미분은 합성 함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다.

을 예로 설명하면, $\frac{\partial z}{\partial x}$ (x에 대한 z의 미분)은 $\frac{\partial z}{\partial t}$ (t에 대한 z의 미분)과 $\frac{\partial t}{\partial x}$ (x에 대한 t의 미분)의 곱으로 나타낼 수 있다는 것이죠. 수식으로는 [식 5.2]처럼 쓸 수 있습니다.

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} \qquad \text{[식 5.2]}$$

5.2.3 연쇄법칙과 계산 그래프

2 제곱 계산을 ' ** 2 ' 노드로 나타내면 [그림 5 - 7]처럼 그릴 수 있다.



$$z = t^2$$
$$t = x + y \qquad \text{[식 5.1]}$$

그림 5-7 [식 5 . 4]의 계산 그래프 : 순전파와는 반대 방향으로 국소적 미분을 곱하여 전달한다.

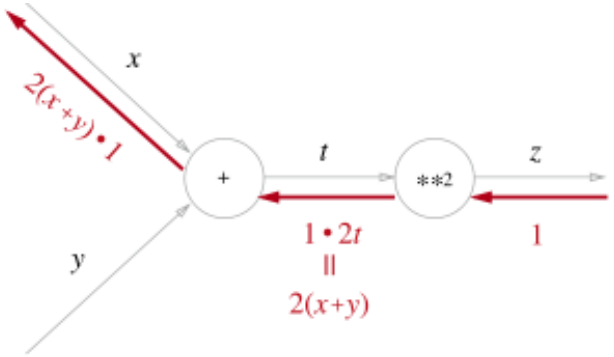


그림 5-8 계산 그래프의 역전파 결과에 따르면 는 2 (x + y)가 된다.

SECTION 05 오차역전파법

5.3 역전파

5.3.1 덧셈 노드의 역전파

$$z = x + y \qquad \frac{\partial z}{\partial x} = 1 \qquad \frac{\partial z}{\partial y} = 1 \qquad \text{[식 5.5]}$$

이를 계산 그래프로는 [그림 5-9]처럼 그릴 수 있다.

그림 5-9 덧셈 노드의 역전파 : 왼쪽이 순전파, 오른쪽이 역전파다. 덧셈 노드의 역전파는 입력 값을 그대로 흘려보낸다

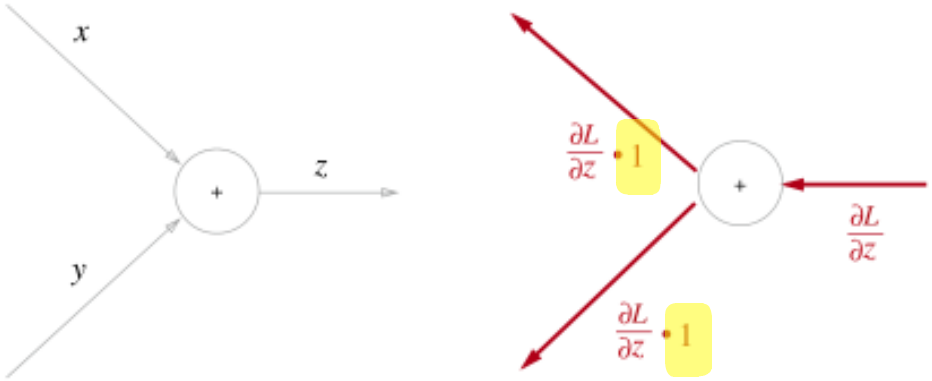
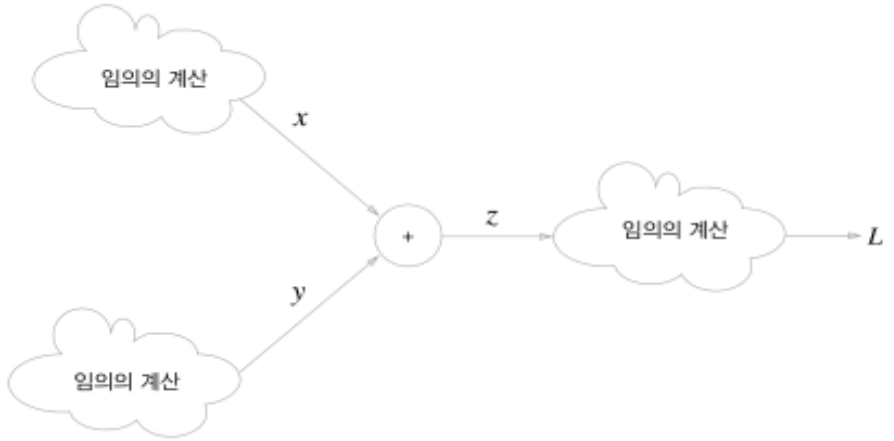


그림 5-10 최종 출력으로 가는 계산의 중간에 덧셈 노드가 존재한다. 역전파에서는 국소적 미분이 가장 오른쪽의 출력에서 시작하여 노드를 타고 역방향(왼쪽)으로 전파된다



5.3.2 곱셈 노드의 역전파

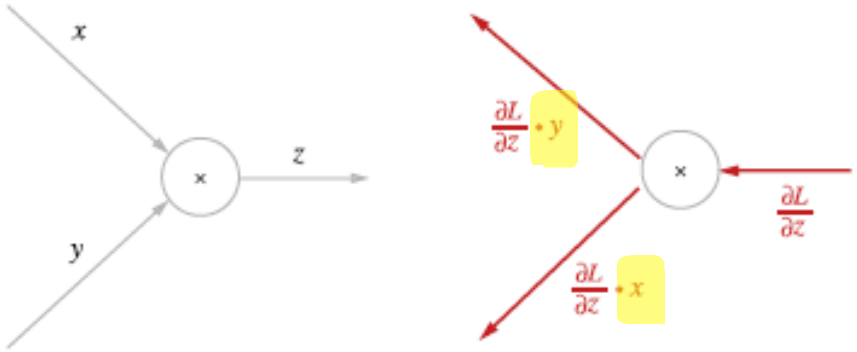
$$z = xy$$

$$\frac{\partial z}{\partial x} = y$$

$$\frac{\partial z}{\partial y} = x$$

[식 5.6]

[식 5.6]에서 계산 그래프는 다음과 같이 그릴 수 있다.



SECTION 05 오차역전파법

5.3.3 사과 쇼핑의 예

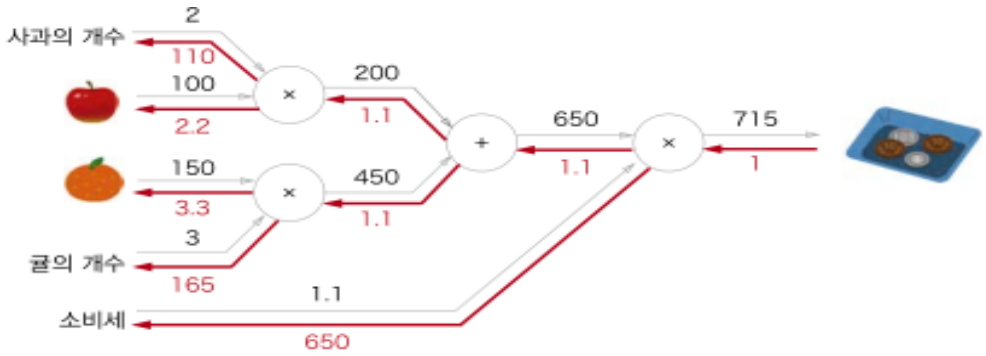
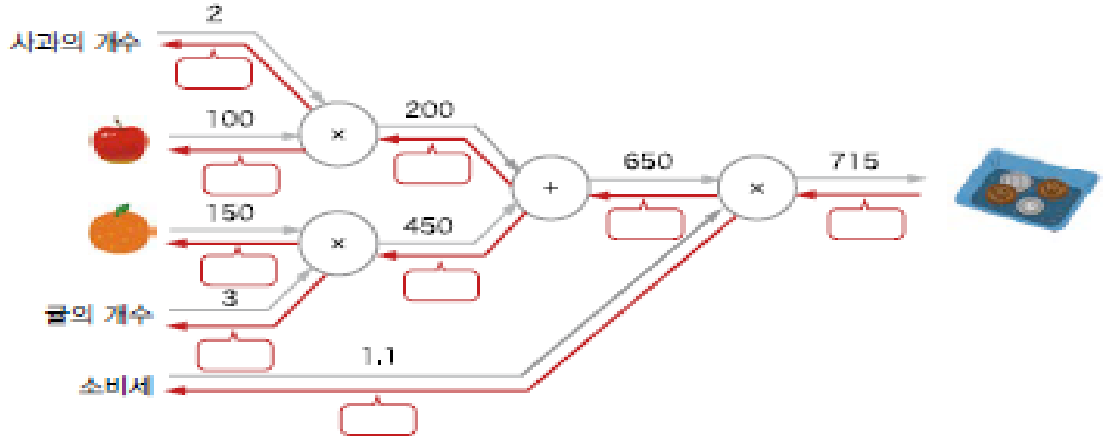


그림 5-17 사과 2 개와 귤 3 개 구입

지금까지 설명한 바와 같이 곱셈 노드의 역전파에서는 입력 신호를 서로 바꿔서 하류로 흘린다.
[그림 5-14]의 결과를 보면 사과 가격의 미분은 2.2, 사과 개수의 미분은 110, 소비세의미분은 200이다.
이는 소비세와 사과 가격이 같은 양만큼 오르면 최종 금액에는 소비세가200의 크기로, 사과 가격이 2.2 크기로 영향을 준다고 해석할 수 있다.

그림 5-15 사과와 귤 쇼핑의 역전파 예 : 빈 상자 안에 적절한 숫자를 넣어 역전파를 완성하시오.



SECTION 05 오차역전파법

5.4 단순한 계층 구현하기 5.4.1 곱셈 계층

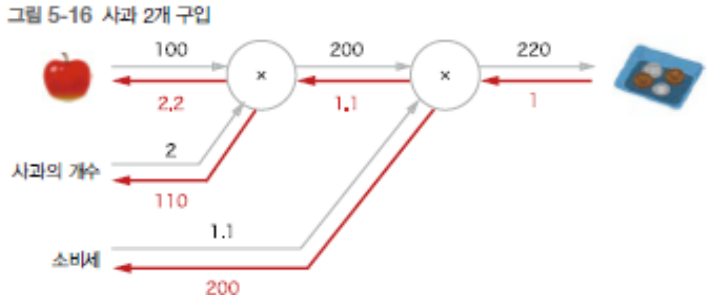
ch05/layer_naive.py

```
class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x, y):
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout):
        dx = dout * self.y # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

__init__( )에서는 인스턴스 변수인 x와 y를 초기화합니다. 이 두 변수는 순전파 시의 입력 값을 유지하기 위해서 사용한다.

Forward( )에서는 x와 y를 인수로 받고 두 값을 곱해서 반환한다.

반면 backward( )에서는 상류에서 넘어온 미분(dout)에 순전파 때의 값을 '서로 바꿔' 곱한 후 하류로 흘린다.



[그림 5-16]의 순전파를 다음과 같이 구현할 수 있다

ch05/buy_apple.py

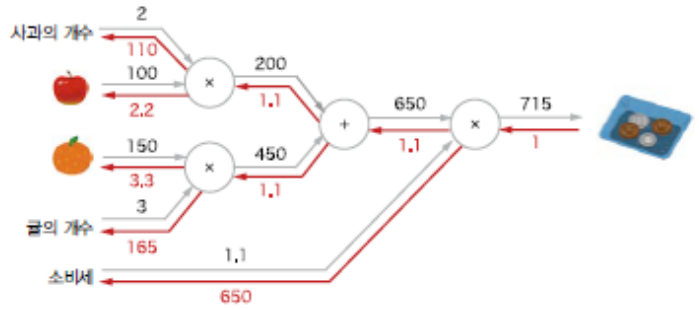
```
from layer_naive import *

apple = 100
apple_num = 2
tax = 1.1

mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

# backward
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dTax:", dtax)
```

5.4.2 덧셈 계층

ch05/layer_naive.py

```
class AddLayer:
    def __init__(self):
        pass

    def forward(self, x, y):
        out = x + y
        return out

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```

그림 5-17 사과 2개와 귤 3개 구입



[그림 5-17]의 계산 그래프를 파이썬으로 구현하면 다음과 같다

ch05/buy_apple_orange.py

```
from layer_naive import *

apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# layer
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

# forward
apple_price = mul_apple_layer.forward(apple, apple_num)  # (1)
orange_price = mul_orange_layer.forward(orange, orange_num)  # (2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price)  # (3)
price = mul_tax_layer.forward(all_price, tax)  # (4)

# backward
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice)  # (4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price)  # (3)
dorange, dorange_num = mul_orange_layer.backward(dorange_price)  # (2)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)  # (1)

print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dOrange:", dorange)
print("dOrange_num:", int(dorange_num))
print("dTax:", dtax)
```

SECTION 05 오차역전파법

5.5 활성화 함수 계층 구현하기

5.5.1 ReLU 계층

활성화 함수로 사용되는 ReLU의 수식은 다음과같다.

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \le 0) \end{cases} \qquad [식 5.7]$$

[식 5.7]에서 x에 대한 y의 미분은 [식 5.8]처럼 구한다

$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \le 0) \end{cases} \qquad [식 5.8]$$

그림 5-18 ReLU 계층의 계산 그래프

common/layers.py

```
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out

    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

Relu 클래스는 mask라는 인스턴스 변수를 가진다. mask는 True/False로 구성된 넘파이 배열로, 순전파의 입력인 x의 원소 값이 0 이하인 인덱스는 True, 그 외(0보다 큰 원소)는False로 유지한다

5.5.2 Sigmoid 계층

$$y = \frac{1}{1+\exp(-x)} \qquad [식 5.9]$$

[식 5.9]를 계산 그래프로 그리면 [그림 5-19]처럼 된다

그림 5-19 Sigmoid 계층의 계산 그래프(순전파)

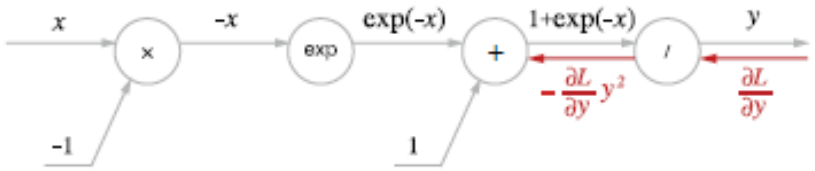
SECTION 05 오차역전파법

5.5.2 Sigmoid 계층

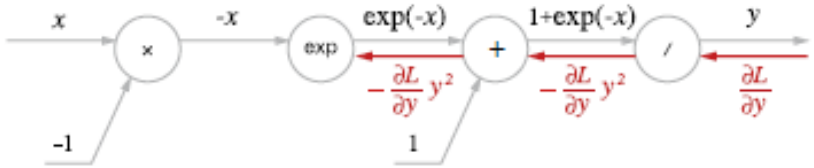
1 단계

'/' 노드, 즉 $y = \frac{1}{x}$ 을 미분하면 다음 식이 됩니다.

$$\frac{\partial y}{\partial x} = \frac{1}{x^2} = -y^2 \quad \text{[식 5.10]}$$

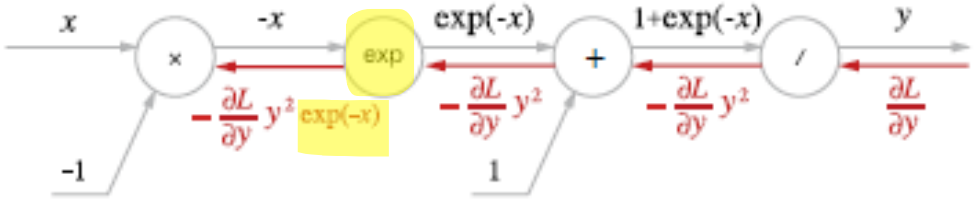


2 단계



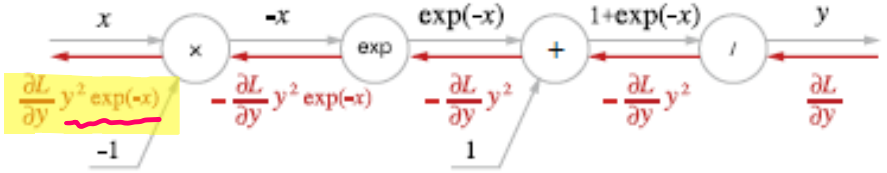
3 단계

$$\frac{\partial y}{\partial x} = \exp(x) \quad \text{[식 5.11]}$$



4 단계

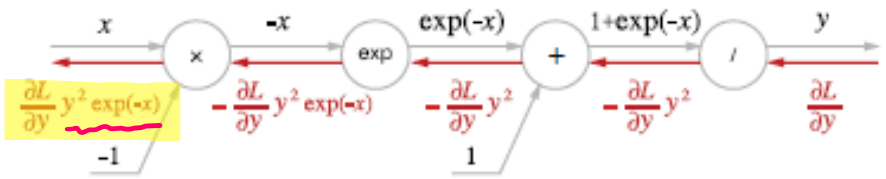
그림 5-20 Sigmoid 계층의 계산 그래프



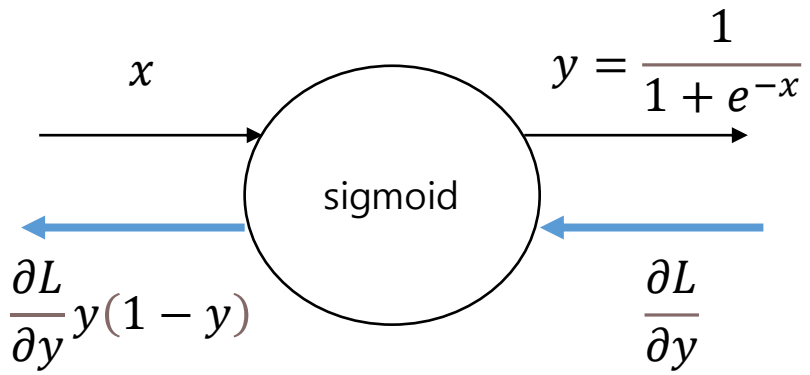
5.5.2 Sigmoid 계층

4 단계

그림 5-20 Sigmoid 계층의 계산 그래프



$$y^2 \exp(-x) = \frac{1}{1+\mathrm{e}^{-x}} \cdot \frac{\mathrm{e}^{-x}}{1+\mathrm{e}^{-x}} = y(1-y)$$



common/layers.py

```
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

SECTION 05 오차역전파법

5.6 Affine/Softmax 계층 구현하기

5.6.1 Affine 계층

```
>>> X = np.random.rand(2)     # 입력
>>> W = np.random.rand(2,3)   # 가중치
>>> B = np.random.rand(3)     # 편향
>>>
>>> X.shape # (2,)
>>> W.shape # (2, 3)
>>> B.shape # (3,)
>>>
>>> Y = np.dot(X, W) + B
```

그림 5-23 행렬의 곱에서는 대응하는 차원의 원소 수를 일치시킨다.

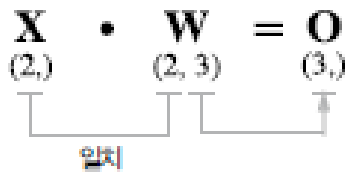
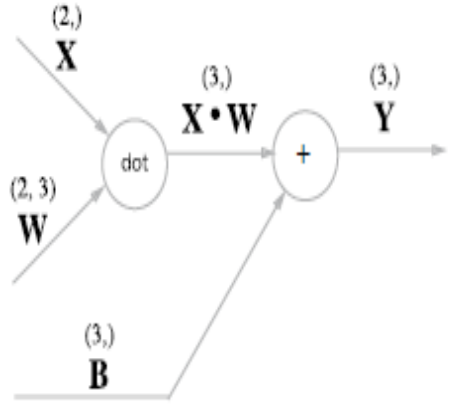


그림 5-24 Affine 계층의 계산 그래프 : 변수가 행렬임에 주의. 각 변수의 형상을 변수명 위에 표기했다.



$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$$\mathbf{W}^{\mathrm{T}} = \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix} \quad [식\ 5.14]$$

$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n} \right) \quad [식\ 5.15]$$

5.6.1 Affine 계층

$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n}\right) \quad [식\ 5.15]$$

$Y = X \cdot W$

$$(y_1\ y_2\ y_3) = (x_1\ x_2)\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

$y_1 = x_1 w_{11} + x_2 w_{21}$

$y_2 = x_1 w_{12} + x_2 w_{22}$

$y_3 = x_1 w_{13} + x_2 w_{23}$

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial x_1} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial x_1} = \frac{\partial L}{\partial y_1} w_{11} + \frac{\partial L}{\partial y_2} w_{12} + \frac{\partial L}{\partial y_3} w_{13}$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial y_1} w_{21} + \frac{\partial L}{\partial y_2} w_{22} + \frac{\partial L}{\partial y_3} w_{23}$$

$$\Rightarrow \frac{\partial L}{\partial X} = \begin{pmatrix} \frac{\partial L}{\partial x_1} & \frac{\partial L}{\partial x_2} \end{pmatrix} = \begin{pmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{pmatrix} \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix} = \frac{\partial L}{\partial Y} \cdot W^T$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial y_1}\frac{\partial y_1}{\partial w_{11}} + \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial w_{11}} + \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial w_{11}} = \frac{\partial L}{\partial y_1} x_1$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial y_2}\frac{\partial y_2}{\partial w_{12}} = \frac{\partial L}{\partial y_2} x_1 \qquad \frac{\partial L}{\partial w_{13}} = \frac{\partial L}{\partial y_3}\frac{\partial y_3}{\partial w_{13}} = \frac{\partial L}{\partial y_3} x_1$$

$$\frac{\partial L}{\partial w_{21}} = \frac{\partial L}{\partial y_1} x_2 \qquad \frac{\partial L}{\partial w_{22}} = \frac{\partial L}{\partial y_2} x_2 \qquad \frac{\partial L}{\partial w_{23}} = \frac{\partial L}{\partial y_3} x_2$$

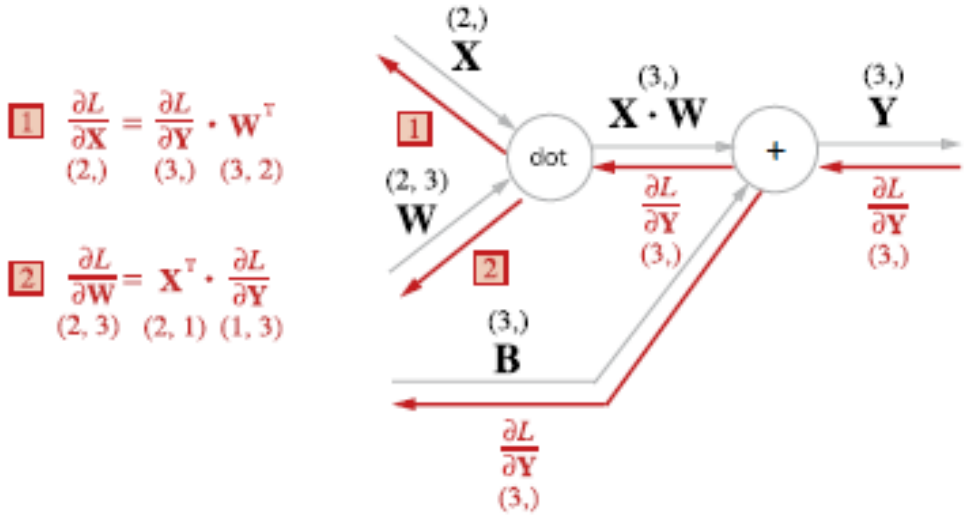
$$\Rightarrow \frac{\partial L}{\partial W} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \begin{pmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{pmatrix} = X^T \frac{\partial L}{\partial Y}$$

(cf. $Y = XW \rightarrow X = YW^+, \ W = X^+ Y$

$+$: pseudo-inverse)

5.6.1 Affine 계층

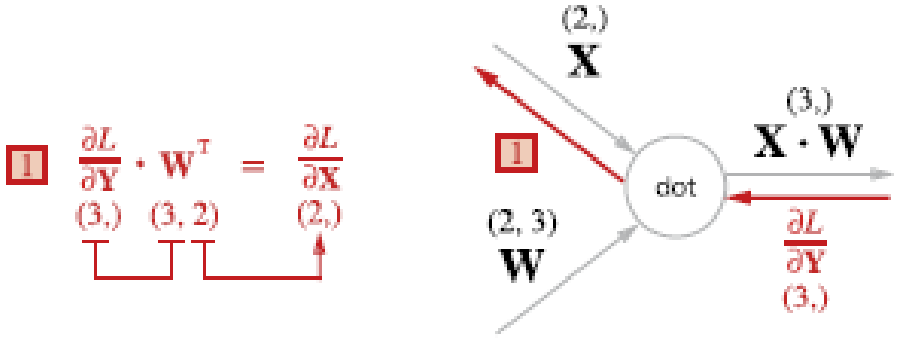
그림 5-25 Affine 계층의 역전파 : 변수가 다차원 배열임에 주의. 역전파에서의 변수 형상은 해당 변수명 아래에 표기했다.



$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

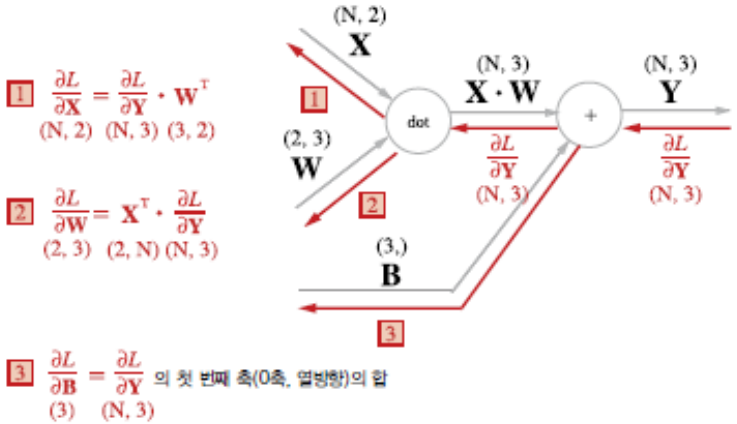
$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n}\right) \quad \text{[식 5.15]}$$

그림 5-26 행렬 곱('dot' 노드)의 역전파는 행렬의 대응하는 차원의 원소 수가 일치하도록 곱을 조립하여 구할 수 있다.



5.6.2 배치용 Affine 계층

그림 5-27 배치용 Affine 계층의 계산 그래프



$$1\ \frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathsf{T}} \quad (N, 2)\ (N, 3)\ (3, 2)$$

$$2\ \frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathsf{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}} \quad (2, 3)\ (2, N)\ (N, 3)$$

$$3\ \frac{\partial L}{\partial \mathbf{B}} = \frac{\partial L}{\partial \mathbf{Y}} \text{ 의 첫 번째 축(0축, 열방향)의 합} \quad (3)\ (N, 3)$$

```
>>> X_dot_W = np.array([[0, 0, 0], [10, 10, 10]])
>>> B = np.array([1, 2, 3])
>>>
>>> X_dot_W
array([[ 0,  0,  0],
       [ 10, 10, 10]])
>>> X_dot_W + B

array([[ 1,  2,  3],
       [11, 12, 13]])
```

순전파의 편향 덧셈은 각각의 데이터(1번째 데이터, 2번째 데이터, ...)에 더해진다.
역전파 때는 각 데이터의 역전파 값이 편향의 원소에 모여야 한다.

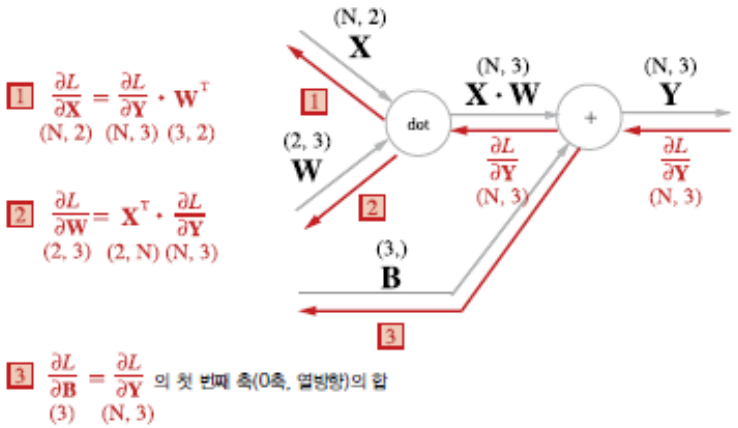
코드

```
>>> dY = np.array([[1, 2, 3], [4, 5, 6]])
>>> dY
array([[1, 2, 3],
       [4, 5, 6]])
>>>
>>> dB = np.sum(dY, axis=0)
>>> dB
array([5, 7, 9])
```

SECTION 05 오차역전파법

5.6.2 배치용 Affine 계층

그림 5-27 배치용 Affine 계층의 계산 그래프



1 $\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$
(N, 2) (N, 3) (3, 2)

2 $\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$
(2, 3) (2, N) (N, 3)

3 $\frac{\partial L}{\partial \mathbf{B}} = \frac{\partial L}{\partial \mathbf{Y}}$ 의 첫 번째 축(0축, 열방향)의 합
(3) (N, 3)

```python
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None
        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):
        # 텐서 대응
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)  # 입력 데이터 모양 변경(텐서 대응)
        return dx
```

5.6.3 Softmax-with-Loss 계층

소프트맥스 함수는 입력 값을 정규화하여 출력

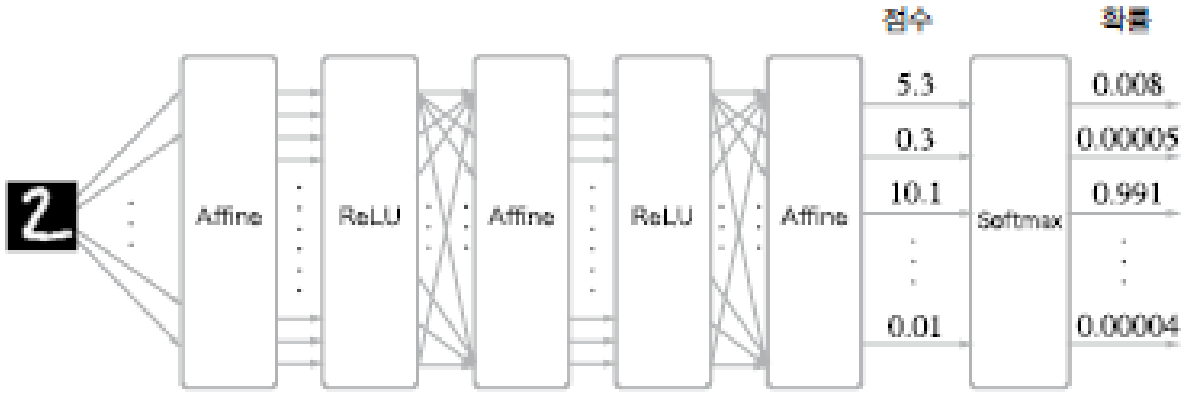
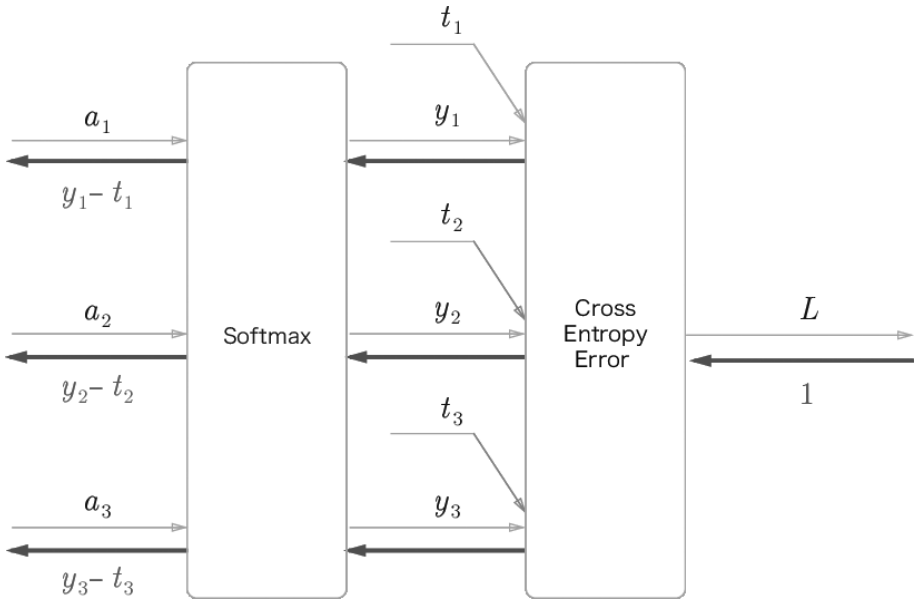


그림 **5-30** '간소화한' Softmax-with-Loss 계층의 계산 그래프



SECTION 05 오차역전파법

A.1 Softmax-with-Loss 계층: 순전파

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

$$L = -\sum_k t_k \log y_k$$

그림 A-2 Softmax 계층의 계산 그래프(순전파만)

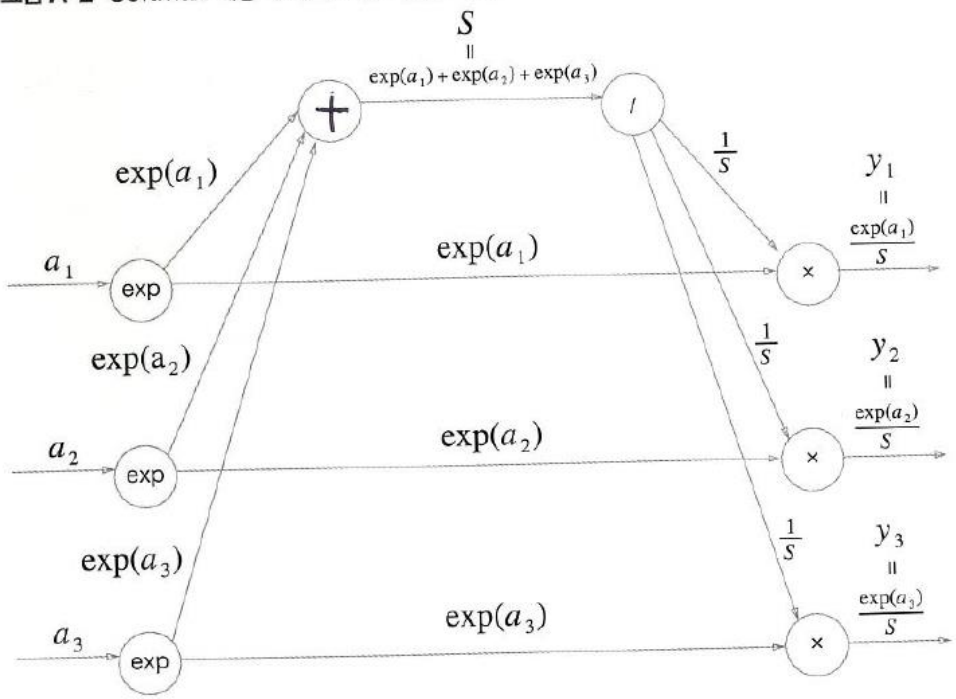
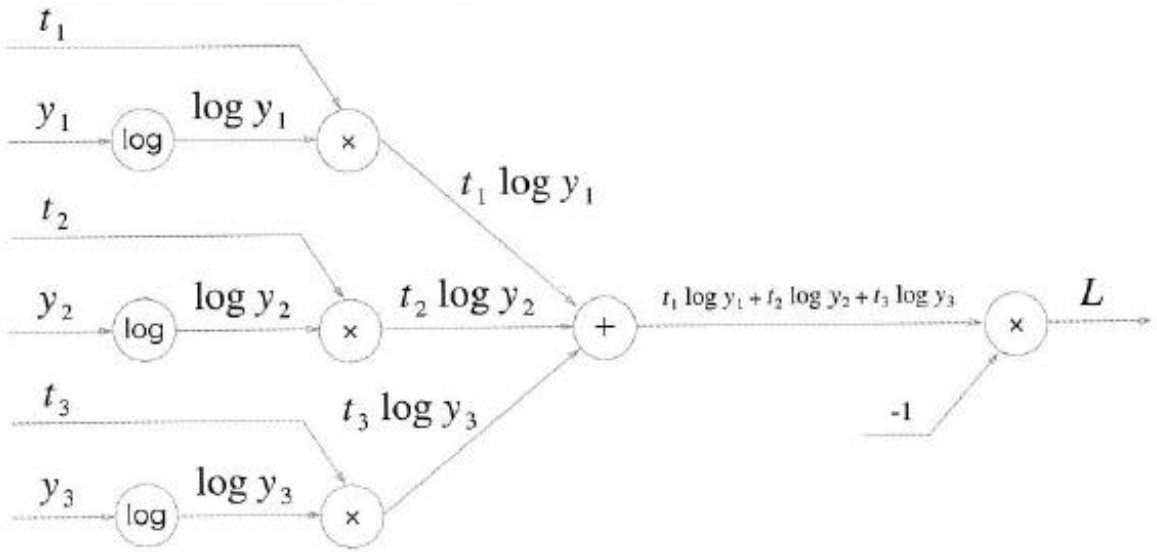


그림 A-3 Cross Entropy Error 계층의 계산 그래프(순전파만)

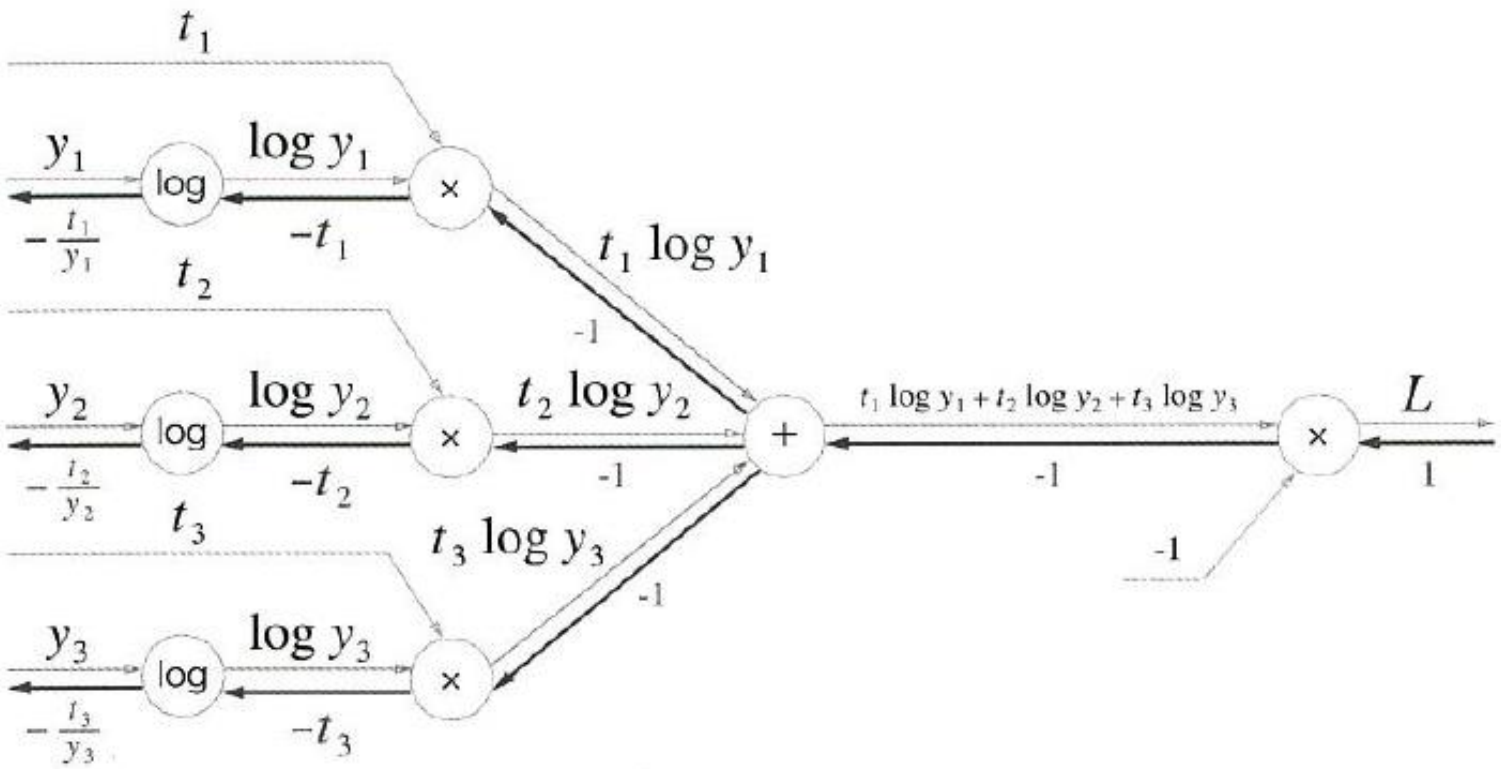


SECTION 05 오차역전파법

A.2 Softmax-with-Loss 계층: 역전파

- 역전파의 초깃값. 즉 [그림 A-4]의 가장 오른쪽 역전파의 값은 1입니다($\frac{\partial L}{\partial L}=1$이므로).
- '×' 노드의 역전파는 순전파 시의 입력들의 값을 '서로 바꿔'* 상류의 미분에 곱하고 하류로 흘립니다.
- '+' 노드에서는 상류에서 전해지는 미분을 그대로 흘립니다.

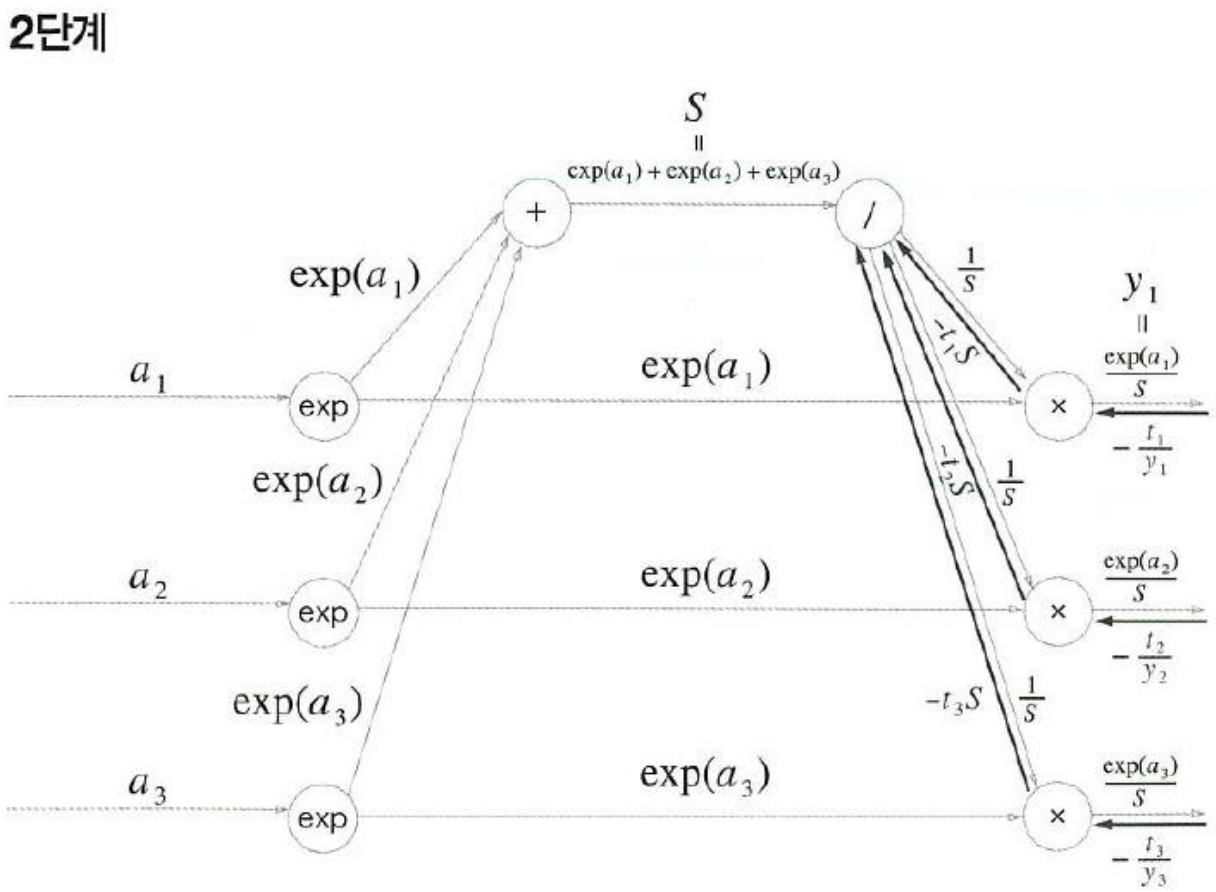
그림 A-4 Cross Entropy Error 계층의 역전파



$$y = \log x$$

$$\frac{\partial y}{\partial x} = \frac{1}{x}$$

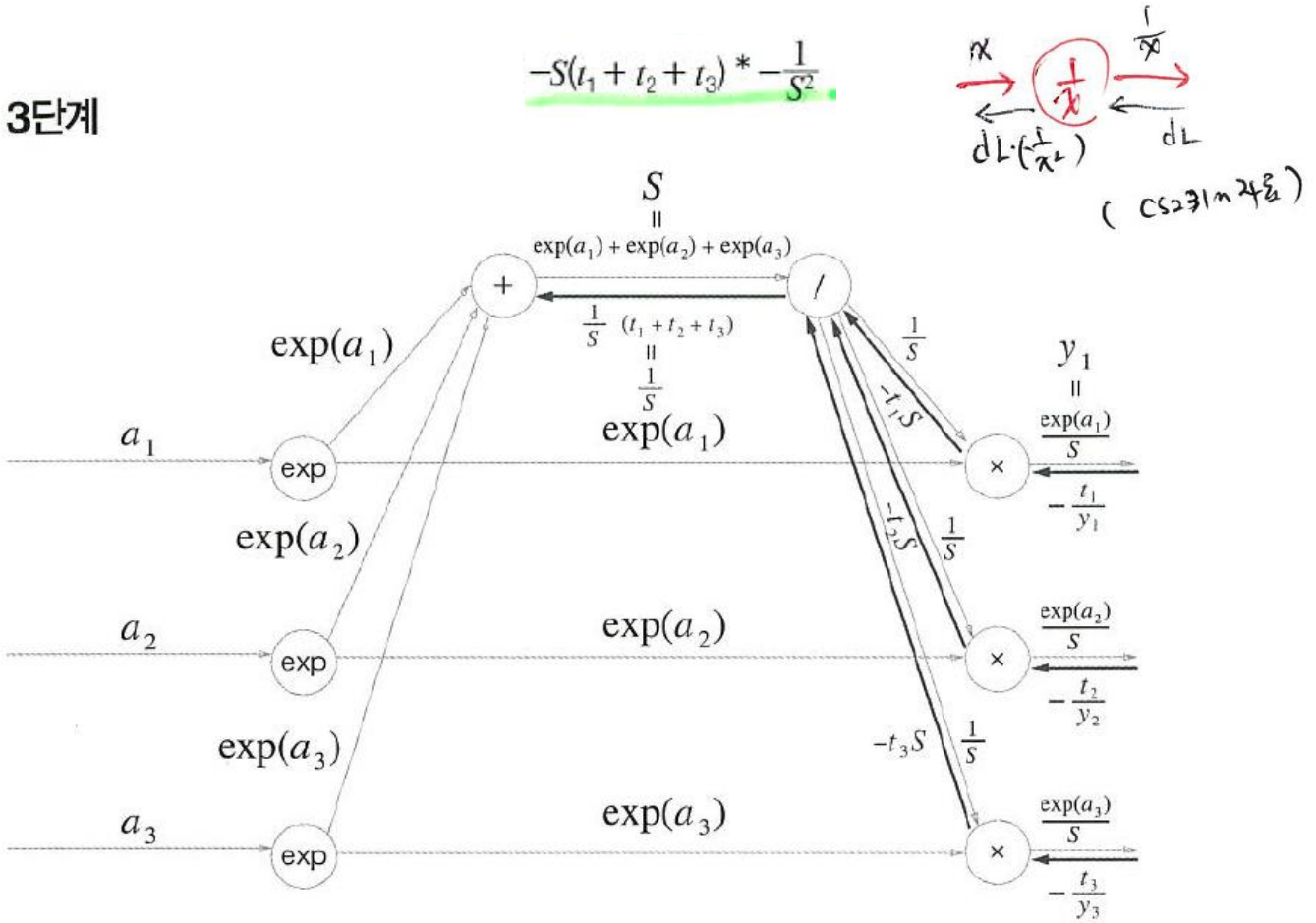
A.2 Softmax-with-Loss 계층: 역전파



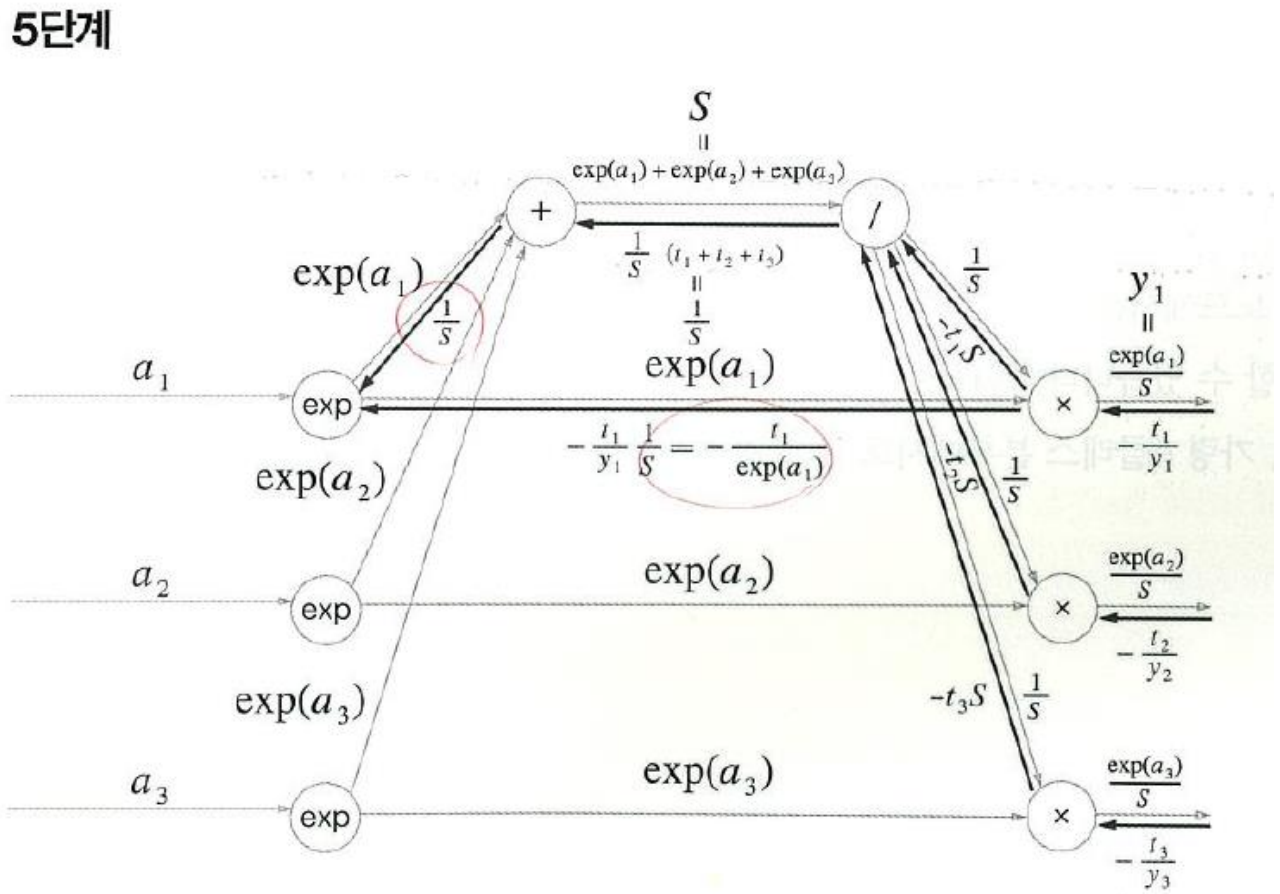
$$-\frac{t_1}{y_1}\exp(a_1) = -t_1\frac{S}{\exp(a_1)}\exp(a_1) = -t_1S$$

SECTION 05 오차역전파법

A.2 Softmax-with-Loss 계층: 역전파



A.2 Softmax-with-Loss 계층: 역전파



5.6.3 Softmax-with-Loss 계층

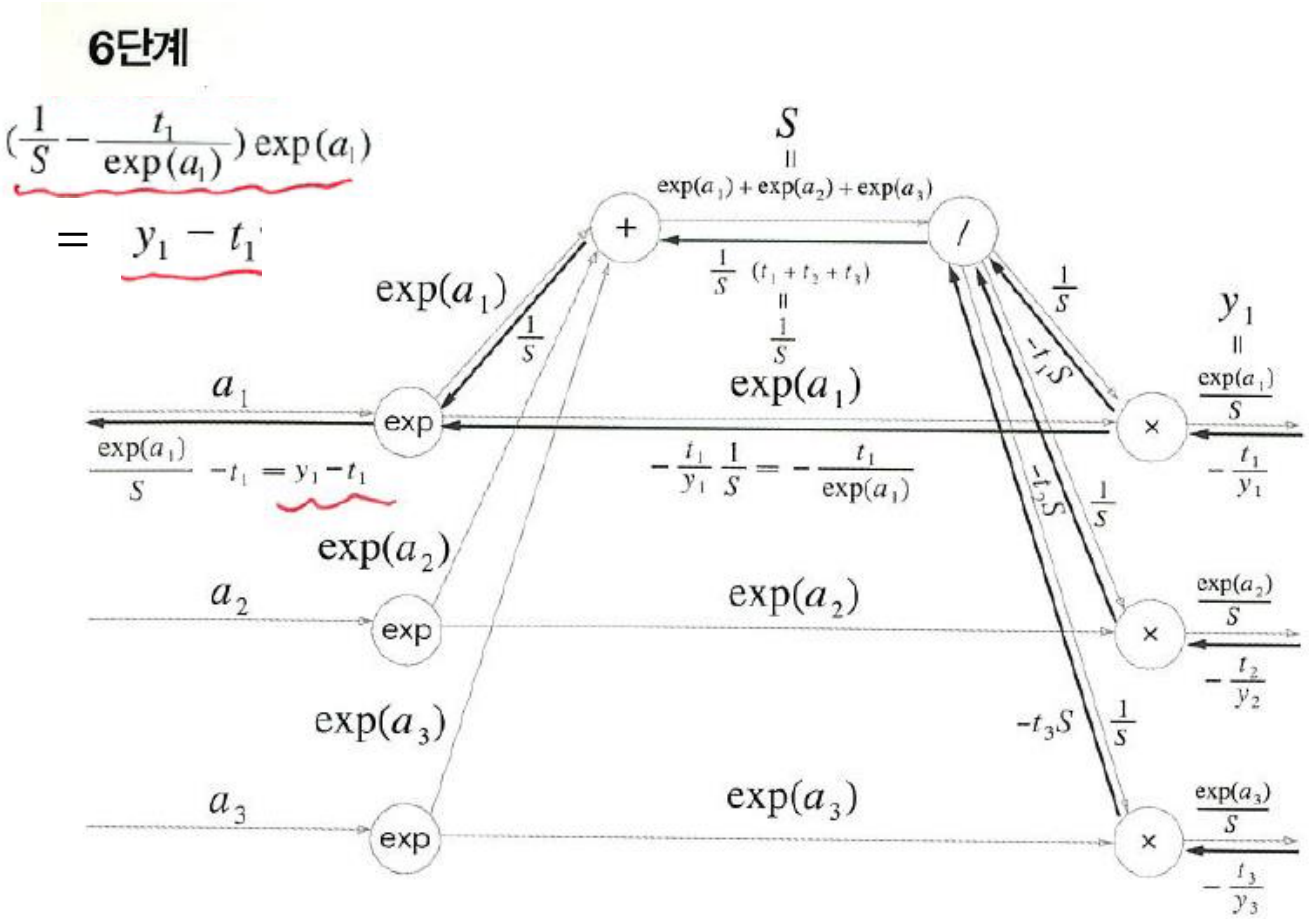
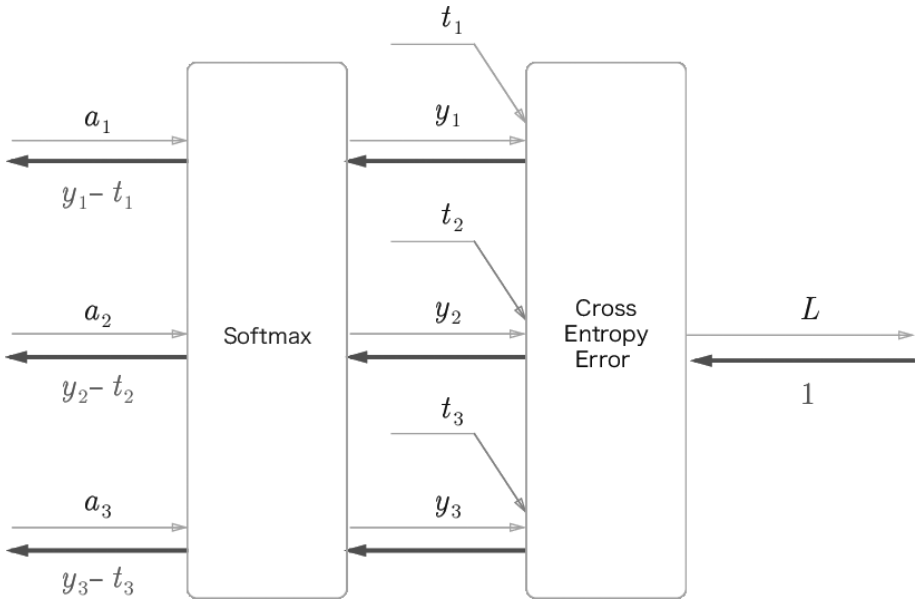
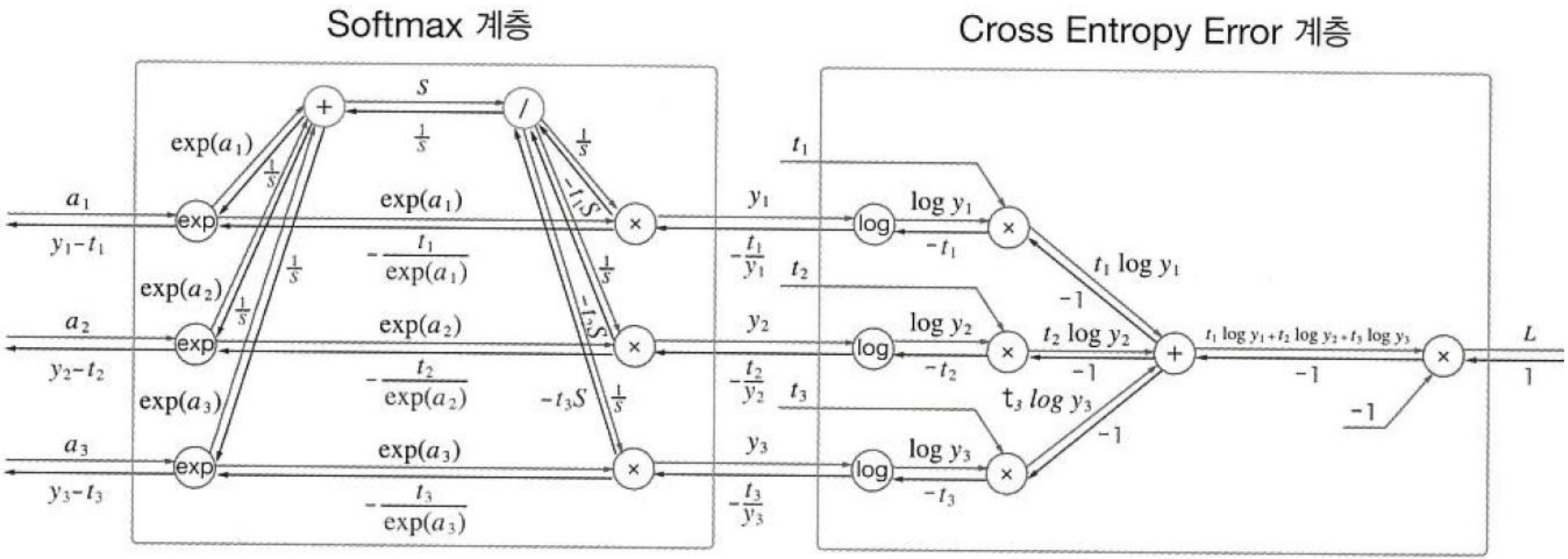


그림 **5-30** '간소화한' Softmax-with-Loss 계층의 계산 그래프



5.6.3 Softmax-with-Loss 계층

그림 A-5 Softmax-with-Loss 계층의 계산 그래프



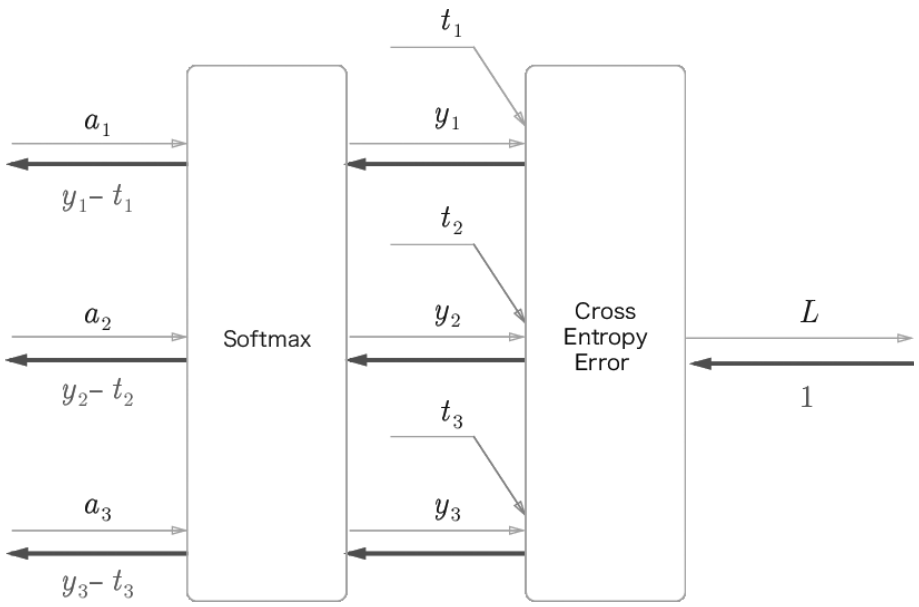
SECTION 05 오차역전파법

5.6.3 Softmax-with-Loss 계층

```python
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None # 손실함수
        self.y = None    # softmax의 출력
        self.t = None    # 정답 레이블(원-핫 인코딩 형태)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때
            dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

common/layers.py

그림 **5-30** '간소화한' Softmax-with-Loss 계층의 계산 그래프



5.7 오차 역전파법 구현하기

5.7.1 신경망 학습의 전체 그림

전제

신경망에는 적응 가능한 가중치와 편향이 있고, 이 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 한다.
신경망 학습은 다음과 같이 4단계로 수행한다.

1단계 - 미니배치
훈련 데이터 중 일부를 무작위로 가져온다. 이렇게 선별한 데이터를 미니배치라 하며, 그 미니배치의 손실함수 값을 줄이는 것이 목표

2단계 - 기울기 산출
미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구한다.
기울기는 손실 함수의 값을 가장 작게 하는 방향을 제시.

3단계 - 매개변수 갱신
가중치 매개변수를 기울기 방향으로 아주 조금 갱신.

4단계 - 반복
1~3단계를 반복.

5.7.2 오차역전파법을 적용한 신경망 구현하기

표 5-1 TwoLayerNet 클래스의 인스턴스 변수

인스턴스 변수	설명
params	딕셔너리 변수로, 신경망의 매개변수를 보관 params['W1']은 1번째 층의 가중치, params['b1']은 1번째 층의 편향 params['W2']는 2번째 층의 가중치, params['b2']는 2번째 층의 편향
layers	순서가 있는 딕셔너리 변수로, 신경망의 계층을 보관 layers['Affine1'], layers['Relu1'], layers['Affine2']와 같이 각 계층을 순서대로 유지
lastLayer	신경망의 마지막 계층 이 예에서는 SoftmaxWithLoss 계층

표 5-2 TwoLayerNet 클래스의 메서드

메서드	설명
__init__(self, input_size, hidden_size, output_size, weight_init_std)	초기화를 수행한다. 인수는 앞에서부터 입력층 뉴런 수, 은닉층 뉴런 수, 출력층 뉴런 수, 가중치 초기화 시 정규분포의 스케일
predict(self, x)	예측(추론)을 수행한다. 인수 x는 이미지 데이터
loss(self, x, t)	손실 함수의 값을 구한다. 인수 x는 이미지 데이터, t는 정답 레이블
accuracy(self, x, t)	정확도를 구한다.
numerical_gradient(self, x, t)	가중치 매개변수의 기울기를 수치 미분 방식으로 구한다(앞 장과 같음).
gradient(self, x, t)	가중치 매개변수의 기울기를 오차역전파법으로 구한다.

5.7.2 오차역전파법을 적용한 신경망 구현하기

계층을 사용함으로써 인식 결과를 얻는 처리(predict ( ))와 기울기를 구하는 처리(gradient ( )) 계층의 전파으로 동작이 이루어지는 것다.그럼 코드를 살펴보자

ch05/two_layer_net.py

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict


class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size, weight_init_std = 0.01):
        # 가중치 초기화
        self.params = {}
        self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)

        # 계층 생성
        self.layers = OrderedDict()
        self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
        self.layers['Relu1'] = Relu()
        self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])

        self.lastLayer = SoftmaxWithLoss()

    def predict(self, x):
        for layer in self.layers.values():
            x = layer.forward(x)

        return x
```

```python
    # x : 입력 데이터, t : 정답 레이블
    def loss(self, x, t):
        y = self.predict(x)
        return self.lastLayer.forward(y, t)

    def accuracy(self, x, t):
        y = self.predict(x)
        y = np.argmax(y, axis=1)
        if t.ndim != 1 : t = np.argmax(t, axis=1)

        accuracy = np.sum(y == t) / float(x.shape[0])
        return accuracy

    # x : 입력 데이터, t : 정답 레이블
    def numerical_gradient(self, x, t):
        loss_W = lambda W: self.loss(x, t)

        grads = {}
        grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
        grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
        grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
        grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

        return grads

    def gradient(self, x, t):
        # forward
        self.loss(x, t)

        # backward
        dout = 1
        dout = self.lastLayer.backward(dout)

        layers = list(self.layers.values())
        layers.reverse()
        for layer in layers:
            dout = layer.backward(dout)

        # 결과 저장
        grads = {}
        grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

        return grads
```

5.7.3 오차역전파법으로 구한 기울기 검증하기

두 방식으로 구한 기울기가 일치함(엄밀히 말하면 거의 같음)을 확인하는 작업을 기울기 확인gradient check이라고 한다.

```
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

x_batch = x_train[:3]
t_batch = t_train[:3]

grad_numerical = network.numerical_gradient(x_batch, t_batch)
grad_backprop = network.gradient(x_batch, t_batch)

# 각 가중치의 절대 오차의 평균을 구한다.
for key in grad_numerical.keys():
    diff = np.average( np.abs(grad_backprop[key] - grad_numerical[key]) )
    print(key + ":" + str(diff))
```

코드의 실행 결과는?

5.7.4 오차역전파법을 사용한 학습 구현하기

```
import sys, os
sys.path.append(os.pardir)

import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

# 데이터 읽기
(x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1

train_loss_list = []
train_acc_list = []
test_acc_list = []
iter_num=[]

iter_per_epoch = max(train_size / batch_size, 1)
```

5.7.4 오차역전파법을 사용한 학습 구현하기

```
for i in range(iters_num):
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 기울기 계산
    #grad = network.numerical_gradient(x_batch, t_batch) # 수치 미분 방식
    grad = network.gradient(x_batch, t_batch) # 오차역전파법 방식(훨씬 빠르다)

    # 갱신
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        iter_num.append(i)
        print(i,train_acc, test_acc)

import matplotlib.pyplot as plt
plt.plot(iter_num,train_acc_list, label="train")
plt.plot(iter_num,test_acc_list, label="test")
plt.legend()
plt.show()
```

4.9.1 sklearn의 필기 숫자 데이터셋

■ [프로그램 4-3]

프로그램 4-3 sklearn 필기 숫자 데이터에 다층 퍼셉트론 적용

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

4.9.1 sklearn의 필기 숫자 데이터셋

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

4.9.1 sklearn의 필기 숫자 데이터셋

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

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

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

테스트 집합에 대한 정확률은 97.2183588317107%입니다.
```

4.9.2 MNIST 데이터셋으로 확장하기

- [프로그램 4-4]
  - [프로그램 4-3]과 매우 유사

09~10행: 앞 6만 자를 훈련, 뒤 1만 자를 테스트로 분할

8행: [0,255] 범위를 [0,1] 범위로 변환

프로그램 4-4 MNIST 데이터셋을 다층 퍼셉트론으로 인식

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

4.9.2 MNIST 데이터셋으로 확장하기

- 실행 결과 97.78% 정확률을 얻음

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

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

테스트 집합에 대한 정확률은 97.78%입니다.
```

5.9 좋은 프로그래밍 스킬

■ 프로그래밍 스킬의 중요성

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

1. 모듈화하라.

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

2. 언어의 좋은 특성을 최대한 활용하라.

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

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

①

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

②

5.9 좋은 프로그래밍 스킬

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

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

5. 도구에 한없이 익숙해져라.
   - 통합개발환경인 스파이더 사용법에 익숙
   - 라이브러리 사용에 익숙

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