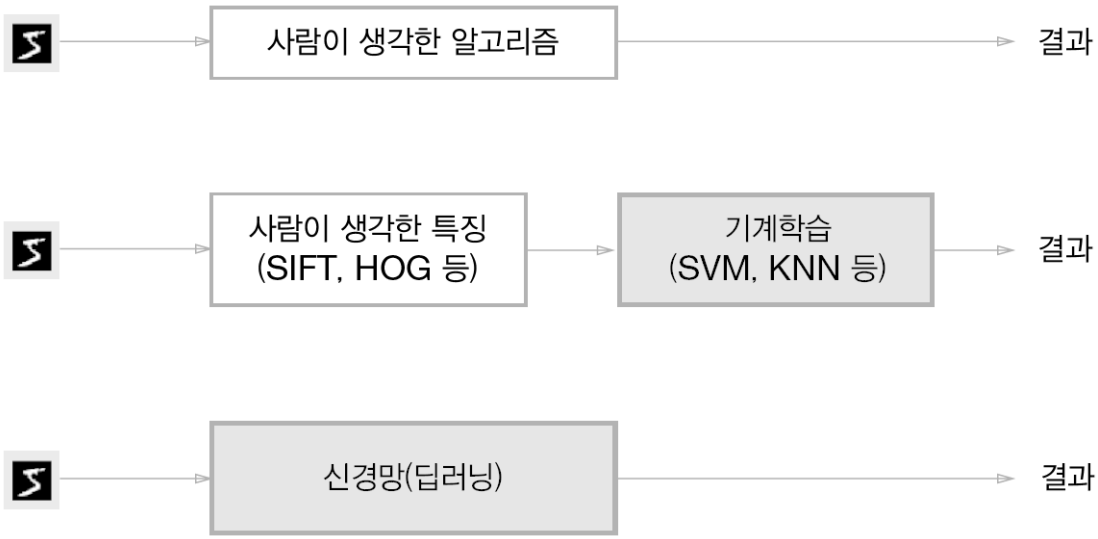
3월 23일

chapter 4 신경망 학습

4.1 데이터에서 학습한다.

4.1.1 데이터 주도 학습



딥러닝을 종단간 기계학습 (end-to-end machine learning) : 데이터에서 목표한 결과를 얻음

4.1.2 훈련 데이터와 시험 데이터

훈련데이터(training data)만 사용하여 매개변수를 찾음

시험데이터(test data)를 사용하여 훈련된 모델의 성능 평가

➔ 모델의 범용 능력을 평가하기 위해

➔ 범용 능력: 아직 보지 못한 데이터(unseen data: 훈련데이터에 포함되지 않은 데이터)로도 정확한 성능

➔ 예) 임의의 사람이 쓴 임의의 글자

오버피팅(overfitting): 과적합, 학습자료에만 지나치게 최적화

underfitting : 학습자료에서도 성능이 낮은 것, 모델 학습이 부족한 것

4.2 손실함수 (loss function)

현재의 신경망이 훈련 데이터에 대한 오류

손실함수를 기준으로 최적의 매개변수 탐색

➔ 평균 제곱 오차, 교차 엔트로피 오차

4.2.1 평균 제곱 오차 : mean squared error, MSE

$$E = \frac{1}{2}\sum_{k}(y_k - t_k)^2$$

y_k: 신경망 출력,

예) 소프트맥스 출력 [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]

t_k: 정답 레이블

예) one-hot 인코딩 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

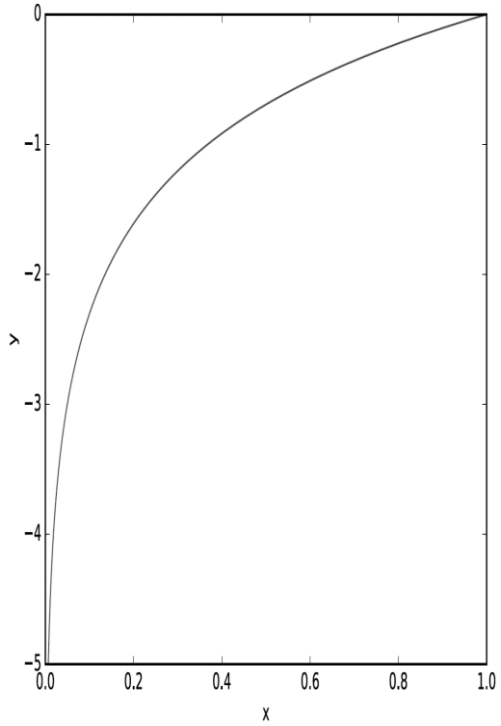
```
import numpy as np


t = np.array( [0, 0, 1, 0, 0, 0, 0, 0, 0, 0] )
y1=np.array( [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0] )
```

```
y2=np.array( [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0] )

def mse(y,t) :
    return 0.5 * np.sum( (y-t)**2 )

print (mse(y1, t))
print (mse(y2, t))
```

4.2.2 교차 엔트로피 오차 cross entropy error

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

로그함수 형태 [0, 1] 구간



$-\log(\ y_k\ )$는 y_k가 0에 가까울수록 무척 커짐: MSE와 같음

결론적으로 t, y가 [0, 1]사이, t가 1일 때 y가 1에 가까워야 작은 손실

```
def crossEnt (y,t) :
    return -np.sum( t*np.log(y+ 1e-7) )

print (crossEnt(y1, t))
print (crossEnt(y2, t))
```

4.2.3 미니배치 학습

학습이란 훈련데이터에 대한 손실함수의 값을 최소화하는 매개 변수를 탐색하는 과정

모든 자료에 대한 손실의 평균 : 평균을 사용해서 비교 일관성 유지

$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk}\log y_{nk}$$

N: 데이터 개수, t_{nk}: n번째 자료의 k번째 차원

미니배치: 전체 자료중 일부만 사용하여 학습 ch04/train_neuralnet.py

```
batch_mask = np.random.choice(train_size, batch_size)
# np.random.choice(100,5)
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]
```

4.2.4 배치용 교차 엔트로피 오차 구현하기

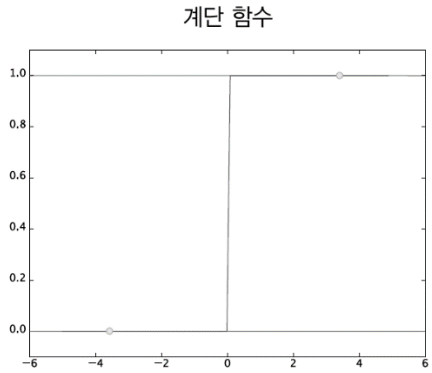
```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 훈련 데이터가 정답 레이블이 원-핫 벡터라면 인덱스로 반환
    if t.size == y.size:
```

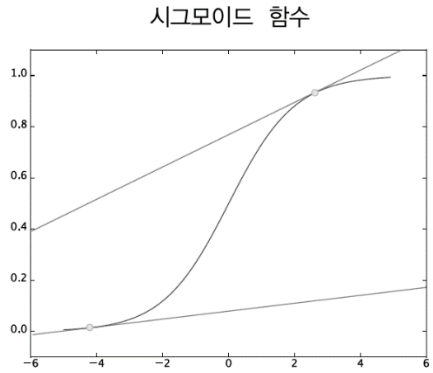
```
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log( y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

4.2.5 왜 손실 함수를 설정하는가?

정확도 대신 손실 함수를 왜?: 미분가능해서 매개변수 탐색에 이용

활성화함수의 경우에도





계단함수 미분 값 0 → 경사하강법으로 학습 안됨

4.3 수치미분

4.3.1 미분

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

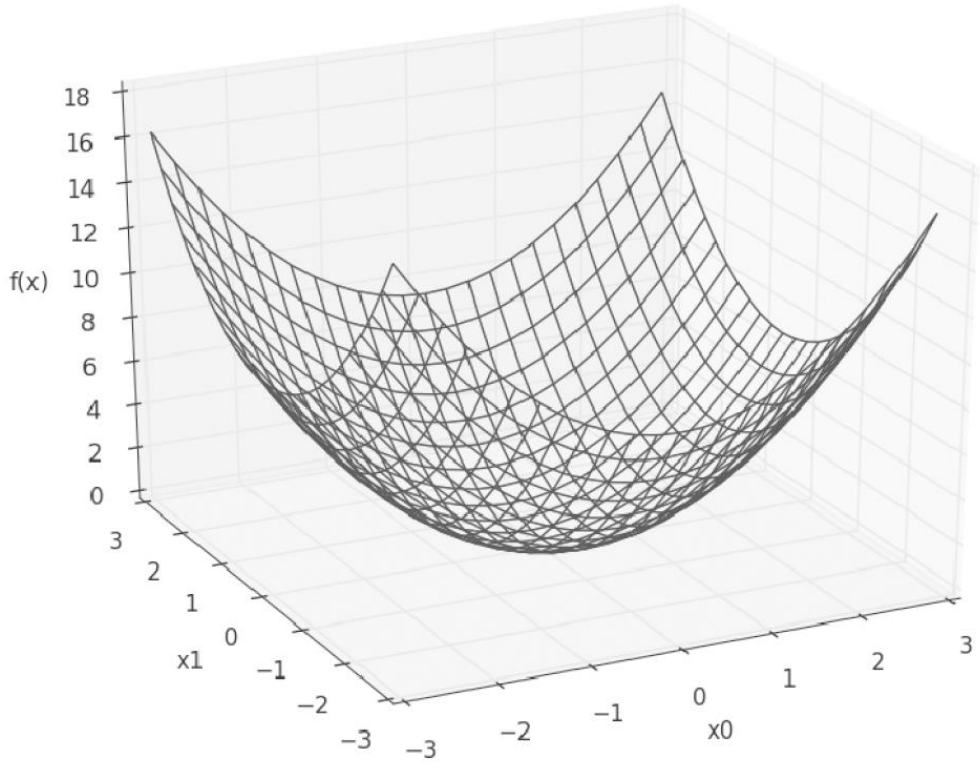
h 방향으로 증가하는 비율

4.3.2 수치 미분의 예

see ch04/gradient_1d.py

4.3.3 편미분

$$f(x_0, x_1) = x_0^2 + x_1^2$$



목표가 되는 변수이외 변수는 상수로 생각하고 미분

4.4 기울기 : gradient $\nabla f(x_0, x_1, \ldots, x_{d-1}) = (\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \cdots, \frac{\partial f}{\partial x_{d-1}})$

ch04/gradient_2d.py main 생략(틀림)하고 함수만 읽기

4.4.1 경사법(경사 하강법: gradient descent method)

기울기를 계산하여 손실함수가 작아지는 방향으로 점차적으로 이동

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

η(에타)는 학습률(learning rate)은 한 번에 매개 변수를 얼마나 갱신하는지를 결정 : 0.01 ~ 1E-9

경사하강법은 위 과정을 여러 번 반복

ch04/gradient_method.py 실행

참고: hyper parameter: 신경망의 가중치처럼 학습되어 최적화 값으로 변경되는 것이 아니라, 사용자가 설정하는 값, 예) 학습률, 신경망 층 수, 뉴런 수, 신경망 구조, 학습 회수 등

4.4.2 신경망에서의 기울기

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{31}} \\ \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{32}} \end{pmatrix}$$

ch04/gradient_simplenet.py 실행 : f 를 계산할 때 net.W 가 내부에서 이용됨

2022. 3. 30일

4.5 학습 알고리즘 구현하기

1. 미니배치
2. 기울기 산출
3. 매개변수 갱신
4. 반복

확률적 경사 하강법 (stochastic gradient descent: SGD) : 무작위로 학습자료를 골라 냄 (한 epoch내에서 shuffling하는 방법 주로 사용)

4.5.1 2층 신경망 클래스 구현하기

ch04/two_layer_net.py 자세히 읽어보고

4.5.2 미니배치 학습 구현하기

ch04/train_neuralnet.py 분석

epoch : 학습자료를 모두 사용하는 회수, 예) 10,000개의 자료를 100개의 미니배치로 학습할 경우 10,000/100=100 개 iteration이 1 epoch

4.6 정리

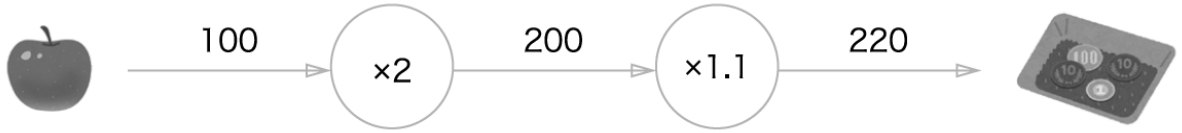
읽어본다.

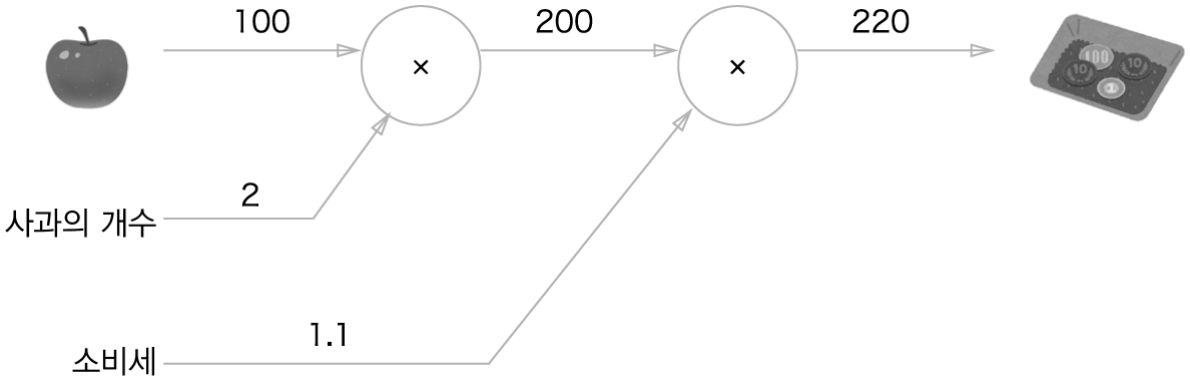
chapter 5. 오차 역전파

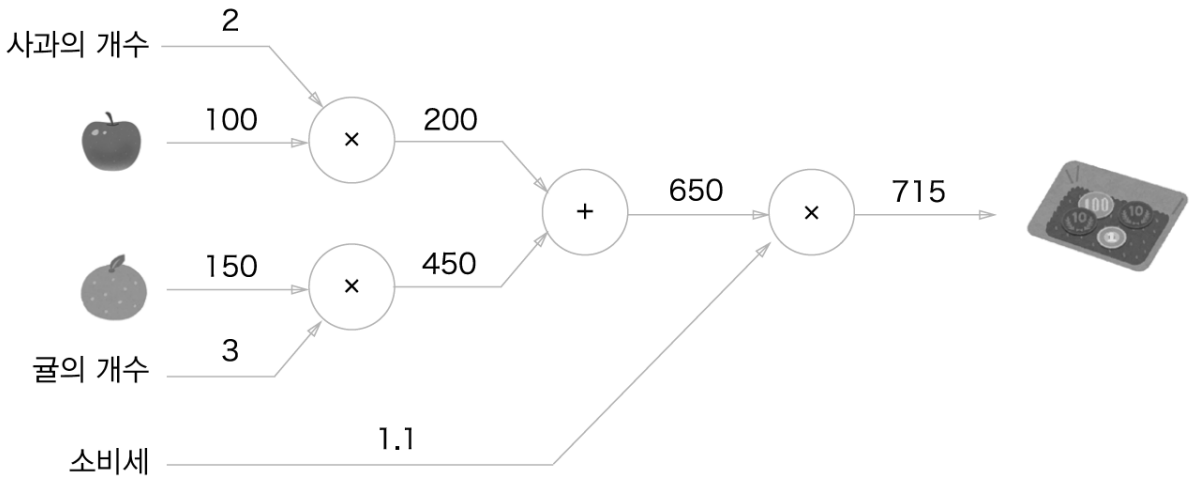
앞장의 수치 미분 대신에 계산이 빠르고 정확한 가중치 매개변수의 기울기를 구하는 방법을 학습 : automatic differentiation

5.1 계산 그래프 (computational graph)

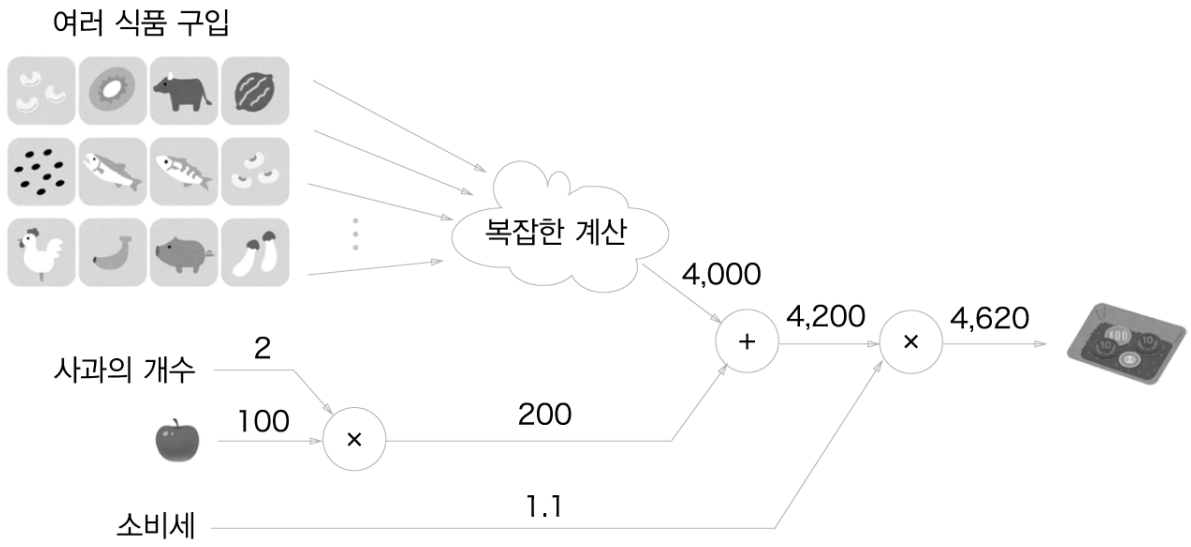
5.1.1 계산 그래프로 풀다


100
×2
200
×1.1
220


100
×
200
×
220
2
사과의 개수
1.1
소비세


2
사과의 개수
100
×
200
+
650
×
715
150
×
450
3
귤의 개수
1.1
소비세

5.1.2 국소적 계산



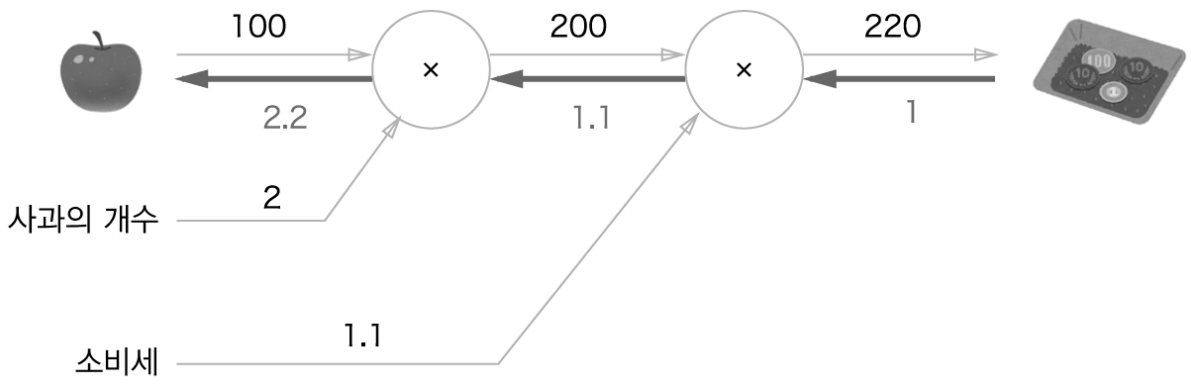
5.1.3 왜 계산 그래프로 푸는가?

1) 전체 계산이 복잡해도 각 노드에서는 단순한 계산

2) 중간 계산 결과를 보관

3) 역전파를 통해 미분 계산 (→ 순전파 값 계산, ← 역전파 미분계산)

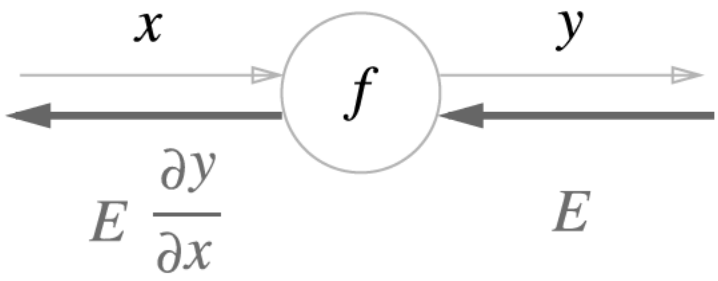
backward propagation



사과값, 사과 개수, 소비세가 미소변위할 때 영향은?

5.2 연쇄법칙

5.2.1 계산 그래프의 역전파: 역방향으로 국소적 미분을 곱함



앞단의 미분값 E에 $\frac{\partial y}{\partial x}$ 을 곱하면 됨: 이유는 다음장

5.2.2 연쇄법칙이란?

합성함수의 미분은 구성함수의 미분값의 곱

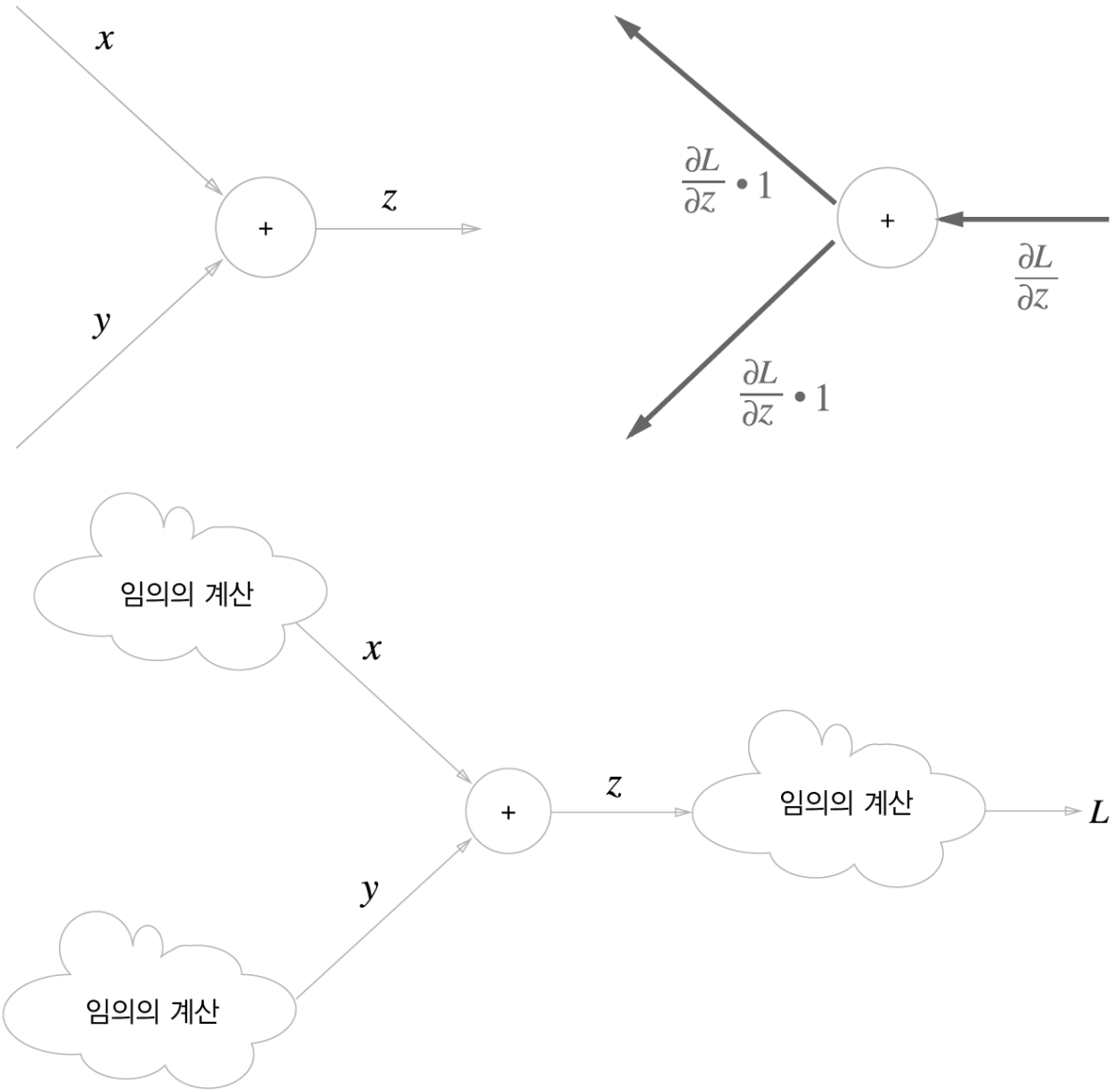
$$[f(g(x))]' = f'(g(x))g'(x)$$

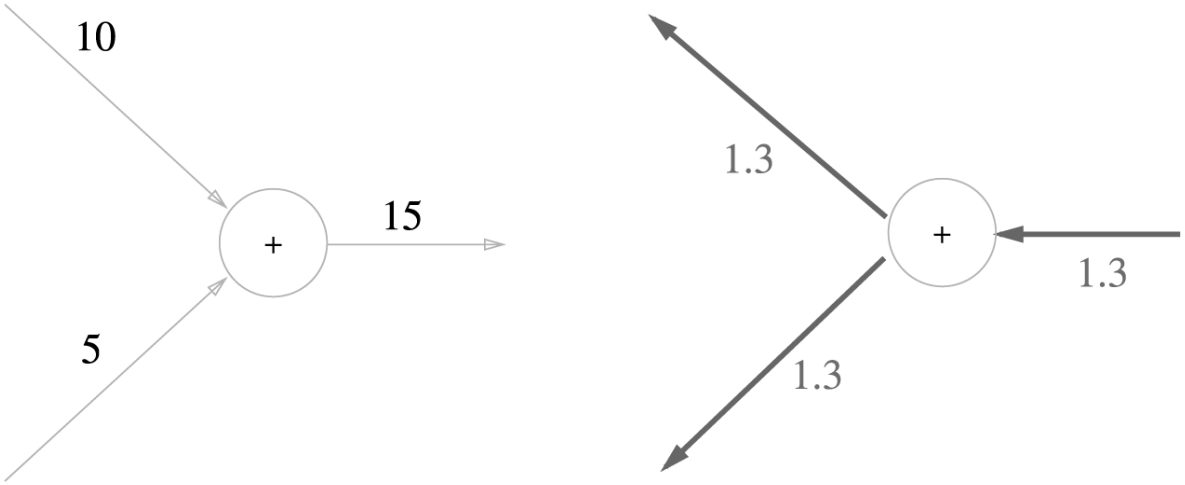
$$\left[f\left(g(h(x))\right)\right]' = f'\left(g(h(x))\right)g'(h(x))h'(x)$$

$(x^3 + y)^2$ 의 x로 미분은 $2(x^3 + y)3x^2$

5.3 역전파

5.3.1 덧셈 노드의 역전파 : z = x + y ← z = g(x,y),

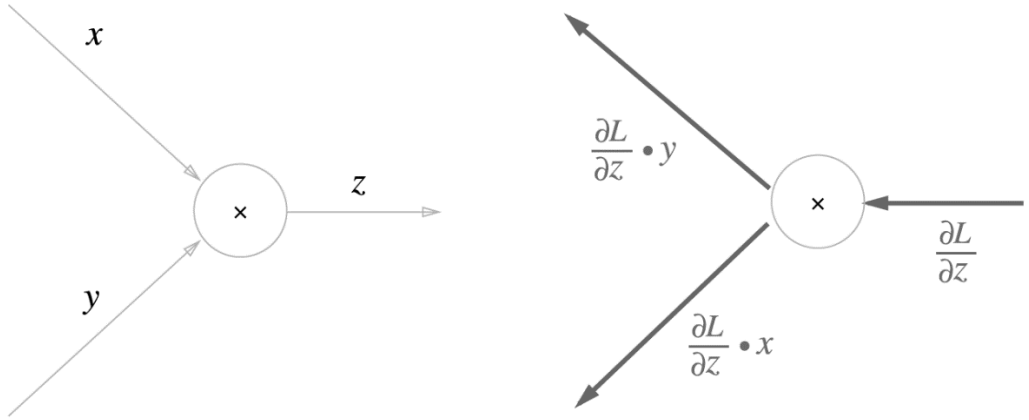


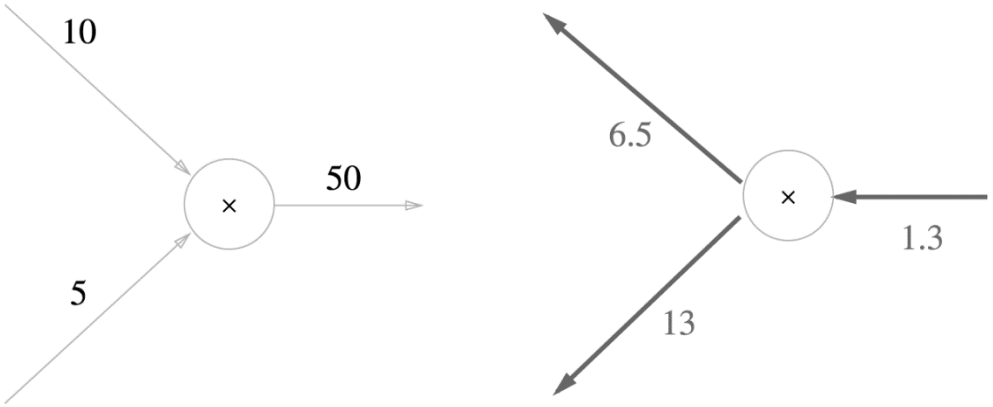


5.3.2 곱셈 노드의 역전파 : z = xy : z = g(x,y)

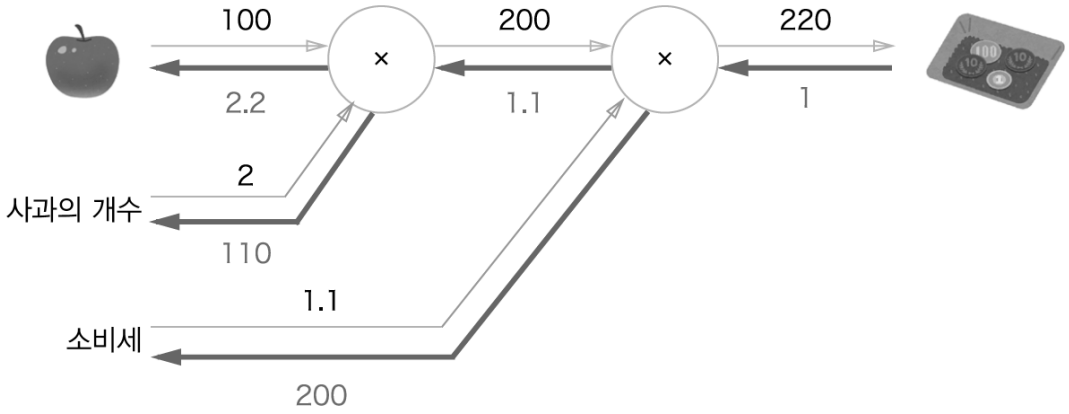
$$\frac{\partial z}{\partial x} = y$$

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





5.3.3 사과 쇼핑의 예



5.4 단순한 계층 구현하기

5.4.1 곱셈계층

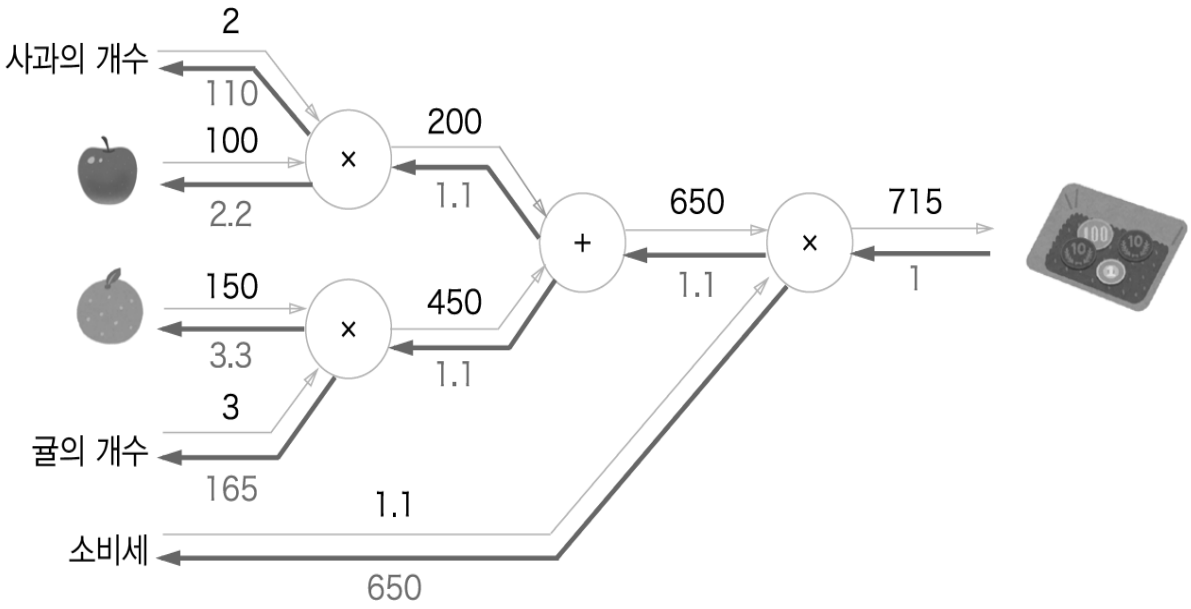
ch05/layer_naive.py MulLayer읽기

ch05/buy_apple.py 결과는 위 그림: 사과 쇼핑의 예

5.4.2 덧셈 계층

ch05/layer_naive.py AddLayer읽기

ch05/buy_apple_orange.py : 순서대로 호출



5.5 활성화 함수 계층 구현하기

5.5.1 ReLU 계층

$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \qquad \frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

common/layers.py Relu 읽기

4월 6일

5.5.2 Sigmoid 계층

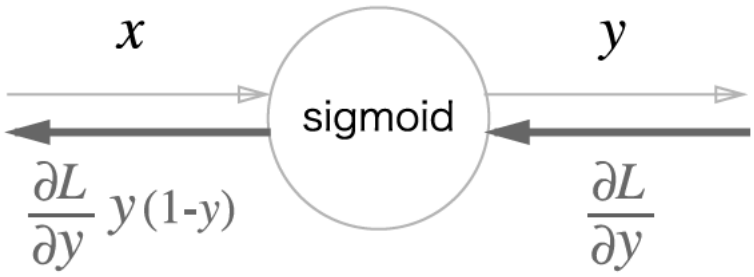
$$y = \frac{1}{1 + \exp(-x)}$$

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

$$= (1 + e^{-x})^{-1}[(1 + e^{-x})^{-1} e^{-x}]$$

$$= (1 + e^{-x})^{-1}[1 - (1 + e^{-x})^{-1}] = y(1 - y)$$

따라서 다음처럼 간결하게 표현됨



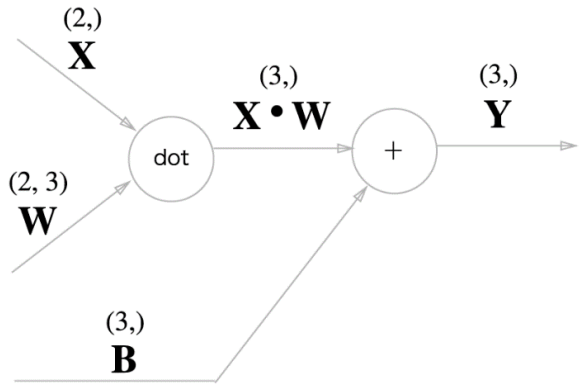
common/layers.py Sigmoid 읽기

5.6 Affine/Softmax 계층 구현하기

5.6.1 Affine 계층

행렬의 내적 + 벡터 : Y = np.dot(X,W) + B

행렬로 구성된 계산 그래프



$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$$

$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

$$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y}$$

이 식의 유도는 아래처럼 복잡, 전체적 모양은 곱셈노드 유사

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$\mathbf{W}^{\mathrm{T}} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

index에 주의

$X = (x_1, x_2, \cdots, x_n),$

$XW = (x_1 w_{11} + x_2 w_{12} + \cdots + x_n w_{1n}, \ldots, x_1 w_{k1} + x_2 w_{k2} + \cdots + x_n w_{kn}, \ldots)$

XW가 Y를 구성 (Y는 덧셈노드의 출력이므로 X에 대한 미분을 구할 때는 B는 잠시 없는 것으로 생각)

x_i는 $Y = (y_1, y_2, \cdots, y_m)$ 각각의 입력변수이므로 x_i로 L을 미분하는 것

은 각각의 y_j로 L을 미분한것에 y_j를 x_i로 미분한 w_{ji}곱함 (연쇄법칙),

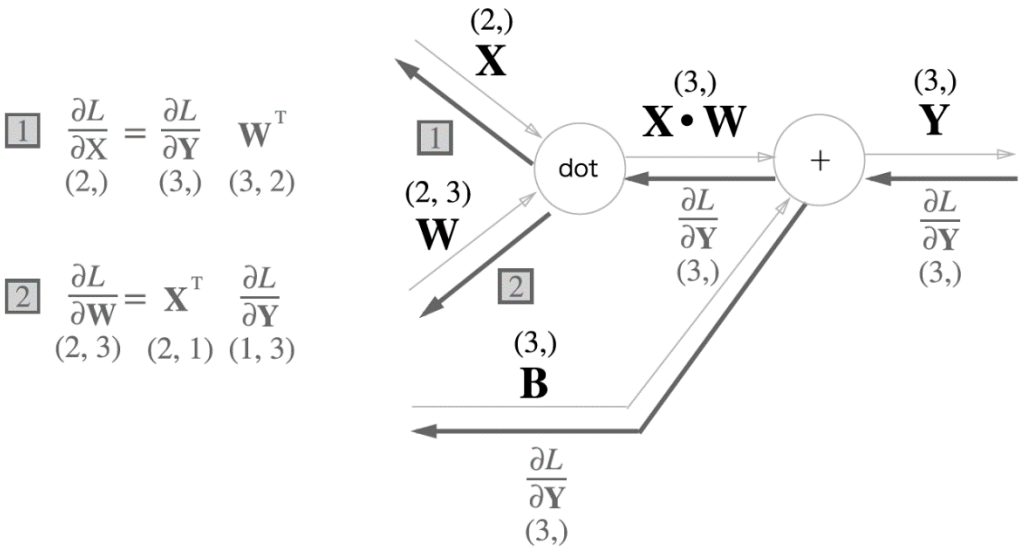
$$\frac{\partial L}{\partial y_1}w_{1i} + \frac{\partial L}{\partial y_2}w_{2i} + \cdots + \frac{\partial L}{\partial y_{\mathrm{m}}}w_{mi} \leftarrow ith\ \ component\ \ of\ \ \frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y}W^T$$

마찬가지로

$XW = (x_1w_{11} + x_2w_{12} + \cdots + x_nw_{1n}, \ldots, x_1w_{k1} + x_2w_{k2} + \cdots + x_nw_{kn}, \ldots)$

w_{ij}는 y_i를 계산하는 변수이므로 w_{ij}로 L을 미분하는 것은 y_i로 L을 미분한 것에 대하여 y_i를 w_{ij}로 미분한 x_j를 곱함

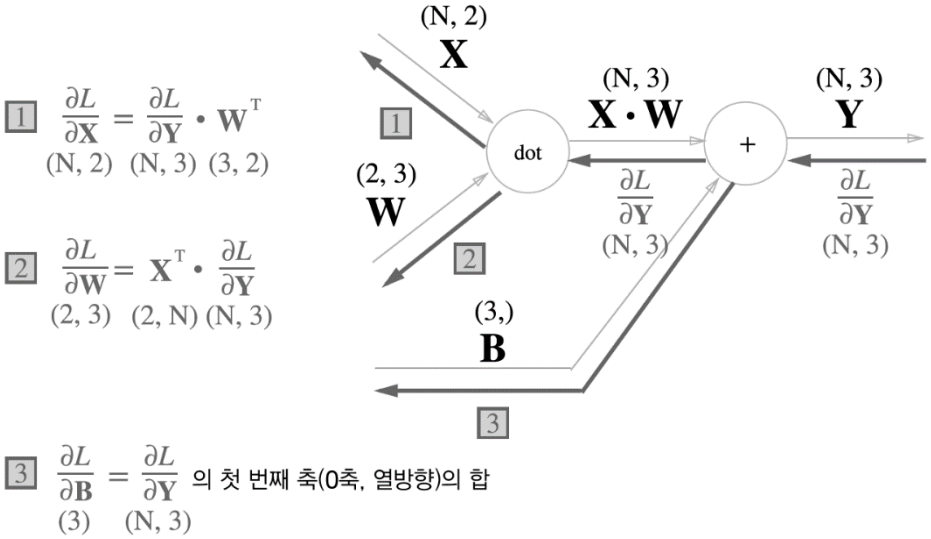
즉, $\frac{\partial L}{\partial w_{\mathrm{ij}}} = \frac{\partial L}{\partial y_{\mathrm{i}}}x_j \longrightarrow \frac{\partial L}{\partial W} = X^T\frac{\partial L}{\partial Y}$



B에 대한 미분은 덧셈노드이므로 Y로 미분한 것과 동일

5.6.2 배치용 Affine 계층

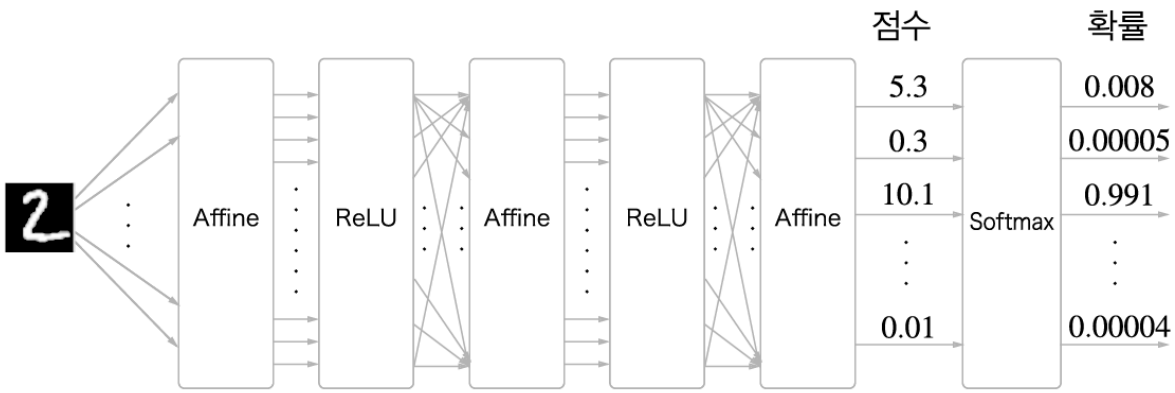
데이터 N개를 묶어 순전파하는 경우

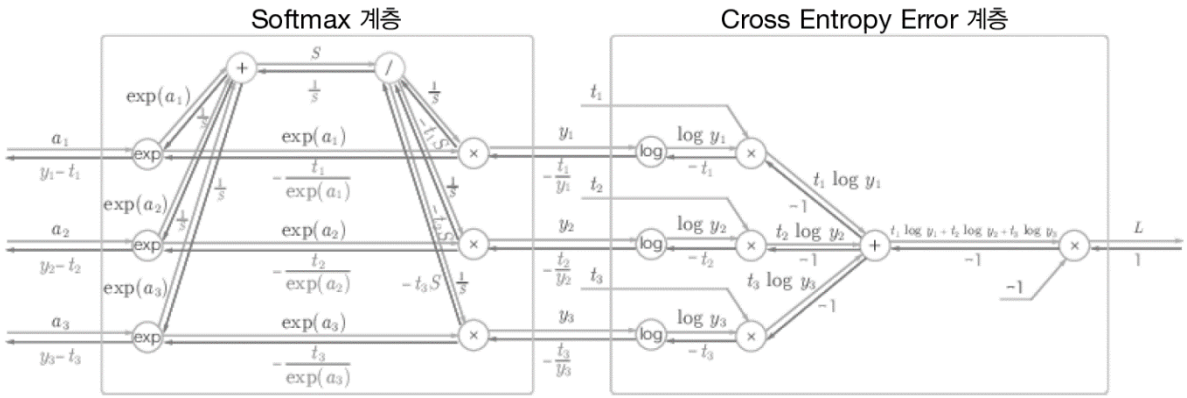


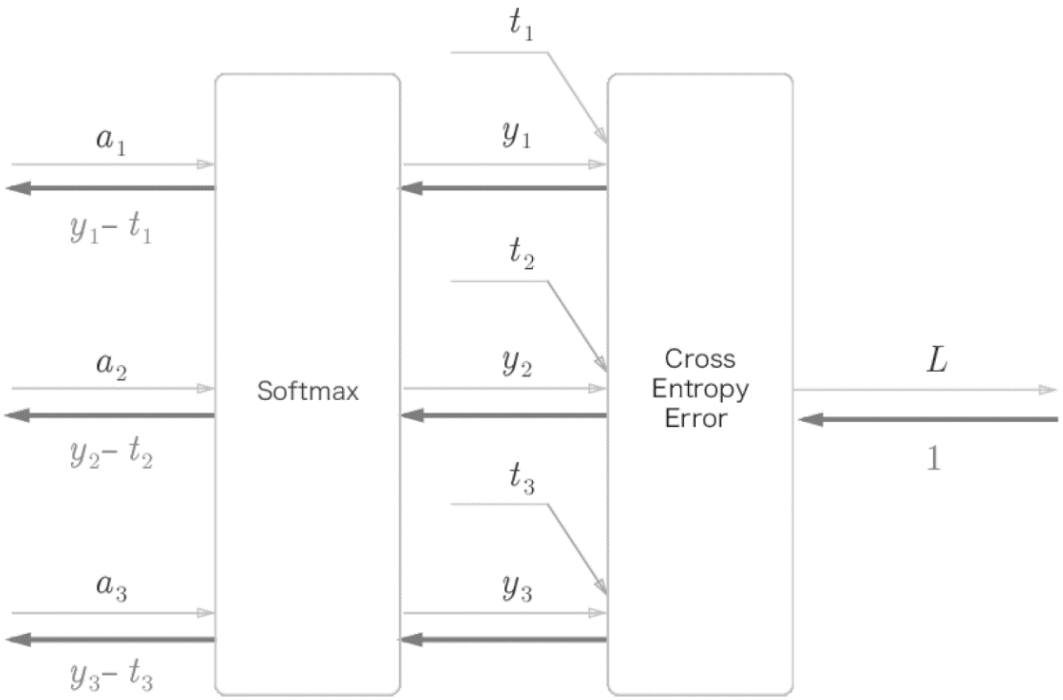
주의 [3]식의 순전파의 편향은 각 데이터에 더해짐으로 $a_1x + a_2x + \cdots + a_nx$를 x로 미분하는 $a_1 + a_2 + \cdots + a_n$ 로 처리해야 함

common/layers.py, Affine class

5.6.3 Softmax-with-Loss 계층







유도과정은 부록 A. Softmax-with-loss

구현은 common/layers.py, class SoftmaxWithLoss

5.7 오차 역전파법 구현하기

4.5 학습 알고리즘 구현하기와 동일

1. 미니배치

2. 기울기 산출

3. 매개변수 갱신

4. 반복

기울기 산출부분만 오차역전파로 바뀜

5.7.2 오차역전파를 적용한 신경망 구현하기

ch05/TwoLayNet

5.7.3 오차역전파법으로 구현한 기울기 검증하기

정확하므로 생략

5.7.4 오차역전파를 이용한 학습

ch05/train_neuralnet.py