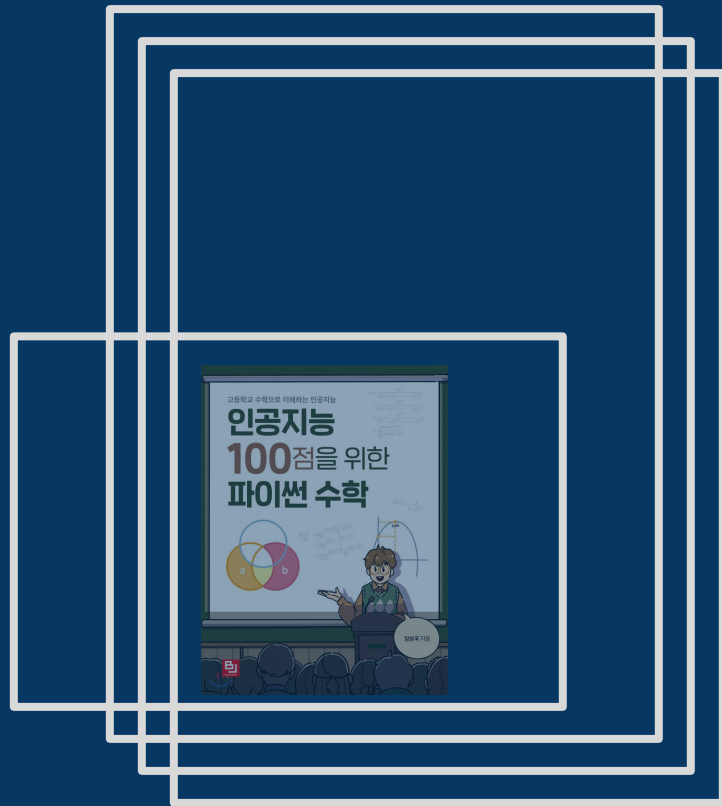


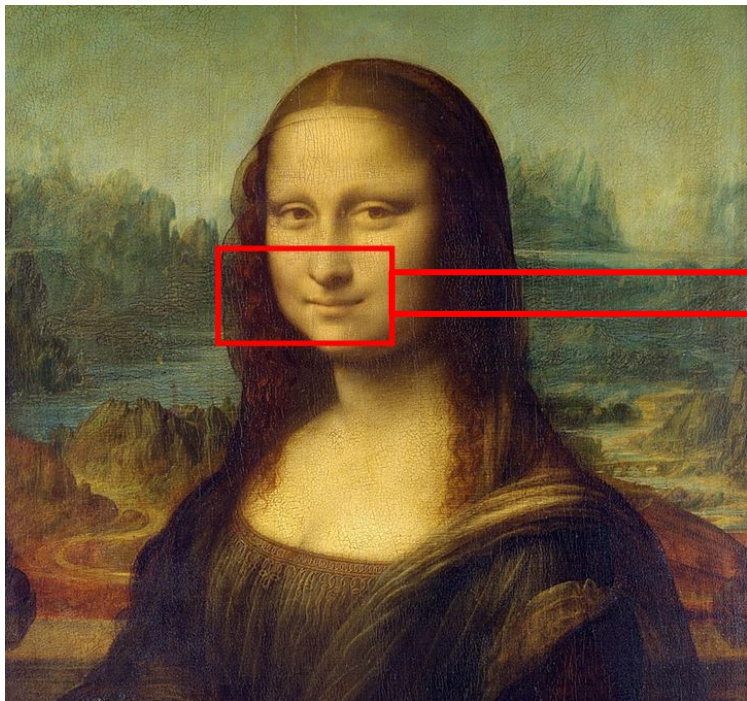
파이썬으로 구현해보는 딥러닝

임성국



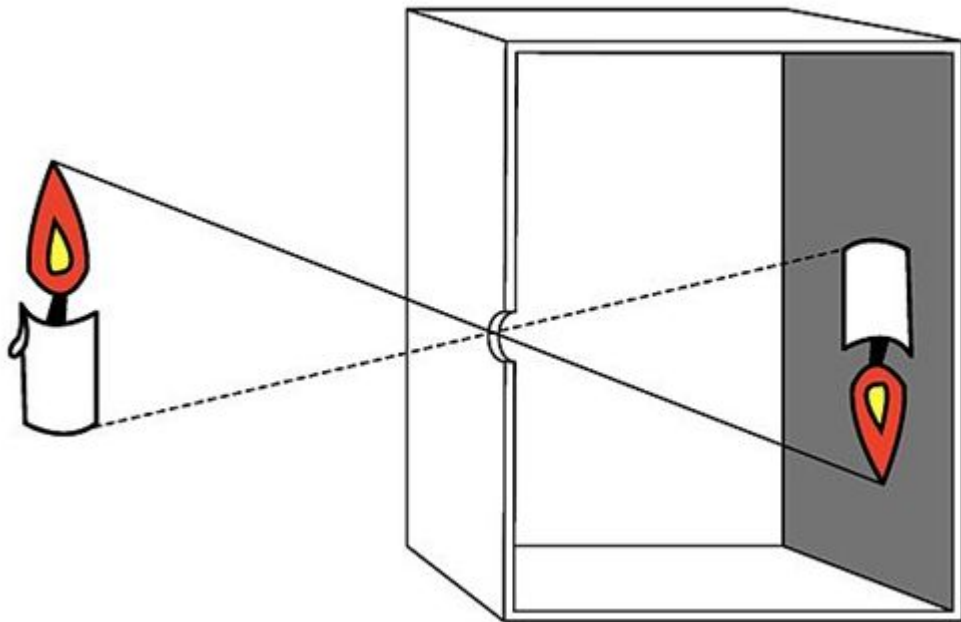
인공지능, 머신러닝, 딥러닝

모나리자



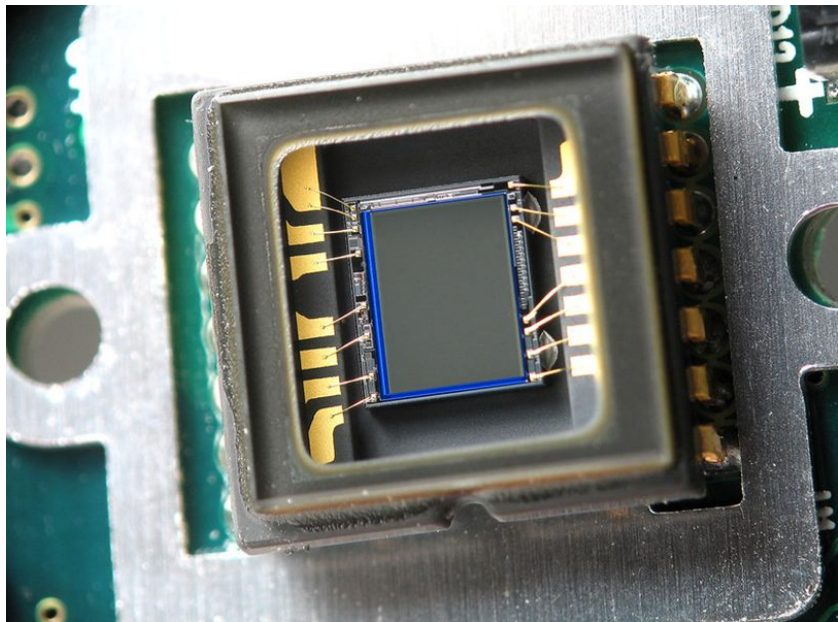
영상의 이해

● 사진기의 원리



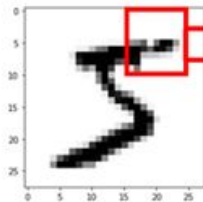
디지털과 아날로그

CMOS 센서



디지털과 아날로그

● MNIST 데이터의 모습



0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
18	126	136	175	26	166	255	247	127	0
253	253	253	225	172	253	242	195	64	0
253	253	251	93	82	82	56	39	0	0
182	247	241	0	0	0	0	0	0	0
0	43	154	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

샘플링과 양자화

○ 양자화

양자화는 그림의
색을 저장할 때 어느
정도의 정밀도를
가지게 저장할
것인지를 결정합니다

2 단계로 표현한 희고 검은 정도입니다.

단계 1 [■] - 검다 ■ □ 희다

0 [0] 1 bit

단계 2 [□] - 검다 □ ■ 희다

1 [1] 1 bit

4 단계로 표현해 봅시다.

단계 1 [■] - 검다 ■ □ □ □ 희다

0 [00] 2 bits

단계 2 [■] - 검다 □ ■ □ □ 희다

1 [01] 2 bits

단계 3 [■] - 검다 □ □ ■ □ 희다

2 [10] 2 bits

단계 4 [■] - 검다 □ □ □ ■ 희다

3 [11] 2 bits

샘플링과 양자화

○ 양자화

비트수에 따라
양자화의 정도가
결정됨

8 단계로도 표현해 봅시다.

단계 1 [■] - 검다		희다	0 [000] 3 bits
단계 2 [■] - 검다		희다	1 [001] 3 bits
단계 3 [■] - 검다		희다	2 [010] 3 bits
단계 4 [■] - 검다		희다	3 [011] 3 bits
단계 5 [■] - 검다		희다	4 [100] 3 bits
단계 6 [■] - 검다		희다	5 [101] 3 bits
단계 7 [■] - 검다		희다	6 [110] 3 bits
단계 8 [□] - 검다		희다	7 [111] 3 bits

샘플링과 양자화

○ 양자화

8 bit = 256

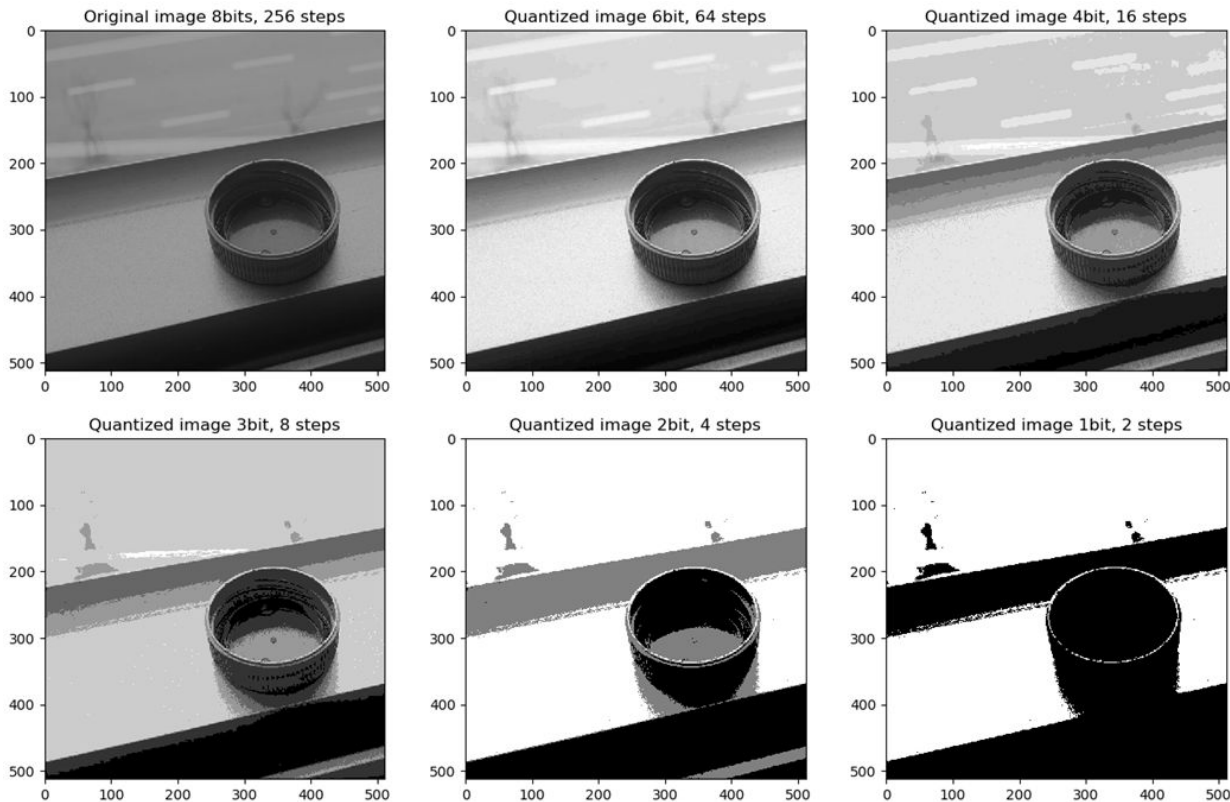
6 bit = 64

4 bit = 16

3 bit = 8

2 bit = 4

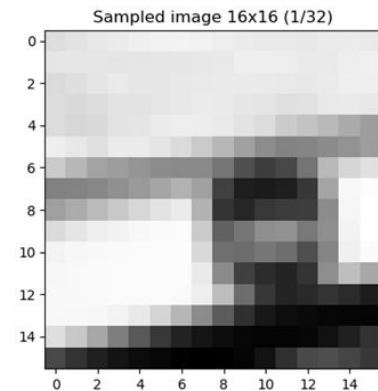
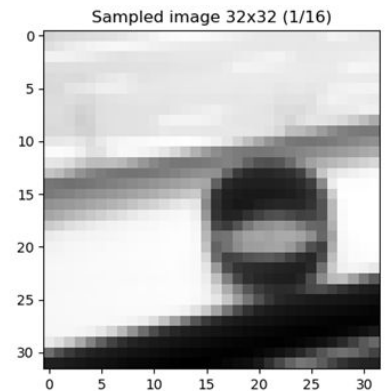
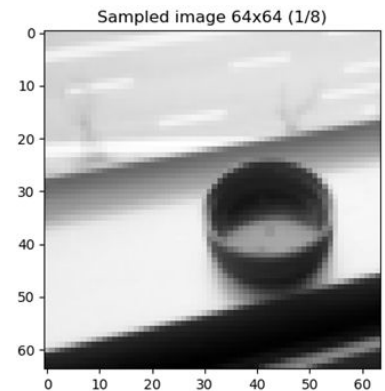
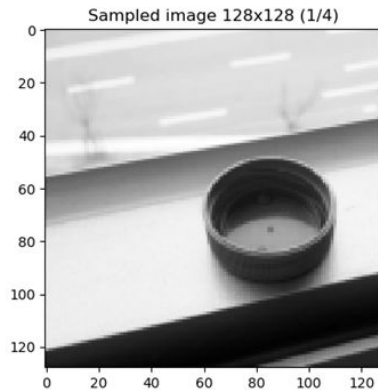
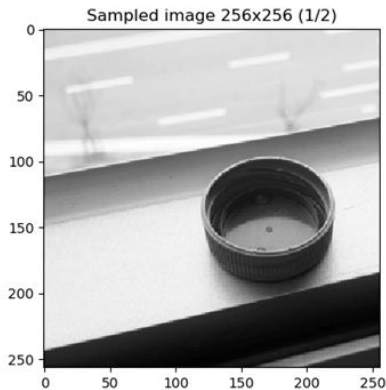
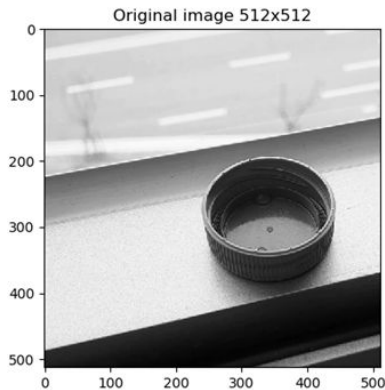
1 bit = 2



샘플링과 양자화

● 샘플링

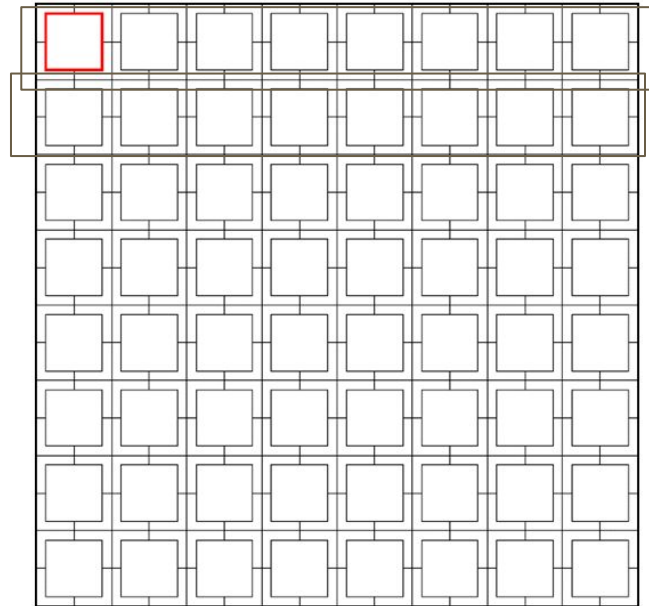
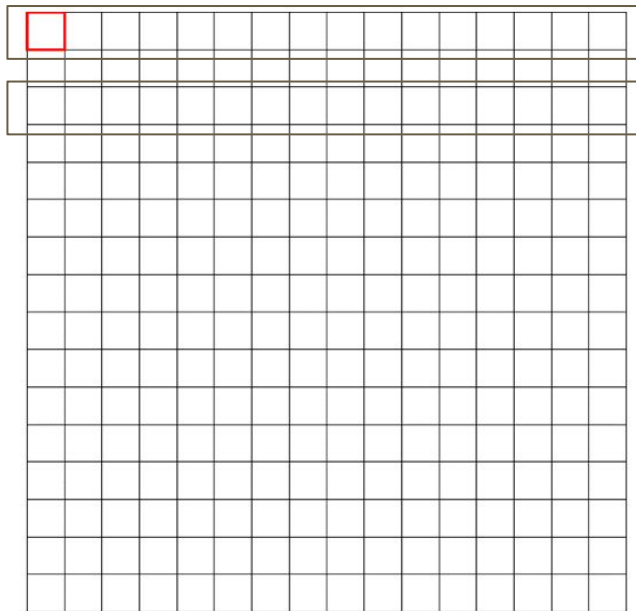
샘플링은 모든
입력값을 다
저장하는 것이
아니라 일정
간격으로 샘플을
모으고 모인 샘플을
사용하겠다는 것



샘플링과 양자화

● 서브샘플링

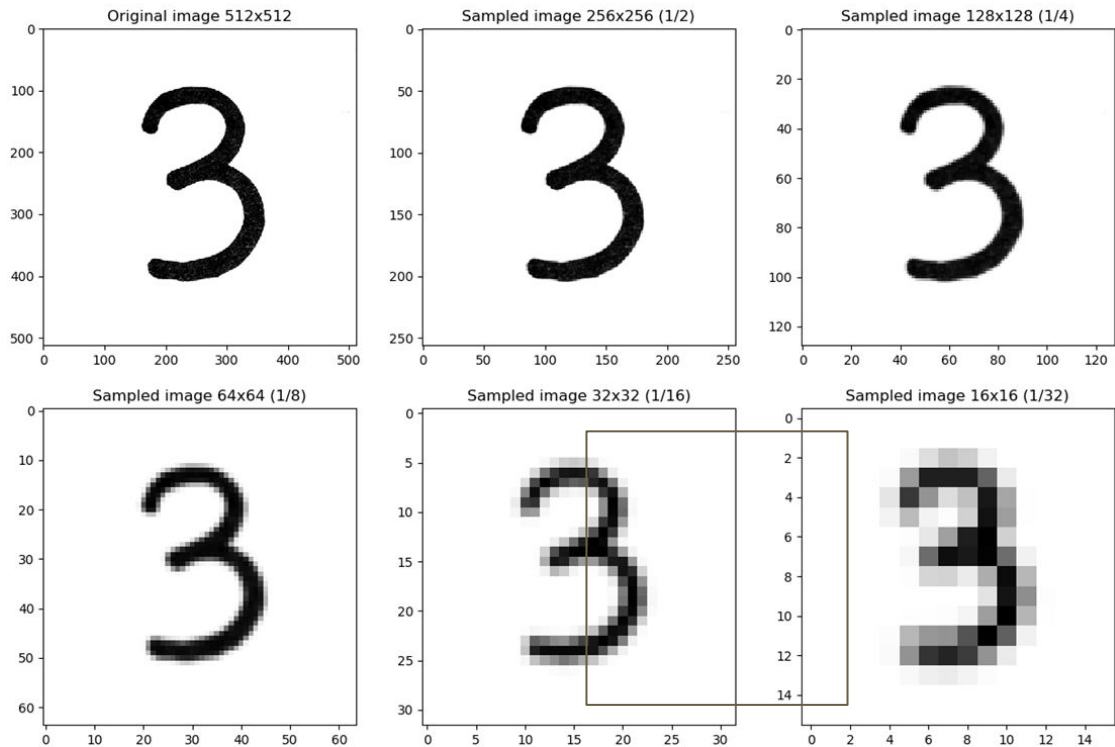
이미지 픽셀의 값을
건너뛰면서 가져오는
것



샘플링과 양자화

서브샘플링

512x512 를
서브샘플링으로
축소한 그림



행렬식의 이해

행렬식의 이해

○ 행렬곱

$$A \cdot B = C$$

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}$$

$$A \cdot B = \begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} a_1b_1 + a_2b_3 & a_1b_2 + a_2b_4 \\ a_3b_1 + a_4b_3 & a_3b_2 + a_4b_4 \end{pmatrix}$$

행렬의 계산과 선형대수

● 연습문제. 행렬을 이용한 연산

포도, 귤, 사과, 배를 각각 1개를 구입하고 2,350원을 지불했습니다. 다음날 순서대로 5개, 2개, 4개, 7개를 구입하고 10,600원을 지불했습니다. 다시 다음날 귤 1개, 사과 1개, 배 2개를 구입하고 2,450원을 지불했습니다. 다음날 5개, 2개, 2개, 3개를 구입하고 6,800원을 지불했습니다. 받은 영수증에는 포도, 귤, 사과, 배의 개별 가격이 기록되지 않은 채 총금액만 적혀 있었습니다. 그러면 과일들의 개당 가격은 얼마일까요?

행렬의 계산과 선형대수

행렬을 이용한 연산

포도, 귤, 사과, 배를 각각 W , X , Y , Z 라고 놓으면,

$$W + X + Y + Z = 2350 \text{ --- (1)}$$

$$5W + 2X + 4Y + 7Z = 10600 \text{ --- (2)}$$

$$X + Y + 2Z = 2450 \text{ --- (3)}$$

$$5W + 2X + 2Y + 3Z = 6800 \text{ --- (4)}$$

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 5 & 2 & 4 & 7 \\ 0 & 1 & 1 & 2 \\ 5 & 2 & 2 & 3 \end{bmatrix} \begin{bmatrix} W \\ X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 2350 \\ 10600 \\ 2450 \\ 6800 \end{bmatrix}$$

행렬 계산식의 수학적 이해

● 행렬을 이용한 연산

귤 하나, 사과 하나의 가격이 1, 1 0 0 원이고, 귤 둘, 사과 셋의 가격이 2, 8 0 0 원이라면

$$\begin{array}{rcl} X + Y = 1\ 1\ 0\ 0 & \text{----} & (1) \\ 2X + 3Y = 2\ 8\ 0\ 0 & \text{--} & (2) \end{array} \quad \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 1100 \\ 2800 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}, \quad X = \begin{pmatrix} X \\ Y \end{pmatrix}, \quad B = \begin{pmatrix} 1100 \\ 2800 \end{pmatrix}$$

행렬 계산식의 수학적 이해

● 행렬을 이용한 연산

$$A(2,2) \cdot X(2,1) = B(2,1)$$

A 행의 크기 = B 행의 크기
 B 열의 크기 = B 열의 크기
 A 열의 크기 = X 행의 크기

행렬 계산식의 수학적 이해

행렬을 이용한 연산

$$\begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 1100 \\ 2800 \end{pmatrix} \quad \begin{matrix} \mathbf{AX} = \mathbf{B} \\ \mathbf{X} = \mathbf{A}^{-1}\mathbf{B} \end{matrix}$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 1100 \\ 2800 \end{pmatrix}$$

파이썬 코드

행렬계산식

$$A = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}, \quad X = \begin{pmatrix} X \\ Y \end{pmatrix}, \quad B = \begin{pmatrix} 1100 \\ 2800 \end{pmatrix} \text{ 일 때}$$

$$\begin{pmatrix} X \\ Y \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 3 \end{pmatrix}^{-1} \begin{pmatrix} 1100 \\ 2800 \end{pmatrix} \text{ 이므로}$$

$$X = A^{-1} B$$

파이썬 코드

1.2.1

CODE

$$X = A^{-1} B$$

```
import numpy as np
from numpy.linalg import inv

A = np.array([[1,1],
              [2,3]])
B = np.array([[1100],
              [2800]])

invA = inv(A)
print(np.dot(invA , B))
```

파이썬 코드

🔴 연습문제 8-1 을 파이썬코드로 풀어보세요

포도, 귤, 사과, 배를 각각 1개를 구입하고 2,350원을 지불했습니다. 다음날 순서대로 5개, 2개, 4개, 7개를 구입하고 10,600원을 지불했습니다. 다시 다음날 귤 1개, 사과 1개, 배 2개를 구입하고 2,450원을 지불했습니다. 다음날 5개, 2개, 2개, 3개를 구입하고 6,800원을 지불했습니다. 받은 영수증에는 포도, 귤, 사과, 배의 개별 가격이 기록되지 않은 채 총금액만 적혀 있었습니다. 그러면 과일들의 개당 가격은 얼마일까요?

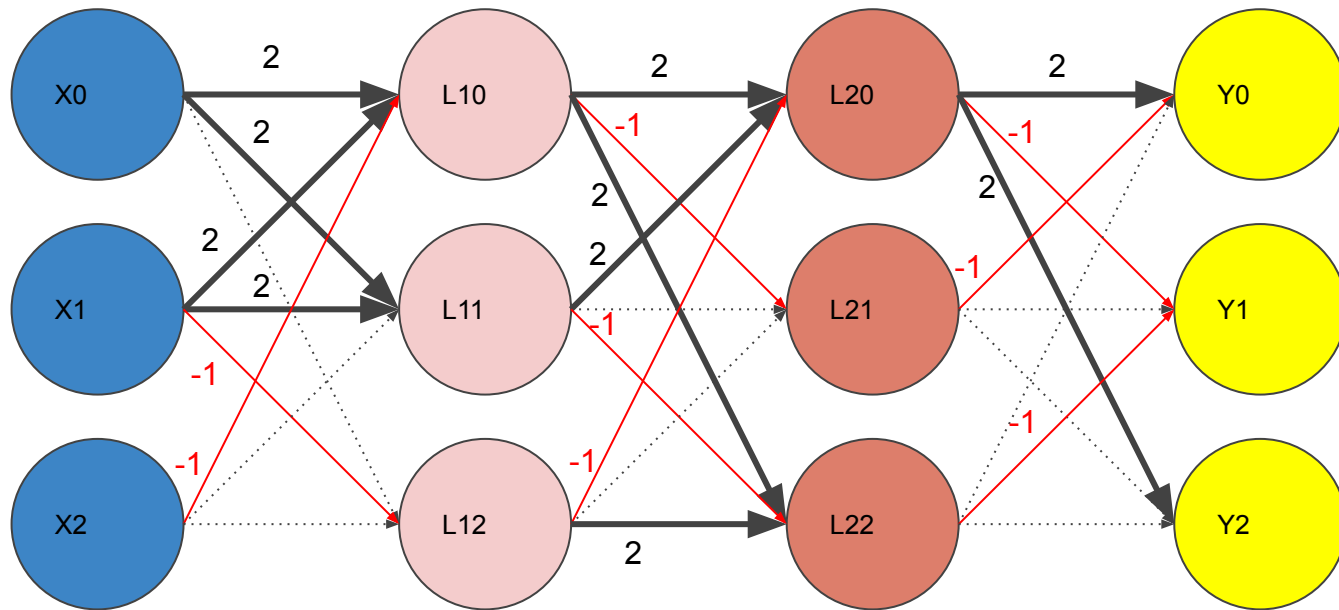
CODE

```
# ml_ex_08_01.py
import numpy as np
from numpy.linalg import inv

.....
```


행렬식과 신경망

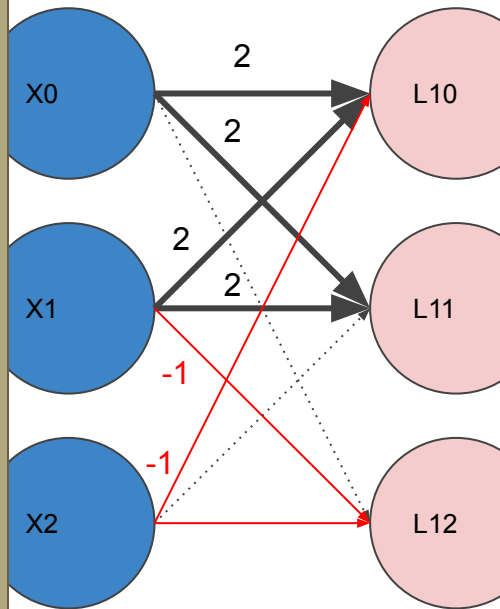
행렬식과 신경망



행렬식과 신경망

입력 :
X0, X1, X2

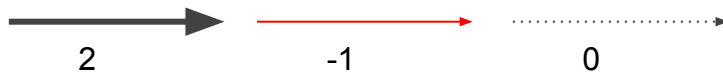
출력 :
L10, L11, L12



$$L10 = 2X0 + 2X1 - X2$$

$$L11 = 2X0 + 2X1$$

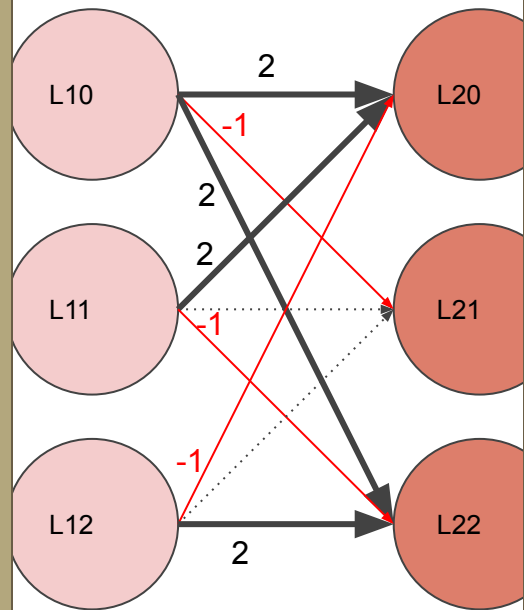
$$L12 = -X1 - X2$$



행렬식과 신경망

입력 : L10, L11, L12

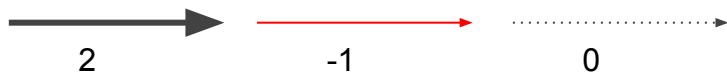
출력 : L20, L21, L22



$$L20 = 2L10 + 2L11 - L12$$

$$L21 = -L10$$

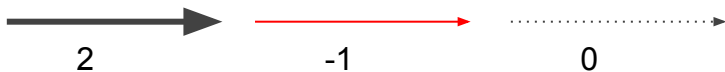
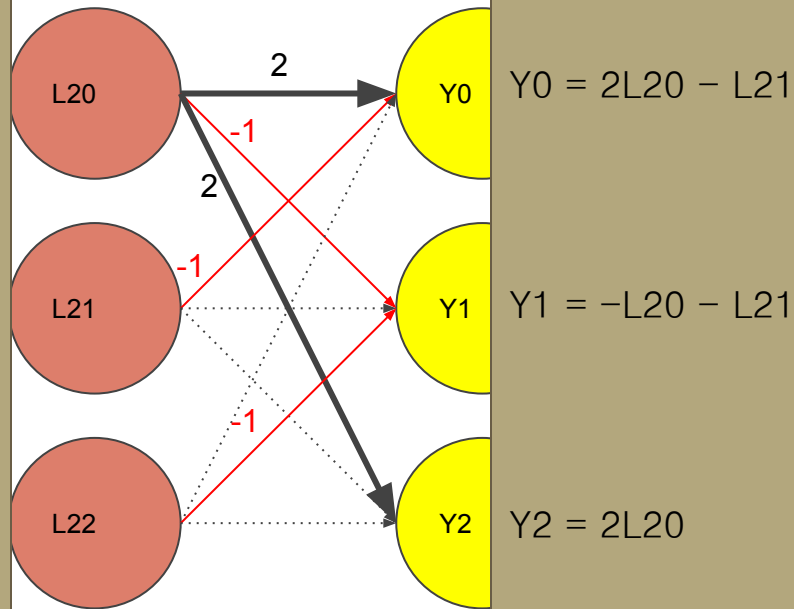
$$L22 = 2L10 - L11 + 2L12$$

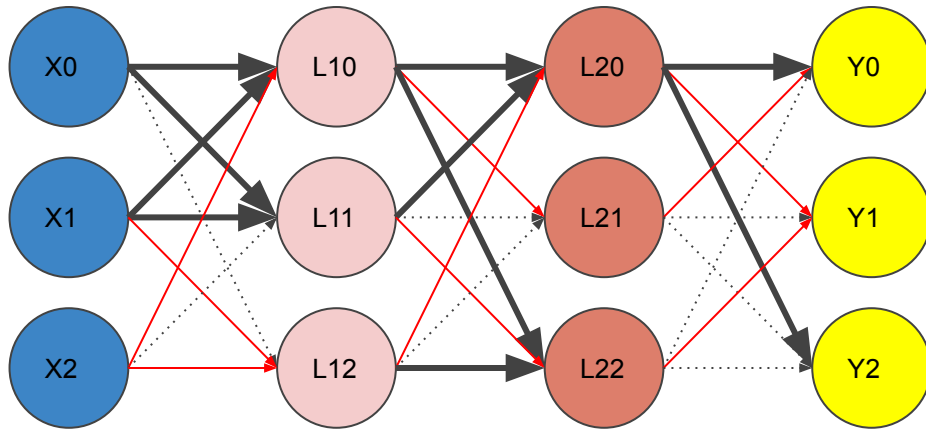


행렬식과 신경망

입력 : L20, L21, L22

출력 : Y0, Y1, Y2





$$L10 = 2X0 + 2X1 - X2$$

$$L11 = 2X0 + 2X1$$

$$L12 = -X1 - X2$$

$$L20 = 2L10 + 2L11 - L12$$

$$L21 = -L10$$

$$L22 = 2L10 - L11 + 2L12$$

$$Y0 = 2L20 - L21$$

$$Y1 = -L20 - L21$$

$$Y2 = 2L20$$

$$L10 = 2X0 + 2X1 - X2$$

$$L11 = 2X0 + 2X1$$

$$L12 = -X1 - X2$$

$$L20 = 2L10 + 2L11 - L12$$

$$L21 = -L10$$

$$L22 = 2L10 - L11 + 2L12$$

$$Y0 = 2L20 - L21$$

$$Y1 = -L20 - L21$$

$$Y2 = 2L20$$

$$\begin{bmatrix} L10 \\ L11 \\ L12 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ 2 & 2 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} X0 \\ X1 \\ X2 \end{bmatrix}$$

$$\begin{bmatrix} L20 \\ L21 \\ L22 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ -1 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} L10 \\ L11 \\ L12 \end{bmatrix}$$

$$\begin{bmatrix} Y0 \\ Y1 \\ Y2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} L20 \\ L21 \\ L22 \end{bmatrix}$$

출력

세번째 계산

두번째 계산

첫번째 계산

입력

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} L_{20} \\ L_{21} \\ L_{22} \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ -1 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} L_{10} \\ L_{11} \\ L_{12} \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ 2 & 2 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}$$

$$\begin{aligned}
 L_{10} &= 2X_0 + 2X_1 - X_2 \\
 L_{11} &= 2X_0 + 2X_1 \\
 L_{12} &= -X_1 - X_2
 \end{aligned}$$

$$\begin{aligned}
 L_{20} &= 2L_{10} + 2L_{11} - L_{12} \\
 L_{21} &= -L_{10} \\
 L_{22} &= 2L_{10} - L_{11} + 2L_{12}
 \end{aligned}$$

$$\begin{aligned}
 Y_0 &= 2L_{20} - L_{21} \\
 Y_1 &= -L_{20} - L_{21} \\
 Y_2 &= 2L_{20}
 \end{aligned}$$

$$\begin{bmatrix} L_{10} \\ L_{11} \\ L_{12} \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ 2 & 2 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} X_0 \\ X_1 \\ X_2 \end{bmatrix}$$

$$\begin{bmatrix} L_{20} \\ L_{21} \\ L_{22} \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ -1 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} L_{10} \\ L_{11} \\ L_{12} \end{bmatrix}$$

$$\begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} L_{20} \\ L_{21} \\ L_{22} \end{bmatrix}$$

`np.array([])`

출력

세번째 계산

두번째 계산

첫번째 계산

입력

$$\begin{bmatrix} Y0 \\ Y1 \\ Y2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 2 & 2 & -1 \\ -1 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} 2 & 2 & -1 \\ 2 & 2 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} X0 \\ X1 \\ X2 \end{bmatrix}$$

`np.dot(A,B)`

`X = np.array([0,0,1])`

`W1 = np.array([[2,2,-1],[2,2,0],[0,-1,-1]])`

`W2 = np.array([[2,2,-1],[-1,0,0],[2,-1,2]])`

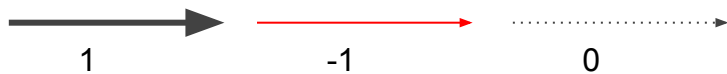
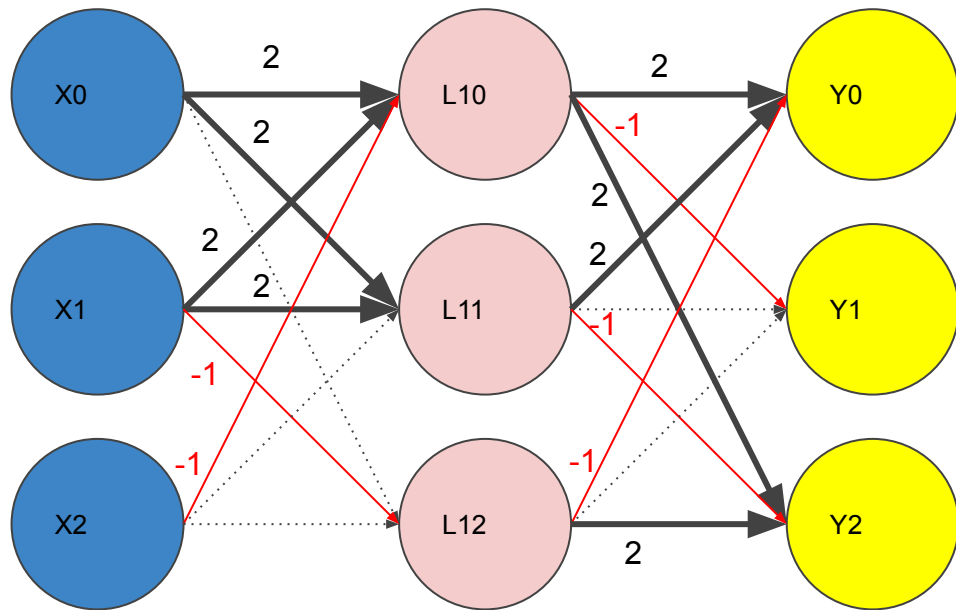
`W3 = np.array([[2,-1,0],[-1,-1,0],[2,0,0]])`

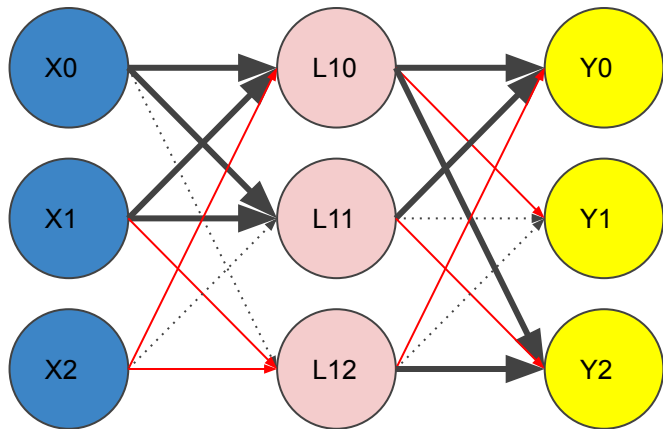
`Y = np.dot(np.dot(np.dot(W3, W2), W1), X)`

`Y = np.dot(W3, np.dot(W2, np.dot(W1, X)))`

$((A \times B) \times C) \times D = A \times (B \times (C \times D)) \neq D \times C \times B \times A$

행렬식과 신경망





$$\begin{aligned} L10 &= 2X0 + 2X1 - X2 \dots + w783 X783 \\ L11 &= 2X0 + 2X1 \\ L12 &= -X1 - X2 \end{aligned}$$

$$\begin{aligned} L20 &= 2L10 + 2L11 - L12 \\ L21 &= -L10 \\ L22 &= 2L10 - L11 + 2L12 \end{aligned}$$

$$\begin{aligned} Y0 &= 2L20 - L21 \\ Y1 &= -L20 - L21 \\ Y2 &= 2L20 \end{aligned}$$

$$\begin{aligned} L10 &= 2X0 + 2X1 - X2 \\ L11 &= 2X0 + 2X1 \\ L12 &= -X1 - X2 \end{aligned}$$

$$\begin{aligned} L20 &= 2L10 + 2L11 - L12 \\ L21 &= -L10 \\ L22 &= 2L10 - L11 + 2L12 \end{aligned}$$

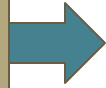
$$\begin{aligned} Y0 &= 2L20 - L21 \\ Y1 &= -L20 - L21 \\ Y2 &= 2L20 \end{aligned}$$



$$\begin{bmatrix} L10 \\ L11 \\ L12 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ 2 & 2 & 0 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} X0 \\ X1 \\ X2 \end{bmatrix}$$



$$\begin{bmatrix} L20 \\ L21 \\ L22 \end{bmatrix} = \begin{bmatrix} 2 & 2 & -1 \\ -1 & 0 & 0 \\ 2 & -1 & 2 \end{bmatrix} \begin{bmatrix} L10 \\ L11 \\ L12 \end{bmatrix}$$



$$\begin{bmatrix} Y0 \\ Y1 \\ Y2 \end{bmatrix} = \begin{bmatrix} 2 & -1 & 0 \\ -1 & -1 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} L20 \\ L21 \\ L22 \end{bmatrix}$$

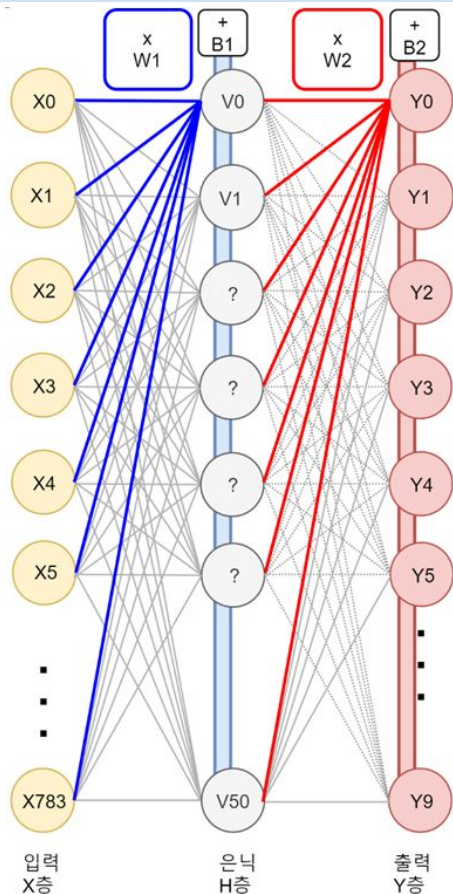
신경망 데이터의 행렬 특징

신경망 데이터의 행렬 특징

● 신경망

여기서는 신경망 전체의 코드 중 행렬을 주의해서 살펴봐야 하는 부분에 대해서 다루겠습니다.

784개의 입력과 10개의 출력이 있는 신경망에서 은닉층이 하나 있고 은닉층에 50개의 노드가 있는 간단한 신경망을 만들어보겠습니다.



무작위 데이터로 신경망함수 만들기

● 신경망함수의 파라미터들

지금까지 공부한 것으로 신경망함수를 만들 수 있습니다. 단, 학습되지 않은 신경망은 전혀 쓸모가 없습니다. 학습한다는 것은 $W1$, $W2$, $B1$, $B2$ 행렬에 담긴 값들을 변화시켜가면서 정답에 가까워지게 한다는 것입니다. 즉, $W1$, $W2$, $B1$, $B2$ 행렬에 있는 모든 수치를 변화시켜야 한다는 뜻인데, $50 \times 784 + 10 \times 50 + 50 + 10$ 인 총 39,760개의 수치를 변화시켜야 합니다. 즉, 39,760개의 수치들이 적절한 값을 가지면 28×28 크기의 손글씨 영상인 784개의 픽셀 값을 받아서 출력인 $Y0$ 에서 $Y9$ 까지, 0에서 9 사이의 답에 1을 넣고 나머지에 0을 출력하게 된다는 것입니다.

무작위 데이터로 신경망함수 만들기

● 신경망함수의 파라미터들

입력 784 개 (고정)

은닉 50 개 (고정) : w_{ij} 784*50, b_i 50

$$v_0 = w_{0_0}x_0 + w_{0_1}x_1 + w_{0_2}x_2 + \dots + w_{0_{783}}x_{783} + b_0$$

$$v_1 = w_{1_0}x_0 + w_{1_1}x_1 + w_{1_2}x_2 + \dots + w_{1_{783}}x_{783} + b_1$$

...

$$v_{49} = w_{49_0}x_0 + w_{49_1}x_1 + w_{49_2}x_2 + \dots + w_{49_{783}}x_{783} + b_{49}$$

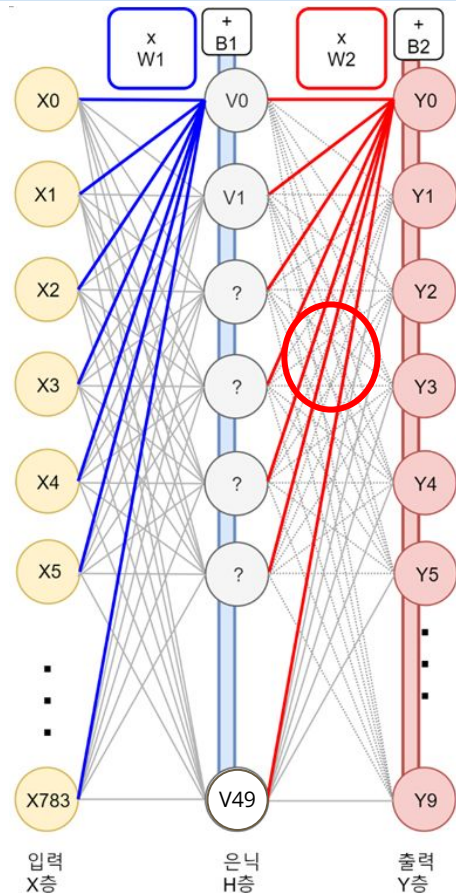
출력 10 개 (고정) : w_{ij} 50*10 b_i 10

$$Y_0 = w_{2_0_0}v_0 + w_{2_0_1}v_1 + w_{2_0_2}v_2 + \dots + w_{2_0_{49}}v_{49} + b_{2_0}$$

$$Y_1 = w_{2_1_0}v_0 + w_{2_1_1}v_1 + w_{2_1_2}v_2 + \dots + w_{2_1_{49}}v_{49} + b_{2_1}$$

...

$$Y_9 = w_{2_9_0}v_0 + w_{2_9_1}v_1 + w_{2_9_2}v_2 + \dots + w_{2_9_{49}}v_{49} + b_{2_9}$$



소프트맥스

항등 함수

○ 항등함수(identity function)

입력을 그대로 출력으로 내보는 것을 항등함수라고 부릅니다. 입력된 것을 그대로 내보내기 때문에 하는 것은 아무것도 없습니다. 수식은 다음과 같습니다.

$$f(x) = x$$

비례 확률 함수

● 비례값 출력

비례확률함수는 신경망에서 사용되지 않습니다. 일정한 값들이 들어왔을 때 그 값들을 크기의 비율로 쓴 것으로 총합이 1이 되게 만든 함수입니다. 이후에 사용할 소프트맥스 함수를 사용하기 위해 만들어보았습니다. 비례확률함수는 수식으로는 다음과 같습니다.

$$y_k = \frac{a_k}{\sum_{all} a_i}$$
$$= \frac{a_k}{a_1 + a_2 + a_3 + \dots + a_n}$$

소프트맥스 함수

● 출력의 비중을 조정

소프트맥스 함수는 출력의 합은 1이 되면서 큰 값의 비중은 더 크게, 작은 값의 비중은 더 작게 만들어주는 특징을 가집니다. 신경망에서 나오는 출력값을 보다 확실하게 하기 위해 원래의 값을 보정해서 확률로 만든 값입니다. 수식은 다음과 같습니다.

$$y_k = \frac{\exp(a_k)}{\sum_{\text{all}} \exp(a_i)}$$
$$= \frac{e^{a_k}}{e^{a_1} + e^{a_2} + e^{a_3} + \dots + e^{a_n}}$$

소프트맥스 함수

● 출력의 비중을 조정

분모와 분자에 상수 c 를 곱한 다음
 사용합니다. 식을 이렇게 변형한 이유는
 $\exp(x)$ 함수의 값이 x 가 조금만 커져도
 함수의 출력값이 상당히 커지기
 때문입니다. 그래서 c 의 자리에 a_i 중
 가장 큰 값을 음의 부호를 붙여서
 넣습니다.

$$\begin{aligned}
 y_k &= \frac{Ce^{a_k}}{Ce^{a_1} + Ce^{a_2} + \dots + Ce^{a_n}} \\
 &= \frac{e^c e^{a_k}}{e^c e^{a_1} + e^c e^{a_2} + \dots + e^c e^{a_n}} \\
 &= \frac{e^{a_k + c}}{e^{a_1 + c} + e^{a_2 + c} + \dots + e^{a_n + c}} \\
 &= \frac{\exp(a_k + c)}{\sum_{all} \exp(a_i + c)}
 \end{aligned}$$

손실함수

손실함수란

○ 항등함수(identity function)

정확도는 학습을 진행할 때 큰 의미를 부여하지 못한다. 이유는 신경망의 결과와 정답의 차이를 보정해주지 못하기 때문입니다. 이를테면 신경망으로 3이 나온 결과값이 0.51인 것과 0.99인 것의 차이가 없기 때문입니다. 그래서 정확도 대신 손실함수라는 것을 사용합니다. 조금 더 정확하게는 $W1$, $W2$, $B1$, $B2$ 의 값이 변화할 때 정확도는 연속적으로 변화하지 않고 단계적으로 변화하기 때문에 좋은 방향을 찾을 수 없다. 이것을 불연속성에 의한 미분불가라고 말합니다. 미분을 할 수 없고, 그래서 기울기를 찾을 수 없어서 학습하기 어렵기 때문에 최종 결과값이 아닌 확률값을 사용하고, 이것을 손실함수로 정의해서 사용합니다.

손실함수란

○ 항등함수(identity function)

y 는 신경망을 통해서 나온 결과값이고 t 는 정답 레이블에 저장된 값입니다.

```
y = [0.01, 0.02, 0.05, 0.10, 0.12, 0.05, 0.05, 0.50, 0.00, 0.10]
t = [ 0,    0,    0,    0,    0,    0,    0,    1,    0,    0]
```


손실함수란

● $10 == 0.1 * 100$?

0.1 을 100번 더하면 어떤값이 되는가?

평균, 중간값, 표준편차, 분산

● 평균, 기댓값

평균과 기댓값은 같은 내용을 다르게 표현하는 것.

표본을 모두 더한 후 그 갯수로 나눈것이 평균이고, 각 표본이 일어날 확률값에 표본의 값을 곱해서 전체에 대해 더한것이 기댓값이다.

$$m = \frac{1}{N} \sum_{i=1}^N x_i$$

$$E(X) = \sum_{i=1}^N x_i P(X = x_i)$$

평균, 중간값, 표준편차, 분산

● 중간값 (median)

중간값은 데이터를 일렬로 늘어놓았을 때 한 중앙에 위치하는 값이다. 데이터의 대표값으로 많이 사용되는 것이 평균과 중간값이다. 데이터에 이상값이 많은 경우 중간값이 자주 사용된다.

평균, 중간값, 표준편차, 분산

표준편차, 분산

- 분산은 전체의 데이터가 평균과 얼마나 멀리 떨어진 곳에 분포하는지를 나타내는 값입니다. 모든 데이터의 값에서 평균을 뺀 다음 제곱한 값의 평균입니다.
- 표준편차는 분산에 제곱근을 씌워준 값입니다.

$$\begin{aligned}
 \sigma^2 &= V(X) \\
 &= E[(x - m)^2] \\
 &= \frac{1}{N} \sum_{i=1}^N (x_i - m)^2 \\
 \sigma &= \sqrt{\sigma^2}
 \end{aligned}$$

평균제곱오차

평균제곱오차

● 출력의 비중을 조정

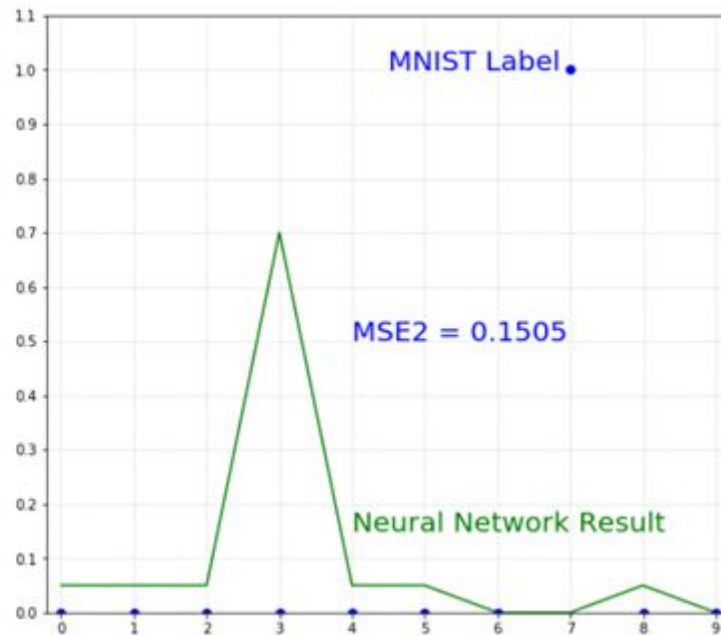
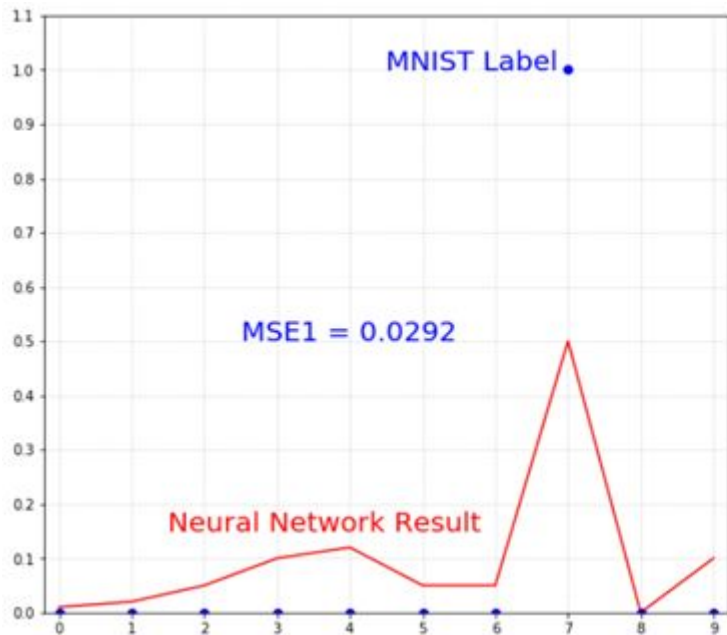
평균제곱오차는 영어로 mean square error이며 줄여서 MSE로 사용합니다. 우리말로 풀어보면 ‘오차들의 제곱을 평균한 것’이라고 볼 수 있습니다.

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - t_i)^2$$

```
y_pred = [0.01 0.02 0.01 0.01 0.5 0 0 0 0 0] - index: 4
t       = [0 0 0 0 1 0 0 0 0 0] - index: 4
MSE = ( 0.01**2 + 0.02**2 + 0.01**2 + 0.01**2 +(0.5-1)**2 )/10
```

평균제곱오차

● 출력의 비중을 조정
$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - t_i)^2$$



크로스엔트로피 오차

크로스엔트로피 오차

● 출력의 비중을 조정

90%의 확률로 질 것이 뻔한 경기에 임했다고 가정을 해보겠습니다. I지역 조기축구회와 축구 명문대학인 A대 축구부가 경기를 했습니다. 여기서 A대 축구부가 이기면 아무도 놀라지 않습니다. 당연한 일이 발생했다고 여기고 그냥 넘어갈 것입니다. 반면 I지역 조기축구회가 이기면 어떻게 될까요? 아주 놀랄 만한 일이 발생한 것입니다. 이때의 놀라는 정도는 9:1에 비례하지 않습니다.

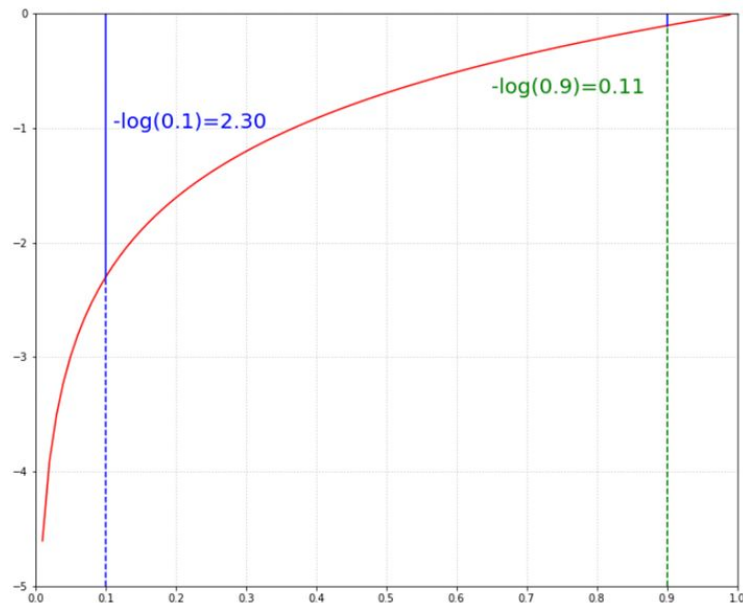
크로스엔트로피 오차

● 출력의 비중을 조정

A대 축구부가 승리할 경우 : $-\log(0.9) = 0.1054$

I지역 조기축구회가 승리할 경우 : $-\log(0.1) = 2.3026$

A대 축구부가 승리할 때의 놀람의 정도는 0.1이고, I지역 조기축구회가 승리할 때의 놀람의 정도는 2.30이 됩니다. 이 수치가 곧 정보의 양이 됩니다. 조기축구회의 승리는 A대 축구부의 승리보다 **21.85배**($=2.3026/0.1054$) 놀람을 가집니다. 여기서 살펴볼 것은 10%의 확률로 이기는 것이 10%의 정보량이나 놀람을 의미하지는 않는다는 것입니다. 더 많은 정보량을 가지게 됩니다.



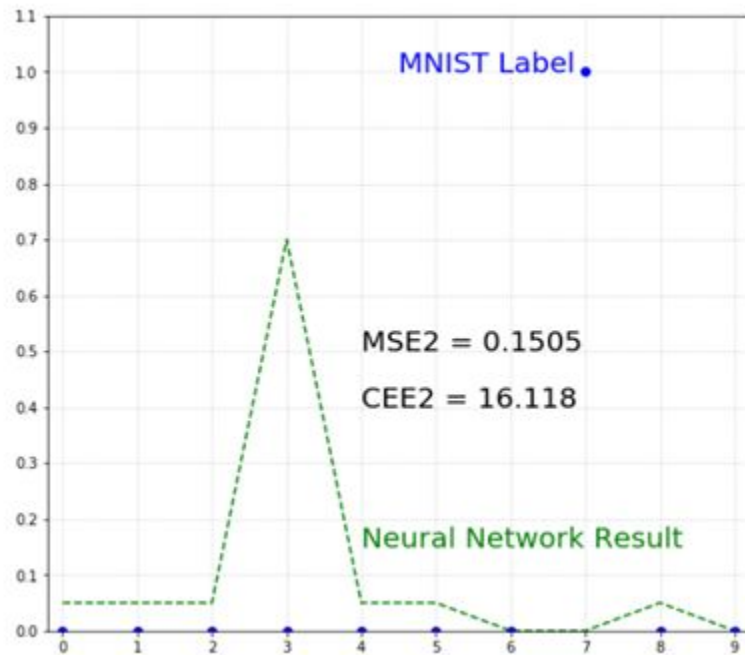
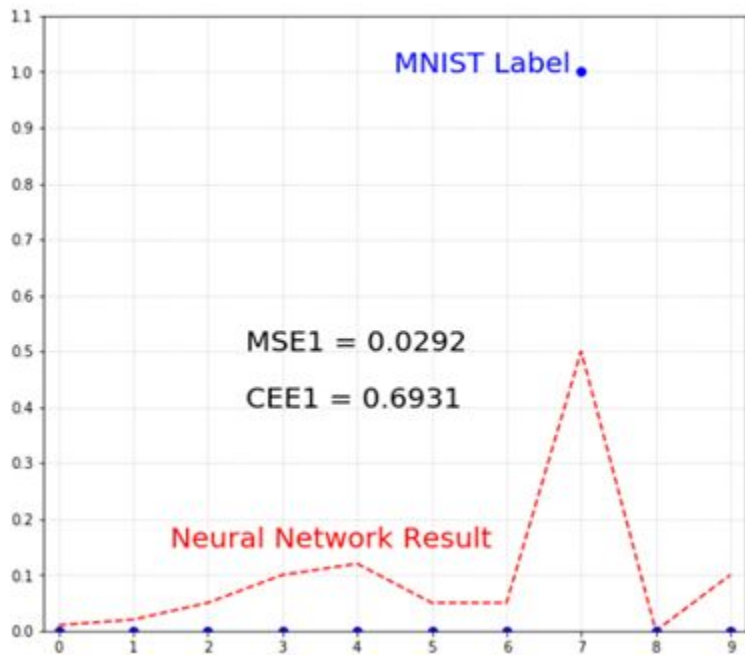
크로스엔트로피 오차

● 출력의 비중을 조정

Case I : 90% 유리 A 대 VS I 조기축구회	Case II : 50 대 50 승부 A 대 VS B 대
<p>엔트로피</p> $S = - \{ 0.9 \times \log(0.9) + 0.1 \times \log(0.1) \}$ $= 0.3251$	<p>엔트로피</p> $S = - \{ 0.5 \times \log(0.5) + 0.5 \times \log(0.5) \}$ $= 0.6931$

크로스엔트로피 오차

○ MSE 와 CEE 비교



경사와 미분

미분의 수학적 정의

● 평균속도

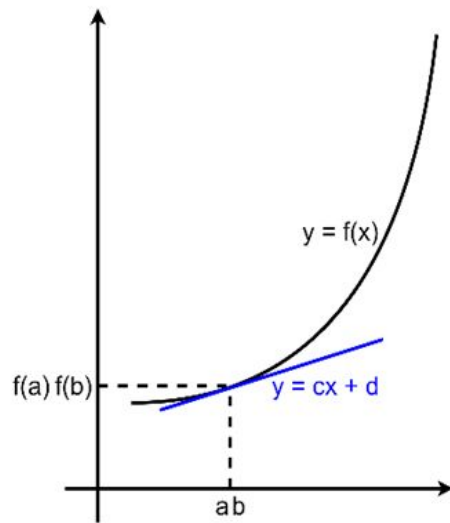
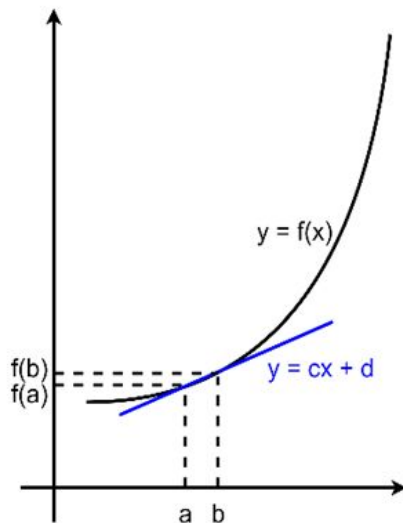
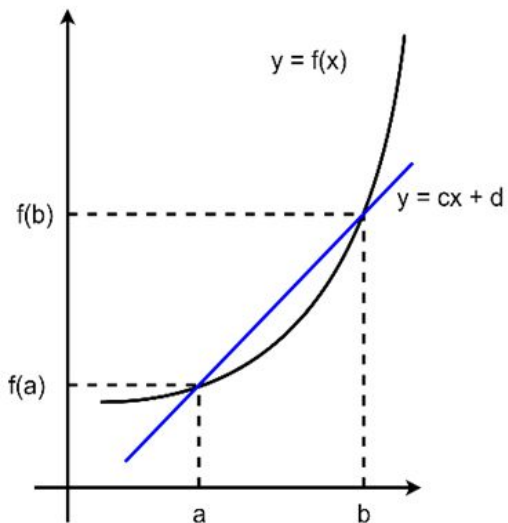
그리스의 철학자 제논은 운동이 불가능하다는 논증을 하면서 어느 한 짧은 순간에서의 위치의 변화는 일어나지 않는다는 것과 시간이 그 짧은 위치 변화가 일어나지 않는 순간들의 연속임을 말합니다. 즉, 움직이고 있다는 것은 모두 아주 짧은 찰나에 보면 정지해 있는 상태란 것입니다. 결국 정지한 상태의 연속일 뿐이니 움직임은 없다는 주장(궤변)이었습니다. 그로부터 2천 년 후 아이작 뉴턴과 라이프니츠는 움직이는 것은 어느 한순간을 보면 분명히 움직이지 않고 있지만 여전히 운동을 나타내는 무언가를 가지고 있다고 생각하고 그것을 찾기 시작했습니다. 그리고 그들이 찾아낸 것이 아주 짧은 시간 동안의 변화였습니다. 이 변화를 알기 위해 움직이고 있는 자동차를 생각해 보겠습니다.

$$v_{av} = \frac{s_2 - s_1}{t_2 - t_1}$$

미분의 수학적 정의

● 속도 = 거리 / 시간

$$c = \frac{f(b) - f(a)}{b - a}$$



미분의 수학적 정의

순간속도

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

수치미분

○ 미분정의

$$\begin{aligned}\frac{df(x)}{dx} &= \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \\ &= \lim_{h \rightarrow 0} \frac{f(x) - f(x-h)}{h} \\ &= \lim_{h \rightarrow 0} \frac{f(x+h) - f(x-h)}{2h}\end{aligned}$$

수치미분

● 도함수

$$f(x) = x^n \rightarrow f'(x) = nx^{n-1}$$

$$f(x) = e^x \rightarrow f'(x) = e^x$$

수치미분

● 곱셈과 덧셈의 도함수

f와 g는 함수이고, c가 상수일 때

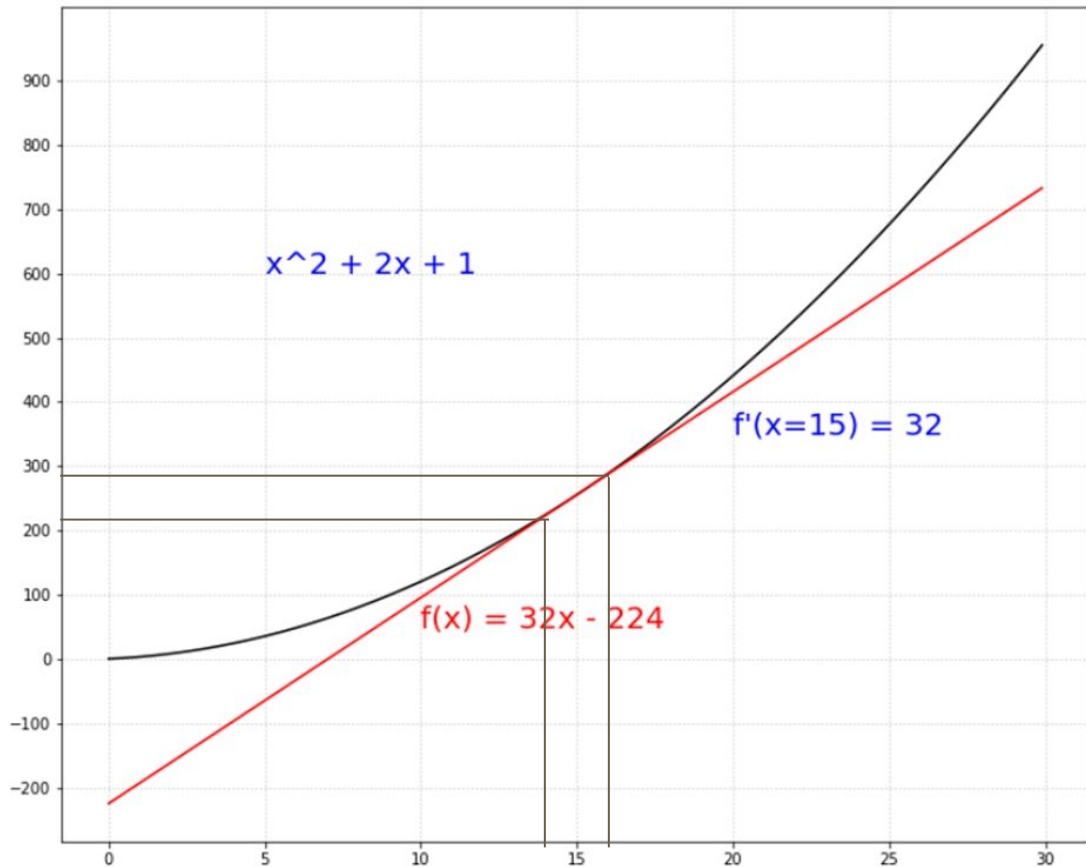
$$(cf)' = cf'$$

$$(f + c)' = f'$$

$$(f + g)' = f' + g'$$

$$(fg)' = f'g + fg'$$

수치미분



$$f(x) = x^2 + 2x + 1$$

$$f'(x) = 2x + 2$$

$$f(15+0.1) - f(15-0.1)$$

$$0.2$$

편미분

● 출력의 비중을 조정

변수가 2개 이상인 함수의 미분

$$f(x,y) = x^2 + y^2$$

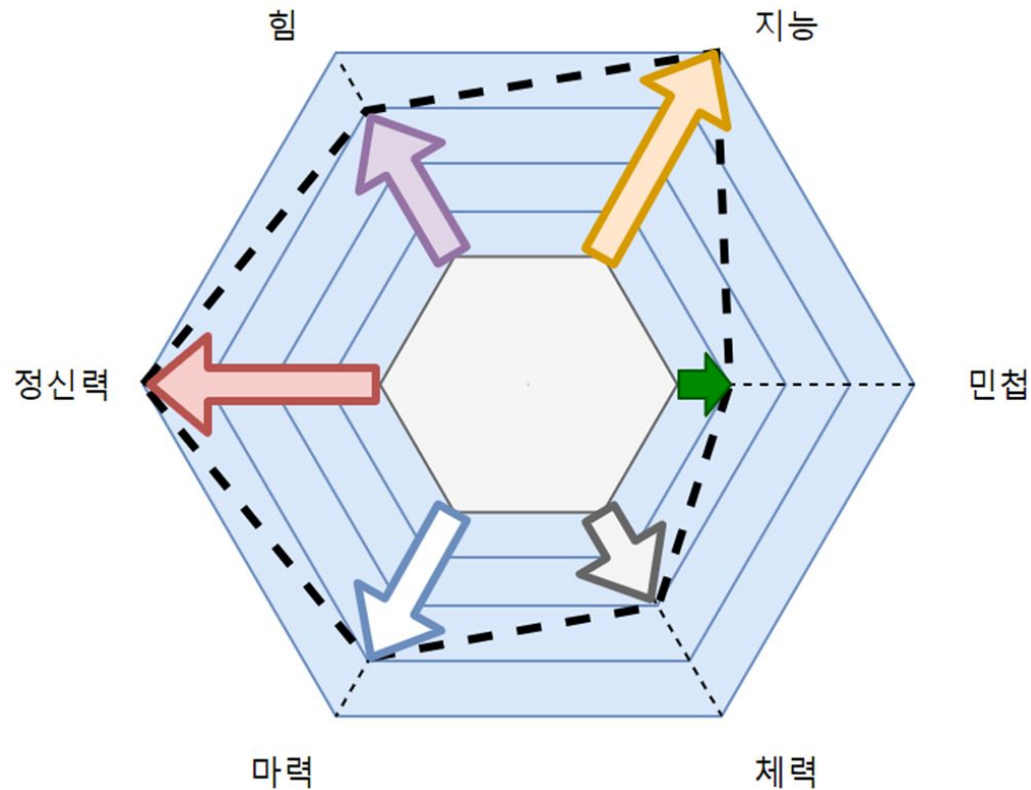
$$f'(x,y) \rightarrow X$$

$f(x,y)$ 를 x 에 대해서 미분 // $f(x,y)$ 를 y 에 대해서 미분..

N차원

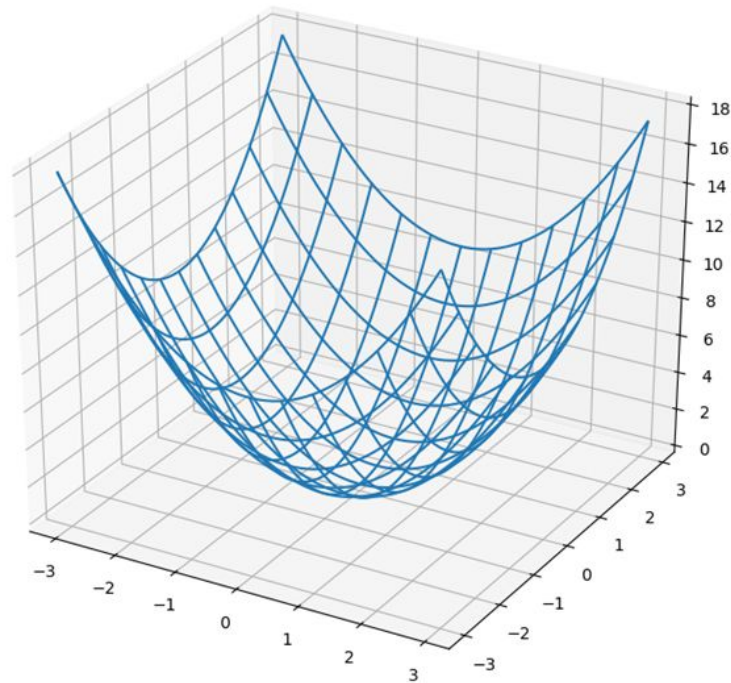
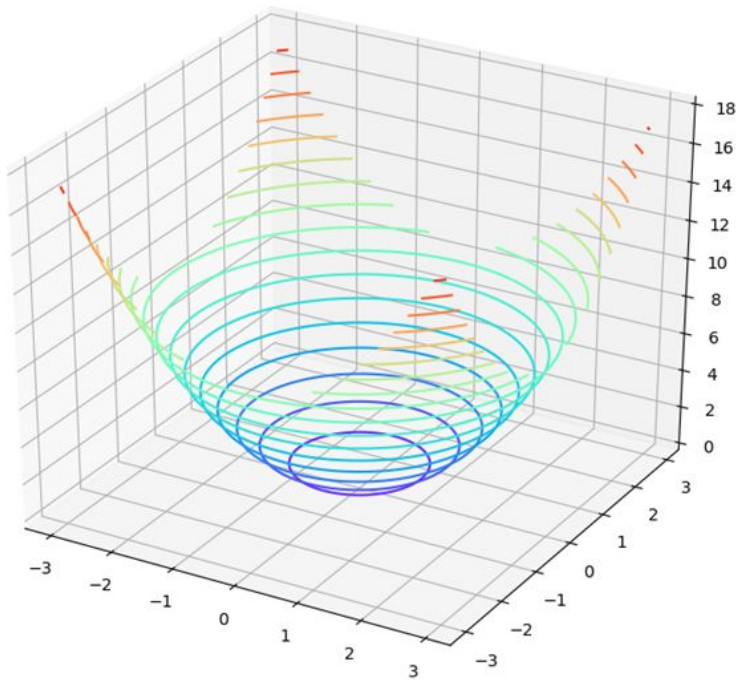
● 게임 캐릭터

게임에서 캐릭터의 힘, 민첩, 지능, 체력, 마력, 정신력을 각각 하나의 변수로 하면 6개의 변수가 되고, 이 6개의 변수에 따라 캐릭터가 가지는 특성이 정해지게 됩니다.



2차원 함수의 그래프와 편미분

● $f(x_0, x_1) = x_0^2 + x_1^2$



2차원 함수의 그래프와 편미분

○ 편미분

이렇게 쓸 수 있습니다. $\partial f(x_0, x_1) / \partial x_0$ 는 x_0 를 변수로 취급하면서 x_1 은 상수로 취급합니다. 즉, $x_0^2 + x_1^2$ 을 미분할 때 x_1^2 은 상수로 취급되기 때문에 x_1^2 의 미분값은 0이 됩니다. 남는 것은 x_0^2 이고, x_0^2 을 미분하면 $2x_0$ 가 됩니다.

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

$$\frac{\partial f(x_0, x_1)}{\partial x_0} = 2x_0$$

$$\frac{\partial f(x_0, x_1)}{\partial x_1} = 2x_1$$

편미분 프로그래밍 코드

● 변수가 2개인 함수의 미분, 편미분

$f(x_0, x_1) = x_0^2 + x_1^2$ 이고, x_0, x_1 이 각각 1과 3일 때 편미분 값을 수치미분 방식으로 구해보겠습니다. 상미분 때에는 변수가 하나였지만 편미분은 변수를 2개 이상 사용합니다. 여기서는 x_0, x_1 의 2개 입니다.

편미분 프로그래밍 코드

● 편미분 파이썬 코드

CODE

```
h = 1e-5
def function_rx(x, y):
    return x**2 + y**2
def ndiff2(f, x, y):
    rf_rx = (f(x+h, y)-f(x-h, y))/(2*h)
    rf_ry = (f(x, y+h)-f(x, y-h))/(2*h)
    return [rf_rx, rf_ry]

ndiff2(function_rx, 1, 3)
```

numpy nditer

numpy nditer

7.1 for 반복문

CODE

```
market = ['apple', 'strawberry', 'grape']  
  
for fruit in market:  
    print(fruit)
```

numpy nditer

for를 이용한 이중 반복문

CODE

```
market = np.array(['apple', 'strawberry', 'grape'], ['water', 'cola', 'orangejuice'])  
for category in market:  
    for item in category:  
        print(item)
```

numpy nditer

🕒 nditer를 이용한 행렬의 멀티인덱스 처리

CODE

```
market = np.array([[ 'apple', 'strawberry', 'grape'],  
                  [ 'water', 'cola', 'orangejuice']])  
  
it = np.nditer(market, flags=['multi_index'], op_flags=['readwrite'])  
while not it.finished:  
    idx = it.multi_index  
    print(market[idx])  
  
it.iternext()
```

numpy nditer

🕒 nditer 사용법

```
it = np.nditer(market, flags=['multi_index'], op_flags=['readwrite'])
```

명령을 통해서 market이라는 행렬의 각 원소의 값과 인덱스를 가져올 수 있습니다. it은 nditer의 객체이고, it 안에 인덱스가 순서대로 들어갑니다. nditer의 객체인 it의 사용법은 아래 세 가지만 기억해 두세요.

- `it.finished` ----- 행렬의 끝이면 True를 아니면 False를 반환
- `it.multi_index` ----- 행렬의 인덱스를 반환
- `it.iternext()` ----- 현재의 인덱스에서 1 증가한 인덱스로 변환

신경망 계산 과정에서의 미분 이해

신경망 계산 과정에서의 미분 이해

○ 노드

입력층 784개 ---- 은닉층 50개 ----- 출력층 10개
 $x(784)$ ----- $h(50)$ ----- $y(10)$

여기서 $x(784)$ 는 상수입니다. 손으로 쓴 글씨의 가로 세로 픽셀값입니다. 이 값은 변하거나 수정되는 것이 아닙니다. 반면 $h(50)$ 의 값은 입력으로 들어온 상수 x 와 변수 w_0 , b_0 에 따라서 결정됩니다. $y(10)$ 도 변수 w_1 과 b_1 과 $h(50)$ 에 따라서 결정됩니다.

신경망 계산 과정에서의 미분 이해

● 편차 w_0 와 바이어스 b_0

입력과 은닉층 사이에 필요한 w_0 와 b_0 는

$$w_0(784, 50), b_0(50)$$

입니다. w_0 는 총 39,200개의 변수를 가지고 있습니다. b_0 의 개수 50개를 더하면 w_0 와 b_0 는 총 39,250개의 변수로 이루어져 있습니다. 여기서 우리는 784개의 상수(x)와 39,250개의 변수(w_0, b_0)의 곱과 합으로 구성된 50개의 결과로 h 가 만들어지는 것을 알 수 있습니다.

신경망 계산 과정에서의 미분 이해

● 연산 순서

`v = affine()` ----- (1) `w0`, `b0`를 이용 [변수의 개수 39,250개]

`h = relu(v)` ----- (2) `v`를 `h`로 변환

`y = affine()` ----- (3) `w1`, `b1`을 이용 [변수의 개수 510개]

`y = softmax(y)` ----- (4) 확률값으로 변환

`y = loss(y,t)` ----- (5) 오차 계산

네트워크 변수의 편미분값인 기울기

네트워크 변수의 편미분값인 기울기

○ 파라미터 크기

입력층(X)의 크기 : A

은닉층(H)의 크기 : B

출력층(Y)의 크기 : C

입력층과 은닉층 사이 w_0 크기 : $A \times B$

입력층과 은닉층 사이 b_0 크기 : B

은닉층과 출력층 사이 w_1 크기 : $B \times C$

은닉층과 출력층 사이 b_1 크기 : C

네트워크변수의 크기 : $A \times B + B + B \times C + C$

네트워크 변수의 편미분값인 기울기

○ 기울기

네트워크변수에서 w_0 의 첫 번째 변수인 $w_0(0,0)$ 변수 하나를 $w_0(0,0)-0.0001$ 로 대체해서 앞 장의 신경망 계산 과정(1)에서 (5)까지를 순차적으로 진행한 후 오차값을 계산합니다. 오차값을 f_1 에 저장한 다음 $w_0(0,0)$ 을 다시 $w_0(0,0)+0.0001$ 로 대체하여 오차값을 계산합니다. 이때의 오차값을 f_2 에 저장합니다. $(f_2-f_1)/(2 \times 0.0001)$ 을 계산하면 바로 미분값이 나옵니다.

$$grad = \frac{f_2 - f_1}{2 \times 0.0001}$$

네트워크 변수의 편미분값인 기울기

● 기울기

변숫값 = 변숫값 - 기울기 × 변숫값 × 학습률

$$\begin{aligned}w0(0,1) &= w0(0,1) - \text{grad}.w(0,1)*w0(0,1)*lr \\ &= 2.381 - 0.4*2.381*0.01\end{aligned}$$

nditer 편미분 코드

● 출력의 비중을 조정

```
import numpy as np
```

```
def numerical_diff(f, x):
```

```
    h = 1e-4    # 0.0001
```

```
    nd_coef = np.zeros_like(x)
```

```
    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
```

```
    while not it.finished:
```

```
        index = it.multi_index
```

```
        tmp = x[index]
```


nditer 편미분 코드

● 출력의 비중을 조정

```
x[index] = tmp + h
fxh2 = f()    # f(x+h)
x[index] = tmp - h
fxh1 = f()    # f(x-h)
nd_coef[index] = (fxh2 - fxh1) / (2*h)
x[index] = tmp
it.iternext()
```

```
return nd_coef
```

경사하강법

경사하강법

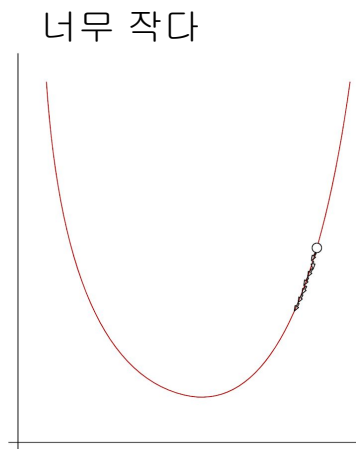
● 손실함수와 경사하강법

미분의 기울기를 이용해서 손실함수의 최솟값을 향해 변수를 변화시키는 것이 바로 ‘ 경사하강법 ’ 입니다. 마치 계곡의 경사를 따라 아래로 내려가면 산에서 마을로 갈 수 있는 것처럼, 정확한 산의 지리를 잘 모르더라도 낮은 곳을 향해 내려가면 가장 낮은 곳에 위치한 강이나 마을을 찾을 수 있습니다. η 는 그리스 알파벳 소문자로 ‘ 에타(eta) ’ 라고 읽고 학습률(learning rate)을 의미합니다.

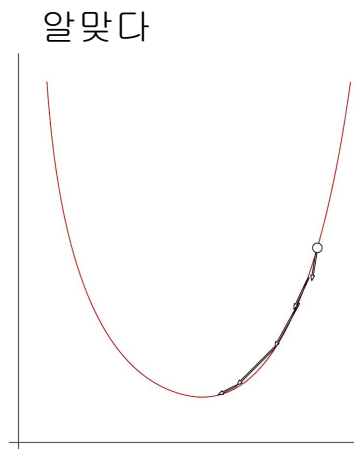
$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

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

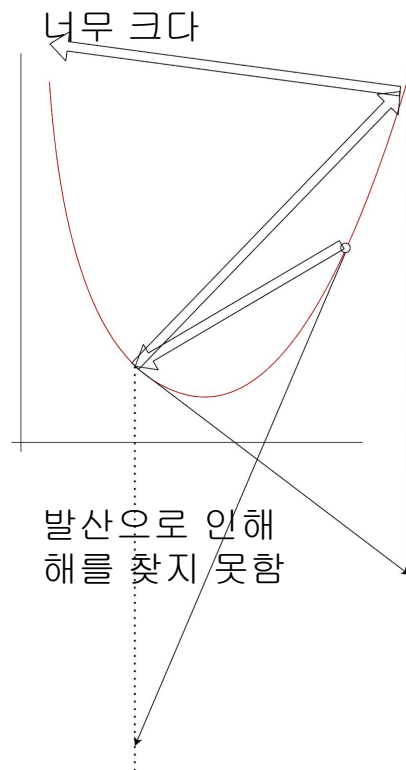
● 학습률



시간이 오래 걸림



적당함



순전파 과정

MNIST 데이터 입력

○ MNIST

keras 데이터셋을 사용

함수정의

○ 함수

미분함수와 함께 시그모이드, 소프트맥스,
크로스엔트로피오차 함수를 만들어 둡니다.

클래스

● Class

Relu, Sigmoid, Affine, SoftmaxWithLoss 클래스를 정의합니다. 앞 클래스들은 설정을 위한 `__init__()` 함수와 실행을 위한 `forward()` 함수를 가집니다. 개별 클래스는 객체를 만들 수 있고, 만들어진 객체는 단계별로 프로세스를 진행할 수 있는 기능함수와 정보를 저장할 수 있는 변수를 가지게 됩니다.

각 클래스에는 `__init__()` 함수와 `forward()` 함수가 있습니다.

`__init__()` 함수를 통해 초기 설정을 하고, `forward()` 함수를 통해 단계별 연산을 앞에서 뒤쪽으로 진행할 수 있습니다.

네트워크 클래스

SimpleNetwork

앞에서 만든 프로세스별 클래스를 사용하기 위해 네트워크변수를 담고 있는 네트워크클래스를 만들어보겠습니다. 초깃값 설정을 위해 `__init__()` 함수를 가지고 있고, 프로세스를 진행하기 위한 `predict()` 함수와 오차값을 계산할 수 있는 `loss()` 함수, 정확도를 계산할 수 있는 `accuracy()` 함수, 편미분을 사용해서 네트워크변수의 기울기를 구할 수 있는 `numerical_gradient()` 함수를 가지고 있습니다.

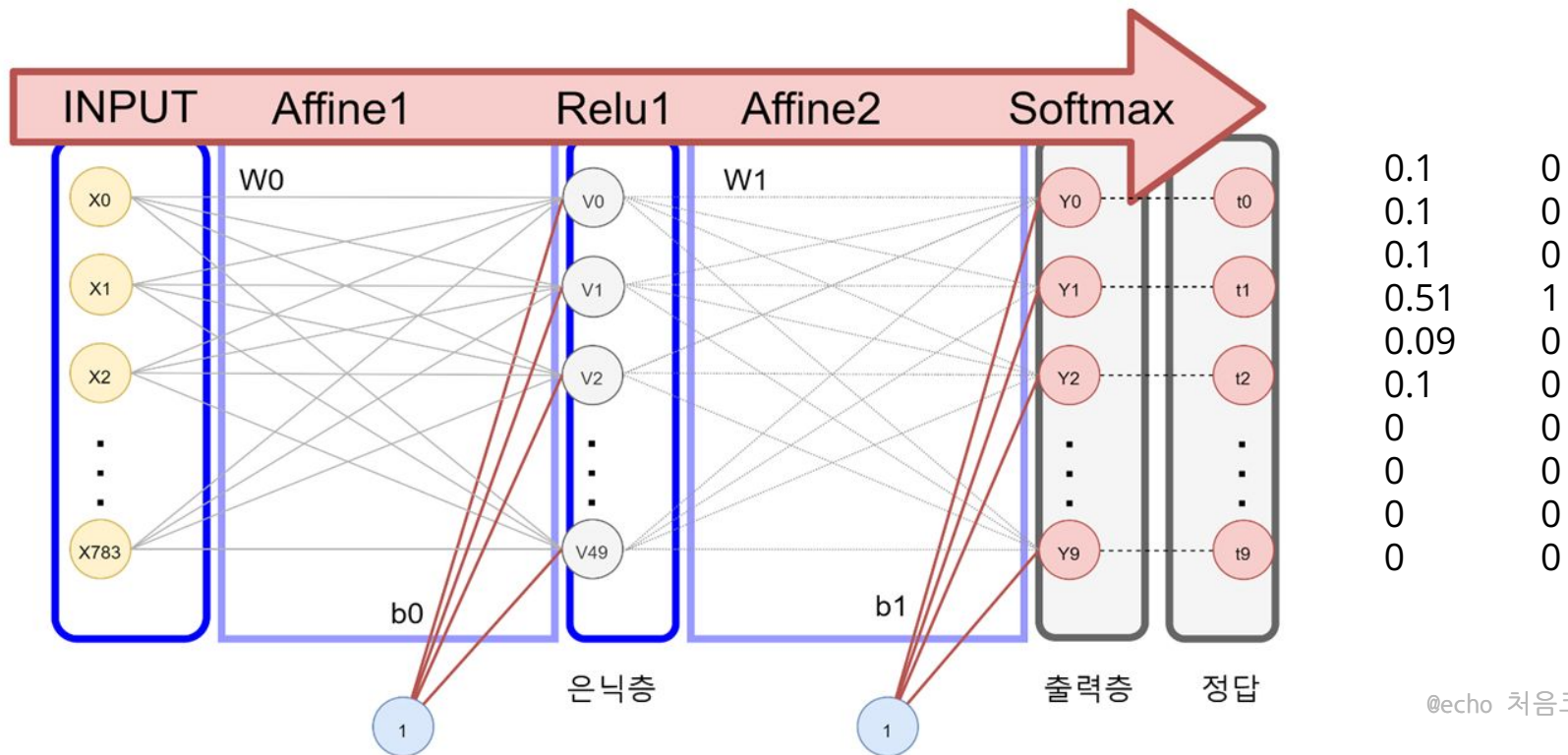
네트워크 클래스의 이름을 `SimpleNetwork`로 하겠습니다.

네트워크 클래스

SimpleNetwork

```
class SimpleNetwork:
    def __init__(self, 인자1, 인자2, ...):
        ...
    def predict(self, 인자1, 인자2, ...):
        ...
    def loss(self, 인자1, 인자2, ...):
        ...
    def accuracy(self, 인자1, 인자2, ...):
        ...
```

순전파(forward)



학습과 검증

● 순전파(forward)

배치(batch), 에포크(epoch) 라는 단어는 인공지능 학습 분야에서 자주 이야기되는 단어들입니다. 간단하게 뜻을 이해하고 넘어가겠습니다.

배치 : PC 에서 처리할 수 있도록 빅데이터를 어느 정도의 크기로 분산하여 한번에 처리할 수 있는 정도로 만든 데이터의 묶음

에포크 : 전체 데이터의 수만큼 1회 처리하는 것

이터레이션(iteration)은 1 에포크 진행에 필요한 반복 횟수

정확도 테스트

○ 실행결과

위 코드의 실행 결과는 다음과 같습니다. 전체를 출력하는 것은 큰 의미가 없으므로 일부만을 기록합니다.

```
loss = _____ time = _____ n = _____ | [TrainAcc] [TestAcc]
loss =  2.1689   time =  14.4418   n = 000001 |   0.1100   0.1096
loss =  2.6437   time =  29.0018   n = 000002 |   0.1415   0.1397
loss =  1.7238   time =  43.5033   n = 000003 |   0.1693   0.1727
....
```

역전파와 계산그래프

계산그래프

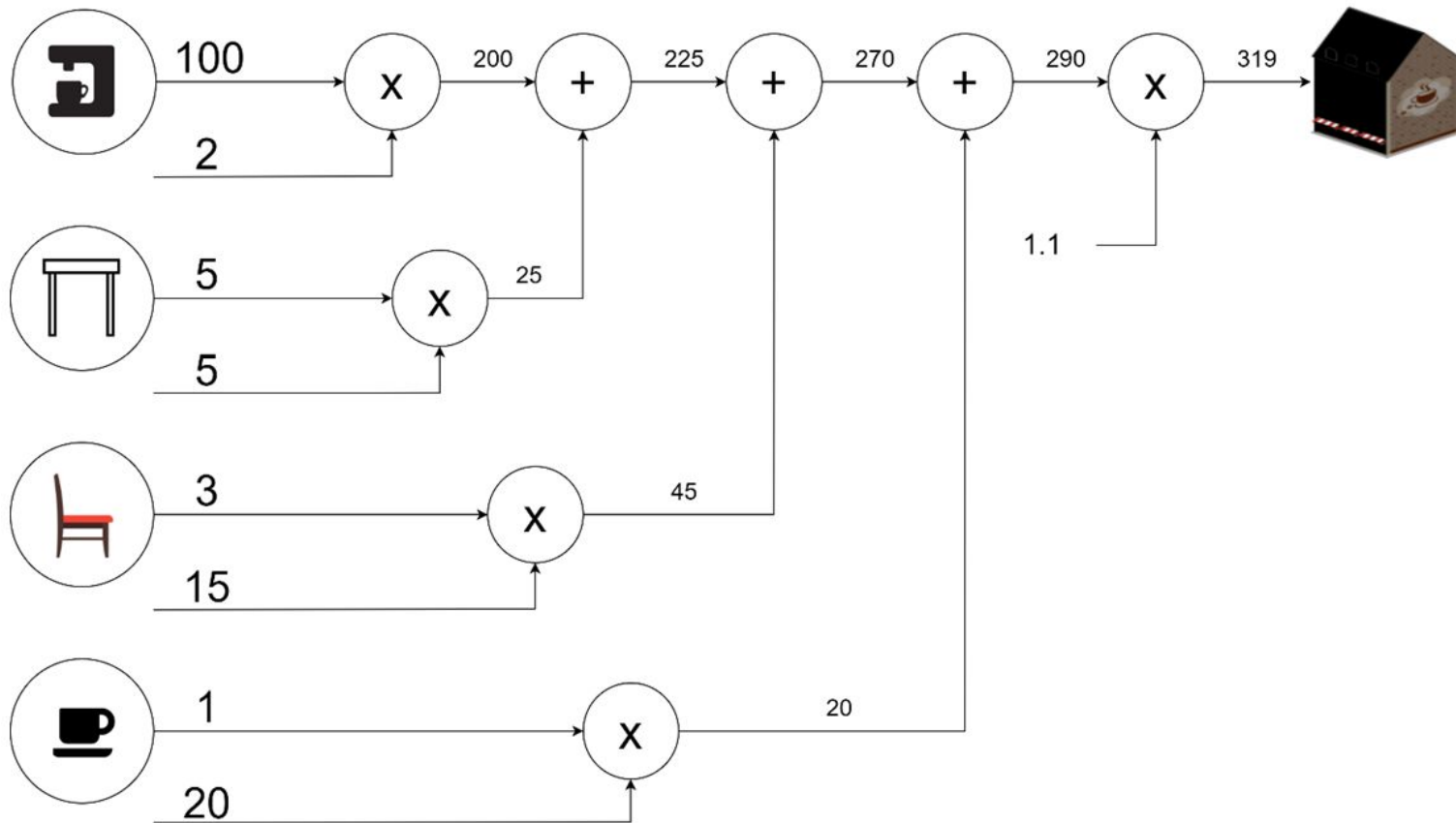
○ 계산 그래프

계산그래프는 계산 과정을 그래프로 나타낸 것으로 원으로 표시되는 정점(**node**), 정점과 정점을 연결하는 간선(**edge**)으로 구성됩니다. 보통 정점이라는 말 대신 노드를 그냥 사용하는 경우가 많아서 앞으로는 노드와 간선이라는 말을 사용하겠습니다. 그래프를 사용하는 이유로 빠른 이해가 있습니다. 그래프를 이용하여 표기하면 보다 직관적으로 이해할 수 있습니다. 간단한 문제를 풀어보겠습니다.

계산그래프

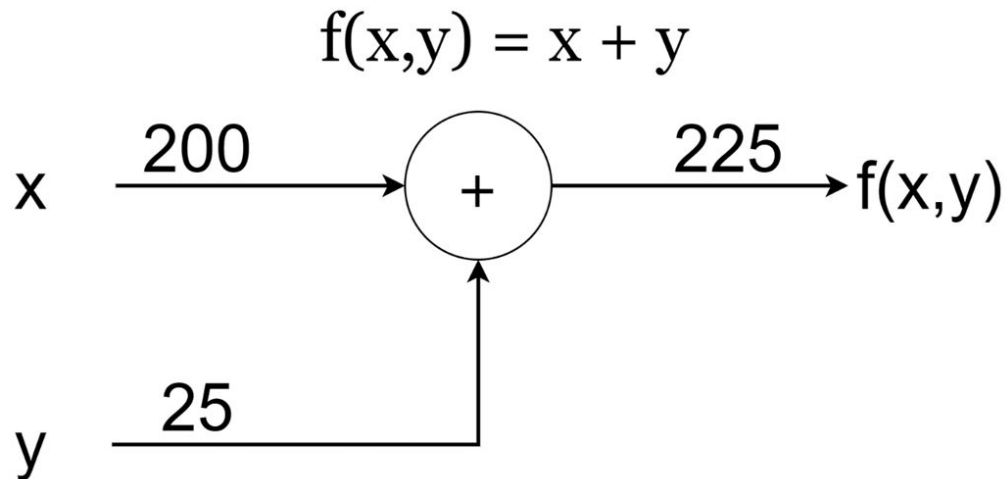
○ 계산 그래프

길동은 카페 창업을 하기 위해 100만 원짜리 커피머신 2개, 5만 원 테이블 5개, 3만 원 의자 15개, 1만 원 컵 20개를 구입했습니다. 세금 10%를 포함해서 창업에 든 비용을 계산해 봅시다. 이 문제를 계산그래프로 표현하면 다음과 같습니다.



계산그래프

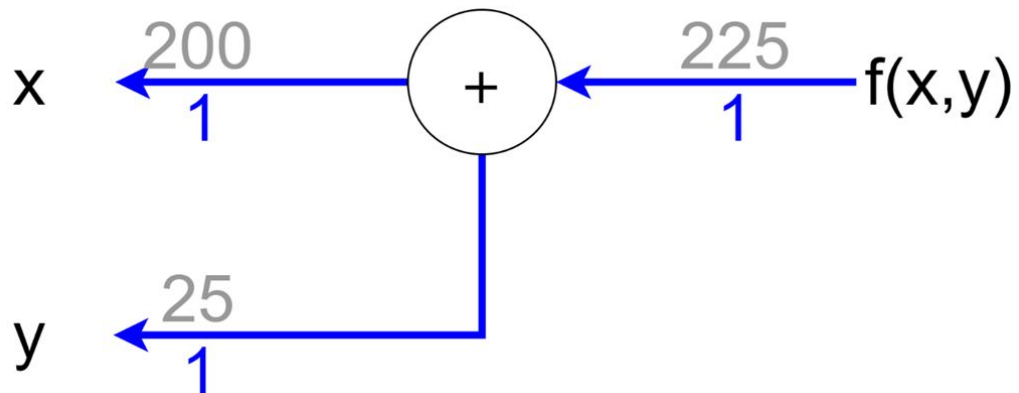
○ 덧셈노드 순전파



$$f(x,y) = x + y \Rightarrow \frac{\partial f}{\partial x} = 1, \quad \frac{\partial f}{\partial y} = 1$$

계산그래프

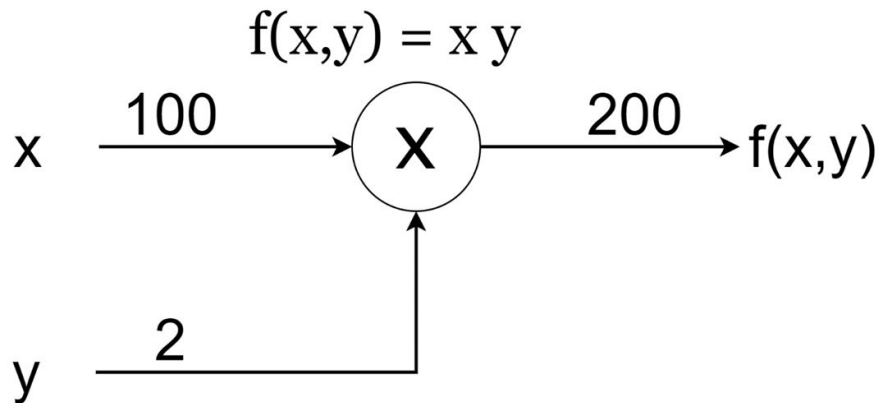
○ 덧셈노드 역전파



$$f(x, y) = x + y \Rightarrow \frac{\partial f}{\partial x} = 1, \frac{\partial f}{\partial y} = 1$$

계산그래프

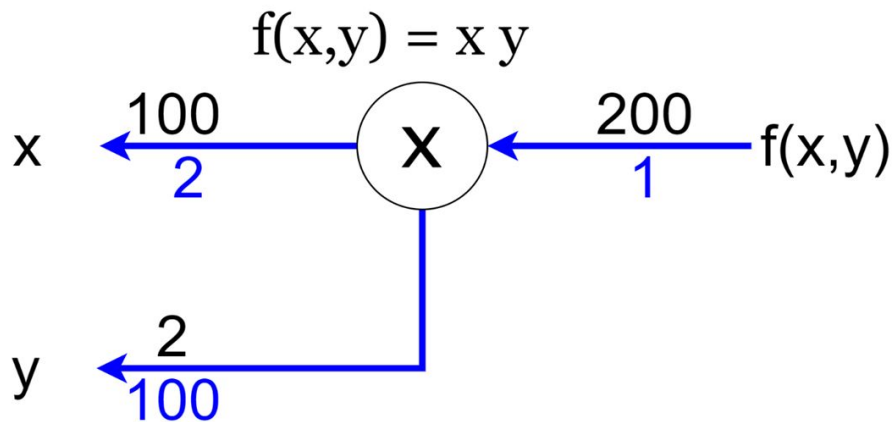
● 곱셈노드 순전파



$$f(x,y) = xy \quad \Rightarrow \quad \frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x$$

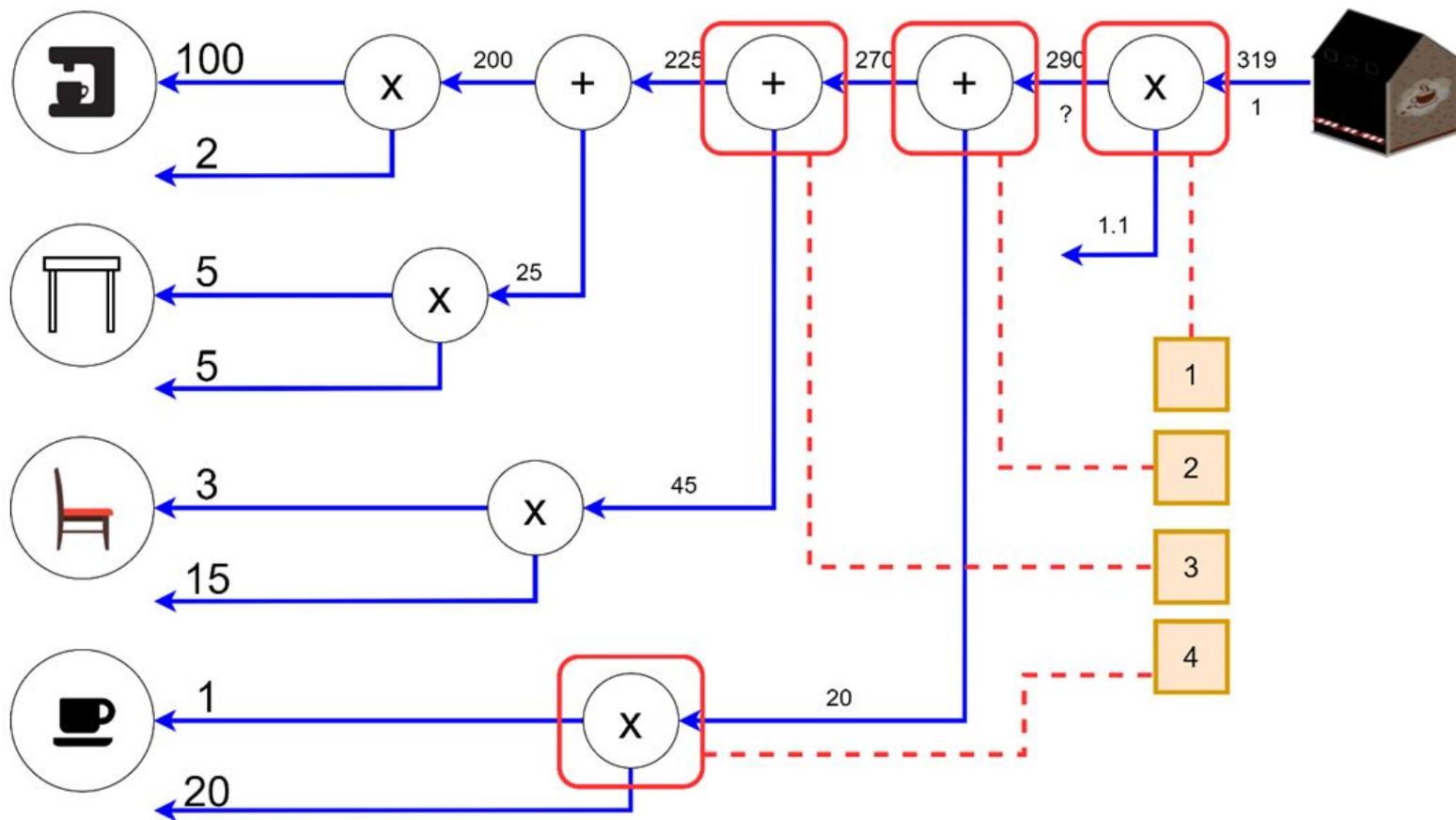
계산그래프

● 곱셈노드 역전파



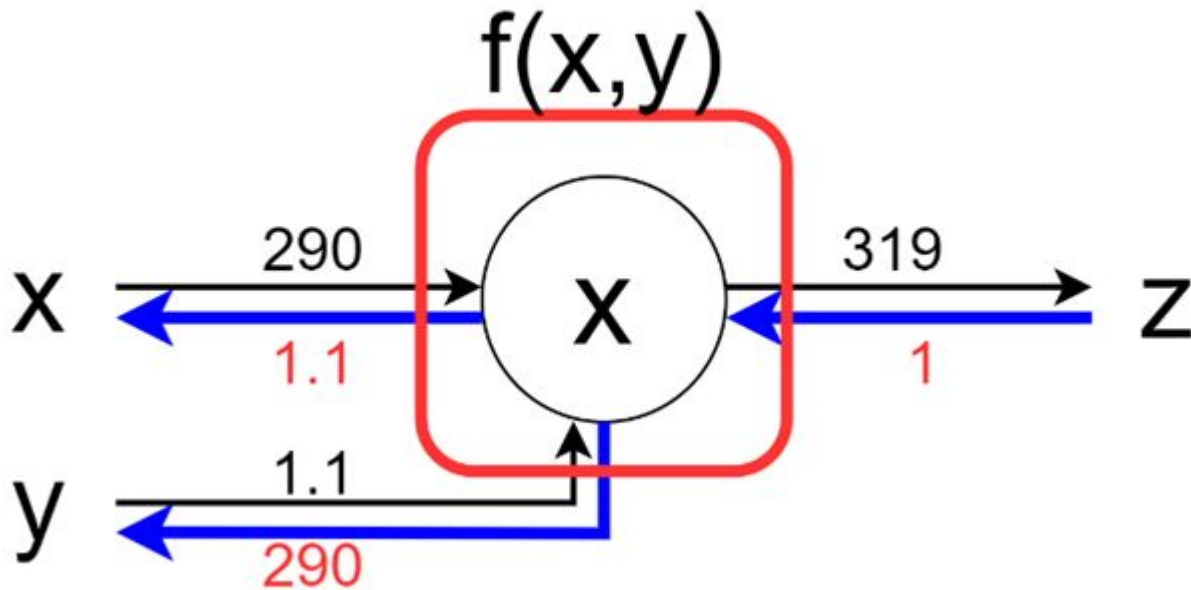
$$f(x,y) = xy \quad \Rightarrow \quad \frac{\partial f}{\partial x} = y, \quad \frac{\partial f}{\partial y} = x$$

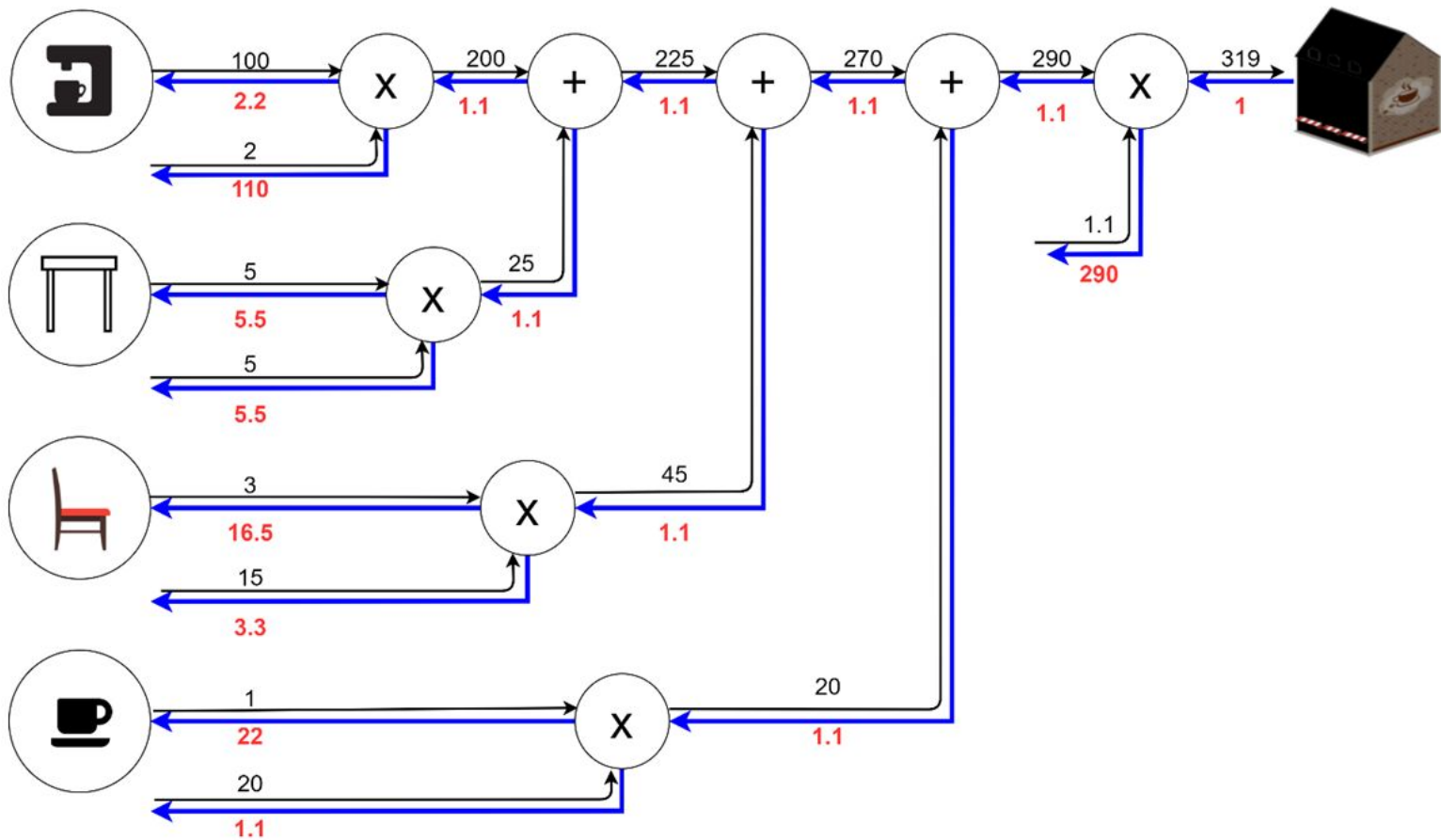
계산그래프



계산그래프

카페창업 #1 - 곱셈노드 & 역전파





계산그래프

● 체인룰(Chain Rule)

덧셈, 곱셈 등과 위의 미분함수들은 개별적으로는 그다지 어렵지 않습니다. 하지만 서로 복합적으로 연결되면 복잡하게 보이게 됩니다. 이런 복잡한 함수를 단순화시키기 위해서 체인룰을 사용합니다.

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

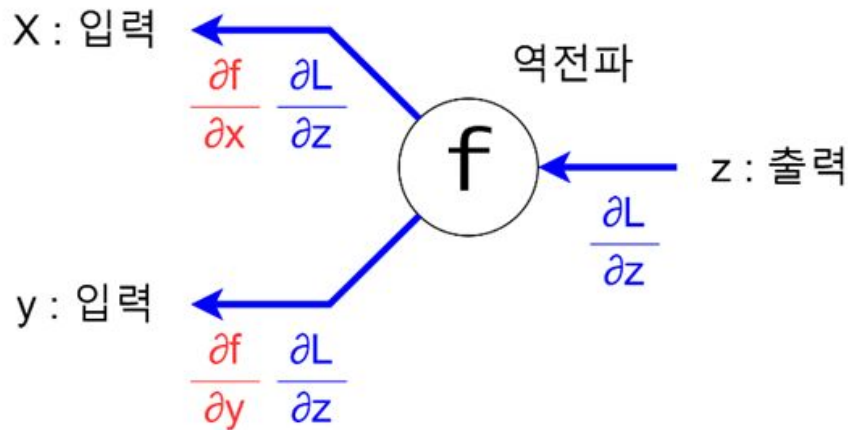
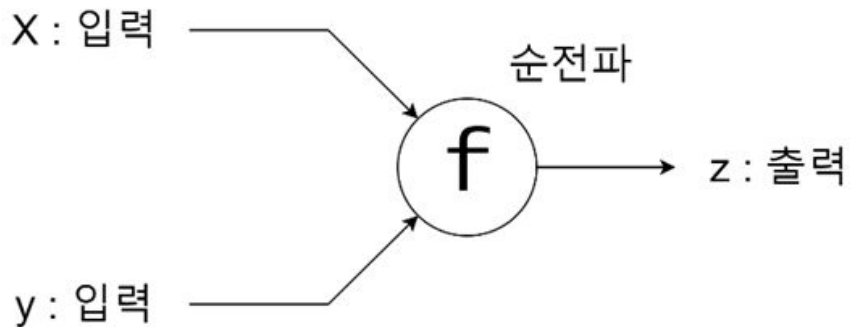
$$F'(x) = f'(g(x)) g'(x)$$

Differentiate
outer function

Differentiate
inner function

계산그래프

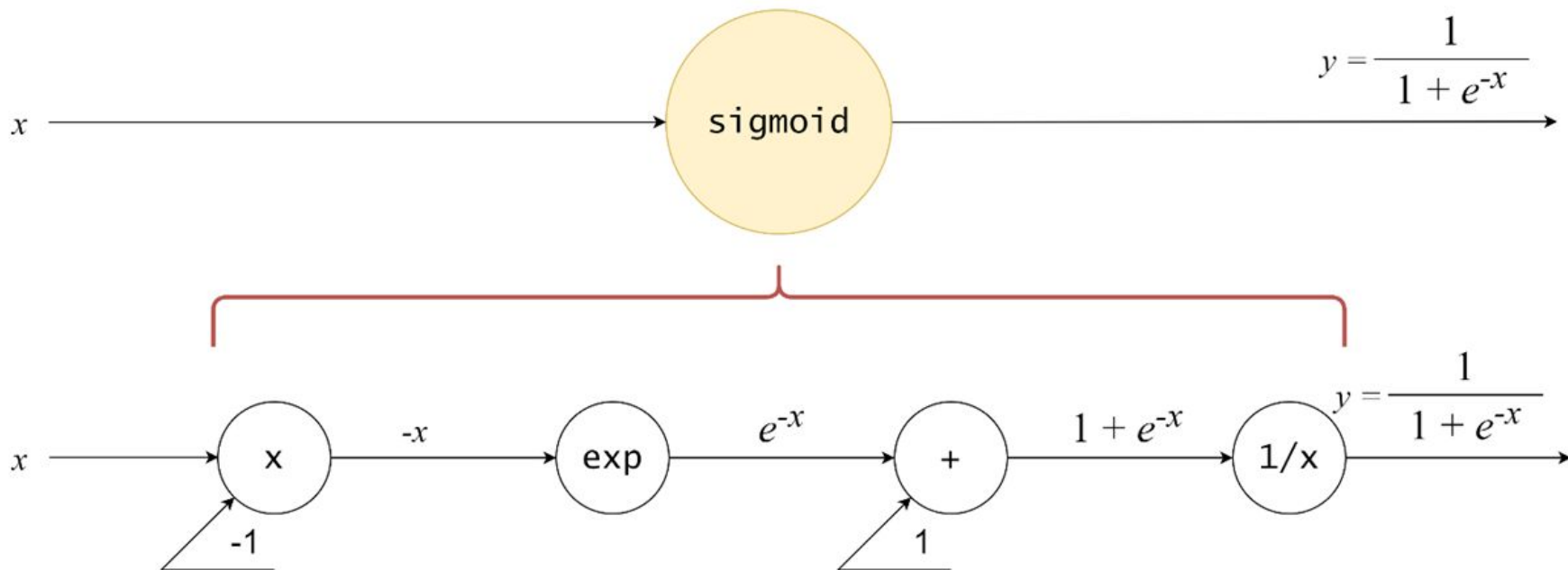
함수의 기울기와 계산 그래프



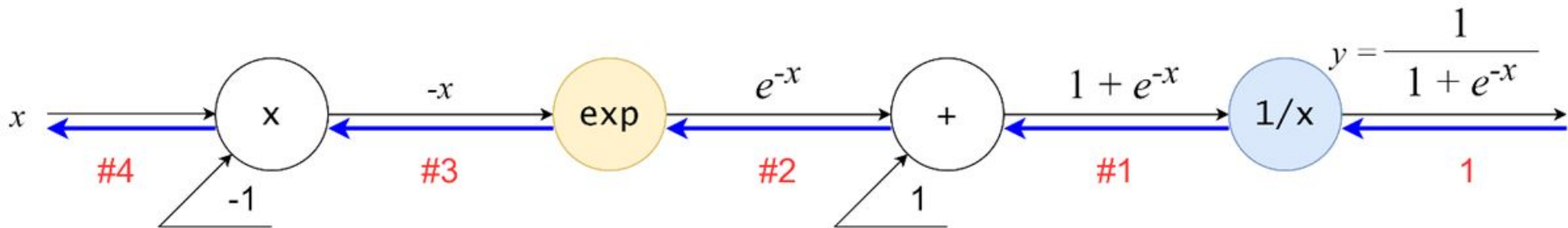
시그모이드 기울기

시그모이드 기울기

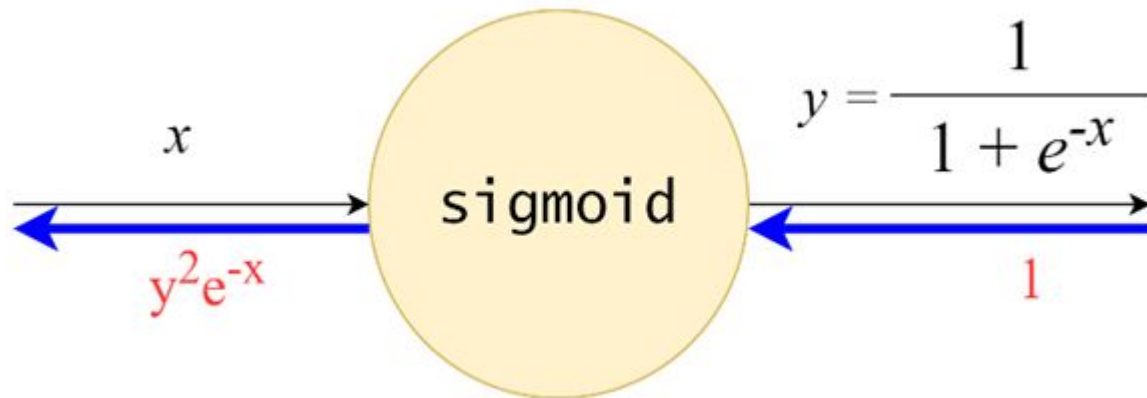
● 시그모이드



시그모이드



$$\begin{aligned} \#1 &= -y^2 \\ \#2 &= -y^2 \\ \#3 &= -y^2 \times e^{-x} \\ \#4 &= y^2 \times e^{-x} \end{aligned}$$



시그모이드 기울기

● 시그모이드

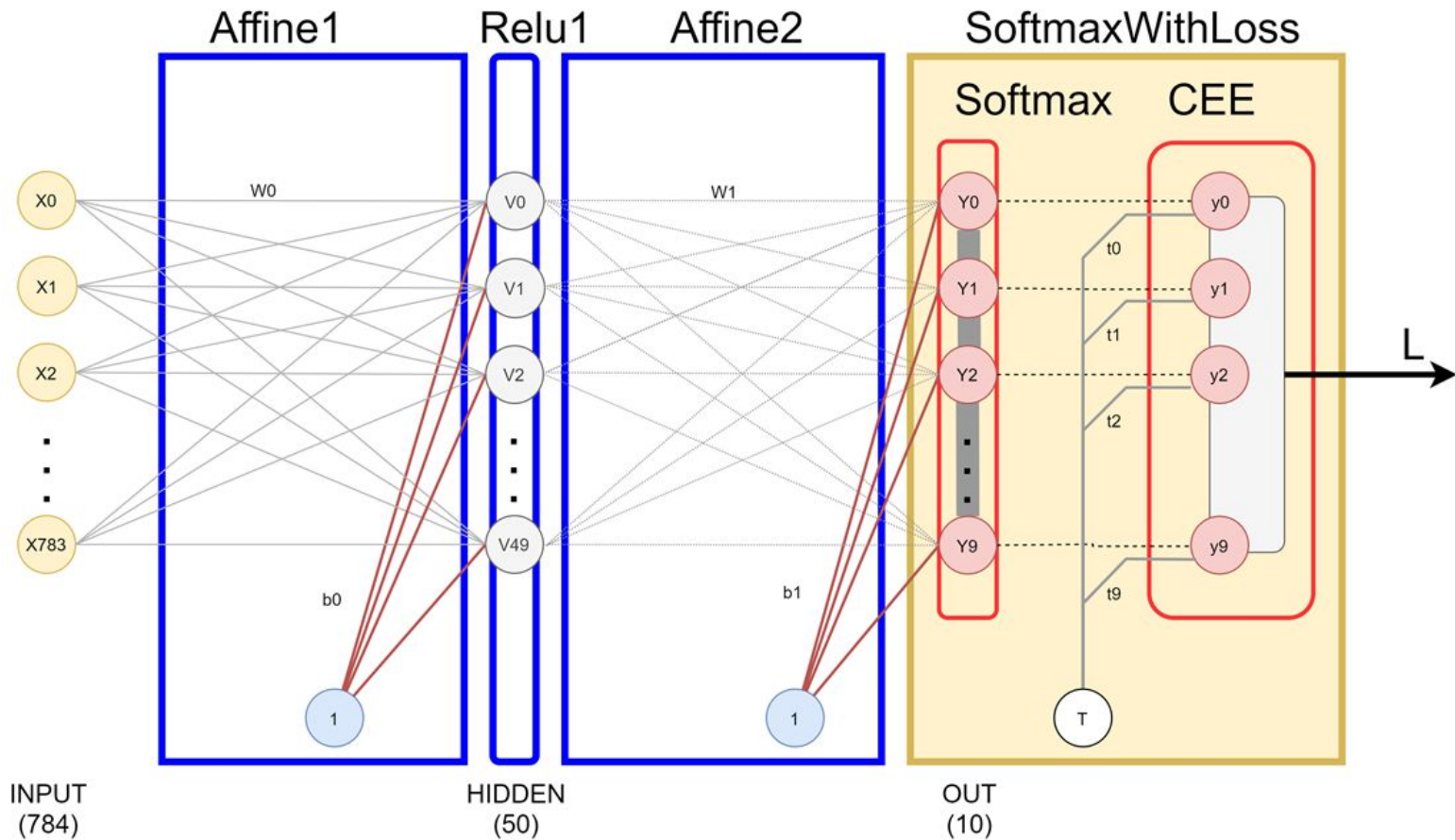
$$\#1 = -y^2$$

$$\#2 = -y^2$$

$$\#3 = -y^2 \times e^{-x}$$

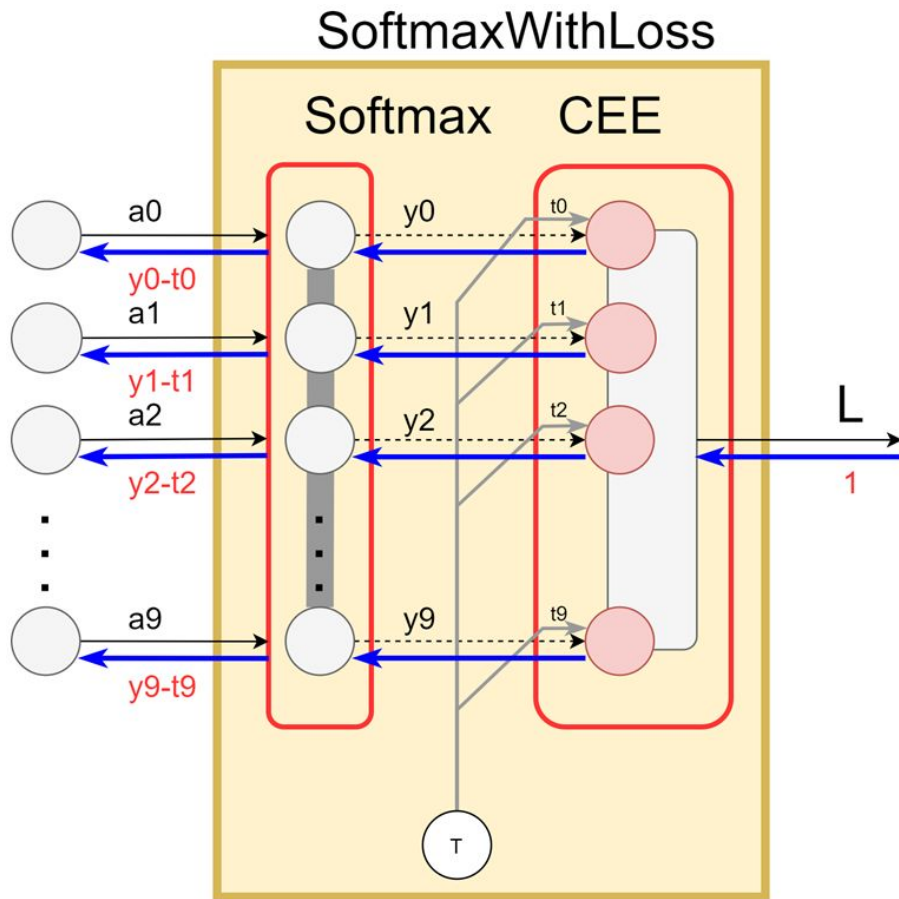
$$\#4 = y^2 \times e^{-x}$$

$$\begin{aligned} y^2 e^{-x} &= \frac{1}{(1 + e^{-x})^2} e^{-x} \\ &= \frac{1}{1 + e^{-x}} \frac{e^{-x}}{1 + e^{-x}} \\ &= \frac{1}{1 + e^{-x}} \left(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right) \\ &= y(1 - y) \end{aligned}$$



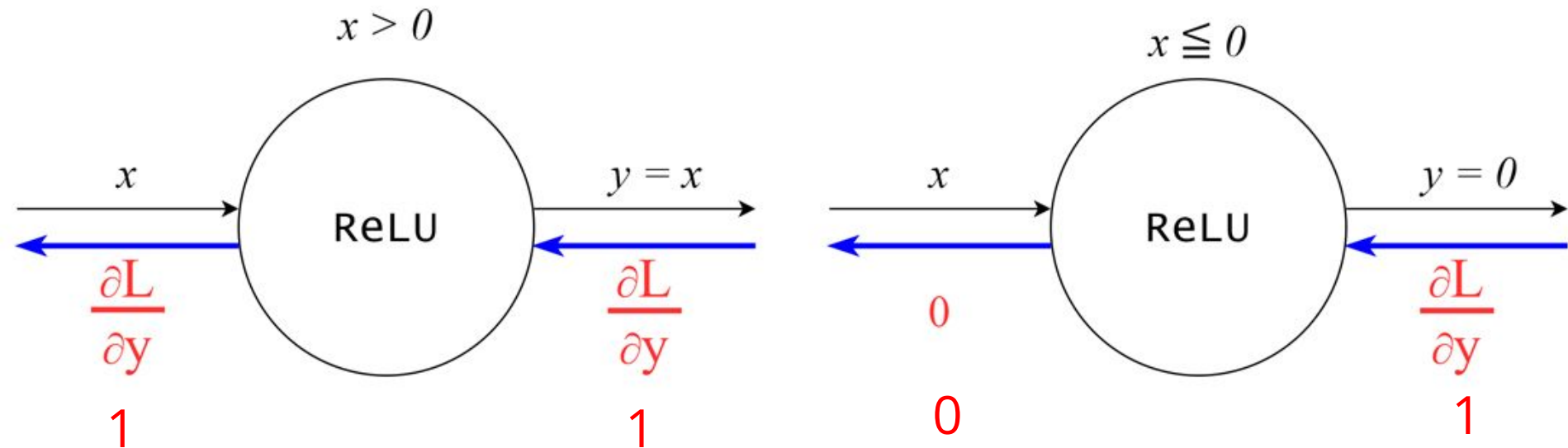
소프트맥스와 CEE

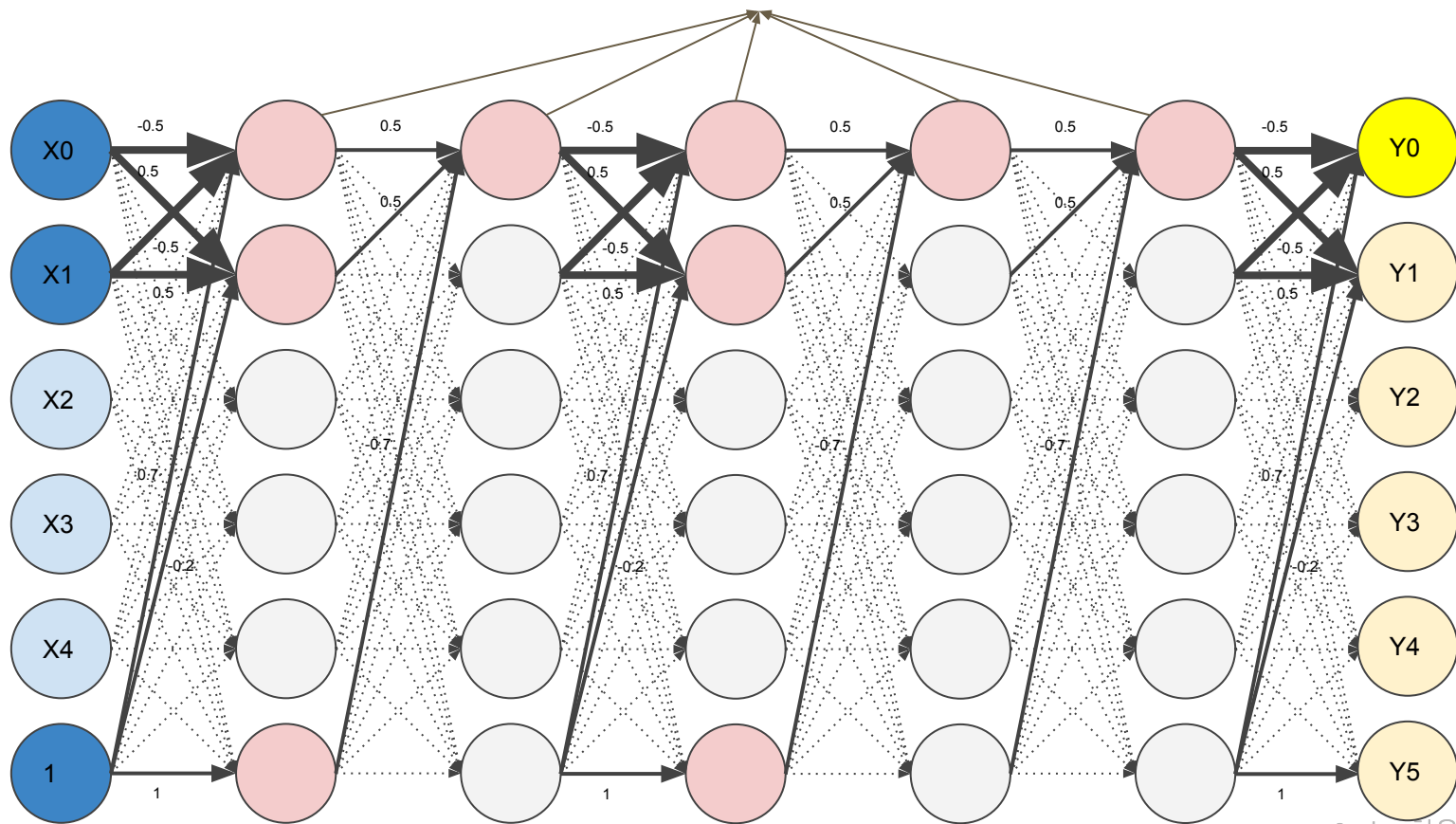
소프트맥스 역전파



활성함수

ReLU 역전파





Affine 계층/레이어

Affine 계층은 ...

행렬 내적을 기하학에서 어파인 변환(**affine transformation**)이라고 하고, 어파인 변환을 수행하는 처리를 **Affine** 계층이라는 이름으로 만든다. 즉, 이전 계층의 모든 뉴런과 연결되어 있어 행렬의 내적(**np.dot()**)을 사용하여 계산하는 계층/레이어를 **Affine** 계층/레이어라 부른다.

```
>>>import numpy as np
>>>X = np.random.rand(2)
>>>W = np.random.rand(2,3)
>>>B = np.random.rand(3)
>>>Y = np.dot(X,W) + B
```

Affine 계층

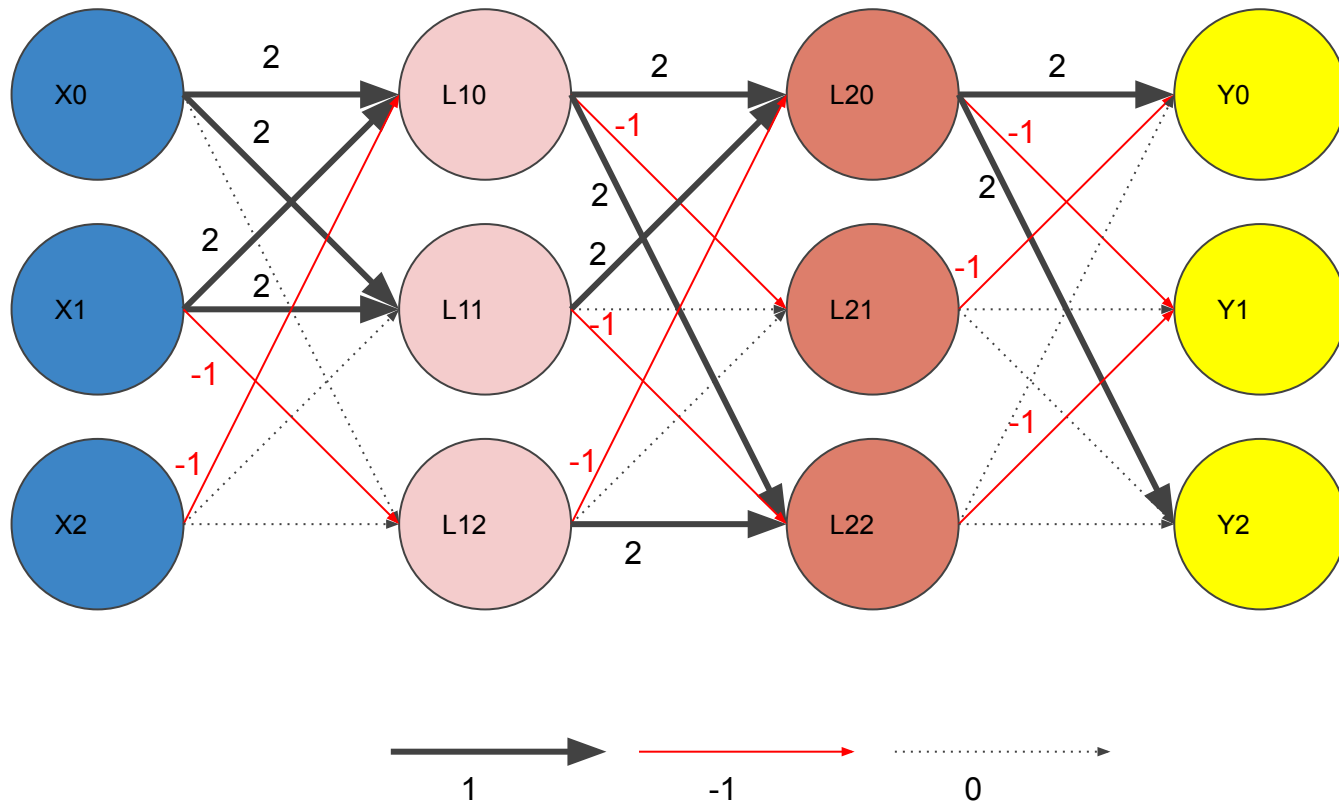
○ 행렬계산 (순전파)

$$\begin{matrix} X & \cdot & W & = & O \\ (2,) & & (2,3) & & (3,) \end{matrix}$$

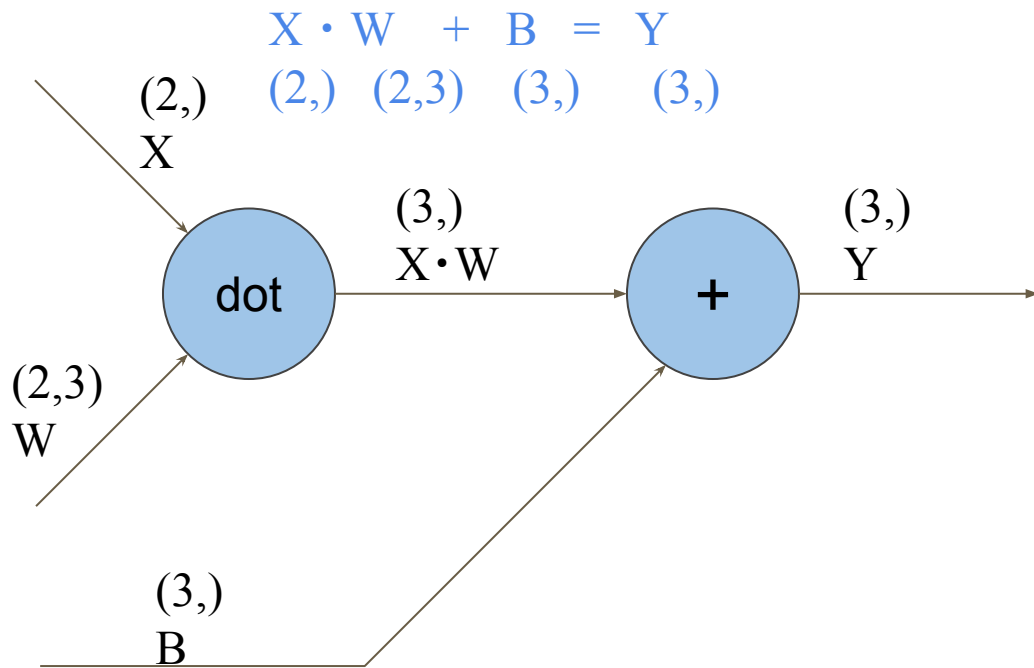
Affine 계층

○ 행렬계산 (순전파)

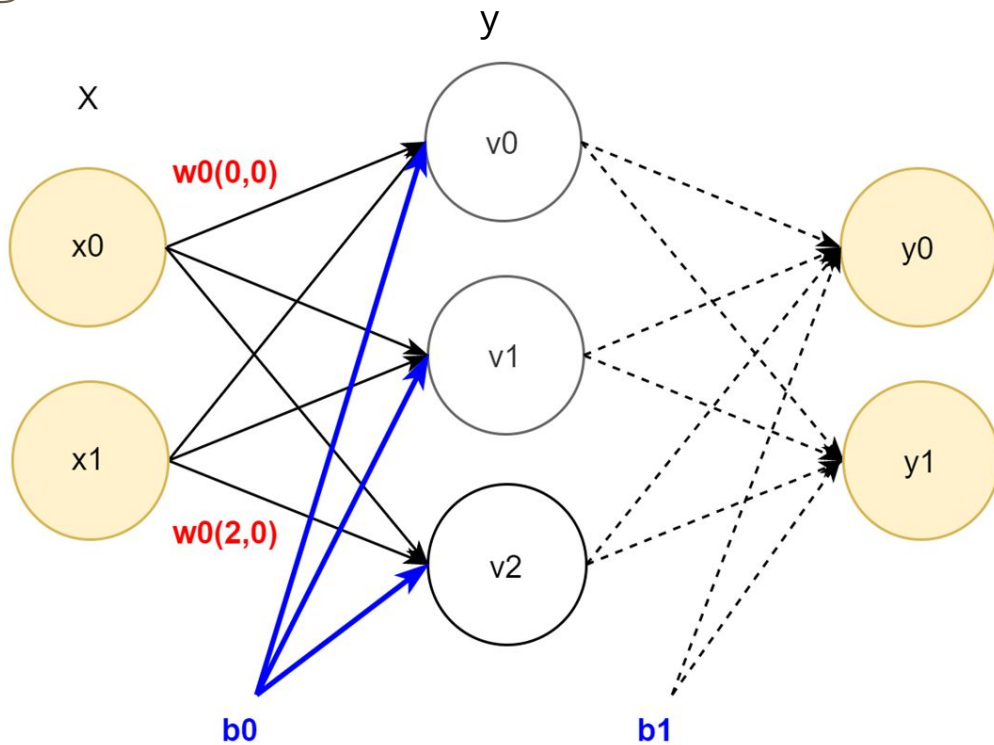
$$\begin{matrix} X & \cdot & W & = & O \\ (2,) & & (2,3) & & (3,) \\ (1,2) & & (2,3) & & (1,3) \end{matrix}$$



Affine 계층/레이어 (순전파)



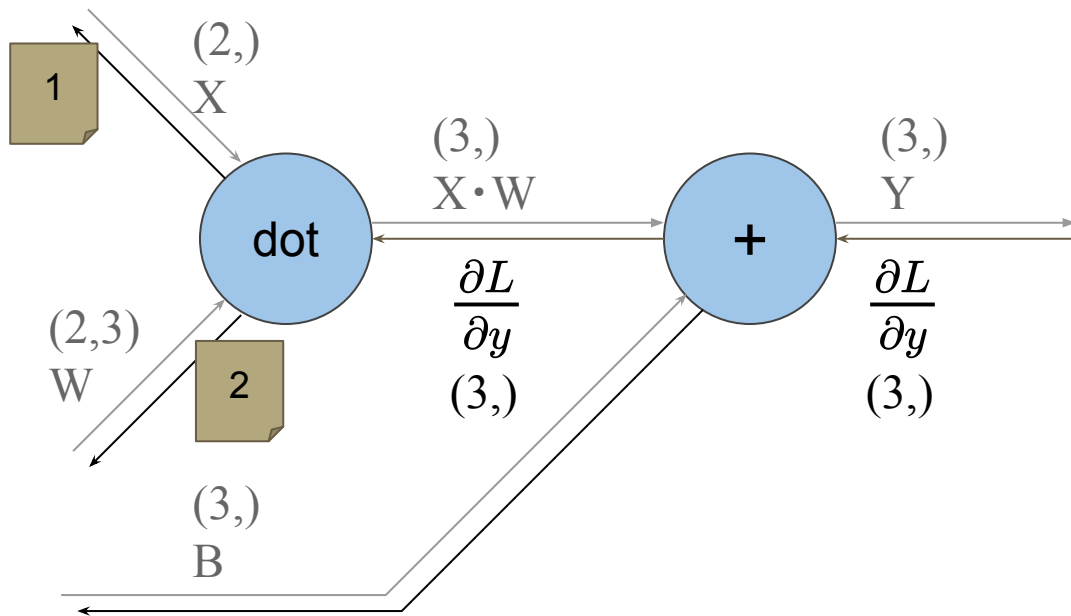
어파인 계층



Affine 계층/레이어

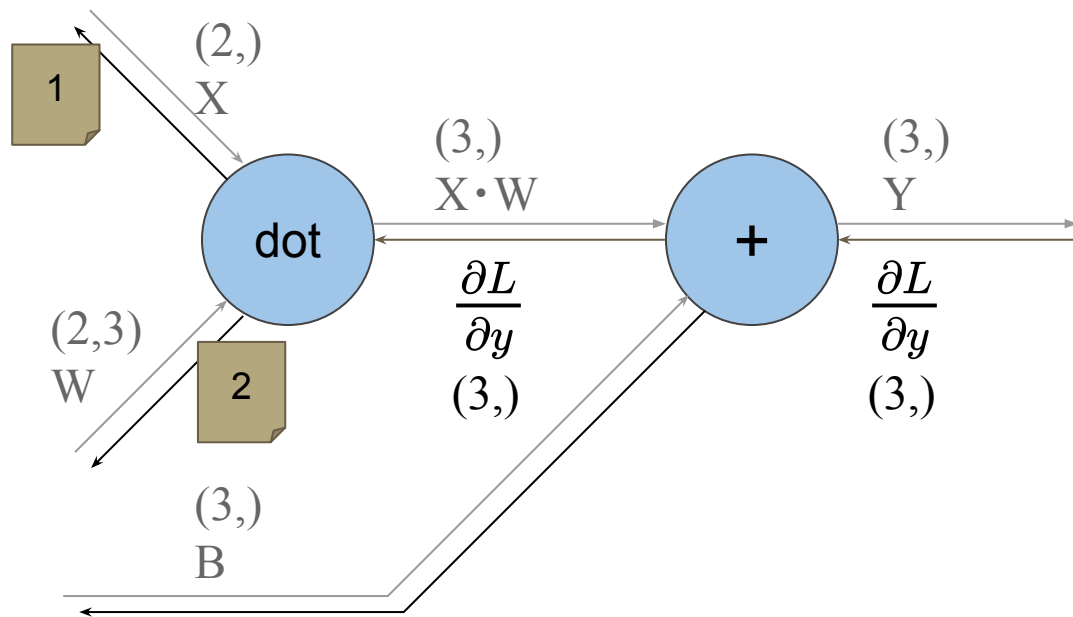
$$X \cdot W + B = Y$$

(2,) (2,3) (3,) (3,)



Affine 계층/레이어

$$\begin{matrix}
 X & \cdot & W & + & B & = & Y \\
 (2,) & & (2,3) & & (3,) & & (3,)
 \end{matrix}$$



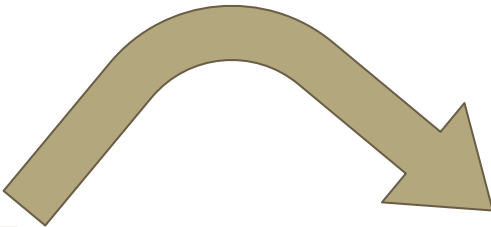
1

$$\begin{matrix}
 \frac{\partial L}{\partial X} & = & \frac{\partial L}{\partial Y} \cdot W^T \\
 (1,2) & & (1,3) \quad (3,2)
 \end{matrix}$$

2

$$\begin{matrix}
 \frac{\partial L}{\partial W} & = & X^T \cdot \frac{\partial L}{\partial Y} \\
 (2,3) & & (2,1) \quad (1,3)
 \end{matrix}$$

전치행렬


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

전치행렬

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

$$(1,2) \quad (1,3) \quad (3,2)$$

$$(lx_{11} \quad lx_{12}) = (ly_{11} \quad ly_{12} \quad ly_{13}) \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix}$$

$$(1,2) \quad (1,3) \quad (3,2)$$

전치행렬

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

(1,2) (1,3) (3,2)

$$(lx_{11} \quad lx_{12}) = (ly_{11} \quad ly_{12} \quad ly_{13}) \begin{pmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{pmatrix}$$

$$= (ly_{11}w_{11} + ly_{12}w_{12} + ly_{13}w_{13} \quad ly_{11}w_{21} + ly_{12}w_{22} + ly_{13}w_{23})$$

전치행렬

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

(1,2) (1,3) (3,2)

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

(1,2) (1,3) (3,2)

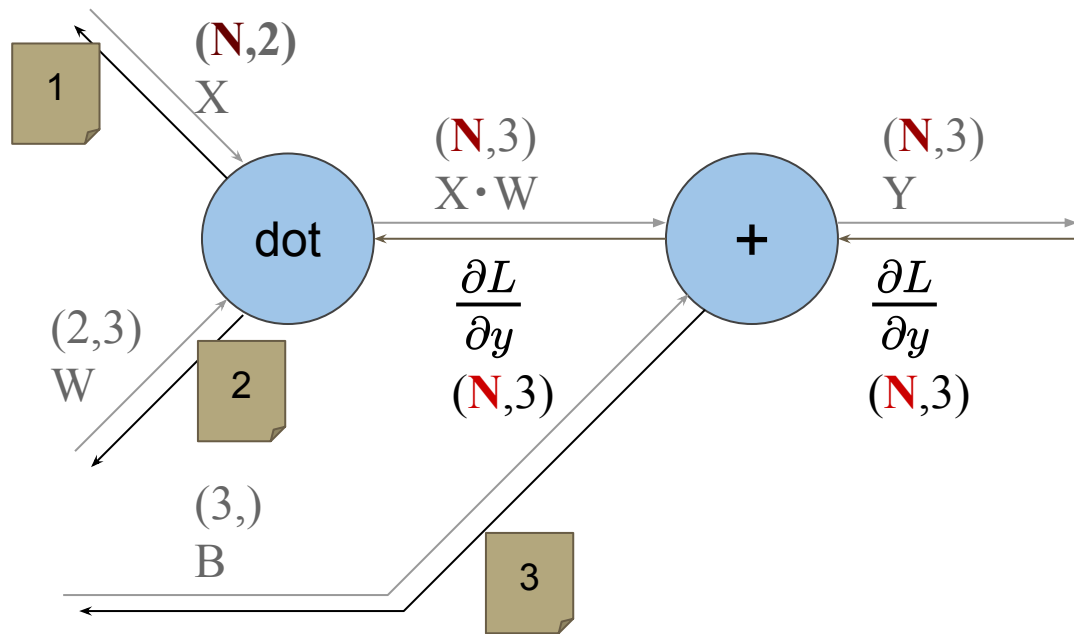
$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

(2,3) (2,1) (1,3)

$$\begin{pmatrix} lw_{11} & lw_{12} & lw_{13} \\ lw_{21} & lw_{22} & lw_{23} \end{pmatrix} = \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} \begin{pmatrix} ly_{11} & ly_{12} & ly_{13} \end{pmatrix}$$

(2,3) (2,1) (1,3)

N 개의 데이터를 묶어서 처리하는 (순전파) Affine 계층 - 배치 처리



오차역전파

오차역전파

○ 과정별

13.6.1 [STEP1] 미분과 역전파 선택

13.6.2 [STEP2] MNIST 데이터 가져오기

13.6.3 [STEP3] 함수 정의 : 수치미분, 소프트맥스, CEE

13.6.4 [STEP4] 클래스 정의 : ReLU, Affine, SoftmaxWithLoss,

13.6.5 [STEP5] 클래스 정의 : SimpleNetwork

13.6.6 [STEP6] 학습을 위한 설정치 입력

13.6.7 [STEP7] 학습과 검증

수고하셨습니다

CoLab 파일 실습