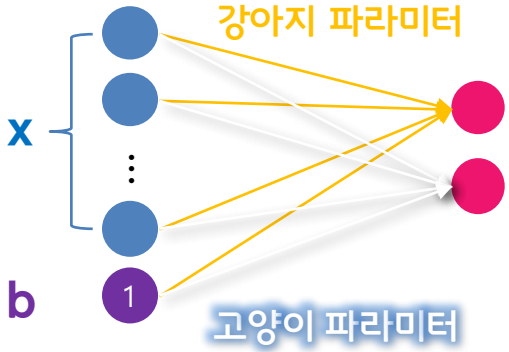
모두의연구소

Optimization

모두의연구소
박은수 Research Director

돌아보기 ...

모두의연구소



강아지 점수

고양이 점수

$$\underset{2\times1}{f(x, W)} = \underset{2\times3,072}{Wx} + \underset{2\times1}{b} \quad (3,072\times1)$$

32x32x3 영상 = 3,072 픽셀

강아지 파라미터	0.2	0.1	...	0.3	0.7	x	155
고양이 파라미터	0.7	0.8	...	0.9	0.4		200
	W						...
							110
							78
							x

$$+ \begin{matrix} 0.1 \\ 0.2 \end{matrix}_{b} = \begin{matrix} 0.3 \\ 0.7 \end{matrix}$$

0.3 강아지 점수

0.7 고양이 점수

선형분류기

돌아보기 ...

모두의연구소

Softmax loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_i e^{s_j}}\right)$$

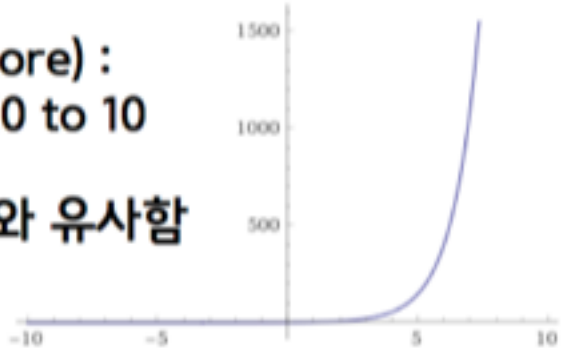
Softmax

강아지 점수	2.3	exp	9.97
고양이 점수	-1.2	→	0.3

전부다 양수가 됨

exp(score) :
from -10 to 10

Max함수와 유사함

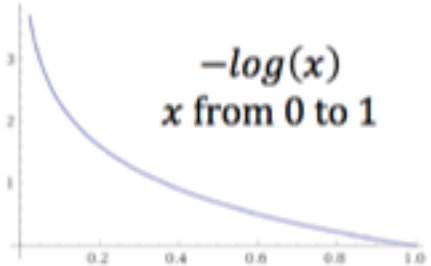


돌아보기 ...

모두의연구소

Softmax loss

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$



강아지 점수	2.3	exp	9.97	normalize	0.97
고양이 점수	-1.2		0.3		**0.03**
			전부다 양수가 됨		**확률**

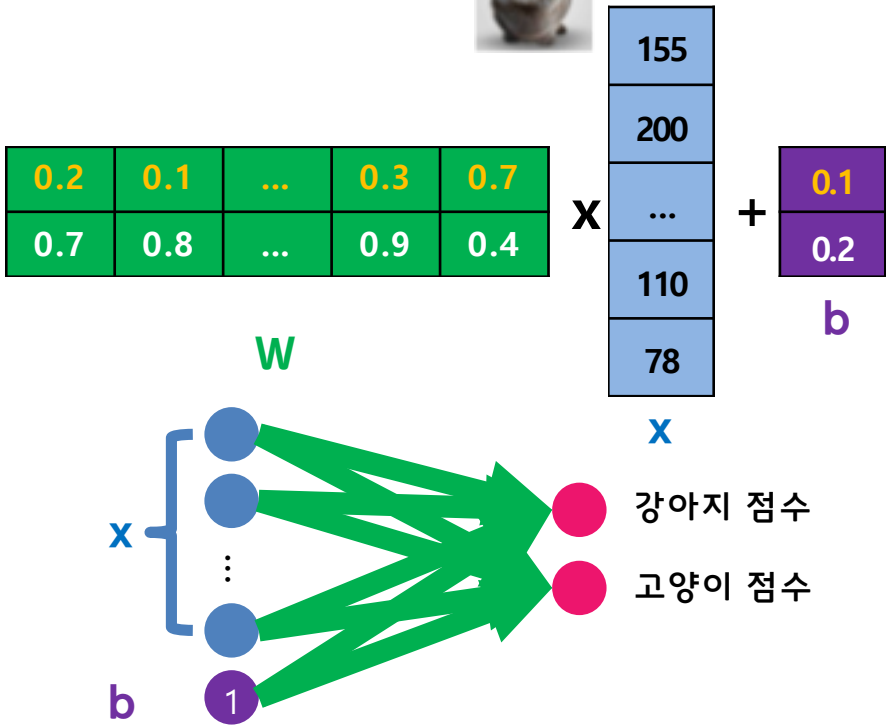
$L_i = -log(0.03) = 3.5$

돌아보기 …

모두의연구소

- 분류기의 구성
  - Score function
  - Loss function
  - Optimization

고양이가 입력이면 고양이 점수가 높아야 함



돌아보기 …

모두의연구소

- 분류기의 구성
  - Score function
  - Loss function
  - Optimization

0.2	0.1	…	0.3	0.7
0.7	0.8	…	0.9	0.4

W

x

155
200
…
110
78

x

+

0.1
0.2

b

=

Cross-entropy loss (Softmax)

강아지 점수	2.3	exp	9.97	normalize	0.97	강아지 확률
고양이 점수	-1.2		0.3		0.03	고양이 확률

$-log(0.03) = 3.5$

현재의 분류기는 3.5만큼 안 좋음. 이 loss 값을 줄이는게 목표

돌아보기 …

모두의연구소

- 분류기의 구성
  - Score function
  - Loss function
  - Optimization

오늘의 주제

Loss를 최소화하는 w와 b를 찾아라

최저점을 향해 가시오

From : lecture note 2 - cs231n stanford university

최저점을 향해 가시오

헉?!

눈이 없으면 ?

최저점을 향해 가시오

넘어질 것 같아..
어디까지 갈 수 있을까..

[대안]
발로 더듬더듬 해서 내리막이면 가자!

내리막을 찾는 방법

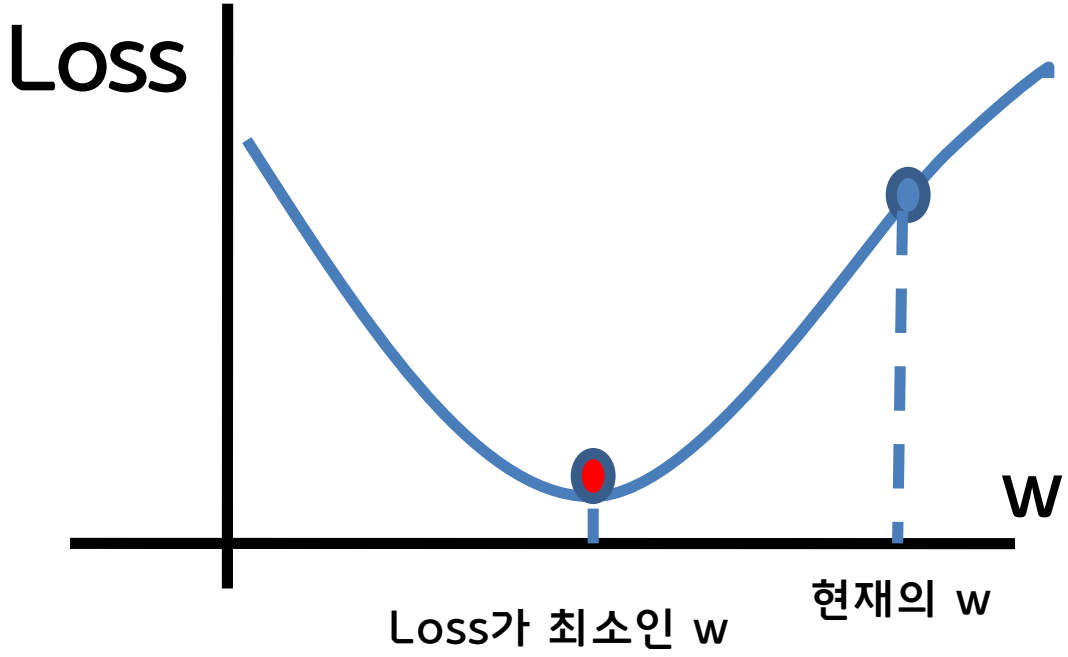
모두의연구소

내리막 ?? == 기울기 ??

기울기를 따라 내려 가보자

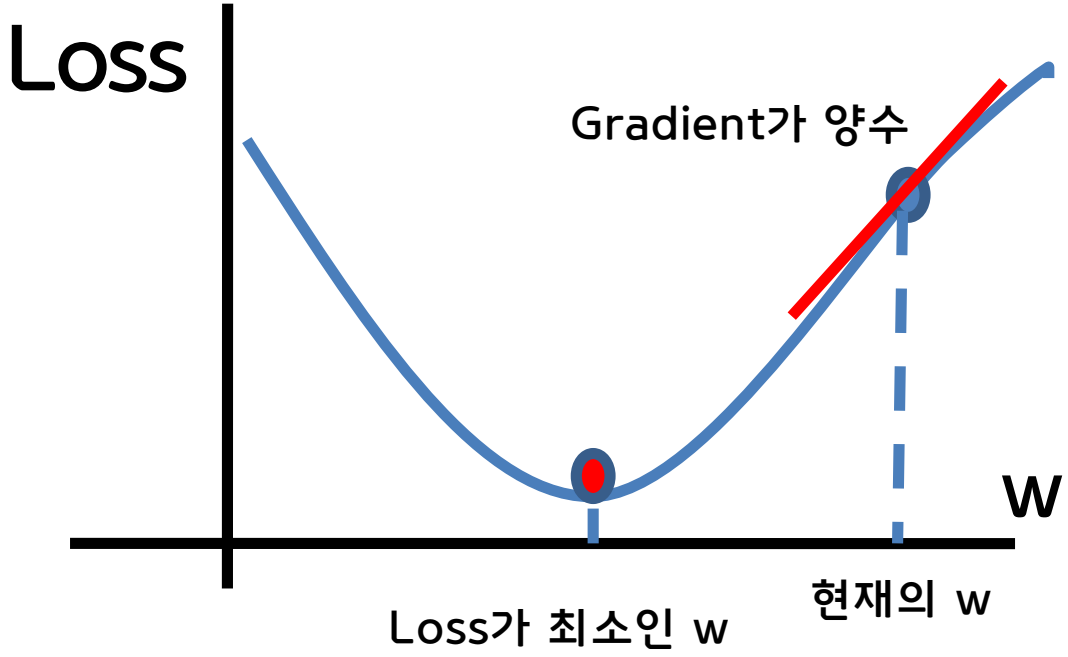
Gradient Descent

Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

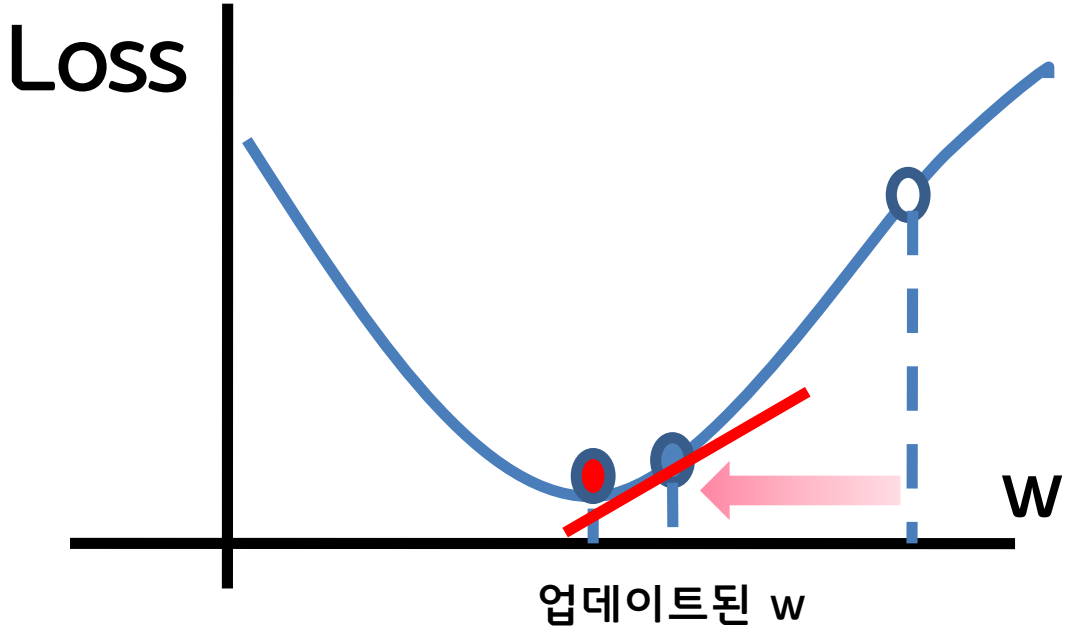
Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

η : learning rate

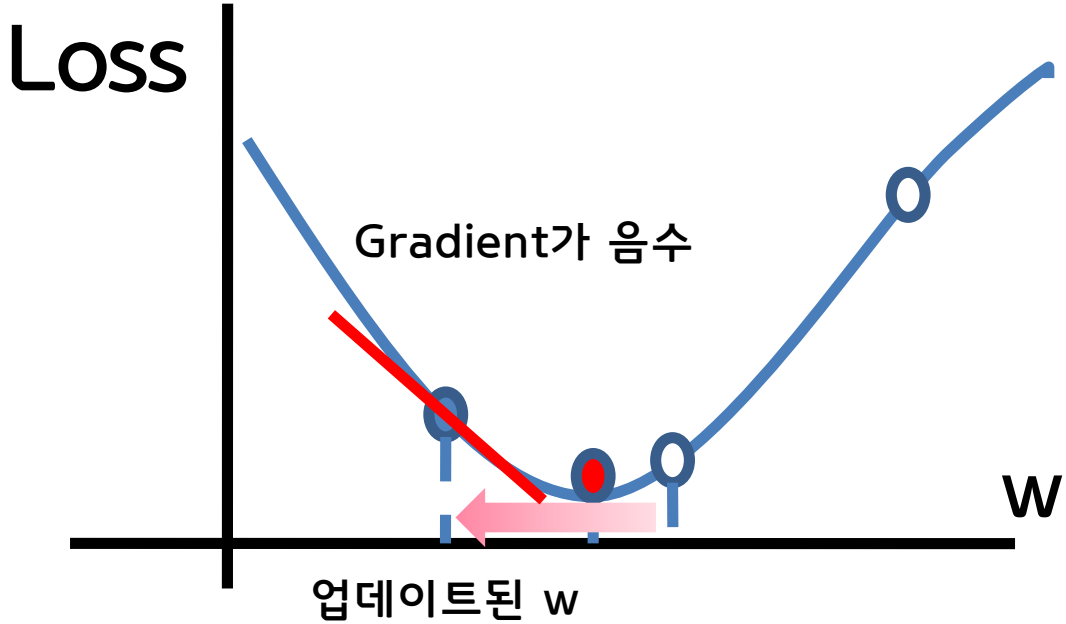
Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

η : learning rate

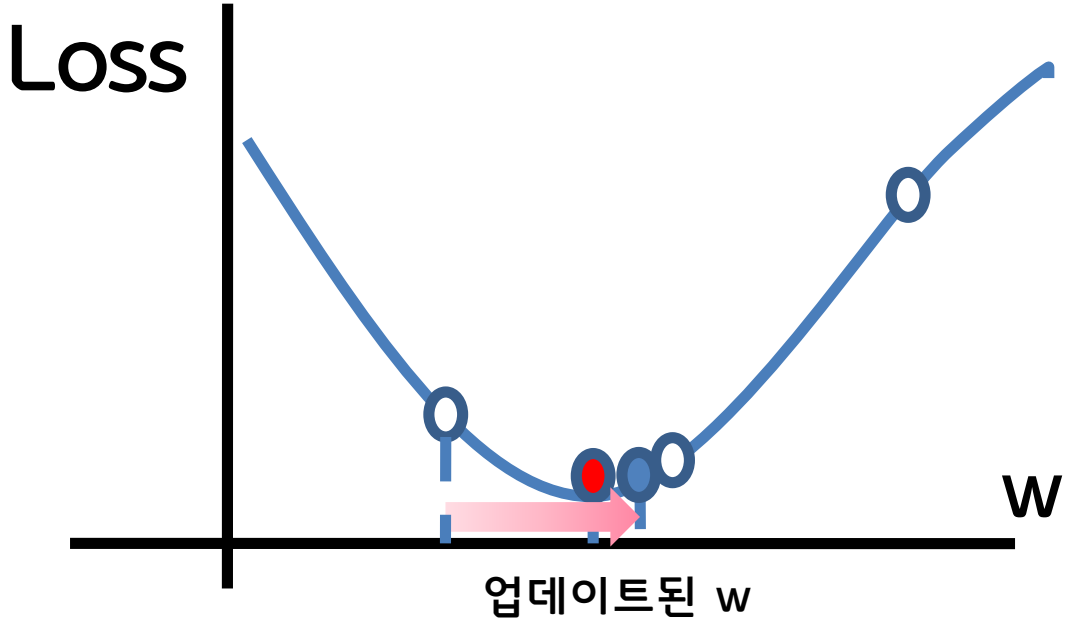
Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

η : learning rate

Gradient Descent



$$w = w - \eta \frac{\partial L}{\partial w}$$

Loss 함수에 대한 w의 음의 Gradient 찾아서 연속적으 업데이트해 주면 되는군요

그런데 어떻게 Gradient를 찾죠?

Gradient Descent

Strategy #2: **Follow the slope**

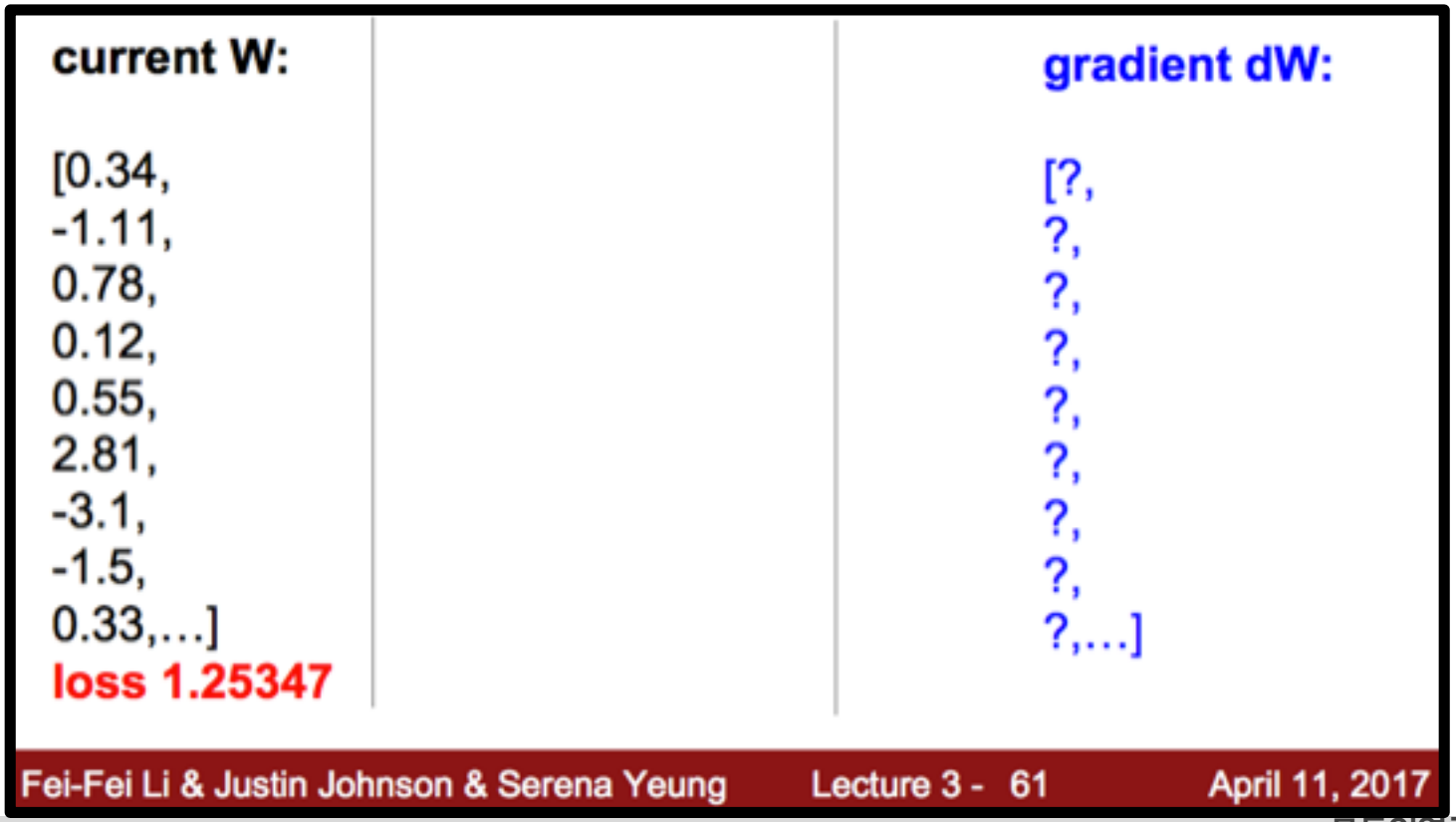
In 1-dimension, the derivative of a function:

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

In multiple dimensions, the **gradient** is the vector of (partial derivatives) along each dimension

The slope in any direction is the **dot product** of the direction with the gradient
The direction of steepest descent is the **negative gradient**

Gradient Descent



0.2	0.1	...	0.3	0.7
0.7	0.8	...	0.9	0.4

W

x

155
200
...
110
78

x

+

0.1
0.2

b

=

강아지 점수

2.3
-1.2

고양이 점수

exp

9.97
0.3

normalize

강아지 확률

0.97
0.03

고양이 확률

$-log(0.03) = 3.5$

현재의 분류기는 3.5만큼 안 좋음. 이 loss 값을 줄이는게 목표

Gradient Descent

current W:	W + h (first dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34 + **0.0001**, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[?, ?, ?, ?, ?, ?, ?, ?, ?,…]
loss 1.25347	loss 1.25322	

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 62 April 11, 2017

Gradient Descent

current W:	W + h (first dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34 + **0.0001**, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[-2.5, ?, ?, ?, ?,…]
loss 1.25347	loss 1.25322	

(1.25322 - 1.25347)/0.0001
= -2.5

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 63 April 11, 2017

Gradient Descent

current W:	W + h (second dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	?,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	loss 1.25353	

Gradient Descent

current W:	W + h (second dim):	gradient dW:
[0.34, -1.11, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[0.34, -1.11 + **0.0001**, 0.78, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,…]	[-2.5, **0.6**, ?, ?,
loss 1.25347	loss 1.25353	?,…]

(1.25353 - 1.25347)/0.0001
= 0.6

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

Gradient Descent

current W:	W + h (third dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	?,
0.12,	0.12,	?,
0.55,	0.55,	?,
2.81,	2.81,	?,
-3.1,	-3.1,	?,
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	loss 1.25347	

Gradient Descent

current W:	W + h (third dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11,	0.6,
0.78,	0.78 + **0.0001**,	**0**,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	loss 1.25347	

(1.25347 - 1.25347)/0.0001
= 0

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

Gradient Descent

This is silly. The loss is just a function of W:

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Gradient Descent

This is silly. The loss is just a function of W:

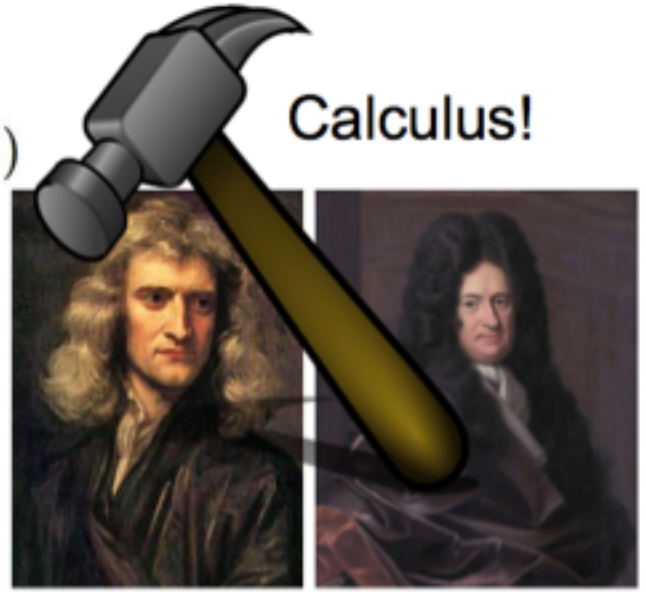
$$L = \frac{1}{N}\sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

want $\nabla_W L$

Use calculus to compute an **analytic gradient**



수치 미분은 구현은 쉽지만 시간이 오래걸립니다

또한 해석적 방법에 비해 부정확합니다

오차역전파법 (Backpropagation)

Computational Graph

모두의연구소

- 연쇄법칙 (chain rule) 이란?
  - 합성함수의 미분은 합성 함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다

$$z = (x + y)^2 \Rightarrow \begin{matrix} z = t^2 \\ t = x + y \end{matrix}$$

z의 x에 대한 미분의 연쇄법칙 표현

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} \Rightarrow \frac{\partial z}{\partial x} = \frac{\partial z}{\cancel{\partial t}}\frac{\cancel{\partial t}}{\partial x}$$

Computational Graph

모두의연구소

- **연쇄법칙이란?**

$$z = (x + y)^2 \Rightarrow \begin{cases} z = t^2 \\ t = x + y \end{cases}$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} \Rightarrow \begin{cases} \frac{\partial z}{\partial t} = 2t \\ \frac{\partial t}{\partial x} = 1 \end{cases}$$

결과 $$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x + y)$$

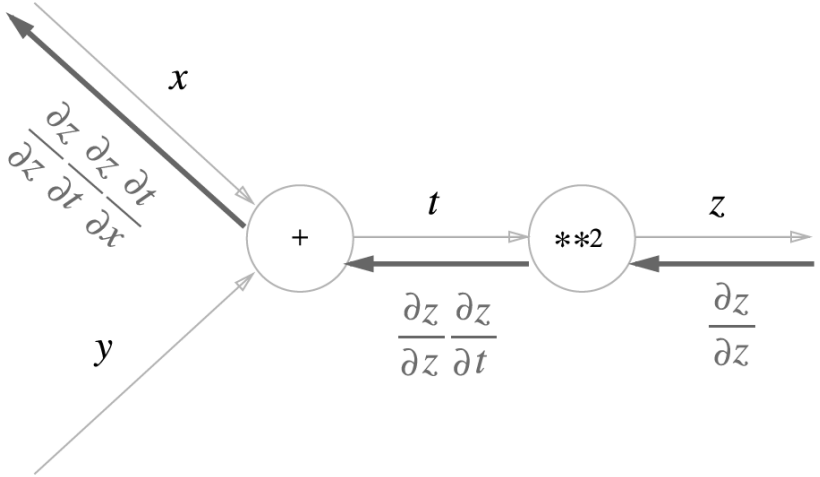
Computational Graph

모두의연구소

- **연쇄법칙과 계산 그래프**

$$z = (x + y)^2 \Rightarrow \begin{matrix} z = t^2 \\ t = x + y \end{matrix}$$

역전파 과정



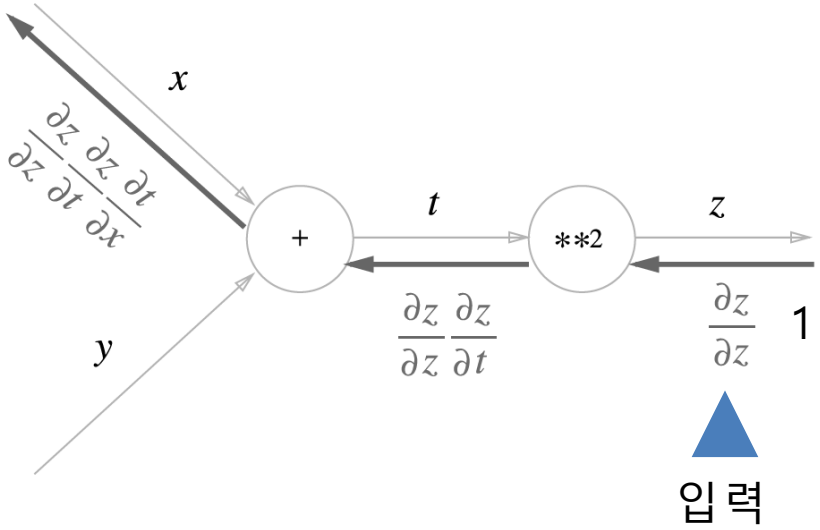
Computational Graph

모두의연구소

- **연쇄법칙과 계산 그래프**

$z = (x + y)^2$ ⇨ $z = t^2$, $t = x + y$

역전파 과정



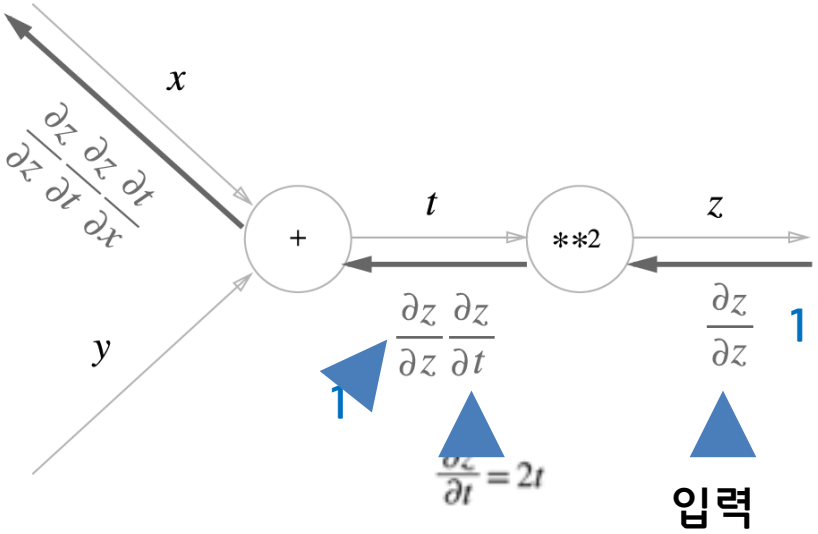
Computational Graph

모두의연구소

- **연쇄법칙과 계산 그래프**

$z = (x + y)^2$ ⇨ $z = t^2$, $t = x + y$

역전파 과정



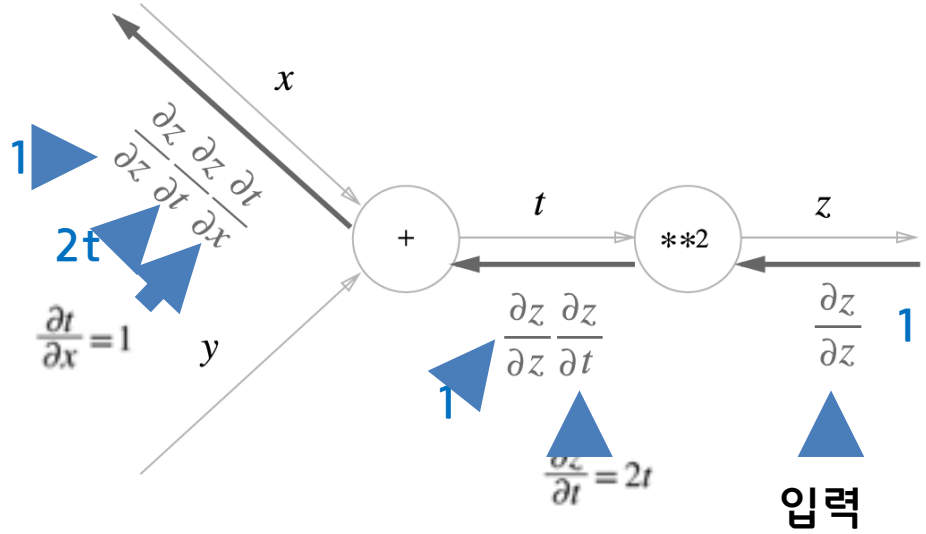
Computational Graph

모두의연구소

- **연쇄법칙과 계산 그래프**

$z = (x + y)^2$ ⇨ $z = t^2$, $t = x + y$

역전파 과정

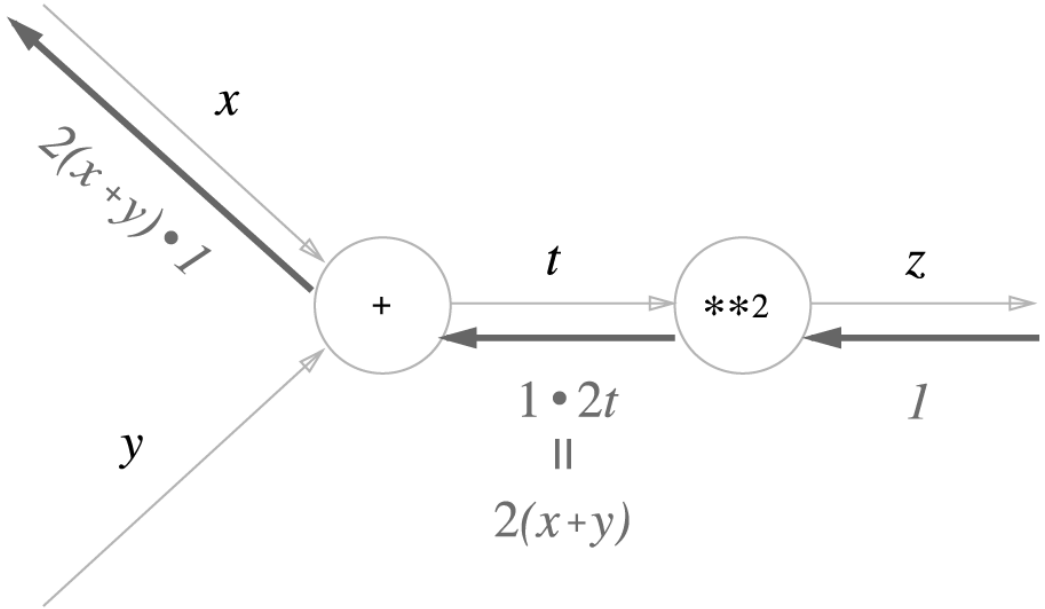


Computational Graph

- **연쇄법칙과 계산 그래프**

$z = (x + y)^2$ ⇨ $z = t^2$, $t = x + y$

역전파 과정



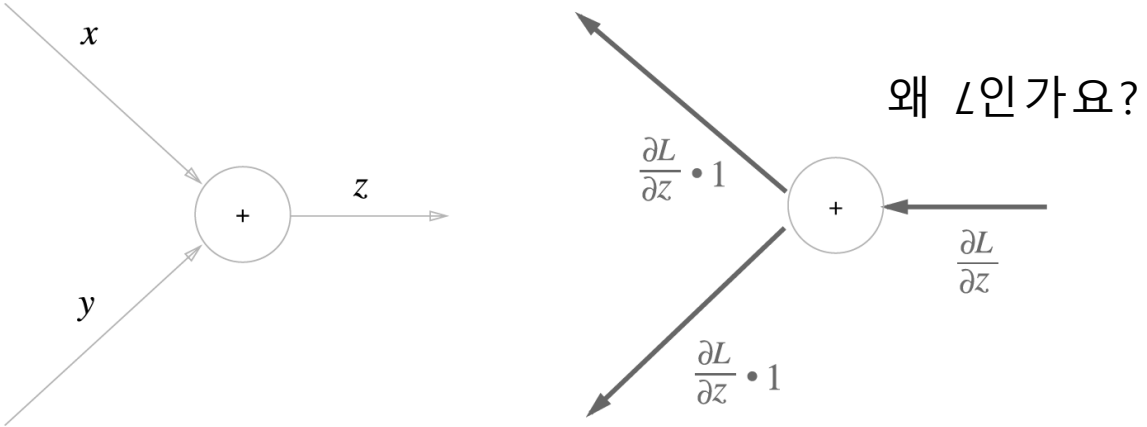
Computational Graph

- 덧셈 노드의 역전파

$$z = x + y$$

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

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

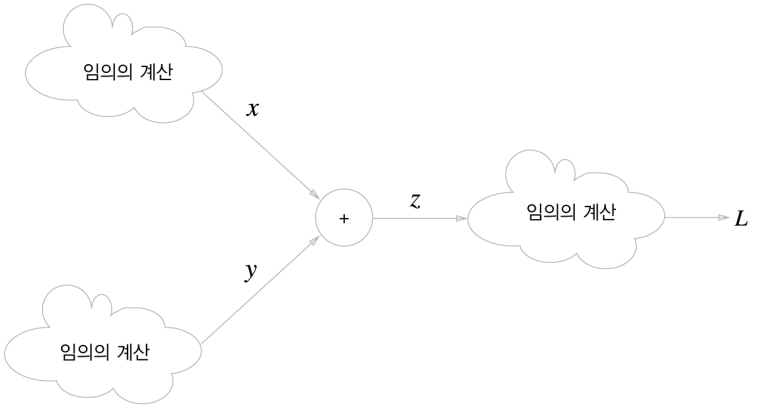


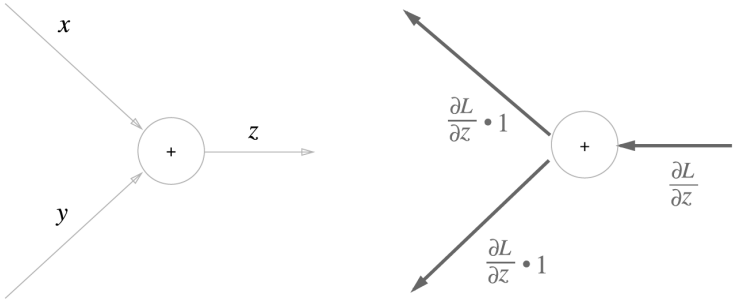
왜 1인가요?

Computational Graph

- 덧셈 노드의 역전파

실제로 큰 계산을 가정하고 그중 한 노드가 덧셈노드임을 가정한 겁니다





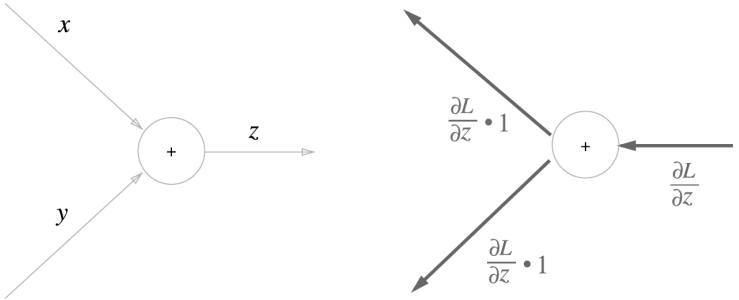
Computational Graph

- **덧셈 노드의 역전파**

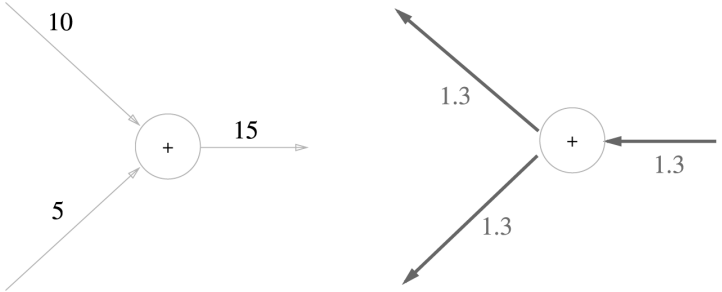
$$z = x + y$$

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

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



예시



Computational Graph

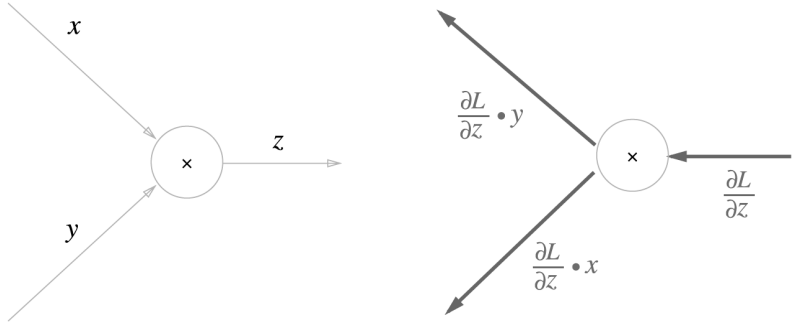
모두의연구소

- **곱셈 노드의 역전파**

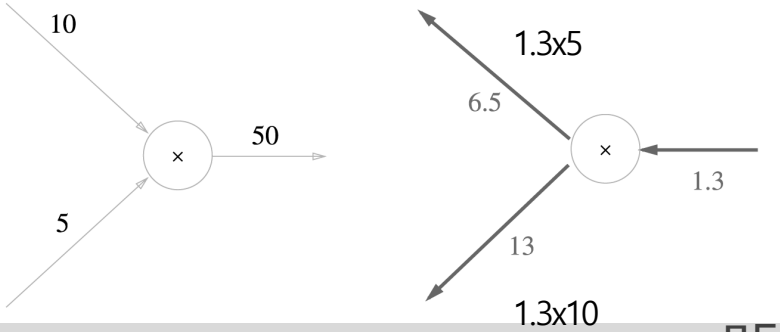
$$z = xy$$

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

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



예시

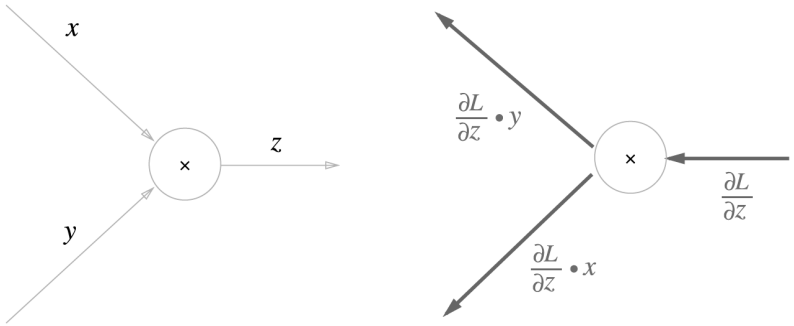


Computational Graph

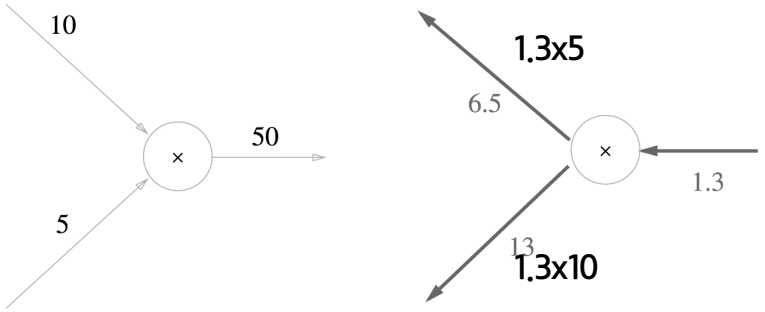
모두의연구소

- 곱셈 노드의 역전파

$$z = xy \Rightarrow \frac{\partial z}{\partial x} = y, \quad \frac{\partial z}{\partial y} = x$$



- 곱셈의 역전파는 순방향 입력신호의 값이 필요합니다
- 곱셈노드 구현 시 순전파 입력신호를 변수에 저장해 둡니다

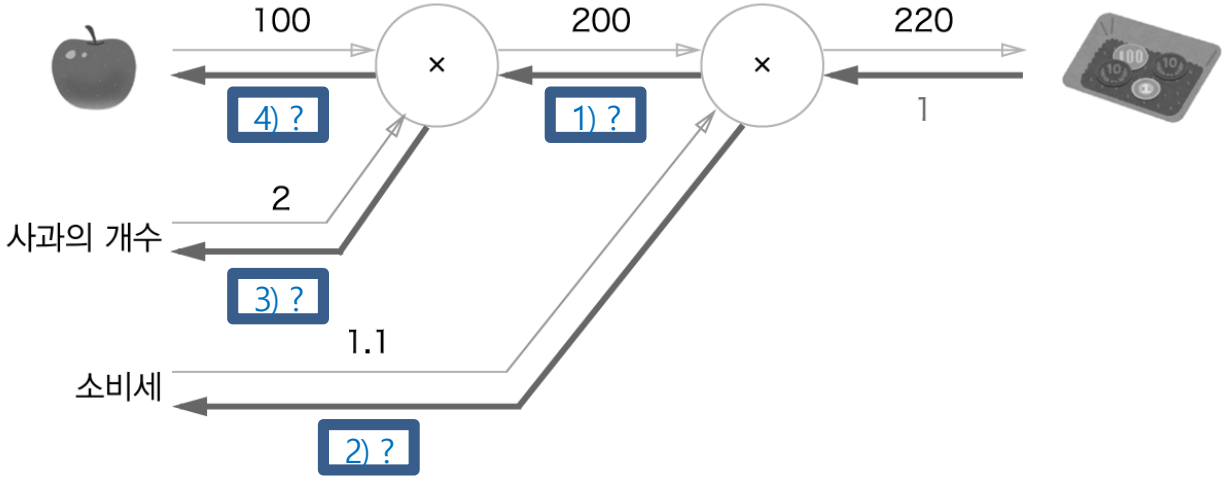


Reference : "밑바닥부터 시작하는 딥러닝", 사이토 고키

Computational Graph

모두의연구소

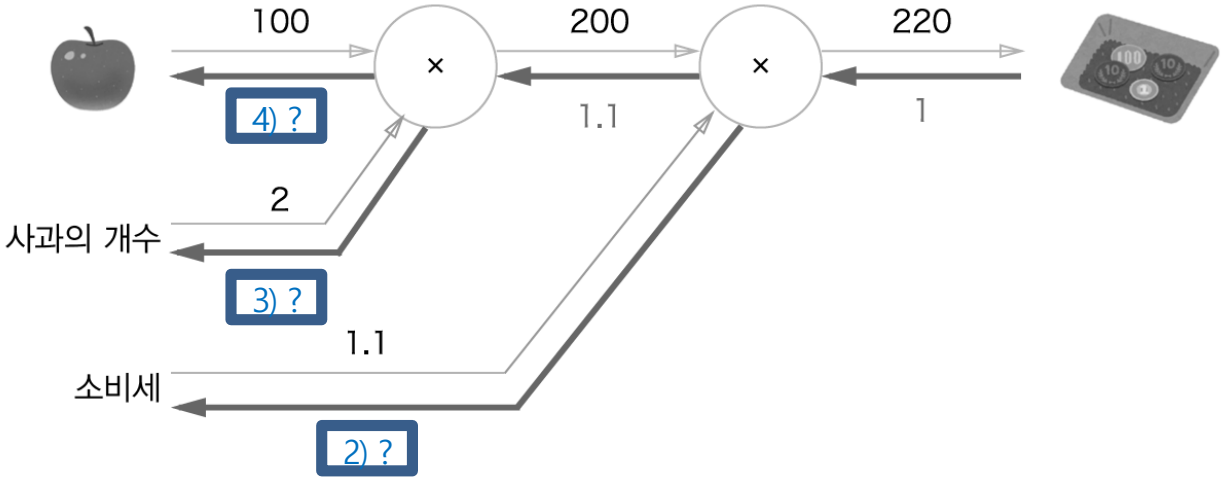
- **사과쇼핑의 예**



Computational Graph

모두의연구소

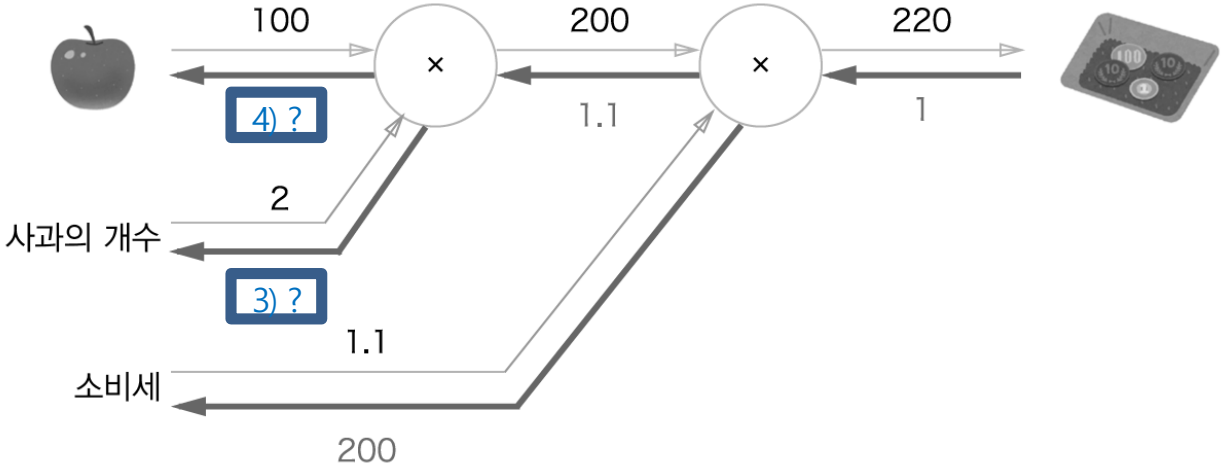
- **사과쇼핑의 예**



Computational Graph

모두의연구소

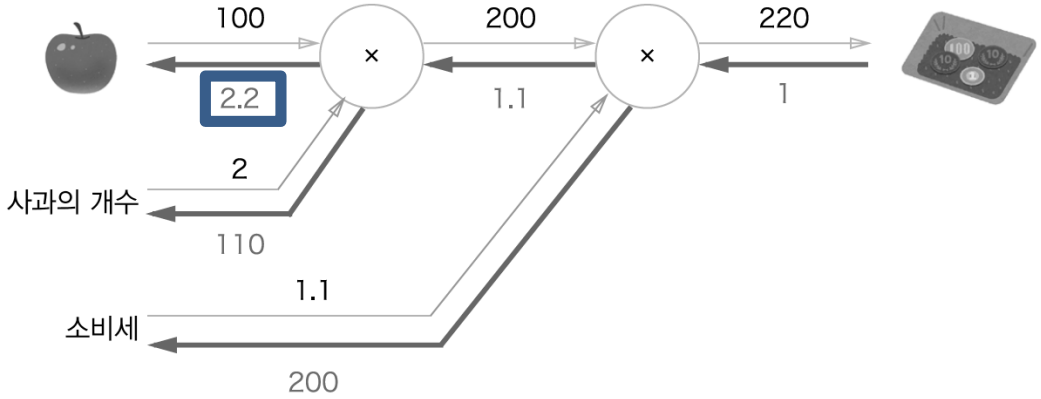
- **사과쇼핑의 예**



Computational Graph

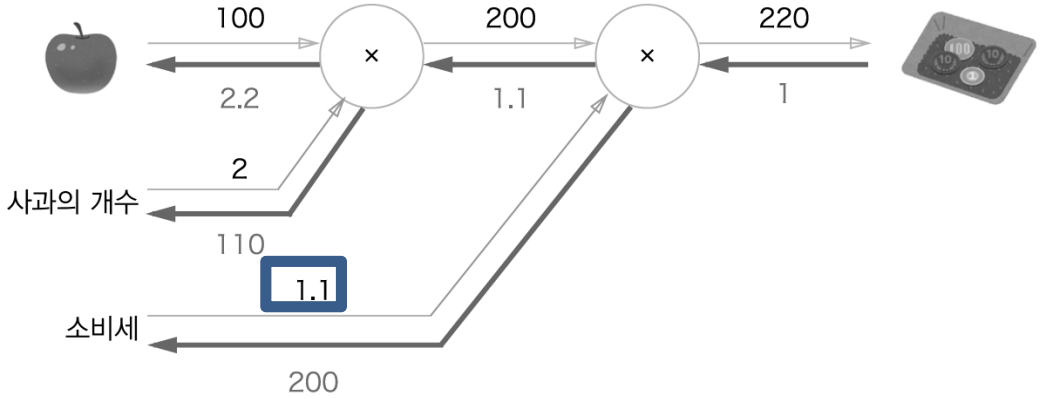
모두의연구소

- **사과쇼핑의 예**



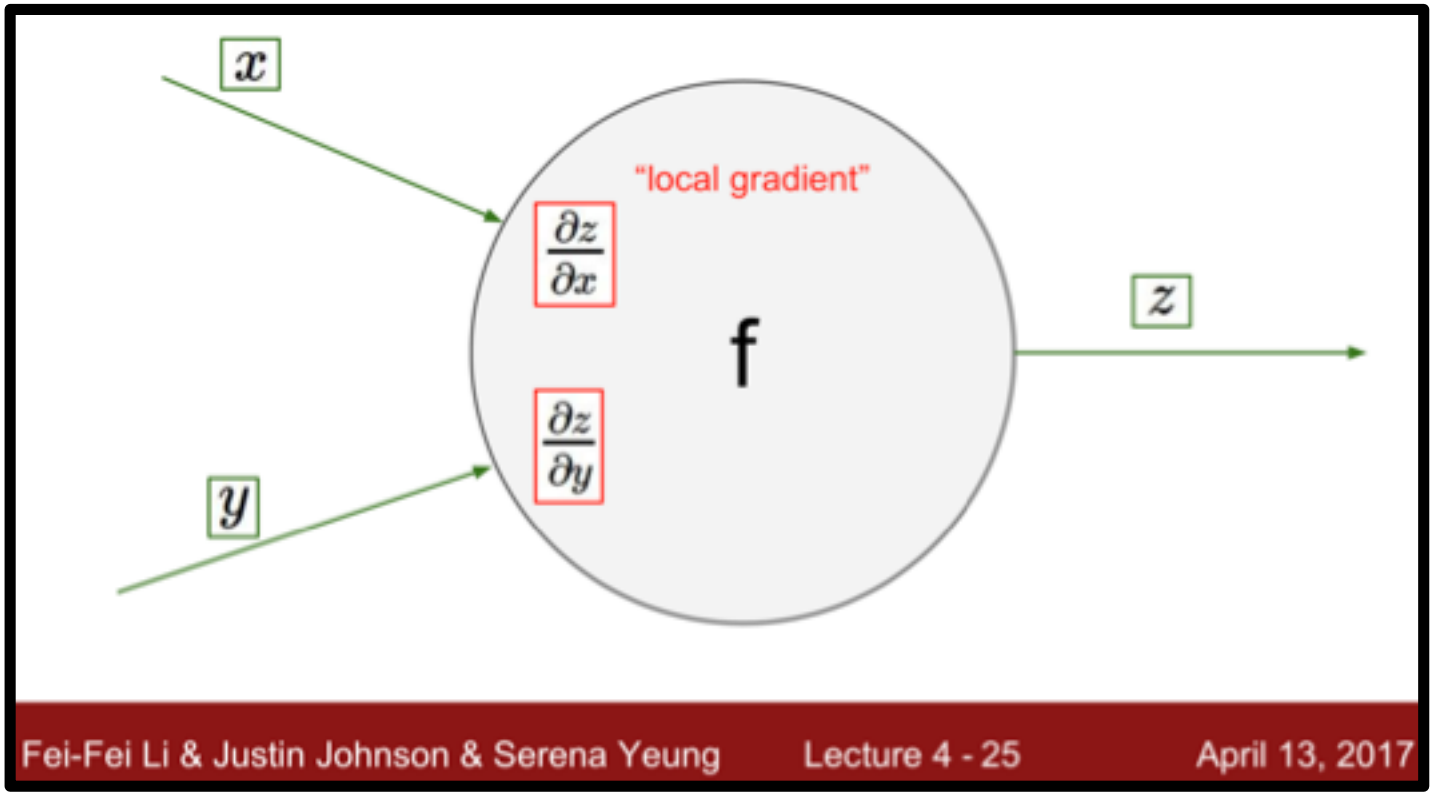
  - **사과가격이 1원 오르면, 최종 가격 2.2 증가**
    - **101(사과 가격)x2x1.1 = 222.2 원**

Computational Graph

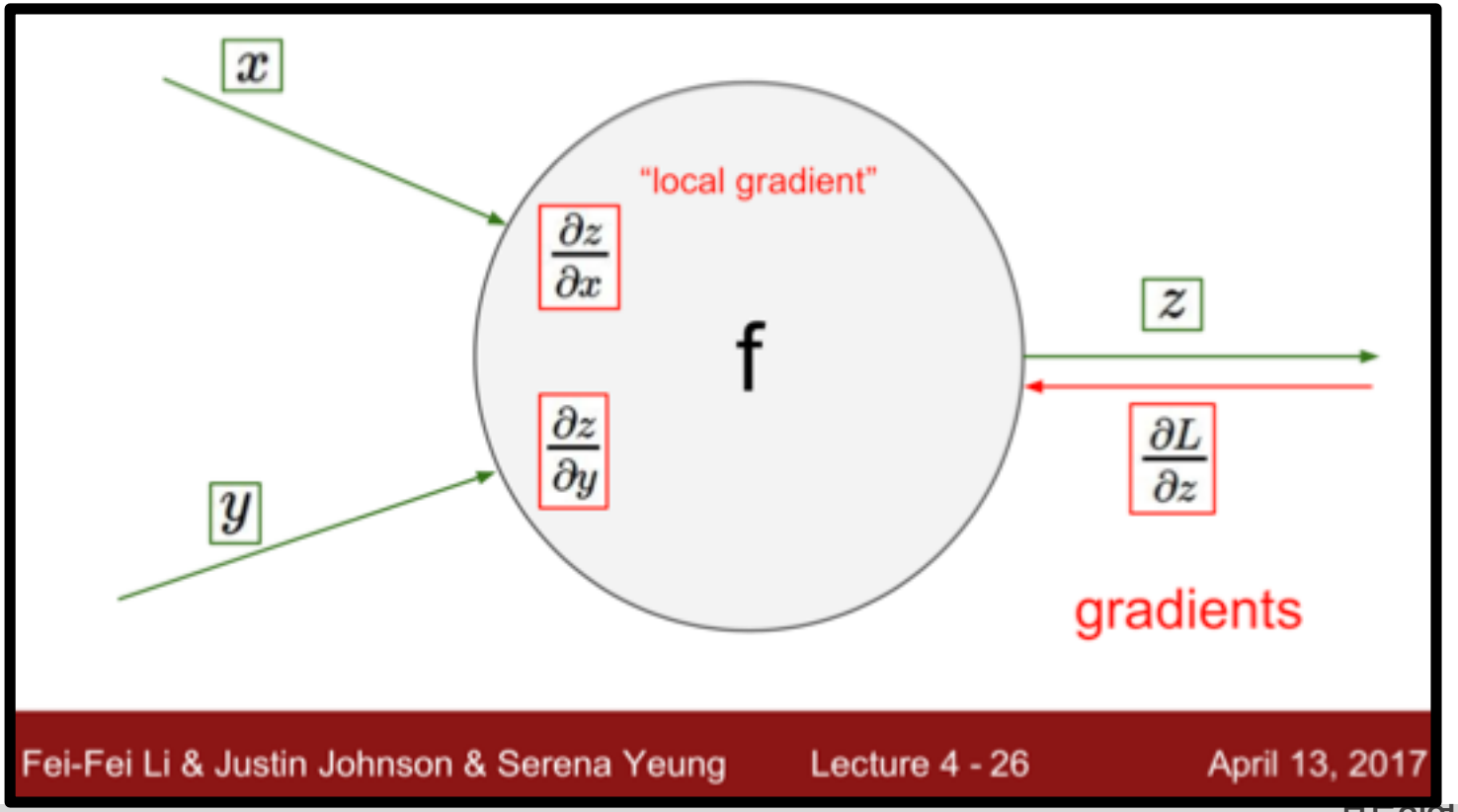


- 사과가격이 1원 오르면, 최종 가격 2.2 증가
  - 101(사과 가격)x2x1.1 = 222.2 원
- 소비세가 100%(1.1+1 = 2.1)오르면 가격 200증가
  - 100x2x2.1(소비세) = 420 원

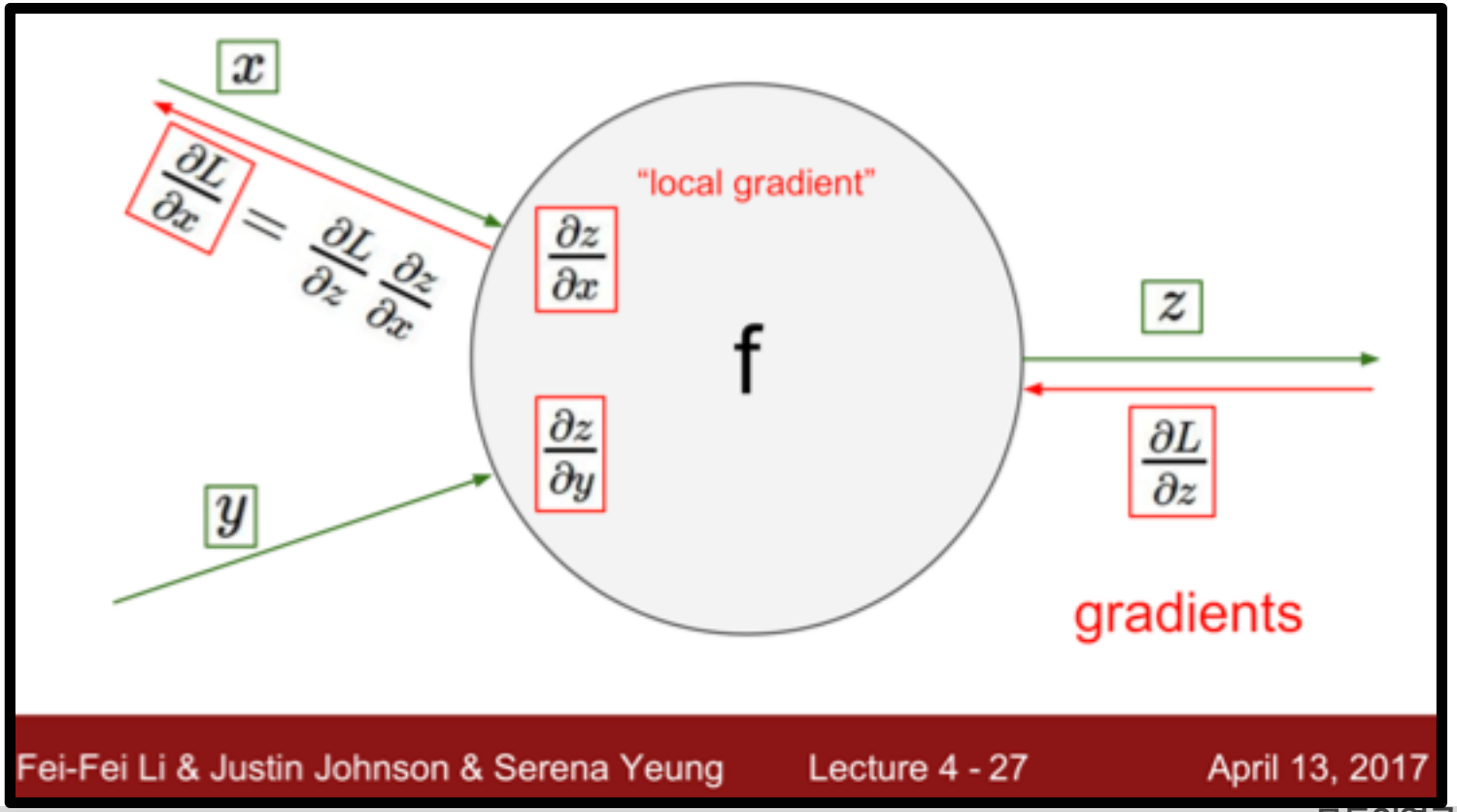
Computational Graph



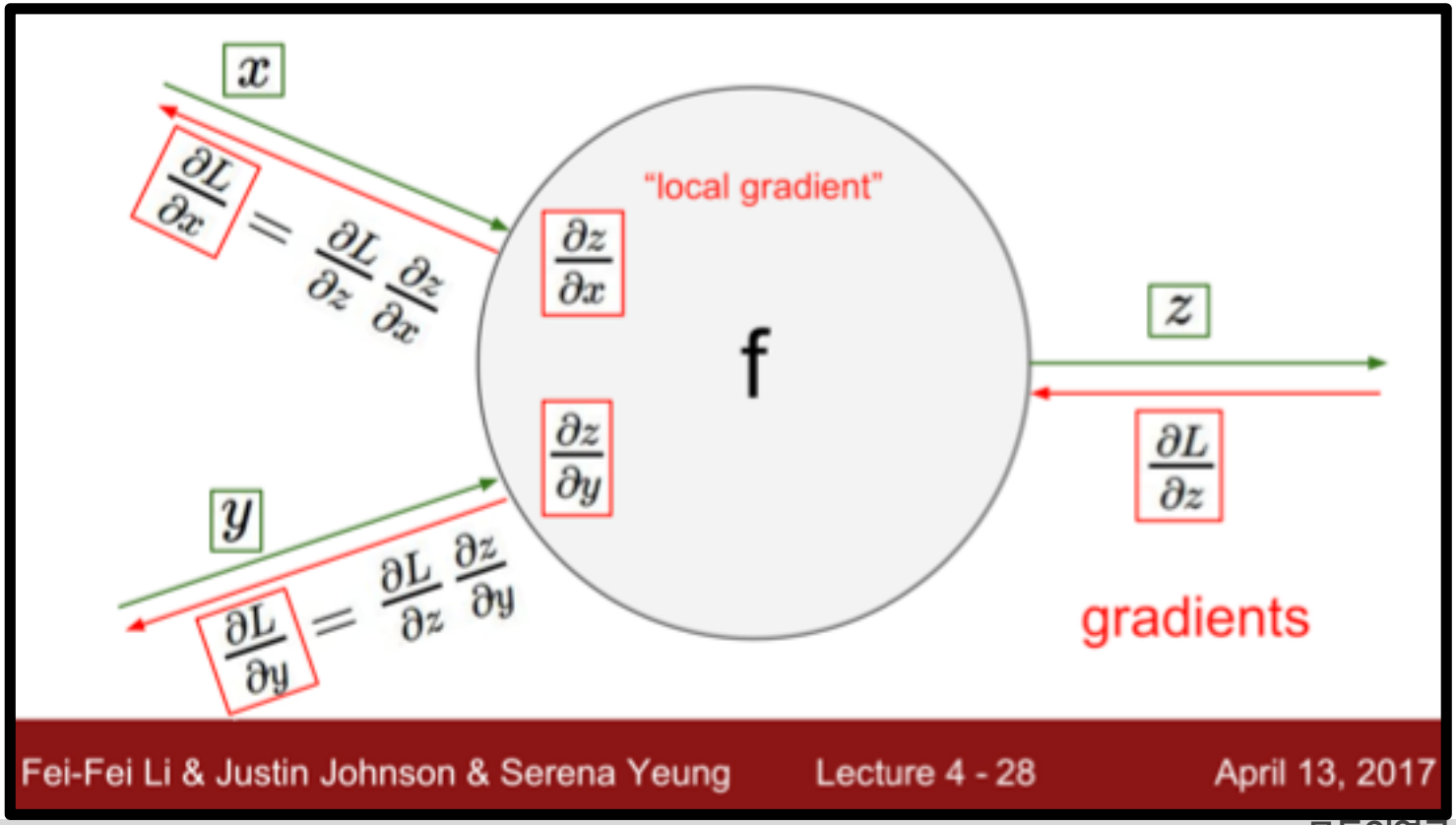
Computational Graph



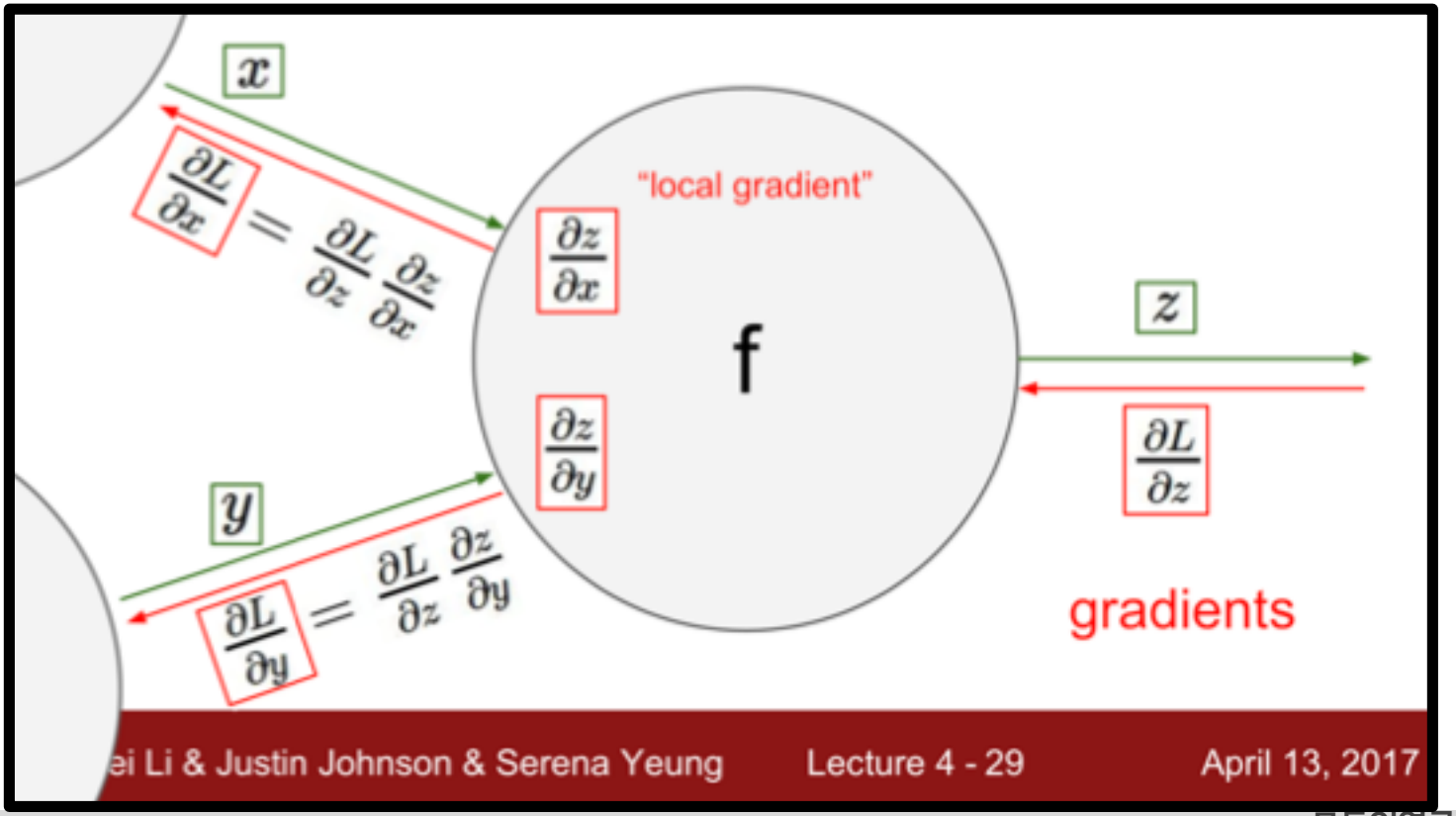
Computational Graph



Computational Graph



Computational Graph

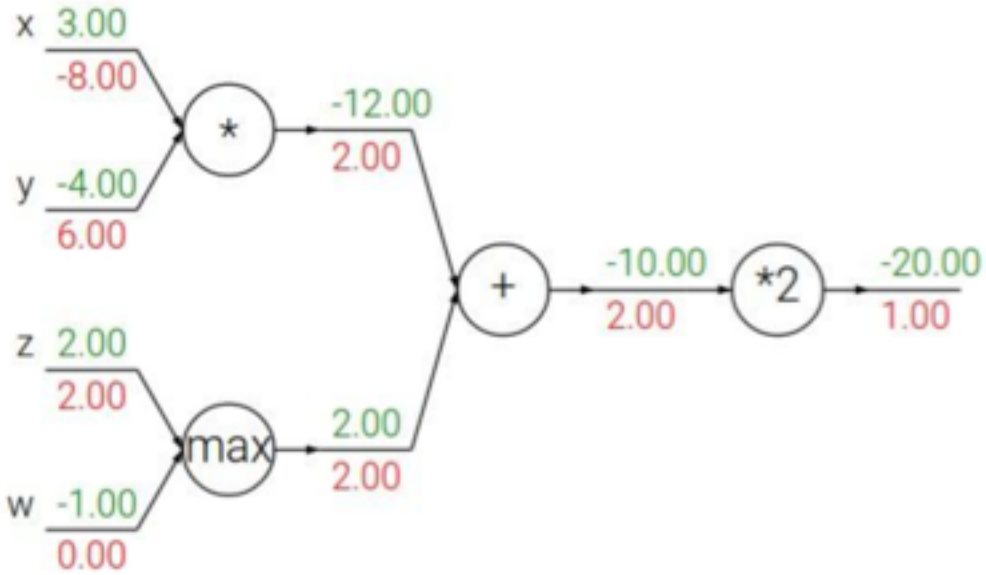


Patterns in backward flow

add gate: gradient distributor

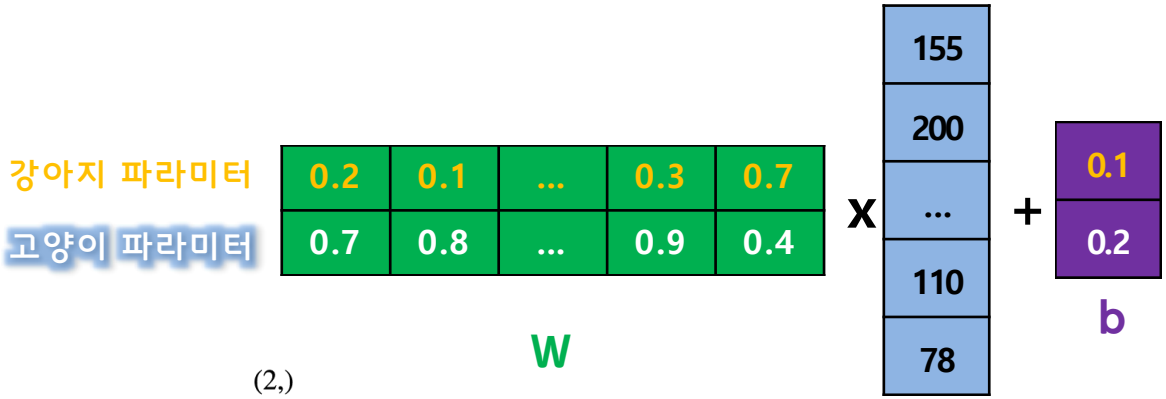
max gate: gradient router

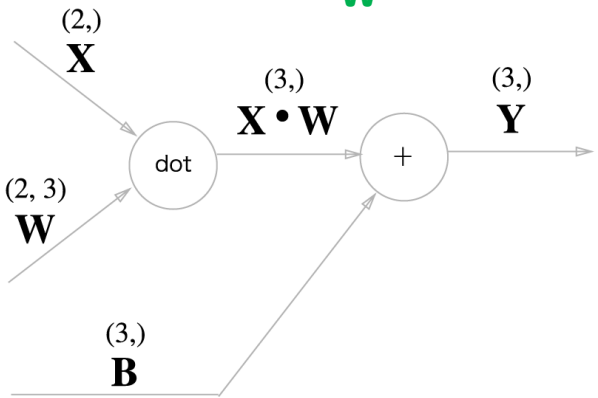
mul gate: gradient switcher



Computational Graph

모두의연구소



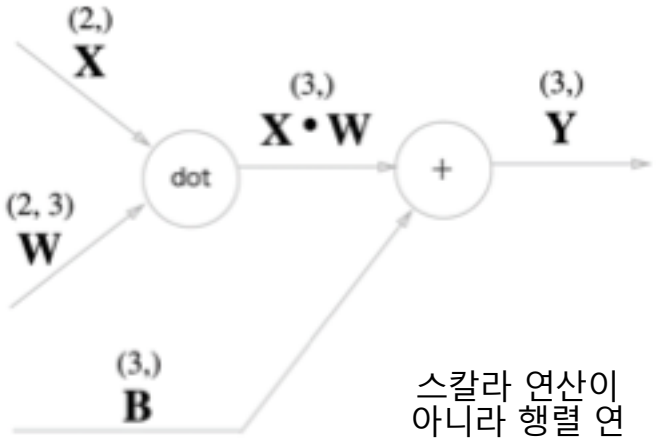


- Affine 계층

Affine / Softmax 계층 구현하기

- Affine 계층

```
In [18]: X = np.random.rand(2)

In [19]: W = np.random.rand(2,3)

In [20]: B = np.random.rand(3)

In [21]: X.shape
Out[21]: (2,)

In [22]: W.shape
Out[22]: (2, 3)

In [23]: B.shape
Out[23]: (3,)

In [24]: Y = np.dot(X, W) + B
```

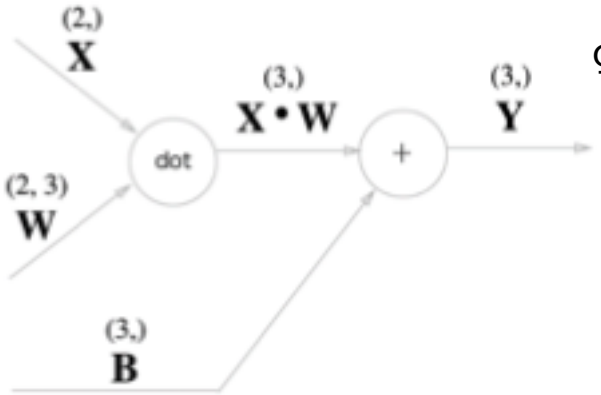


스칼라 연산이 아니라 행렬 연산이 됩니다

- 신경망의 순전파 때 수행하는 행렬의 내적은 기하학에서 어파인 변환(affine transformation)이라고 합니다.

Affine / Softmax 계층 구현하기

- Affine 계층



역전파

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

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

- $\mathbf{W}^{\mathrm{T}}$의 T 는 전치(transpose)행렬을 뜻합니다. 전치행렬은 $\mathbf{W}$의 (i, j) 위치의 원소를 (j, i) 위치로 바꾼 것을 말합니다.

Affine / Softmax 계층 구현하기

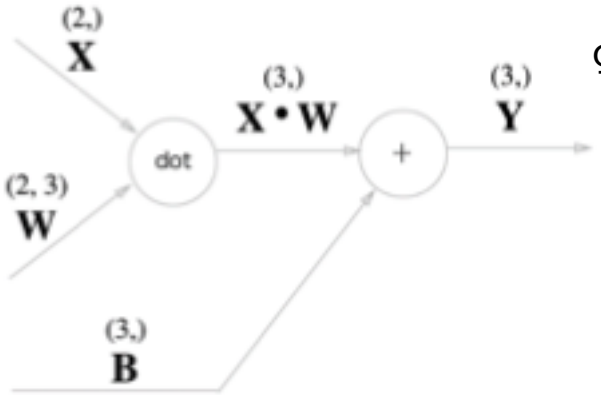
- Affine 계층
  - $\mathbf{W}^{\mathrm{T}}$의 T 는 전치(transpose)행렬을 뜻합니다. 전치행렬은 $\mathbf{W}$의 (i, j) 위치의 원소를 (j, i) 위치로 바꾼 것을 말합니다.

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

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

Affine / Softmax 계층 구현하기

- Affine 계층



역전파

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

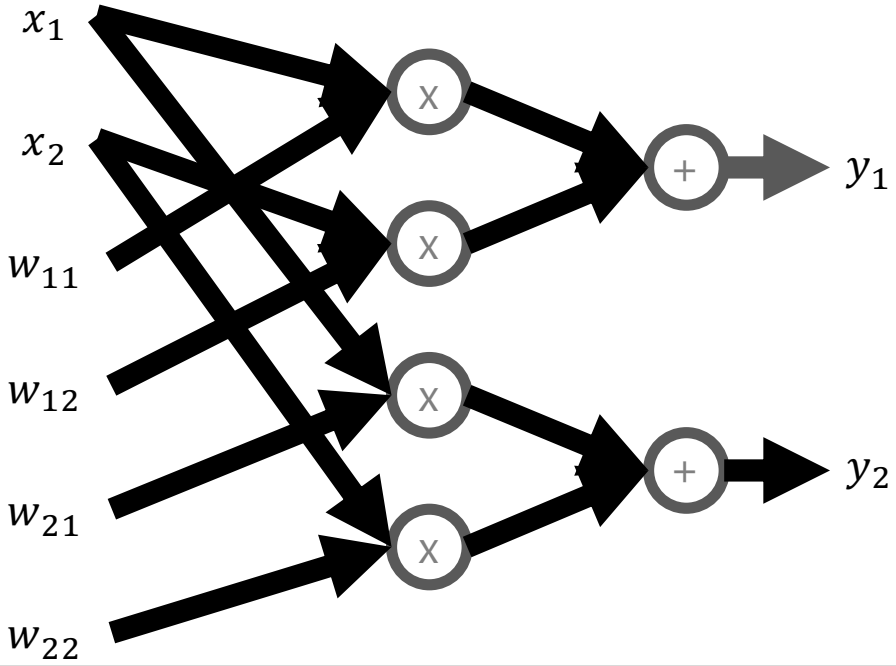
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

이건 왜 이렇게 된건가요?

Computational Graph

- Affine 계층

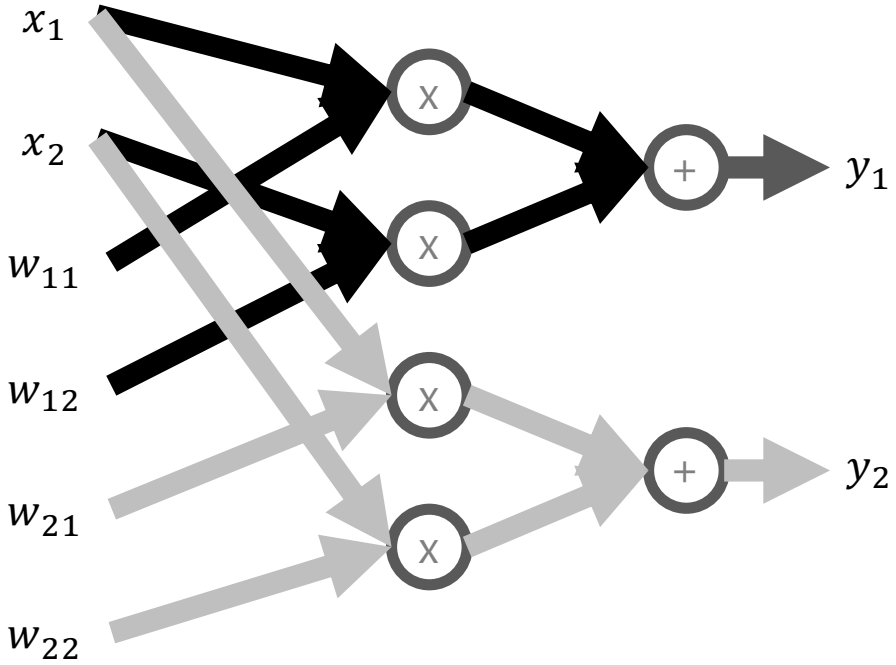
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

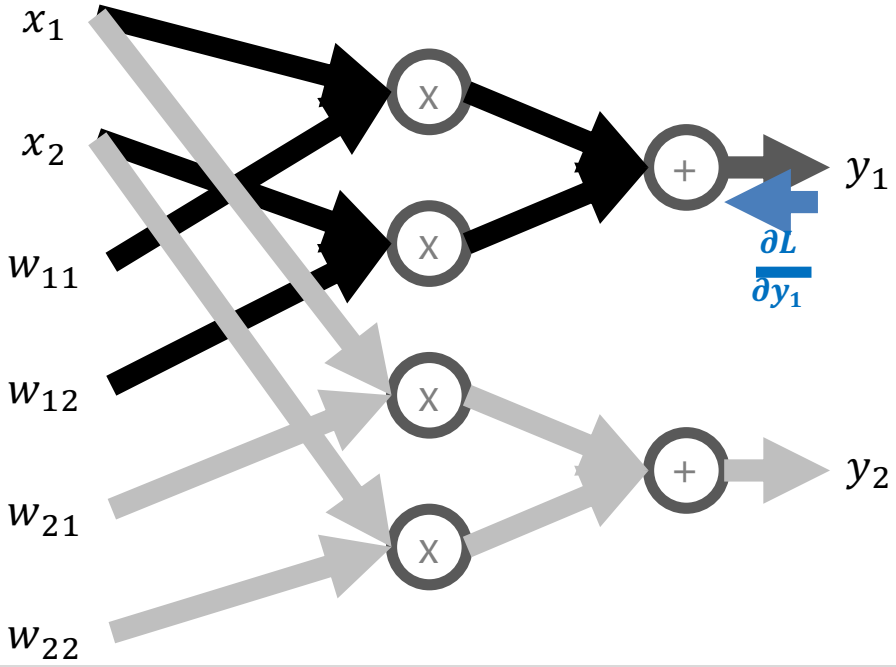
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

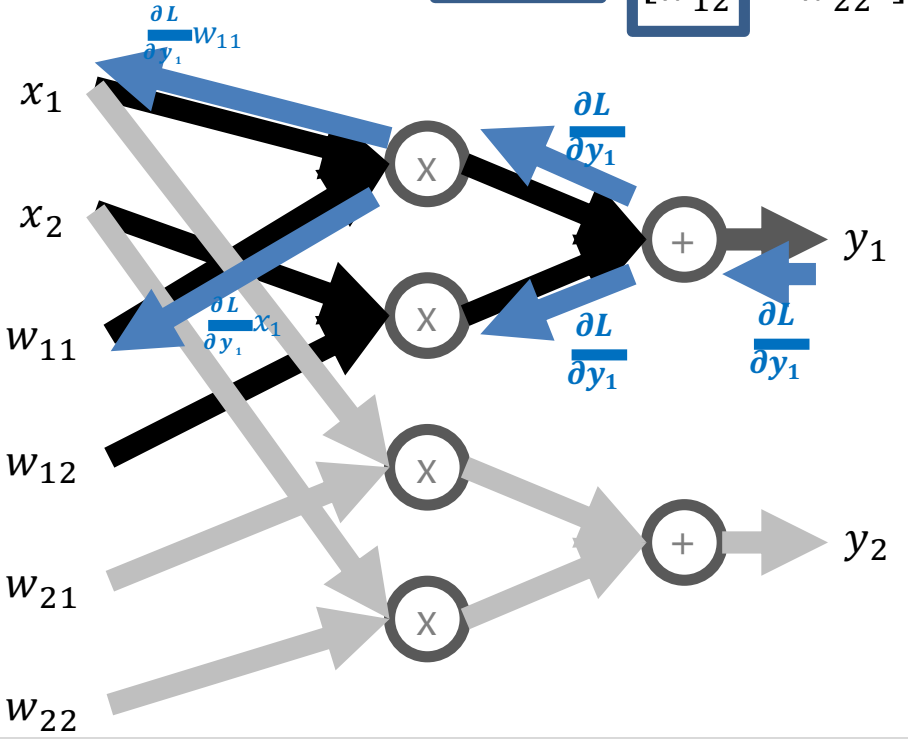
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

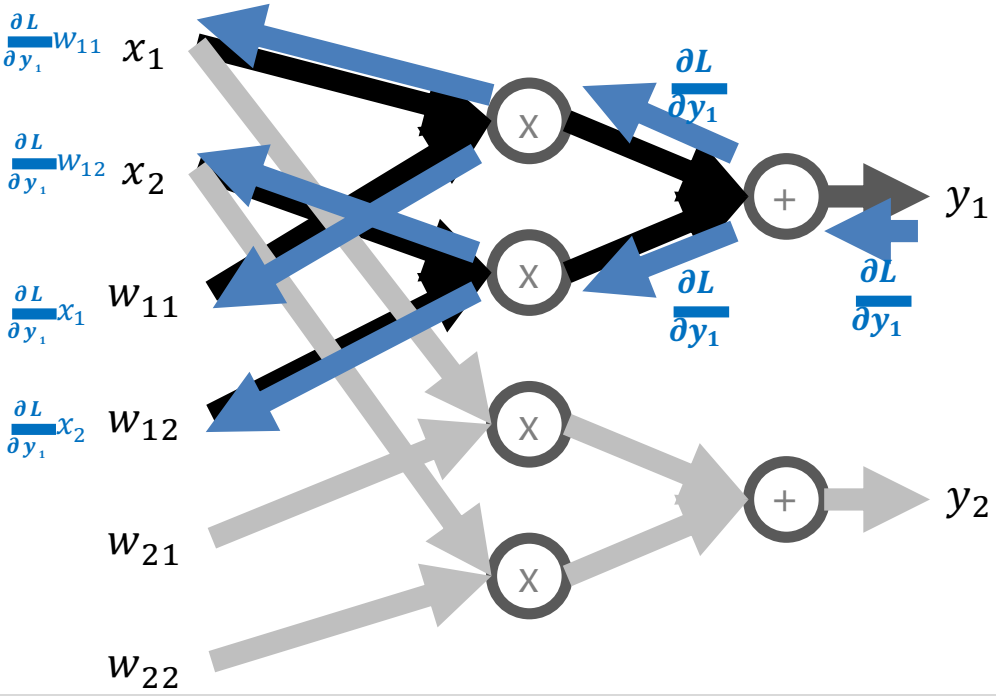
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

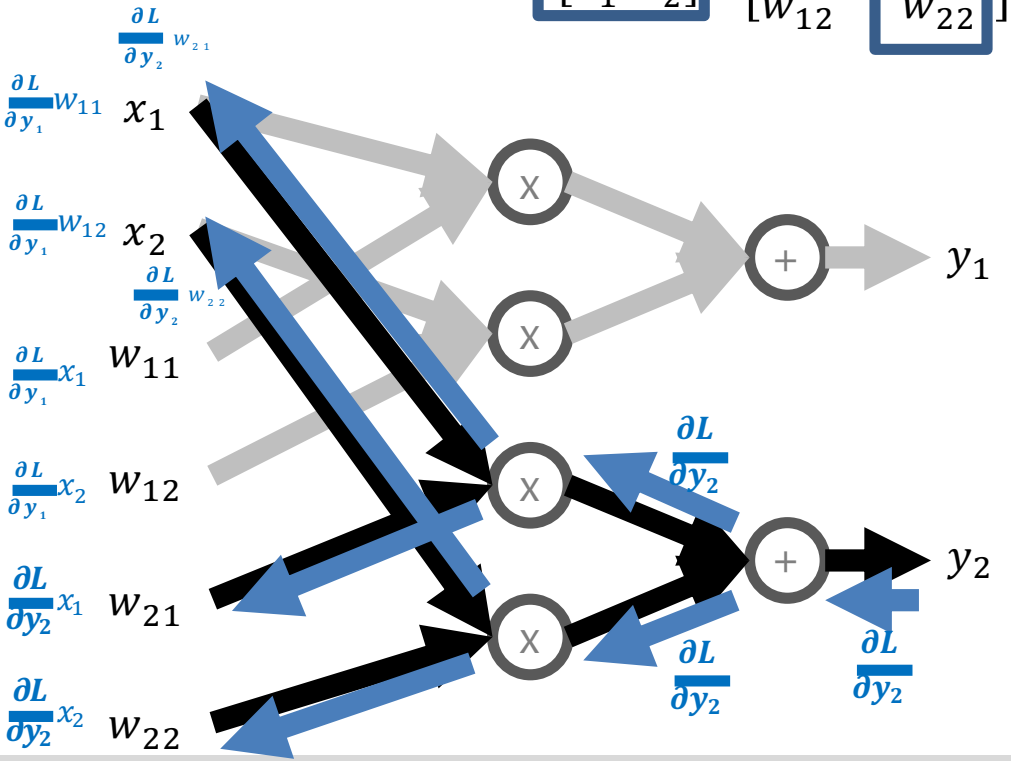
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

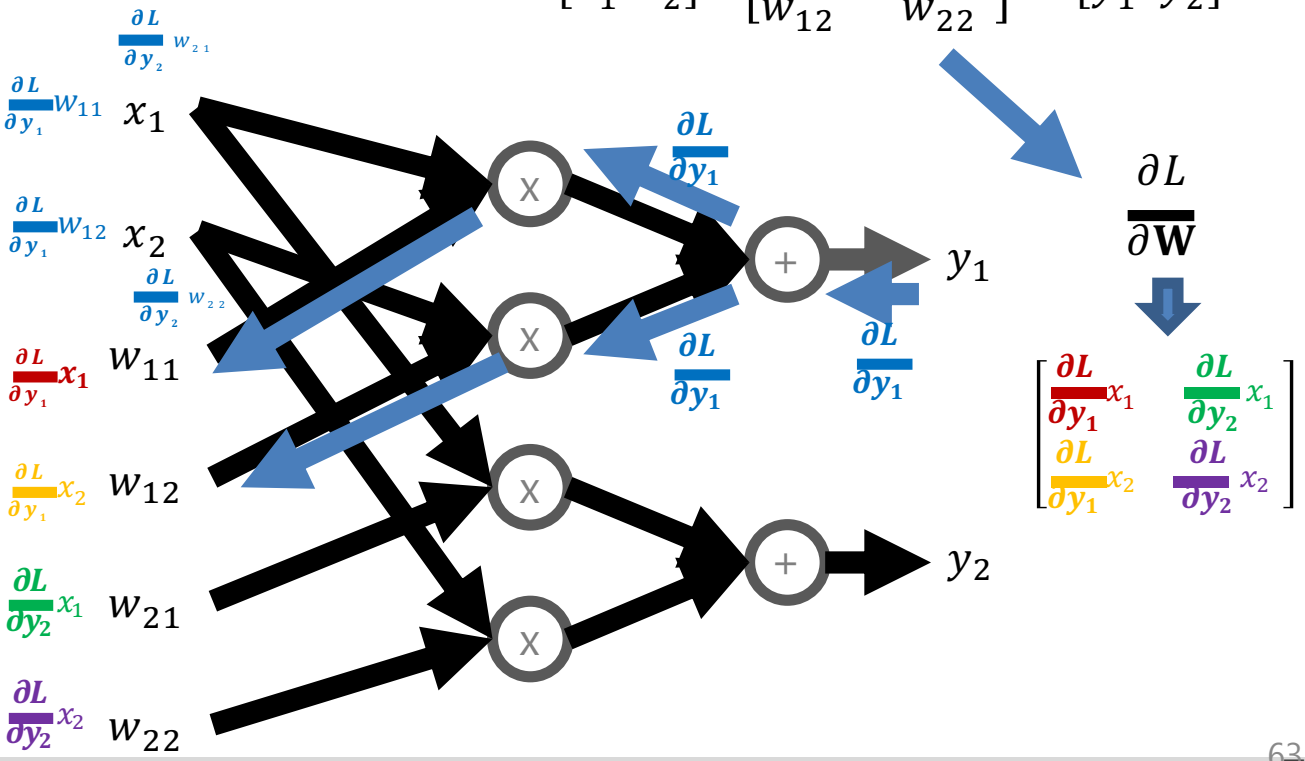
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

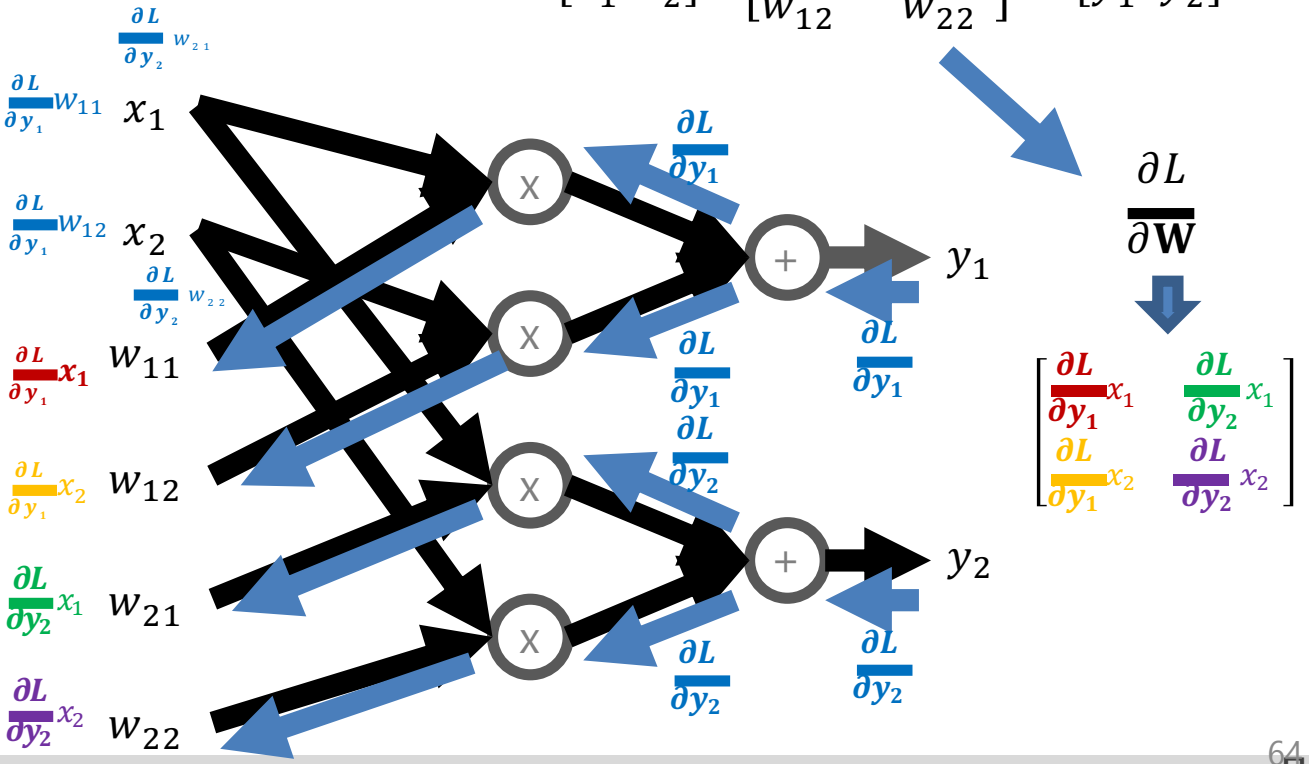
$$[x_1 \;\; x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



Computational Graph

- Affine 계층

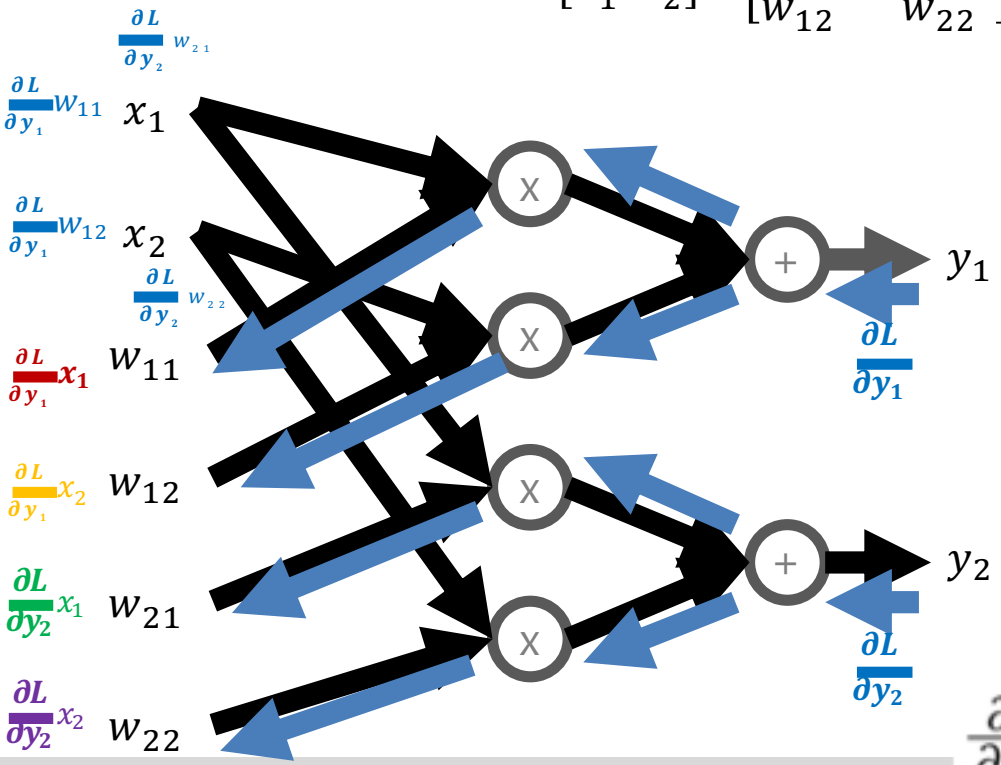
$$[x_1 \;\; x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



Computational Graph

- Affine 계층

$$[x_1 \;\; x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



$$\frac{\partial L}{\partial \mathbf{W}}$$

$$\begin{bmatrix} \frac{\partial L}{\partial y_1} x_1 & \frac{\partial L}{\partial y_2} x_1 \\ \frac{\partial L}{\partial y_1} x_2 & \frac{\partial L}{\partial y_2} x_2 \end{bmatrix}$$

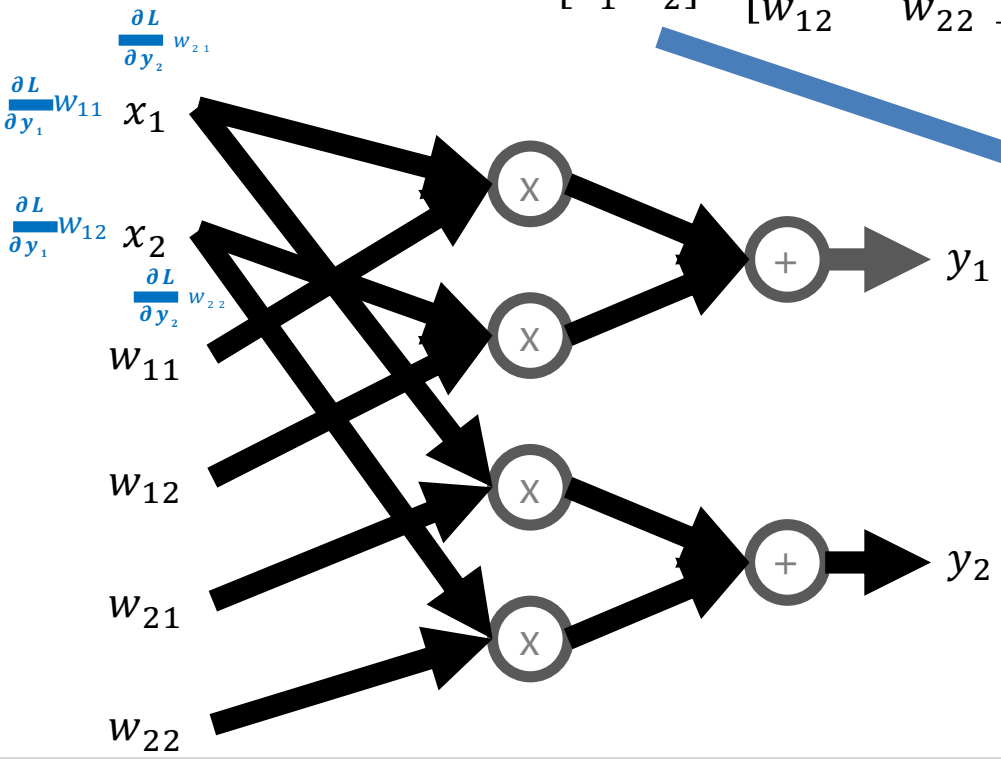
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} \end{bmatrix}$$

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

Computational Graph

- Affine 계층

$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



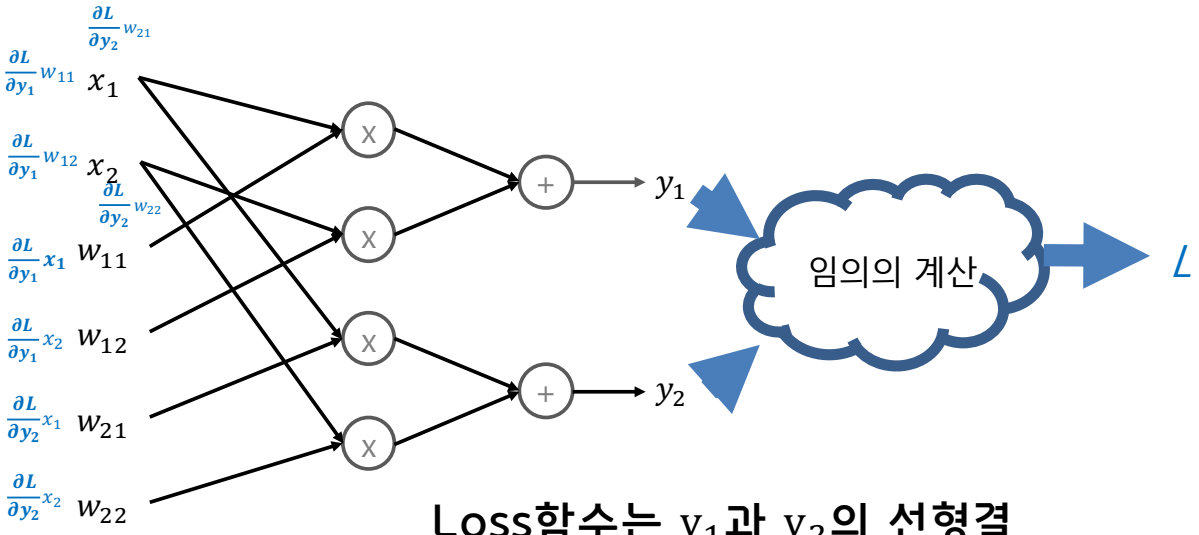
$$\frac{\partial L}{\partial \mathbf{X}}$$

2개씩 나왔어요
ㅠㅠ

Computational Graph

- Affine 계층

$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



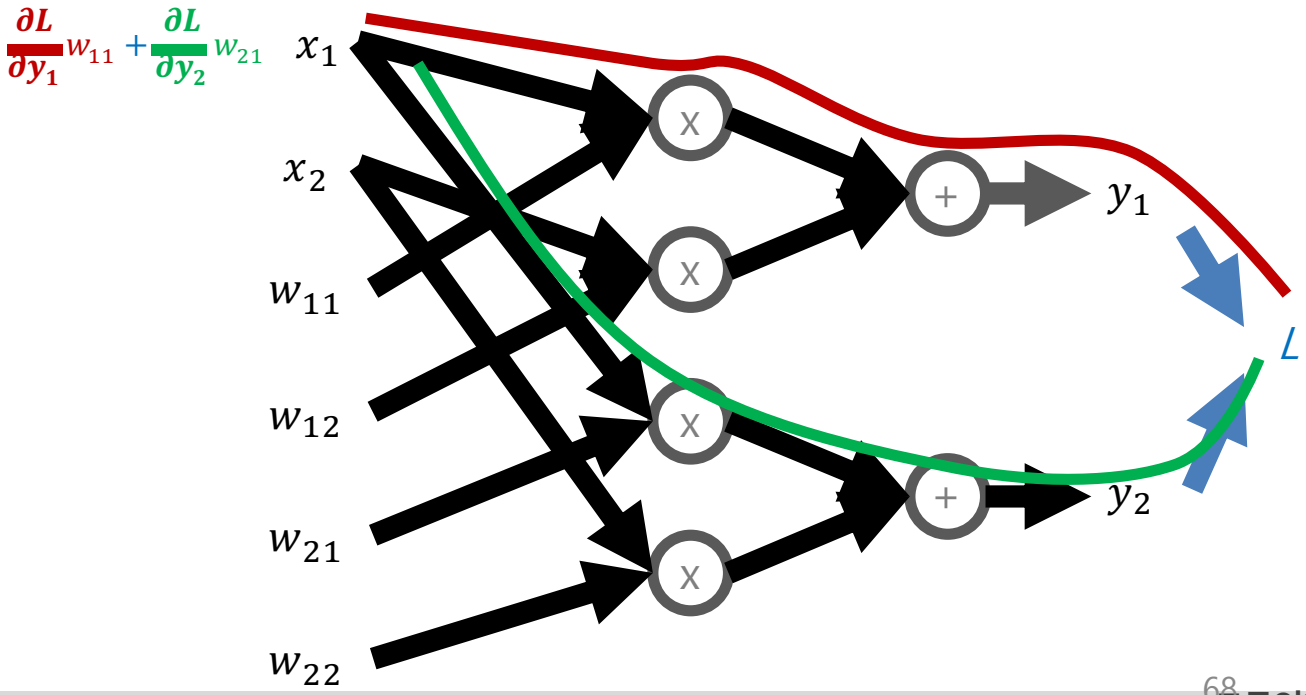
Loss함수는 y_1과 y_2의 선형결합일 겁니다

예) $L = ay_1 + by_2$

Computational Graph

- Affine 계층

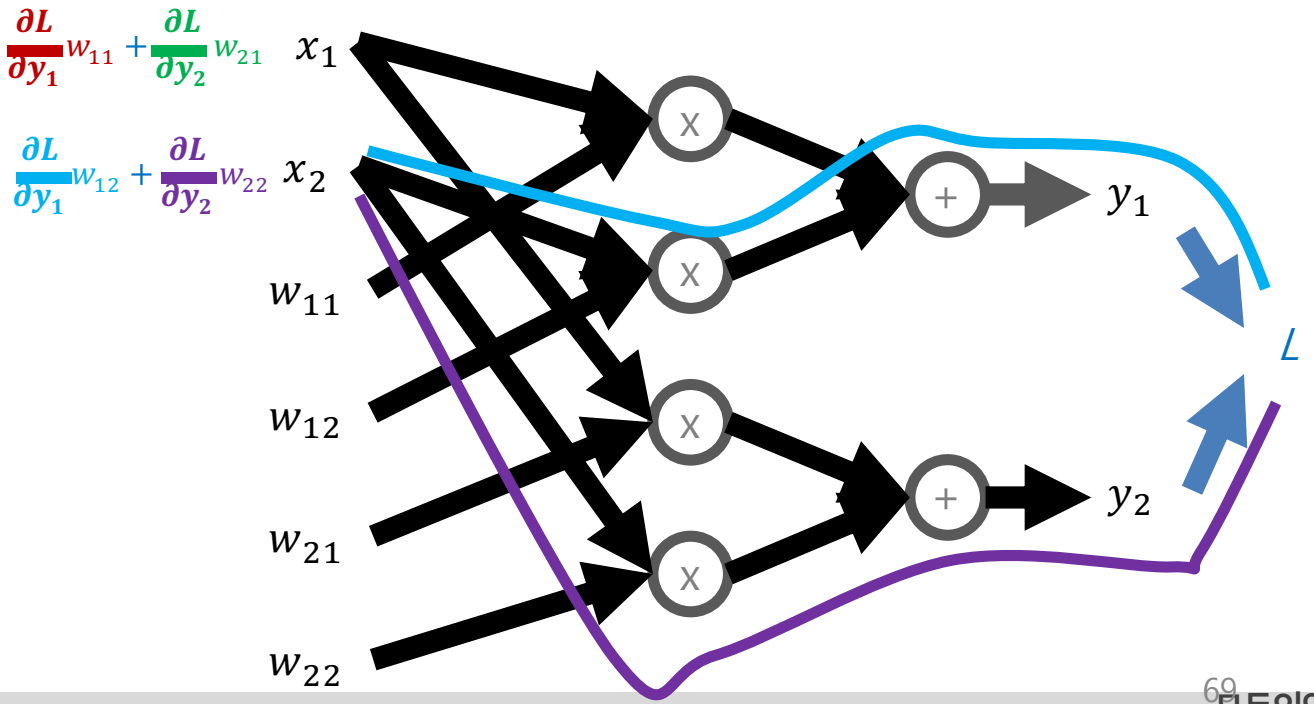
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

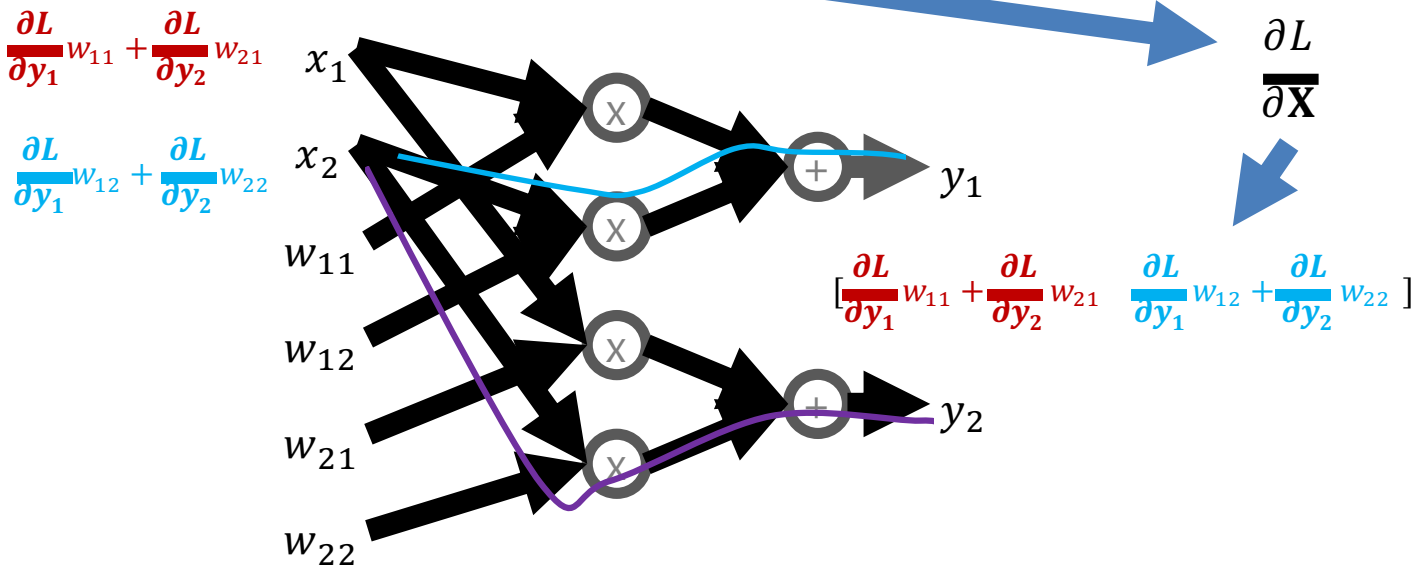
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

$$[x_1 \;\; x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



Computational Graph

- Affine 계층

$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$

$$\frac{\partial L}{\partial \mathbf{X}} = [\frac{\partial L}{\partial y_1} w_{11} + \frac{\partial L}{\partial y_2} w_{21} \quad \frac{\partial L}{\partial y_1} w_{12} + \frac{\partial L}{\partial y_2} w_{22}]$$

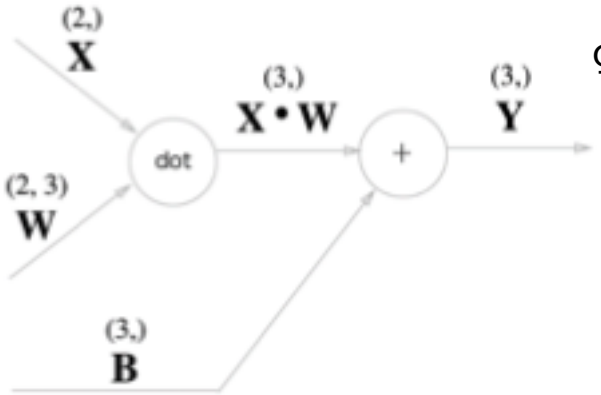
$$\left[\frac{\partial L}{\partial y_1} \quad \frac{\partial L}{\partial y_2}\right] \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

이제 다시 예제로 돌아와 봅시다

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

Computational Graph

- Affine 계층



역전파

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

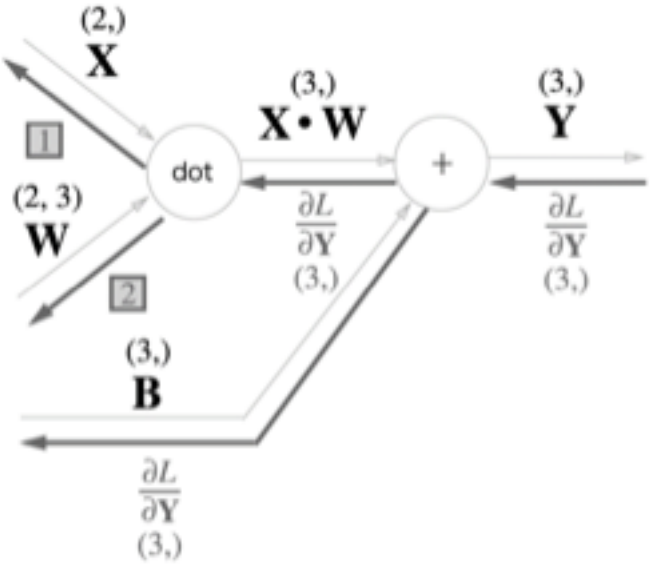
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

Computational Graph

- Affine 계층

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

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



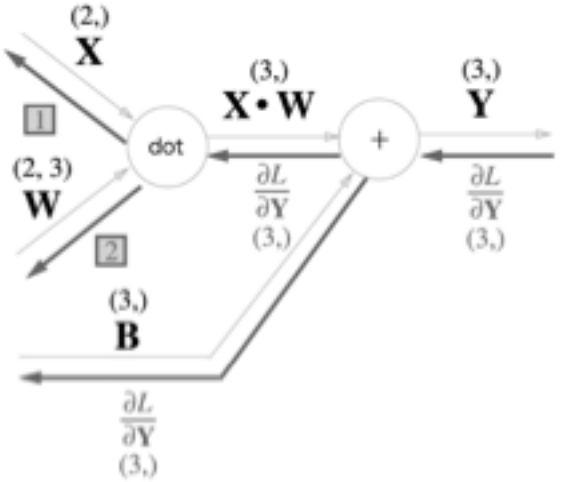
매트릭스의 형상(shape)을 잘 생각해 보면 쉽습니다

Computational Graph

- Affine 계층

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

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



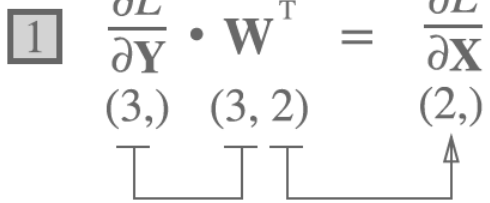
$\boldsymbol{X}$와 $\frac{\partial L}{\boldsymbol{X}}$와 형상이 같아야 하고 $\boldsymbol{W}$와 $\frac{\partial L}{\boldsymbol{W}}$는 형상이 같아야 합니다

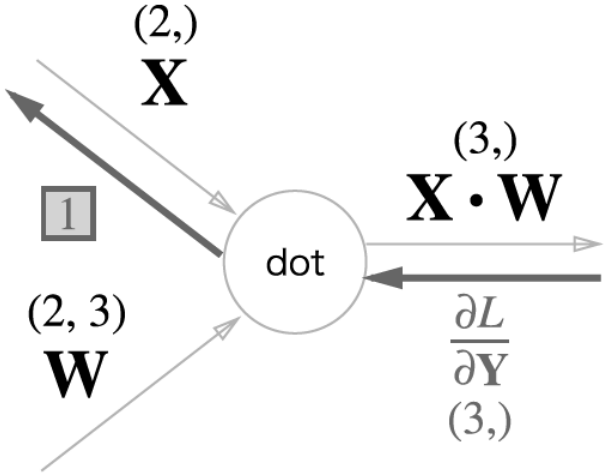
$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left(\frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n}\right)$$

Computational Graph

- Affine 계층

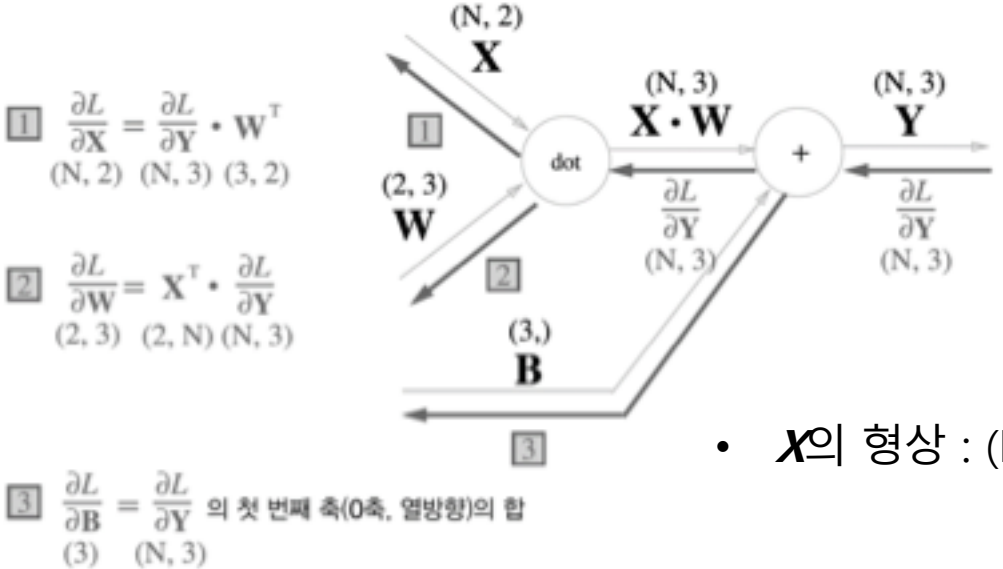




그냥 곱셈계층으로 이해하고 형상을 맞춰주면 됩니다

Computational Graph

- 배치용 Affine 계층



- ***X***의 형상 : (N, 2)

- 편향에 주의하세요

Computational Graph

- 배치용 Affine 계층
  - 데이터가 2개 일 경우(N=2)의 편향은 계산된 각각의 결과에 더해집니다

순전파의 경우

```
In [19]: X_dot_W = np.array([[0, 0, 0],[10, 10, 10]])

In [20]: B = np.array([1, 2, 3])

In [21]: X_dot_W
Out[21]:
array([[ 0,  0,  0],
       [10, 10, 10]])

In [22]: X_dot_W + B
Out[22]:
array([[ 1,  2,  3],
       [11, 12, 13]])
```

데이터 1

데이터 2

Computational Graph

- 배치용 Affine 계층
  - 데이터가 2개 일 경우($N = 2$)의 편향은 계산된 각각의 결과에 더해집니다

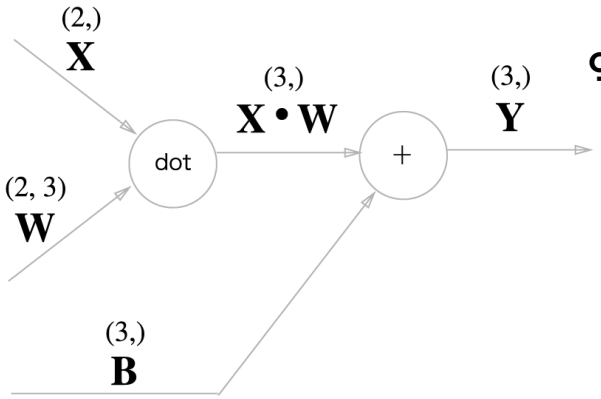
역전파의 경우

```
In [23]: dY = np.array([[1,2,3],[4,5,6]])

In [24]: dY
Out[24]:
array([[1, 2, 3],
       [4, 5, 6]])

In [25]: dB = np.sum(dY, axis=0)

In [26]: dB
Out[26]: array([5, 7, 9])
```

Computational Graph

모두의연구소

- Affine 계층



역전파

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

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

w 매트릭스가 업데이트 되겠군요

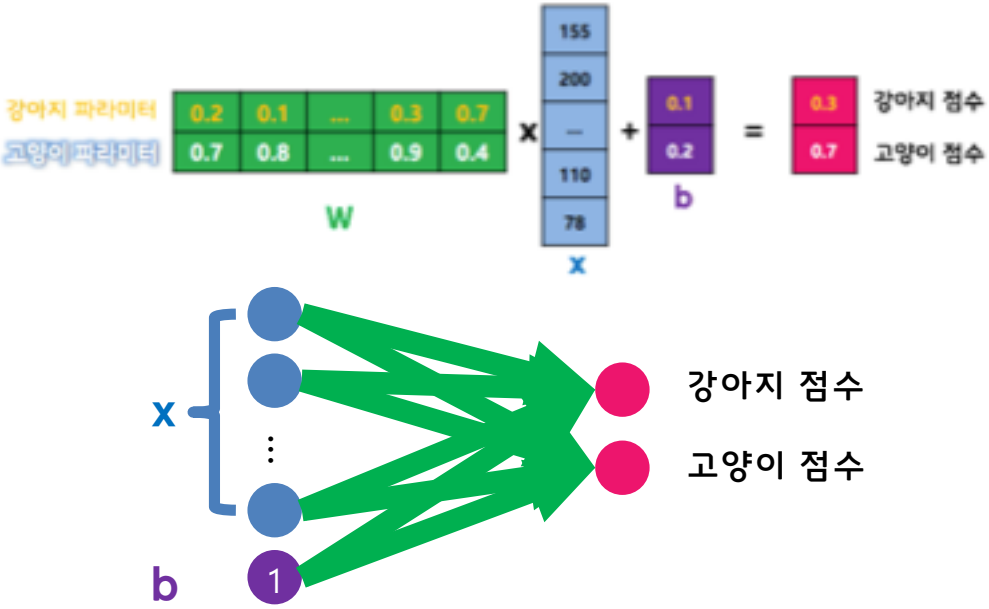
$$\mathrm{w} = \mathrm{w} - \eta \frac{\partial L}{\partial \mathrm{w}}$$

모두의연구소

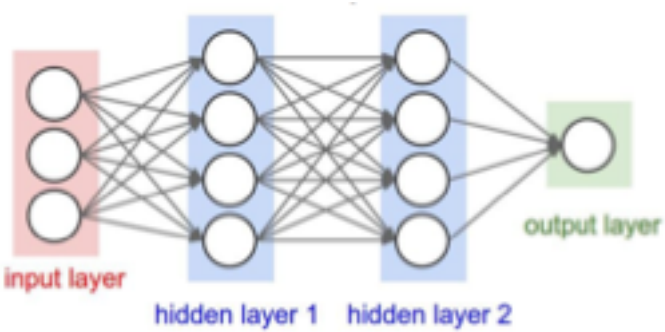
Neural Networks

Neural Networks

모두의연구소



Before



레이어를 쌓아서 더 깊게

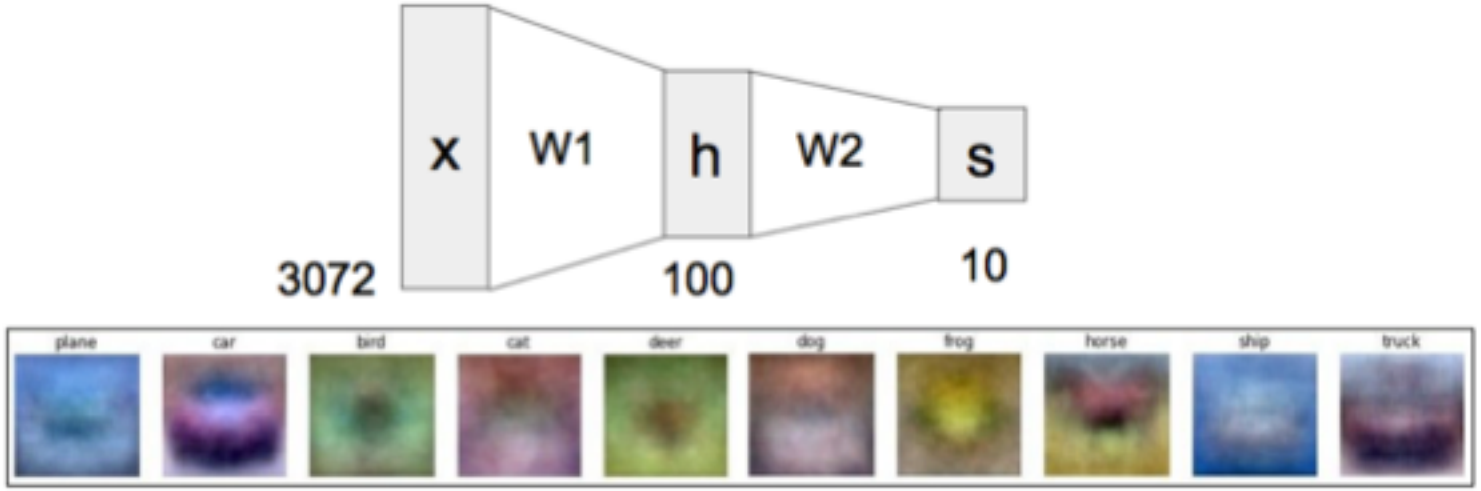
Now

Reference : Stanford University cs231n Lecture note 6

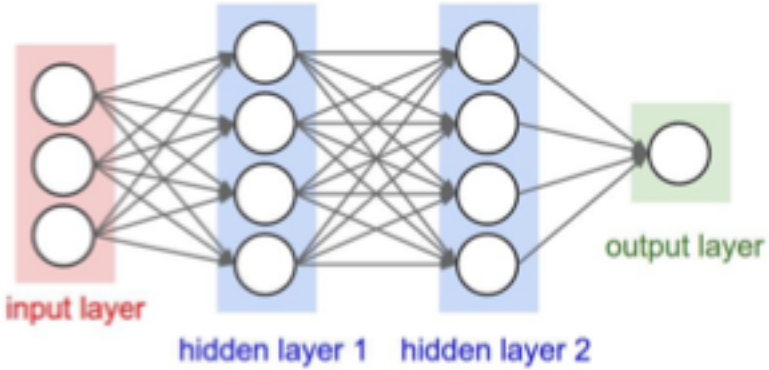
Neural Networks

모두의연구소

- Cifar-10 (영상크기 : 32x32x3, 10개의 클래스) 데이터에서의 첫번째 레이어의 시각화



Neural Networks



$$output\ layer = W_2(W_1 \mathrm{x}) = W_2 W_1 \mathrm{x} = \mathrm{W}\mathrm{x}$$

레이어를 하나 쌓는 것과 같음 → 비선형 Activation function이 필요

Neural Networks

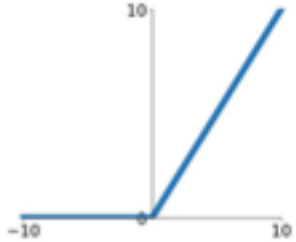
모두의연구소

(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$

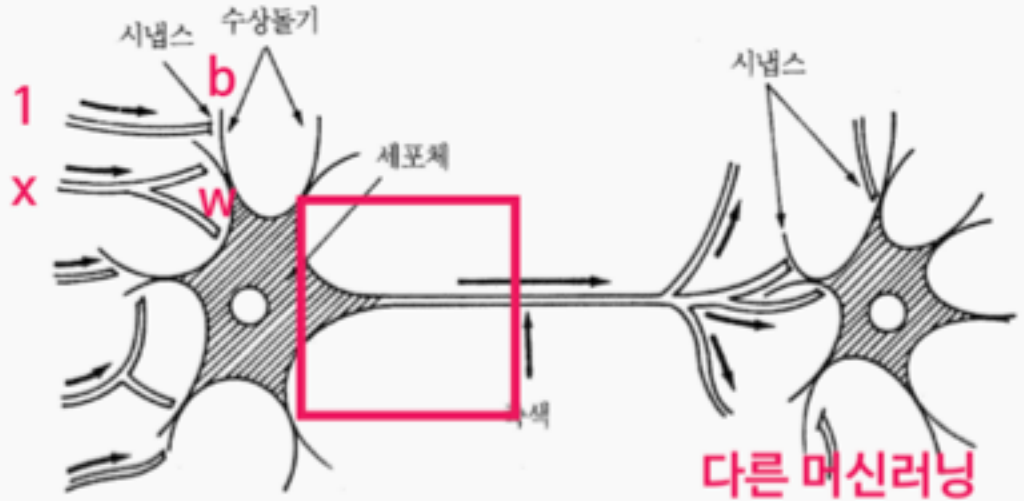
or 3-layer Neural Network

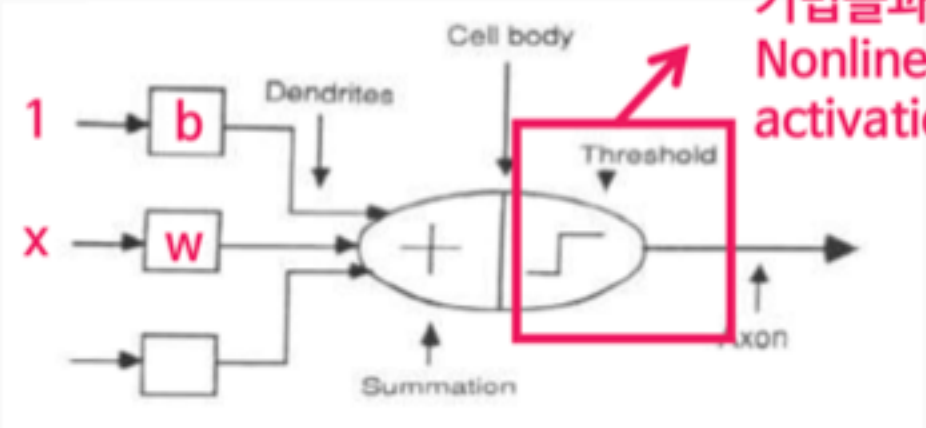
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$



ReLU
(Rectified Linear Unit)

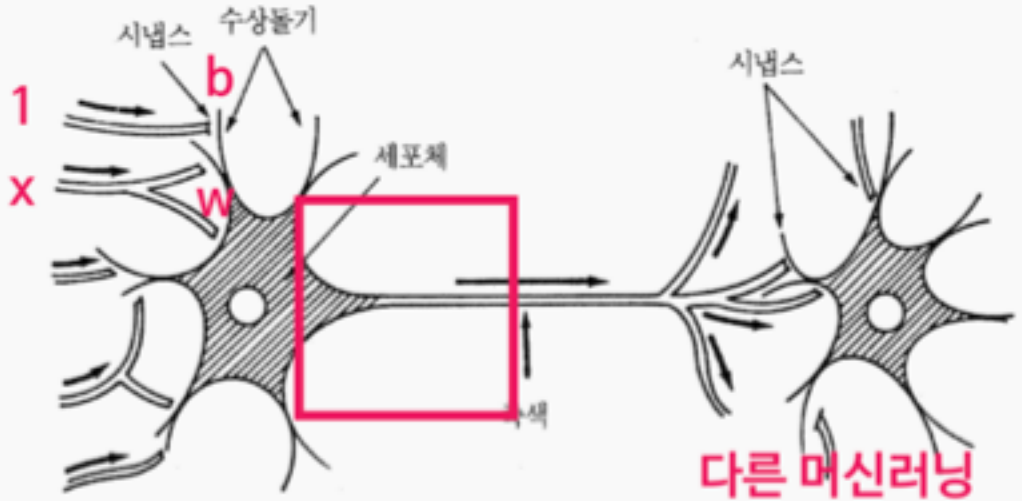
뉴런과 인공뉴런

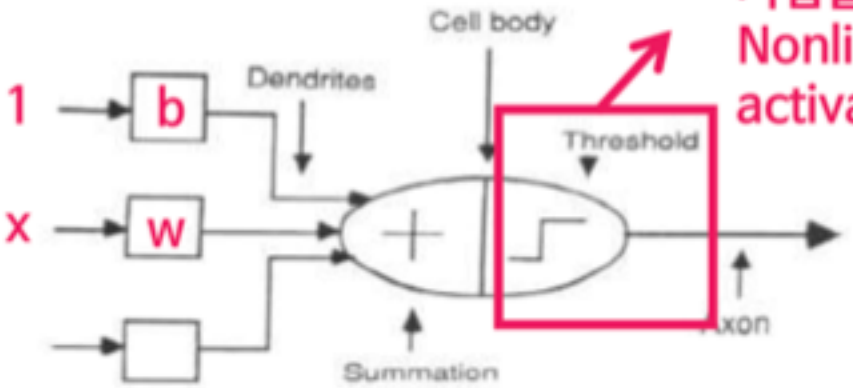




다른 머신러닝 기법들과의 차이점 1: Nonlinear(복잡한) activation function

뉴런과 인공뉴런





다른 머신러닝 기법들과의 차이점 1: Nonlinear(복잡한) activation function

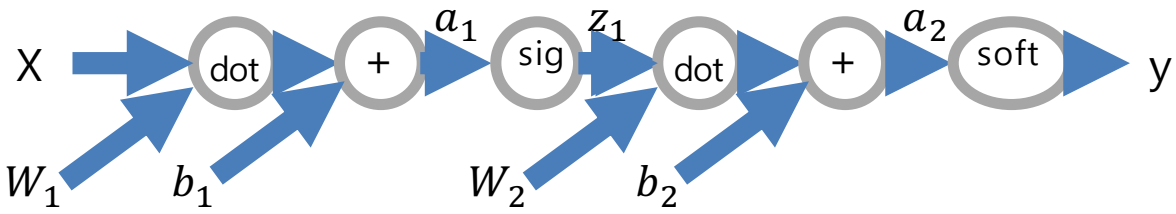
완전히 단순화 시킨 것이니 주의하세요

2레이어 네트워크의 구현 예

모두의연구소

- **Forward**
  - Activation function : Sigmoid
  - Loss function : Softmax loss

```
# forward
a1 = np.dot(x, W1) + b1
z1 = sigmoid(a1)
a2 = np.dot(z1, W2) + b2
y = softmax(a2)
```



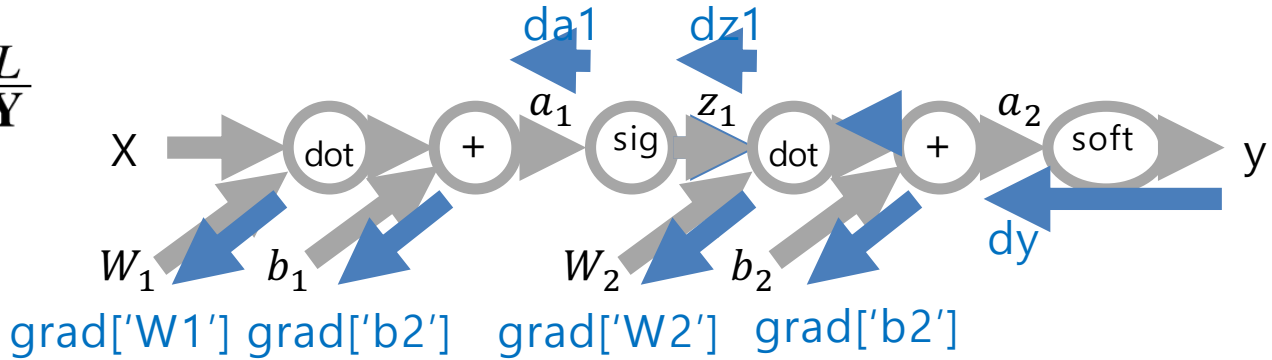
2레이어 네트워크의 구현 예

모두의연구소

- **Backward**
  - Activation function : Sigmoid
  - Loss function : Softmax loss

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

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

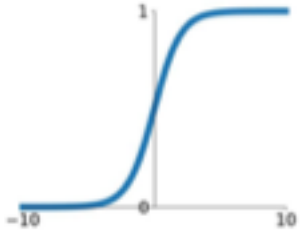


Activation function

- **ReLU vs. Sigmoid**

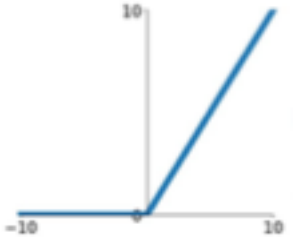
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



ReLU

$$\max(0, x)$$



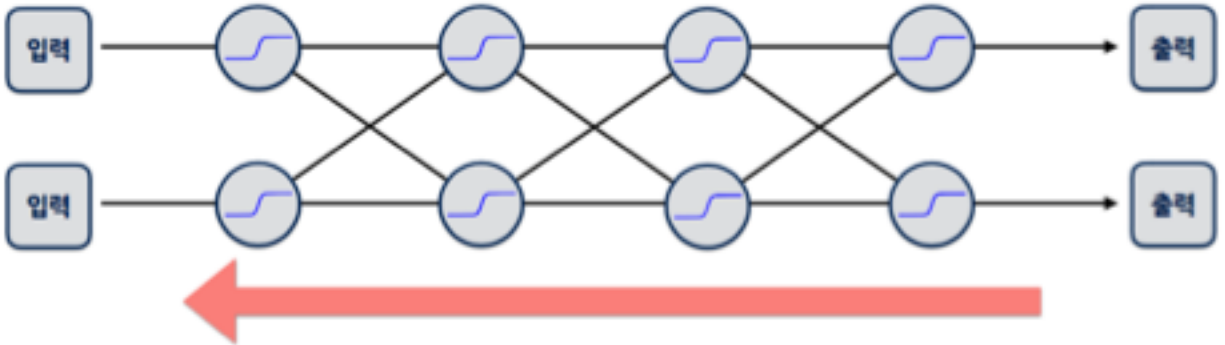
뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 "뒤로 전달")

모두의연구소

뭐를 전달하는가?

현재 내가 틀린정도를 '미분(기울기)' 한 거



미분하고, 곱하고, 더하고를 역방향으로 반복하며 업데이트한다.

근데 문제는?

모두의연구소

우리가 activation 함수로 sigmoid  를 썼다는 것

여기의 미분(기울기)는 뭐라도 있다. 다행

```
# backward
dy = (y - t) / batch_num
grads['W2'] = np.dot(z1.T, dy)
grads['b2'] = np.sum(dy, axis=0)

dz1 = np.dot(dy, W2.T)
da1 = z1*(1-z1)*dz1
    = np.dot(x.T, da1)
grads['b1'] = np.sum(dz1, axis=0)
```

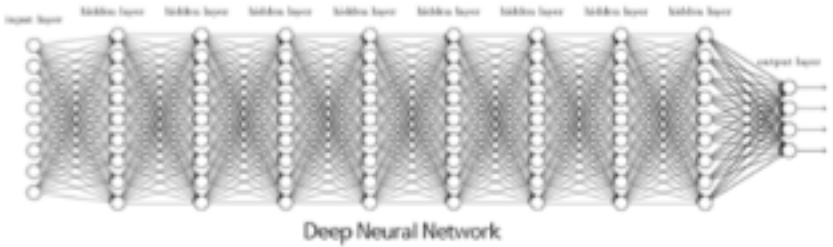
근데 여기는 기울기 0.. 이런거 중간에 곱하면 **뭔가 뒤로 전달할게 없다?!**

그런 상황에서 이걸 반복하면??????

Vanishing gradient 현상 : 레이어가 깊을 수록 업데이트가 사라져간다. 그래서 fitting이 잘 안됨(underfitting)

모두의연구소

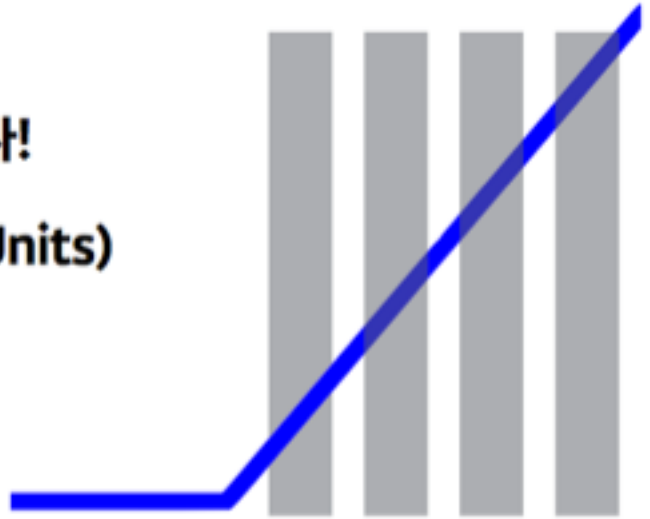
끝줄학생 ← 줄.. 줄.. 줄.. 줄좀.. 줄좀맞추자 → 교장샘



학습이 잘 안됨

사그라드는 sigmoid대신
죽지않는 activation func을 쓰자!

→ **ReLU** (Rectified Linear Units)



이녀석은 양의 구간에서 전부 미분 값(1)이 있다!

끝 줄 학생까지 이야기가 전달이 잘 되고 위치를 고친다!

배치처리

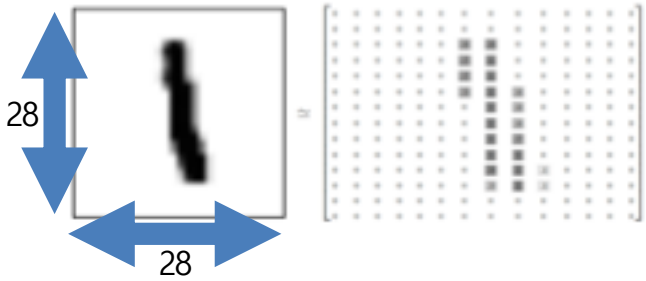
입력이 28x28x1인 784-d의 벡터 x 이고, 10개의 숫자 0~9를 예측할 경우

3 레이어 구조

	X	W1	W2	W3	→	Y
형상 :	784	784×50	50×100	100×10		10
		일치	일치	일치		

MNIST 데이터

```
In [6]: x, _ = get_data()

In [7]: network = init_network()

In [8]: W1, W2, W3 = network["W1"], network["W2"], network["W3"]

In [9]: x.shape
Out[9]: (10000, 784)

In [10]: x[0].shape
Out[10]: (784,)

In [11]: W1.shape
Out[11]: (784, 50)

In [12]: W2.shape
Out[12]: (50, 100)

In [13]: W3.shape
Out[13]: (100, 10)
```

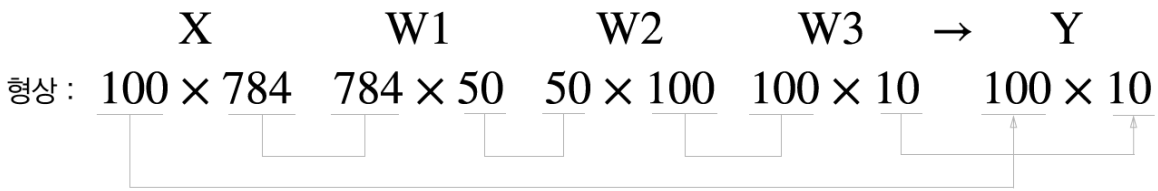


28x28x1 MNIST 데이터

배치처리

입력이 28x28x1인 784-d의 벡터 x 이고, 10개의 숫자 0~9를 예측할 경우
3 레이어 구조

- 이미지 100개를 묶어서 한번에 넘기기



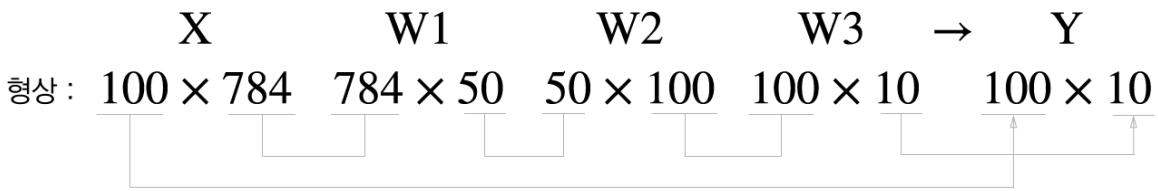
- 입력은 100x784이고 출력은 100x10이 됨
  - 이는 100장의 입력 데이터가 한번에 출력됨을 의미합니다

100장이 한번에 처리 될 수 있군요

배치처리

입력이 28x28x1인 784-d의 벡터 x 이고, 10개의 숫자 0~9를 예측할 경우
3 레이어 구조

- 이미지 100개를 묶어서 한번에 넘기기



- 입력은 100x784이고 출력은 100x10이 됨
  - 이는 100장의 입력 데이터가 한번에 출력됨을 의미합니다

100장이 한번에 처리 될 수 있군요

이처럼 하나로 묶은 데이터를 배치(batch)라고 합니다

배치처리

모두의연구소

- 배치처리의 장점
  - 1장당 처리시간을 대폭 줄여준다
    - 수치계산 라이브러리는 큰 배열을 효과적으로 처리 할 수 있도록 최적화 되어있습니다
    - 큰 배열을 한꺼번에 계산하는 것이 분할된 작은 배열을 여러번 계산하는 것보다 빠릅니다
  - 데이터 병목을 줄일 수 있습니다
    - I/O 횟수가 줄어들기 때문이죠

모두의연구소

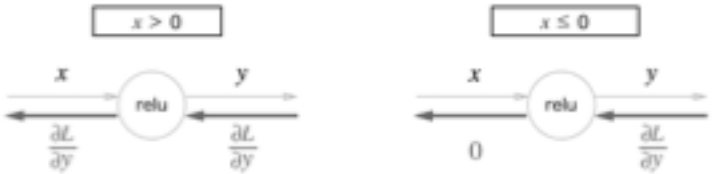
코드와 함께 보는 신경망 관련 계층

활성화 함수계층

- ReLU 계층

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

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




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

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

        return out

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

        return dx
```

활성화 함수계층

모두의연구소

- ReLU 계층

```
In [12]: x = np.array([[1.0, -0.5],[-2.0, 3.0]])

In [13]: print(x)
[[ 1.  -0.5]
 [-2.   3. ]]

In [14]: mask = (x<=0)

In [15]: print(mask)
[[False  True]
 [ True False]]

In [16]: x[mask] = 0

In [17]: print(x)
[[ 1.  0.]
 [ 0.  3.]]
```

```
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
```

백워드 용

백워드 용 저장

저장된 마스크 이용

활성화 함수계층

모두의연구소

- Sigmoid 계층 : $y = \frac{1}{1 + \exp(-x)}$
  - /codes/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
```

출력 y를 저장

계산

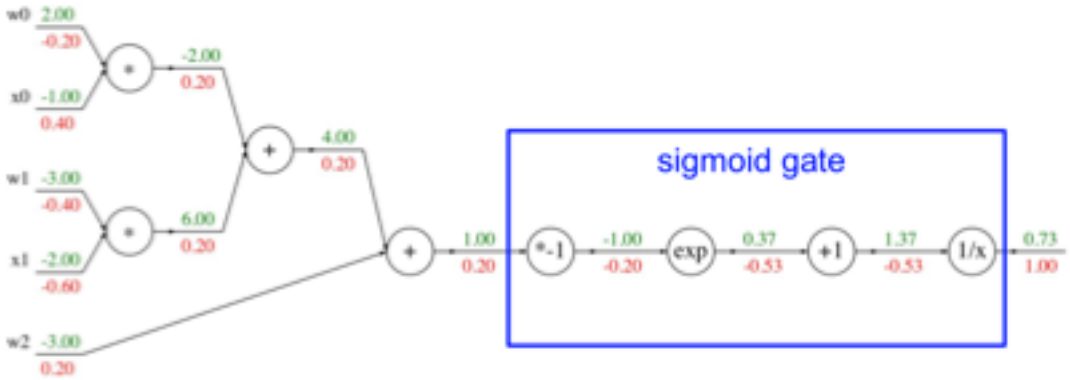
활성화 함수계층 구현하기

모두의연구소

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$\sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$



실습 1 : 2 레이어 네트워크 실습

모두의연구소

- 1) 실행 파일 (손글씨 인식) : 1day/prac2_train_neuralnet.py

```
# 1에폭당 반복 수
iter_per_epoch = max(train_size / batch_size, 1)

for i in range(iters_num):
    # 미니배치 획득
    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.gradient(x_batch, t_batch)

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

    # 학습 경과 기록
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)

    # 1에폭당 정확도 계산
    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)
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

구현할 함수

실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

```
55      def gradient(self, x, t):
56          W1, W2 = self.params['W1'], self.params['W2']
57          b1, b2 = self.params['b1'], self.params['b2']
58          grads = {}
59
60          batch_num = x.shape[0]
61
62          # forward
63          a1 = np.dot(x, W1) + b1
64          z1 = sigmoid(a1)
65          a2 = np.dot(z1, W2) + b2
66          y = softmax(a2)
67
68          # backward
69          dy = (y - t) / batch_num
70          ###################################
71          ######### Write your codes #######
72          ###################################
73          grads['W2'] = None
74          grads['b2'] = None
75
76          da1 = None
77          dz1 = None
78          grads['W1'] = None
79          grads['b1'] = None
80
81          return grads
```

실습 1 : 2 레이어 네트워크 실습

모두의연구소

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py
  - 순전파를 계산 그래프로 표현해 봅시다

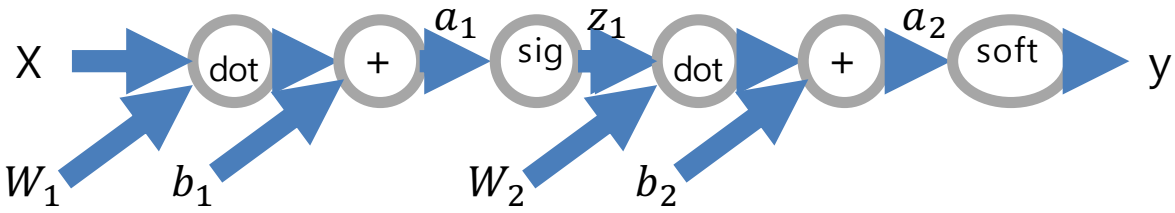
```
55      def gradient(self, x, t):
56          W1, W2 = self.params['W1'], self.params['W2']
57          b1, b2 = self.params['b1'], self.params['b2']
58          grads = {}
59
60          batch_num = x.shape[0]
61
62          # forward
63          a1 = np.dot(x, W1) + b1
64          z1 = sigmoid(a1)
65          a2 = np.dot(z1, W2) + b2
66          y = softmax(a2)
```

실습 1 : 2 레이어 네트워크 실습

모두의연구소

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

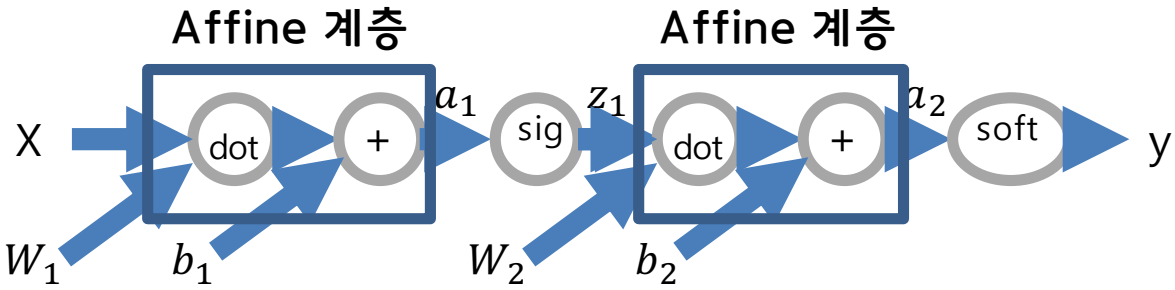
```
62          # forward
63          a1 = np.dot(x, W1) + b1
64          z1 = sigmoid(a1)
65          a2 = np.dot(z1, W2) + b2
66          y = softmax(a2)
```



실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

```
62          # forward
63          a1 = np.dot(x, W1) + b1
64          z1 = sigmoid(a1)
65          a2 = np.dot(z1, W2) + b2
66          y = softmax(a2)
```

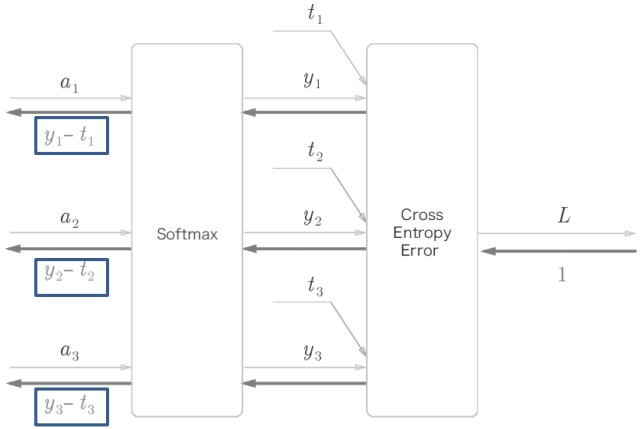


실습 1 : 2 레이어 네트워크 실습

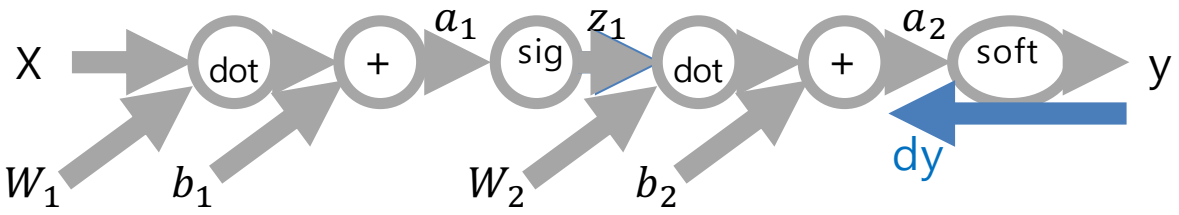
모두의연구소

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

```
68        # backward
69        dy = (y - t) / batch_num
70        ##################################
71        ######### Write your codes #######
72        ##################################
73        grads['W2'] = None
74        grads['b2'] = None
75
76        da1 = None
77        dz1 = None
78        grads['W1'] = None
79        grads['b1'] = None
80
81        return grads
```



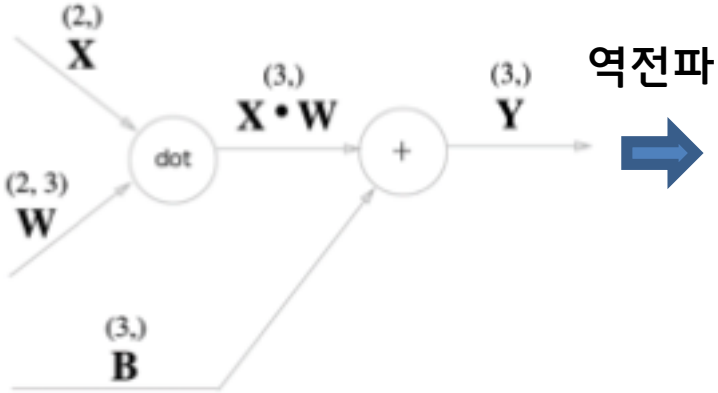
Softmax loss 부분은 이미 미분 해 두었습니다



실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

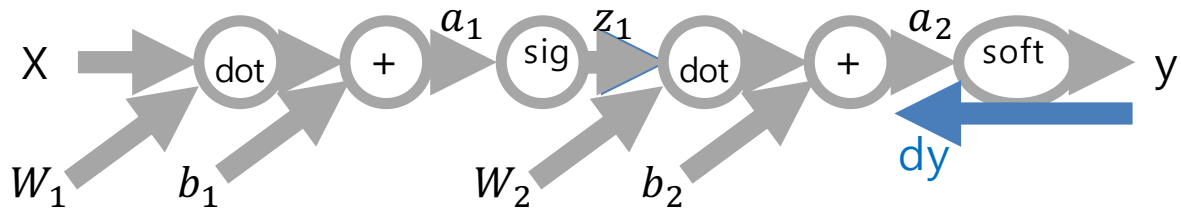
Affine 계층의 역전파를 기억해 봅시다



역전파

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

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



실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

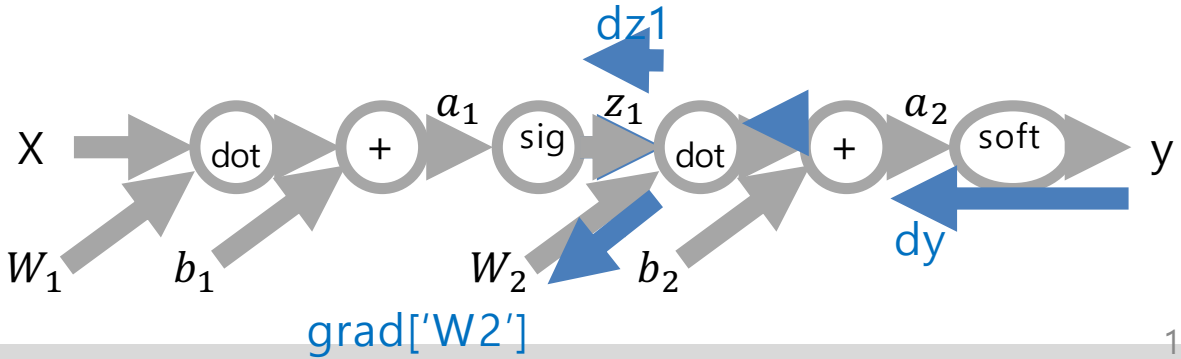
모두의연구소

Affine 계층의 역전파를 기억해 봅시다

```
68    # backward
69    dy = (y - t) / batch_num
70    ##################################
71    ######### Write your codes #######
72    ##################################
73    grads['W2'] = None
74    grads['b2'] = None
75
76    dz1 = None
77    da1 = None
78    grads['W1'] = None
79    grads['b1'] = None
```

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

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



실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

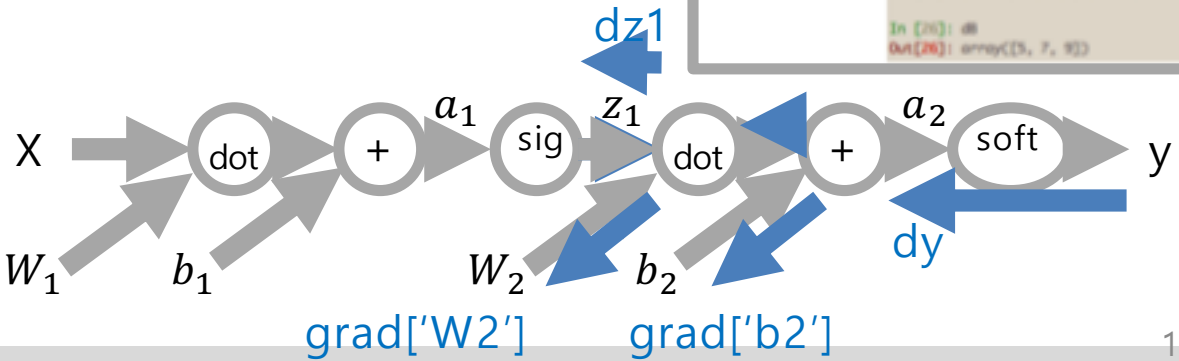
```
68    # backward
69    dy = (y - t) / batch_num
70    ###################################
71    ######### Write your codes ########
72    ###################################
73    grads['W2'] = None
74    grads['b2'] = None
75
76    dz1 = None
77    da1 = None
78    grads['W1'] = None
79    grads['b1'] = None
```

Affine / Softmax 계층 구현하기

- 배치용 Affine 계층
  - 데이터가 2개 일 경우(N=2)의 편향은 계산된 각각의 결과에 더해집니다

역전파의 경우

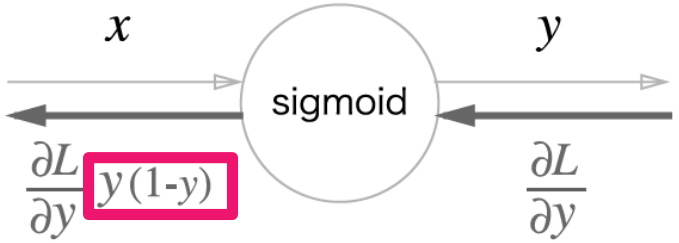
```
In [23]: dY = np.array([[1,2,3],[4,5,6]])

In [24]: dY
Out[24]:
array([[1, 2, 3],
       [4, 5, 6]])

In [25]: dB = np.sum(dY, axis=0)

In [26]: dB
Out[26]: array([5, 7, 9])
```

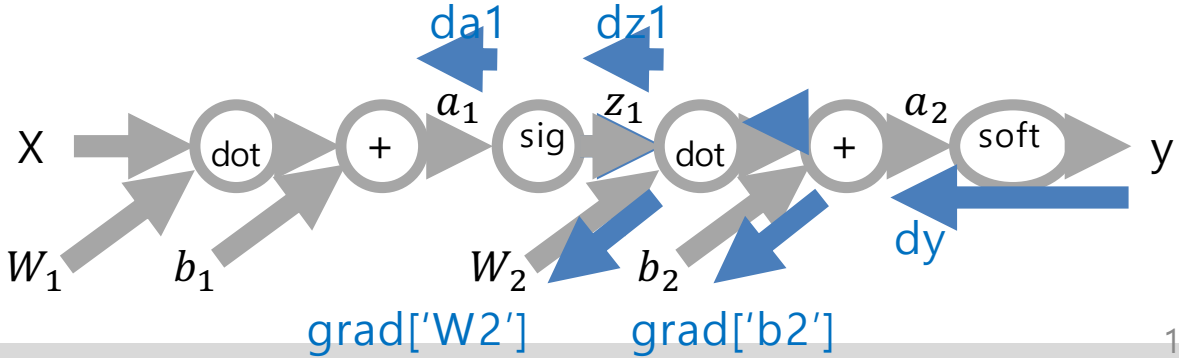


실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

```
68    # backward
69    dy = (y - t) / batch_num
70    ###################################
71    ######### Write your codes ########
72    ###################################
73    grads['W2'] = None
74    grads['b2'] = None
75
76    dz1 = None
77    da1 = None
78    grads['W1'] = None
79    grads['b1'] = None
```





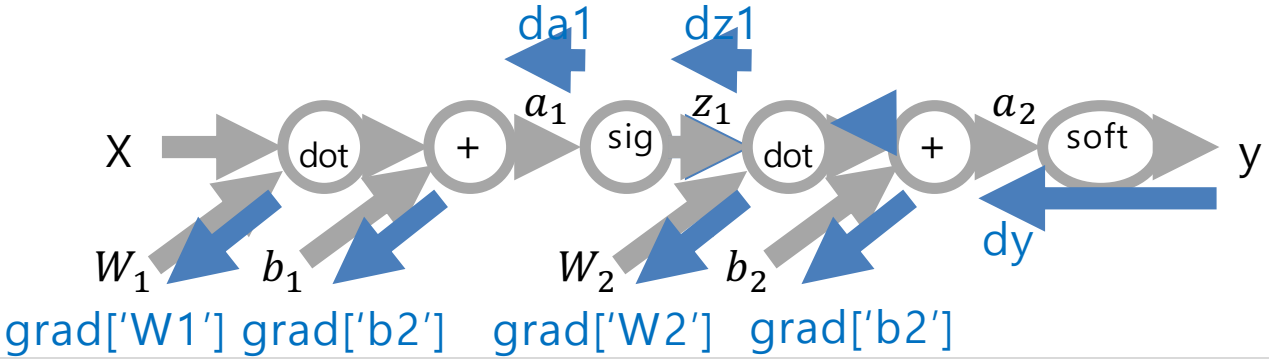
실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 1day/prac2_simple_two_Layer_net.py

```
68    # backward
69    dy = (y - t) / batch_num
70    ##################################
71    ######### Write your codes #######
72    ##################################
73    grads['W2'] = None
74    grads['b2'] = None
75
76    dz1 = None
77    da1 = None
78    grads['W1'] = None
79    grads['b1'] = None
```

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

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



실습 2 : 2 레이어 네트워크 모듈화

- 1) 실행파일 : 2day/prac2_train_neuralnet.py

이전과 같은
API

```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)

import numpy as np
from dataset.mnist import load_mnist
from prac2_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_per_epoch = max(train_size / batch_size, 1)

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)
        print(train_acc, test_acc)
```

실습 2 : 2 레이어 네트워크 모듈화

모두의연구소

- 2) 구현파일 : 2day/prac2_two_layer_net.py

```python
# coding: utf-8
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()  # Dictionary인데 순서를 갖고 있음
        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
```

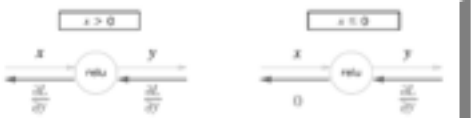
common/layers.py 참조

실습 2 : 2 레이어 네트워크 모듈화

모두의연구소

- 2) 구현파일 : 2day/prac2_two_layer_net.py
  # common/layers.py

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



```python
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
```

```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
```

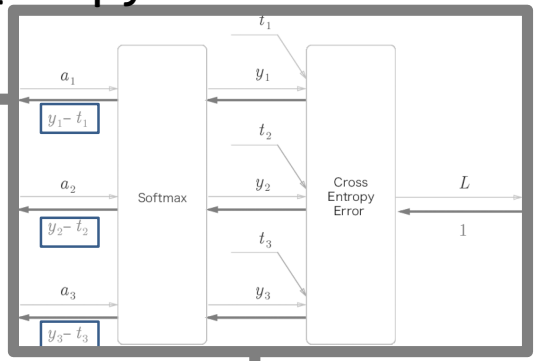
$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^{\mathrm{T}}$$

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

실습 2 : 2 레이어 네트워크 모듈화

- 2) 구현파일 : 2day/prac2_two_layer_net.py

 # common/layers.py




```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
```

실습 2 : 2 레이어 네트워크 모듈화

- 2) 구현파일 : 2day/prac2_two_layer_net.py

```
59      def gradient(self, x, t):
60          # forward
61          self.loss(x, t)
62
63          # backward
64          dout = 1
65          dout = self.lastLayer.backward(dout)
66
67          layers = list(self.layers.values())
68          layers.reverse()
69          ####################################
70          #########  Write you code #########
71          ####################################
72          for layer in layers:
73              dout = None
74          #########  end of you code #########
75          # 결과 저장
76          grads = {}
77          grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
78          grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
79
80          return grads
```

여기만 구현하면 됩니다

지금까지 살펴본 내용

모두의연구소

- 분류기의 구성
  - Score function : Wx+b
  - Loss function : Score Function의 잘못 분류된 정도를 측정
  - Optimization : Loss function의 값을 줄이는 방향으로 파라미터 업데이트

$$\mathbf{w} = \mathbf{w} - \eta \frac{\partial L}{\partial \mathbf{w}}$$

박 은 수 Research Director

E-mail : es.park@modulabs.co.kr