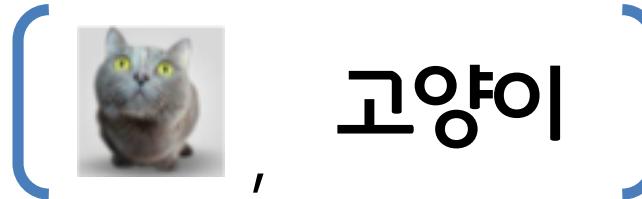


Optimization

모두의연구소
박은수 Research Director

- 데이터셋 (x, y)
- Score Function
- Loss Function



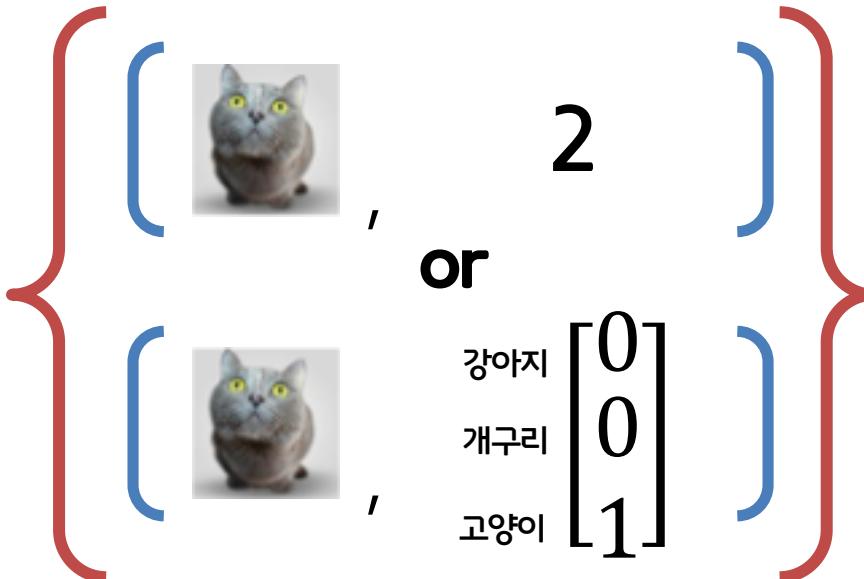
총 class가
강아지,
개구리,
고양이
라 가정



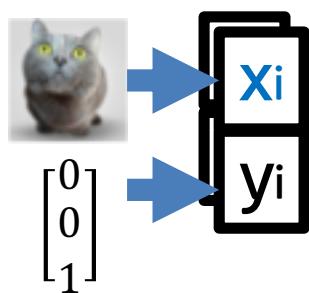
컴퓨터가 연산 가능한 형태로

인덱스 번호

One-hot Encoding



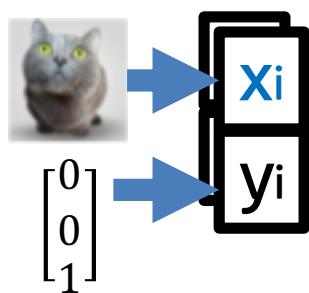
- 데이터셋 (x, y)
- Score Function
- Loss Function



- 데이터 셋 (x, y)
- Score Function
- Loss Function

e.g

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



- 데이터 셋 (x, y)
- Score Function
- Loss Function

고양이가 입력이면 고양이 점
수가 높아야 함



e.g

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



0.2	0.1	...	0.3	0.7
0.7	0.8	...	0.9	0.4
0.1	0.9	...	0.7	0.6

W

155
200
...
110
78

X

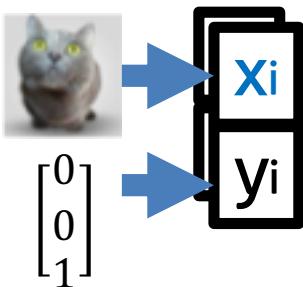
0.1
0.2
0.8

b

-2.85
0.86
0.28

S

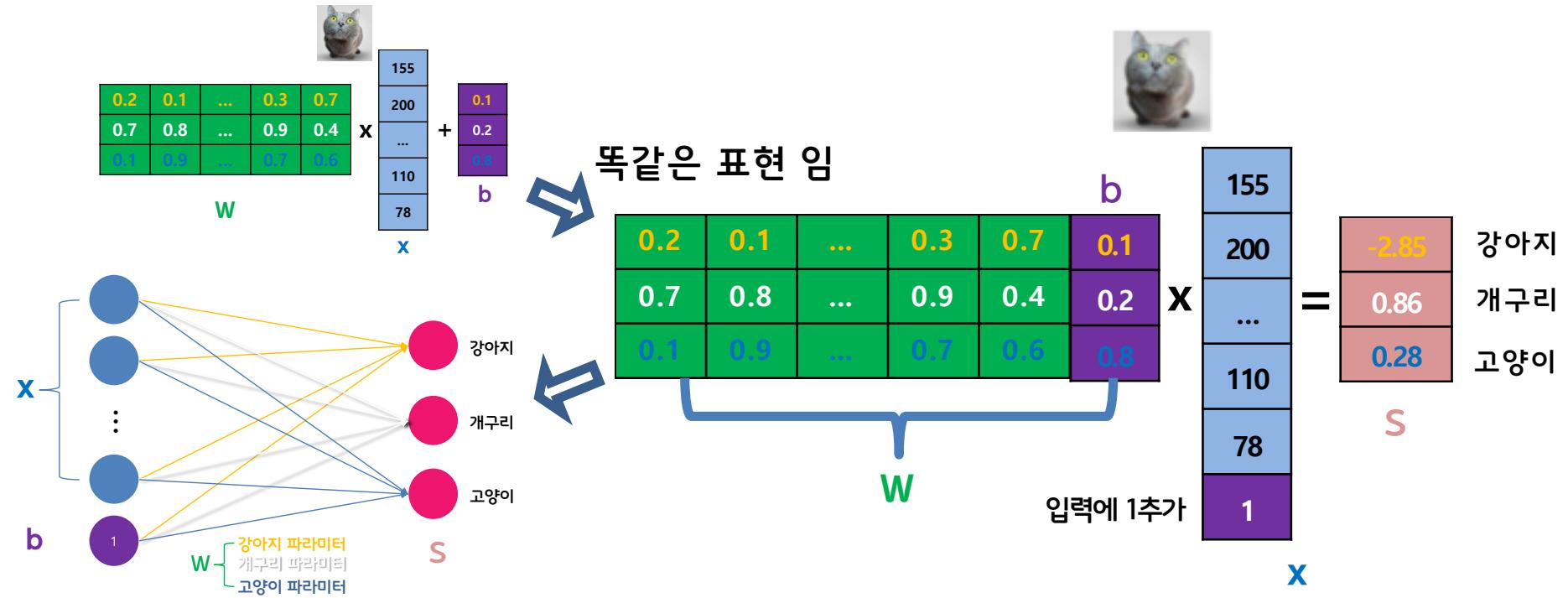
강아지 점수
개구리 점수
고양이 점수



- 데이터 셋 (x, y)
- Score Function
- Loss Function

e.g

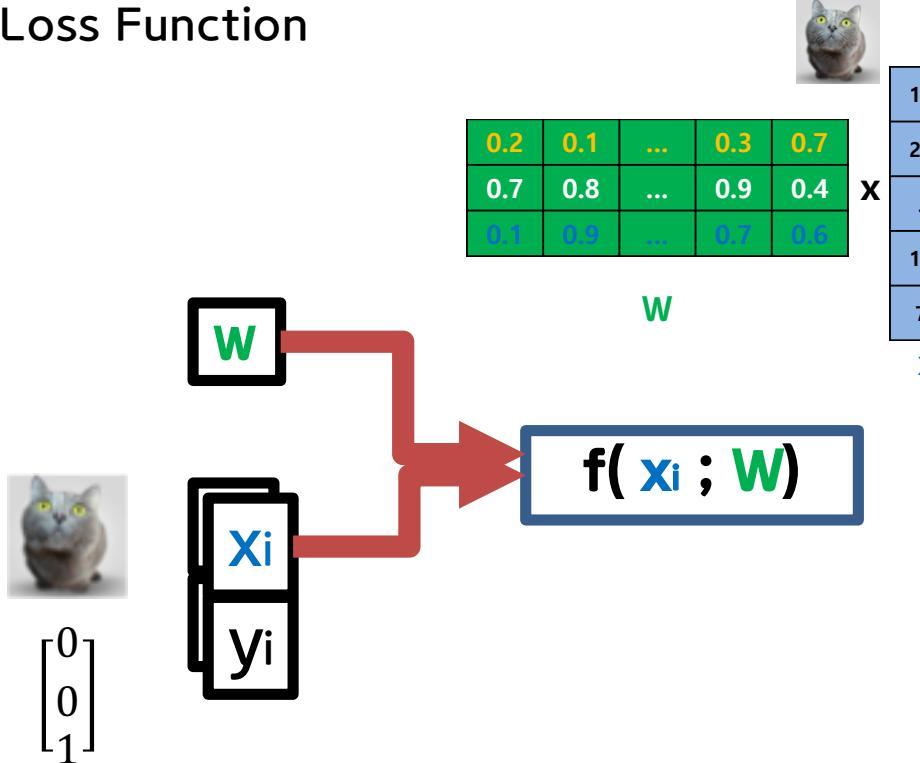
$$S = f(x; W) = Wx$$



- 데이터 셋 (x, y)
- Score Function
- Loss Function

e.g

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





$$\begin{matrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \\ 0.1 & 0.9 & \dots & 0.7 & 0.6 \end{matrix} \times \begin{matrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{matrix} + \begin{matrix} 0.1 \\ 0.2 \\ 0.8 \end{matrix} = \begin{matrix} -2.85 \\ 0.86 \\ 0.28 \end{matrix}$$

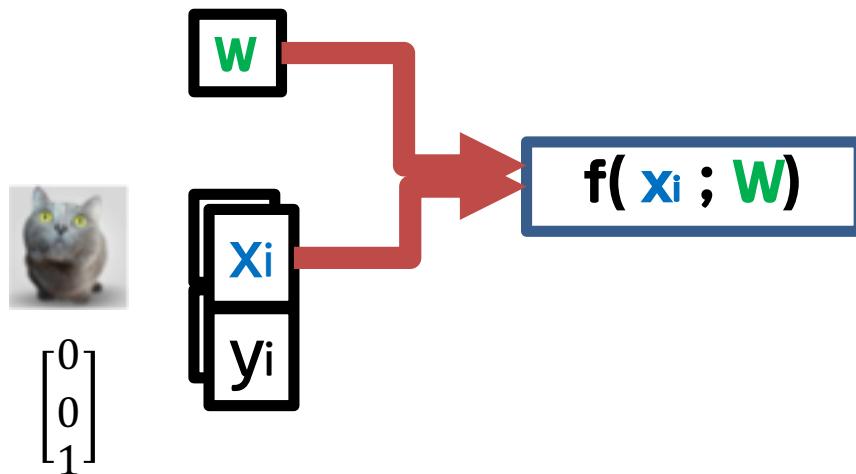
강아지 점수
개구리 점수
고양이 점수

$$f(x_i; W)$$

- 데이터 셋 (x, y)
- Score Function Softmax vs. SVM
- Loss Function

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

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



- 데이터 셋 (x, y)
- Score Function
- Loss Function

Softmax vs. SVM

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

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

 x

0.2	0.1	...	0.3	0.7
0.7	0.8	...	0.9	0.4
0.1	0.3	...	0.7	0.6

\times w

155 200 \dots 110 78

$+ b$

$=$

-2.85
0.86
0.28

s

scores = unnormalized log probabilities of the classes.

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i | X = x_i)$$

물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)
- 손님의 비율은 남자 $\frac{1}{2}$, 여자 $\frac{1}{2}$
(즉, 둘이 같음)

	$\frac{1}{2}$		$\frac{1}{2}$



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 손님의 비율은 남자 $\frac{1}{2}$, 여자 $\frac{1}{2}$
(즉, 둘이 같음)
- 남자 손님의 경우 $\frac{4}{5}$ 가 물냉면,
나머지 $\frac{1}{5}$ 이 비빔냉면

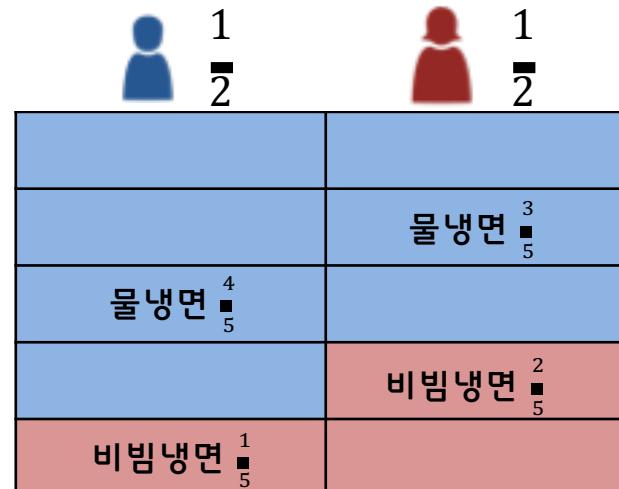
	1 $\frac{1}{2}$		1 $\frac{1}{2}$
	물냉면 $\frac{4}{5}$		
	비빔냉면 $\frac{1}{5}$		



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

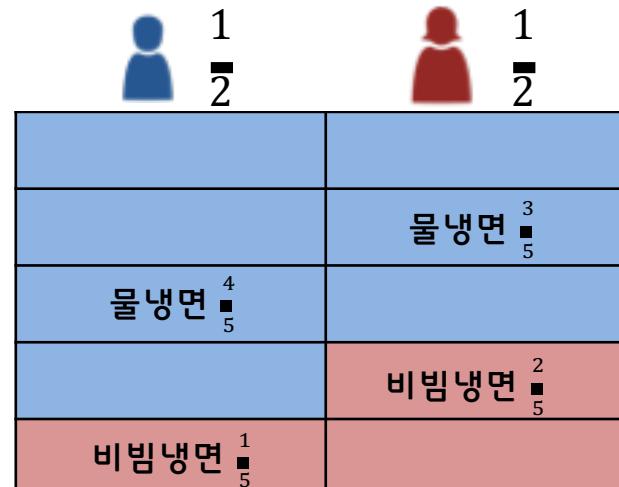
- 손님의 비율은 남자 $\frac{1}{2}$, 여자 $\frac{1}{2}$
(즉, 둘이 같음)
- 남자 손님의 경우 $\frac{4}{5}$ 가 물냉면,
나머지 $\frac{1}{5}$ 이 비빔냉면
- 여자 손님의 경우 $\frac{3}{5}$ 이 물냉면,
나머지 $\frac{2}{5}$ 가 비빔냉면



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 손님의 비율은 남자 $\frac{1}{2}$, 여자 $\frac{1}{2}$
(즉, 둘이 같음)
- 남자 손님의 경우 $\frac{4}{5}$ 가 물냉면,
나머지 $\frac{1}{5}$ 이 비빔냉면
- 여자 손님의 경우 $\frac{3}{5}$ 이 물냉면,
나머지 $\frac{2}{5}$ 가 비빔냉면

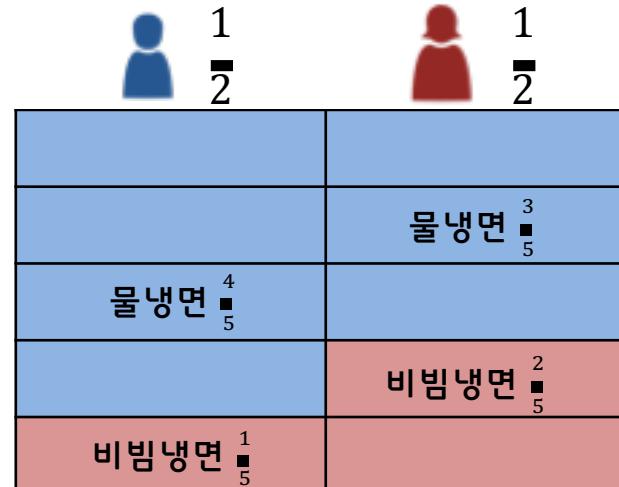


전체 분석이 끝남

물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)
- 돌아온 백종원, 한 사람이 너무 맛났는지 모자를 쓴 채, 얼굴을 푹 숙이고 계속 비빔냉면을 먹고 있는 모습을 봤다.
- 이때 비냉을 먹고 있는 사람이 남자일 가능성이 높을까 여자일 가능성이 높을까?

그림만 보고 맞춰 보세요



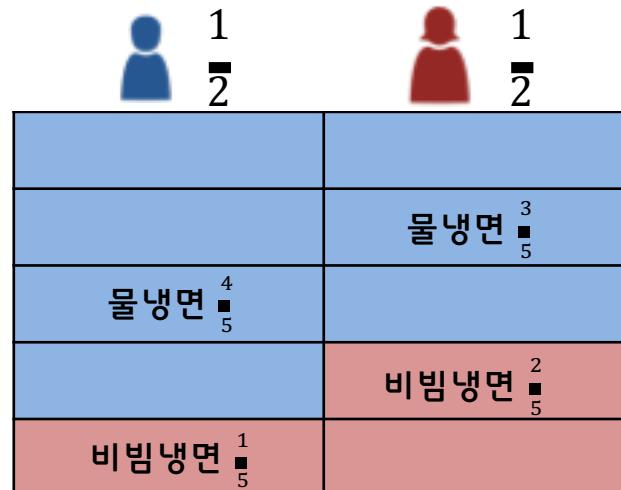
확률은 차지하는 영역의 크기



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)
- 남자 $\frac{1}{10}$ vs. 여자 $\frac{2}{10}$

따라서 여자~!!



확률은 차지하는 영역의 크기

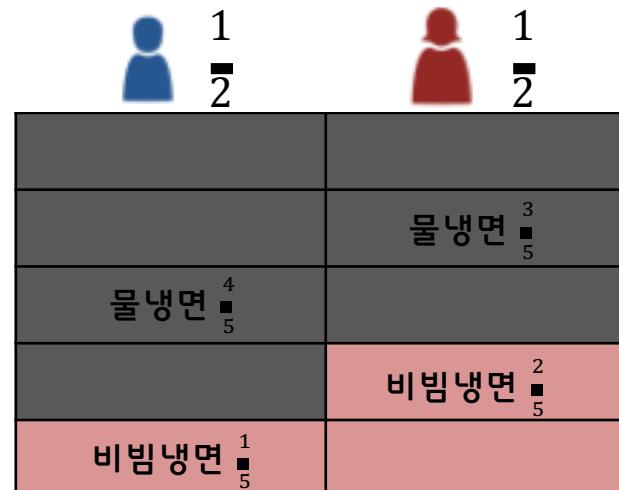


물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

Normalization

- 1) 비빔냉면 먹고 있었으니 물냉면을 볼 필요는 없고



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

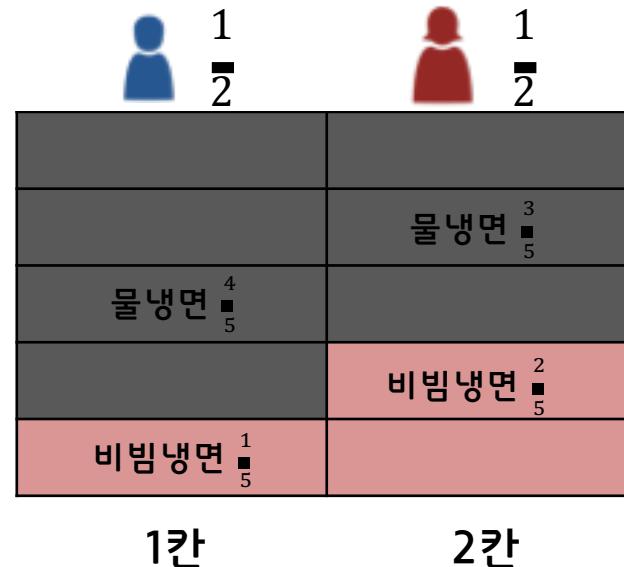
Normalization

- 1) 비빔냉면 먹고 있었으니 물 냉면을 볼 필요는 없고
- 2) 비빔냉면 먹는 사람중에서 는 $\frac{2}{3}$ 의 확률로 여자일 가능성 이 높겠군요

여자일 가능성 $\frac{2}{3}$, 남자일 가능성 $\frac{1}{3}$

여자일 가능성이 높다~!!

합하면 1, 확률로 표현 가능하군요



물냉과 비냉으로 알아보는 베이지안

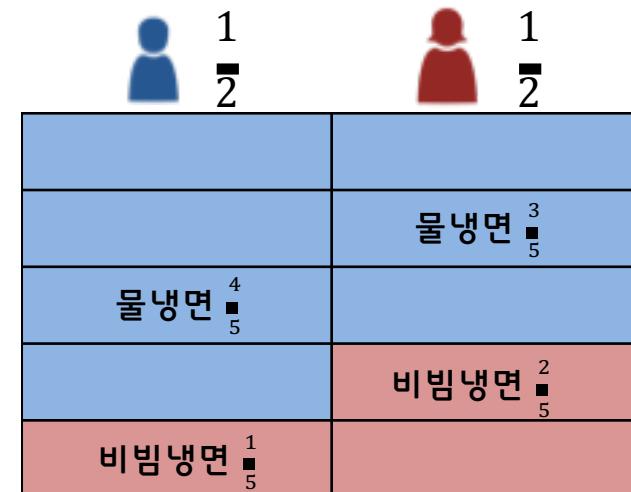
- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 베이지안을 배운 백종원

$P(Y = \text{남자}|X = \text{비냉})$ 과
 $P(Y = \text{여자}|X = \text{비냉})$ 를 구한 후
 더 높은 확률을 고르면 됨~

$$P(Y = \text{남자}|X = \text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{남자}) P(Y=\text{남자})}{P(X=\text{비냉})}$$

우리가 그림보고 얻은 결과 비
 냉먹는 사람이 남자일 확률 $\frac{1}{3}$



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 베이지안을 배운 백종원

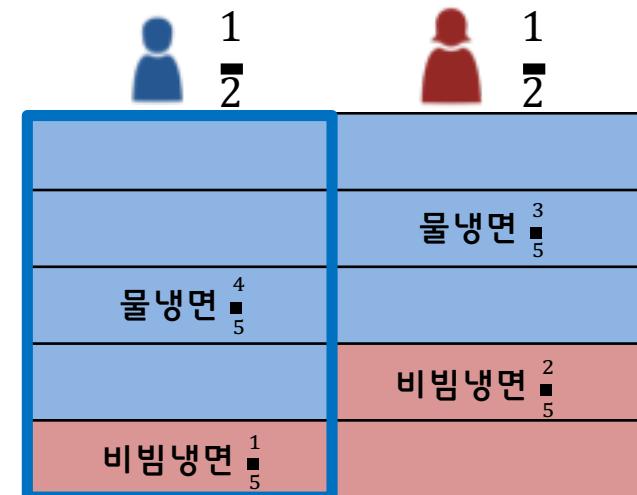
$P(Y = \text{남자} | X = \text{비냉})$ 과

$P(Y = \text{여자} | X = \text{비냉})$ 를 구한 후

더 높은 확률을 고르면 됨~

$$P(Y = \text{남자} | X = \text{비냉}) = \frac{P(X = \text{비냉} | Y = \text{남자}) P(Y = \text{남자})}{P(X = \text{비냉})}$$

우리가 그림보고 얻은 결과 비
냉먹는 사람이 남자일 확률 $\frac{1}{3}$



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 베이지안을 배운 백종원

우리가 그림보고 얻은 결과 비
냉먹는 사람이 남자일 확률 $\frac{1}{3}$

$P(Y = \text{남자} | X = \text{비냉})$ 과

$P(Y = \text{여자} | X = \text{비냉})$ 를 구한 후

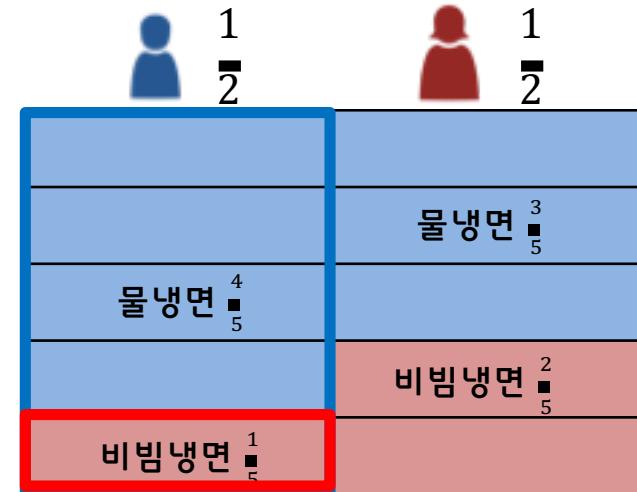
더 높은 확률을 고르면 됨~

$$\frac{1}{5} \times \frac{1}{2} = \frac{1}{10}$$

$$P(Y = \text{남자} | X = \text{비냉}) = \frac{P(X=\text{비냉} | Y=\text{남자}) P(Y=\text{남자})}{P(X=\text{비냉})}$$

$\frac{1}{10}$ 은 전체공간에서 남자가 비빔냉면 먹은 확률임

분모 $P(X = \text{비냉})$ 은 아직 계산 안함



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

- 베이지안을 배운 백종원

$P(Y = \text{남자}|X = \text{비냉})$ 과

$P(Y = \text{여자}|X = \text{비냉})$ 를 구한 후

더 높은 확률을 고르면 됨~

우리가 그림보고 얻은 결과 비냉

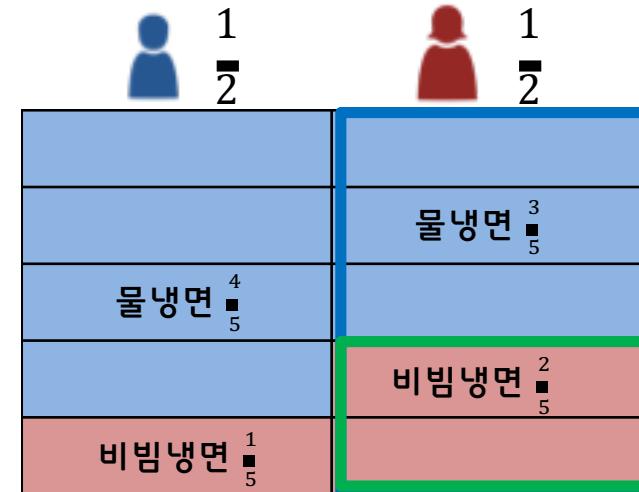
먹는 사람이 남자일 확률 $\frac{1}{3}$

$$P(Y = \text{여자}|X = \text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{여자}) P(Y=\text{여자})}{P(X=\text{비냉})}$$

$\frac{1}{10}$ 은 전체공간에서 남자가 비빔냉면 먹은 확률임

분모 $P(X = \text{비냉})$ 은 아직 계산 안함

$\frac{2}{10}$ 은 전체공간에서 여자가 비빔냉면 먹은 확률임



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)



Unnormalized probability

분모를 계산하지 않아도 누구일 확률이 높은지 맞추는 것은 가능

$$\frac{1}{5} \times \frac{1}{2} = \frac{1}{10}$$

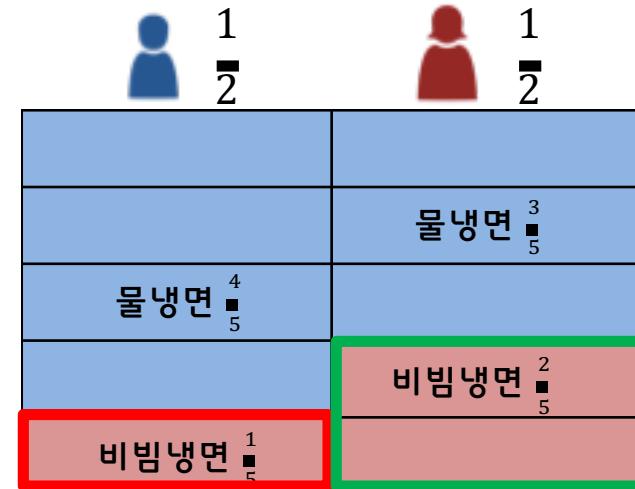
$$P(Y = \text{남자} | X = \text{비냉}) = \frac{P(X = \text{비냉} | Y = \text{남자}) P(Y = \text{남자})}{P(X = \text{비냉})}$$

$$\frac{2}{5} \times \frac{1}{2} = \frac{2}{10}$$

$$P(Y = \text{여자} | X = \text{비냉}) = \frac{P(X = \text{비냉} | Y = \text{여자}) P(Y = \text{여자})}{P(X = \text{비냉})}$$

$\frac{1}{10}$ 은 전체공간에서 남자가 비빔냉면 먹은 확률임

$\frac{2}{10}$ 은 전체공간에서 여자가 비빔냉면 먹은 확률임



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

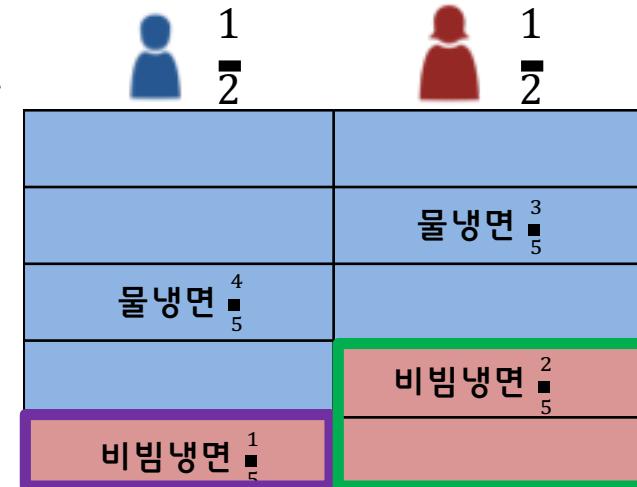


$P(X = \text{비냉})$ 은 normalization 임.

- 즉 $\frac{1}{10} + \frac{2}{10} = \frac{3}{10}$
- 전체 비냉의 확률

$$P(Y = \text{남자}|X = \text{비냉}) = \frac{P(X = \text{비냉}|Y = \text{남자}) P(Y = \text{남자})}{P(X = \text{비냉})}$$

$$P(X = \text{비냉}) = P(X = \text{비냉}|Y = \text{남자}) P(Y = \text{남자}) + P(X = \text{비냉}|Y = \text{여자}) P(Y = \text{여자})$$



물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

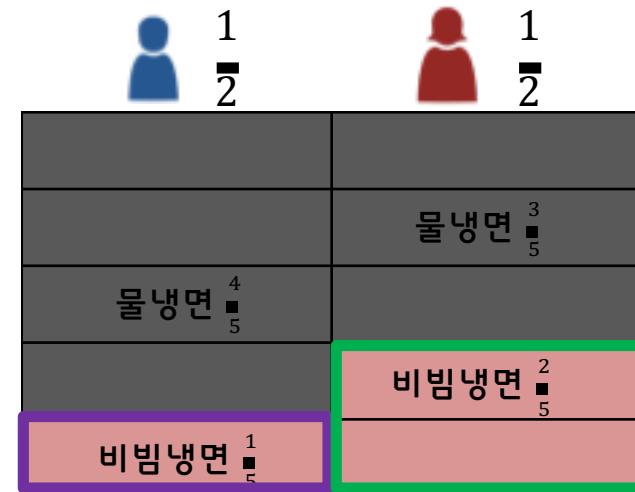


$P(X = \text{비냉})$ 은
normalization 임.

- 즉 $\frac{1}{10} + \frac{2}{10} = \frac{3}{10}$
- 전체 비냉의 확률

$$P(Y = \text{남자}|X = \text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{남자}) P(Y=\text{남자})}{P(X=\text{비냉})} = \frac{\frac{1}{10}}{\frac{3}{10}} = \frac{1}{3}$$

$$P(Y = \text{여자}|X = \text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{여자}) P(Y=\text{여자})}{P(X=\text{비냉})} = \frac{\frac{2}{10}}{\frac{3}{10}} = \frac{2}{3}$$



Normalized probability

- 데이터 셋 (x, y)
- Score Function
- Loss Function

Softmax vs. SVM

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

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

강아지 (0), 개구리 (1), 고양이 (2) 세팅에서 입력된 영상 x_i 가 고양이 일 확률은?

베이지안 정리에 의해서

$$P(Y = 2|X = x_i) = \frac{P(X = x_i|Y = 2)P(Y = 2)}{P(X)}$$

$P(X)$ 는 normalization을 위해 존재 함. 따라서

$$P(X) = P(X = x_i|Y = 0)P(Y = 0) + P(X = x_i|Y = 1)P(Y = 1) + P(X = x_i|Y = 2)P(Y = 2)$$

- 데이터 셋 (x, y)
- Score Function
- Loss Function

Softmax vs. SVM

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

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

e.g

\ln 을 적용하여 표현 : $s_k = \ln(P(X = x_i | Y = k)P(Y = k)) = f(x; W) = Wx$

우리가 만든 Score function을 unnormalized log probability로 표현한 것임

$$P(Y = k | X = x_i) = \frac{P(X=x_i|Y=k)P(Y=k)}{P(X)} = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Cross entropy는 유도는 참고하세요 : [Cross Entropy의 정확한 확률적 의미](#)

- 데이터 셋 (x, y)
- Score Function
- Loss Function

Softmax vs. SVM

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

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

$f(X_i; W)$



$x = \begin{bmatrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{bmatrix}$

$b = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.6 \end{bmatrix}$

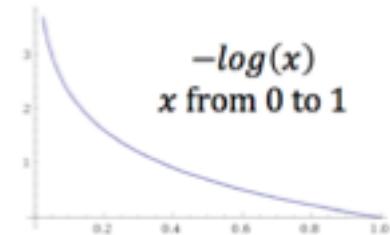
$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} = x + b$$

exp

$$\begin{bmatrix} 0.058 \\ 2.36 \\ 1.32 \end{bmatrix}$$

Normalize

$$\begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$



$$-\log(0.353)$$

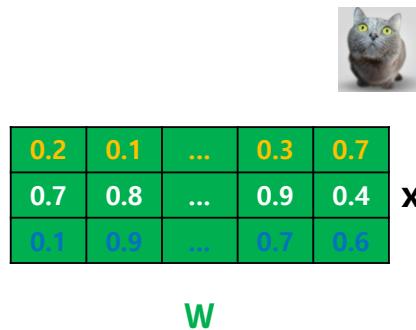
$$\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- 전부 양수
- 값의 차이 벌림

- 합이 1

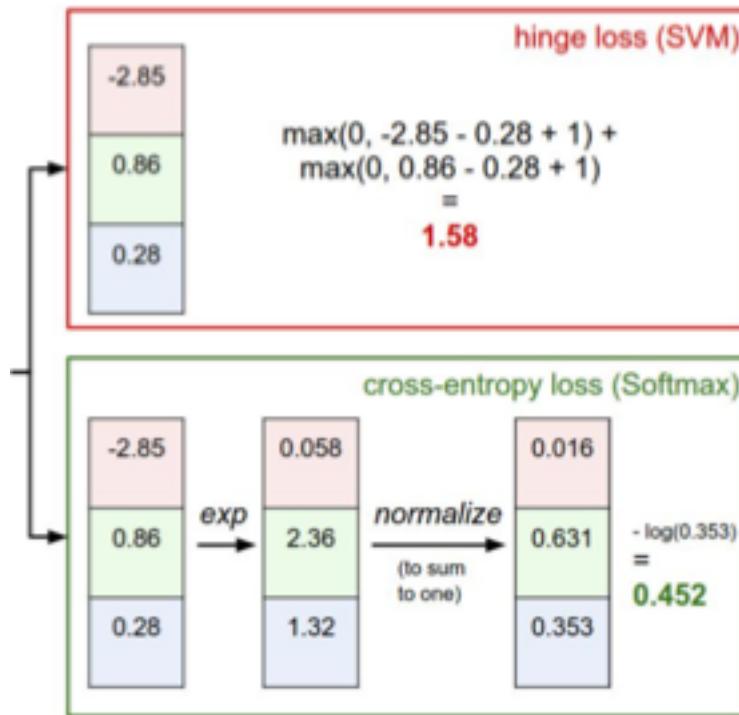
Softmax vs. SVM

- 데이터 셋 (x, y)
- Score Function
- Loss Function



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

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



- 데이터 셋 (x, y)
- Score Function
- Loss Function

$$s = f(x; W) \stackrel{\text{e.g.}}{=} Wx$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{sj}}\right)$$

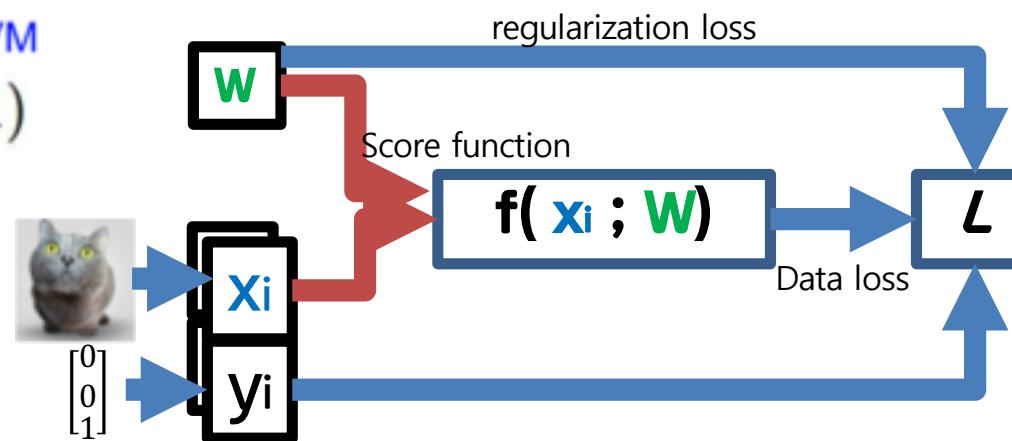
Softmax

SVM

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

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W)$$

Full loss



돌아보기 ...

- 분류기의 구성
 - Score function
 - Loss function
 - Optimization

$$\begin{matrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \end{matrix} \times \begin{matrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{matrix} + \begin{matrix} 0.1 \\ 0.2 \end{matrix} = \text{Output}$$

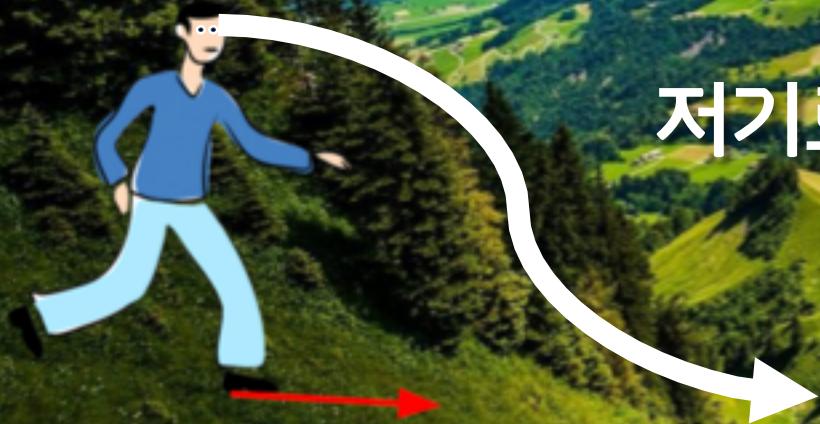
오늘의 주제

Loss를 최소화하는 w와 b
를 찾아라

최저점을 향해 가시오

저기 보이네~~~!!!!

저기로 가자~~!

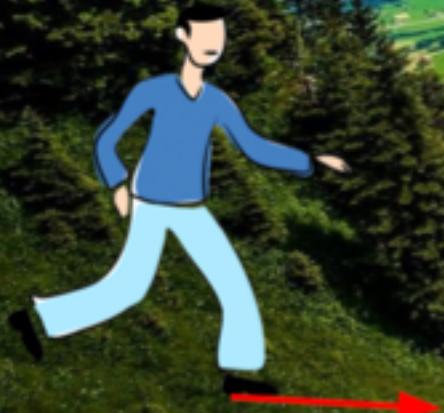




최저점을 향해 가시오

헉?!

눈이 없으면 ?



최저점을 향해 가시오

넘어질 것 같아..
어디까지 갈 수 있을까..



[대안]

발로 더듬더듬 해서 내리막이면 가자!

내리막을 찾는 방법

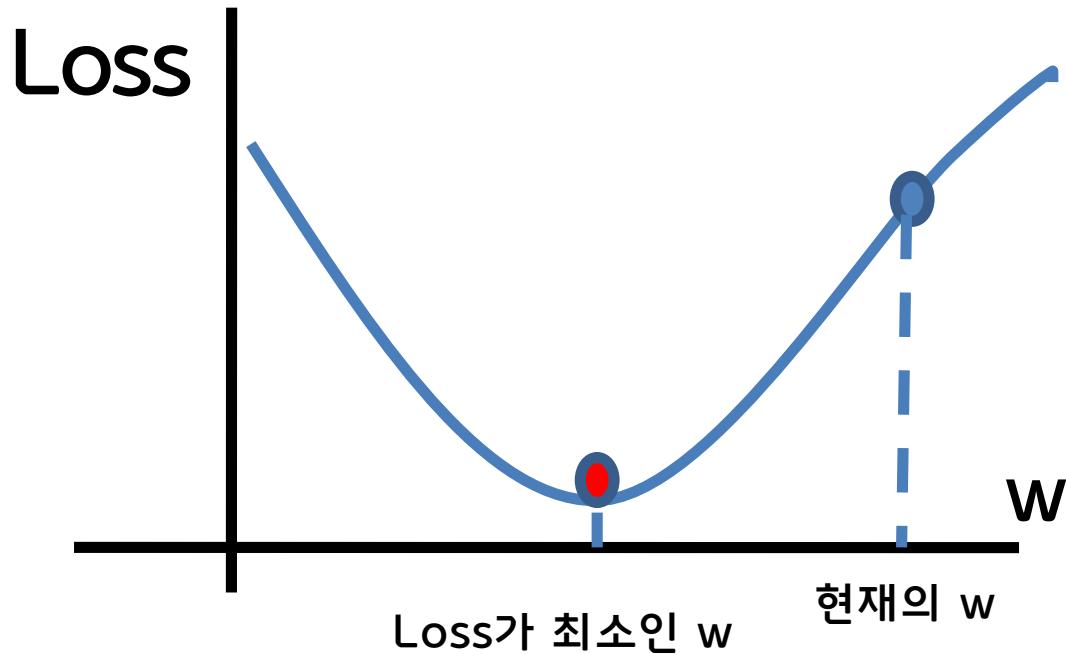


내리막 ?? == 기울기 ??

기울기를 따라 내려 가보자

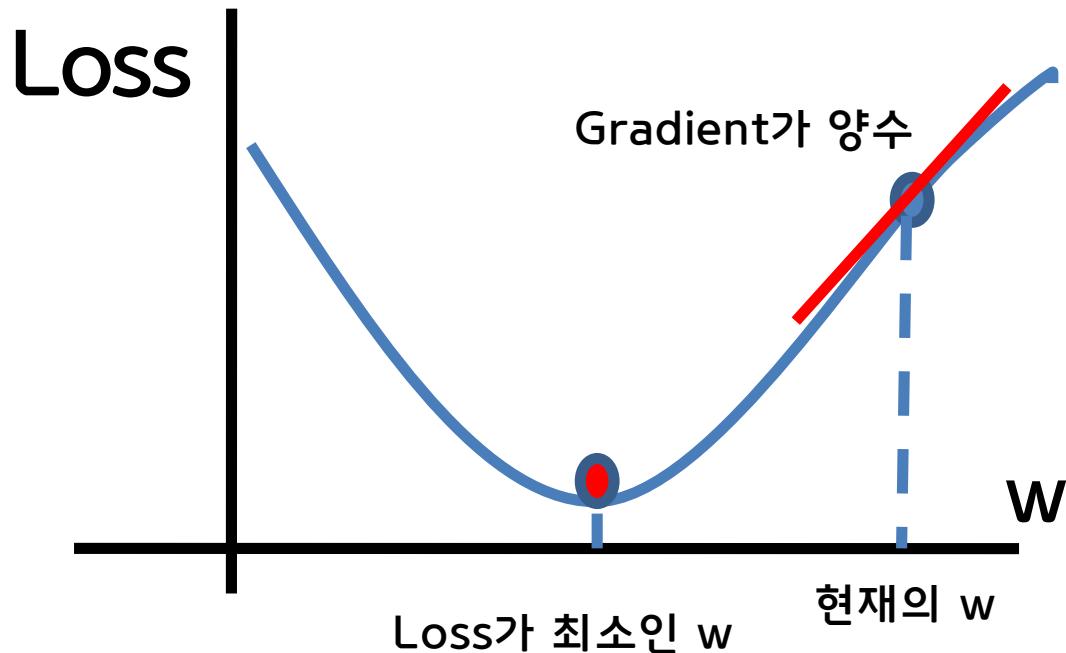
Gradient Descent

Gradient Descent



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

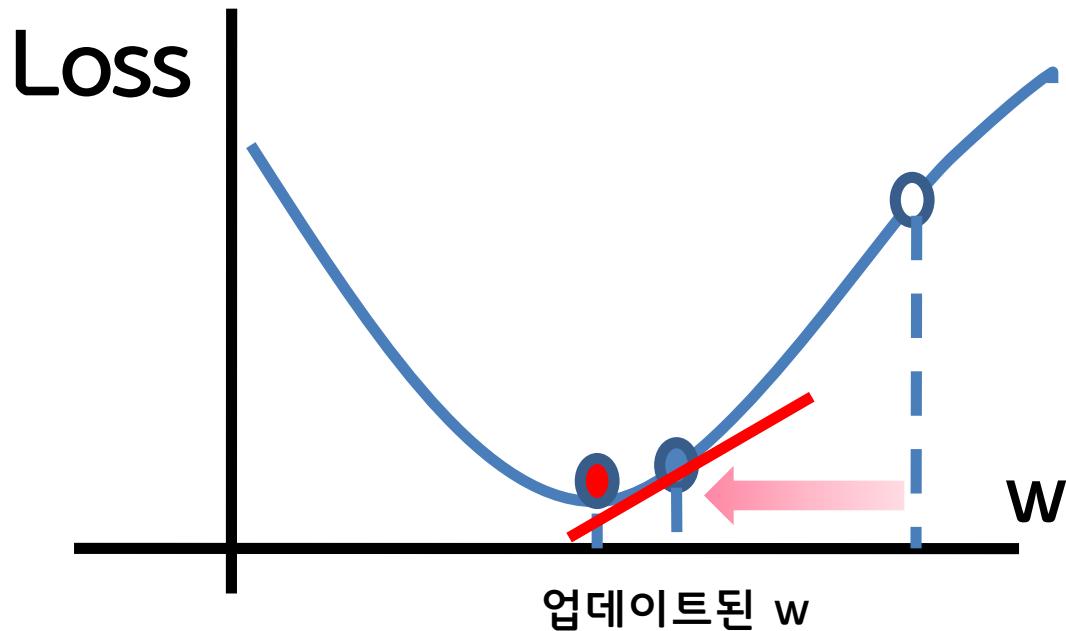
Gradient Descent



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

η : learning rate

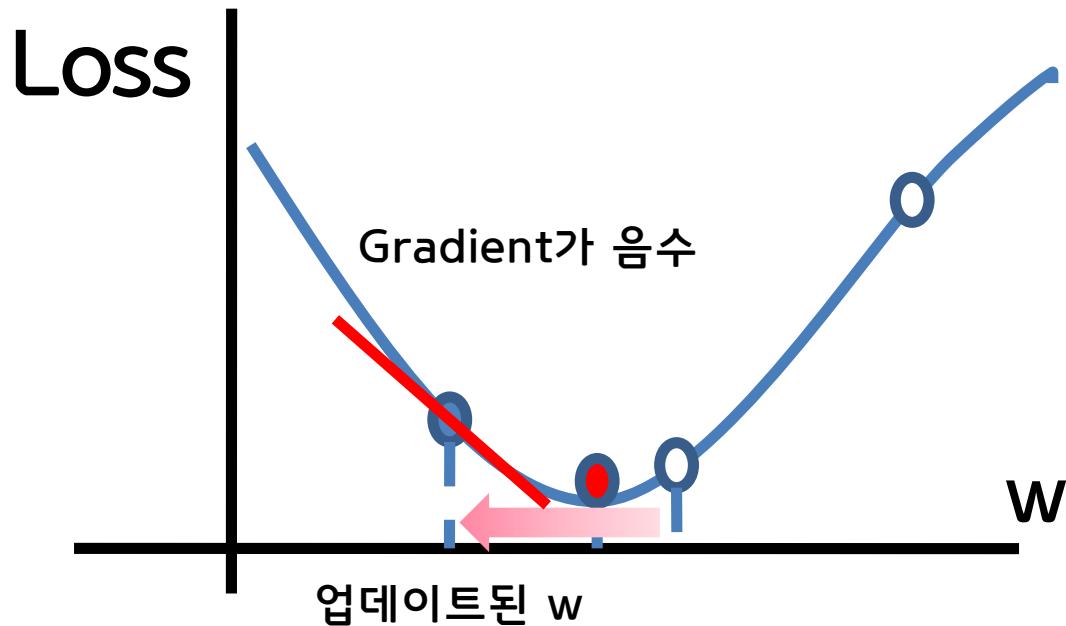
Gradient Descent



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

η : learning rate

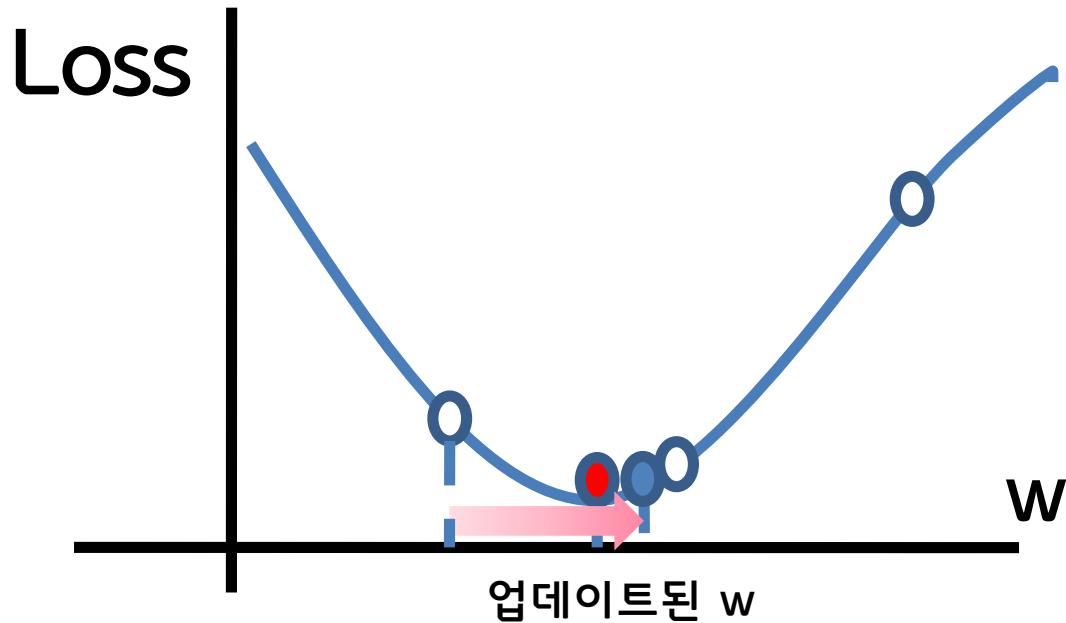
Gradient Descent



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

η : learning rate

Gradient Descent



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

Loss 함수에 대한 w 의 음의 Gradient 찾아서 연속적으로 업데이트해 주면 되는군요

그런데 어떻게 Gradient를 찾죠?

Gradient Descent

Strategy #2: Follow the slope

In 1-dimension, the derivative of a function:

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

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

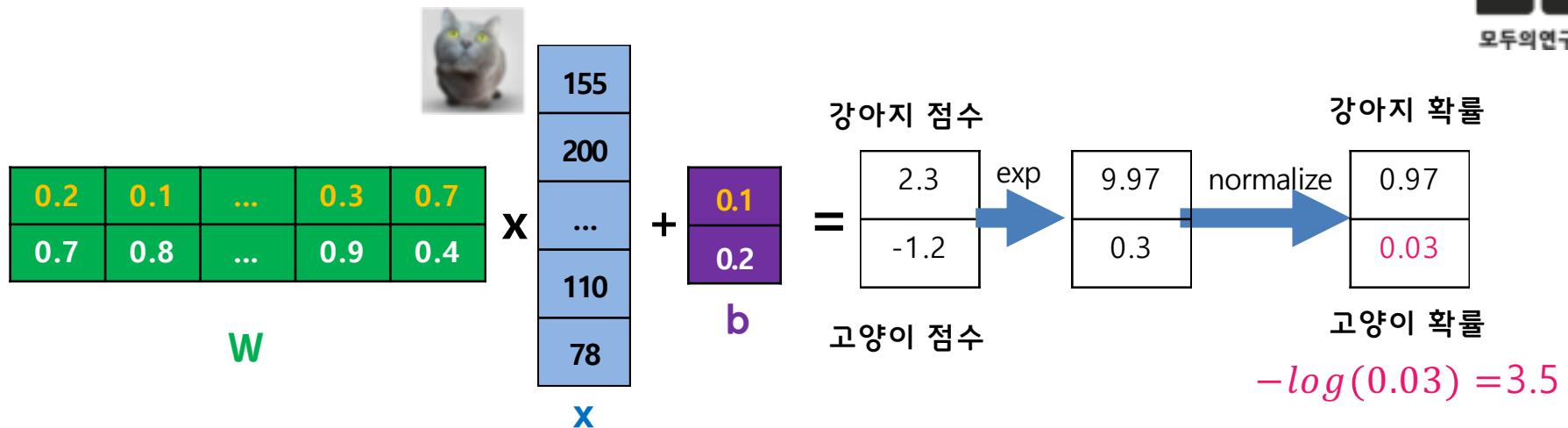
current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

gradient dW:

[?,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]



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**

Gradient Descent

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

W + h (first dim):

[0.34 + 0.0001,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25322

gradient dW:

[-2.5,
?,
?,
?,
?,
?,
?,
?,
?,
?,...]

$$\frac{(1.25322 - 1.25347)}{0.0001} = -2.5$$

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

?,
?,...]

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:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

W + h (second dim):

[0.34,
-1.11 + 0.0001,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25353

gradient dW:

[-2.5,
0.6,
?,
?,
?,
?,
?,
?,
?,
?,...]

$$\frac{(1.25353 - 1.25347)}{0.0001} = 0.6$$

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

?,...]

Gradient Descent

current W:	W + h (third 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, 0.78 + 0.0001, 0.12, 0.55, 2.81, -3.1, -1.5, 0.33,...]	[-2.5, 0.6, ?, ?, ?, ?, ?, ?, ?, ?,...]

loss 1.25347 loss 1.25347

Gradient Descent

current W:

[0.34,
-1.11,
0.78,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

W + h (third dim):

[0.34,
-1.11,
0.78 + 0.0001,
0.12,
0.55,
2.81,
-3.1,
-1.5,
0.33,...]

loss 1.25347

gradient dW:

[-2.5,
0.6,
0,
?,
?]

$$(1.25347 - 1.25347)/0.0001 = 0$$

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

f,...]

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:

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

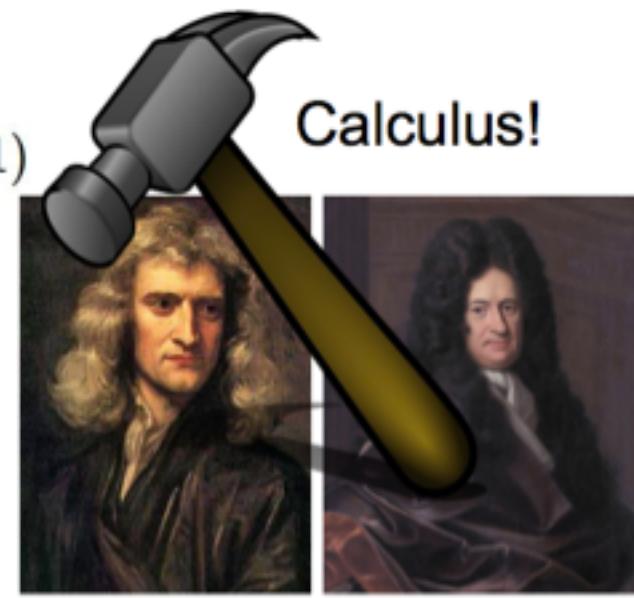
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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

want $\nabla_W L$

Use calculus to compute an
analytic gradient

Hammer image is in the public domain



This image is in the public domain

This image is in the public domain

수치 미분은 구현은 쉽지만 시간이 오래걸립니다

또한 해석적 방법에 비해 부정확합니다



오차역전파법 (Backpropagation)

Computational Graph

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

$$\begin{array}{c} z = t^2 \\ z = (x + y)^2 \quad \rightarrow \\ t = x + y \end{array}$$

z의 x 에 대한
미분의 연쇄법
칙 표현

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} \quad \rightarrow \quad \frac{\partial z}{\partial x} = \cancel{\frac{\partial z}{\partial t}} \frac{\partial t}{\partial x}$$

Computational Graph



- 연쇄법칙이란?

$$z = (x + y)^2 \rightarrow z = t^2$$
$$t = x + y$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} \rightarrow \frac{\partial z}{\partial t} = 2t$$
$$\frac{\partial t}{\partial x} = 1$$

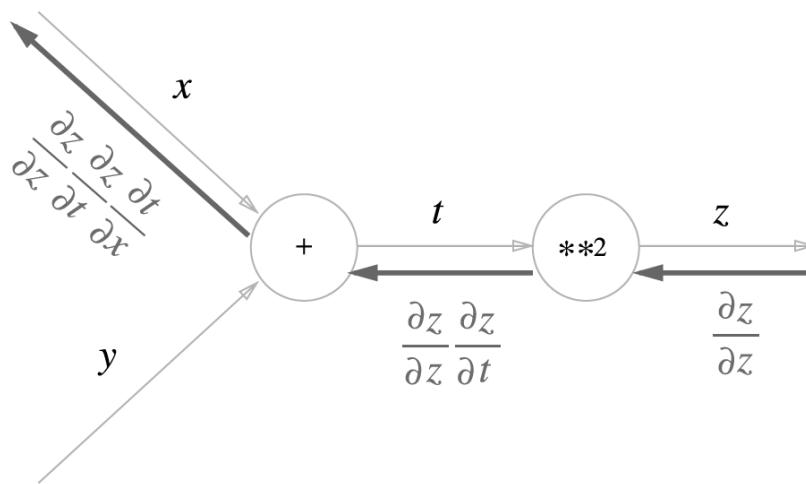
결과 $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = 2t \cdot 1 = 2(x + y)$

Computational Graph

- 연쇄법칙과 계산 그래프

역전파 과정

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

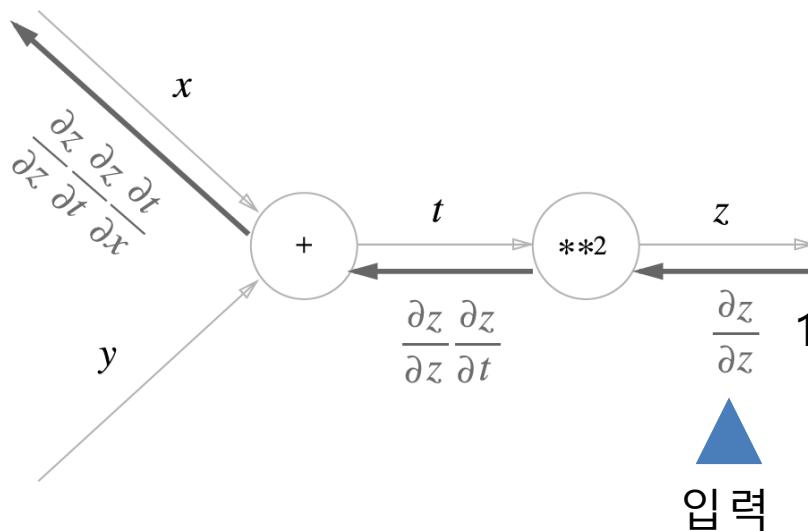


Computational Graph

- 연쇄법칙과 계산 그래프

역전파 과정

$$z = (x + y)^2 \rightarrow z = t^2$$
$$t = x + y$$

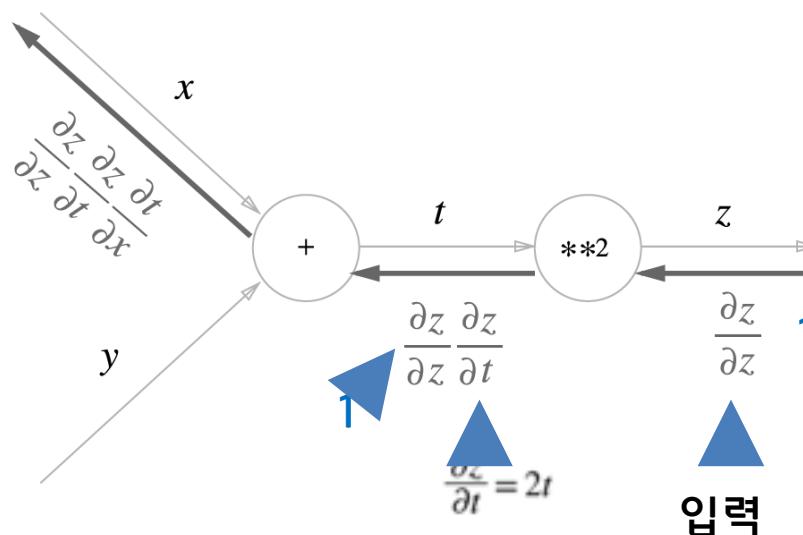


Computational Graph

- 연쇄법칙과 계산 그래프

역전파 과정

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

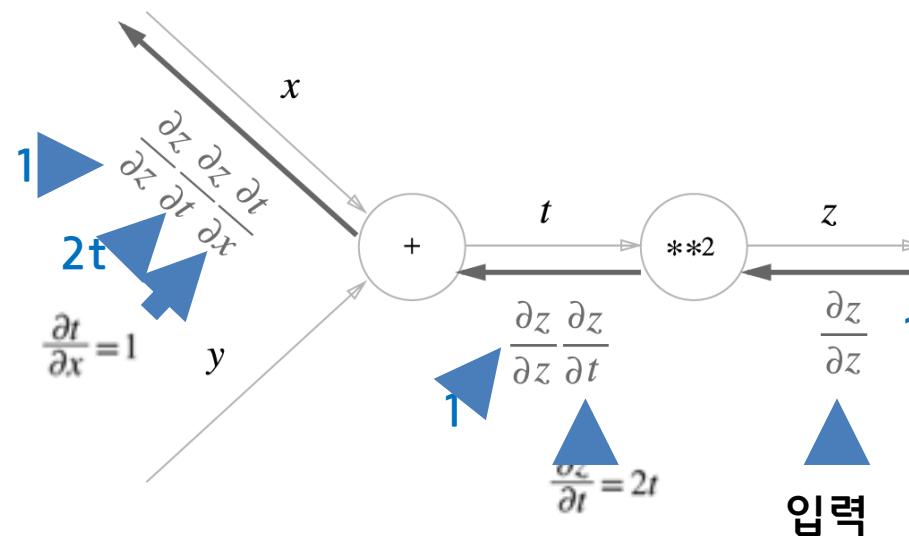


Computational Graph

- 연쇄법칙과 계산 그래프

역전파 과정

$$z = (x + y)^2 \Rightarrow z = t^2$$
$$t = x + y$$

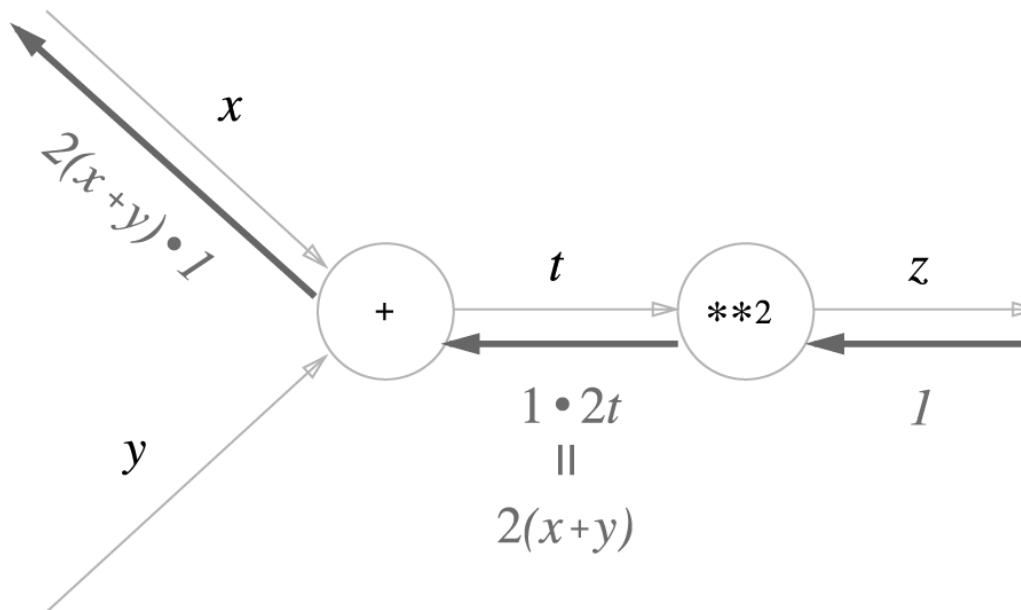


Computational Graph

- 연쇄법칙과 계산 그래프

역전파 과정

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



Computational Graph

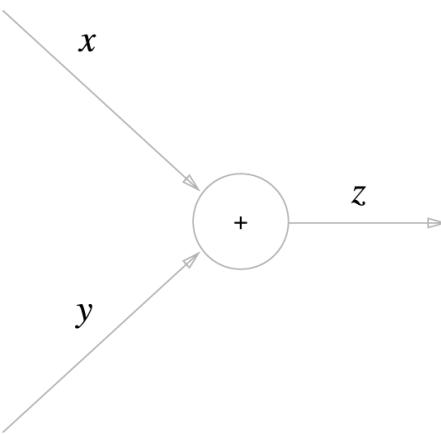
- 덧셈 노드의 역전파

$$z = x + y$$

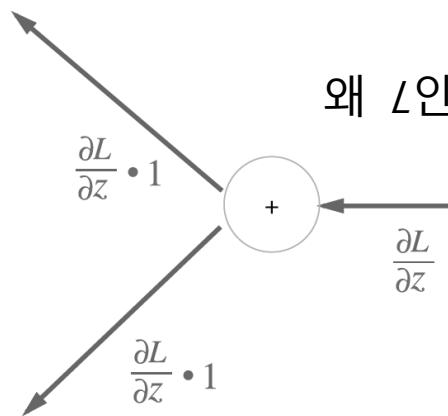


$$\frac{\partial z}{\partial x} = 1$$

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



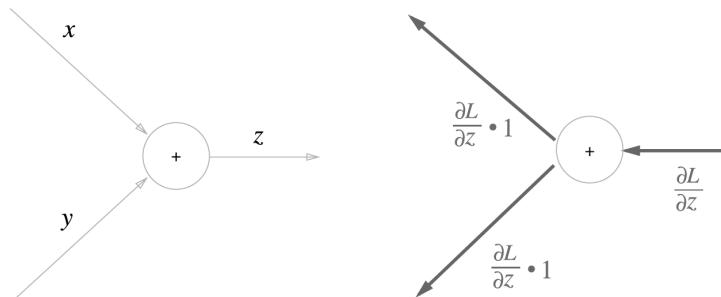
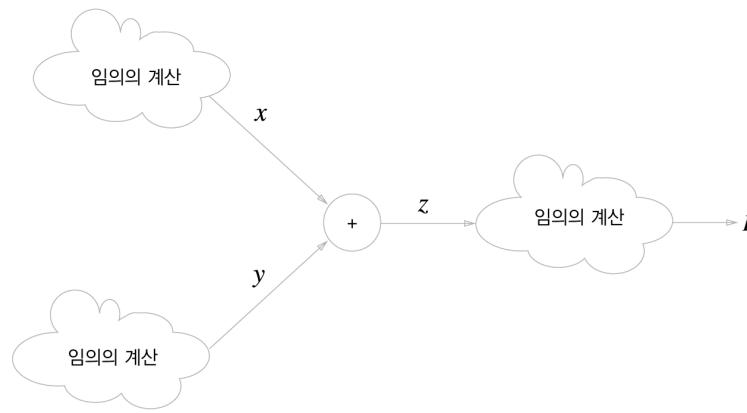
왜 L 인가요?



Computational Graph

- 덧셈 노드의 역전파

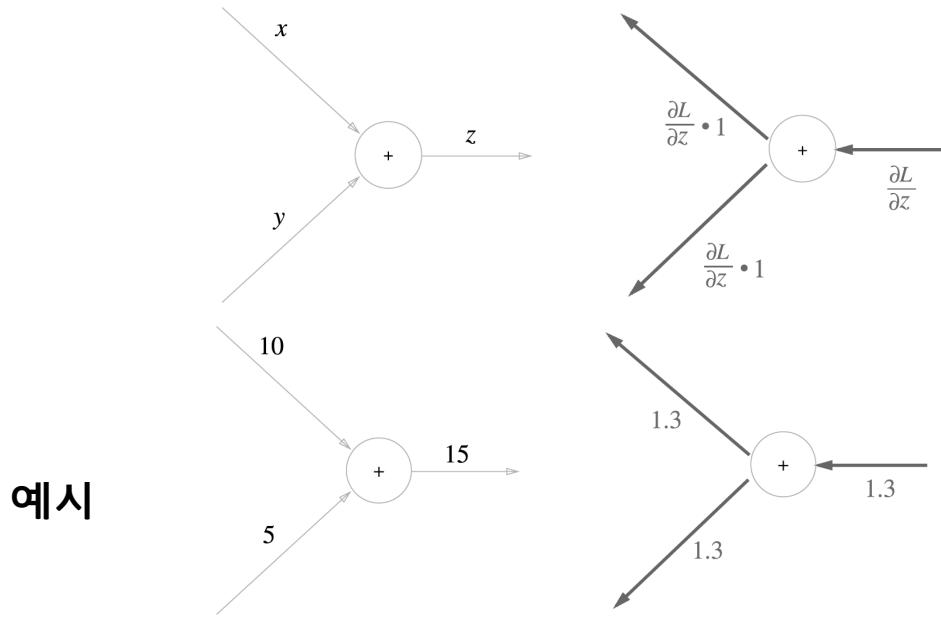
실제로 큰 계산을
가정하고 그중 한
노드가 덧셈노드임
을 가정한 겁니다



Computational Graph

- 덧셈 노드의 역전파

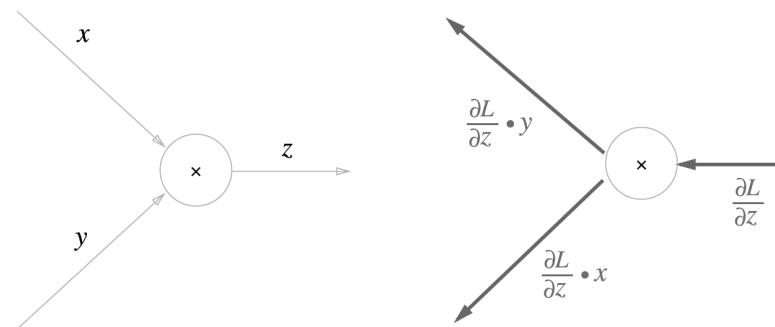
$$z = x + y \quad \Rightarrow \quad \begin{aligned} \frac{\partial z}{\partial x} &= 1 \\ \frac{\partial z}{\partial y} &= 1 \end{aligned}$$



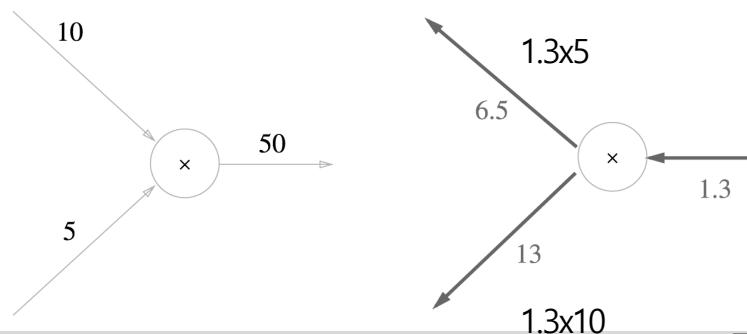
Computational Graph

- 곱셈 노드의 역전파

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



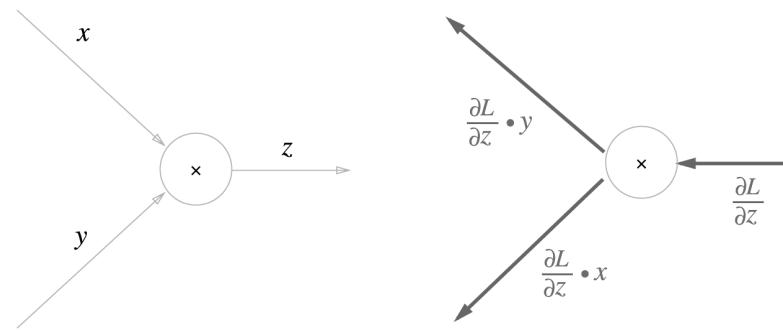
예시



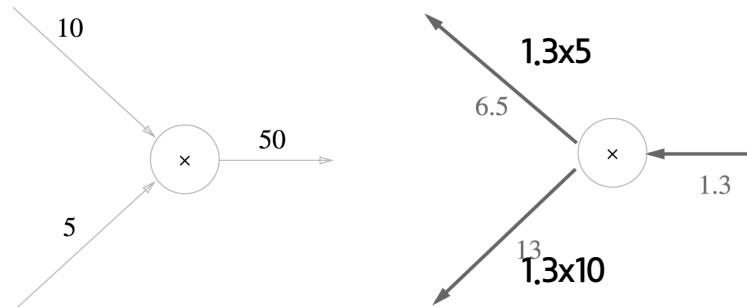
Computational Graph

- 곱셈 노드의 역전파

$$z = xy \quad \rightarrow \quad \begin{aligned} \frac{\partial z}{\partial x} &= y \\ \frac{\partial z}{\partial y} &= x \end{aligned}$$

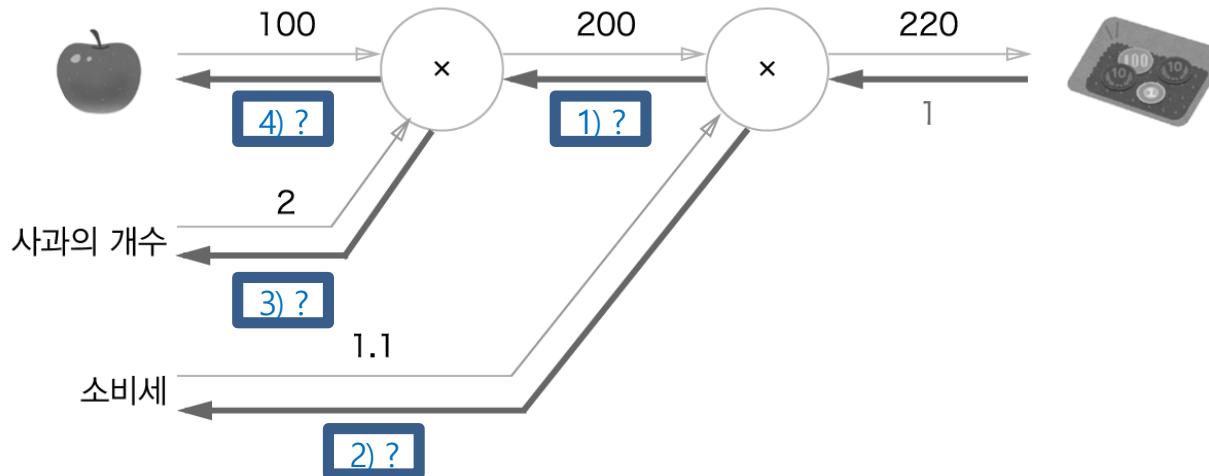


- 곱셈의 역전파는 순방향 입력신호의 값이 필요합니다
- 곱셈노드 구현 시 순전파 입력신호를 변수에 저장해 둡니다



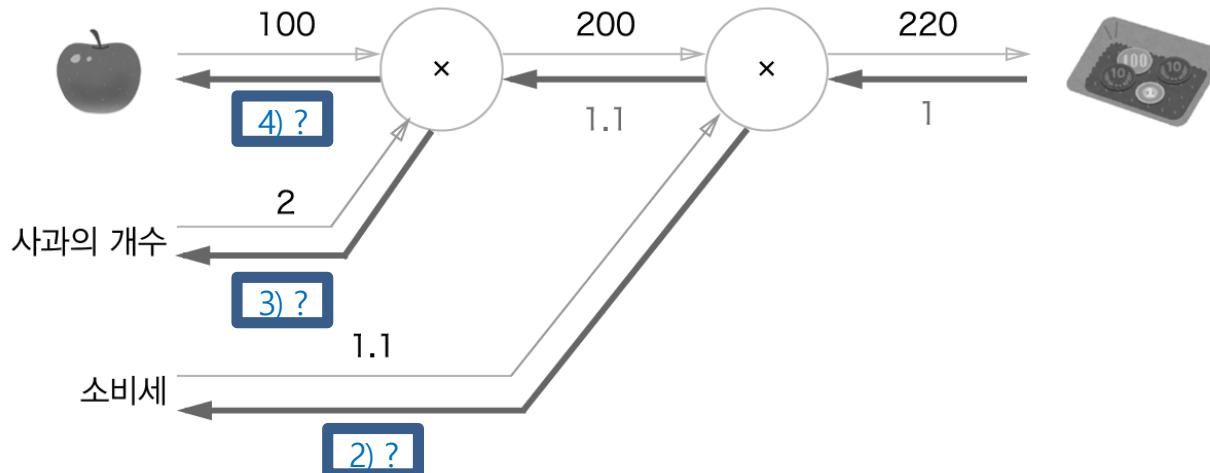
Computational Graph

- 사과쇼핑의 예



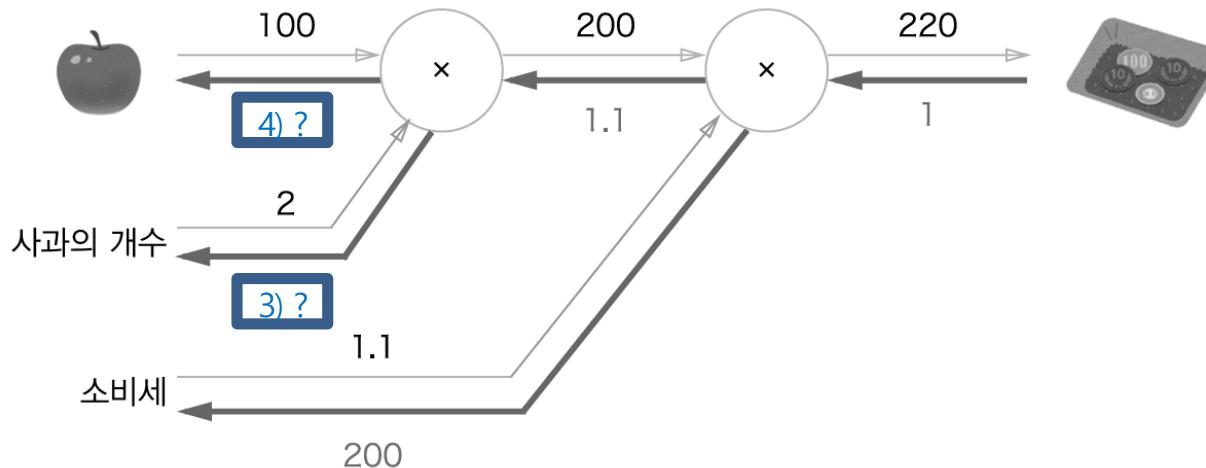
Computational Graph

- 사과쇼핑의 예



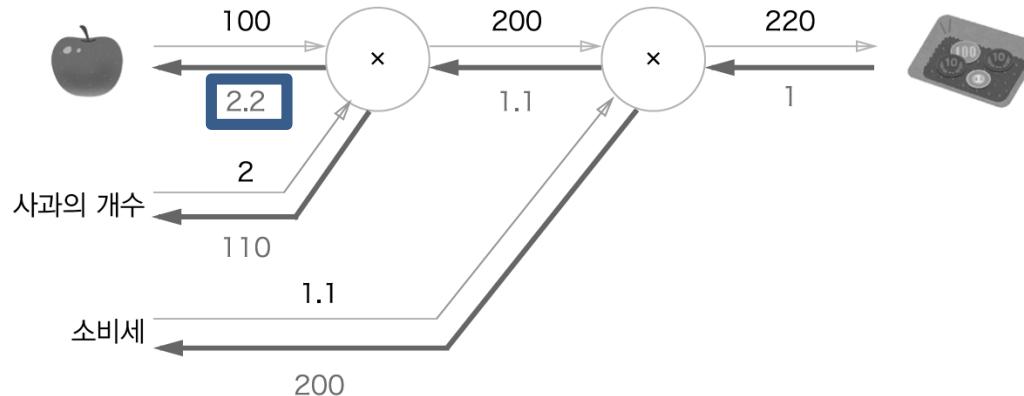
Computational Graph

- 사과쇼핑의 예



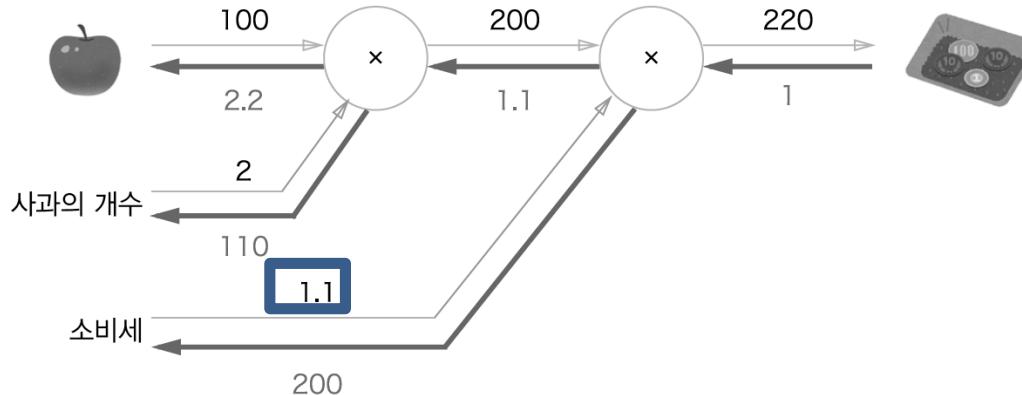
Computational Graph

- 사과쇼핑의 예



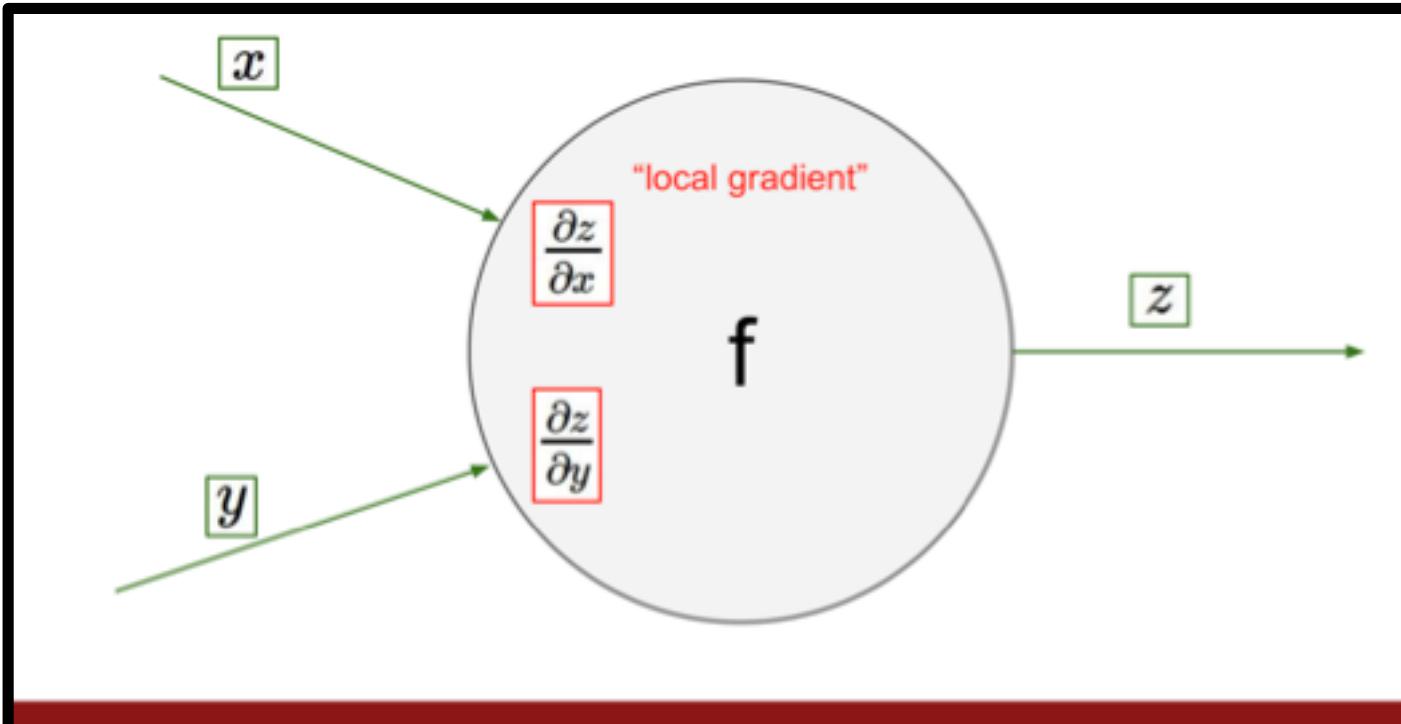
- 사과가격이 1원 오르면, 최종 가격 2.2 증가
 - $101(\text{사과 가격}) \times 2 \times 1.1 = 222.2 \text{ 원}$

Computational Graph

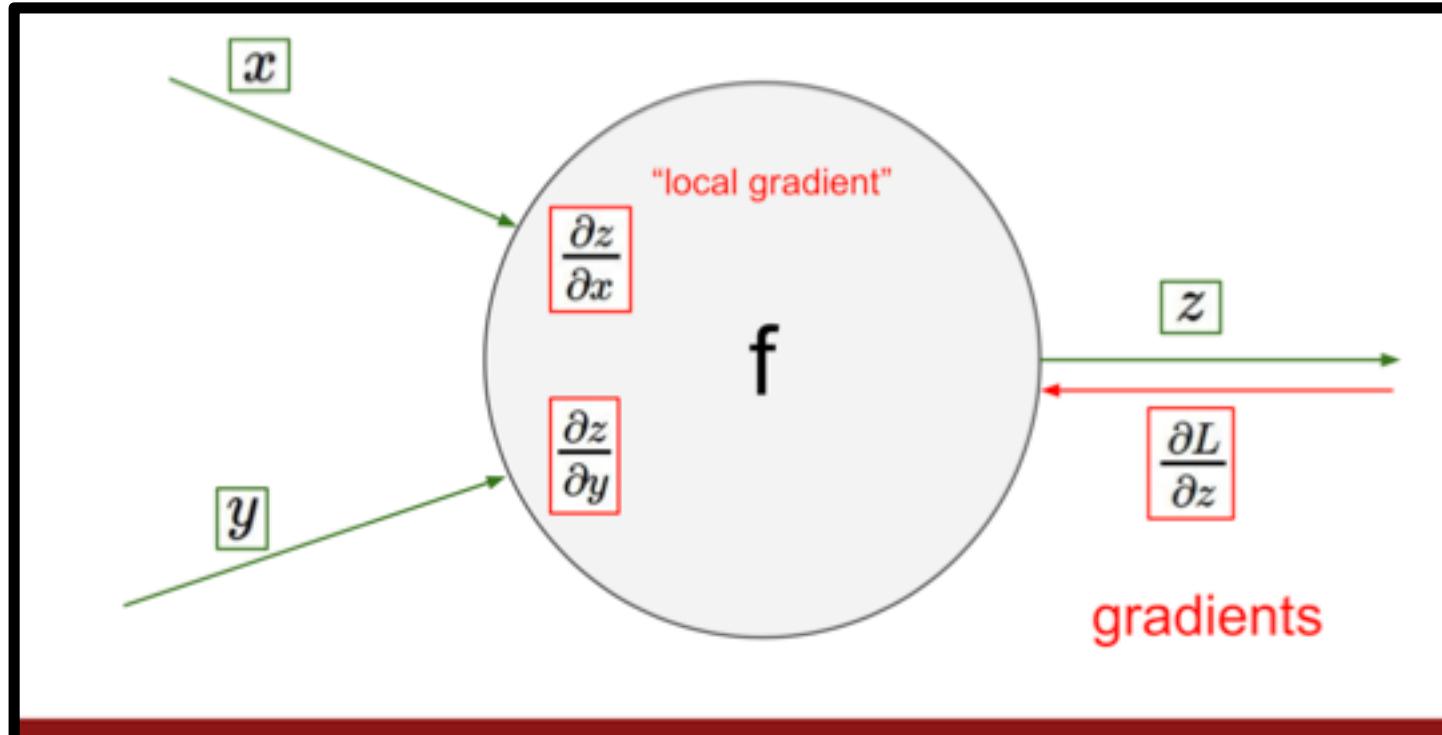


- 사과가격이 1원 오르면, 최종 가격 2.2 증가
 - $101(\text{사과 가격}) \times 2 \times 1.1 = 222.2$ 원
- 소비세가 100% ($1.1 + 1 = 2.1$) 오르면 가격 200증가
 - $100 \times 2 \times 2.1(\text{소비세}) = 420$ 원

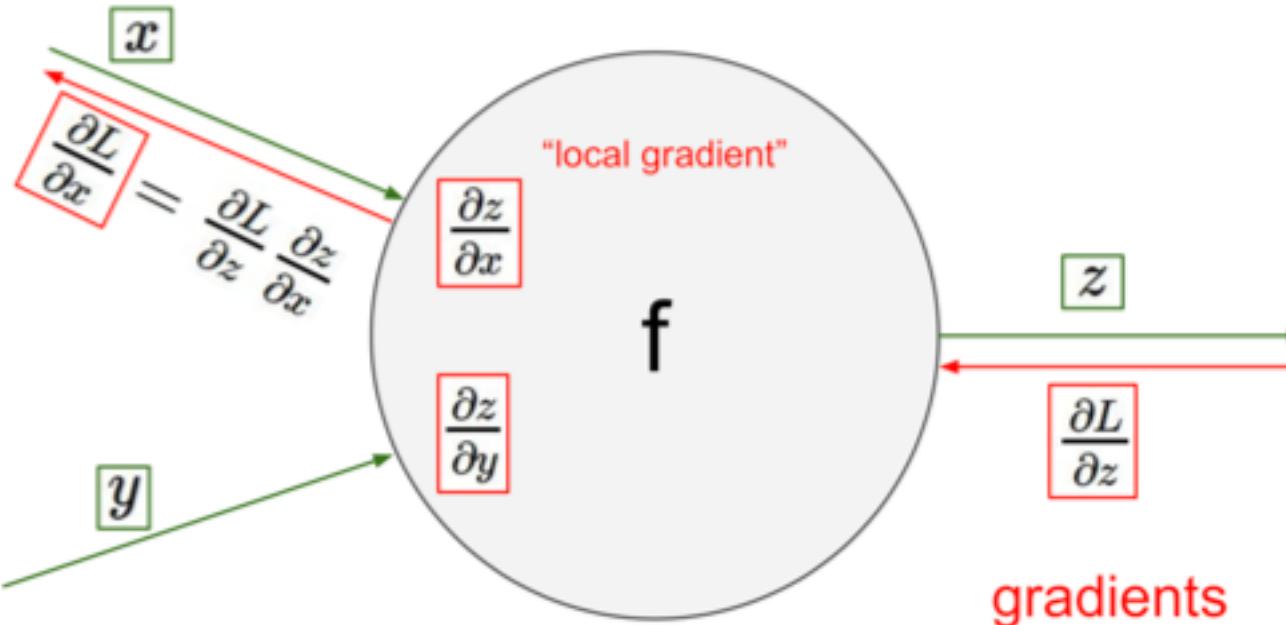
Computational Graph



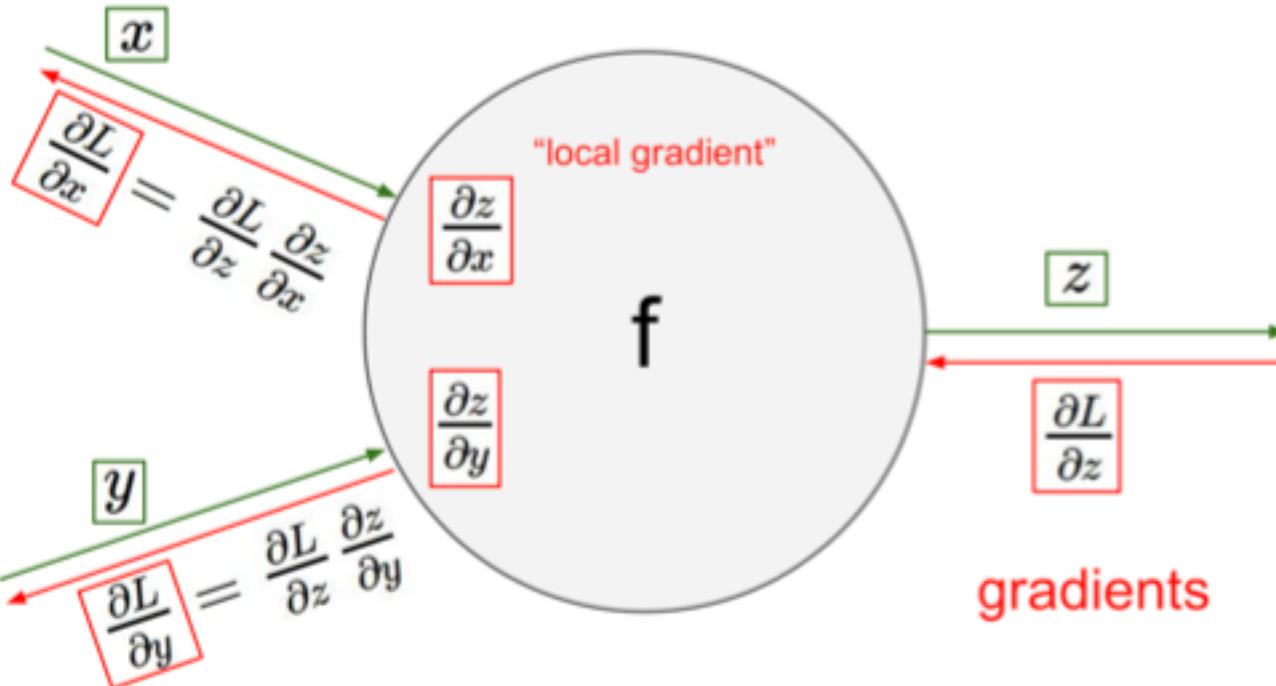
Computational Graph



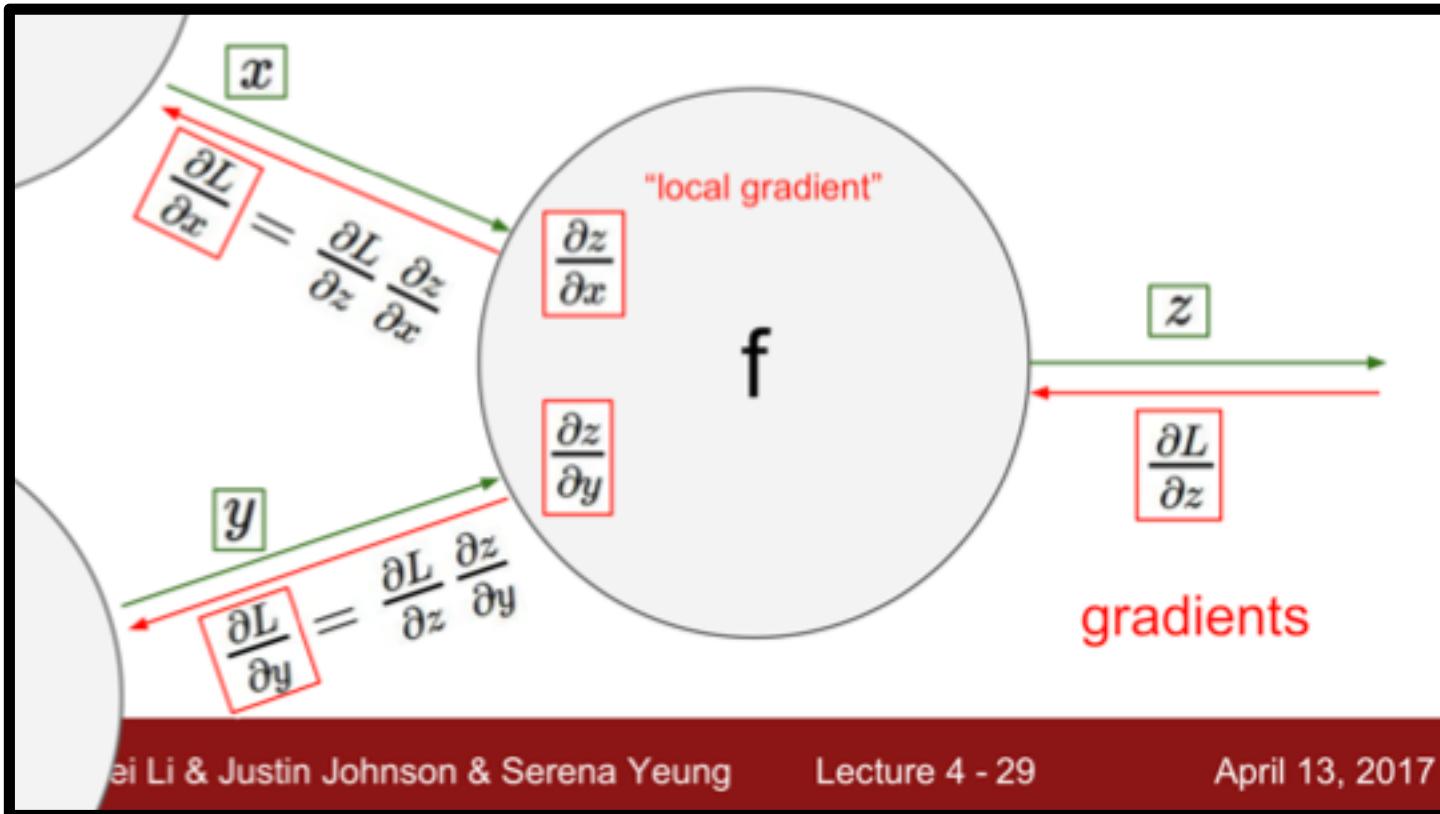
Computational Graph



Computational Graph



Computational Graph

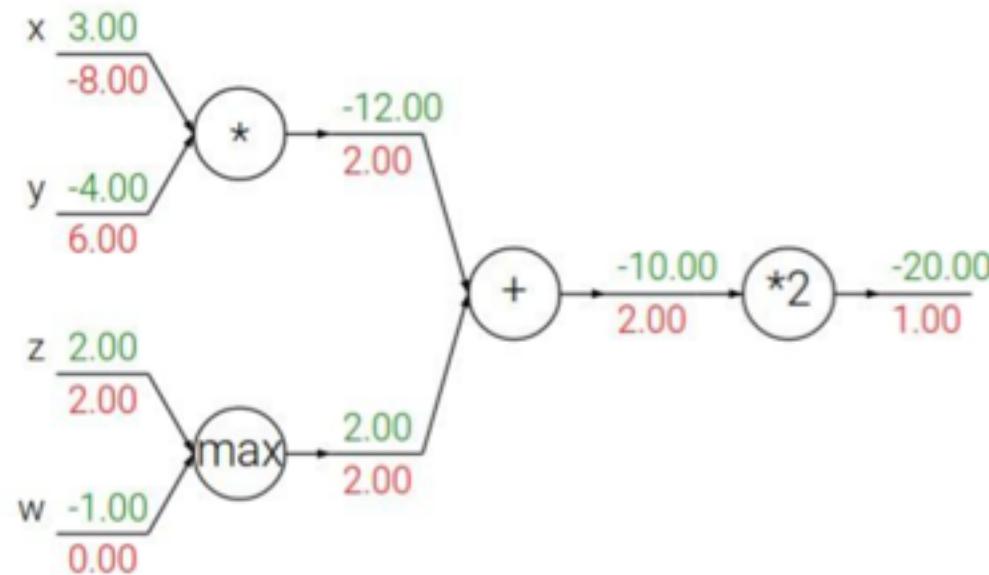


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher

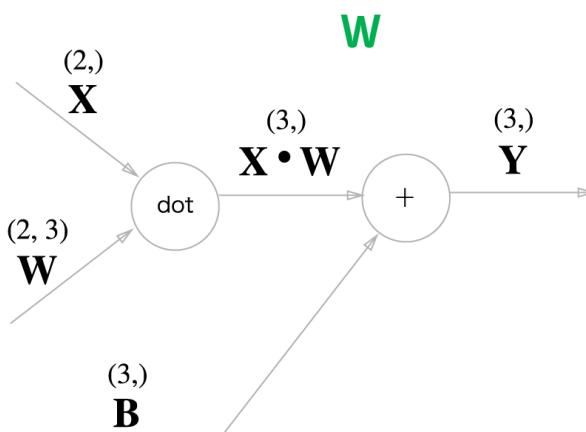


Computational Graph

강아지 파라미터

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

고양이 파라미터



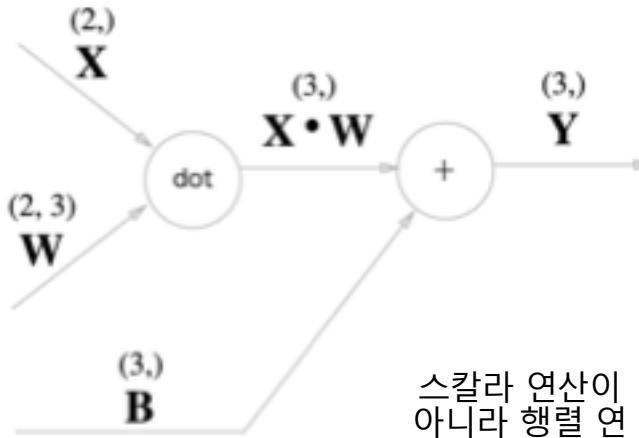
$$\begin{matrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{matrix} \times \begin{matrix} 0.1 \\ 0.2 \end{matrix} + \begin{matrix} 0.1 \\ 0.2 \end{matrix}$$

- Affine 계층

Affine / Softmax 계층 구현 하기

- 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
```



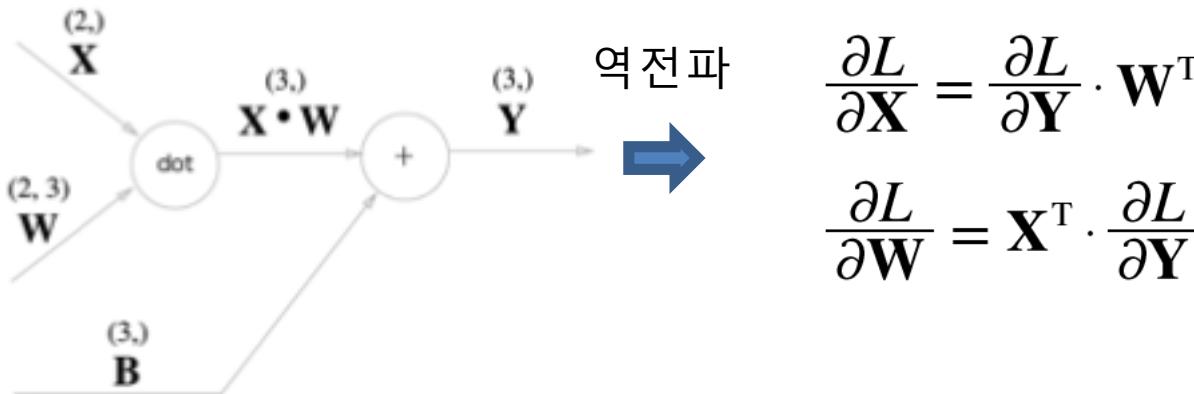
스칼라 연산이
아니라 행렬 연
산이 됩니다

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

Affine / Softmax 계층 구현

하기

- Affine 계층



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

Affine / Softmax 계층 구현 하기



- Affine 계층

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

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

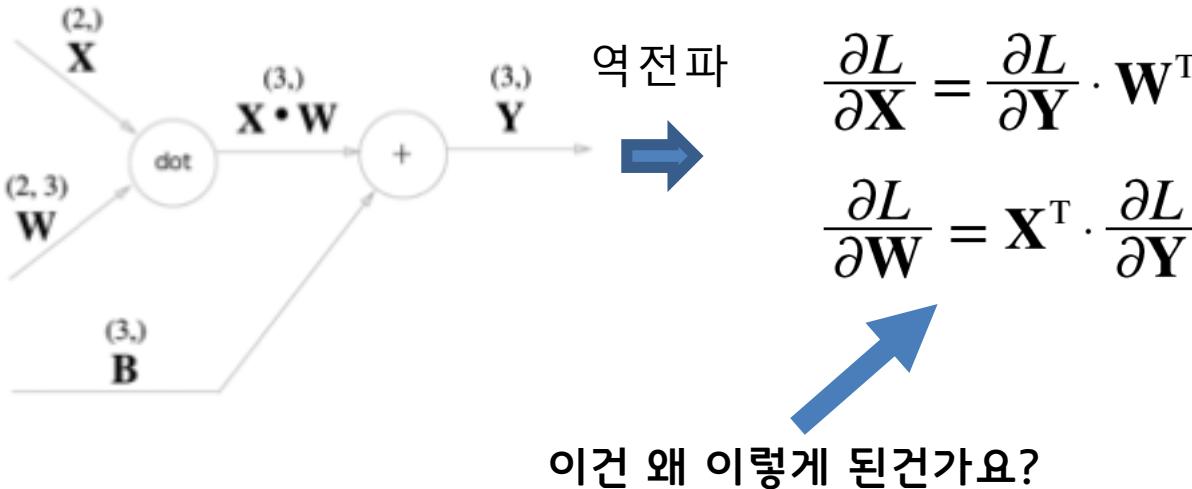
$$\mathbf{W}^T = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

Affine / Softmax 계층 구현



하기

- Affine 계층

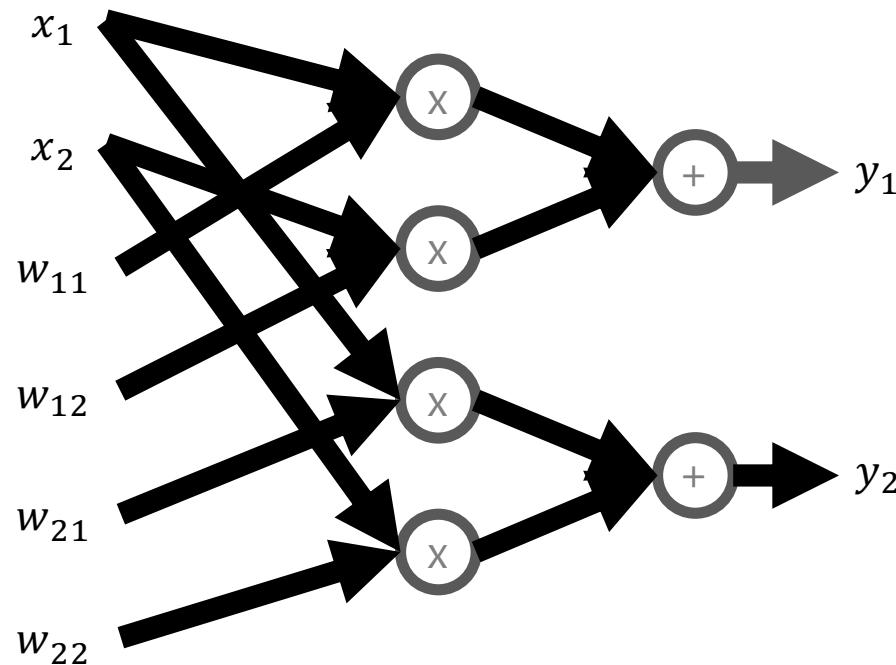


이건 왜 이렇게 된건가요?

Computational Graph

- Affine 계층

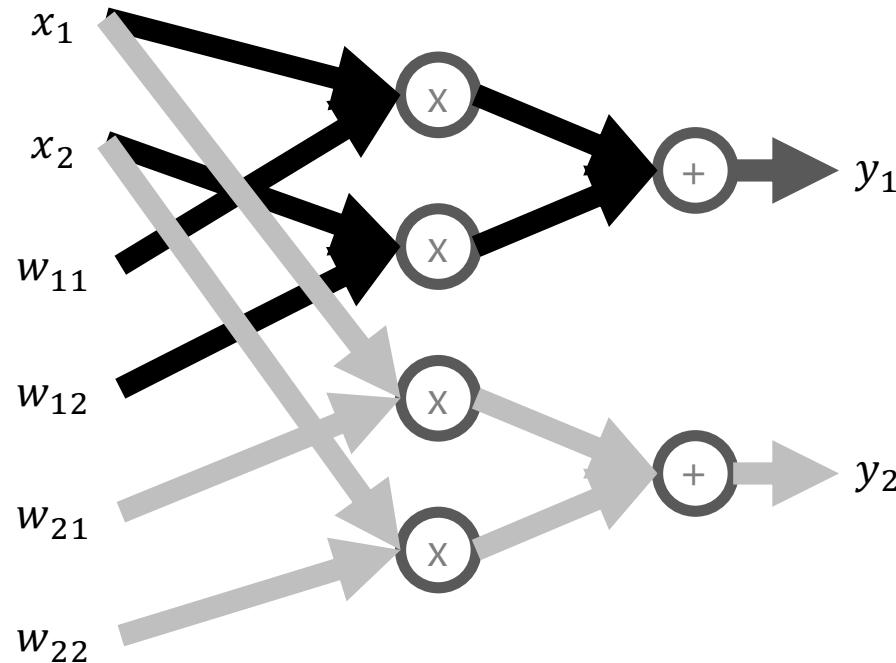
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

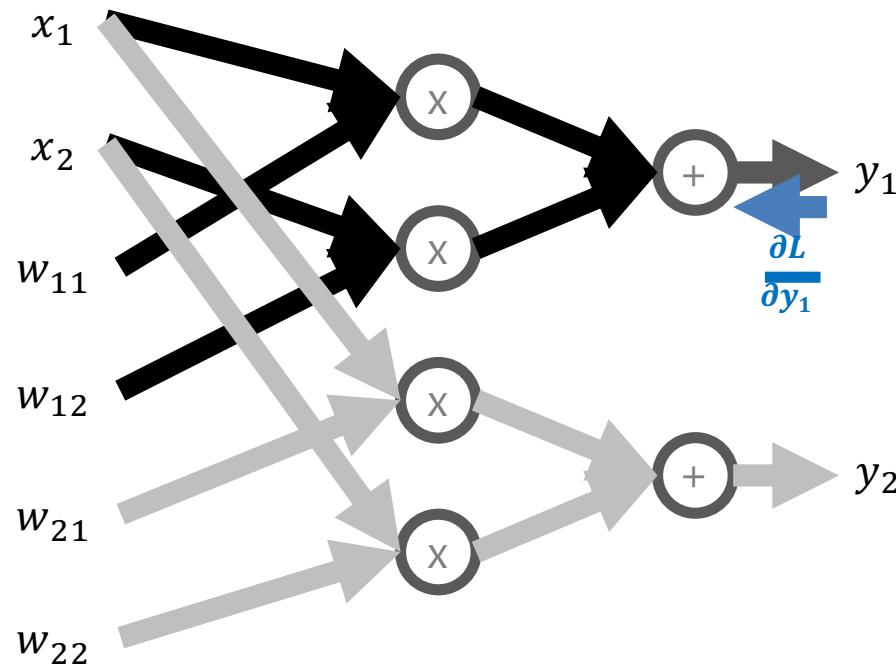
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_{11} \\ w_{12} \end{bmatrix} + \begin{bmatrix} w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$



Computational Graph

- Affine 계층

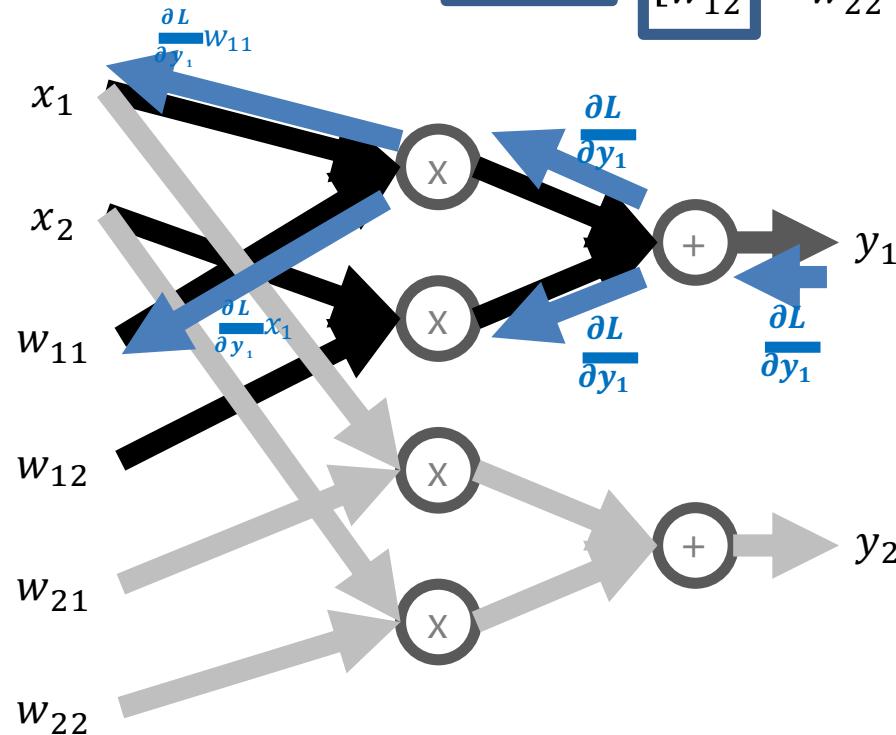
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$



Computational Graph

- Affine 계층

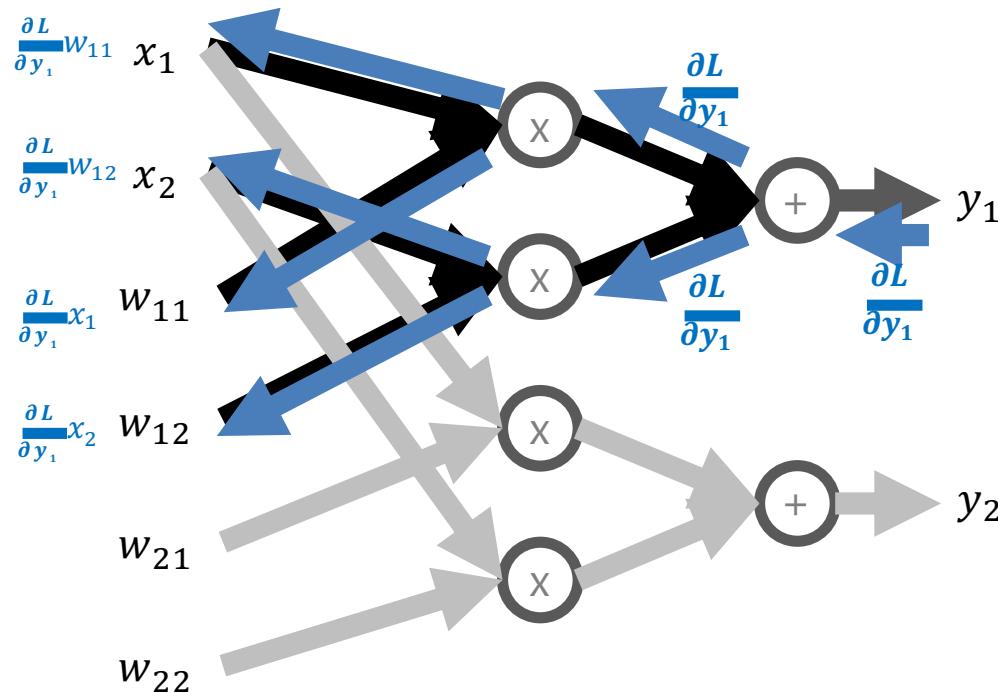
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$



Computational Graph

- Affine 계층

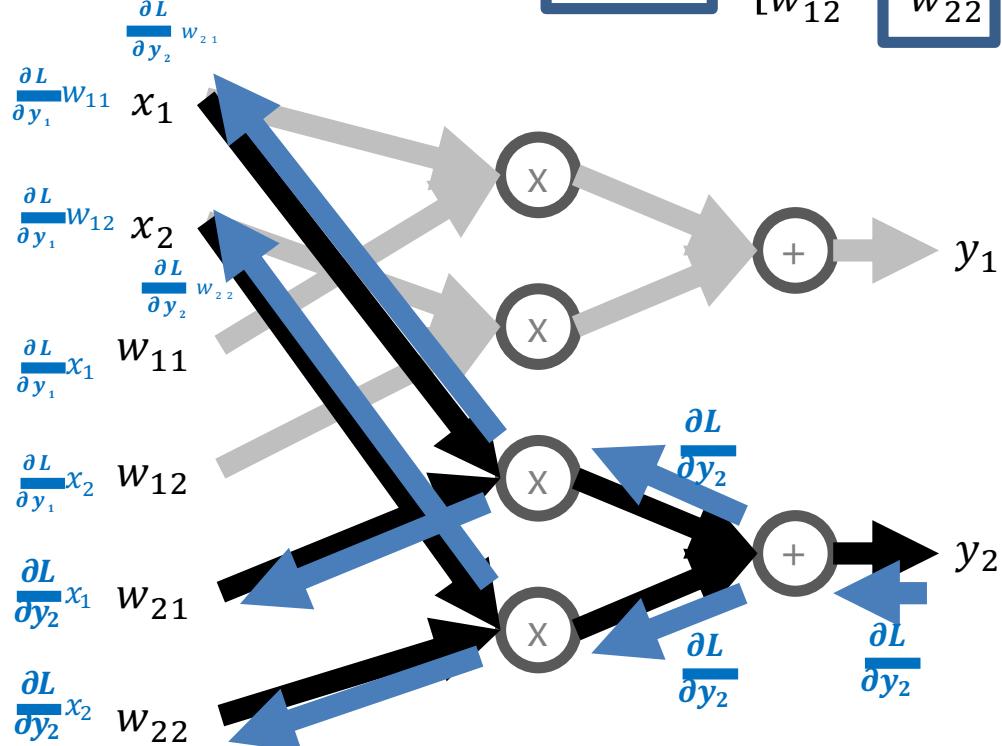
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$



Computational Graph

- Affine 계층

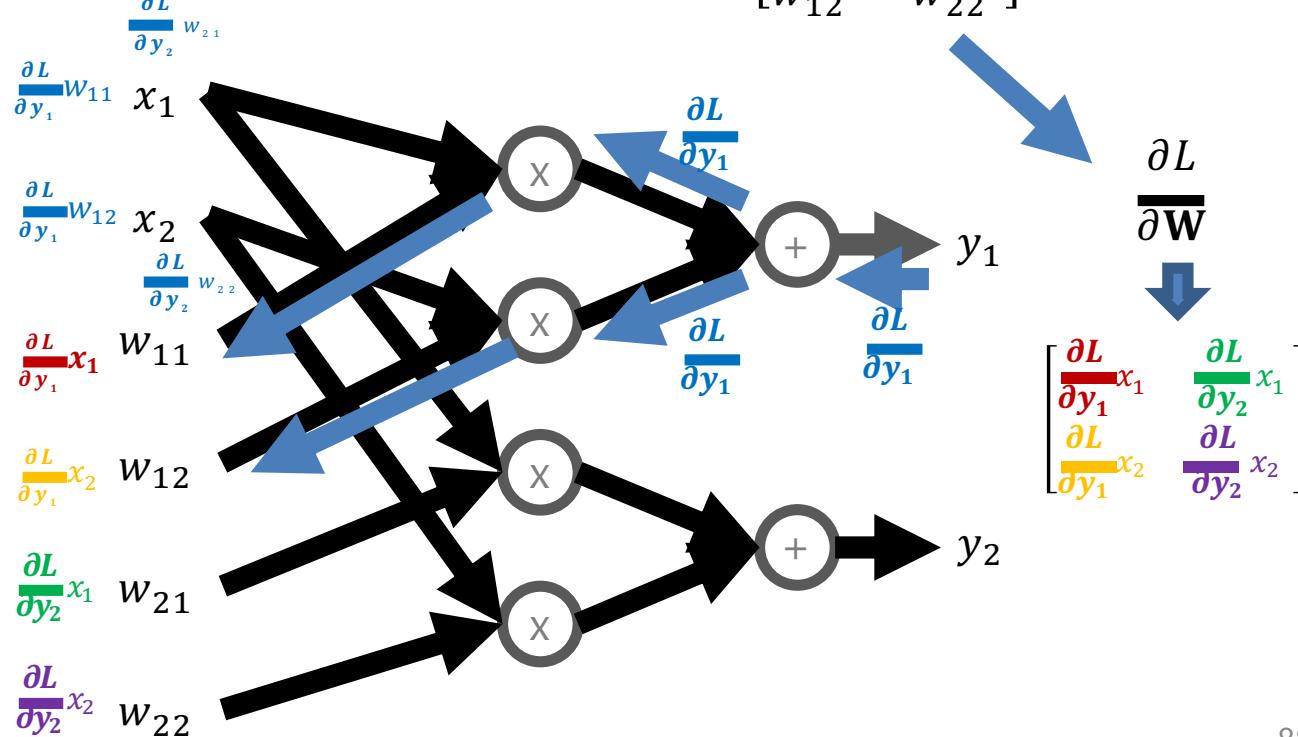
$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = \begin{bmatrix} y_1 & y_2 \end{bmatrix}$$



Computational Graph

- Affine 계층

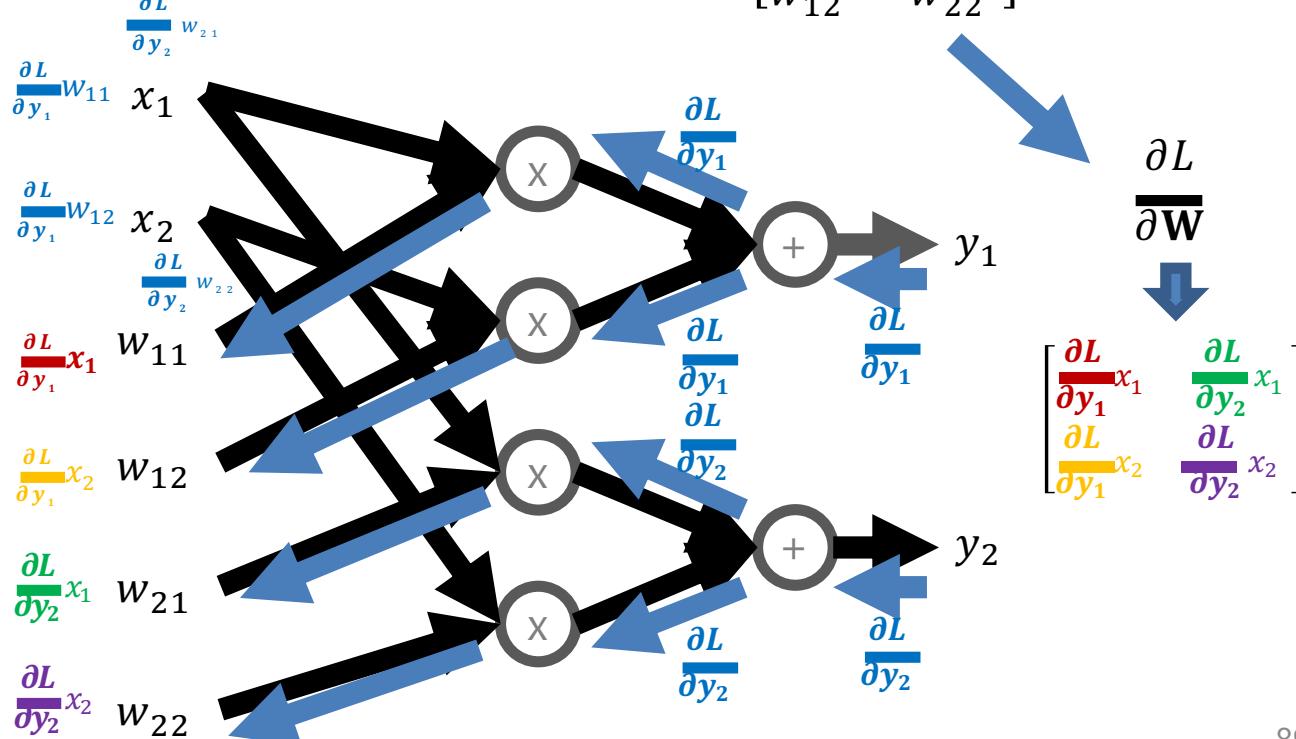
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

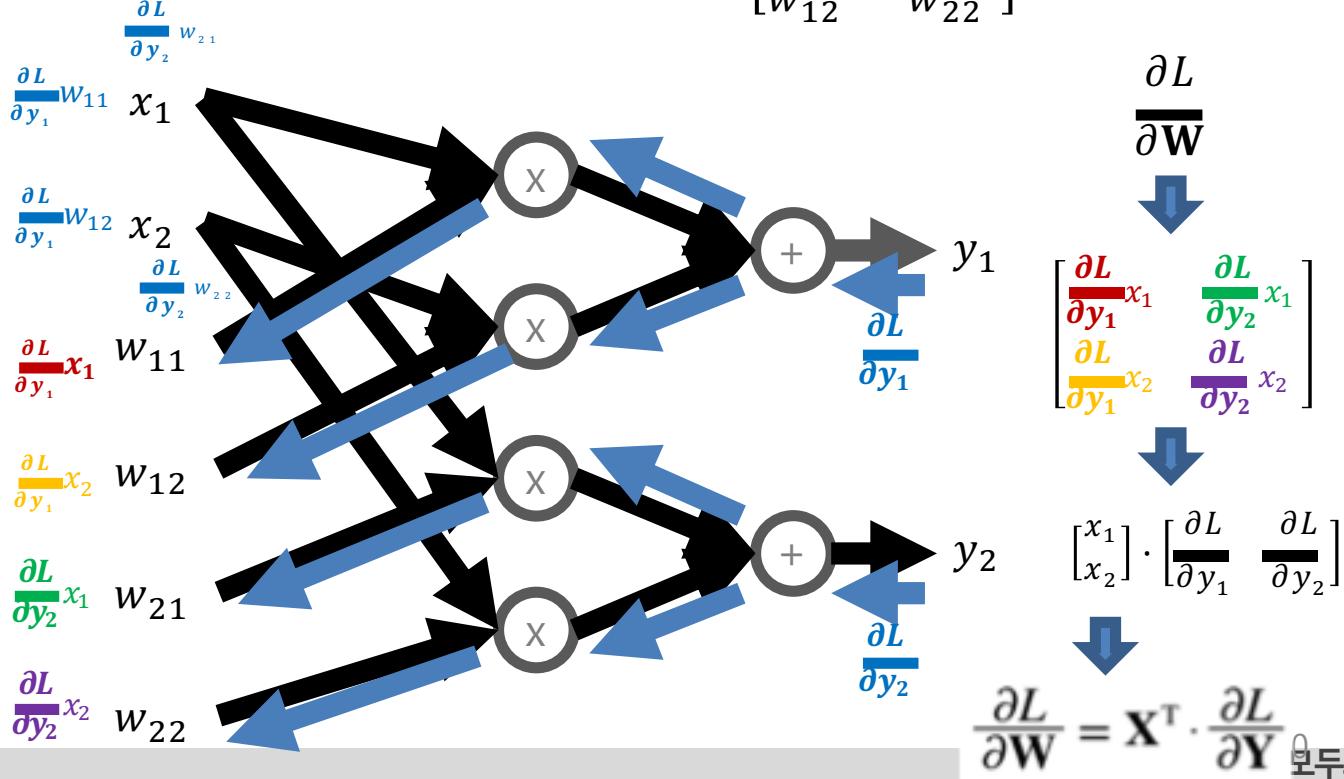
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

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

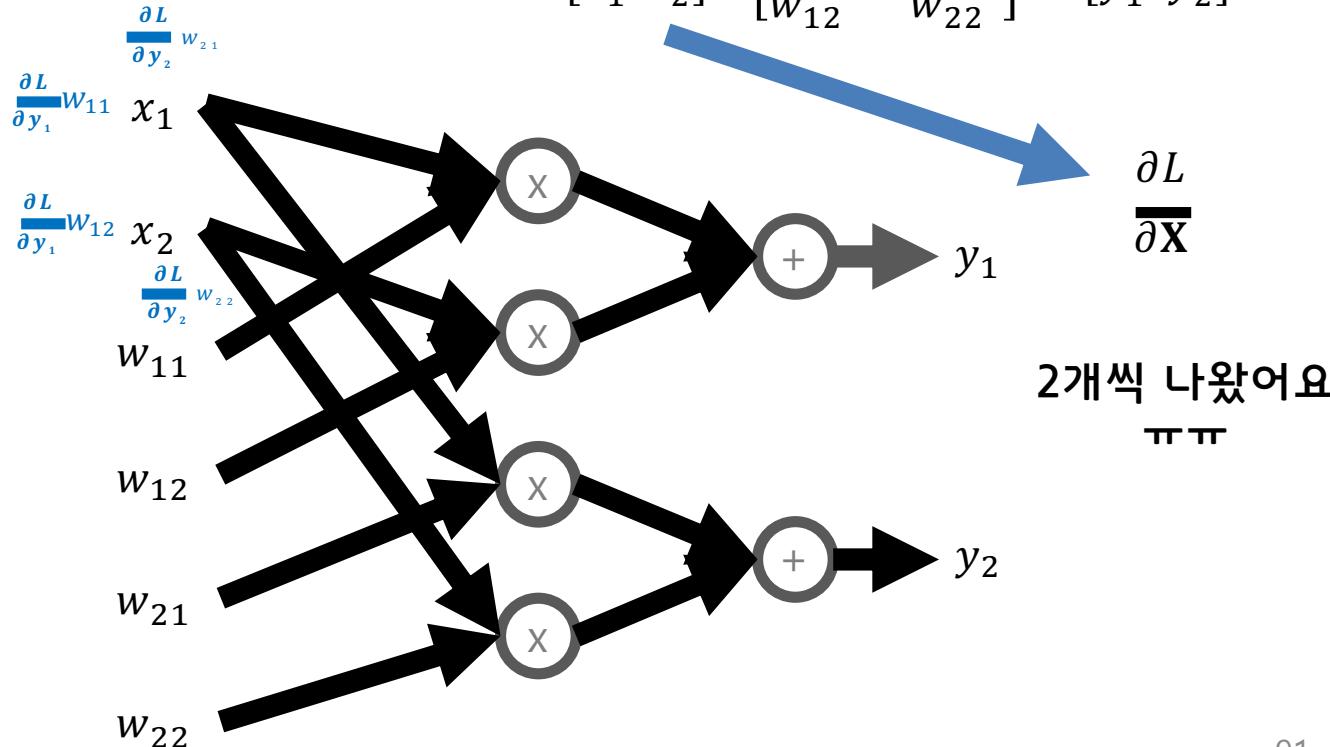


$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

Computational Graph

- Affine 계층

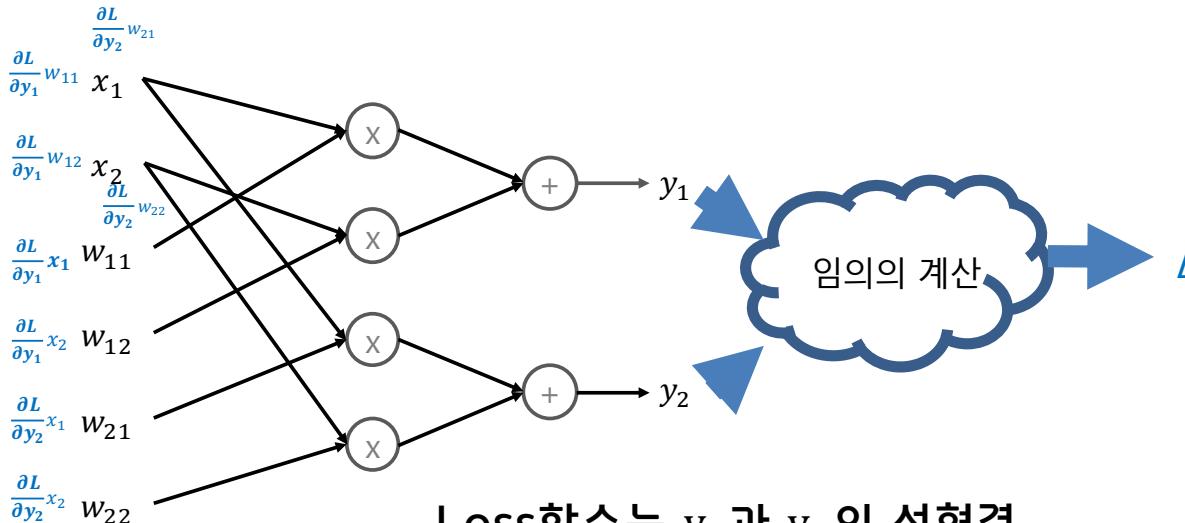
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

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



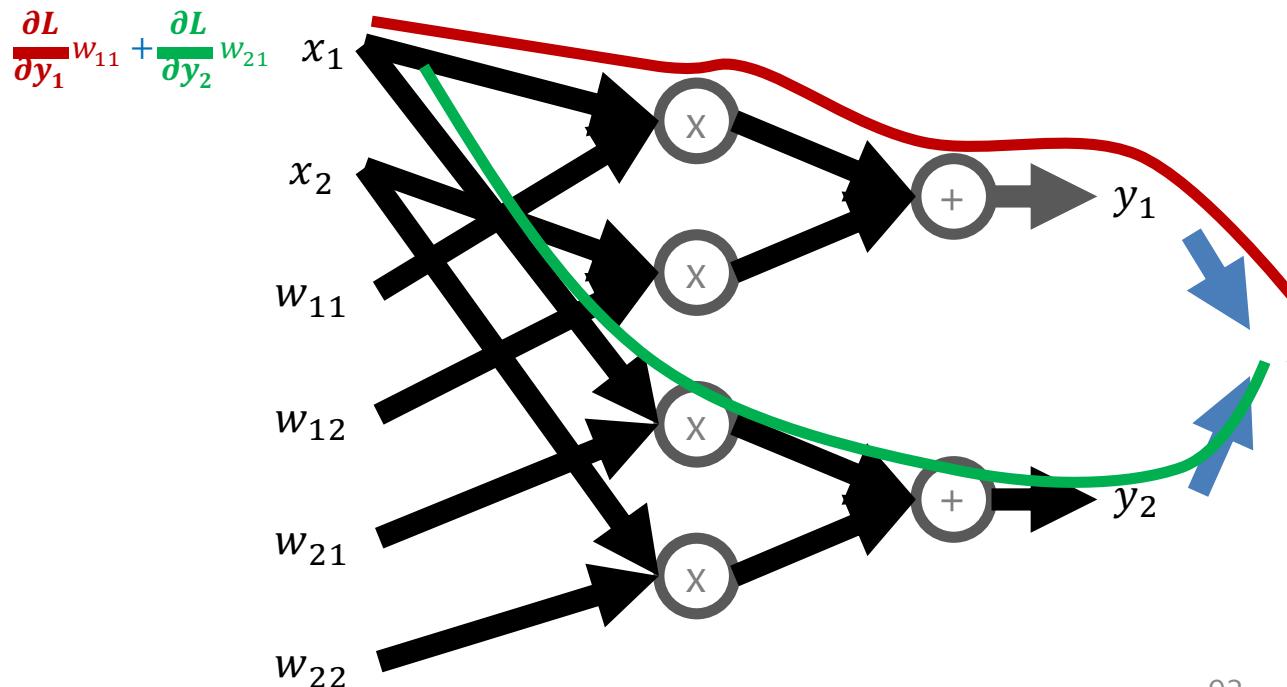
Loss함수는 y_1 과 y_2 의 선형결합일 겁니다

예) $L = ay_1 + by_2$

Computational Graph

- Affine 계층

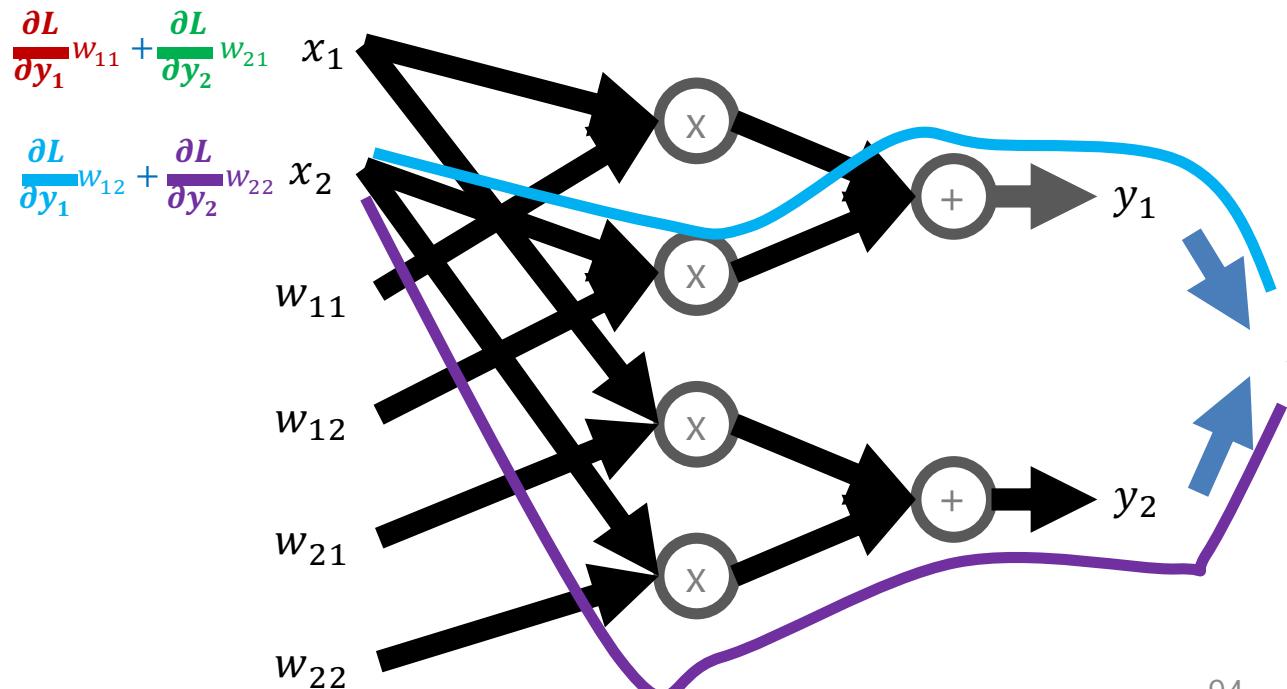
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



Computational Graph

- Affine 계층

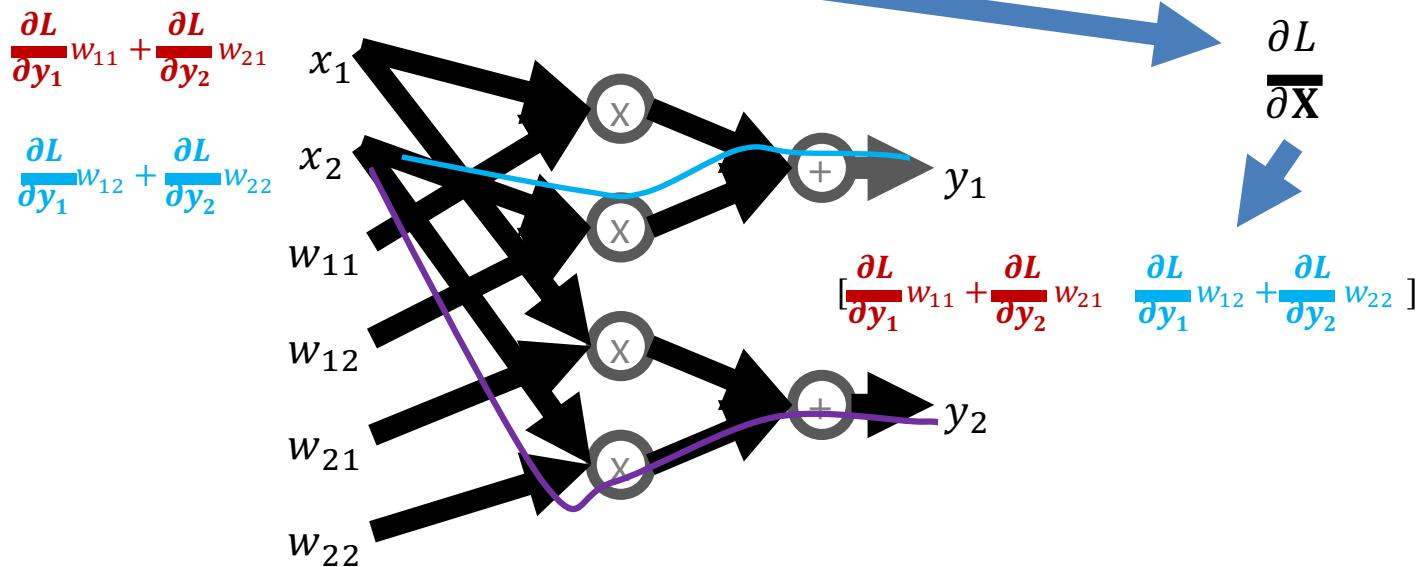
$$[x_1 \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ y_2]$$



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

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

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

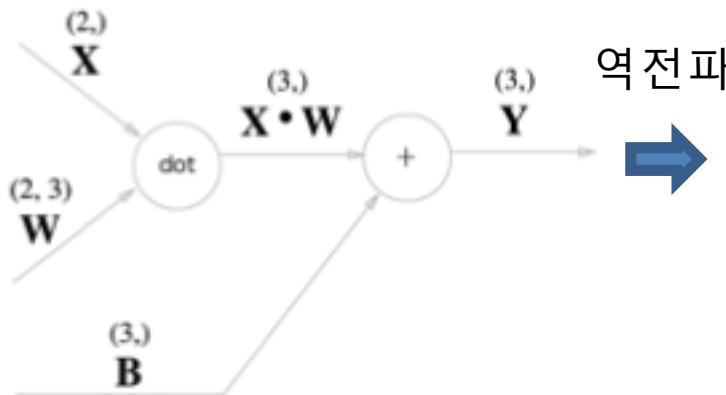
$$\left[\frac{\partial L}{\partial y_1} \quad \frac{\partial L}{\partial y_2} \right] \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

이제 다시 예제
로 돌아와 봅시
다

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

Computational Graph

- Affine 계층



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

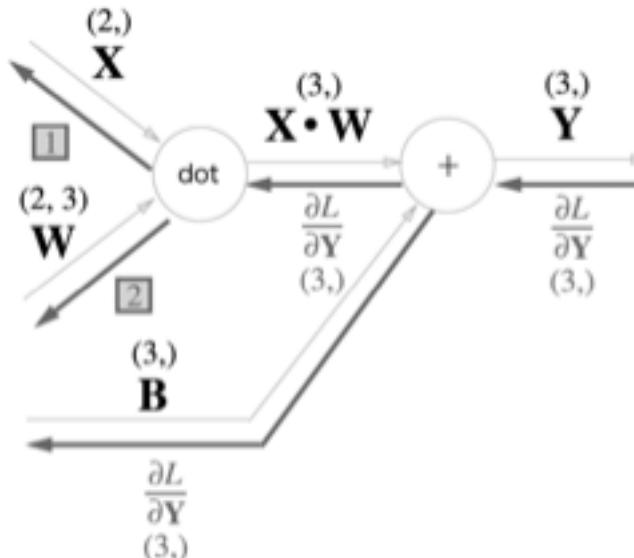
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

Computational Graph

- Affine 계층

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

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



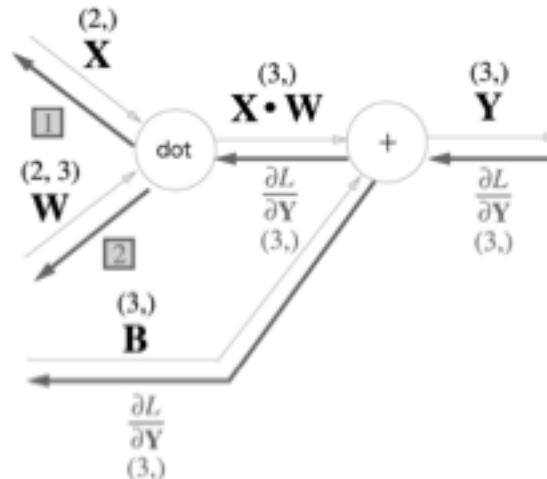
매트릭스의 형상(shape)을 잘 생각해 보면 쉽습니다

Computational Graph

- Affine 계층

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

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



X 와 $\frac{\partial L}{\partial X}$ 와 형상이 같아야 하고

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

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

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

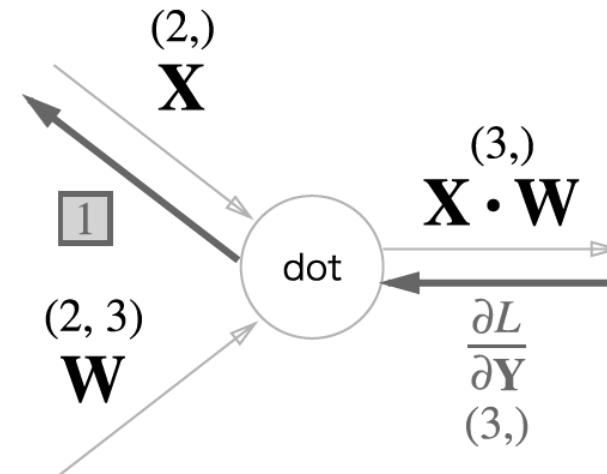
Computational Graph

- Affine 계층

$$\boxed{1} \quad \frac{\partial L}{\partial Y} \cdot W^T = \frac{\partial L}{\partial X}$$

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

↑



그냥 곱셈계층으로 이해하고 형상을 맞춰주면 됩니다

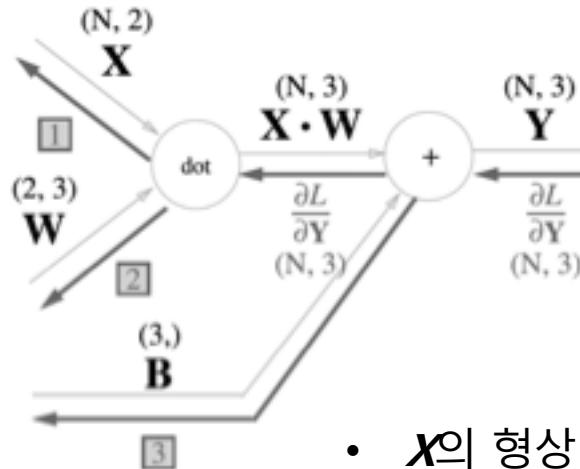
Computational Graph

- 배치용 Affine 계층

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

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

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



• \mathbf{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]])
```

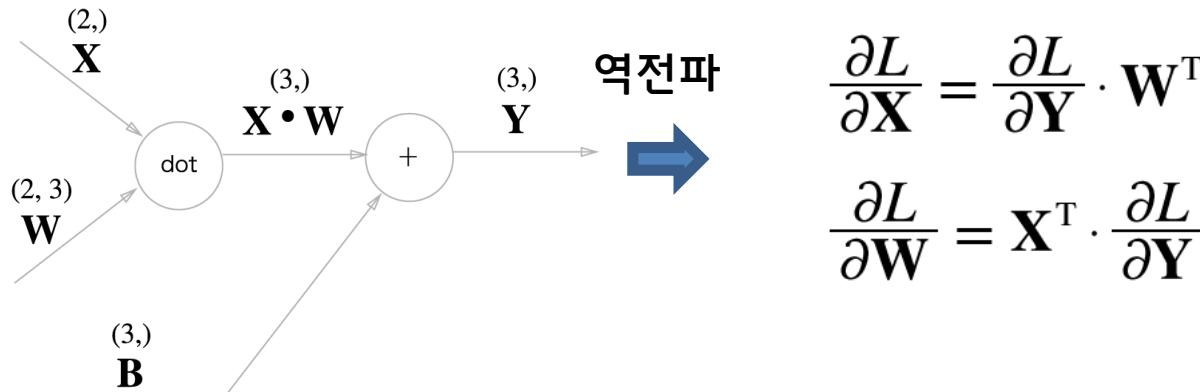
```
In [25]: dB = np.sum(dY, axis=0)
```

```
In [26]: dB
```

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

Computational Graph

- Affine 계층



w 매트릭스가 업데이트 되겠군요

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

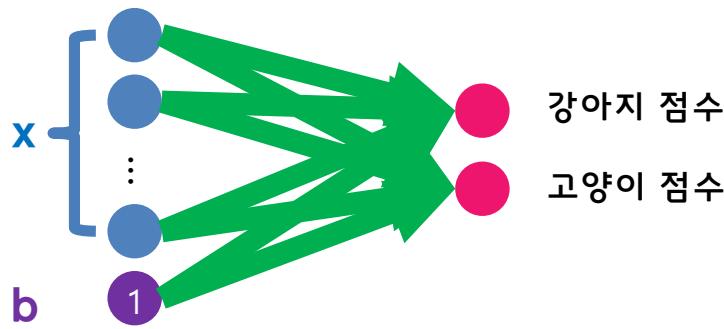


Neural Networks

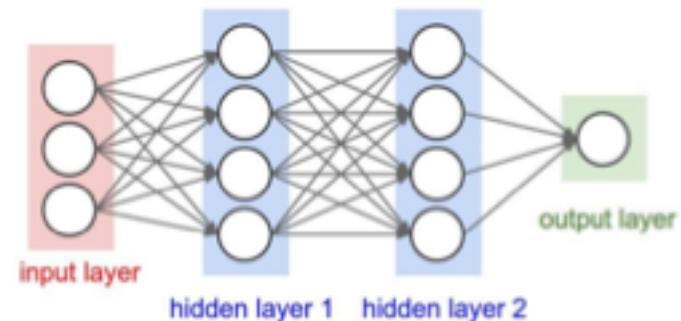
Neural Networks



$$\begin{array}{c}
 \text{강아지 파라미터} \\
 \text{고양이 파라미터}
 \end{array}
 \begin{matrix}
 \begin{matrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \end{matrix} \\
 W
 \end{matrix}
 \times
 \begin{matrix}
 \begin{matrix} 155 \\ 200 \\ - \\ 110 \\ 78 \end{matrix} \\
 X
 \end{matrix}
 +
 \begin{matrix}
 \begin{matrix} 0.1 \\ 0.2 \end{matrix} \\
 b
 \end{matrix}
 =
 \begin{matrix}
 \begin{matrix} 0.3 \\ 0.7 \end{matrix} \\
 X
 \end{matrix}
 \begin{array}{c}
 \text{강아지 점수} \\
 \text{고양이 점수}
 \end{array}$$



Before

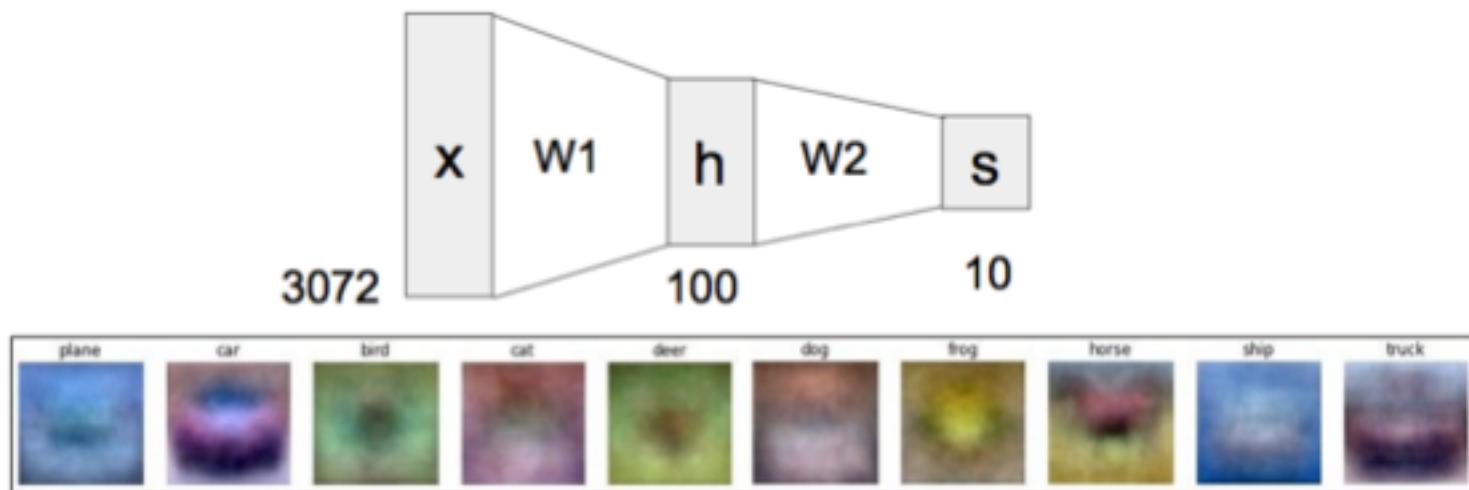


레이어를 쌓아서 더 깊게

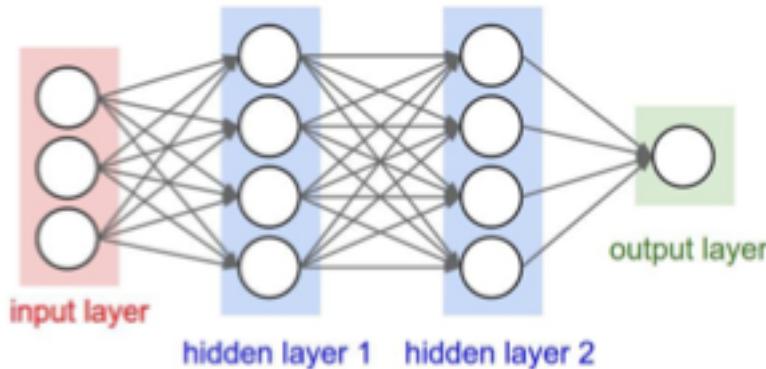
Now

Neural Networks

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



Neural Networks



$$output\ layer = W_2(W_1x) = W_2W_1x = Wx$$

레이어를 하나 쌓는 것
과 같음



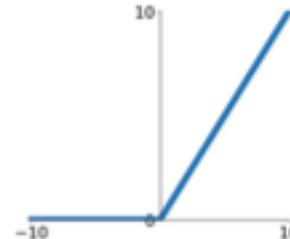
비선형 Activation function
이 필요

Neural Networks

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

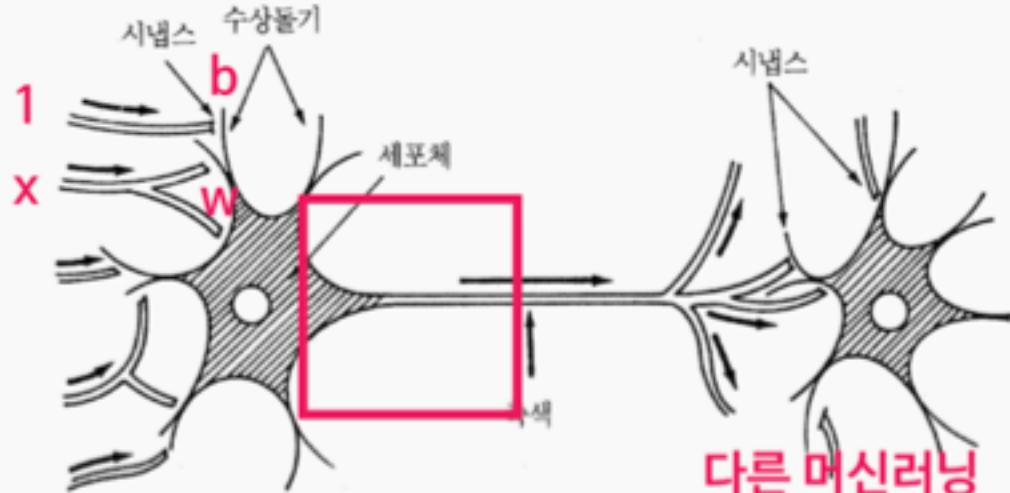
(Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

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

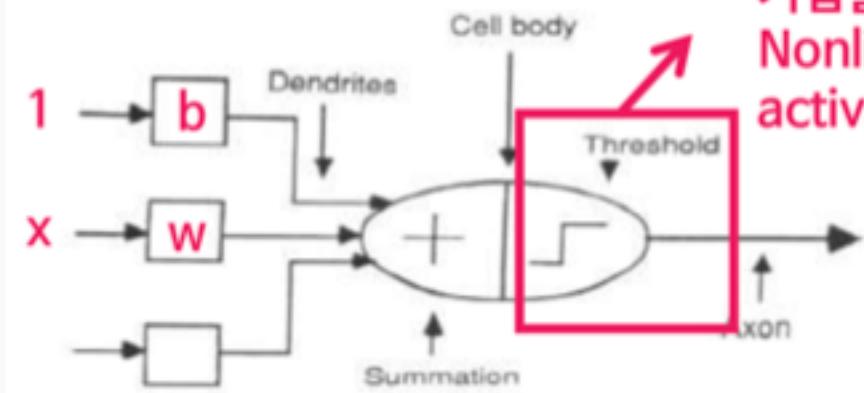


ReLU
(Rectified Linear Unit)

뉴런과 인공뉴런

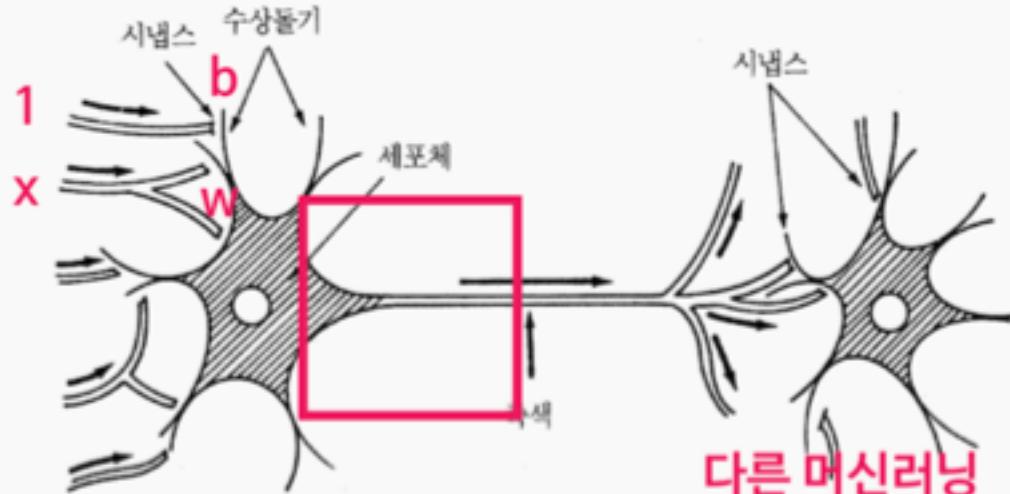


다른 머신러닝
기법들과의 차이점 1:
Nonlinear(복잡한)
activation function

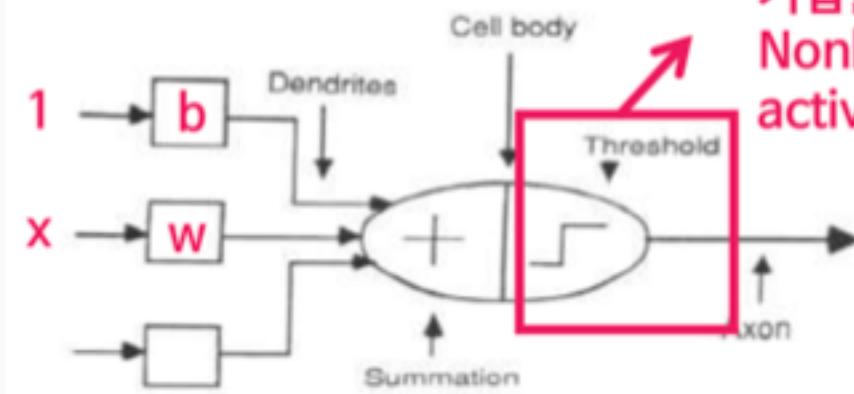


뉴런과 인공뉴런

완전히 단순화 시킨
것이니 주의하세요



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

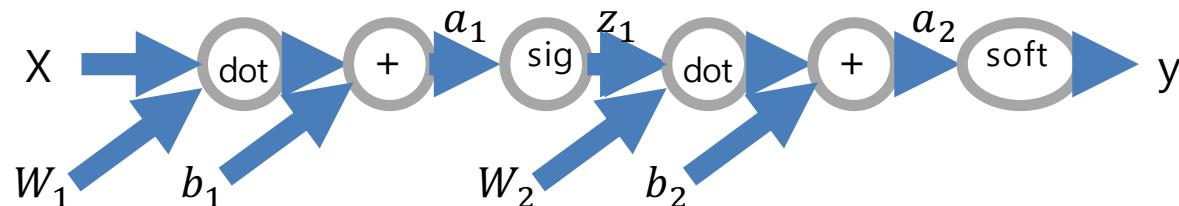


2레이어 네트워크의 구현 예

- **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)
```



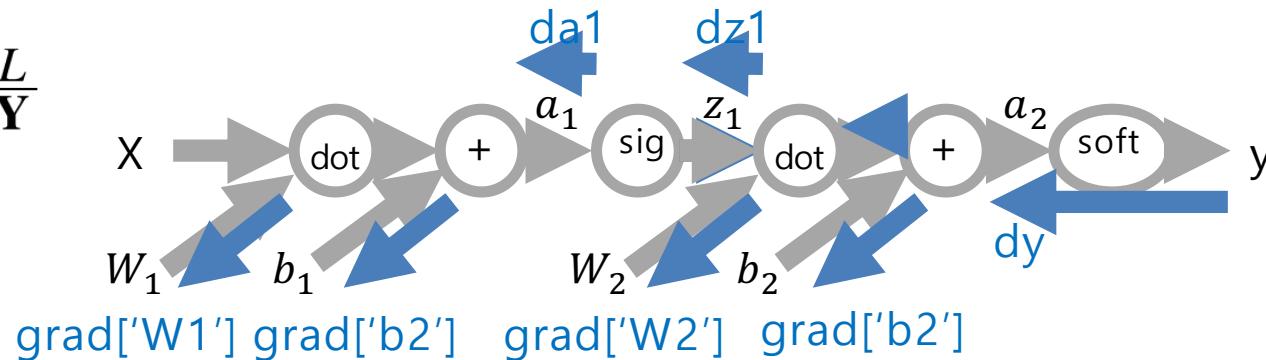
2레이어 네트워크의 구현 예

- **Backward**

- Activation function :
Sigmoid
- Loss function : Softmax loss

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

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

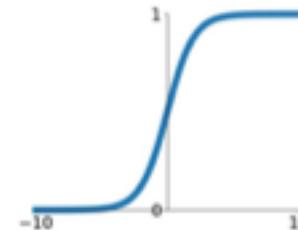


Activation function

- ReLU vs. Sigmoid

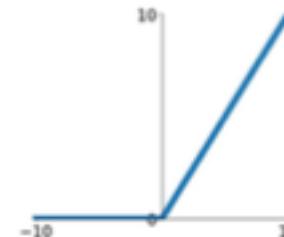
Sigmoid

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



ReLU

$$\max(0, x)$$

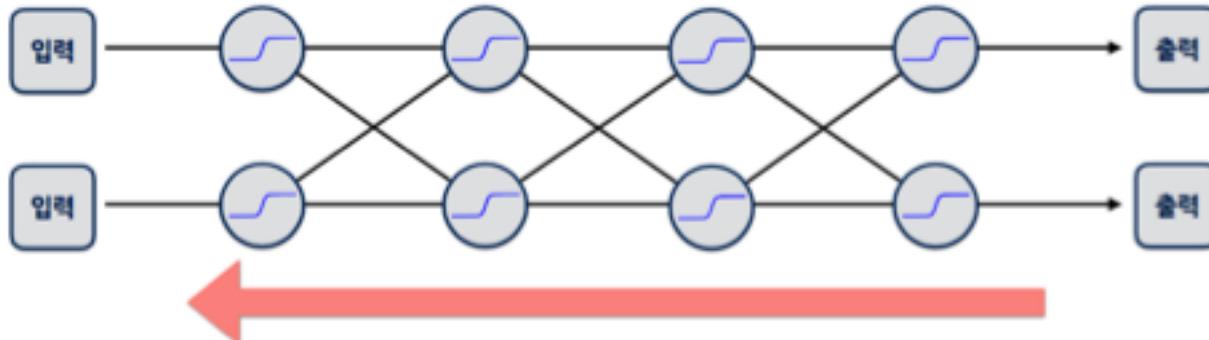


뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 “뒤로 전달”)

뭐를 전달하는가?

현재 내가 틀린정도를 ‘미분(기울기)’ 한 거



미분하고, 곱하고, 더하고를 역방향으로 반복하며 업데이트한다.

근데 문제는?

우리가 activation 함수로 sigmoid  를 썼다는 것

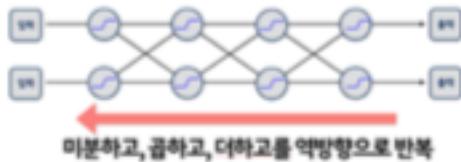


여기의 미분(기울기)는 뭐라도 있다. 다행

```
# 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
grads['W1'] = np.dot(X.T, da1)
grads['b1'] = np.sum(dz1, axis=0)
```

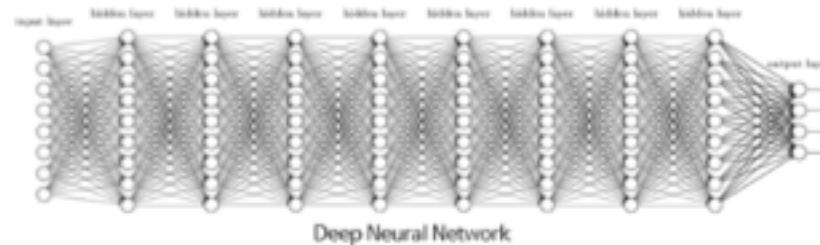
근데 여기는 기울기 0.. 이런거 중간에 곱하면 뭔가 뒤로 전달할게 없다?!



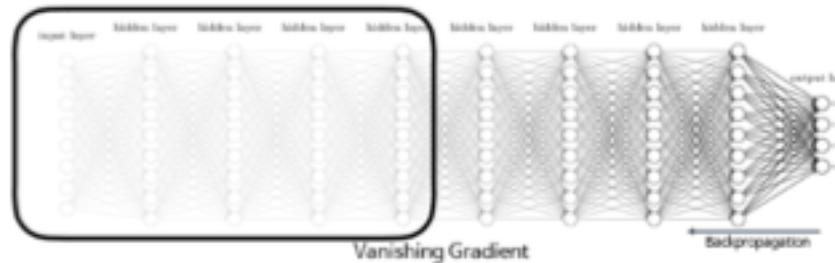
그런 상황에서 이걸 반복하면??????

Vanishing gradient 현상:

레이어가 깊을 수록 업데이트가 사라져간다.
그래서 fitting이 잘 안됨(underfitting)

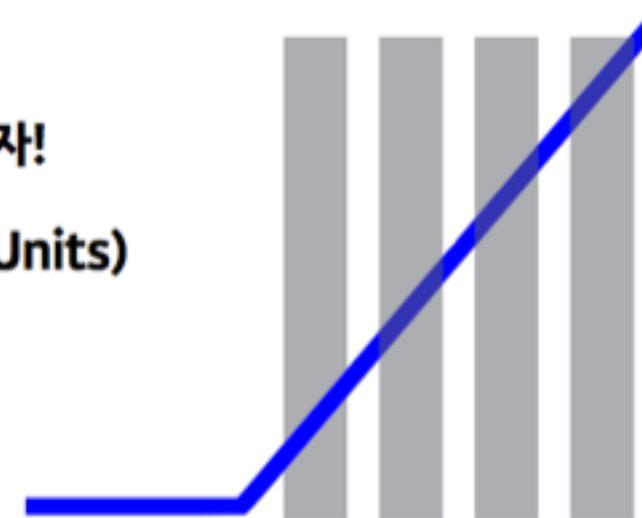


학습이 잘 안됨

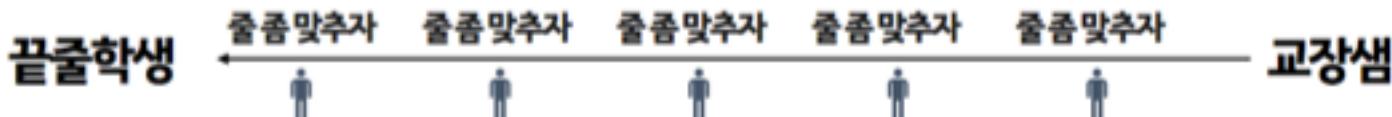


사그라드는 sigmoid 대신
죽지 않는 activation func을 쓰자!

→ **ReLU** (Rectified Linear Units)



이 녀석은 양의 구간에서 전부 미분 값(1)이 있다!

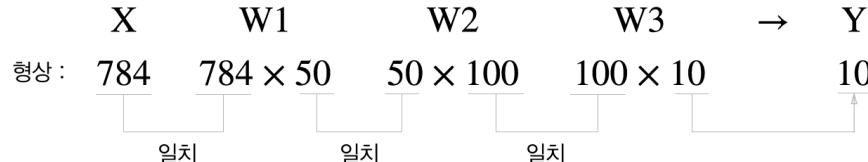


끌 줄 학생까지 이야기가 전달이 잘 되고 위치를 고친다!

배치처리



입력이 $28 \times 28 \times 1$ 인 784-d의 벡터 x 이고, 10개의 숫자 0~9를 예측할 경우
3 레이어 구조



MNIST 데이터

0	4	1	9	2	1	3	1	4	3
5	3	6	1	7	2	8	6	9	2
0	9	1	3	2	4	3	2	7	0
8	6	9	0	5	6	0	7	6	1
8	7	9	3	9	8	5	5	3	3
0	7	4	9	8	0	9	4	1	4
4	6	0	4	5	6	1	0	0	1
7	1	6	3	0	2	1	1	7	9
0	2	6	6	7	8	3	0	4	1
7	4					8	3		5

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

In [7]: network = init_network()

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

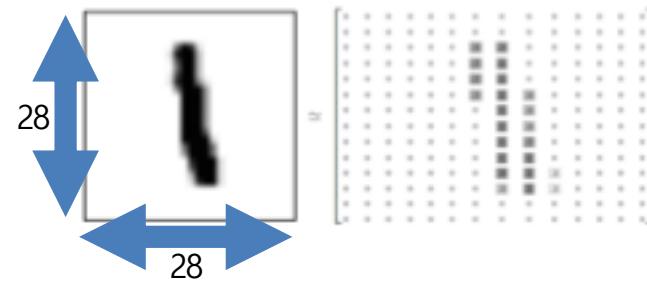
In [9]: x.shape
Out[9]: (10000, 784)

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)
```

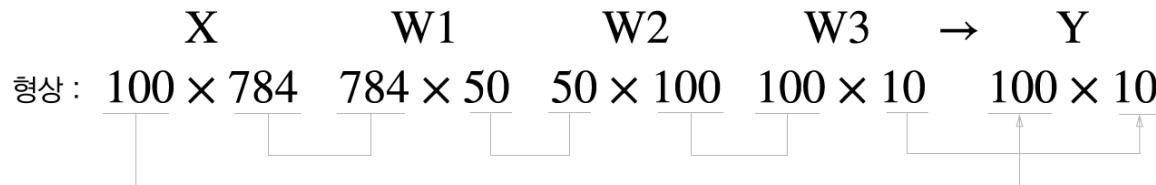


28x28x1 MNIST 데이터

배치처리

입력이 $28 \times 28 \times 1$ 인 784-d의 벡터 x 이고, 10개의 숫자 0~9를 예측할 경우
3 레이어 구조

- 이미지 100개를 묶어서 한번에 넘기기



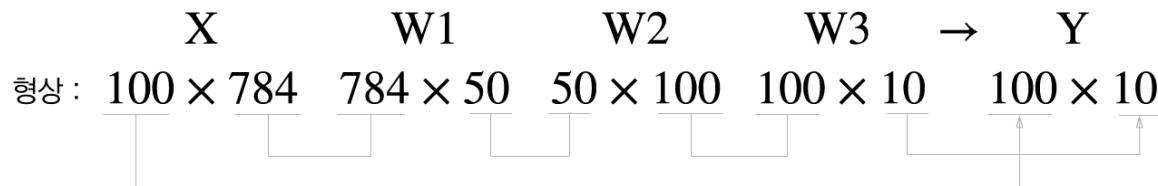
- 입력은 100x784이고 출력은 100x10이 됨
 - 이는 100장의 입력 데이터가 한번에 출력됨을 의미합니다

100장이 한번에 처리 될 수 있군요

배치처리

입력이 $28 \times 28 \times 1$ 인 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



- 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

코드와 함께 보는 신경망 관련 계층

활성화 함수계층

- ReLU 계층

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



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



```

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

- 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.]]
```

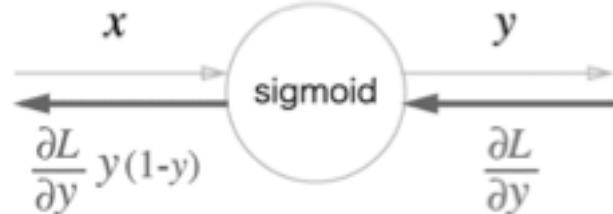
```
class Relu:  
    def __init__(self):  
        self.mask = None  
  
    백워드 용 저장  
    def forward(self, x):  
        self.mask = (x <= 0)  
        out = x.copy()  
        out[self.mask] = 0  
  
        return out  
  
    def backward(self, dout):  
        dout[self.mask] = 0  
        dx = dout  
  
        return dx  
  
    저장된 마  
    스크 이용
```

활성화 함수계층

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

- 2_day/code/common/layers.py

```
class Sigmoid:  
    def __init__(self):  
        self.out = None  
  
    def forward(self, x):  
        out = sigmoid(x)  
        self.out = out    출력 y를 저장  
        return out  
  
    def backward(self, dout):  
        dx = dout * (1.0 - self.out) * self.out  
        계산  
        return dx
```



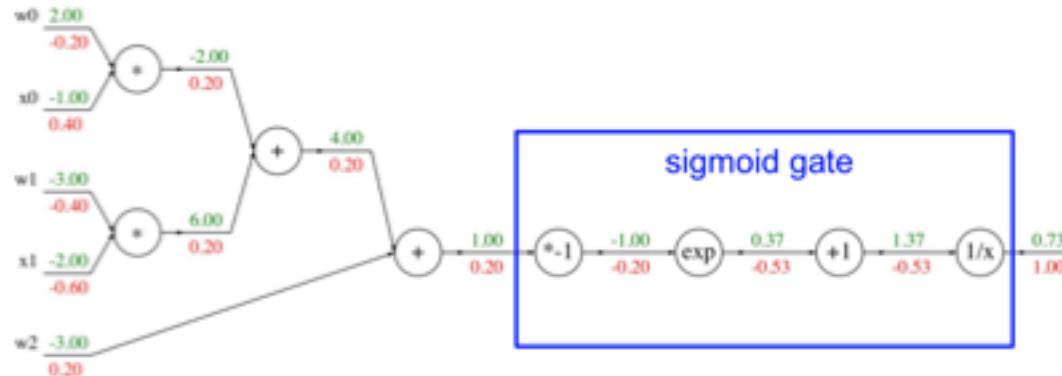
활성화 함수계층 구현하기

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

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

sigmoid function

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



sigmoid gate

실습 1: 2 레이어 네트워크 실습

- 1) 실행 파일 (손글씨 인식) : 2day/prac2_train_neuralnet.py

```
24 # 10회당 반복 수
25 iter_per_epoch = max(train_size / batch_size, 1)
26
27 for i in range(iters_num):
28     # 미니배치 획득
29     batch_mask = np.random.choice(train_size, batch_size)
30     x_batch = x_train[batch_mask]
31     t_batch = t_train[batch_mask]
32
33     # 기울기 계산
34     grad = network.numerical_gradient(x_batch, t_batch)
35     grad = network.gradient(x_batch, t_batch)          구현할 함수
36
37     # 미개변수 경신
38     for key in ('W1', 'b1', 'W2', 'b2'):
39         network.params[key] -= learning_rate * grad[key]
40
41     # 학습 결과 기록
42     loss = network.loss(x_batch, t_batch)
43     train_loss_list.append(loss)
44
45     # 1에폭당 정확도 계산
46     if i % iter_per_epoch == 0:
47         train_acc = network.accuracy(x_train, t_train)
48         test_acc = network.accuracy(x_test, t_test)
49         train_acc_list.append(train_acc)
50         test_acc_list.append(test_acc)
51         print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

실습 1 : 2 레이어 네트워크 실습

- 2) 구현할 함수 : 2day/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



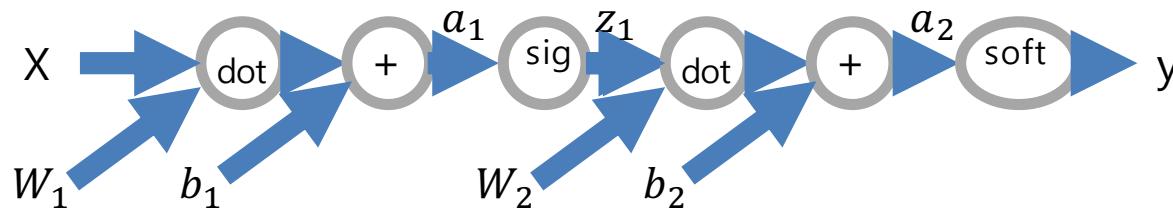
- 2) 구현할 함수 : 2day/prac2_simple_two_Layer_net.py
 - 순전파를 계산 그래프로 표현해 봅시다

```
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

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

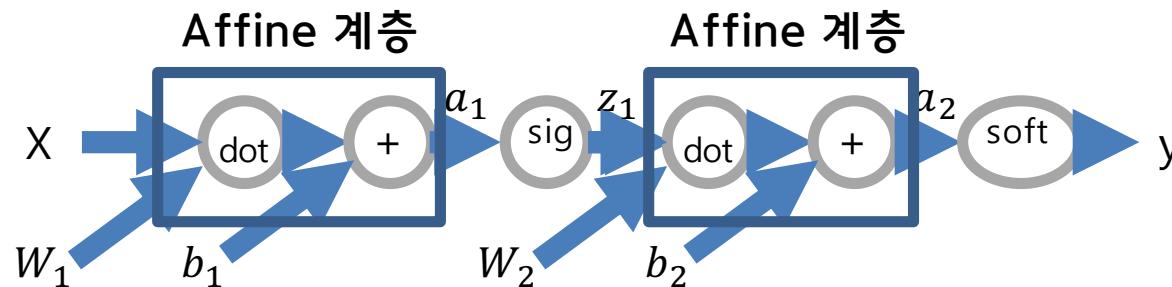
```
62      # forward  
63      a1 = np.dot(x, W1) + b1  
64      z1 = sigmoid(a1)  
65      a2 = np.dot(z1, W2) + b2  
66      y = softmax(a2)
```



실습 1 : 2 레이어 네트워크 실습

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

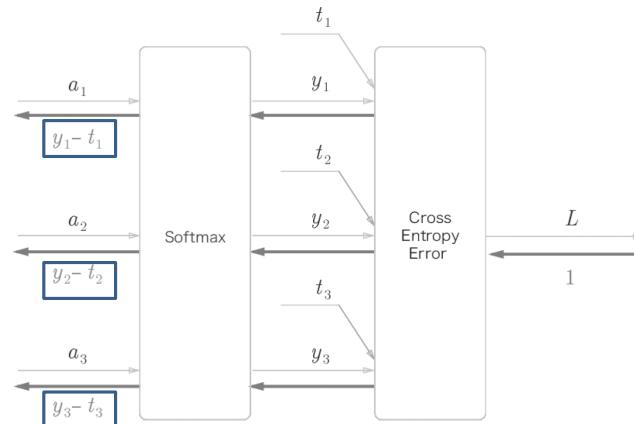
```
62  
63  
64  
65  
66 # forward  
    a1 = np.dot(x, W1) + b1  
    z1 = sigmoid(a1)  
    a2 = np.dot(z1, W2) + b2  
    y = softmax(a2)
```



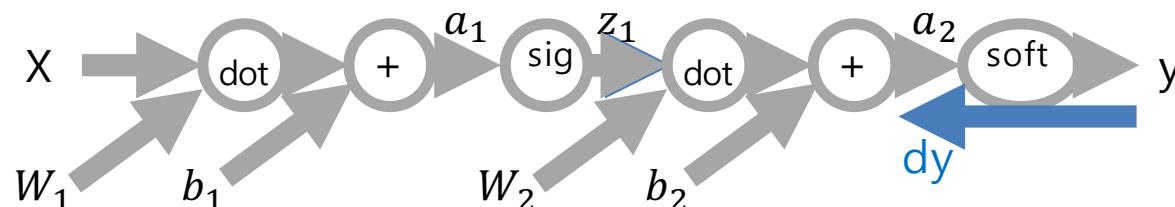
실습 1 : 2 레이어 네트워크 실습

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

```
68
69     # backward
70     dy = (y - t) / batch_num
71     ###### Write your codes #####
72     ###### None #####
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
```



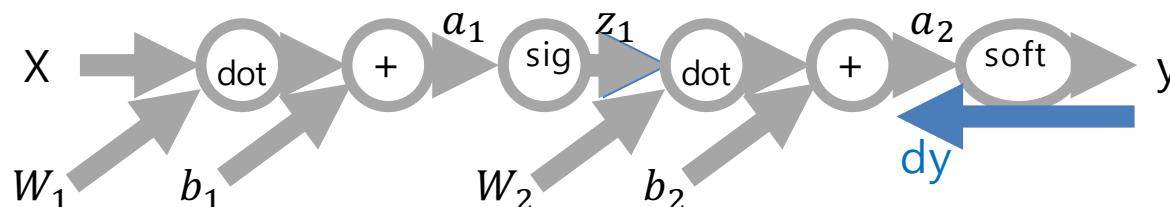
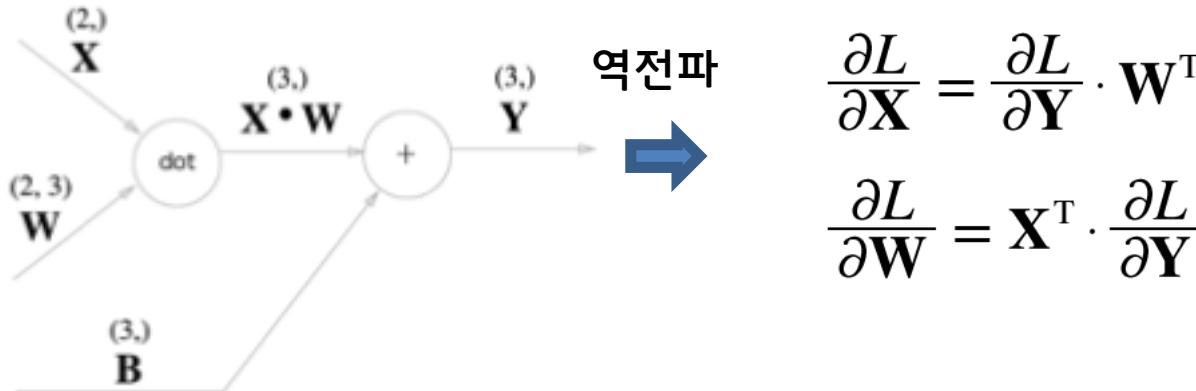
Softmax
loss 부분은
이미 미분 해
두었습니다



실습 1 : 2 레이어 네트워크 실습

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

Affine 계층의 역전파를 기억해 봅시다



실습 1: 2 레이어 네트워크 실습

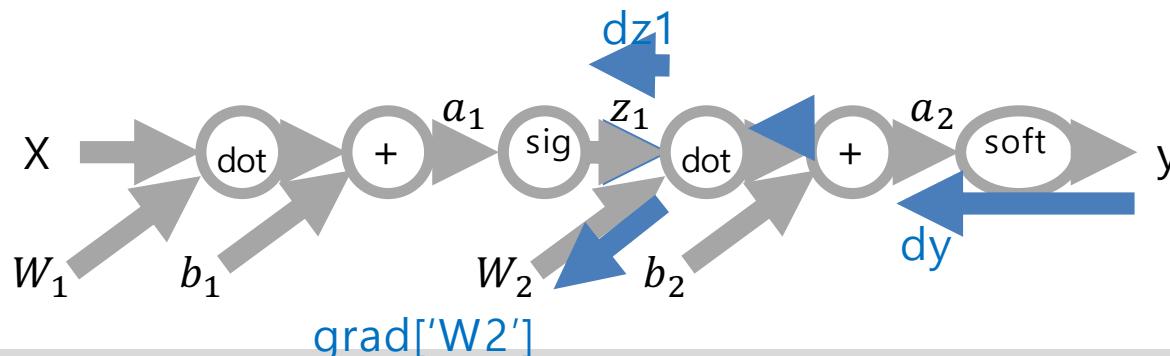
- 2) 구현할 함수 : 2day/prac2_simple_two_Layer_net.py

Affine 계층의 역전파를 기억해 봅시다

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

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

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



실습 1 : 2 레이어 네트워크 실습

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

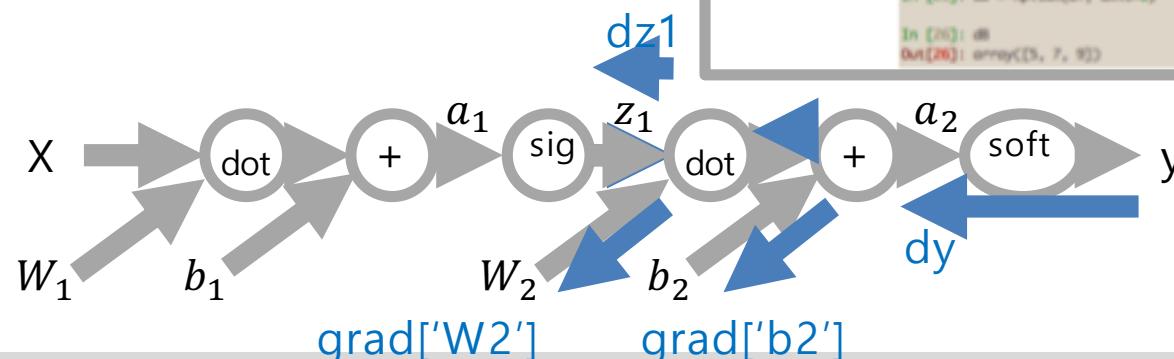
```
68     # backward
69     dy = (y - t) / batch_num
70     ###### Write your codes #####
71
72     ###### None #####
73     grads['W2'] = None
74     grads['b2'] = None
75
76     dz1 = None
77     da1 = None
78     grads['W1'] = None
79     grads['b1'] = None
```

Affine / Softmax 계층 구현하기

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

```
In [23]: dy = np.array([[1,2,3],[4,5,6]])
In [24]: dy
Out[24]:
array([[1, 2, 3],
       [4, 5, 6]])

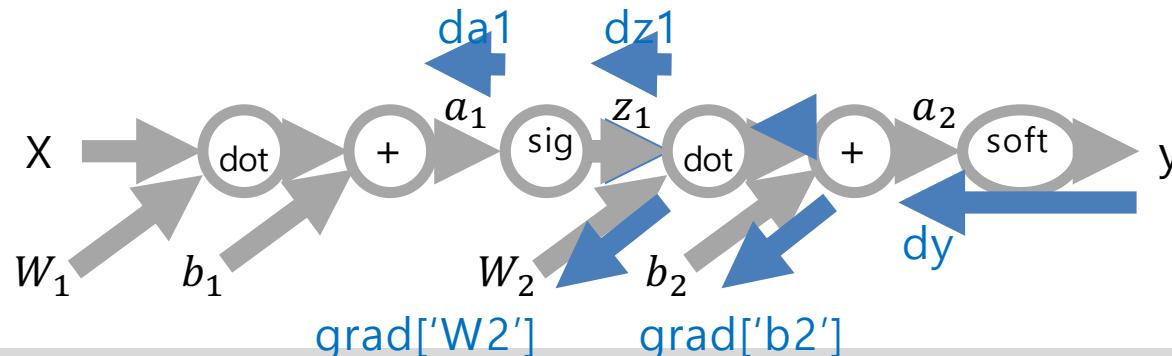
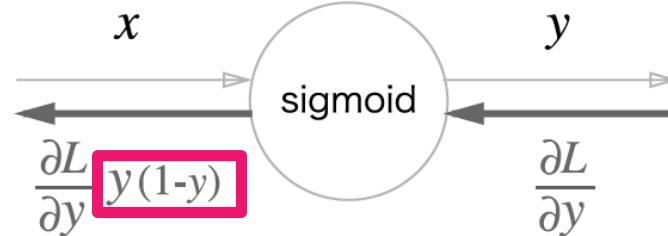
In [25]: db = np.sum(dy, axis=0)
In [26]: db
Out[26]: array([5, 7, 9])
```



실습 1 : 2 레이어 네트워크 실습

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

```
68 # backward
69 dy = (y - t) / batch_num
70 ##### Write your codes #####
71 grads['W2'] = None
72 grads['b2'] = None
73
74 d1 = None
75 da1 = None
76 grads['W1'] = None
77 grads['b1'] = None
78
```



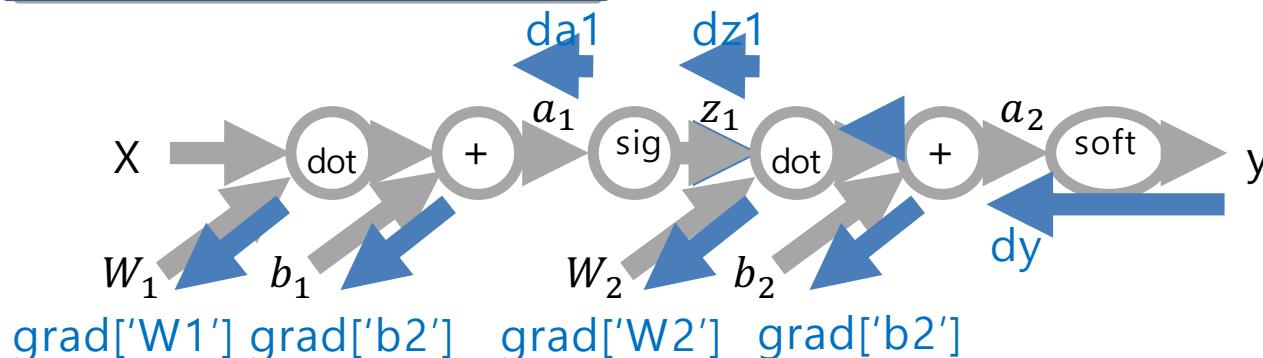
실습 1 : 2 레이어 네트워크 실습

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

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

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

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



실습 2 : 2 레이어 네트워크 모듈화



- 1) 실행파일 : 3day/prac2_train_neuralnet.py

이전과 같은
API

```
1 # coding: utf-8
2 import sys, os
3 sys.path.append(os.pardir)
4
5 import numpy as np
6 from dataset.mnist import load_mnist
7 from prac2_two_layer_net import TwoLayerNet
8
9 # 데이터 읽기
10 (x_train, t_train), (x_test, t_test) = load_mnist(normalize=True, one_hot_label=True)
11
12 network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)
13
14 iterers_num = 10000
15 train_size = x_train.shape[0]
16 batch_size = 100
17 learning_rate = 0.1
18
19 train_loss_list = []
20 train_acc_list = []
21 test_acc_list = []
22
23 iter_per_epoch = max(train_size / batch_size, 1)
24
25 for i in range(iterers_num):
26     batch_mask = np.random.choice(train_size, batch_size)
27     x_batch = x_train[batch_mask]
28     t_batch = t_train[batch_mask]
29
30     # 기울기 계산
31     dgrad = network.numerical_gradient(x_batch, t_batch) # 수치 미분 방식
32     grad = network.gradient(x_batch, t_batch) # 오차역전파법 방식(훨씬 빠르다)
33
34     # 경신
35     for key in ('W1', 'b1', 'W2', 'b2'):
36         network.params[key] -= learning_rate * grad[key]
37
38     loss = network.loss(x_batch, t_batch)
39     train_loss_list.append(loss)
40
41     if i % iter_per_epoch == 0:
42         train_acc = network.accuracy(x_train, t_train)
43         test_acc = network.accuracy(x_test, t_test)
44         train_acc_list.append(train_acc)
45         test_acc_list.append(test_acc)
46         print(train_acc, test_acc)
```

실습 2 : 2 레이어 네트워크 모듈화

- 2) 구현파일 : 3day/prac2_two_layer_net.py

```
1  # coding: utf-8
2  import sys, os
3  sys.path.append(os.pardir) # 부모 디렉터리의 파일을 가져올 수 있도록 설정
4  import numpy as np
5  from common.layers import *
6  from common.gradient import numerical_gradient
7  from collections import OrderedDict
8
9
10 class TwoLayerNet:
11
12     def __init__(self, input_size, hidden_size, output_size, weight_init_std = 0.01):
13         # 가중치 초기화
14         self.params = {}
15         self.params['W1'] = weight_init_std * np.random.randn(input_size, hidden_size)
16         self.params['b1'] = np.zeros(hidden_size)
17         self.params['W2'] = weight_init_std * np.random.randn(hidden_size, output_size)
18         self.params['b2'] = np.zeros(output_size)
19
20         # 계층 생성
21         self.layers = OrderedDict() # Dictionary인데 순서를 갖고 있음
22         self.layers['Affine1'] = Affine(self.params['W1'], self.params['b1'])
23         self.layers['Relu1'] = Relu()
24         self.layers['Affine2'] = Affine(self.params['W2'], self.params['b2'])
25
26         self.lastLayer = SoftmaxWithLoss() # common/layers.py 참조
27
28     def predict(self, x):
29         for layer in self.layers.values():
30             x = layer.forward(x)
31
32     return x
```

실습 2 : 2 레이어 네트워크 모듈화

- 2) 구현파일 : 3day/prac2_two_layer_net.py
common/layers.py

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


```
7  class Relu:
8      def __init__(self):
9          self.mask = None
10
11     def forward(self, x):
12         self.mask = (x <= 0)
13         out = x.copy()
14         out[self.mask] = 0
15
16         return out
17
18     def backward(self, dout):
19         dout[self.mask] = 0
20         dx = dout
21
22         return dx
```

```
48     class Affine:
49         def __init__(self, W, b):
50             self.W = W
51             self.b = b
52
53             self.x = None
54             self.original_x_shape = None
55             # 가중치와 편향 매개변수의 미분
56             self.dW = None
57             self.db = None
58
59
60         def forward(self, x):
61             # 텐서 대용
62             self.original_x_shape = x.shape
63             x = x.reshape(x.shape[0], -1)
64             self.x = x
65
66             out = np.dot(self.x, self.W) + self.b
67
68             return out
69
70         def backward(self, dout):
71             dx = np.dot(dout, self.W.T)
72             self.dW = np.dot(self.x.T, dout)
73             self.db = np.sum(dout, axis=0)
74
75             dx = dx.reshape(*self.original_x_shape) # 입력 데이터 모양 변경(텐서 대용)
76             return dx
```

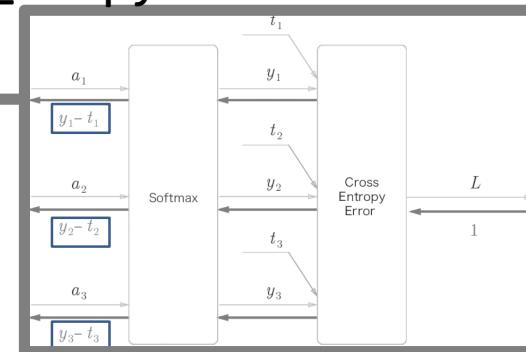
$$\frac{\partial L}{\partial \mathbf{X}} = \frac{\partial L}{\partial \mathbf{Y}} \cdot \mathbf{W}^T$$

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

실습 2 : 2 레이어 네트워크 모듈화

- 2) 구현파일 : 3day/prac2_two_layer_net.py
common/layers.py

```
70 class SoftmaxWithLoss:  
71     def __init__(self):  
72         self.loss = None # 손실함수  
73         self.y = None # softmax의 출력  
74         self.t = None # 정답 레이블(원-핫 인코딩 형태)  
75  
76     def forward(self, x, t):  
77         self.t = t  
78         self.y = softmax(x)  
79         self.loss = cross_entropy_error(self.y, self.t)  
80  
81         return self.loss  
82  
83     def backward(self, dout=1):  
84         batch_size = self.t.shape[0]  
85         if self.t.size == self.y.size: # 정답 레이블이 원-핫 인코딩 형태일 때  
86             dx = (self.y - self.t) / batch_size  
87         else:  
88             dx = self.y.copy()  
89             dx[np.arange(batch_size), self.t] -= 1  
90             dx = dx / batch_size  
91  
92         return dx
```



실습 2 : 2 레이어 네트워크 모듈화



- 2) 구현파일 : 3day/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 your code #####
71         #####
72         for layer in layers:
73             dout = None
74             ##### end of your code #####
75
76         # 결과 저장
77         grads = {}
78         grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
79         grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db
80
81         return grads
```

여기만 구현하면 됩니다

지금까지 살펴본 내용



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

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



박 은 수 Research Director

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