Optimization

모두의연구소
박은수 Research Director

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

( , 고양이)

총 class가
강아지,
개구리,
고양이
라 가정

컴퓨터가 연산 가능한 형태로

인덱스 번호

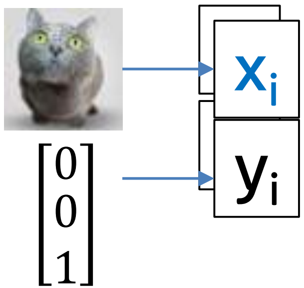
( , 2)

or

One-hot Encoding

( , $\begin{array}{c} \text{강아지} \\ \text{개구리} \\ \text{고양이} \end{array} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$)

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



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

e.g

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

x_i

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

y_i

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

e.g

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

고양이가 입력이면 고양이 점수가 높아야 함

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \cdots & 0.3 & 0.7 \\ 0.7 & 0.8 & \cdots & 0.9 & 0.4 \\ 0.1 & 0.9 & \cdots & 0.7 & 0.6 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix} 155 \\ 200 \\ \cdots \\ 110 \\ 78 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.1 \\ 0.2 \\ 0.8 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}}_{s}$$

강아지 점수
개구리 점수
고양이 점수

x_i

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

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

e.g

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

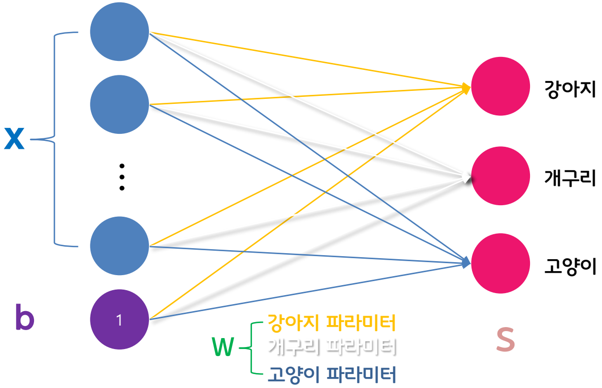
$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \\ 0.1 & 0.9 & \dots & 0.7 & 0.6 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.1 \\ 0.2 \\ 0.8 \end{bmatrix}}_{b}$$

똑같은 표현 임

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 & 0.1 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 & 0.2 \\ 0.1 & 0.9 & \dots & 0.7 & 0.6 & 0.8 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \\ 1 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}}_{s} \begin{matrix} \text{강아지} \\ \text{개구리} \\ \text{고양이} \end{matrix}$$

b

입력에 1추가



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

e.g

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

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \\ 0.1 & 0.9 & \dots & 0.7 & 0.6 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.1 \\ 0.2 \\ 0.8 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}}_{s} \begin{array}{l} \text{강아지 점수} \\ \text{개구리 점수} \\ \text{고양이 점수} \end{array}$$

$$f(x_i ; W)$$

W

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

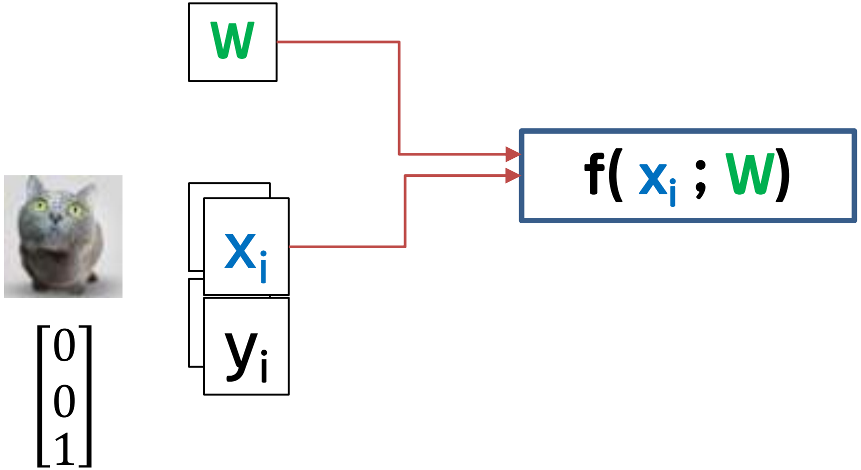
x_i

y_i

$$\boxed{f(x_i ; W)}$$

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

$$L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$



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

Softmax vs. SVM

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

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

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \\ 0.1 & 0.9 & \dots & 0.7 & 0.6 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix} 155 \\ 200 \\ \dots \\ 110 \\ 78 \end{bmatrix}}_{x} + \underbrace{\begin{bmatrix} 0.1 \\ 0.2 \\ 0.8 \end{bmatrix}}_{b} = \underbrace{\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix}}_{S}$$

scores = unnormalized log probabilities of the classes.

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

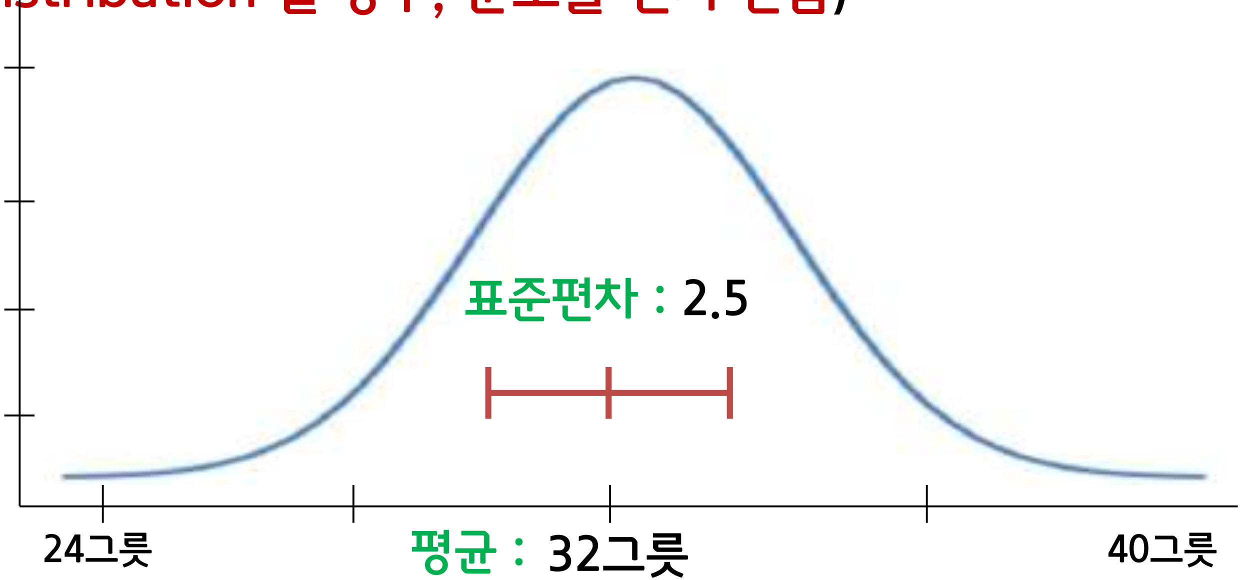
Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class

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

Probability vs Likelihood

Probability

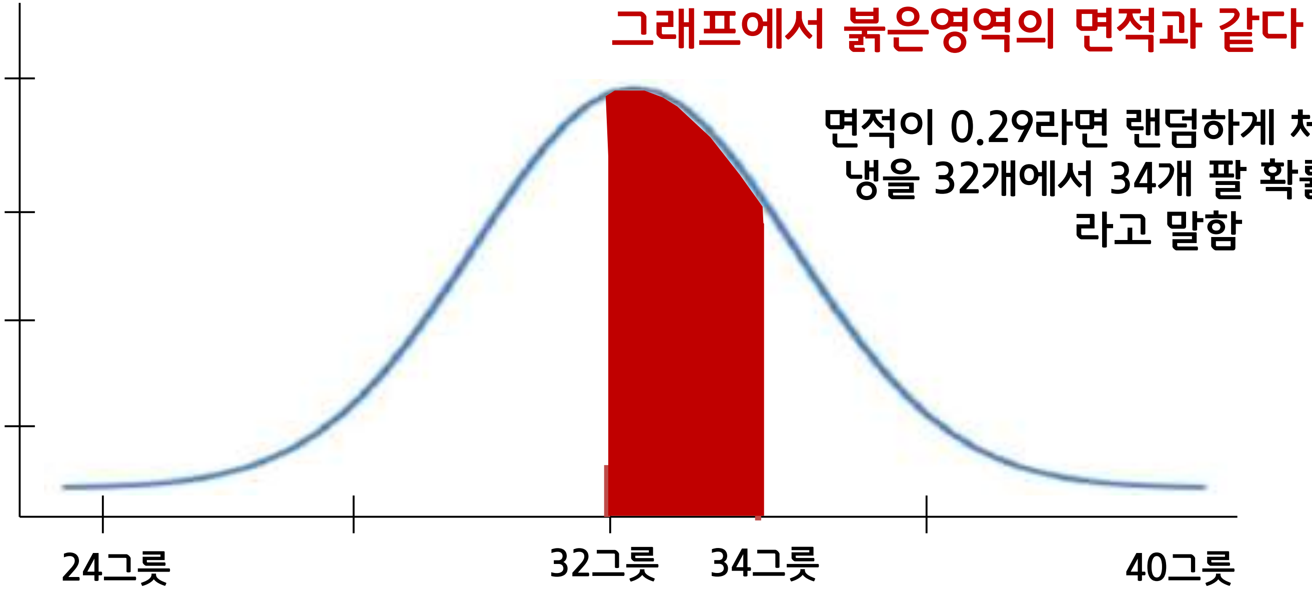
- ‘딥러닝 면옥’ 체인점들의 하루 비빔냉면 판매량 (normal distribution 일 경우, 분포를 먼저 얻음)



Probability vs Likelihood

Probability

랜덤하게 선택한 체인점이 하루에 비빔면을 32개에서 34개 판매할 확률 :



면적이 0.29라면 랜덤하게 체인점이 비냉을 32개에서 34개 팔 확률은 29%라고 말함

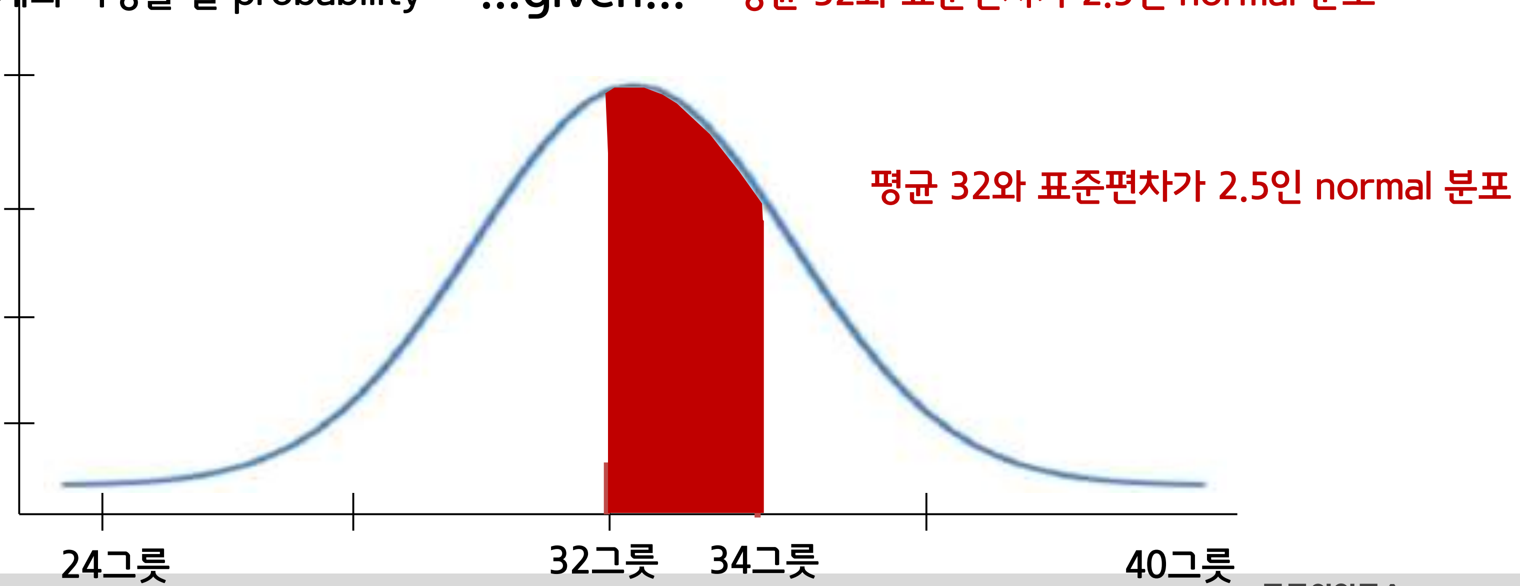
Probability vs Likelihood

Probability

수학적으로 표현해 보면

$$P(\text{32개에서 34개의 비냉을 팜} \mid \boldsymbol{mean = 32\ and\ deviation = 2.5})$$

하루 32개에서 34개의 비냉을 팔 probability ...given... 평균 32와 표준편차가 2.5인 normal 분포



From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Probability

수학적으로 표현해 보면

$$P(32\text{개에서 } 34\text{개의 비냉을 팜} \mid \boldsymbol{mean = 32\ and\ deviation = 2.5})$$

하루 32개에서 34개의 비냉을 팔 probability ...given... 평균 32와 표준편차가 2.5인 normal 분포

From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

모두의연구소

Probability vs Likelihood

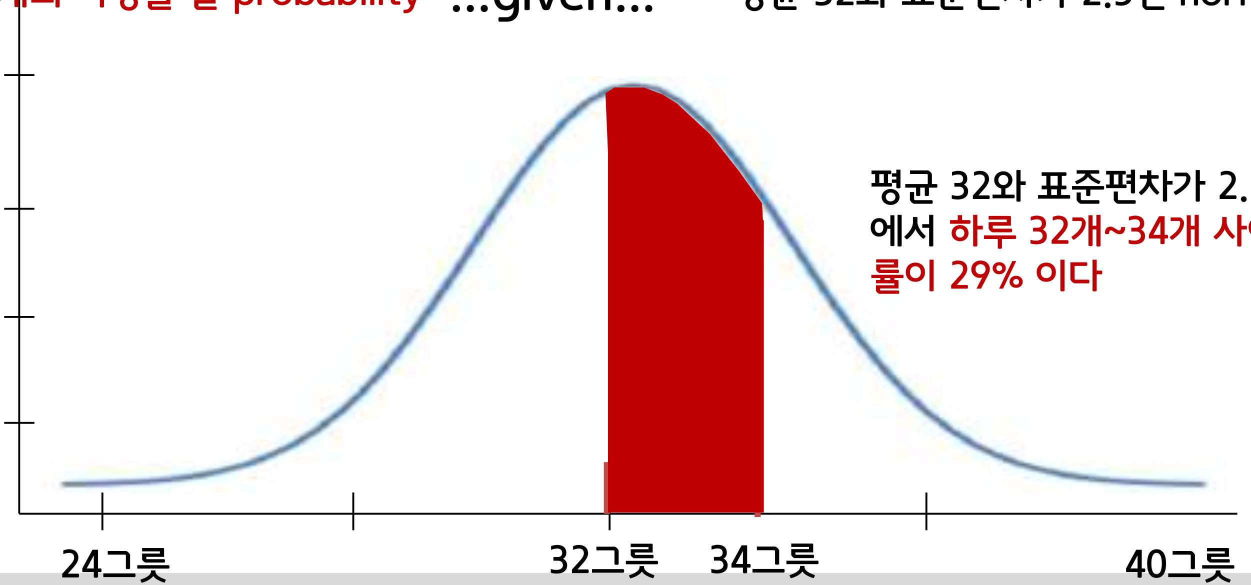
모두의연구소

Probability

수학적으로 표현해 보면

$P(32\text{개에서 } 34\text{개의 비냉을 팜} \mid \boldsymbol{mean} = \mathbf{32}\ \boldsymbol{and}\ \boldsymbol{deviation} = \mathbf{2.5})$

하루 32개에서 34개의 비냉을 팔 probability ...given... 평균 32와 표준편차가 2.5인 normal 분포



평균 32와 표준편차가 2.5인 normal 분포에서 하루 32개~34개 사이의 비냉을 팔 확률이 29% 이다

Probability vs Likelihood

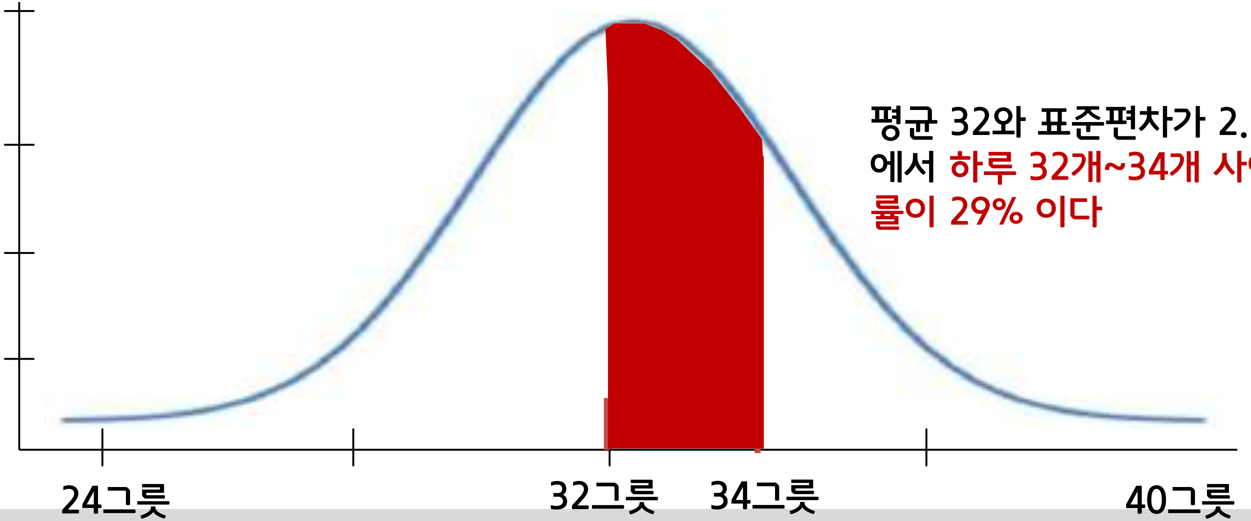
Probability

수학적으로 표현해 보면

고정 값

$$P(32\text{개에서 }34\text{개의 비냉을 팜} \mid \boldsymbol{mean = 32\ and\ deviation = 2.5})$$

우리가 다른 비냉판매 수에 관심이 있다면 변경 가능한 부분



평균 32와 표준편차가 2.5인 normal 분포에서 하루 32개~34개 사이의 비냉을 팔 확률이 29% 이다

Probability vs Likelihood

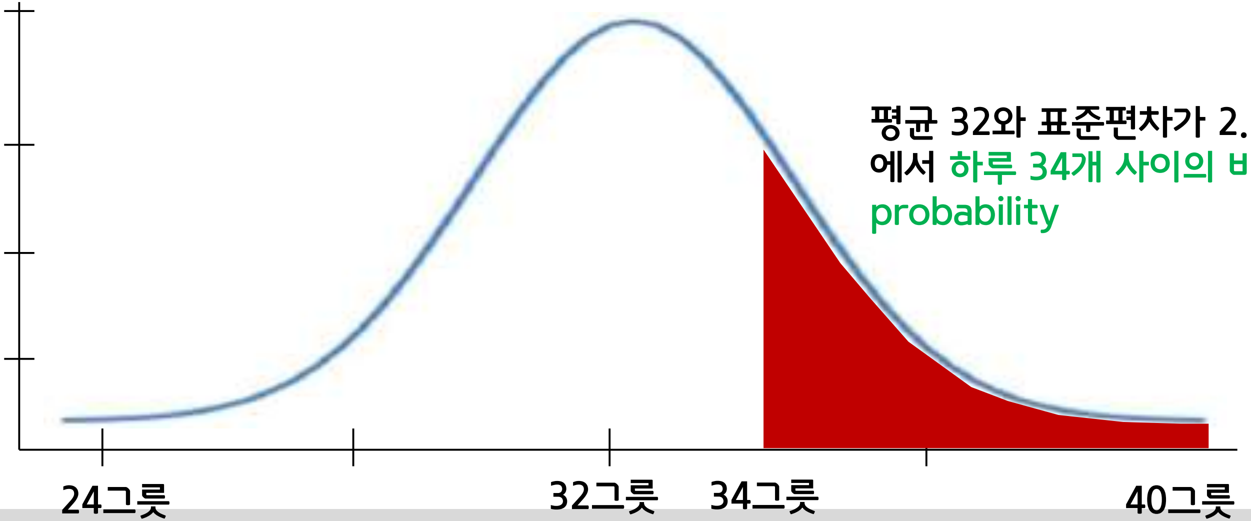
Probability

수학적으로 표현해 보면

고정

$$P(\text{비냉판매 수} > 34 \mid \boldsymbol{mean = 32\ and\ deviation = 2.5})$$

34개 이상 비냉을 판매할 수 체인점의 probability



평균 32와 표준편차가 2.5인 normal 분포에서 하루 34개 사이의 비냉을 팔 probability

From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

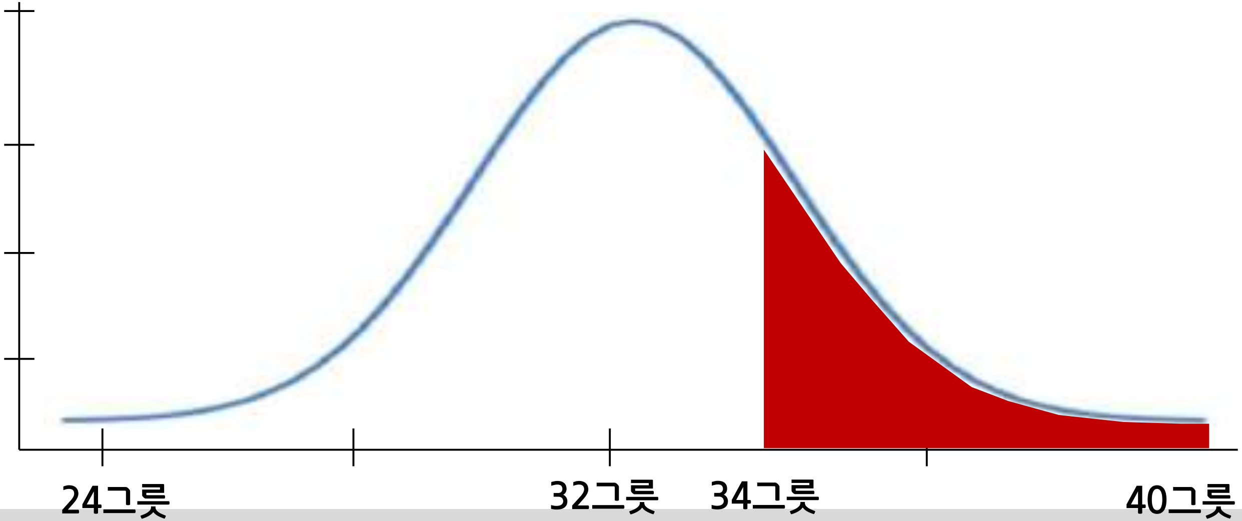
Probability vs Likelihood

Probability

수학적으로 표현해 보면

$$P(\text{비냉판매 수} > 34 \mid mean = 32 \overset{\text{고정}}{and} \; deviation = 2.5)$$

분포를 오른쪽에 고정 시켜 두고 왼쪽을 변경 시켜 새로운 probability를 얻을 수 있다

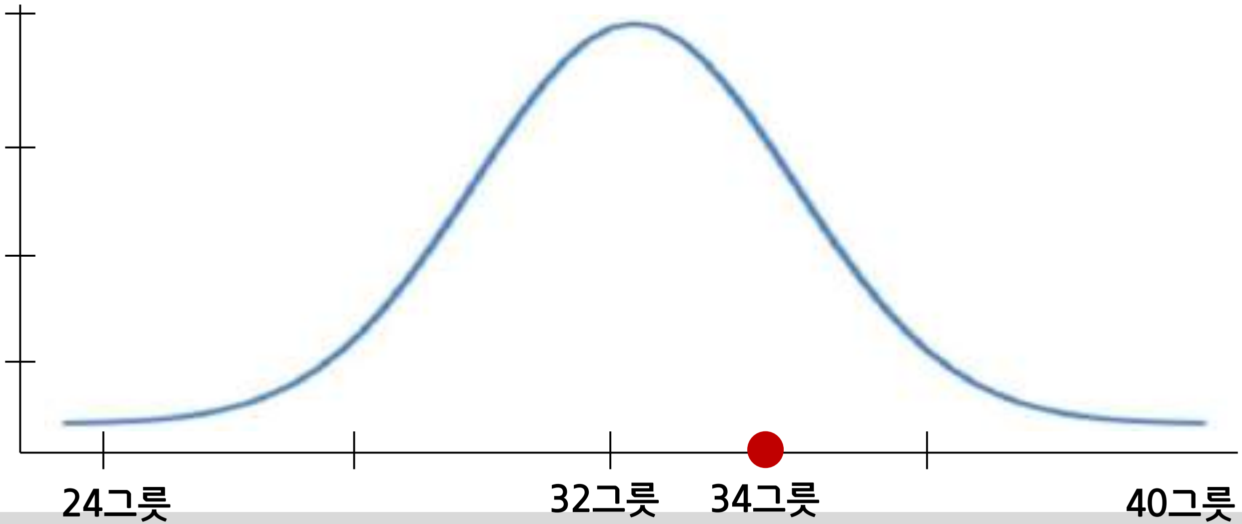


From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Likelihood

먼저 체인점의 하루 평균 비냉판매량을 측정함 : 34그릇이 나왔다고 가정

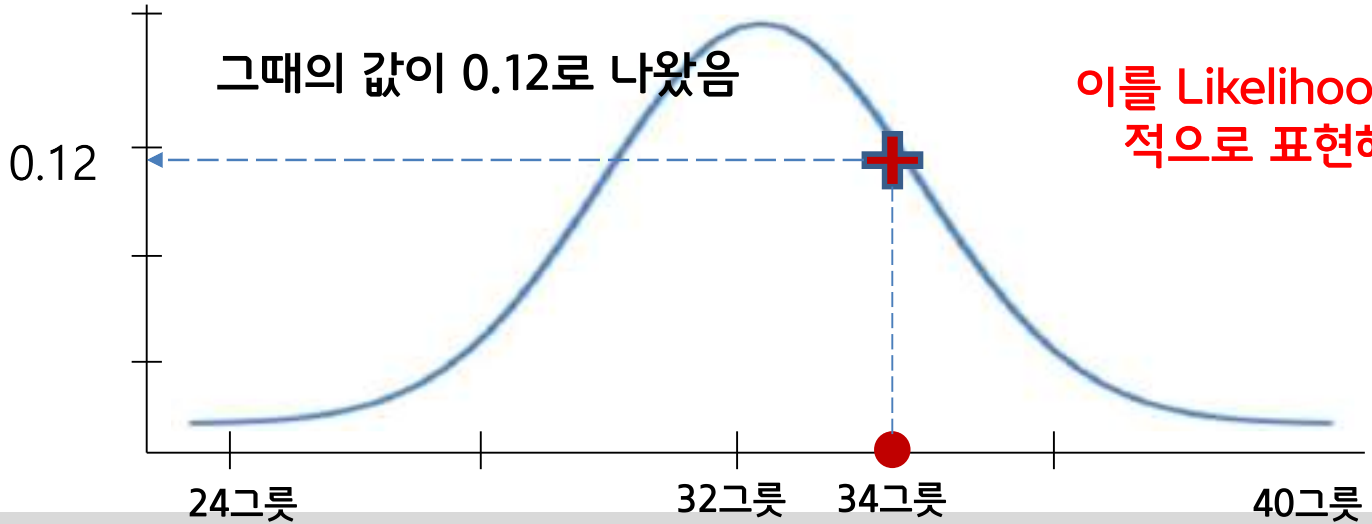


From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Likelihood

먼저 체인점의 하루 평균 비냉판매량을 측정함 : 34그릇이 나왔다고 가정



이를 Likelihood를 수학적으로 표현해 보자

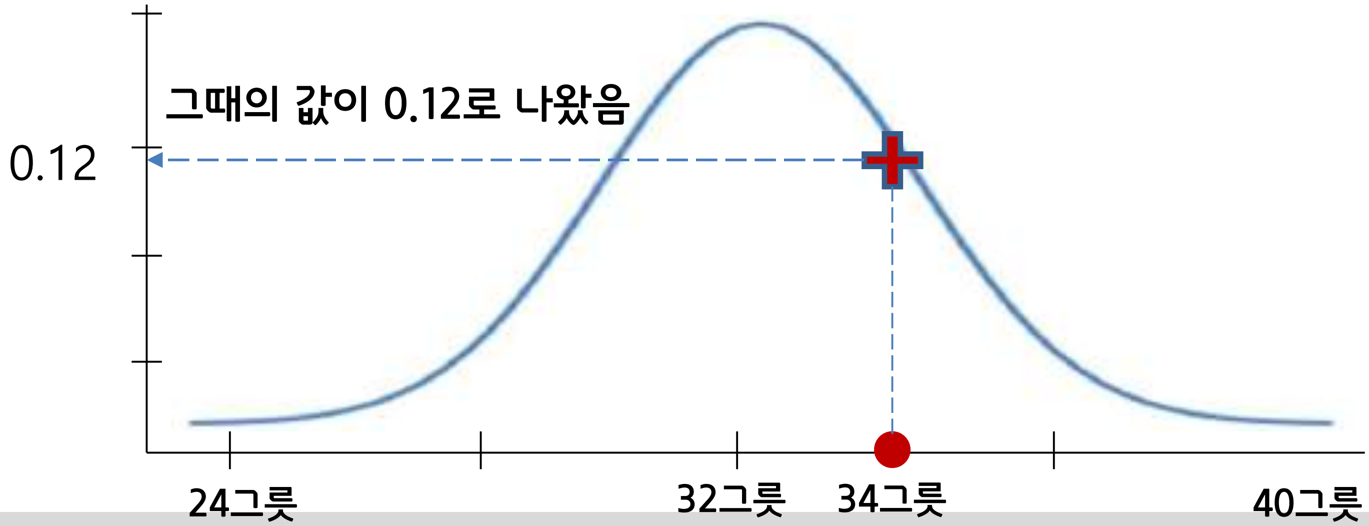
Probability vs Likelihood

Likelihood

분포가아닌 판매 된 값. 즉 x 가 주어짐

$$L(\boldsymbol{mean} = \mathbf{32}\ \boldsymbol{and}\ \boldsymbol{deviation} = \mathbf{2.5} \mid \text{비냉판매 수 34 그릇})$$

비냉이 34그릇 팔렸을

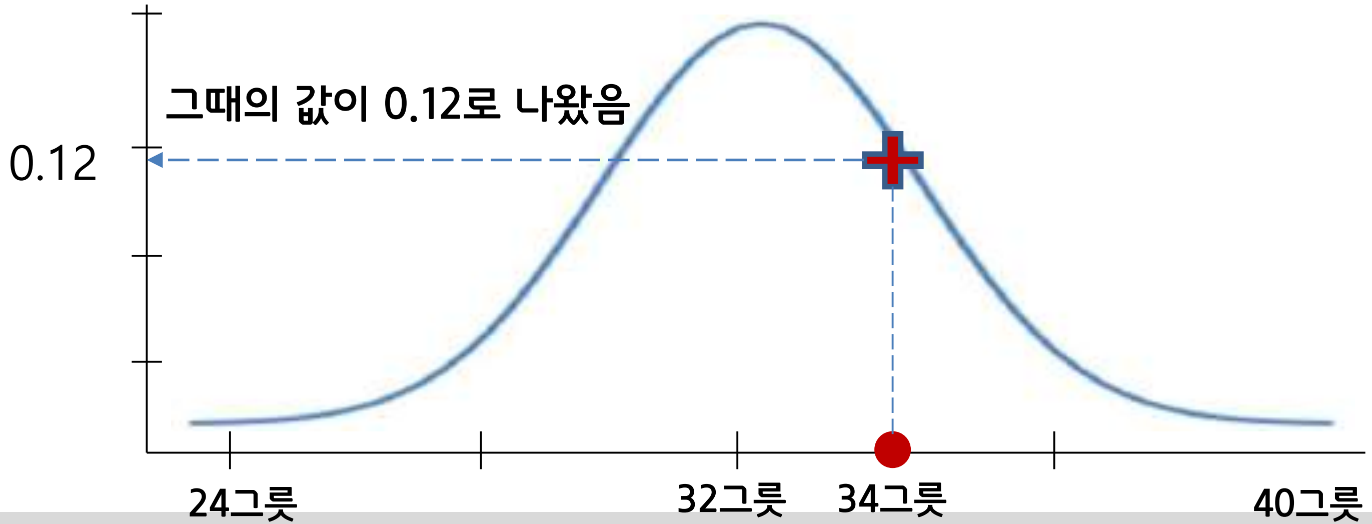


Probability vs Likelihood

Likelihood

$$L(\boldsymbol{mean} = \mathbf{32}\ \boldsymbol{and\ deviation} = \mathbf{2.5}\ |\ \text{비냉판매 수 } 34 \text{ 그릇})$$

비냉이 34그릇 팔렸을 **때**



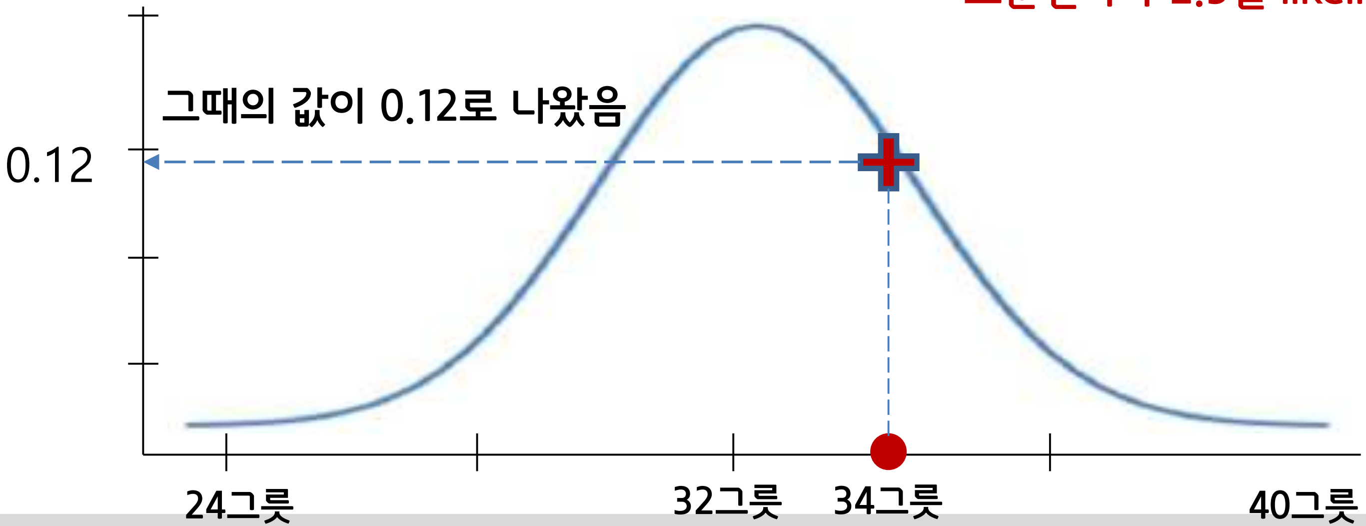
From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Likelihood

$L(mean = 32\ and\ deviation = 2.5 \mid$ 비냉판매 수 34 그릇$)$

비냉이 34그릇 팔렸을 때, 평균이 32 표준편차가 2.5일 likelihood는 0.12임.



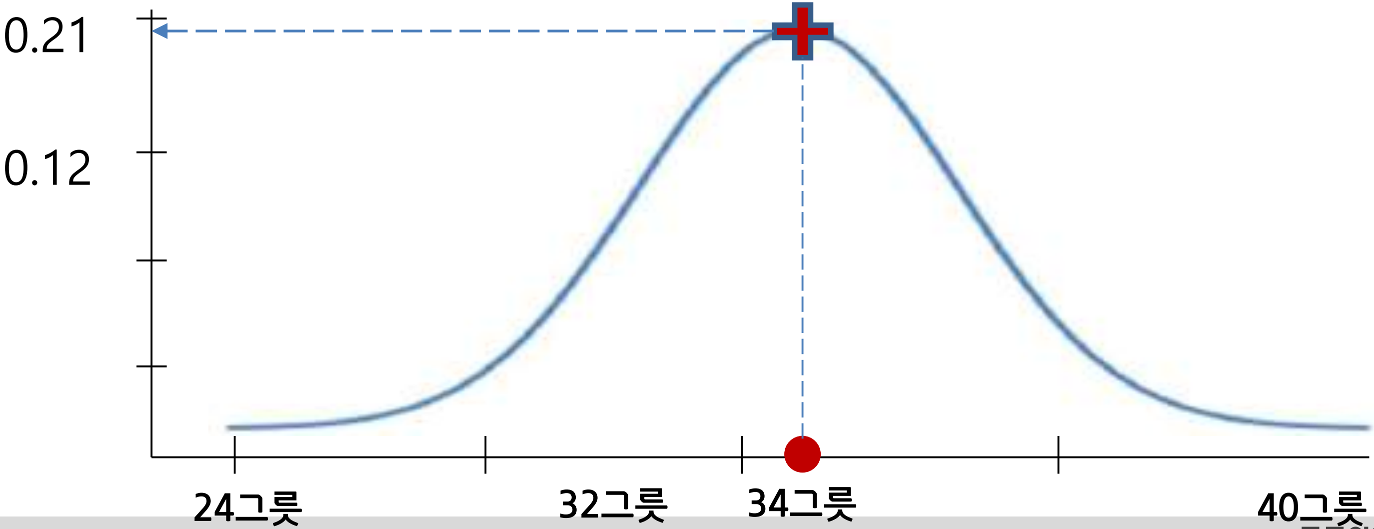
From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Likelihood

$$L(\mathbf{mean} = \mathbf{34}\ \boldsymbol{and\ deviation} = \mathbf{2.5} \mid \text{비냉판매 수 } 34 \text{ 그릇})$$

평균이 34로 변경 될 경우의 likelihood는 0.21로 증가 함



From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

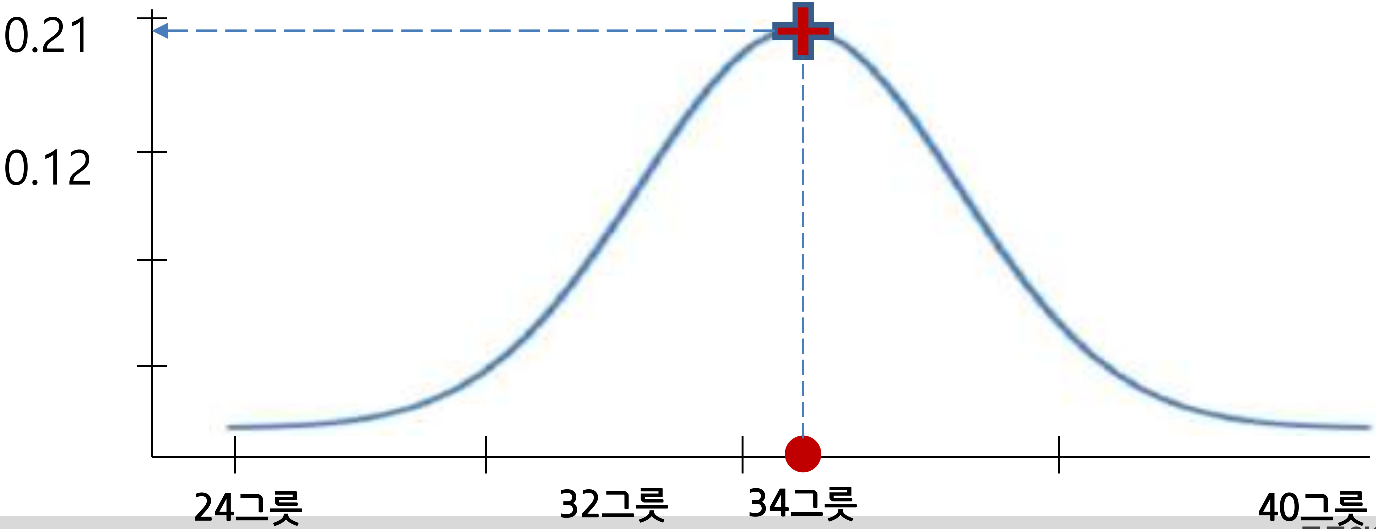
Probability vs Likelihood

Likelihood

Measurement(측정)는 고정

$$L(\mathbf{\mathit{mean} = 34\ \mathit{and}\ \mathit{deviation} = 2.5} \mid \text{비냉판매 수 } 34 \text{ 그릇})$$

Distribution을 바꾸면 likelihood가 변경 됨



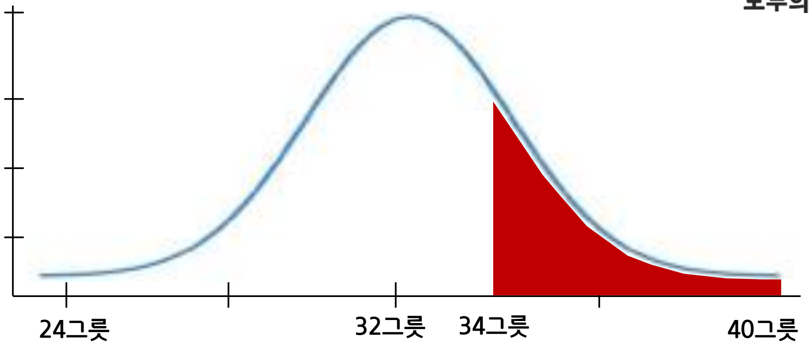
From: StatQuest : Probability vs Lokelihood (https://www.youtube.com/watch?v=pYxNSUDSFH4)

Probability vs Likelihood

Probability : 고정된 분포에서의 우리가 원하는 영역의 면적

P(data | distribution)

Probability of data given distribution

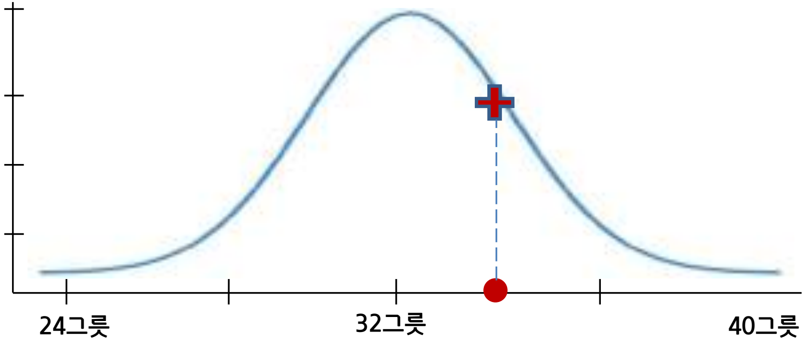


Likelihood : 고정된 데이터 포인트에서 변형 가능한 분포를 이용해 측정한 Y축 값

L(distribution | data)

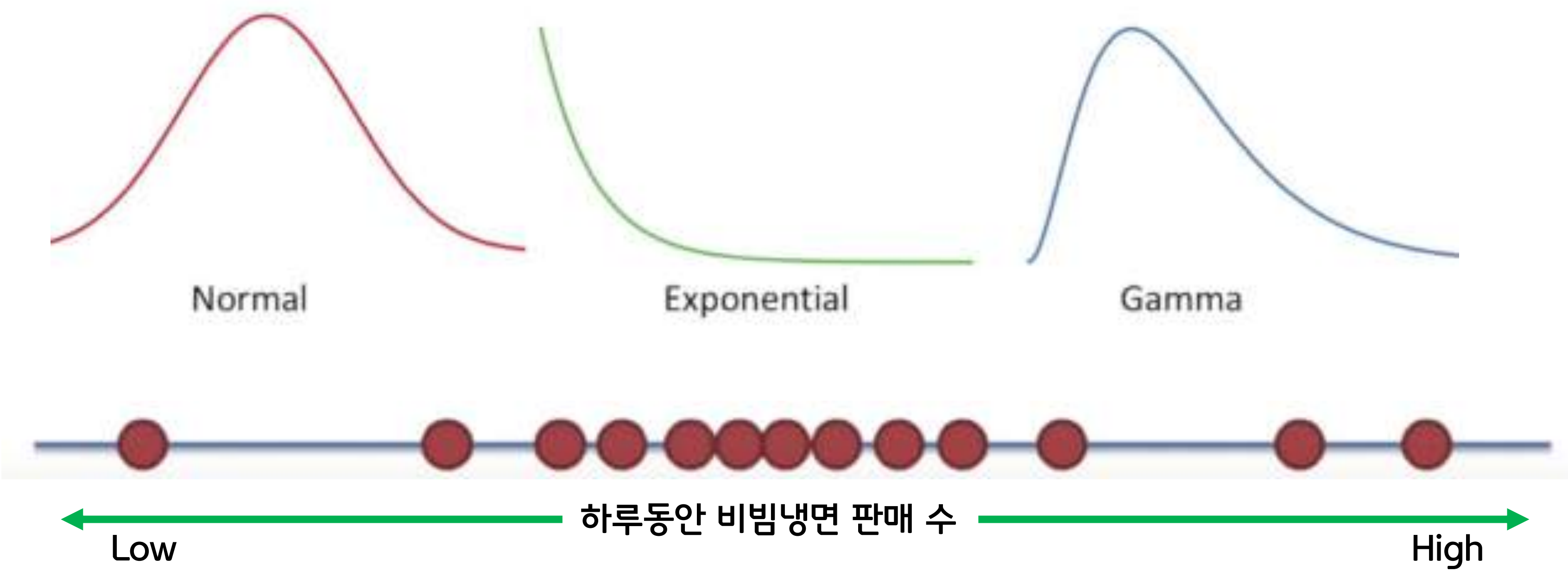
[변수]

Likelihood of distribution given data



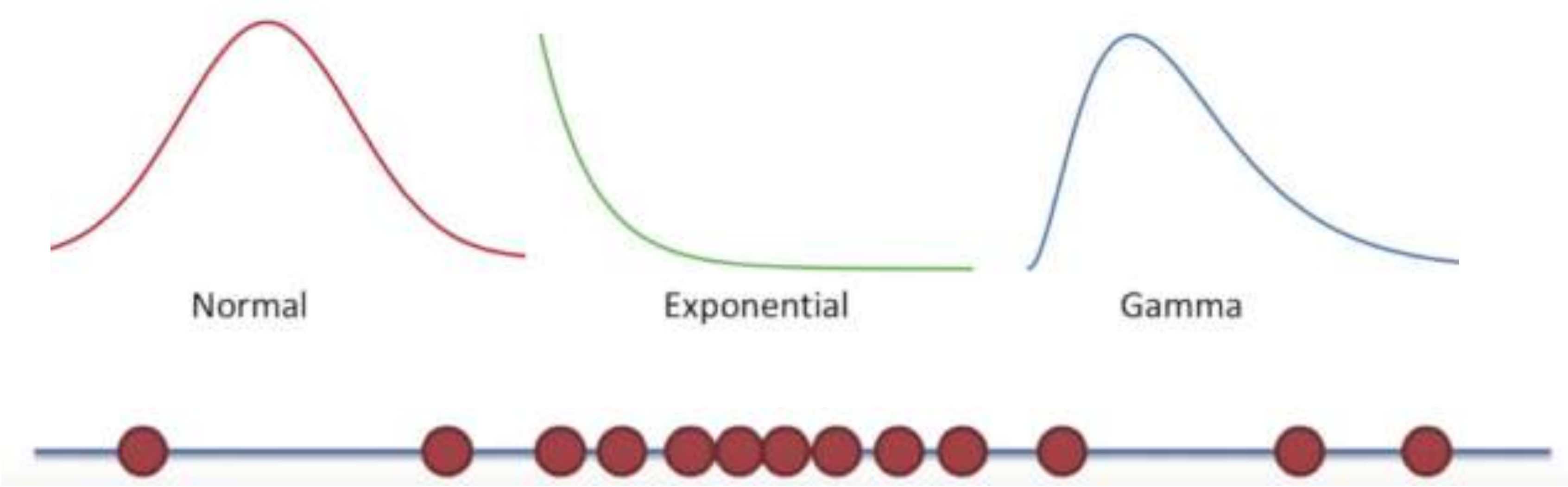
Maximum Likelihood

Maximum likelihood의 목표는 데이터를 가장 잘 표현하는 분포를 찾는 것이다



Maximum Likelihood

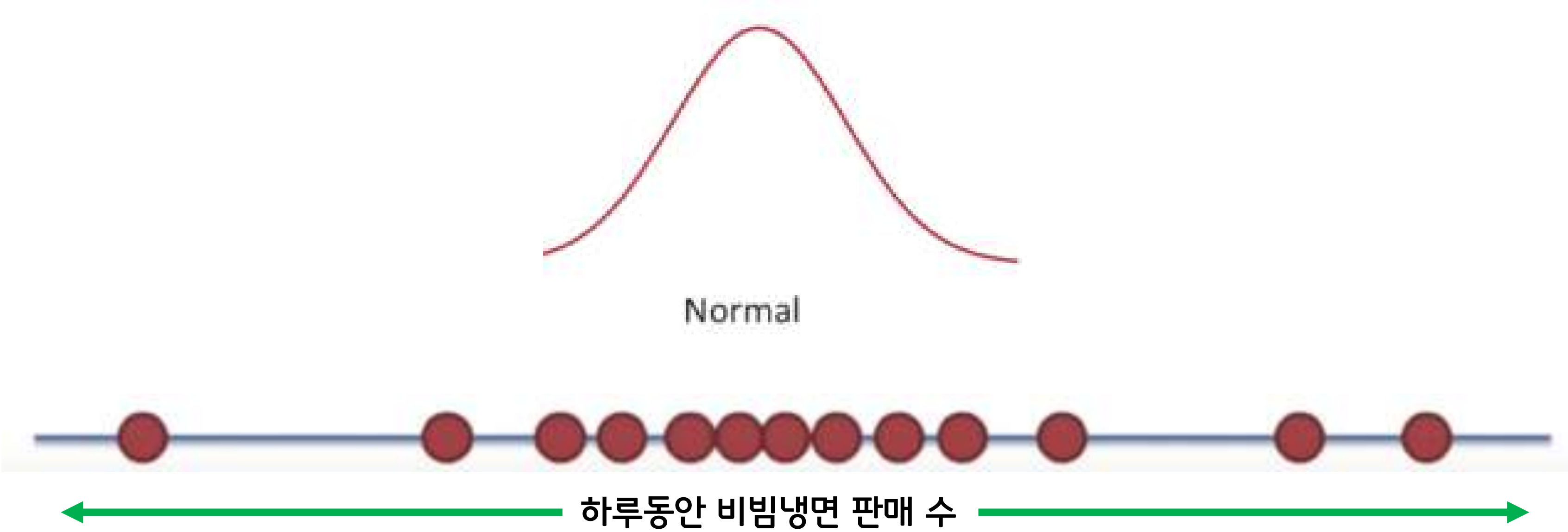
분포를 찾는 이유 : 다루기 쉬워지고 일반화 될 수 있다



하루동안 비빔냉면 판매 수

Maximum Likelihood

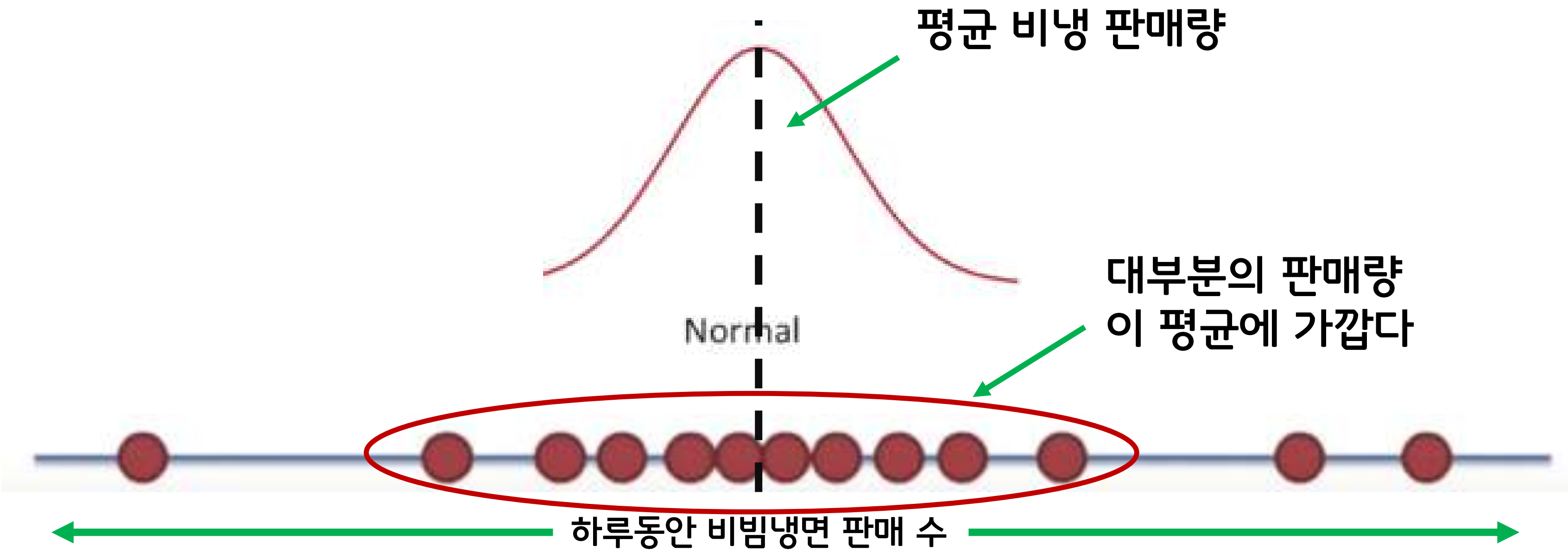
하루 동안의 비빔냉면 판매 수가 normal distribution이라 가정하면



Maximum Likelihood

Normally distributed의 의미 :

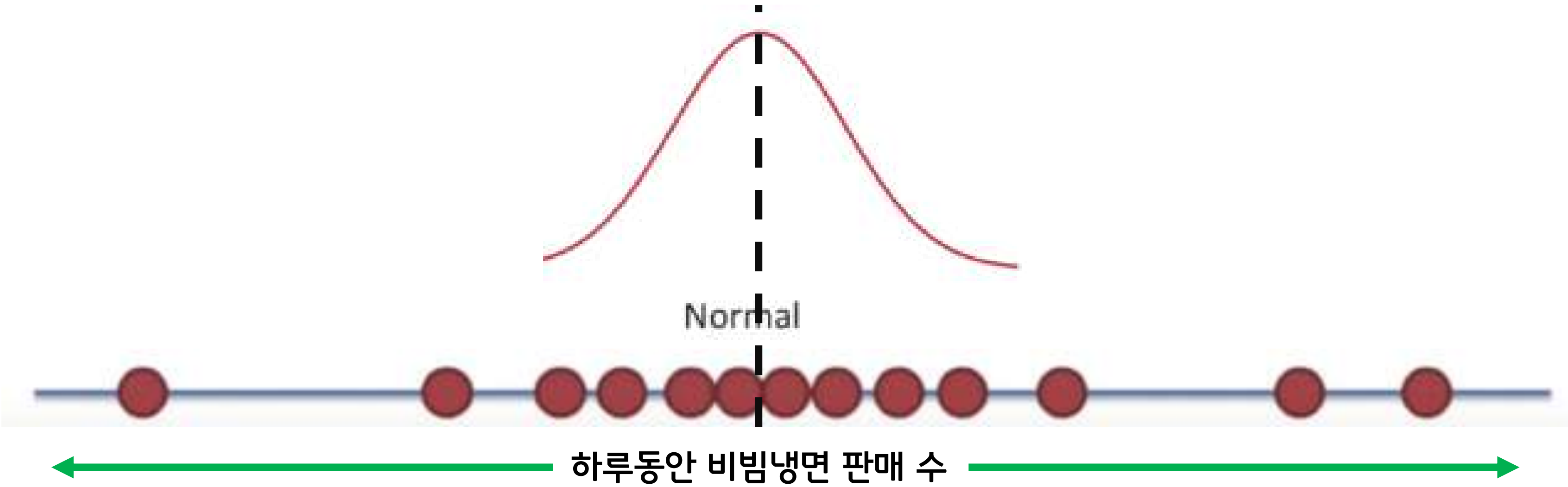
1) 대부분의 측정이 평균(mean)에 가깝다



Maximum Likelihood

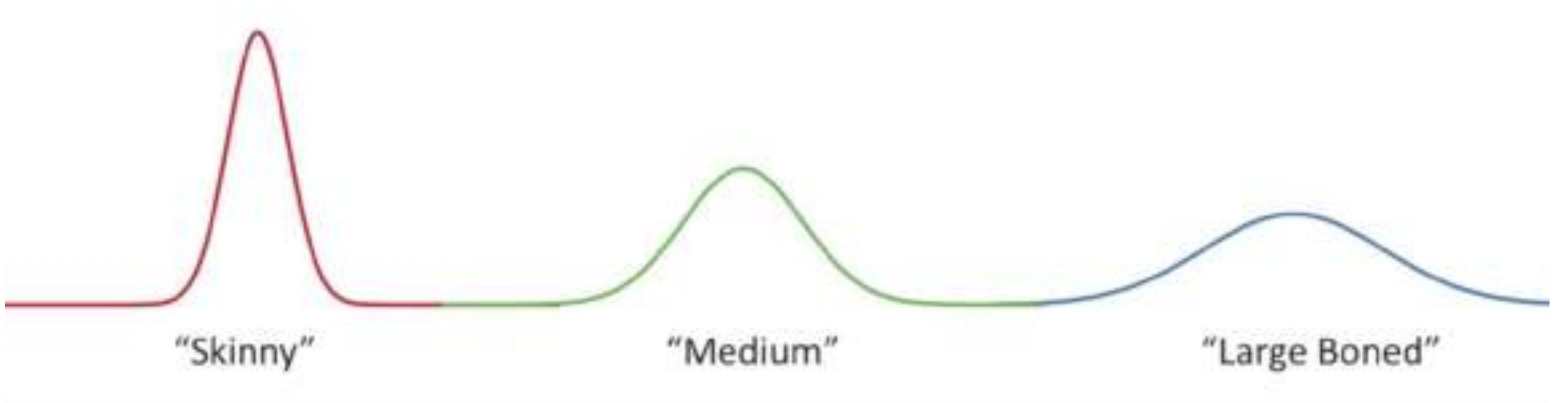
Normally distributed의 의미 :

1) 대부분의 측정이 평균(mean)에 가깝다
2) 측정결과가 평균을 중심으로 대략 대칭형이다



Maximum Likelihood

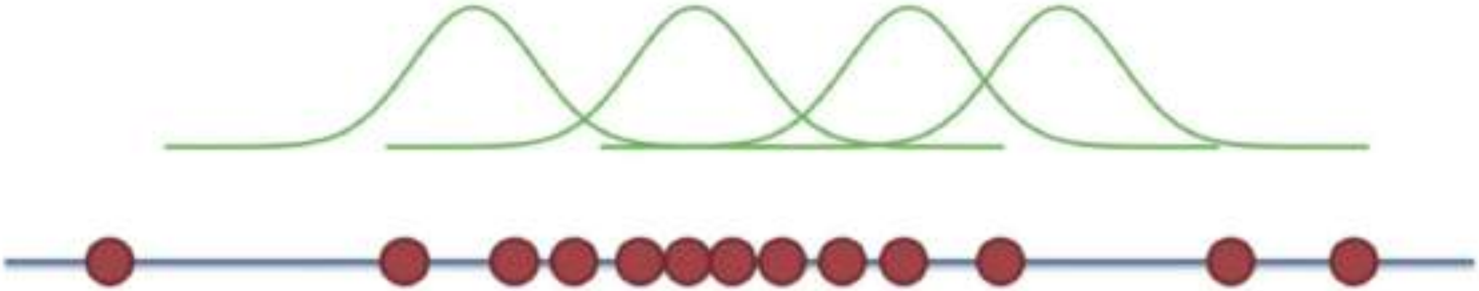
Normally distribution은 여러가지 형태를 가질 수 있습니다



From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

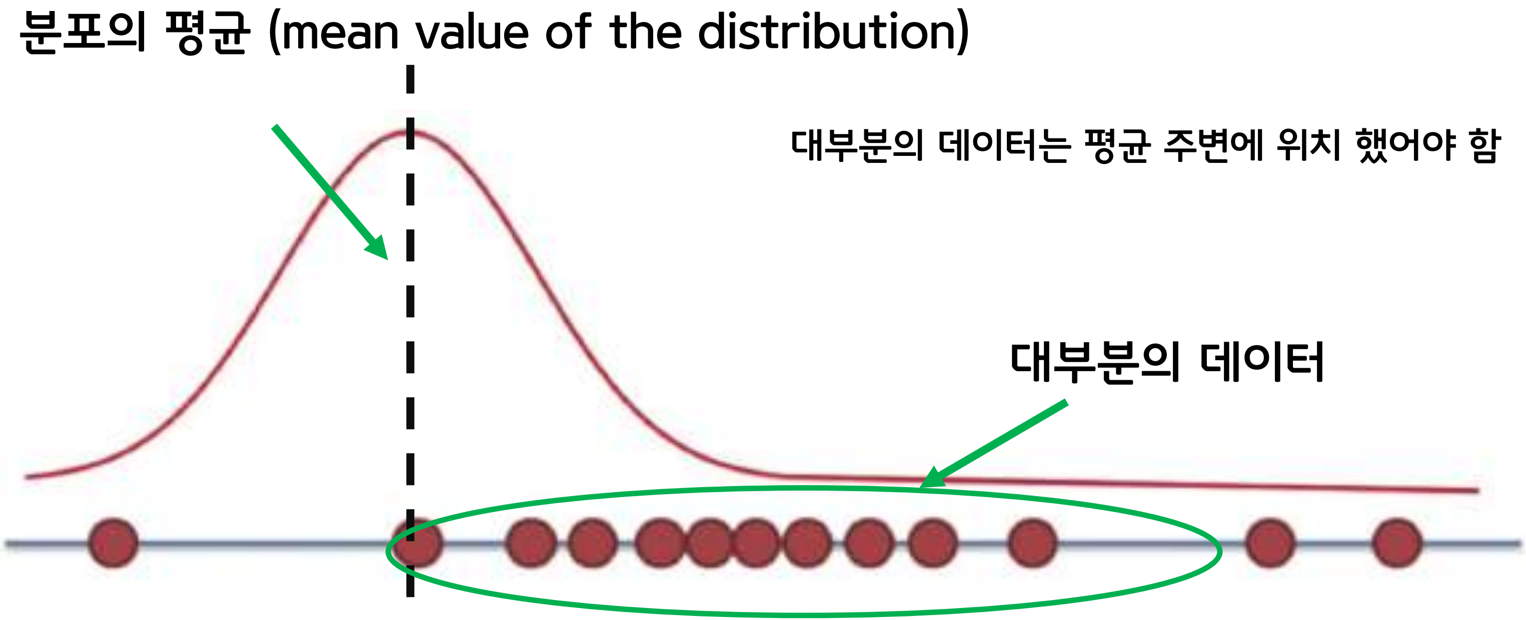
Maximum Likelihood

형태를 찾았으면 이 분포를 어디에다 위치시킬지 결정해야 합니다



From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

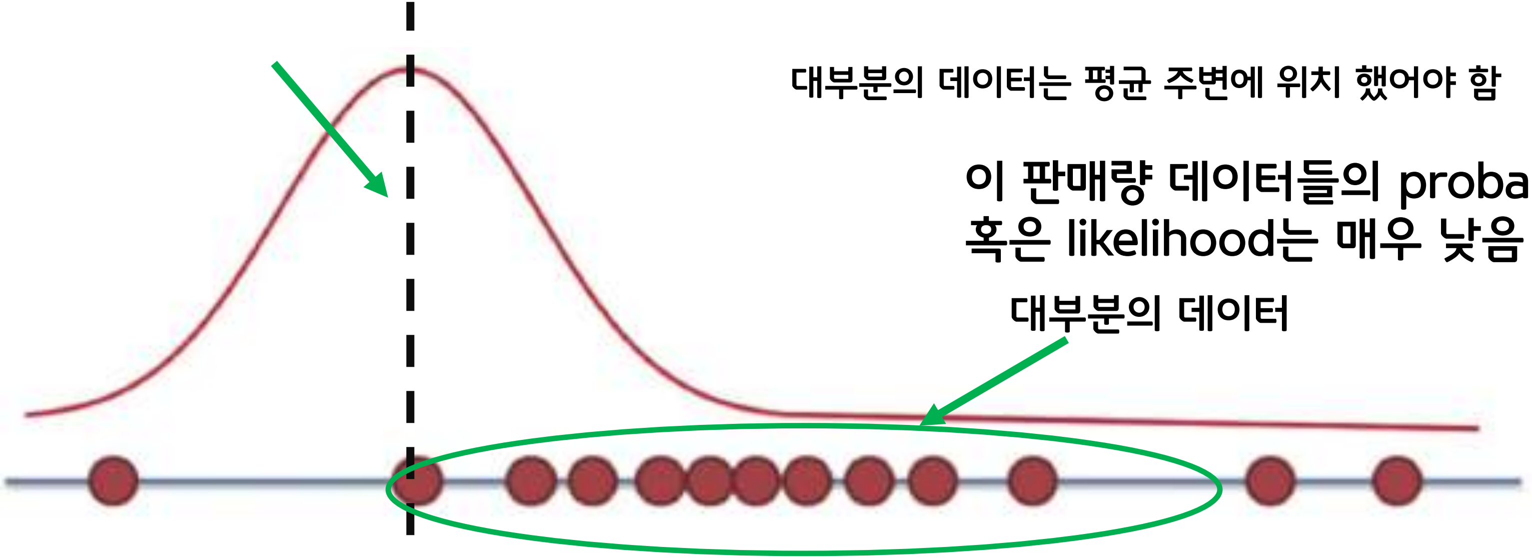
Maximum Likelihood



Maximum Likelihood

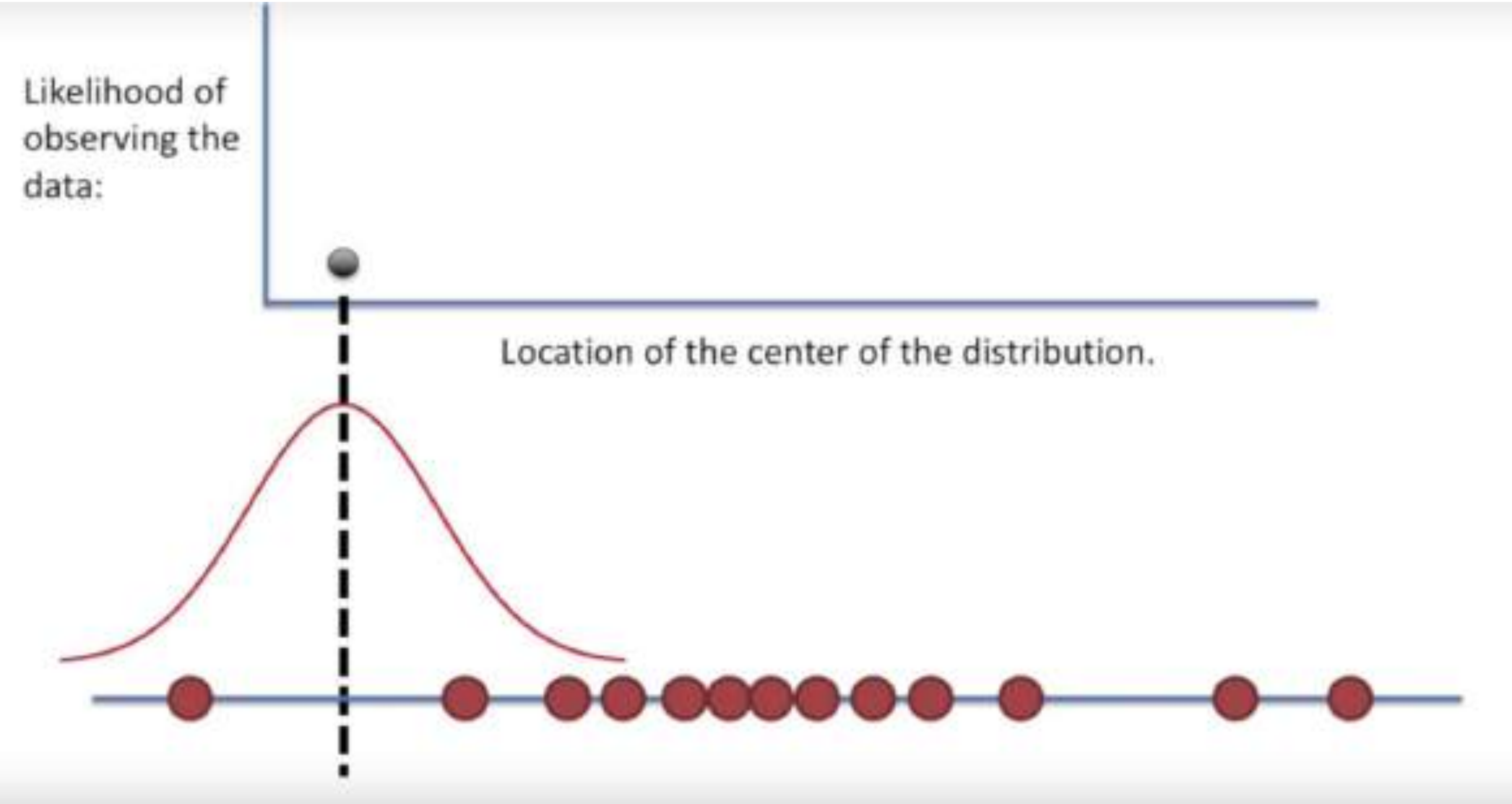
모두의연구소

분포의 평균 (mean value of the distribution)



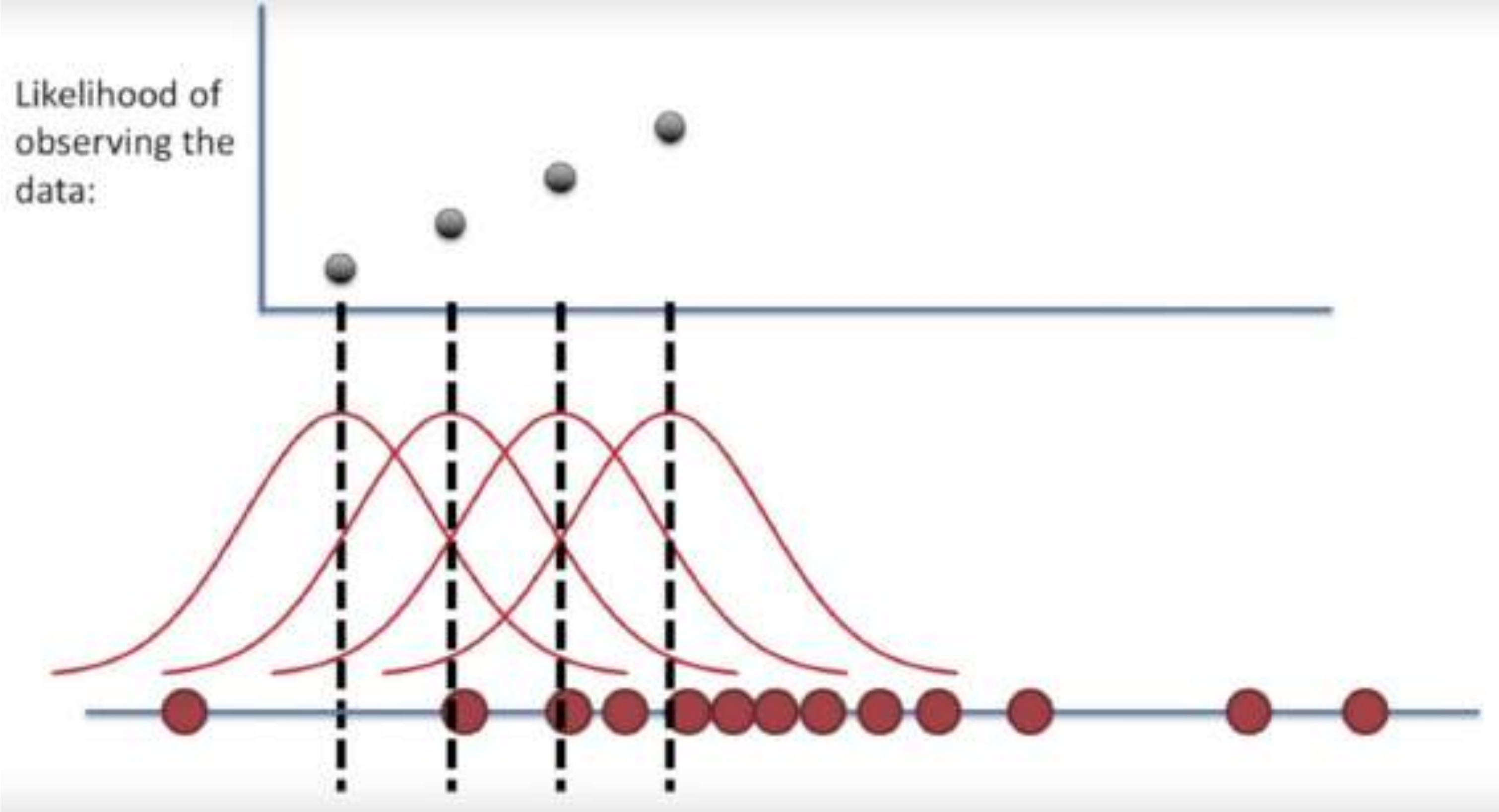
From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood



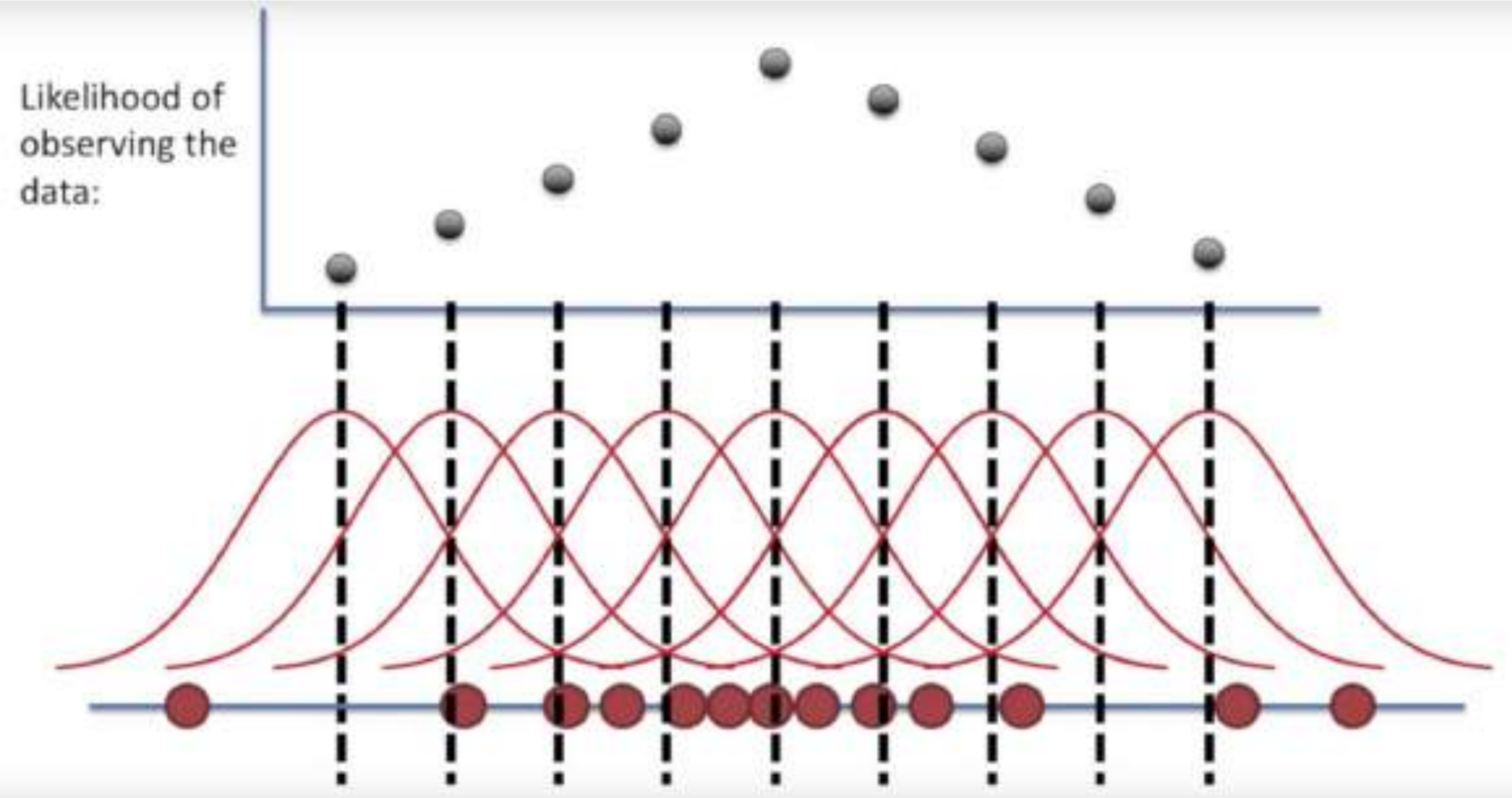
From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood



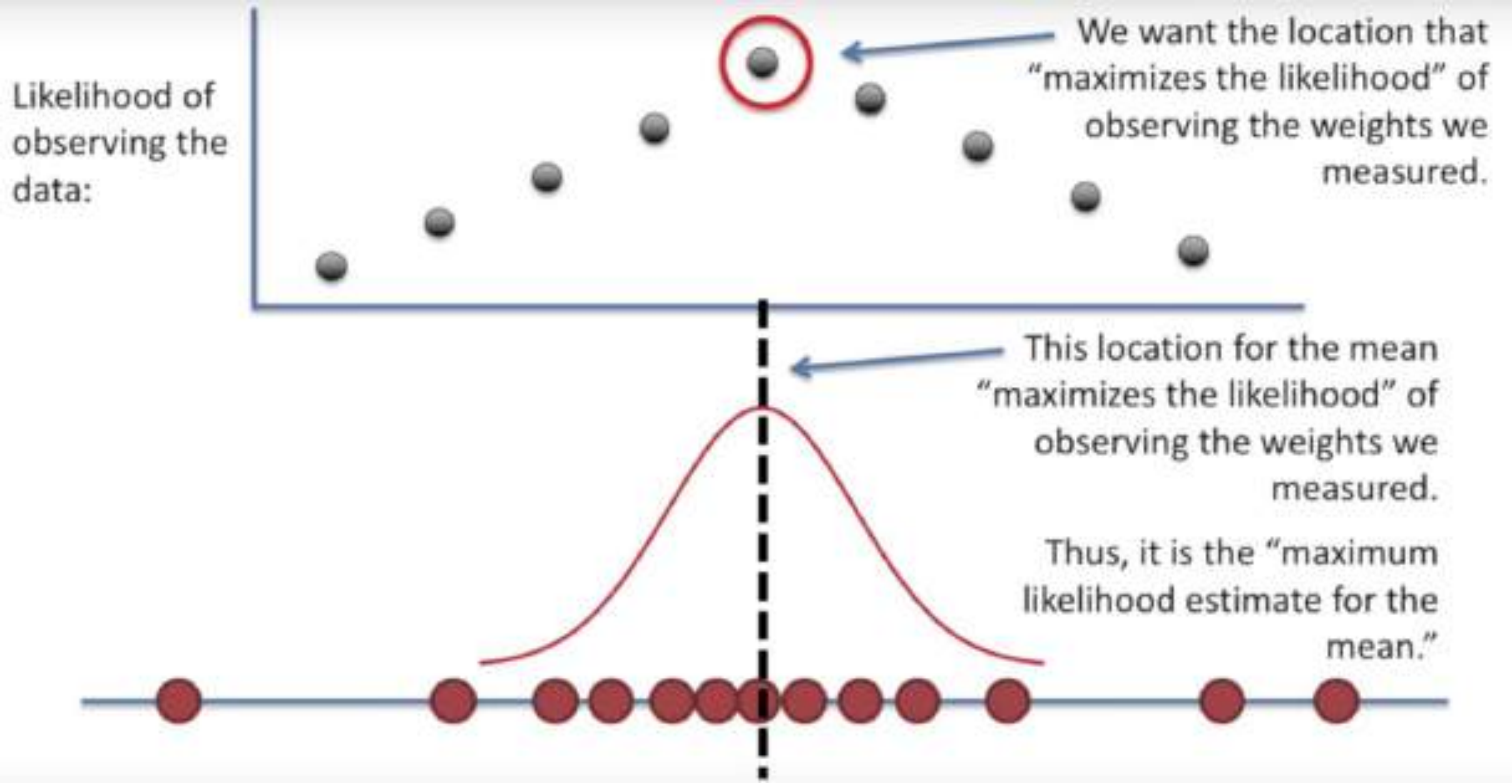
From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood



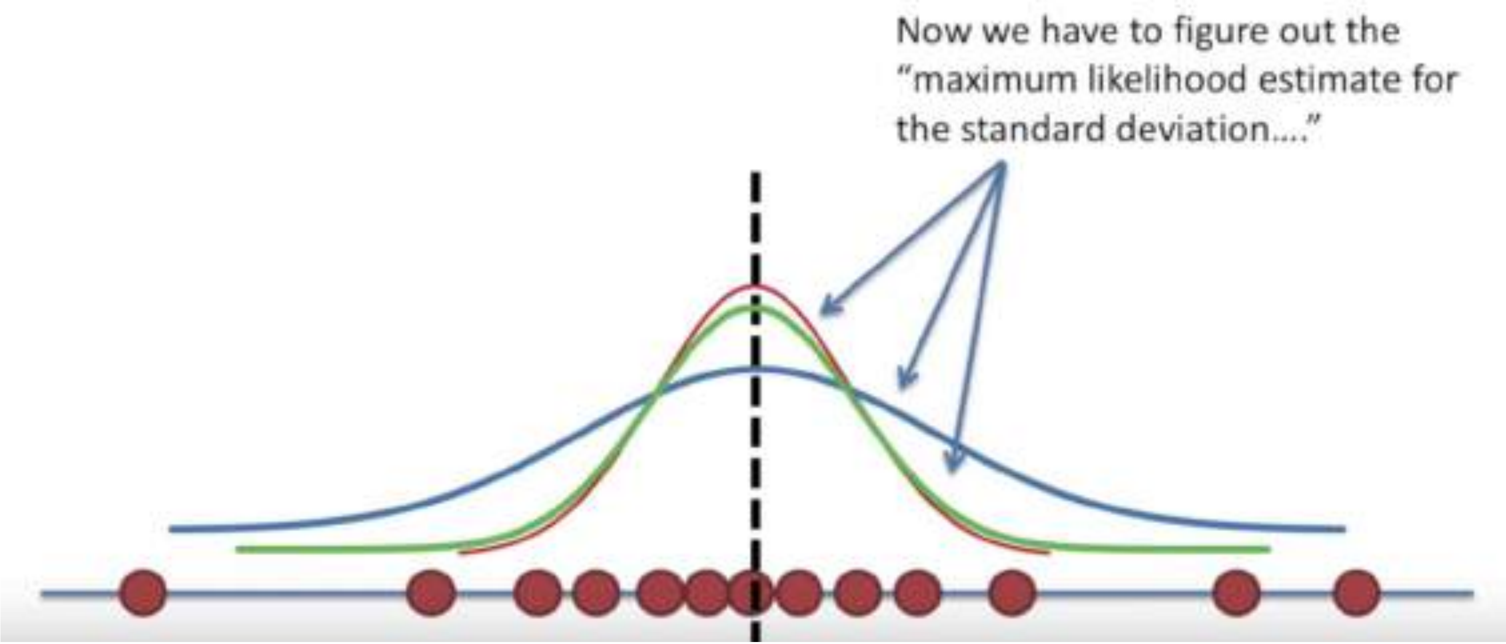
From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood



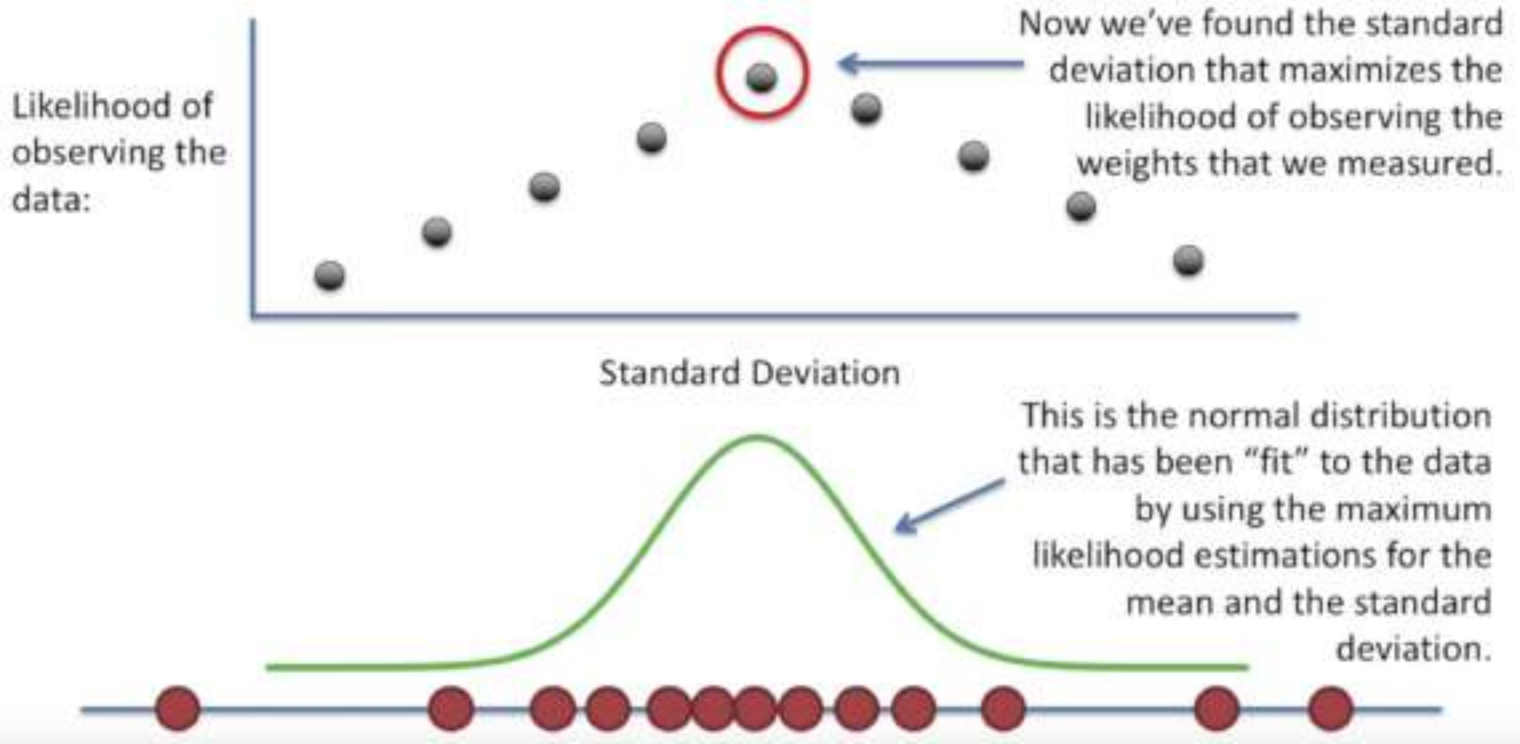
From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood

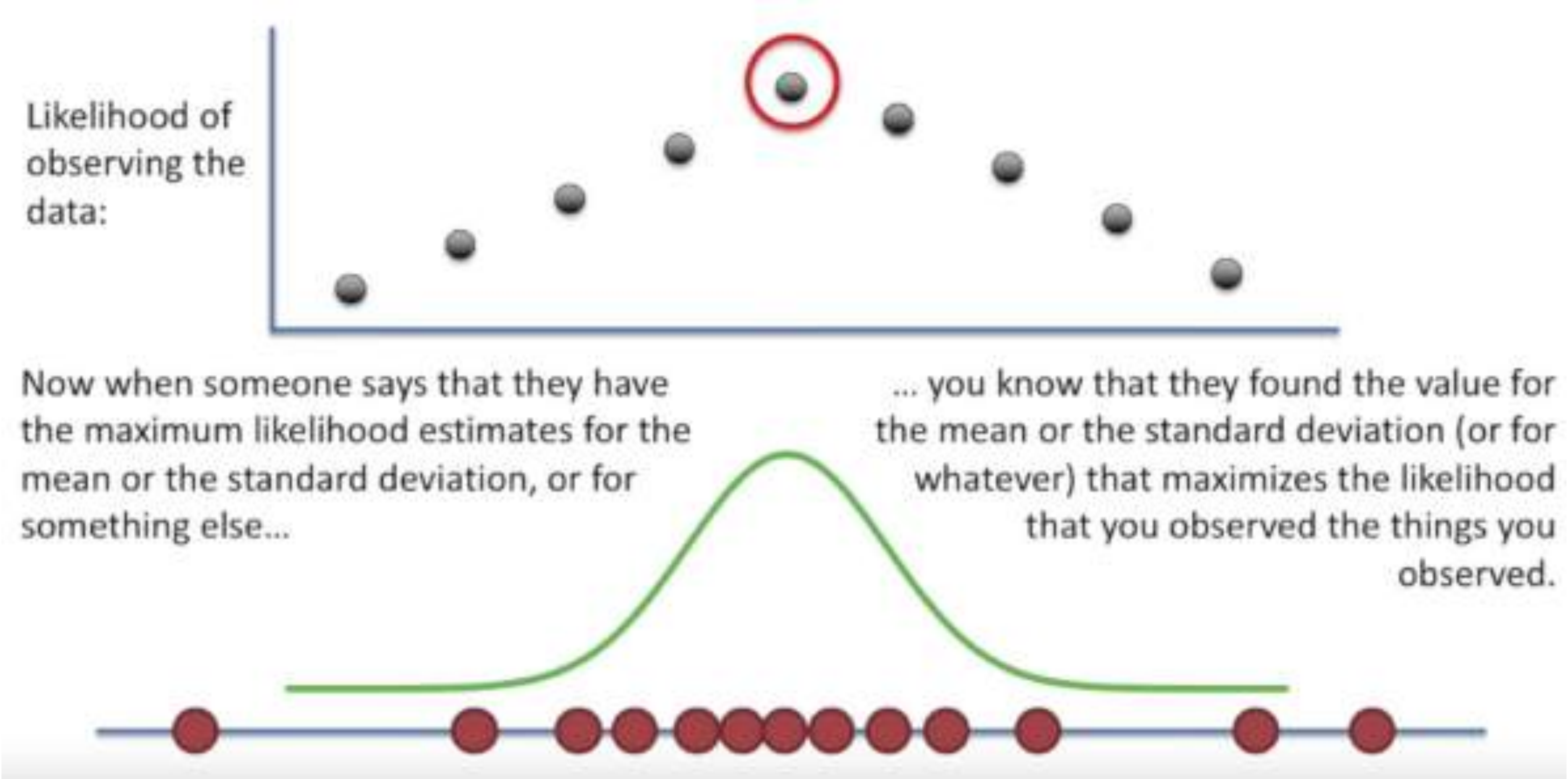


From: StatQuest: Maximum Likelihood, clearly explained!!! (https://youtu.be/XepXtl9YKwc)

Maximum Likelihood



Maximum Likelihood



Maximum Likelihood

Terminology alert!

In everyday conversation, “probability” and “likelihood” mean the same thing. However, in Stats-Land, “likelihood” specifically refers to this situation we’ve covered here; where you are trying to find the optimal value for the mean or standard deviation for a distribution given a bunch of observed measurements.

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

모두의연구소

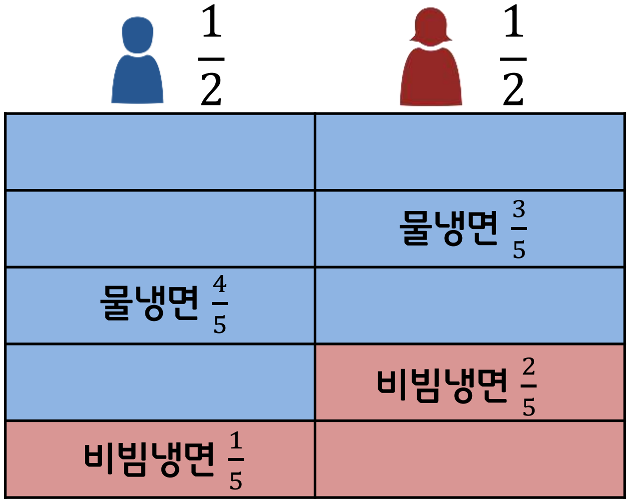
- 냉면 맛집 "딥러닝 면옥"의 실태를 조사해 봄 (백종원)
  - 손님의 비율은 남자 $\frac{1}{2}$, 여자 $\frac{1}{2}$ (즉, 둘이 같음)
  - 남자 손님의 경우 $\frac{4}{5}$ 가 물냉면 , 나머지 $\frac{1}{5}$이 비빔냉면

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

물냉과 비냉으로 알아보는 베이지안

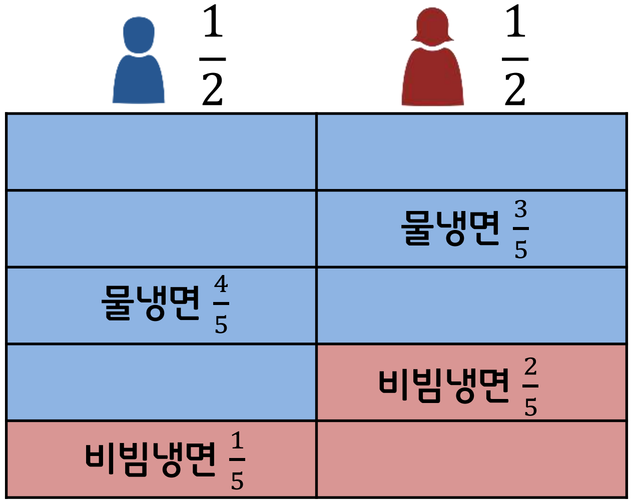
- 냉면 맛집 "딥러닝 면옥"의 실태를 조사해 봄 (백종원)

  - 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

$\frac{1}{2}$	$\frac{1}{2}$
	물냉면 $\frac{3}{5}$
물냉면 $\frac{4}{5}$	
	비빔냉면 $\frac{2}{5}$
비빔냉면 $\frac{1}{5}$	

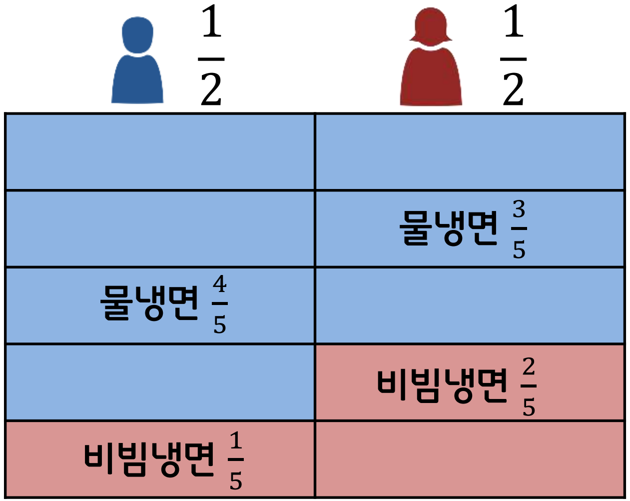
확률은 차지하는 영역의 크기

물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

  - 남자 $\frac{1}{10}$ vs. 여자 $\frac{2}{10}$

따라서 여자~!!



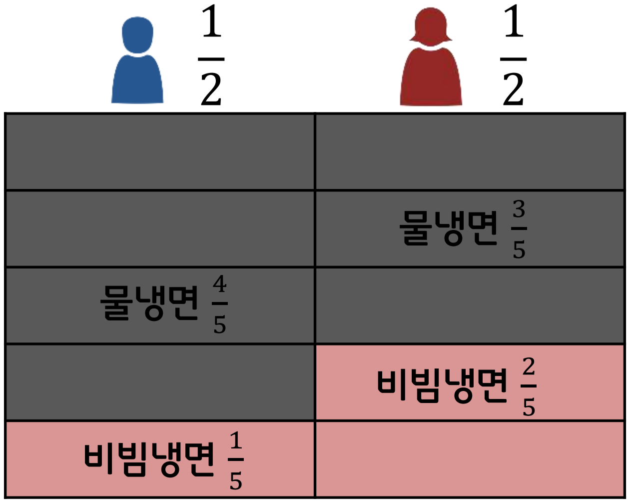
확률은 차지하는 영역의 크기

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

모두의연구소

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)

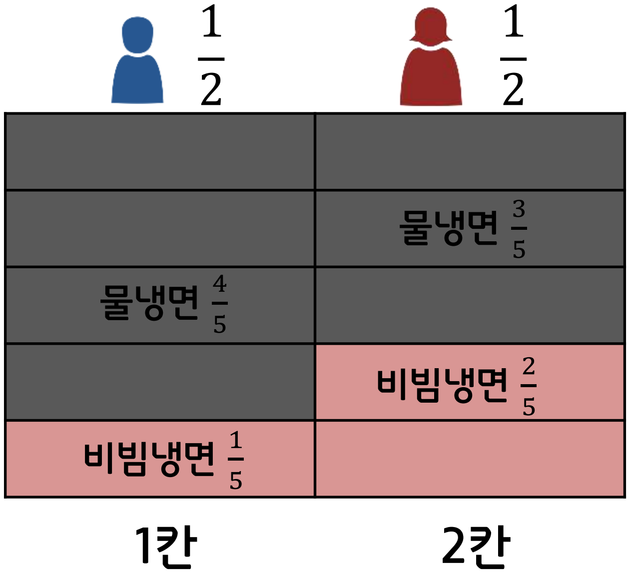
Normalization

- 1) 비빔냉면 먹고 있었으니 물냉면을 볼 필요는 없고
- 2) 비빔냉면 먹는 사람중에서는 $\frac{2}{3}$의 확률로 여자일 가능성이 높겠군요

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

여자일 가능성이 높다~!!

합하면 1, 확률로 표현 가능하군요



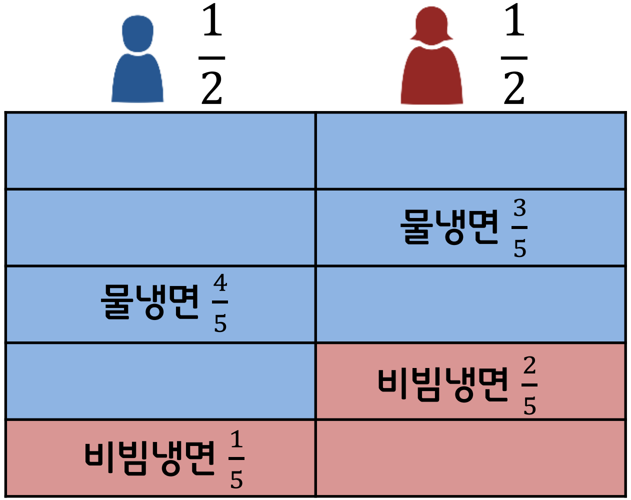
물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 "딥러닝 면옥"의 실태를 조사해 봄 (백종원)
  - 베이지안을 배운 백종원

우리가 그림보고 얻은 결과 비냉 먹는 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~

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



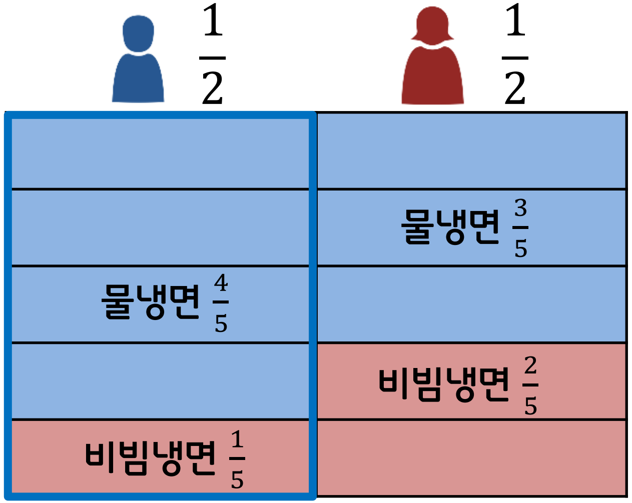
물냉과 비냉으로 알아보는 베이지안

- 냉면 맛집 "딥러닝 면옥"의 실태를 조사해 봄 (백종원)
  - 베이지안을 배운 백종원

우리가 그림보고 얻은 결과 비냉 먹는 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~

$$P(Y=\text{남자}|X=\text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{남자})\,\overset{\frac{1}{2}}{\mathbf{P(Y=\text{남자})}}}{P(X=\text{비냉})}$$



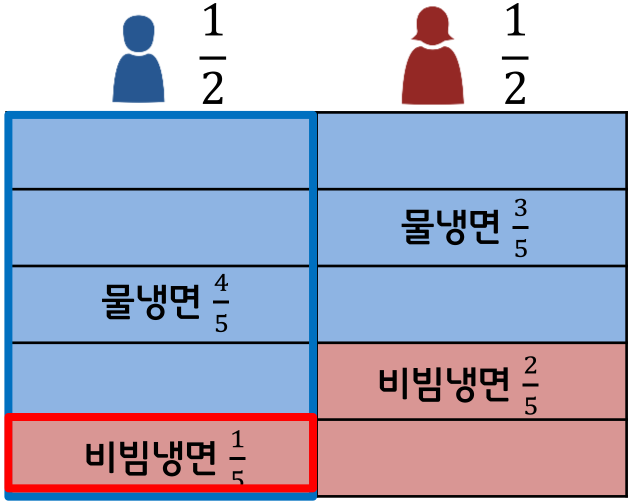
물냉과 비냉으로 알아보는 베이지안

모두의연구소

- 냉면 맛집 “딥러닝 면옥”의 실태를 조사해 봄 (백종원)
  - 베이지안을 배운 백종원

우리가 그림보고 얻은 결과 비냉 먹는 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~



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

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

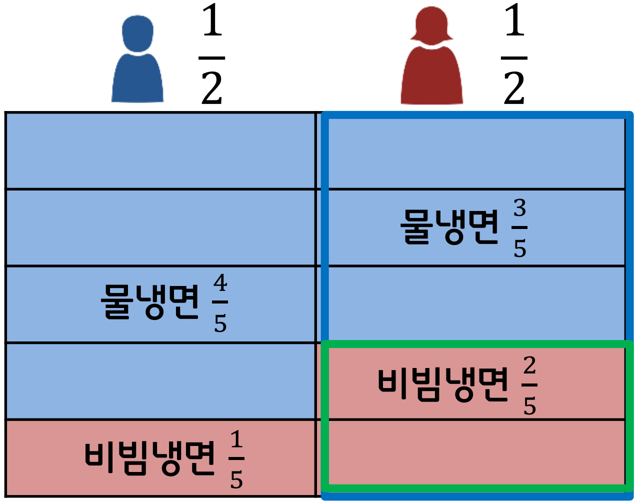
$\frac{1}{10}$은 전체공간에서 남자가 비빔냉면 먹은 확률임

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

우리가 그림보고 얻은 결과 비냉 먹는 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~

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

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

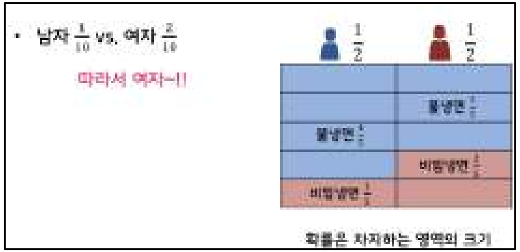
$\frac{1}{10}$은 전체공간에서 남자가 비빔냉면 먹은 확률임

$\frac{2}{10}$는 전체공간에서 여자가 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)



Unnormalized probability

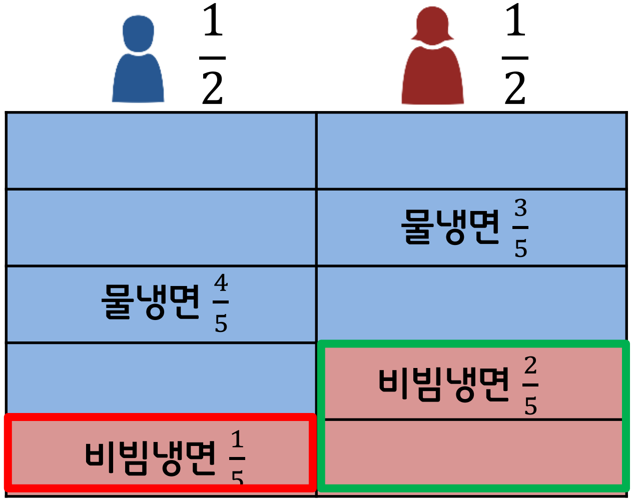
분모를 계산하지 않아도 누구일 확률이 높은지 맞추는 것은 가능

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

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

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

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

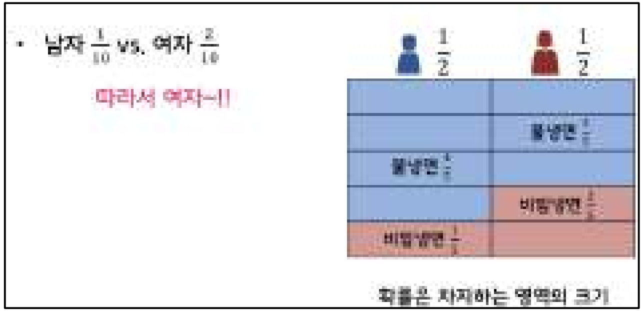


$\frac{1}{10}$은 전체공간에서 남자가 비빔냉면 먹은 확률임

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

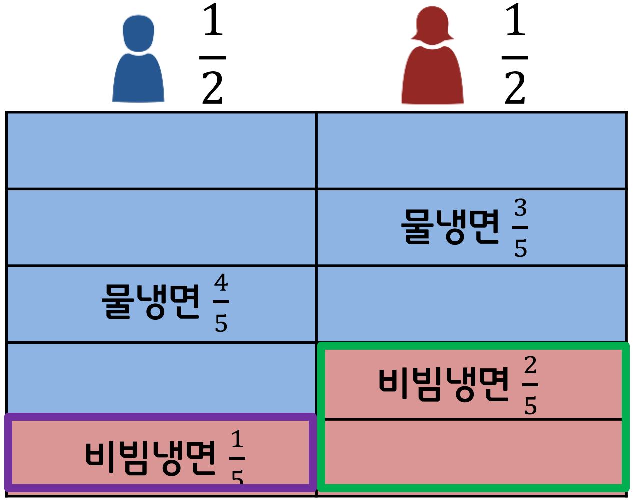
물냉과 비냉으로 알아보는 베이지안

- 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

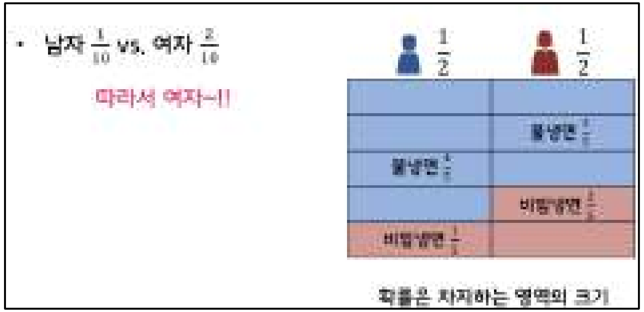


$$P(Y = \text{남자}|X = \text{비냉}) = \frac{P(X=\text{비냉}|Y=\text{남자})\, P(Y=\text{남자})}{P(X=\text{비냉})}$$

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

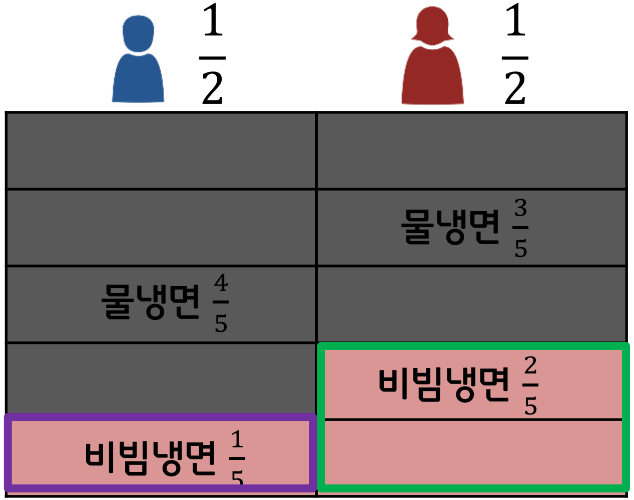
물냉과 비냉으로 알아보는 베이지안

- 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

Logic Function : 2-class classification

- 분류문제는 사후(posterior)확률 을 기준으로 삼는다

$$X : Y_1 \text{ if } P(Y_1|X) > P(Y_2|X)$$ 남자

$$X : Y_2 \text{ if } P(Y_1|X) < P(Y_2|X)$$ 여자

'X = 비냉 ' 을 먹고 있는데 그게 'Y_1 = 여자'일 확률($P(Y_1|X)$)이 높으면 여자, 아니면 남자로 분류한다

Logic Function : 2-class classification

모두의연구소

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

$$P(Y_2|X) = \frac{P(X|Y_2)P(Y_2)}{P(X)}$$

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

Logic Function : 2-class classification

- 분류문제는 사후(posterior)확률 을 기준으로 삼는다

$$X : Y_1 \text{ if } P(Y_1|X) > P(Y_2|X)$$

$$X : Y_2 \text{ if } P(Y_1|X) < P(Y_2|X)$$

'X = 비냉 ' 을 먹고 있는데 그게 'Y_1 = 여자'일 확률($P(Y_1|X)$)이 높으면 여자, 아니면 남자로 분류한다

Logic Function : 2-class classification

모두의연구소

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

$$P(Y_2|X) = \frac{P(X|Y_2)P(Y_2)}{P(X)}$$

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

Logic Function : 2-class classification

posterior　likelihood　prior

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

$$P(Y_2|X) = \frac{P(X|Y_2)P(Y_2)}{P(X)}$$

분모는 같다, 즉, 분자인 likelihood와 prior의 곱이 posterior를 결정

즉, 분자인 각각 성별 비냉을 먹을 확률($P(X|Y)$)과 그 성비($P(Y)$)의 곱이 비냉을 먹고 있는 사람의 성별 ($P(Y|X)$) 을 결정

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

Likelihood와 Prior의 곱이 중요

Logic Function : 2-class classification

posterior　　likelihood　　prior

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X)}$$

$$P(Y_2|X) = \frac{P(X|Y_2)P(Y_2)}{P(X)}$$

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

Gausean 등으로 가정 함.

왜 ln을?

- 단조 증가함수로 극점을 변화시키지 않음
- 각 시행이 독립일 경우 곱이 덧셈으로 바뀌어 계산이 용이
- 확률분포가 Gaussian 등의 Exponential family일 경우 계산이 용이

참고 : Bayes Theorem과 Sigmoid와 Softmax사이의 관계

Logic Function : 2-class classification

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{\boldsymbol{P(X|Y_1)P(Y_1)}}{\boldsymbol{P(X|Y_1)P(Y_1)} + \boldsymbol{P(X|Y_2)P(Y_2)}}$$

$$= \frac{\boldsymbol{e^{s_1}}}{\boldsymbol{e^{s_1}} + \boldsymbol{e^{s_2}}}$$

Sigmoid가 유도됩니다

$$\boldsymbol{P(X)} = \boldsymbol{P(X|Y_1)P(Y_1)} + \boldsymbol{P(X|Y_2)P(Y_2)}$$

참고 : Bayes Theorem과 Sigmoid와 Softmax사이의 관계

Logic Function : 2-class classification

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{\boldsymbol{P(X|Y_1)P(Y_1)}}{\boldsymbol{P(X|Y_1)P(Y_1)} + \boldsymbol{P(X|Y_2)P(Y_2)}}$$

Sigmoid가 유도됩니다

$$= \frac{\boldsymbol{e^{s_1}}}{\boldsymbol{e^{s_1}} + \boldsymbol{e^{s_2}}} = \frac{1}{1 + \boldsymbol{e^{s_2 - s_1}}}$$

$$\boldsymbol{s_2 - s_1 = ln\left(\frac{P(X|Y_2)P(Y_2)}{P(X|Y_1)P(Y_1)}\right)}$$

Log Odds라고 부른답니다

$$\boldsymbol{P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)}$$

$$a = -(\boldsymbol{s_2} - \boldsymbol{s_1})$$

참고 : Bayes Theorem과 Sigmoid와 Softmax사이의 관계

Logic Function : 2-class classification

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)}$$

Sigmoid가 유도됩니다

$$= \frac{e^{s_1}}{e^{s_1} + e^{s_2}} = \frac{1}{1 + e^{s_2 - s_1}}$$

$$s_2 - s_1 = ln\left(\frac{P(X|Y_2)P(Y_2)}{P(X|Y_1)P(Y_1)}\right)$$

$$a = -(s_2 - s_1) = ln\left(\frac{P(X|Y_1)P(Y_1)}{P(X|Y_2)P(Y_2)}\right)$$

$$P(X) = P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)$$

참고 : [Bayes Theorem과 Sigmoid와 Softmax사이의 관계]

Logic Function : 2-class classification

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)}$$

Naive Bayes

$$= \frac{e^{s_1}}{e^{s_1}+e^{s_2}} = \frac{1}{1+e^{s_2-s_1}} = \frac{1}{1+e^{-a}}$$

남자가 비냉먹을 likelihood 남자의 비율

Gaussian 분포가정 Bernoulli 분포가정

$$a = -(s_2 - s_1) = ln\left(\frac{P(X|Y_1)P(Y_1)}{P(X|Y_2)P(Y_2)}\right)$$

Log Odds

→ Naive Bayes

참고 : Bayes Theorem과 Sigmoid와 Softmax사이의 관계

Logic Function : 2-class classification

Unnormalized log probability

$$s_k = \ln(P(X|Y_k)P(Y_k))$$

$$P(Y_1|X) = \frac{P(X|Y_1)P(Y_1)}{P(X|Y_1)P(Y_1) + P(X|Y_2)P(Y_2)}$$

Naive Bayes

$$= \frac{e^{s_1}}{e^{s_1}+e^{s_2}} = \frac{1}{1+e^{s_2-s_1}} = \frac{1}{1+e^{-a}}$$

$$a = -(s_2 - s_1) = ln\left(\frac{P(X|Y_1)P(Y_1)}{P(X|Y_2)P(Y_2)}\right)$$

Log Odds

그냥 a = Wx로 하면
Logistic Regression

참고 : Bayes Theorem과 Sigmoid와 Softmax사이의 관계

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

Softmax vs. SVM

$$\boxed{L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)} \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

모두의연구소

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

$$\boxed{L_i = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)} \qquad L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

ln을 적용하여 표현 : $s_k = \ln(P(X = x_i | Y = k)P(Y = k)) = \mathrm{f}(\mathrm{x}; \mathrm{W}) \overset{\text{e.g}}{=} \mathrm{Wx}$

우리가 만든 Score function을 unnormalized log probability로 표현한 것임

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

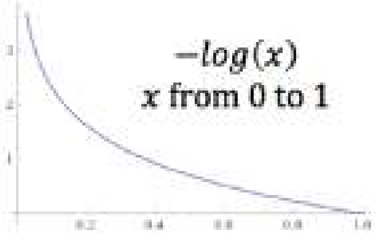
Cross entropy는 유도는 참고하세요 : [Cross Entropy의 정확한 확률적 의미]

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

Softmax vs. SVM

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

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



$\boxed{f(x_i ; W)}$

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

x

155
200
...
110
78

\+

0.1
0.2
0.8

b

=

	exp		Normalize	
-2.85	→	0.058	→	0.016
0.86	→	2.36	→	0.631
0.28	→	1.32	→	0.353

→ -log(0.353)

- 전부 양수
- 값의 차이 벌림

- 합이 1

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

모두의연구소

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

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

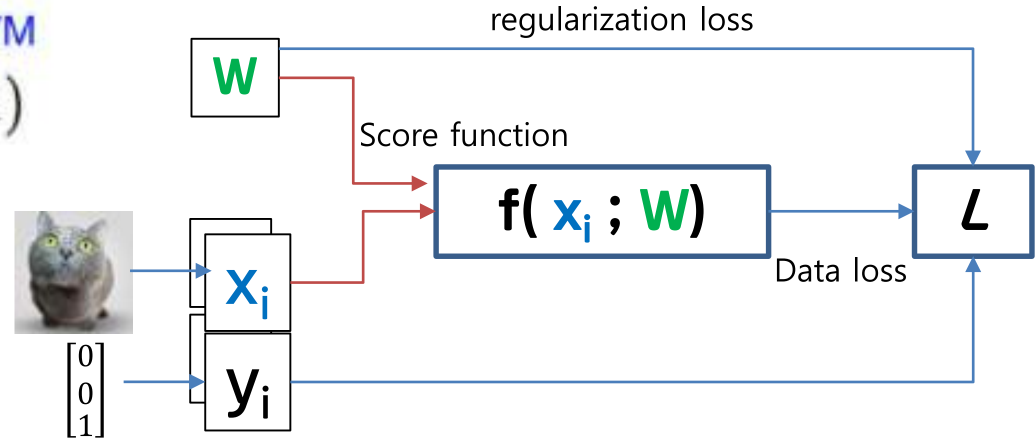
Softmax

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

SVM

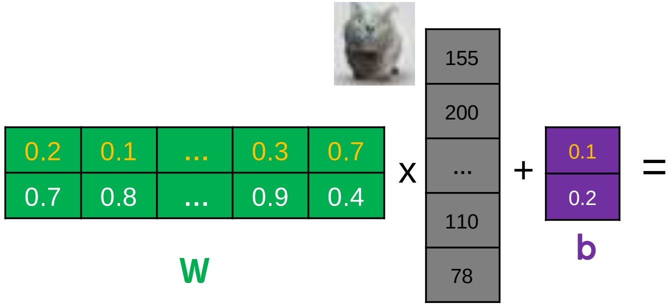
$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

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



돌아보기 ...

- 분류기의 구성
  - Score function
  - Loss function
  - Optimization



오늘의 주제

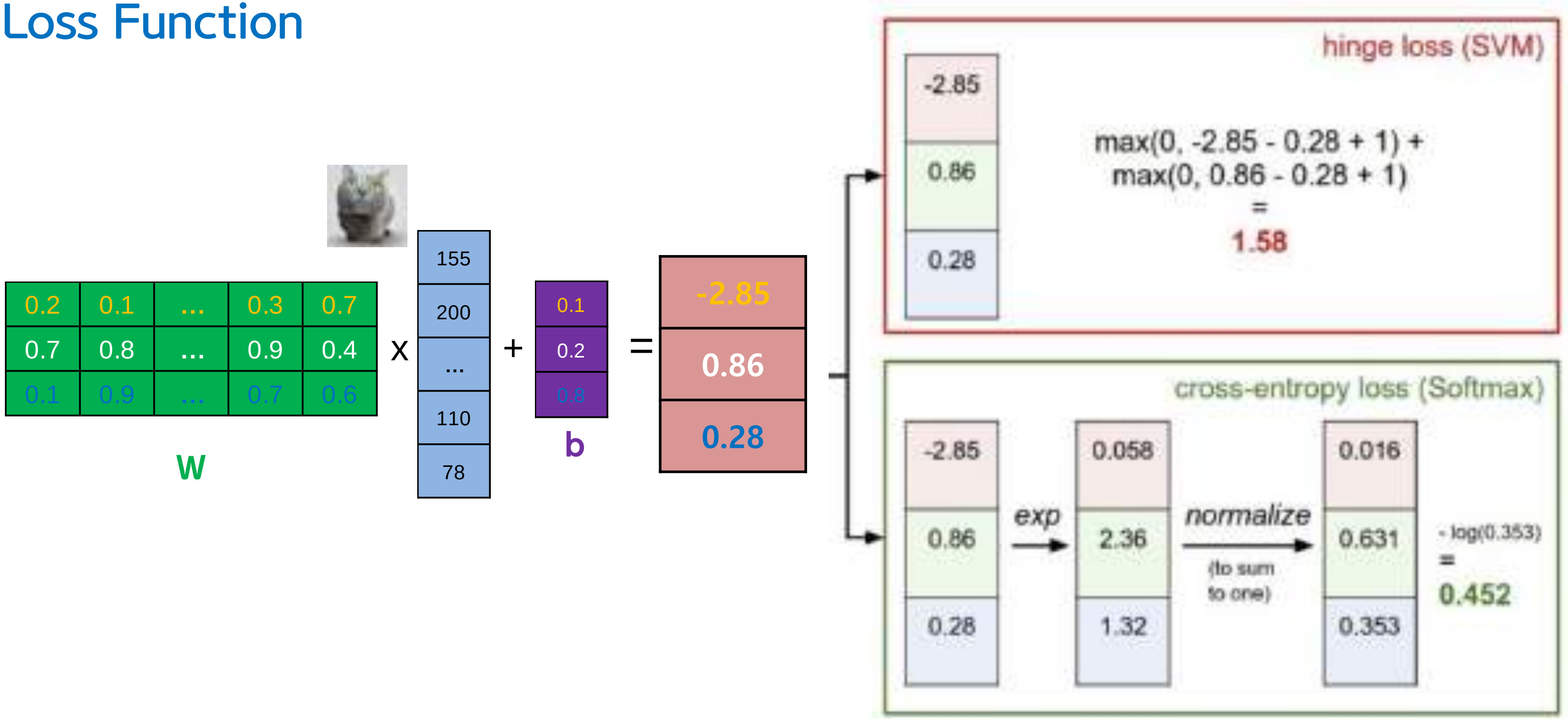
Loss를 최소화하는 w와 b를 찾아라

Softmax vs. SVM

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

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

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




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!
그냥 눈을 다시 그려줘 ...

내리막을 찾는 방법

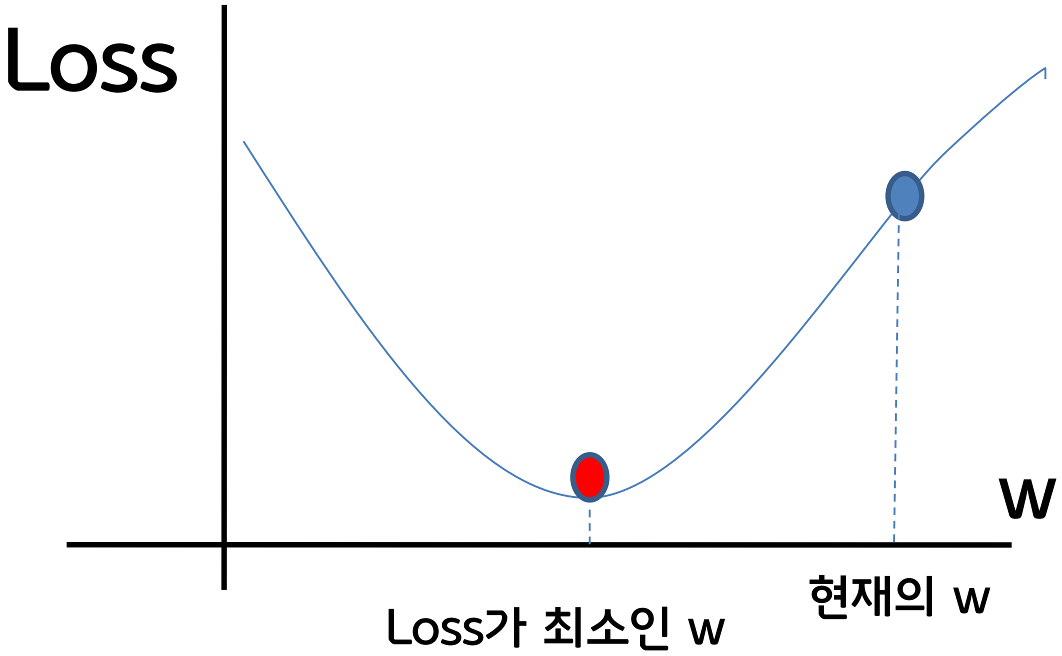
모두의연구소

내리막 ?? == 기울기 ??

기울기를 따라 내려 가보자

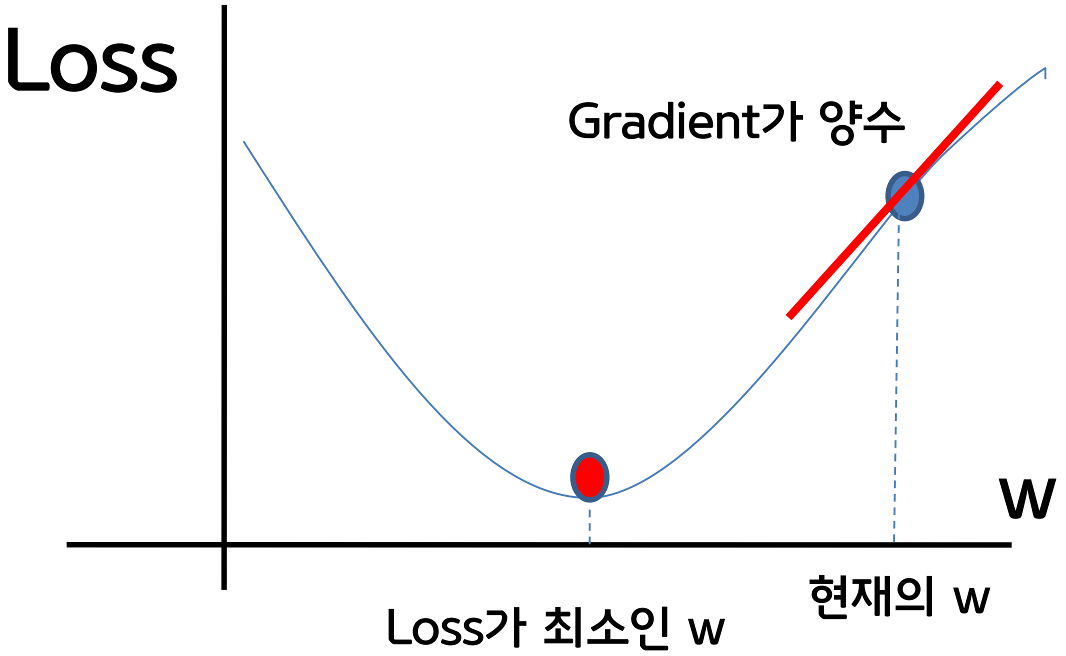
Gradient Descent

Gradient Descent



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

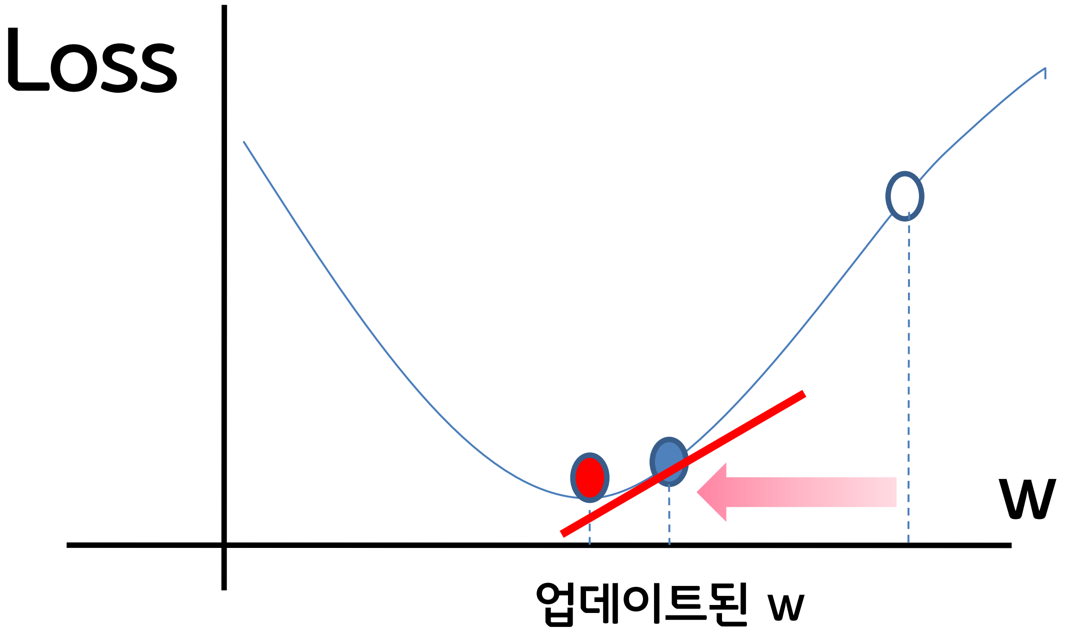
Gradient Descent



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

η : learning rate

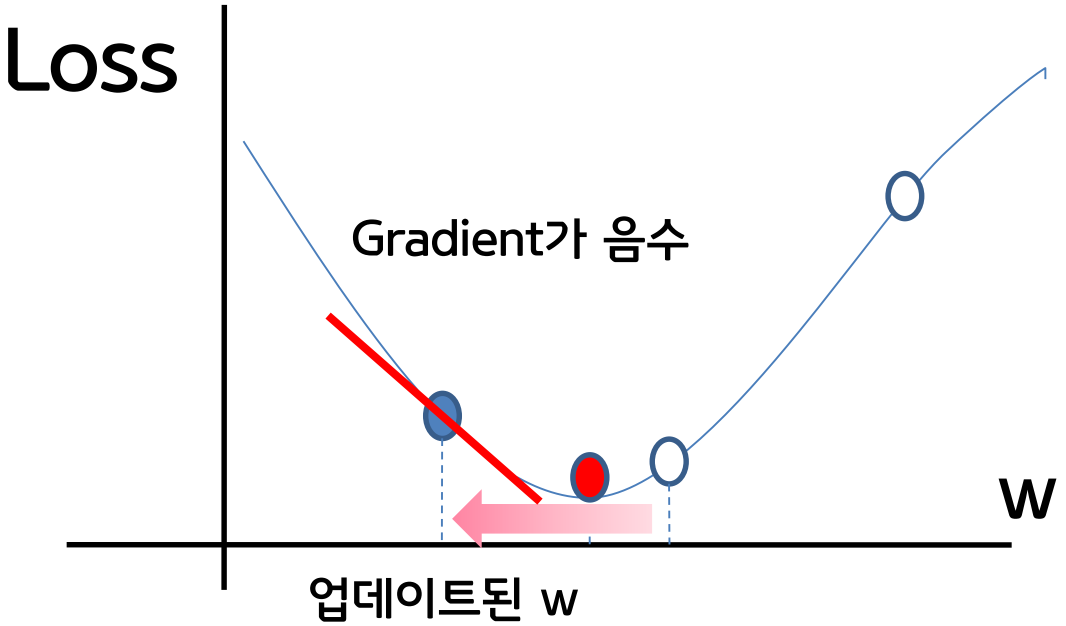
Gradient Descent



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

η : learning rate

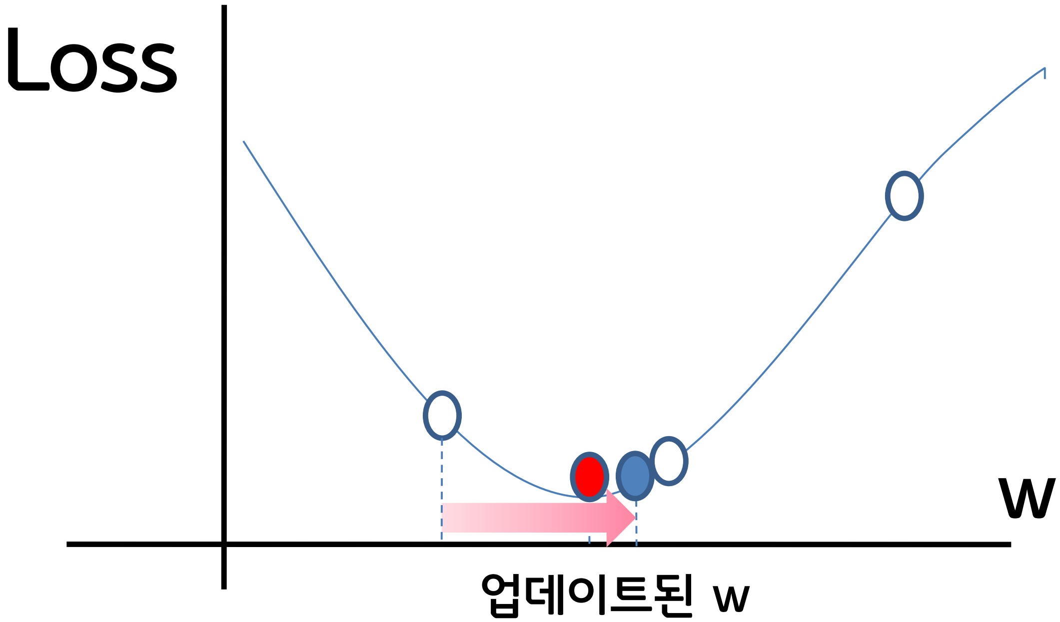
Gradient Descent



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

η : learning rate

Gradient Descent



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

Loss 함수에 대한 w의 음의 Gradient 찾아서 연속적으업데이트해 주면 되는군요

그런데 어떻게 Gradient를 찾죠?

Gradient Descent

모두의연구소

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

모두의연구소

current W:		gradient dW:
[0.34,		[?,
-1.11,		?,
0.78,		?,
0.12,		?,
0.55,		?,
2.81,		?,
-3.1,		?,
-1.5,		?,
0.33,…]		?,…]
loss 1.25347		

모두의연구소

$$\underbrace{\begin{bmatrix} 0.2 & 0.1 & \dots & 0.3 & 0.7 \\ 0.7 & 0.8 & \dots & 0.9 & 0.4 \end{bmatrix}}_{W} \times \underbrace{\begin{bmatrix}155\\200\\\dots\\110\\78\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}0.1\\0.2\end{bmatrix}}_{b} = \begin{bmatrix}2.3\\-1.2\end{bmatrix} \xrightarrow{exp} \begin{bmatrix}9.97\\0.3\end{bmatrix} \xrightarrow{normalize} \begin{bmatrix}0.97\\0.03\end{bmatrix}$$

강아지 점수

고양이 점수

강아지 확률

고양이 확률

$$-log(0.03) = 3.5$$

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

Gradient Descent

모두의연구소

current W:	W + h (first dim):	gradient dW:
[0.34,	[0.34 + **0.0001**,	[?,
-1.11,	-1.11,	?,
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 1.25322	

Gradient Descent

모두의연구소

current W:	W + h (first dim):	gradient dW:
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	(1.25322 - 1.25347)/0.0001 = -2.5
0.55,	0.55,	$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

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

모두의연구소

current W:	**W + h (second dim):**	**gradient dW:**
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	**0.6**,
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**	

(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

모두의연구소

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

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

Fei-Fei Li & Justin Johnson & Serena Yeung Lecture 3 - 67 April 11, 2017

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

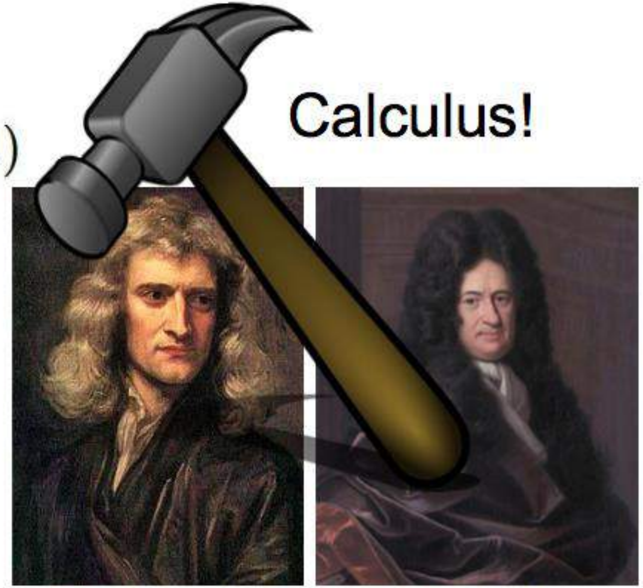
This is silly. The loss is just a function of W:

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

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

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

want $\nabla_W L$



This image is in the public domain This image is in the public domain

Use calculus to compute an **analytic gradient**

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

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

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?**

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

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

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

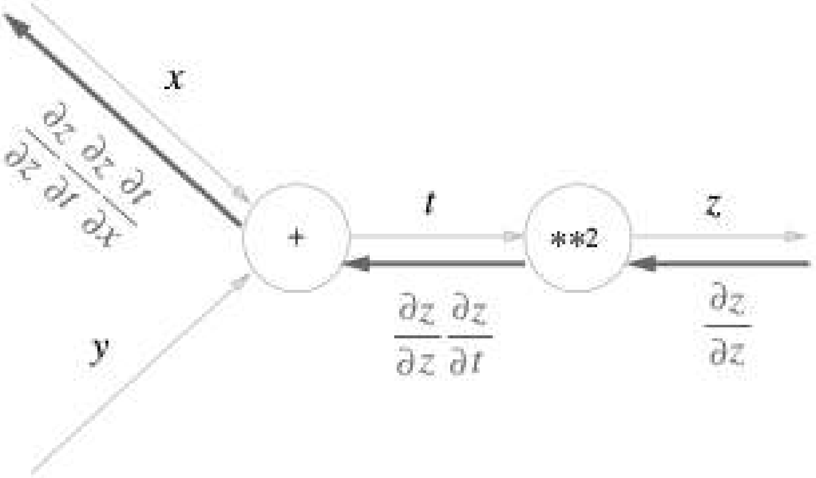
Reference : "밑바닥부터 시작하는 딥러닝", 사이토 고키

Computational Graph

- **연쇄법칙과 계산 그래프**

역전파 과정

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

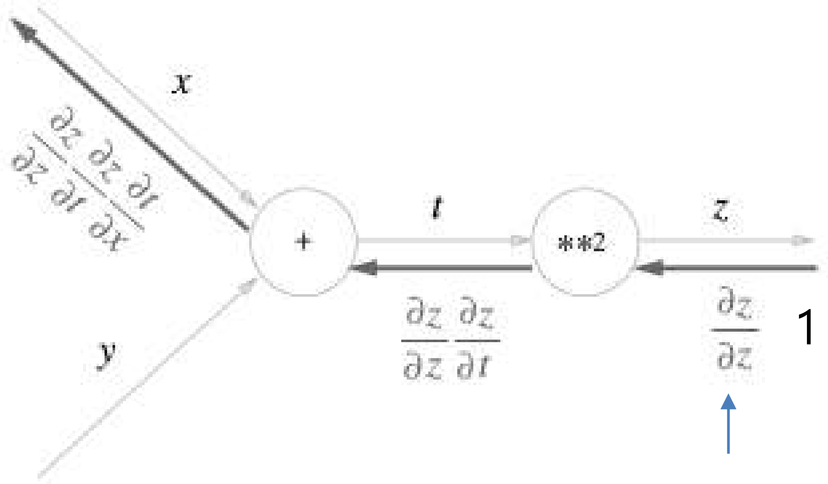


Computational Graph

- **연쇄법칙과 계산 그래프**

역전파 과정

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



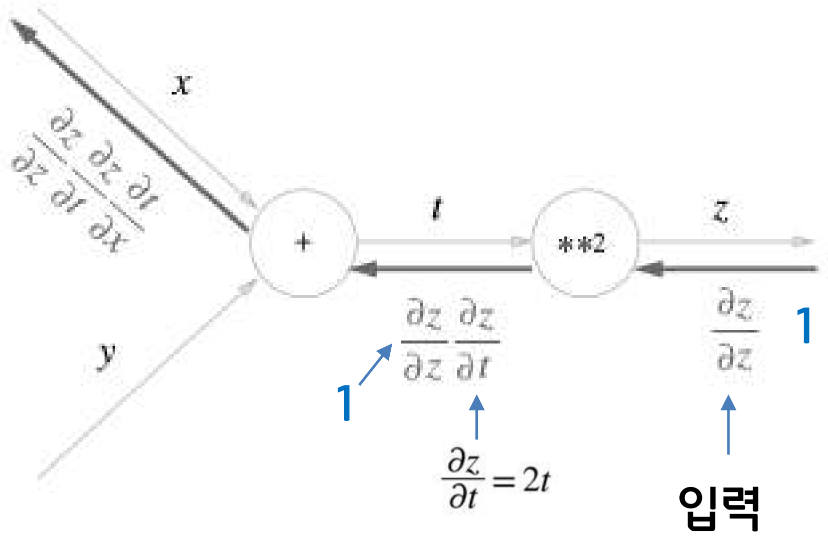
입력

Computational Graph

- **연쇄법칙과 계산 그래프**

역전파 과정

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

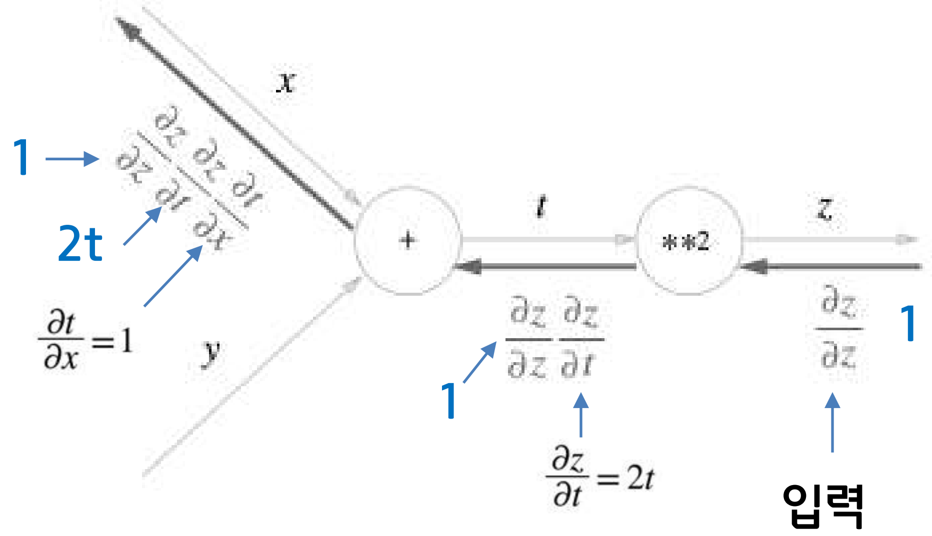


Computational Graph

- **연쇄법칙과 계산 그래프**

역전파 과정

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



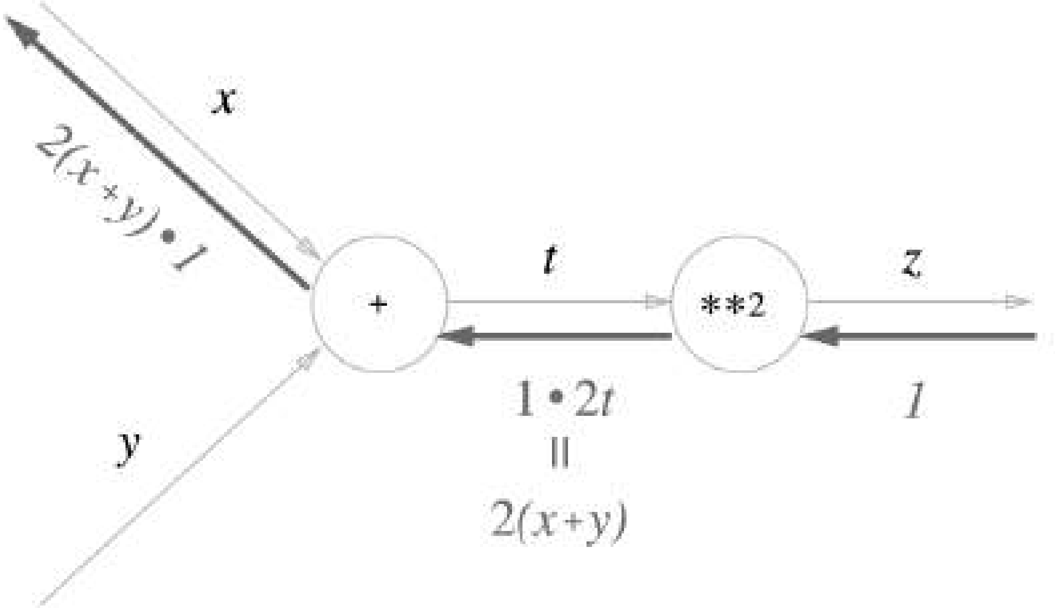
Reference : "밑바닥부터 시작하는 딥러닝", 사이토 고키

Computational Graph

- **연쇄법칙과 계산 그래프**

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

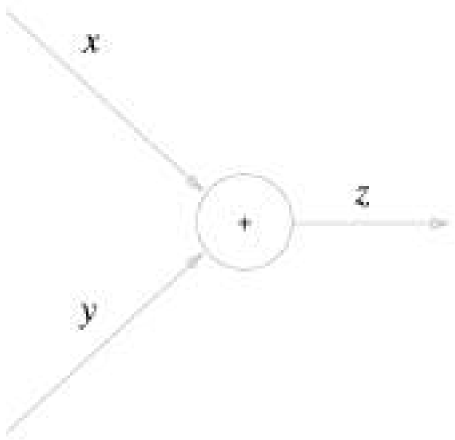
역전파 과정

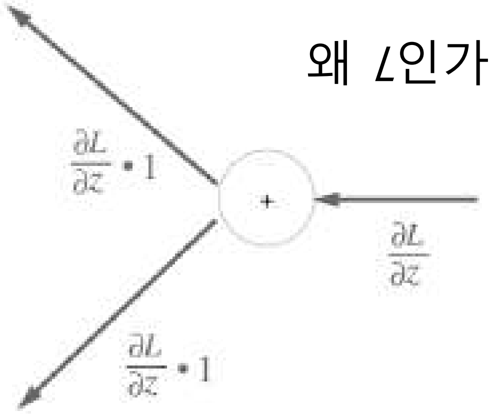


Computational Graph

- **덧셈 노드의 역전파**

$$z = x + y \Rightarrow \frac{\partial z}{\partial x} = 1, \quad \frac{\partial z}{\partial y} = 1$$



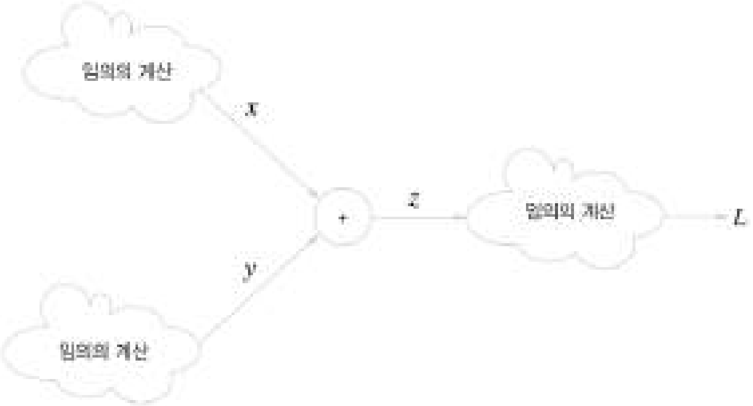


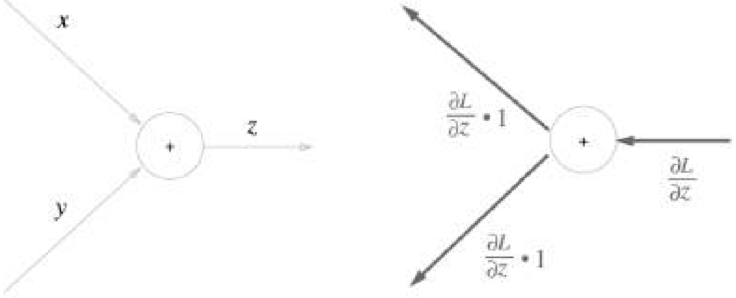
왜 ∠인가요?

Computational Graph

- 덧셈 노드의 역전파

실제로 큰 계산을
가정하고 그중 한
노드가 덧셈노드임
을 가정한 겁니다

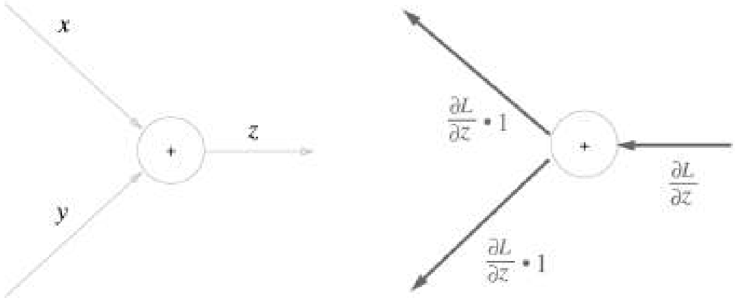




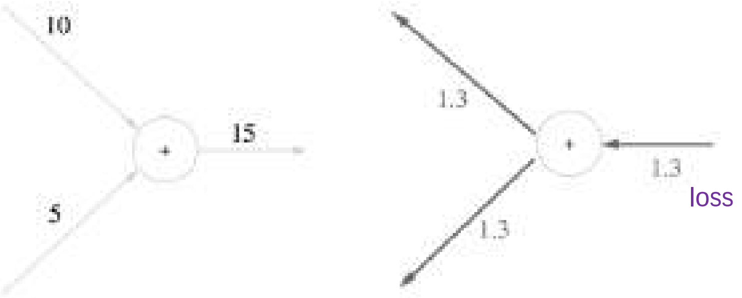
Computational Graph

- **덧셈 노드의 역전파**

$$z = x + y \quad \Rightarrow \quad \frac{\partial z}{\partial x} = 1, \quad \frac{\partial z}{\partial y} = 1$$



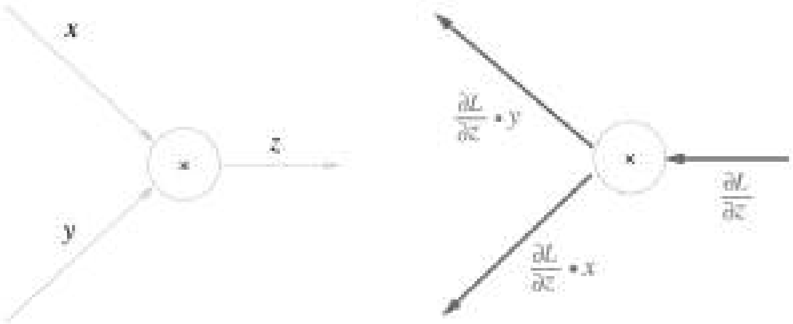
예시



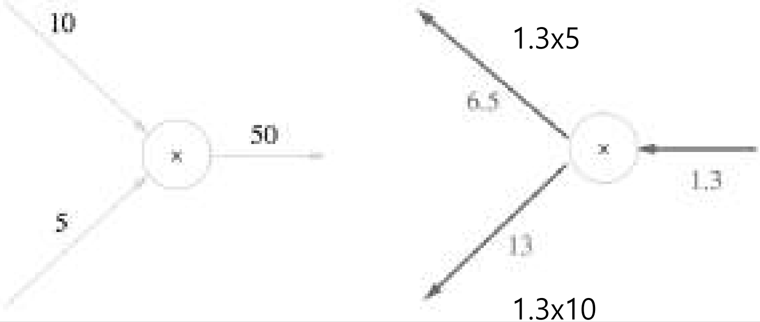
Computational Graph

- **곱셈 노드의 역전파**

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



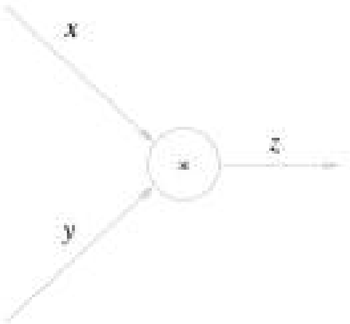
예시

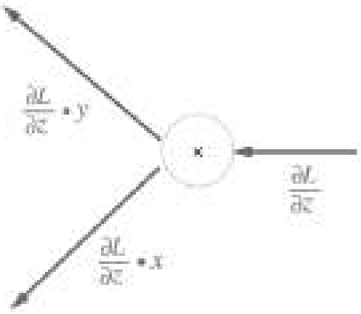


Computational Graph

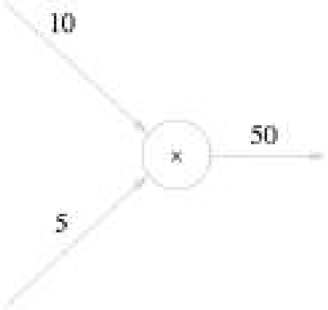
- **곱셈 노드의 역전파**

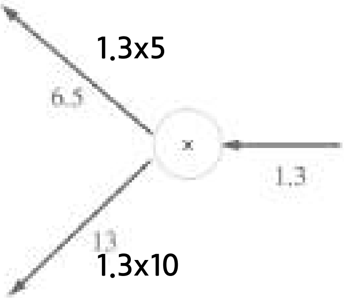
$$z = xy \quad \Rightarrow \quad \frac{\partial z}{\partial x} = \boxed{y} \qquad \frac{\partial z}{\partial y} = \boxed{x}$$





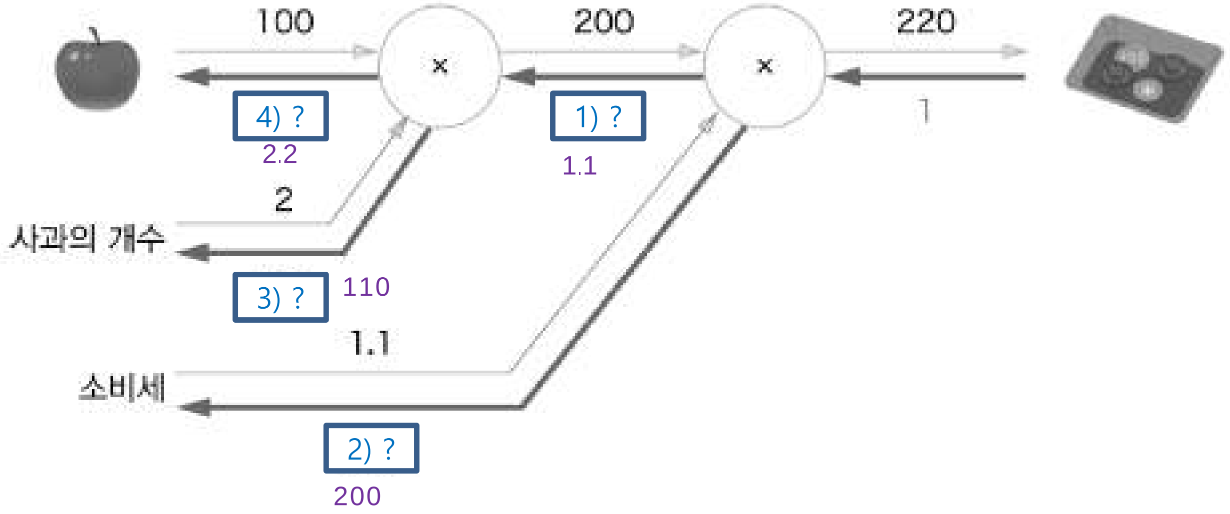
- **곱셈의 역전파는 순방향 입력신호의 값이 필요합니다**
- **곱셈노드 구현 시 순전파 입력신호를 변수에 저장해 둡니다**





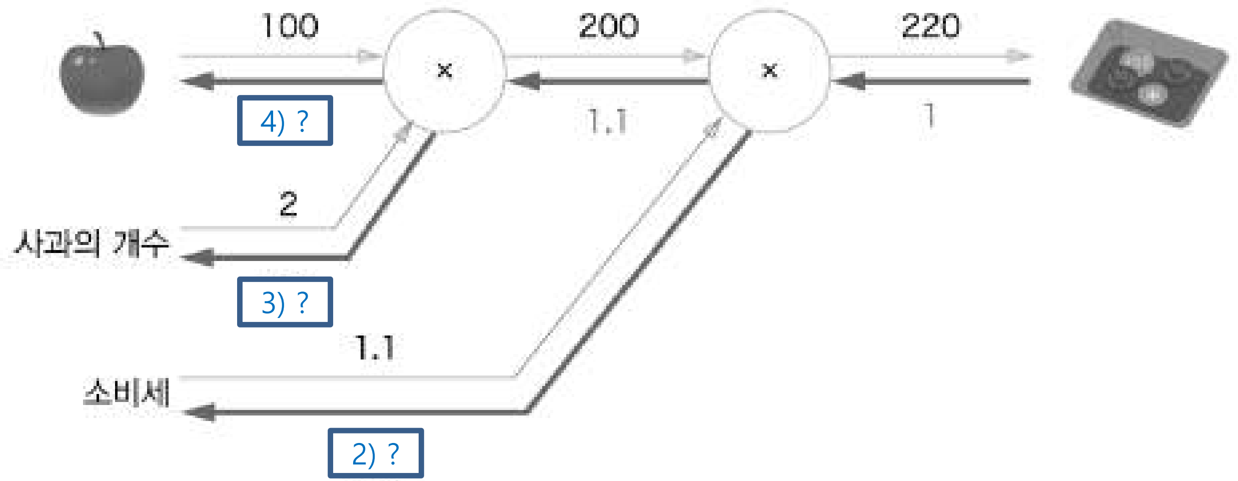
Computational Graph

- **사과쇼핑의 예**



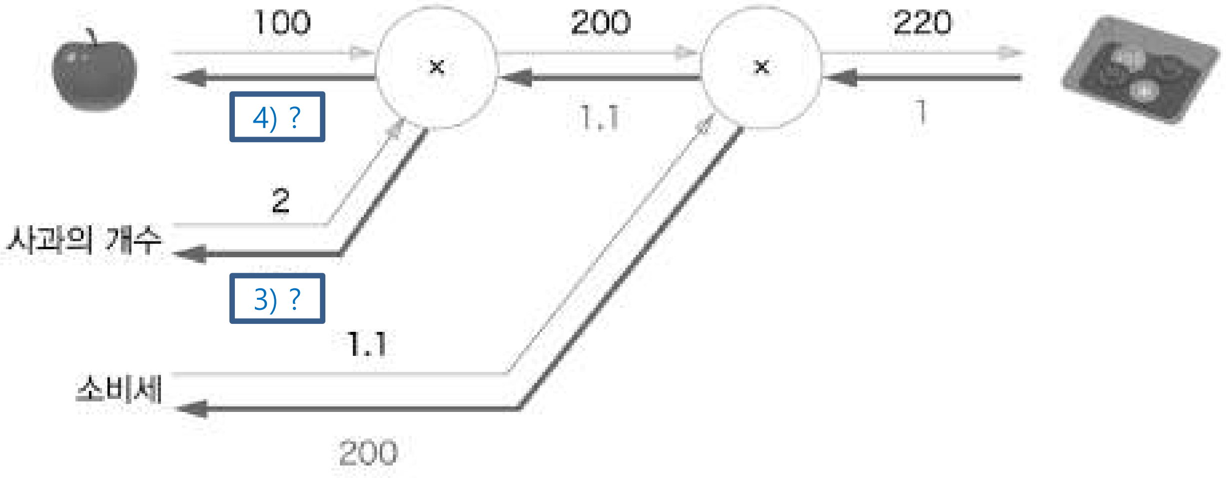
Computational Graph

- **사과쇼핑의 예**



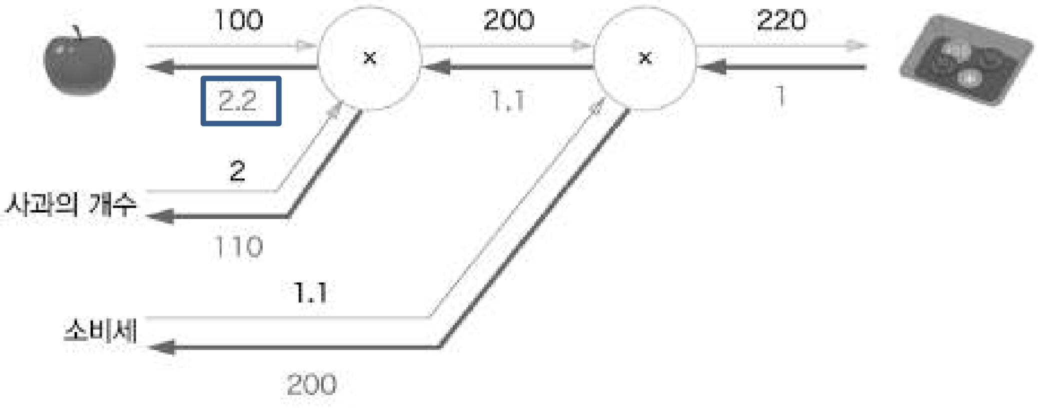
Computational Graph

- **사과쇼핑의 예**



Computational Graph

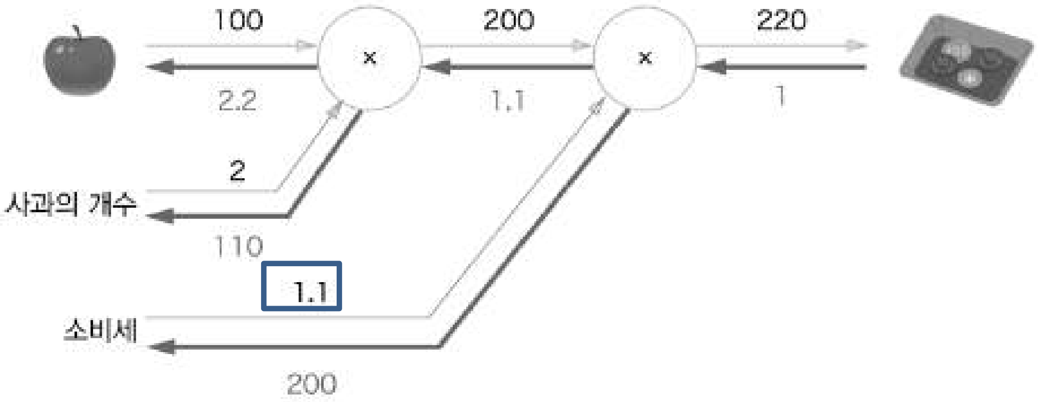
- **사과쇼핑의 예**



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

Reference : "밑바닥부터 시작하는 딥러닝", 사이토 고키

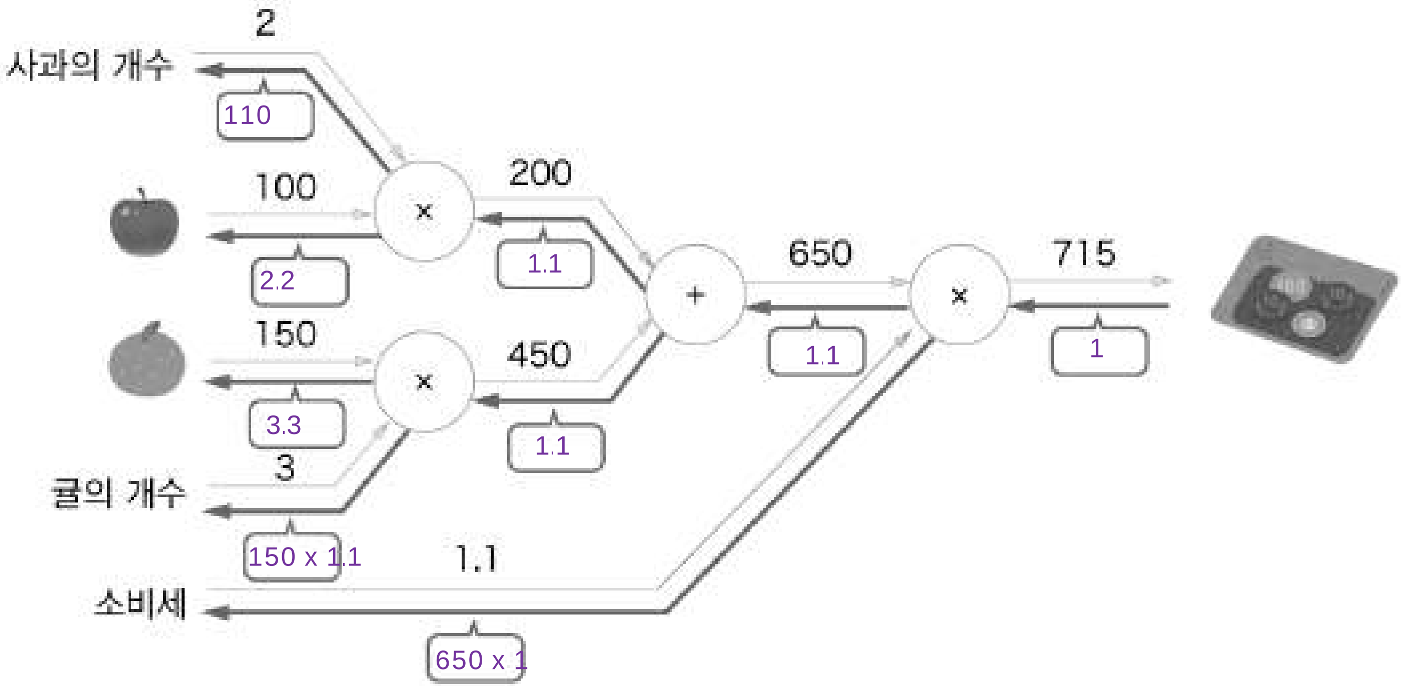
Computational Graph



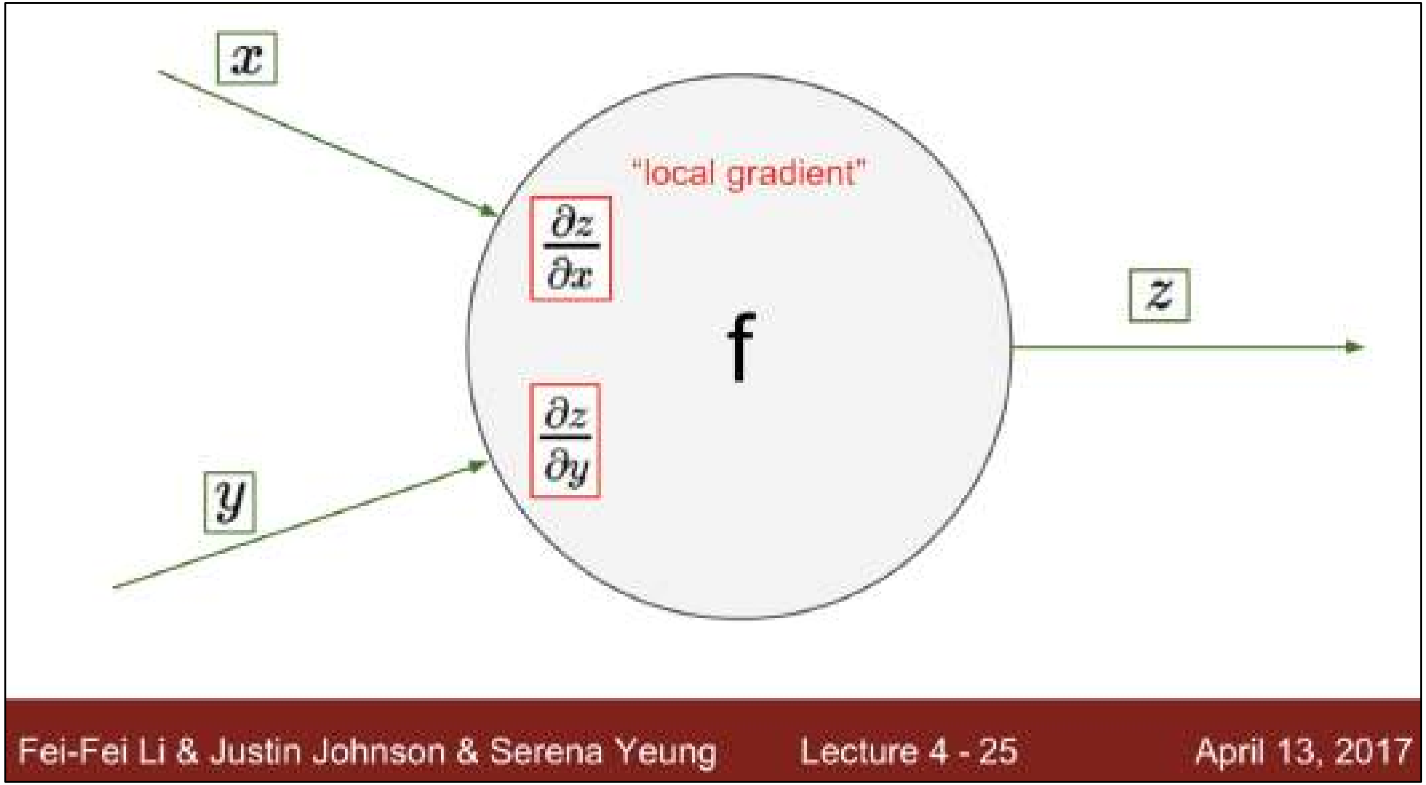
- 사과가격이 1원 오르면, 최종 가격 2.2 증가
  - 101(사과 가격)x2x1.1 = 222.2 원
- 소비세가 100%(1.1+1 = 2.1)오르면 가격 200증가
  - 100x2x2.1(소비세) = 420 원

Computational Graph

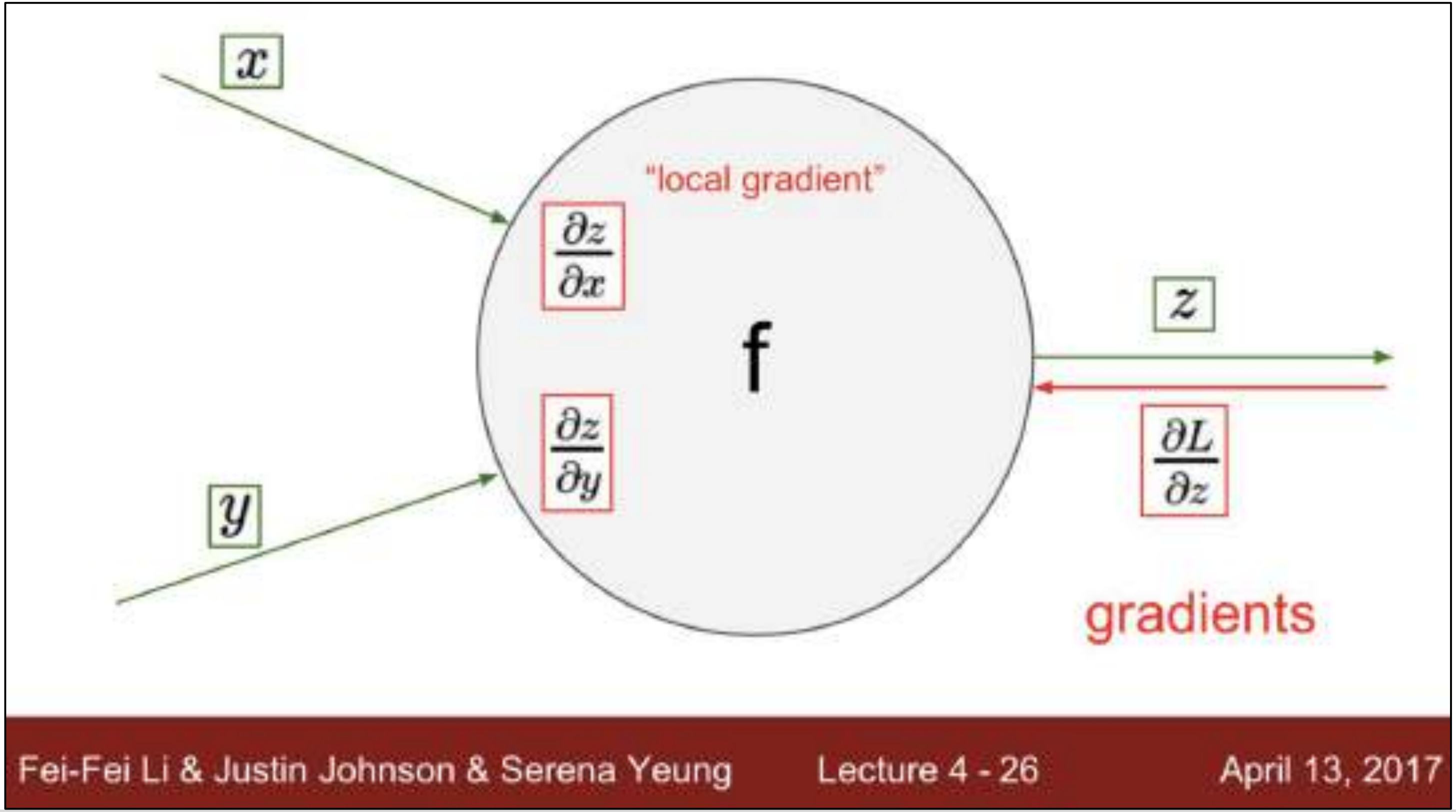
연습해보기



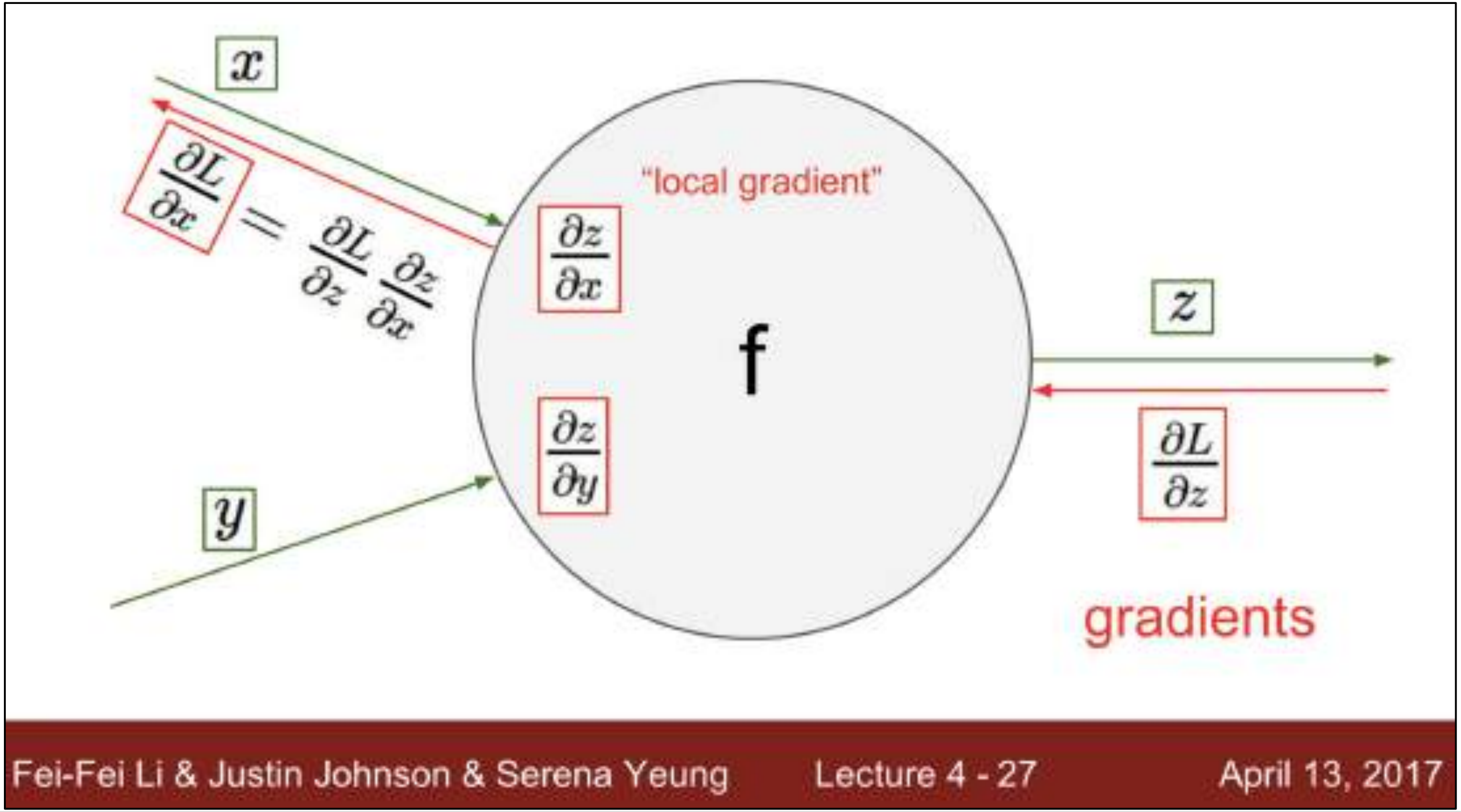
Computational Graph



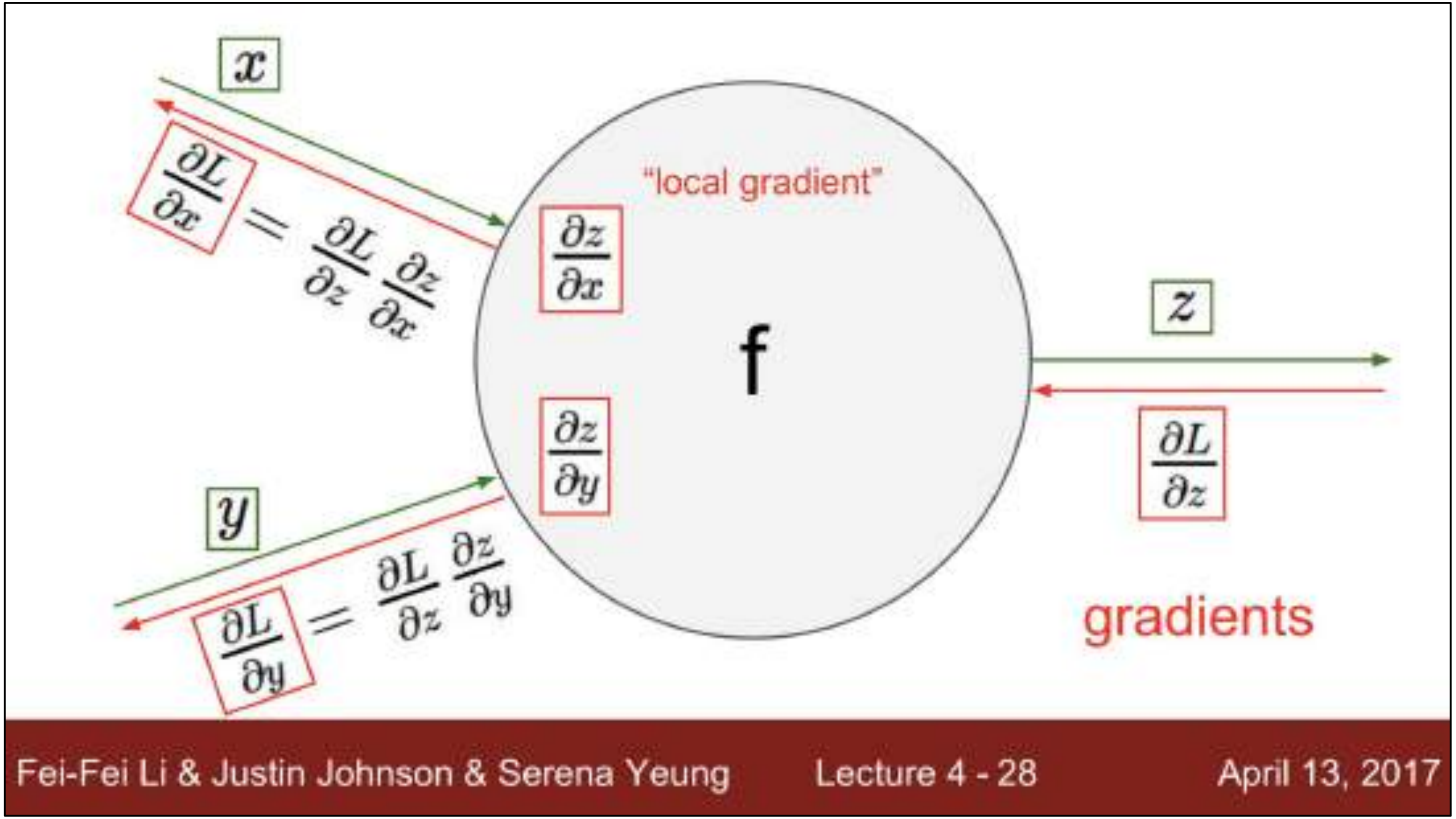
Computational Graph



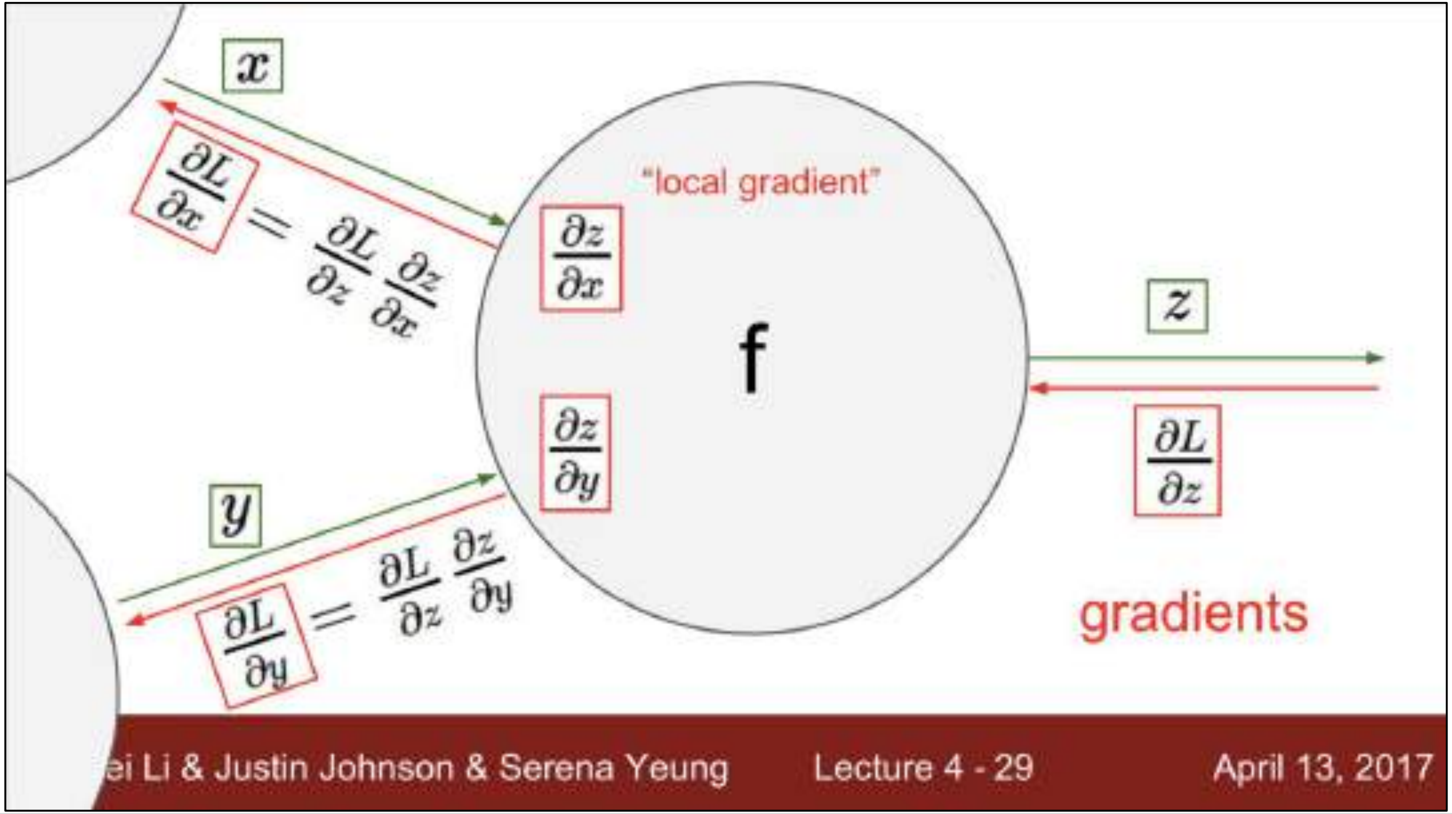
Computational Graph



Computational Graph



Computational Graph

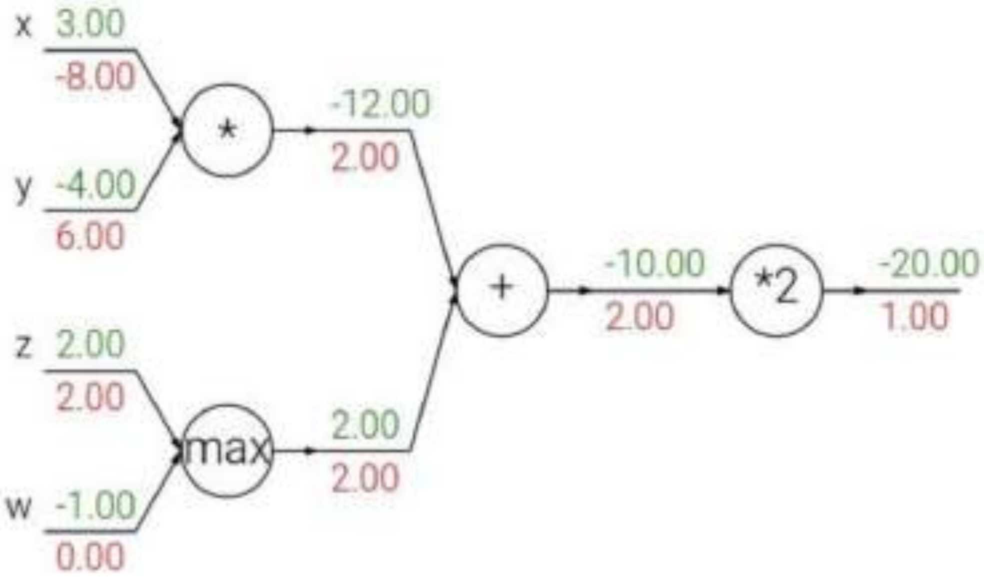


Patterns in backward flow

add gate: gradient distributor

max gate: gradient router

mul gate: gradient switcher



Computational Graph

모두의연구소

강아지 파라미터	0.2	0.1	…	0.3	0.7
고양이 파라미터	0.7	0.8	…	0.9	0.4

W

x

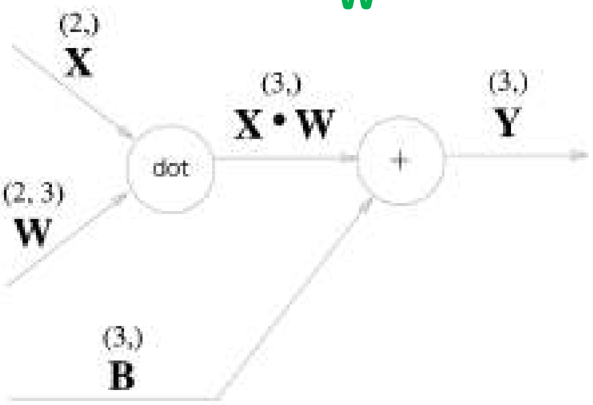
155
200
…
110
78

x

\+

0.1
0.2

b

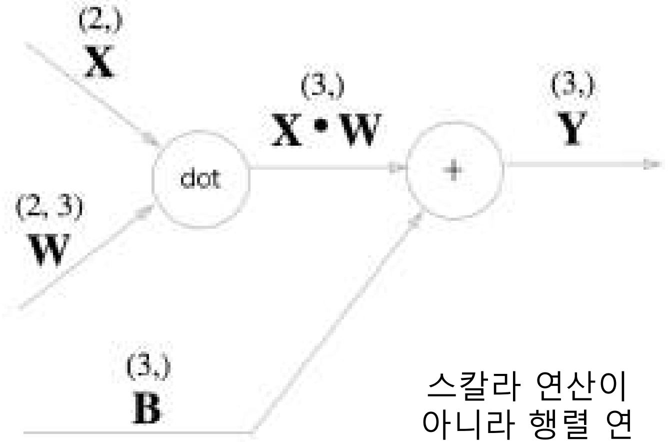


- Affine 계층

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
```

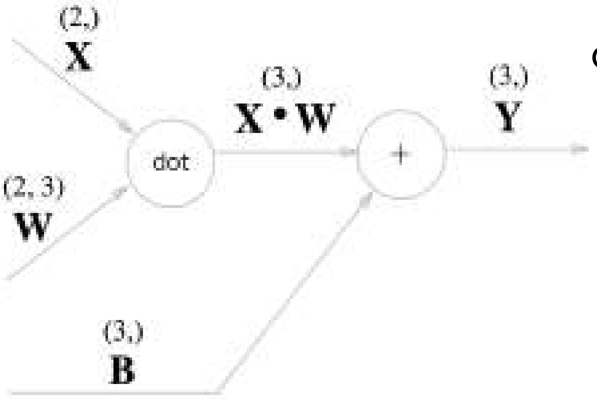


스칼라 연산이 아니라 행렬 연산이 됩니다

- 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

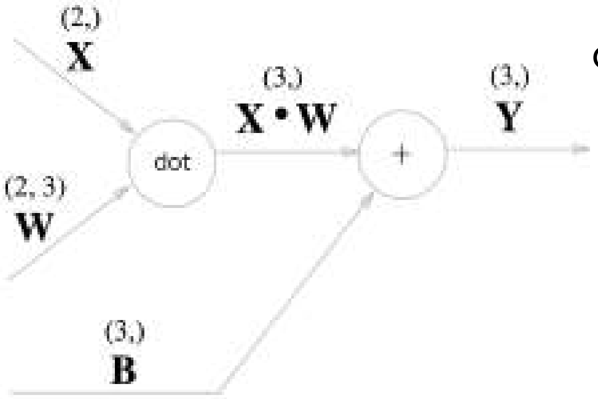
- Affine 계층
  - $\mathbf{W}^{\mathrm{T}}$의 T는 전치(transpose)행렬을 뜻합니다. 전치행렬은 $\mathbf{W}$의 (i, j) 위치의 원소를 (j, i) 위치로 바꾼 것을 말합니다.

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

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

Affine / Softmax 계층 구현하기

- Affine 계층



역전파

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

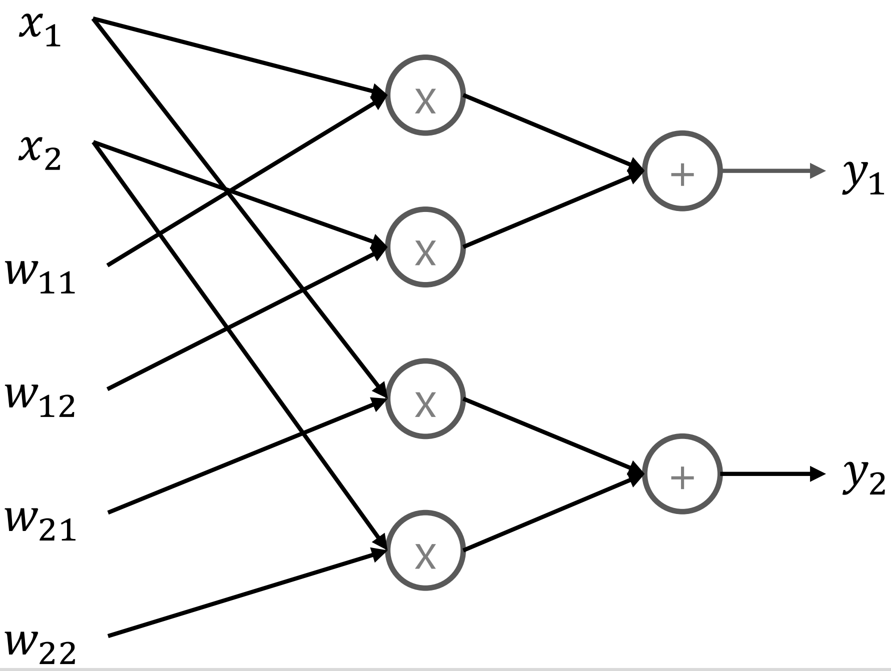
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

이건 왜 이렇게 된건가요?

Computational Graph

- **Affine 계층**

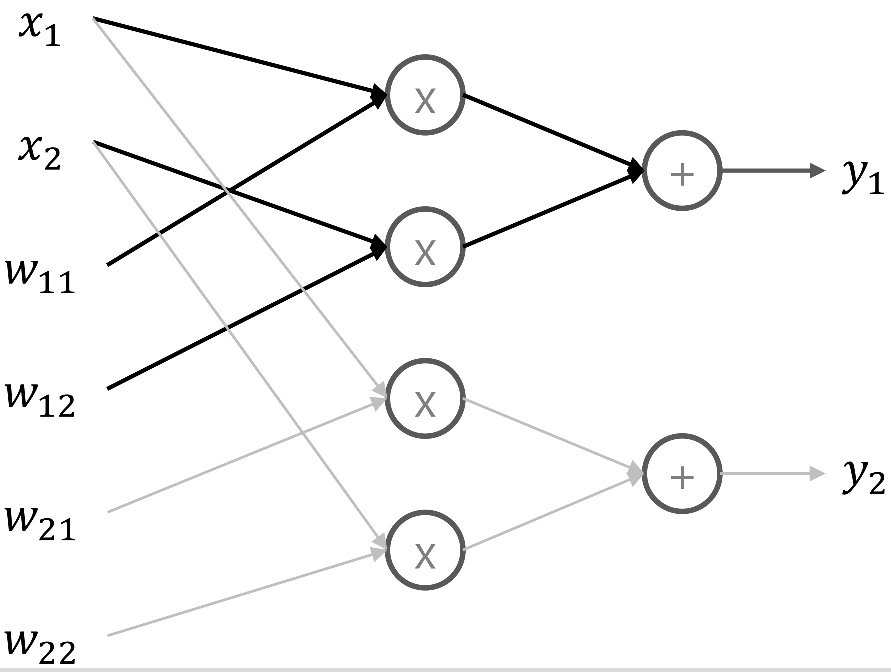
$$[x_1 \ \ x_2] \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \ \ y_2]$$



Computational Graph

- **Affine 계층**

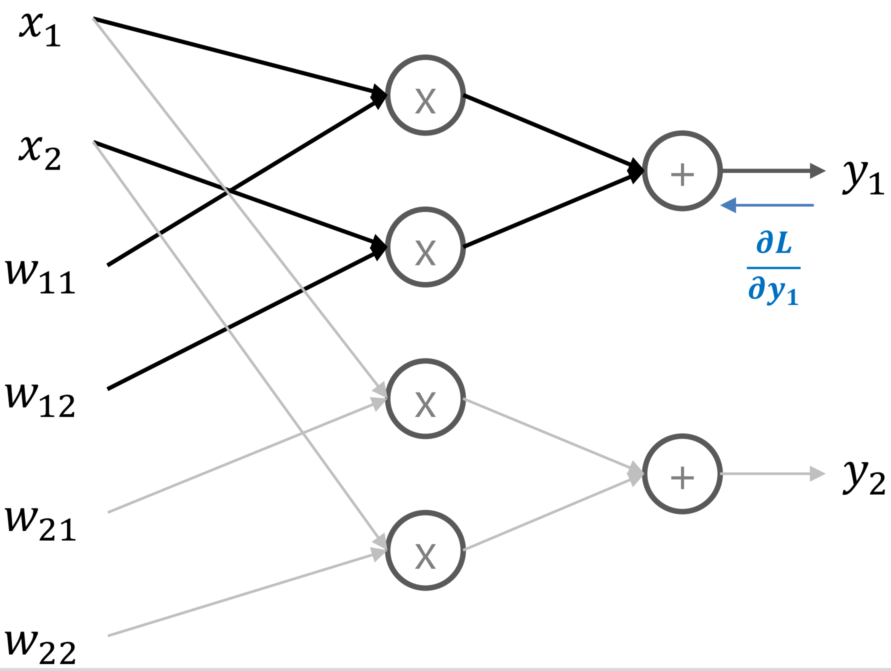
$$\boxed{[x_1\ \ x_2]} \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1\ \ y_2]$$



Computational Graph

- **Affine 계층**

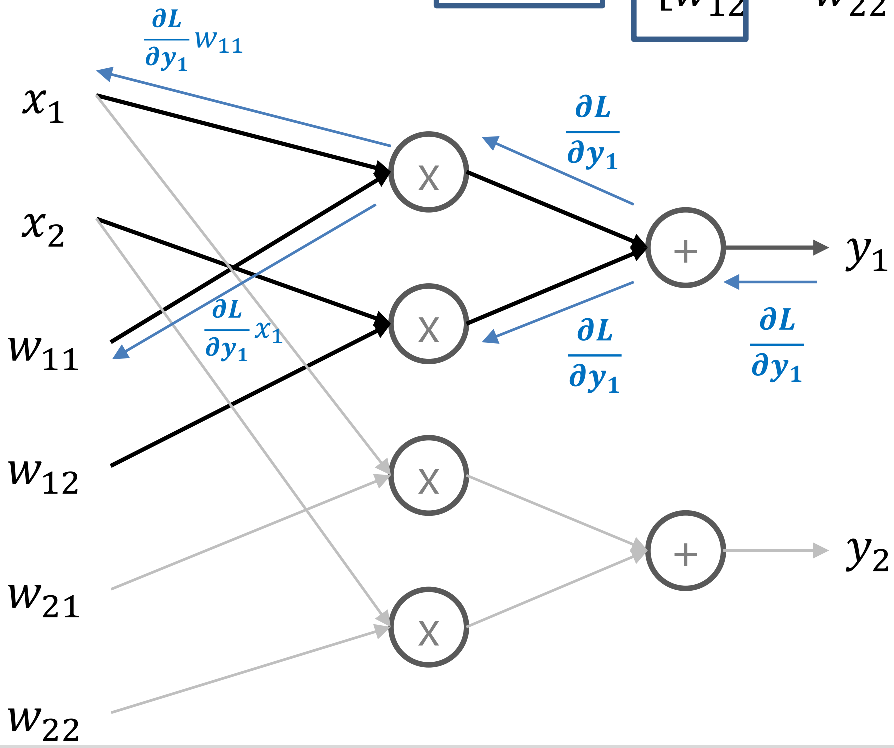
$$\boxed{[x_1 \;\; x_2]} \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



Computational Graph

- Affine 계층

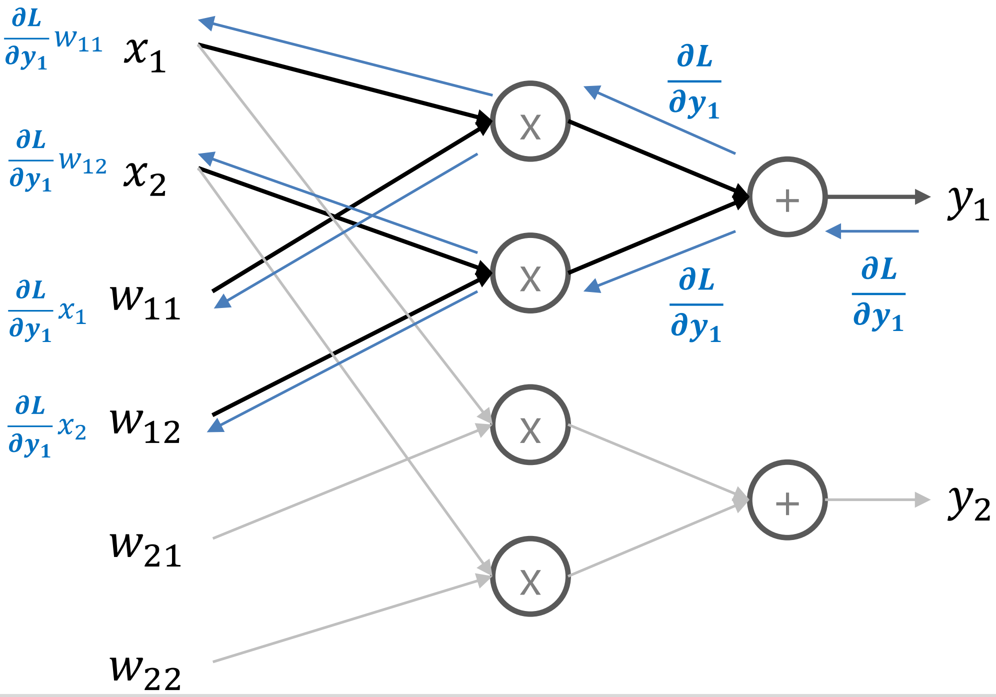
$$\boxed{[x_1\ \ x_2]} \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1\ \ y_2]$$



Computational Graph

- Affine 계층

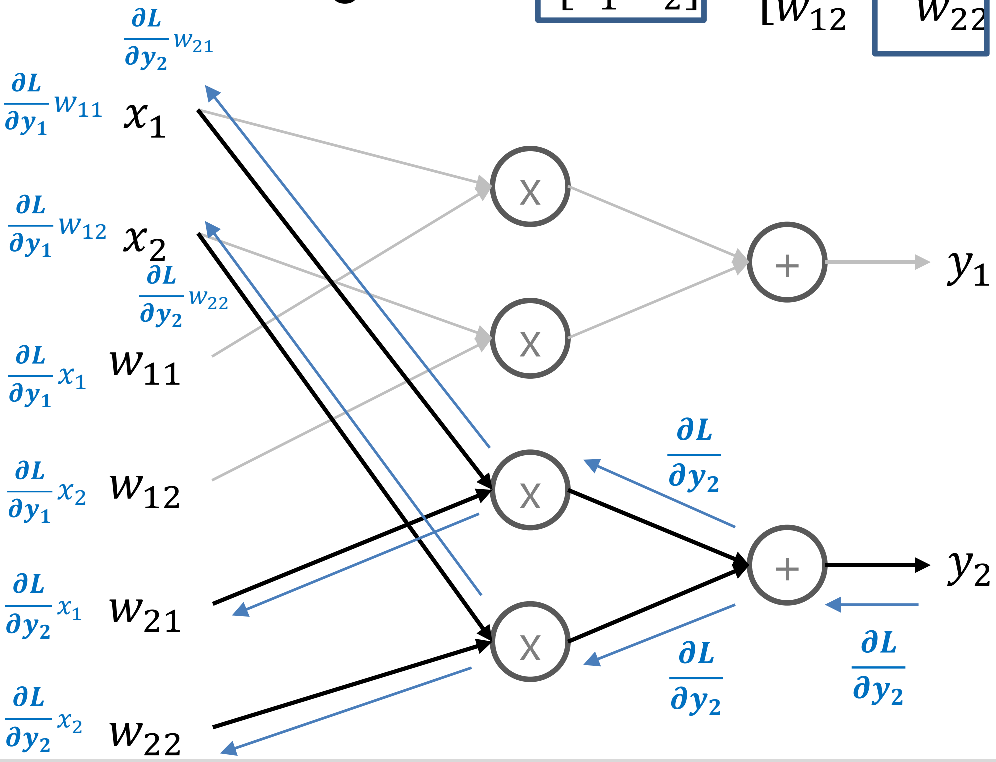
$$\boxed{[x_1 \;\; x_2]} \times \begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \end{bmatrix} = [y_1 \;\; y_2]$$



Computational Graph

- Affine 계층

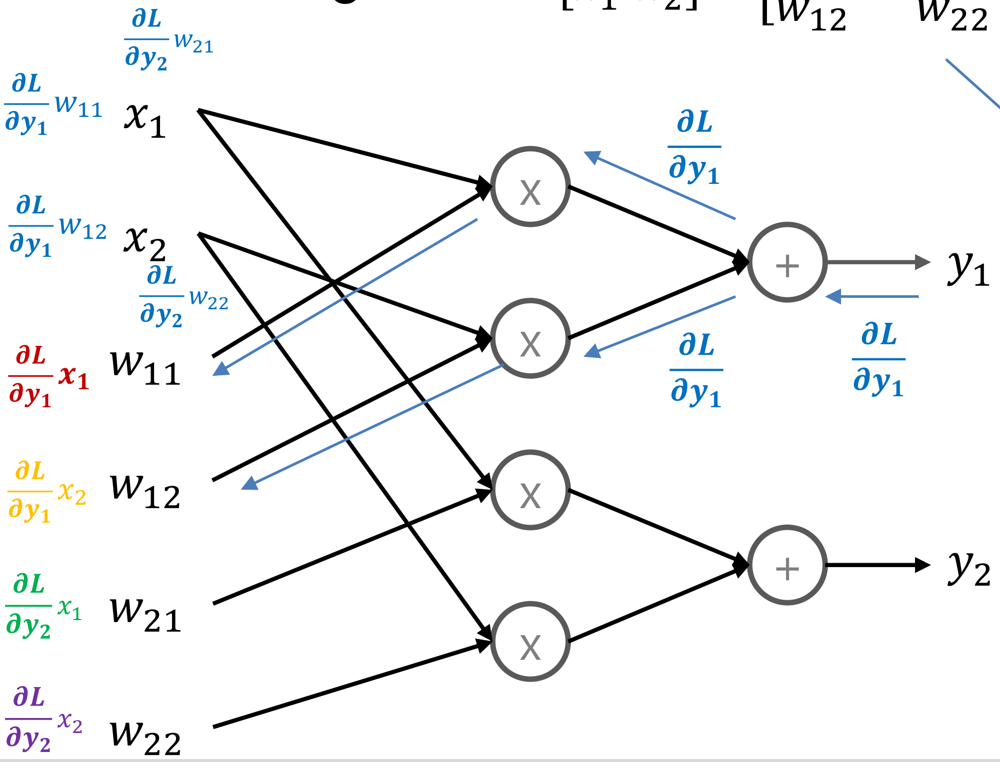
$$\boxed{[x_1 \ \ x_2]} \times \begin{bmatrix} w_{11} & \boxed{w_{21}} \\ w_{12} & \boxed{w_{22}} \end{bmatrix} = [y_1 \ \ \boxed{y_2}]$$



Computational Graph

- Affine 계층

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



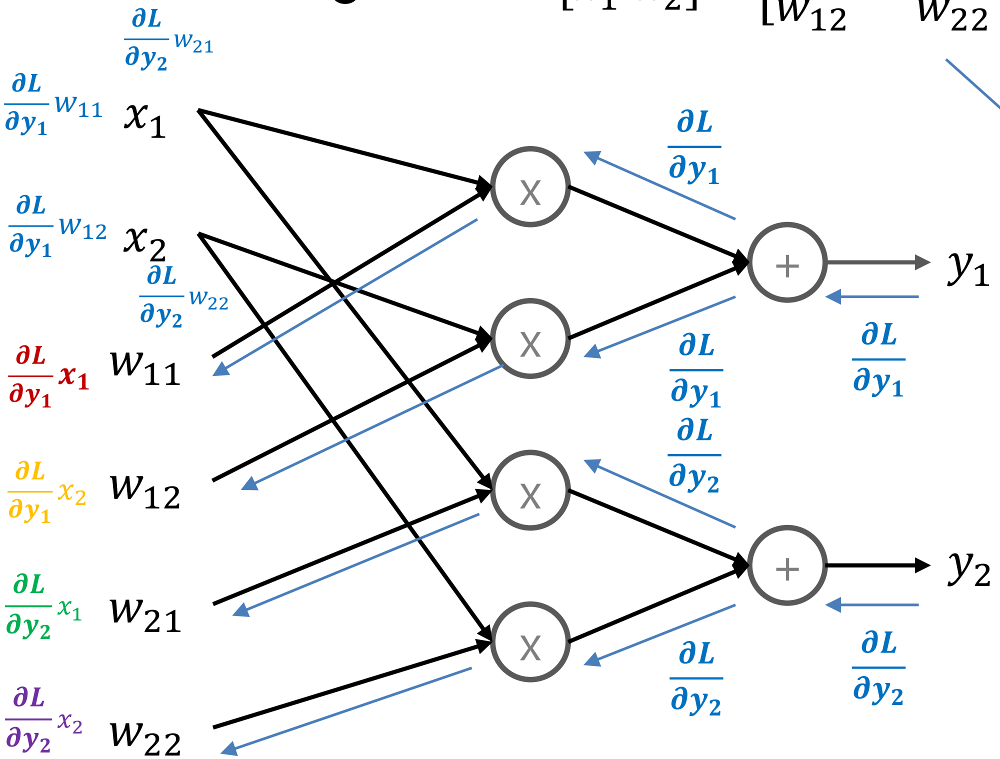
$$\frac{\partial L}{\partial \mathbf{W}}$$

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

Computational Graph

- Affine 계층

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



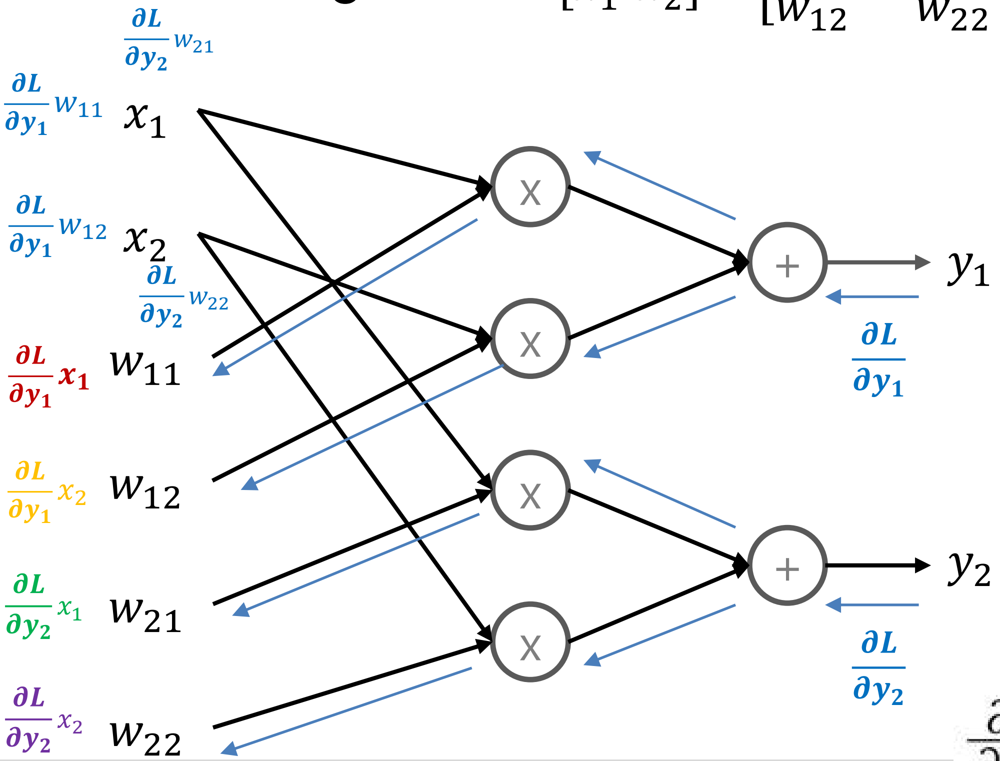
$$\frac{\partial L}{\partial \mathbf{W}}$$

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

Computational Graph

- Affine 계층

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



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

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

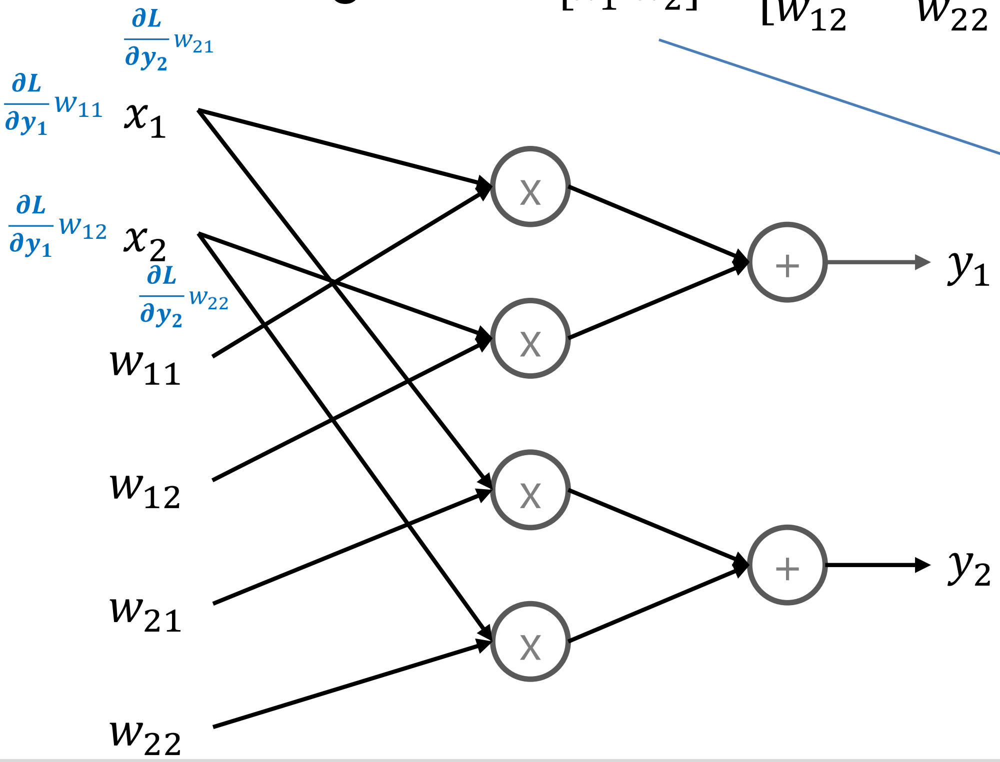
$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} \end{bmatrix}$$

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

Computational Graph

- Affine 계층

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



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

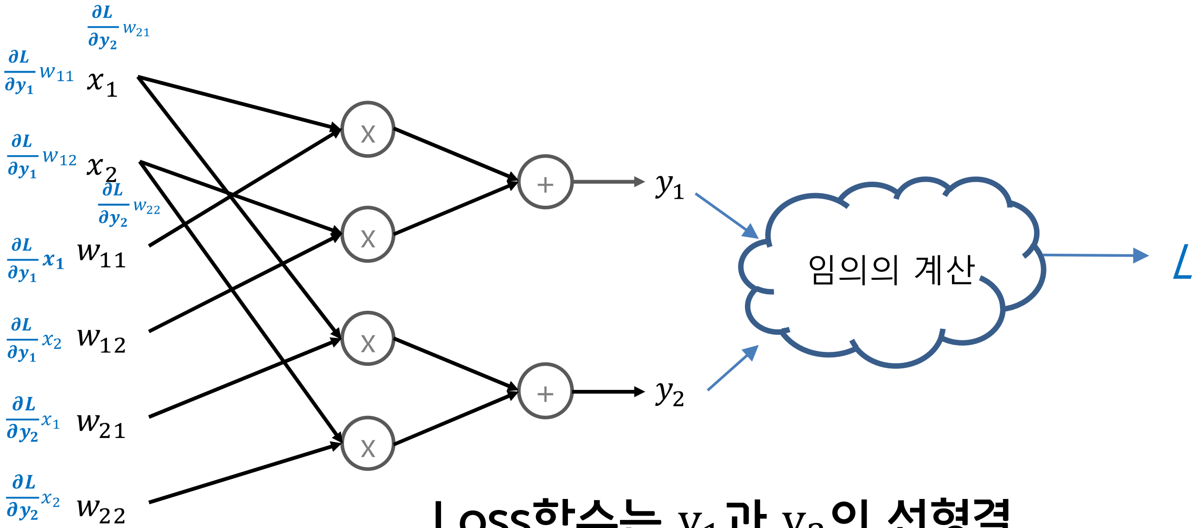
2개씩 나왔어요 ㅠㅠ

모두의연구소

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

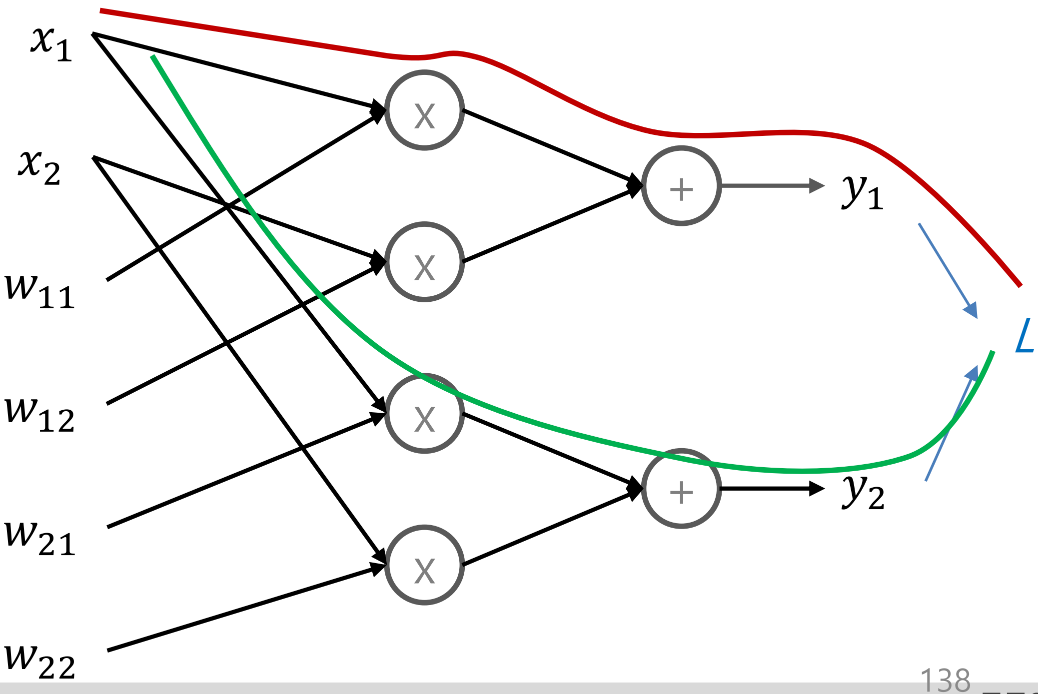
예) $L = ay_1 + by_2$

Computational Graph

- Affine 계층

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

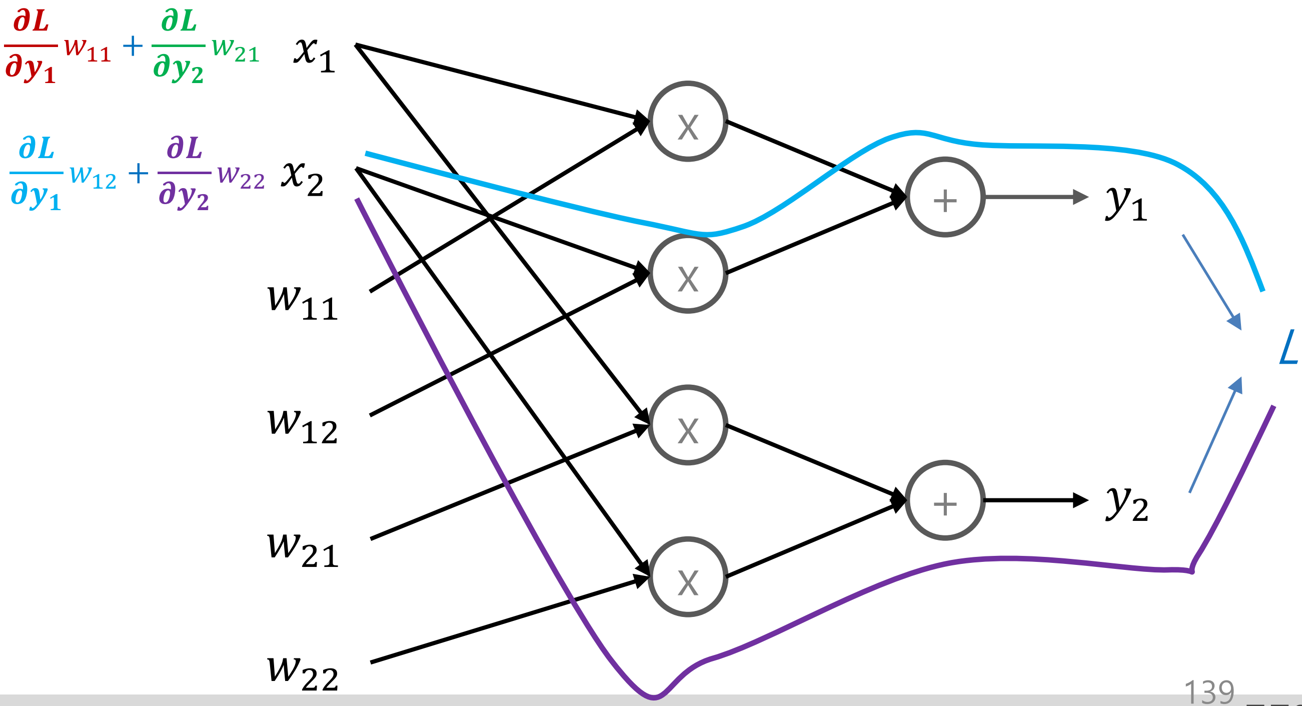
$$\frac{\partial L}{\partial y_1} w_{11} + \frac{\partial L}{\partial y_2} w_{21}$$



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

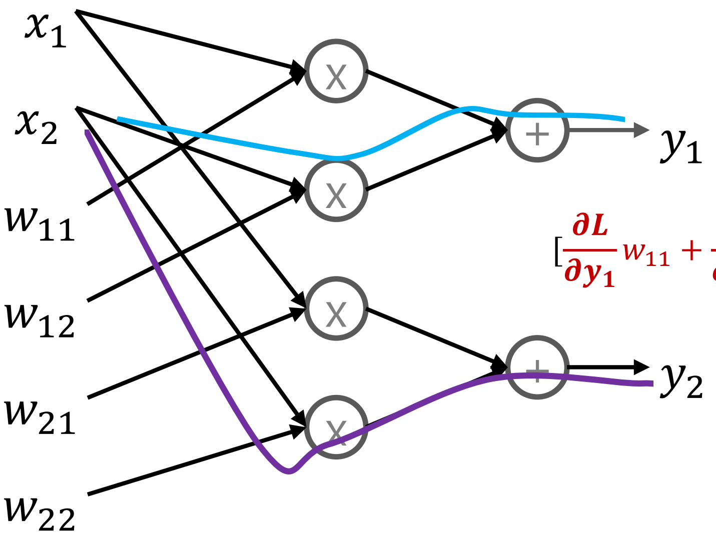
- Affine 계층

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

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

$$\frac{\partial L}{\partial y_1} w_{11} + \frac{\partial L}{\partial y_2} w_{21}$$

$$\frac{\partial L}{\partial y_1} w_{12} + \frac{\partial L}{\partial y_2} w_{22}$$



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

Computational Graph

- Affine 계층

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

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

$$\Downarrow$$

$$\begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} \end{bmatrix} \times \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}$$

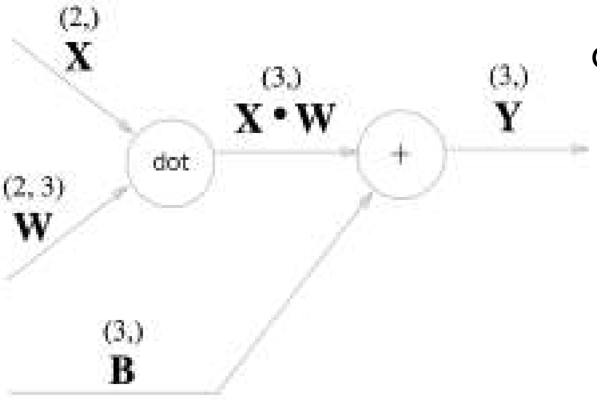
$$\Downarrow$$

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



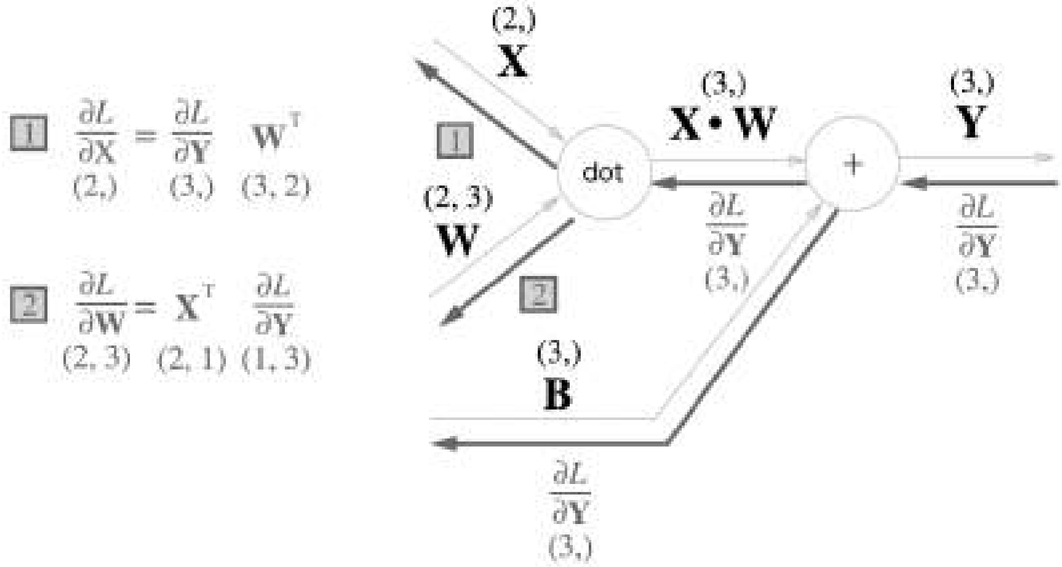
역전파

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

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

Computational Graph

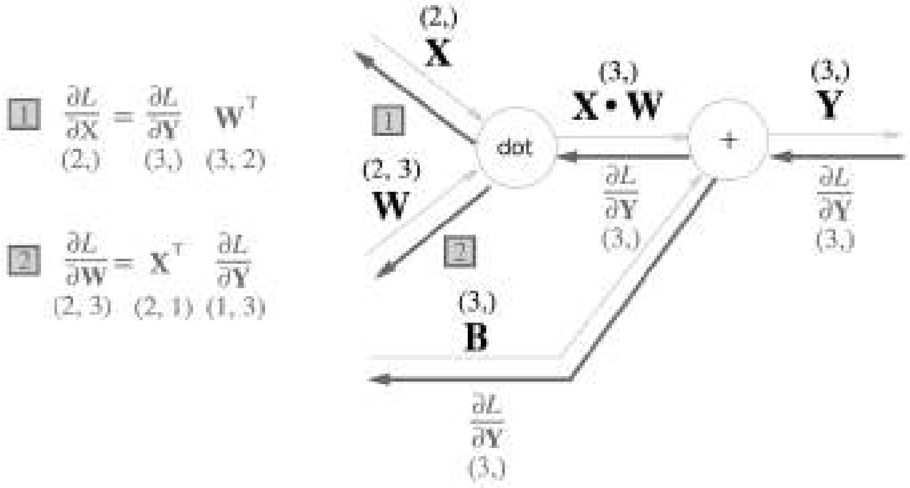
- Affine 계층



매트릭스의 형상(shape)을 잘 생각해 보면 쉽습니다

Computational Graph

- Affine 계층



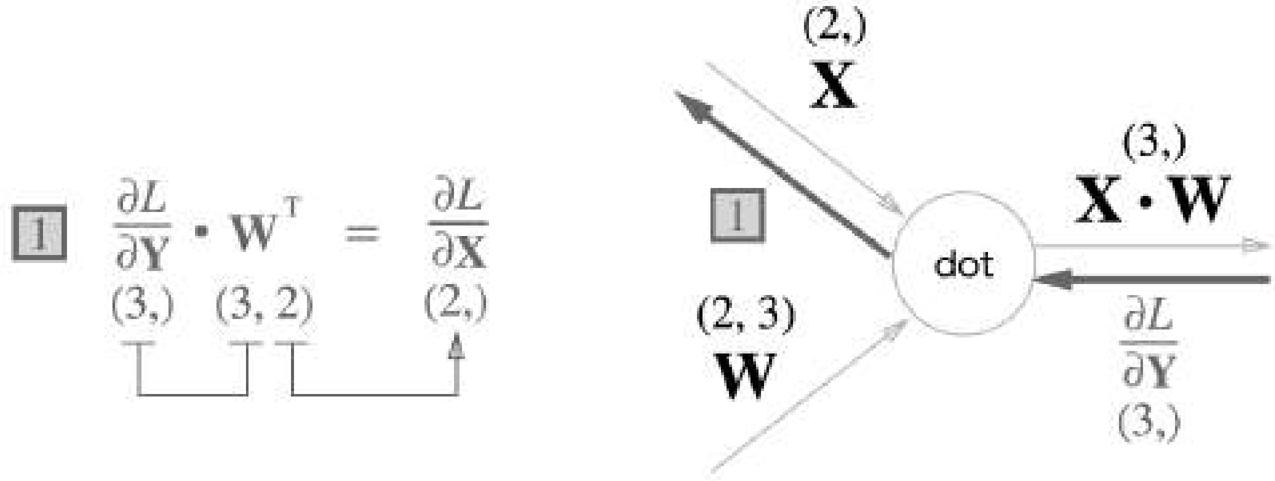
$\boldsymbol{X}$와 $\frac{\partial L}{\boldsymbol{X}}$와 형상이 같아야 하고 $\boldsymbol{W}$와 $\frac{\partial L}{\boldsymbol{W}}$는 형상이 같아야 합니다

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

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

Computational Graph

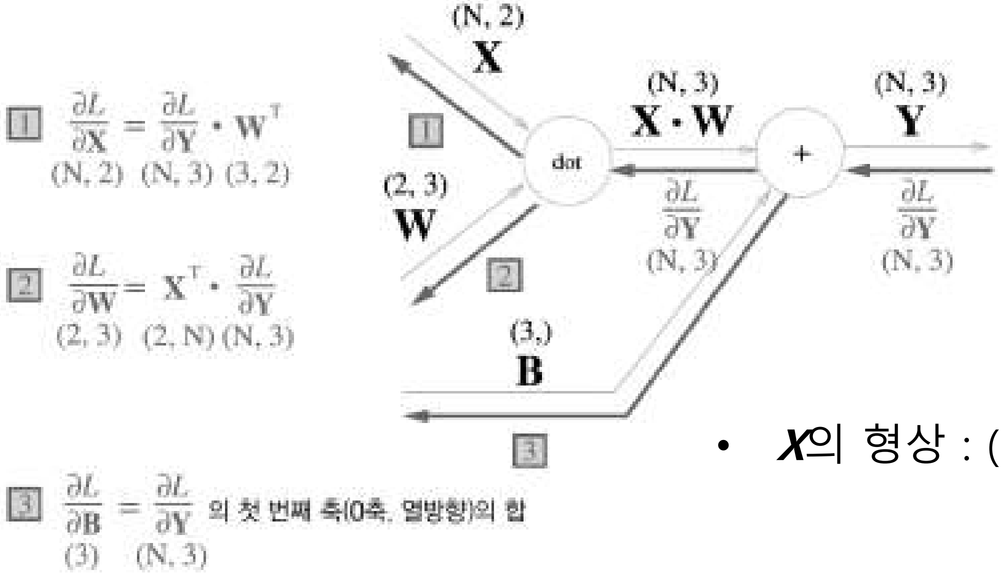
- Affine 계층



그냥 곱셈계층으로 이해하고 형상을 맞춰주면 됩니다

Computational Graph

- 배치용 Affine 계층



- *X*의 형상 : (N, 2)

- 편향에 주의하세요

Computational Graph

모두의연구소

- **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

모두의연구소

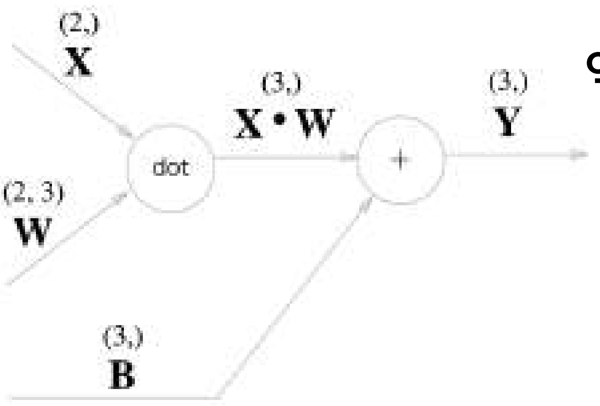
- 배치용 Affine 계층
  - 데이터가 2개 일 경우($N = 2$)의 편향은 계산된 각각의 결과에 더해집니다

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



역전파

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

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

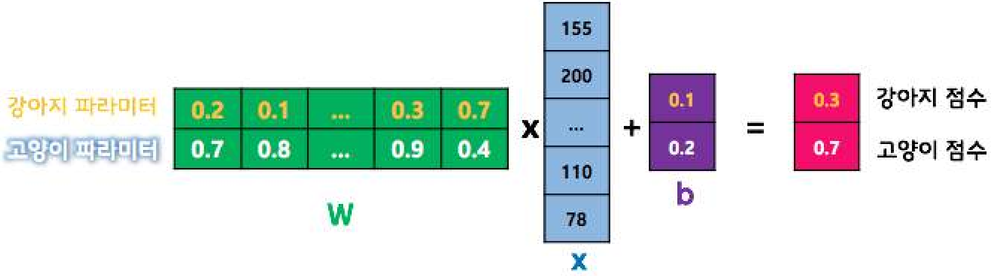
w 매트릭스가 업데이트 되겠군요

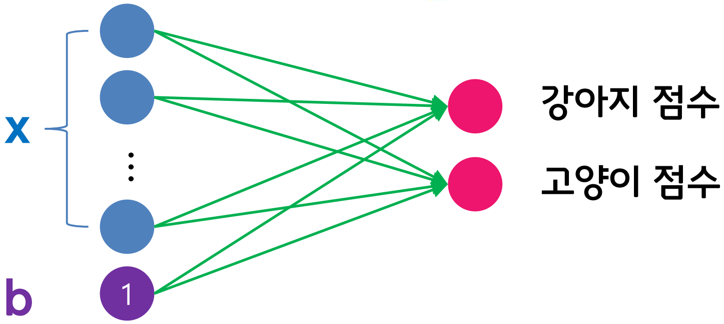
$$\mathrm{w} = \mathrm{w} - \eta \frac{\partial L}{\partial \mathrm{w}}$$

Neural Networks

Neural Networks

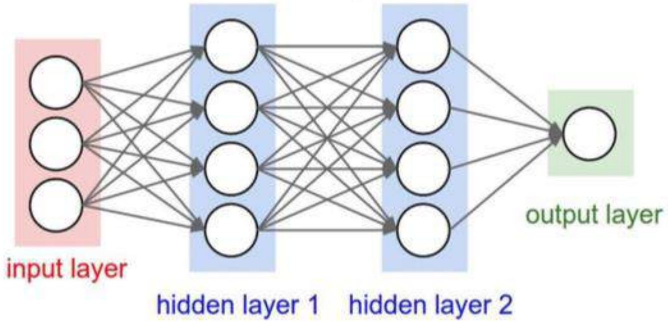
모두의연구소





Before

3 layer Network
:: 2 hidden Network

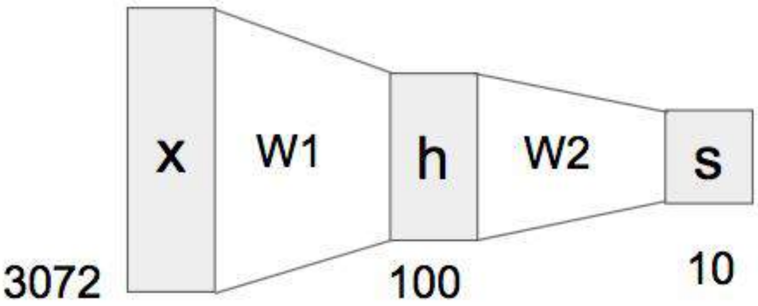


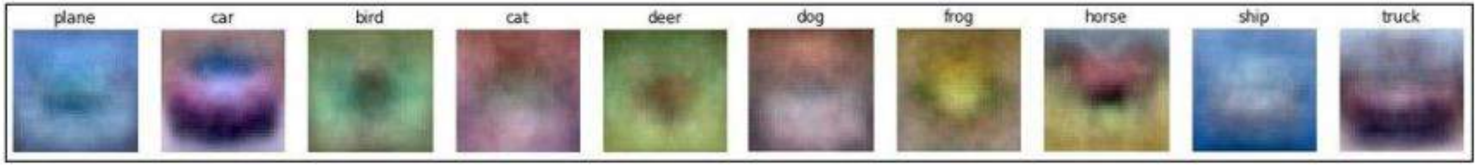
레이어를 쌓아서 더 깊게

Now

Neural Networks

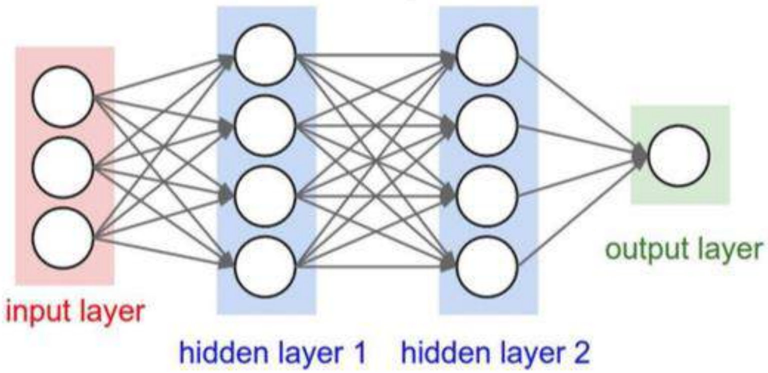
- Cifar-10 (영상크기 : 32x32x3, 10개의 클래스) 데이터에서의 첫번째 레이어의 시각화





Reference : Stanford University cs231n Lecture note 6

Neural Networks



$$output\ layer = W_2(W_1\mathrm{x}) = W_2W_1\mathrm{x} = \mathrm{Wx}$$

레이어를 하나 쌓는 것
과 같음

비선형 Activation function
이 필요

Neural Networks

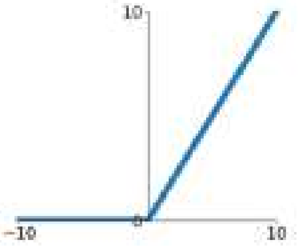
(Before) Linear score function: $f = Wx$

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

0보다 작으면0으로 처리

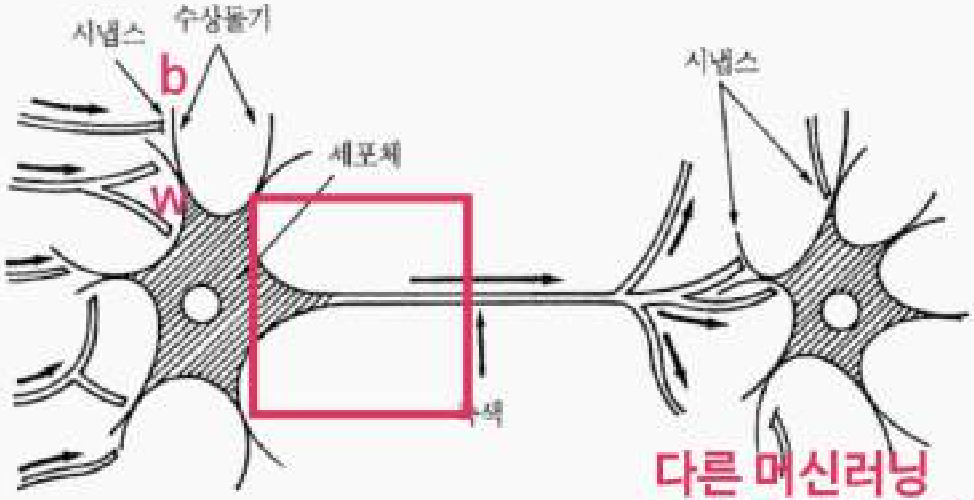
or 3-layer Neural Network

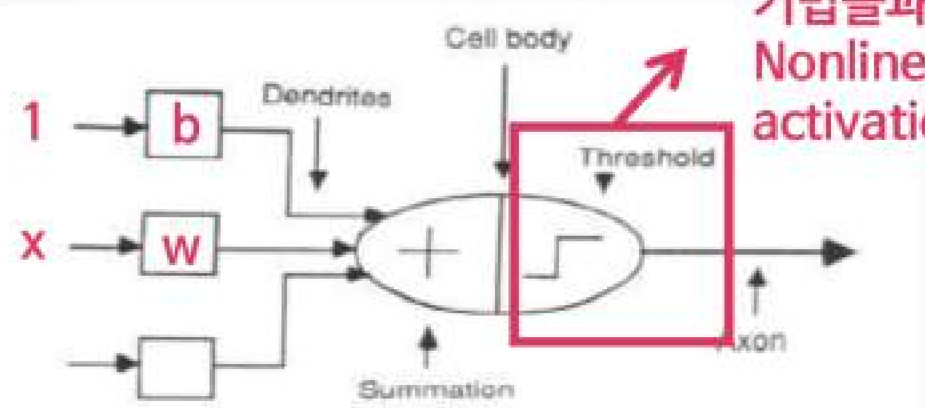
$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$



ReLU
(Rectified Linear Unit)

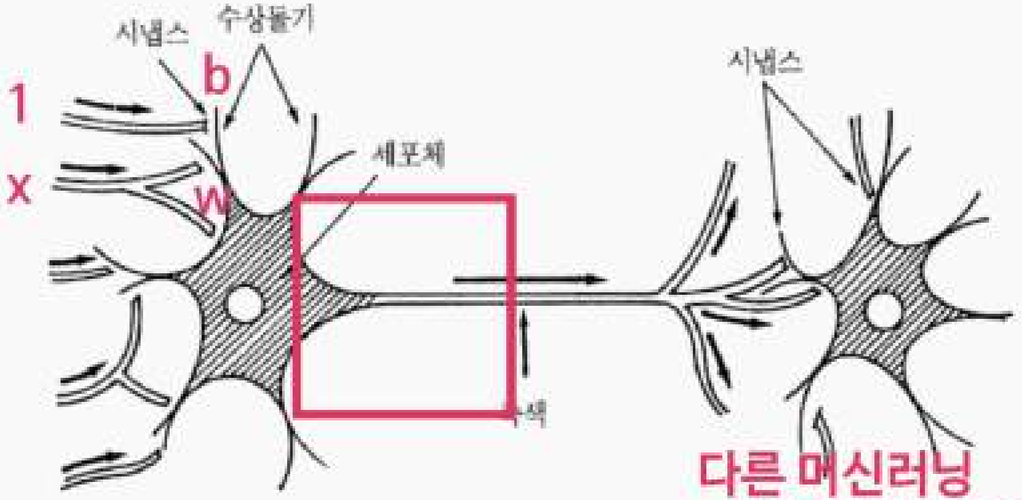
뉴런과 인공뉴런



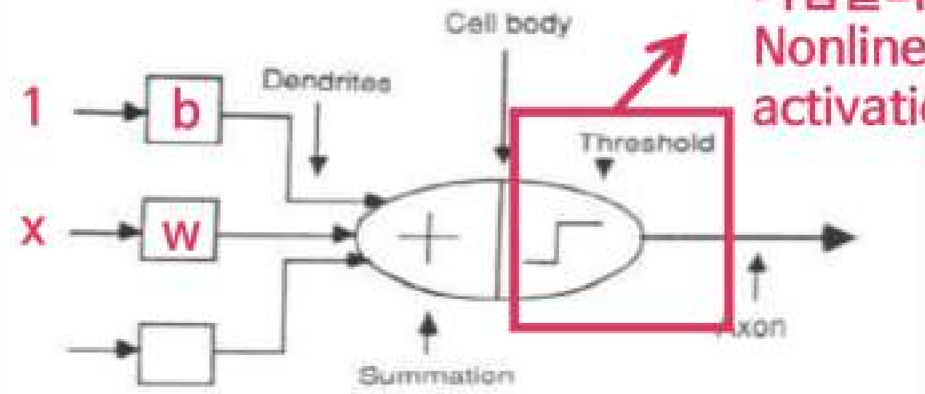


다른 머신러닝
기법들과의 차이점 1:
Nonlinear (복잡한)
activation function

뉴런과 인공뉴런



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

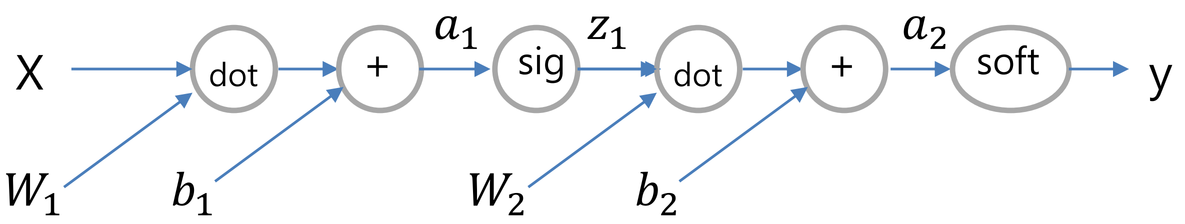


완전히 단순화 시킨 것이니 주의하세요

2레이어 네트워크의 구현 예

- **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)
```



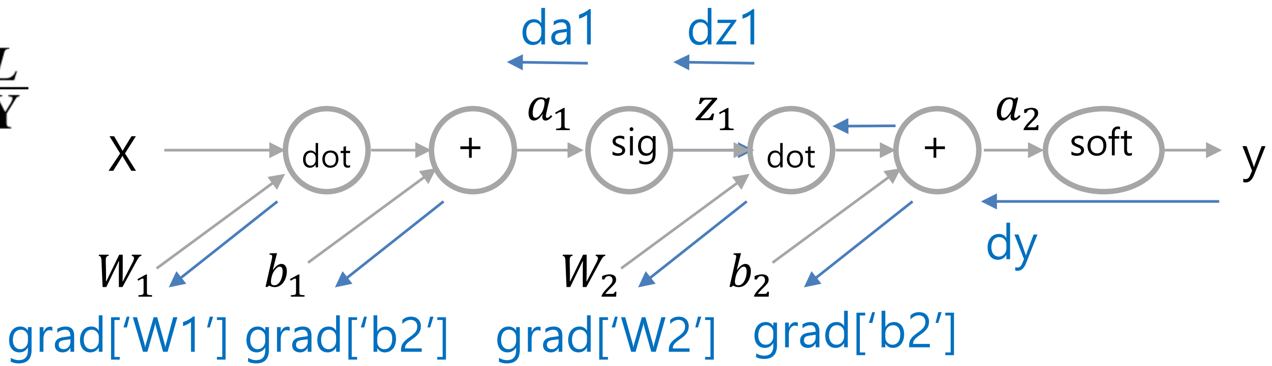
2레이어 네트워크의 구현 예

모두의연구소

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

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

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

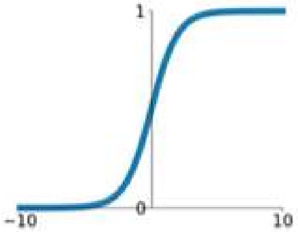


Activation function

- ReLU vs. Sigmoid

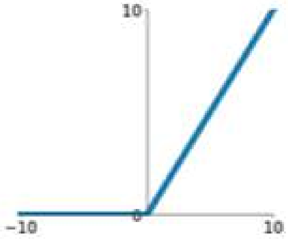
Sigmoid

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



ReLU

$$\max(0, x)$$



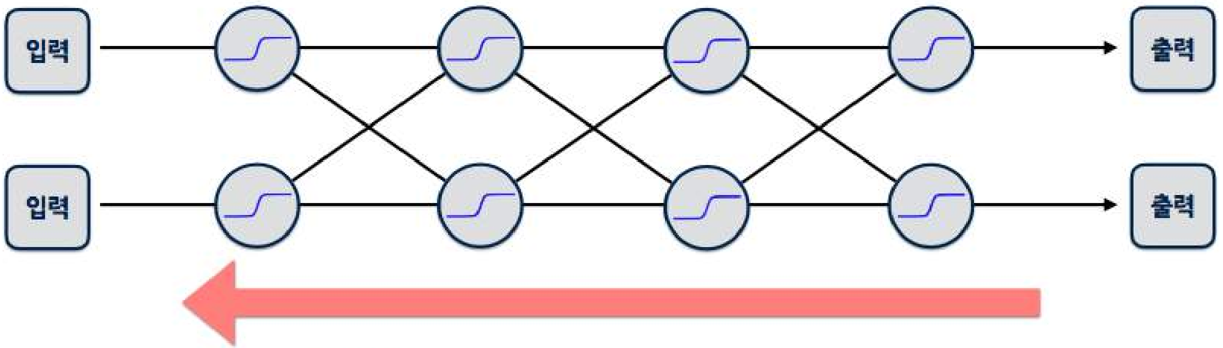
뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 "뒤로 전달")

모두의연구소

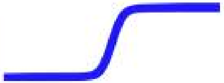
뭐를 전달하는가?

현재 내가 틀린정도를 '미분(기울기)' 한 거

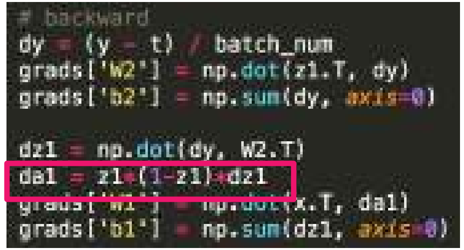


미분하고, 곱하고, 더하고를 역방향으로 반복하며 업데이트한다.

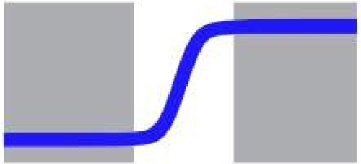
근데 문제는?

우리가 activation 함수로 sigmoid  를 썼다는 것

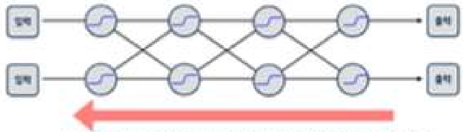
여기의 미분(기울기)는 뭐라도 있다. 다행



```
# 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?!**



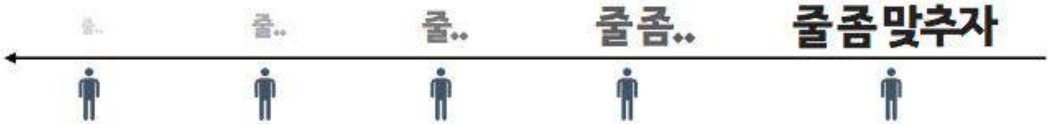
미분하고, 곱하고, 더하고를 역방향으로 반복

그런 상황에서 이걸 반복하면??????

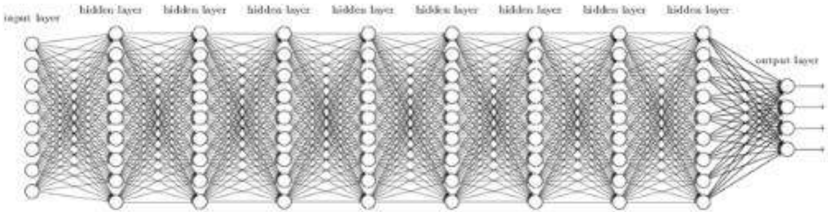
Vanishing gradient 현상 : 레이어가 깊을 수록 업데이트가 사라져간다. 그래서 fitting이 잘 안됨(underfitting)

모두의연구소

끝줄학생

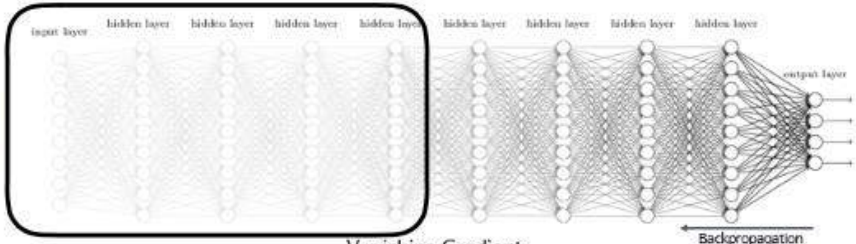


교장샘



Deep Neural Network

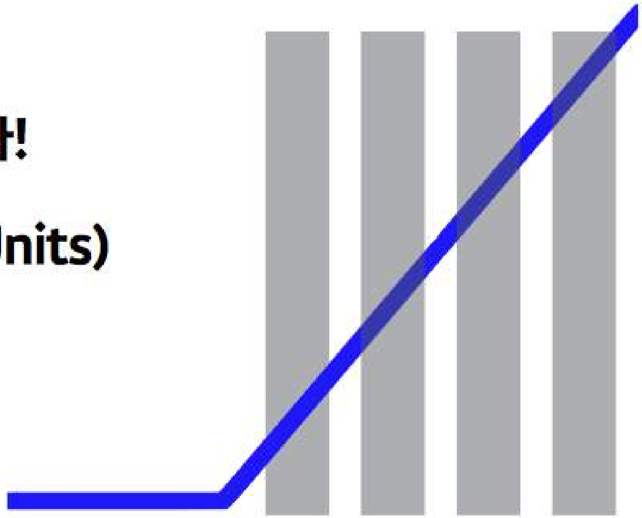
학습이 잘 안됨



Vanishing Gradient

사그라드는 sigmoid대신
죽지않는 activation func을 쓰자!

→ **ReLU** (Rectified Linear Units)



이녀석은 양의 구간에서 전부 미분값(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

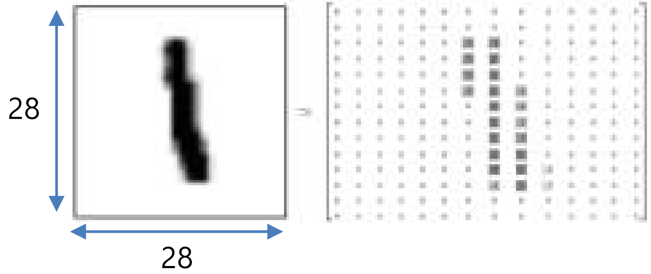
3 레이어 구조

	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)
```

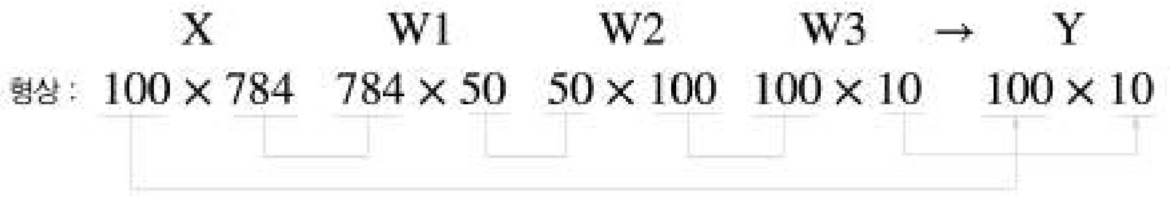


28x28x1 MNIST 데이터

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

모두의연구소

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

- 이미지 100개를 묶어서 한번에 넘기기

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

- 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

코드와 함께 보는 신경망 관련 계층

활성화 함수계층

- ReLU 계층

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

⬇

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


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

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

        return out

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

        return dx
```

- code/common/layers.py

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.]]
```

- common/layers.py

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

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

        return out

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

        return dx
```

활성화 함수계층

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

  - 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

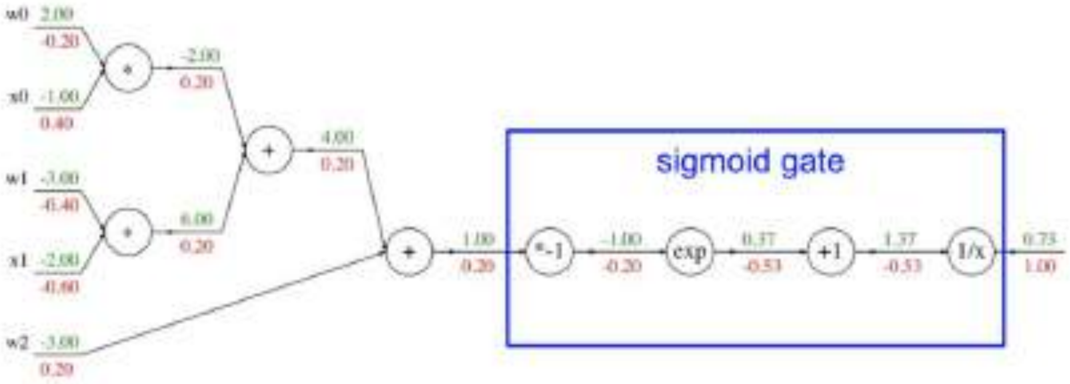
matrix 곱 아님.

활성화 함수계층 구현하기

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

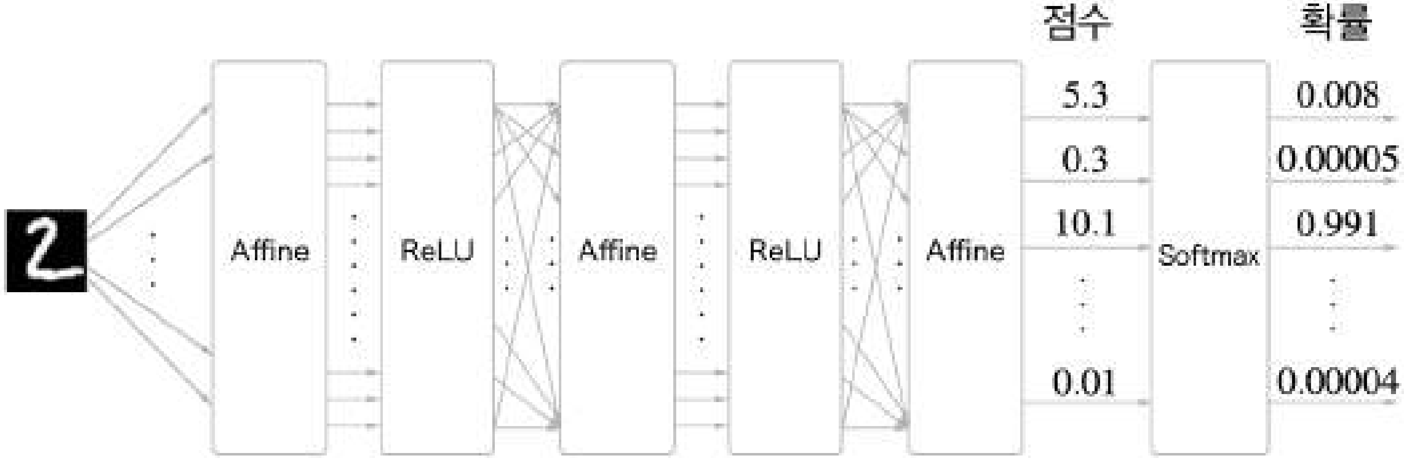
$$\boxed{\sigma(x) = \frac{1}{1+e^{-x}}}$$ sigmoid function

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



Softmax 계층 구현하기

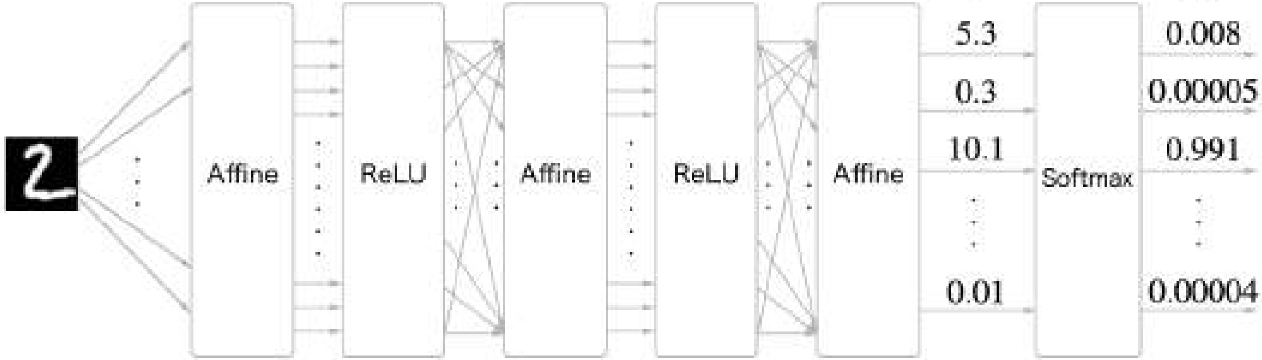
- Softmax-with-Loss 계층



입력이미지가 Affine계층과 ReLU를 통과하며 변환되고, 마지막 Softmax 계층에 의해서 10개의 입력이 정규화된다.
이 그림에서 숫자 '0'의 점수는 5.3이며, 이것이 Softmax계층에 의해서 0.008(0.8%)로 변환된다.
또 '2'의 점수는 10.1에서 0.991(99.1%)로 변환된다.

Softmax 계층 구현하기

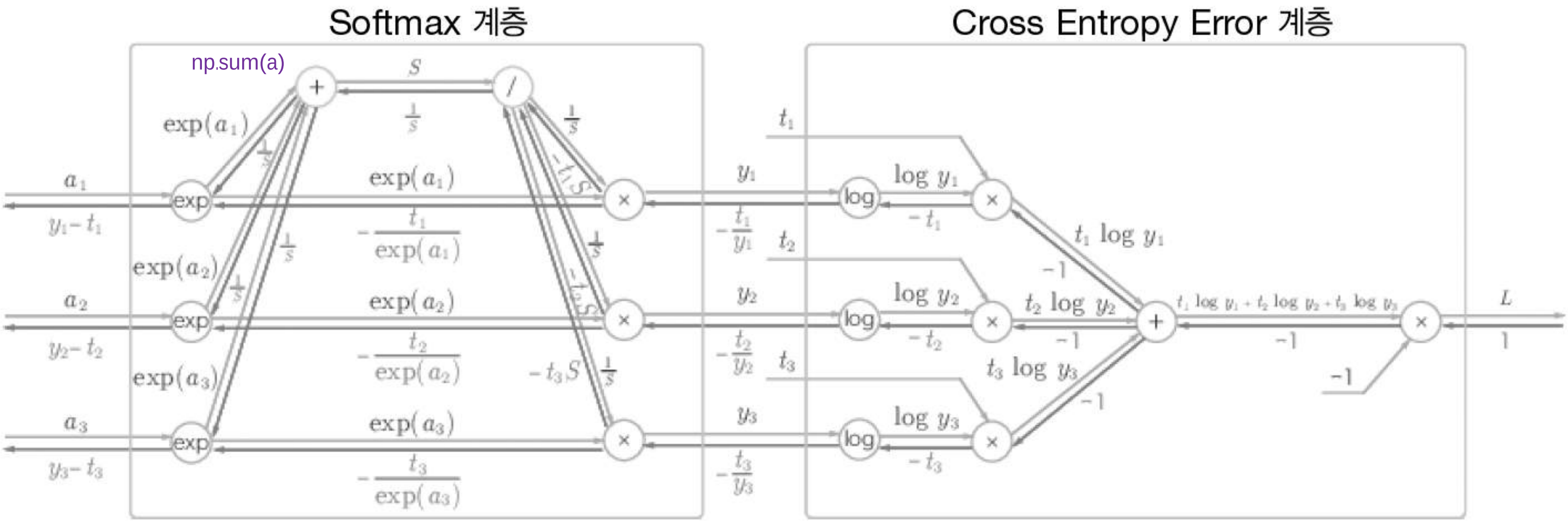
- Softmax-with-Loss 계층



Note : 신경망에서 수행하는 작업은 **학습**과 **추론** 두 가지 입니다. 추론할 때는 일반적으로 Softmax 계층을 사용하지 않습니다. 위 그림에서 신경망은 추론할때 마지막 Affine 계층의 출력을 인식결과로 이용합니다. 또한, 신경망에서 정규화하지 않는 출력 결과 (Softmax 앞의 Affine계층의 출력)를 **점수**(Score)라 합니다. 즉, 신경망 추론에서 답을 하나만 내는 경우에는 가장 높은 점수만 알면 되니 Softmax 계층은 필요 없다는 것이죠. 반면, 신경망을 학습할 때는 Softmax계층이 필요합니다.

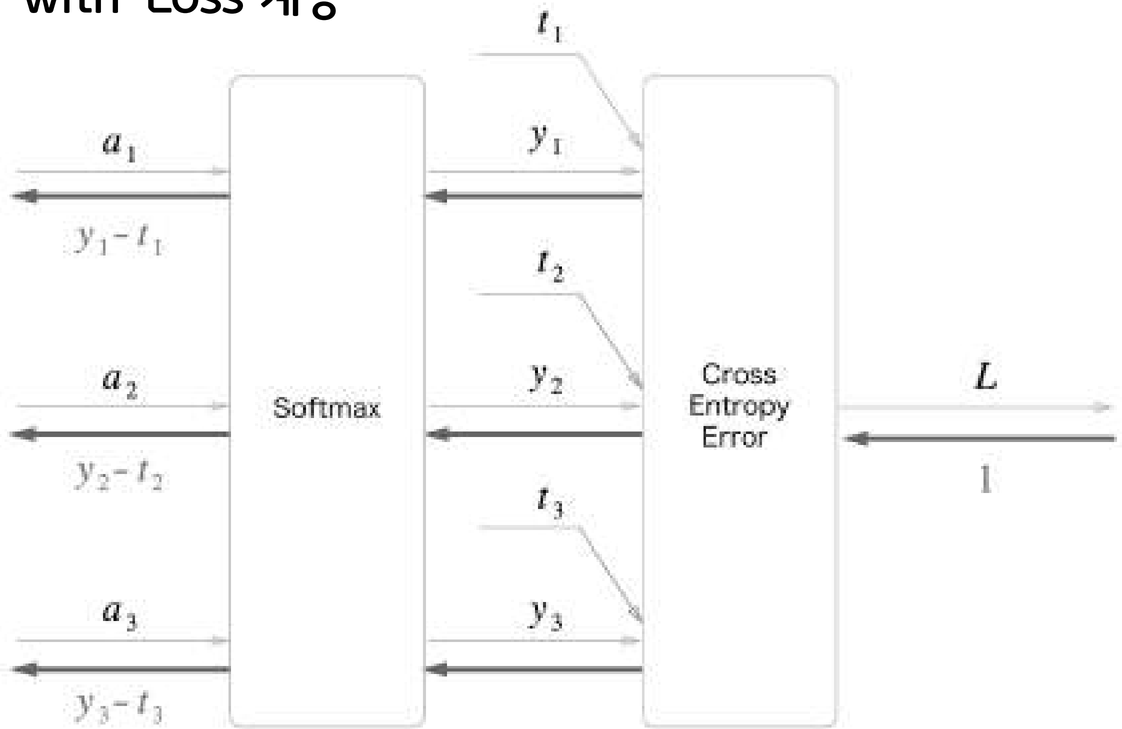
Softmax 계층 구현하기

- Softmax-with-Loss 계층



Softmax 계층 구현하기

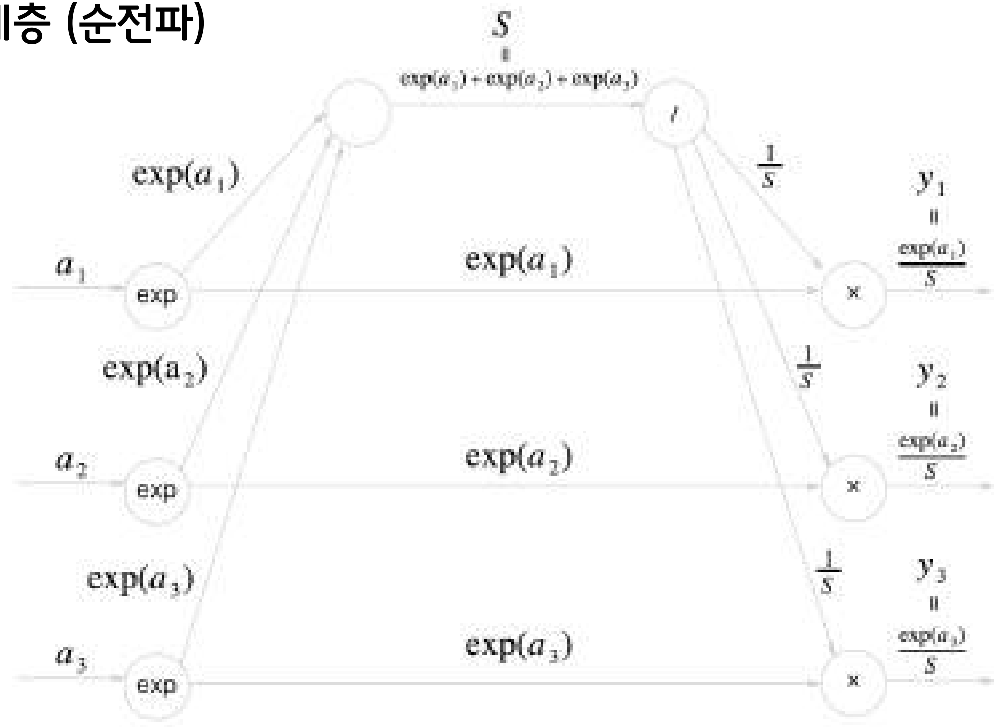
- Softmax-with-Loss 계층



Softmax 계층 구현하기

- Softmax-with-Loss 계층

Softmax계층 (순전파)



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

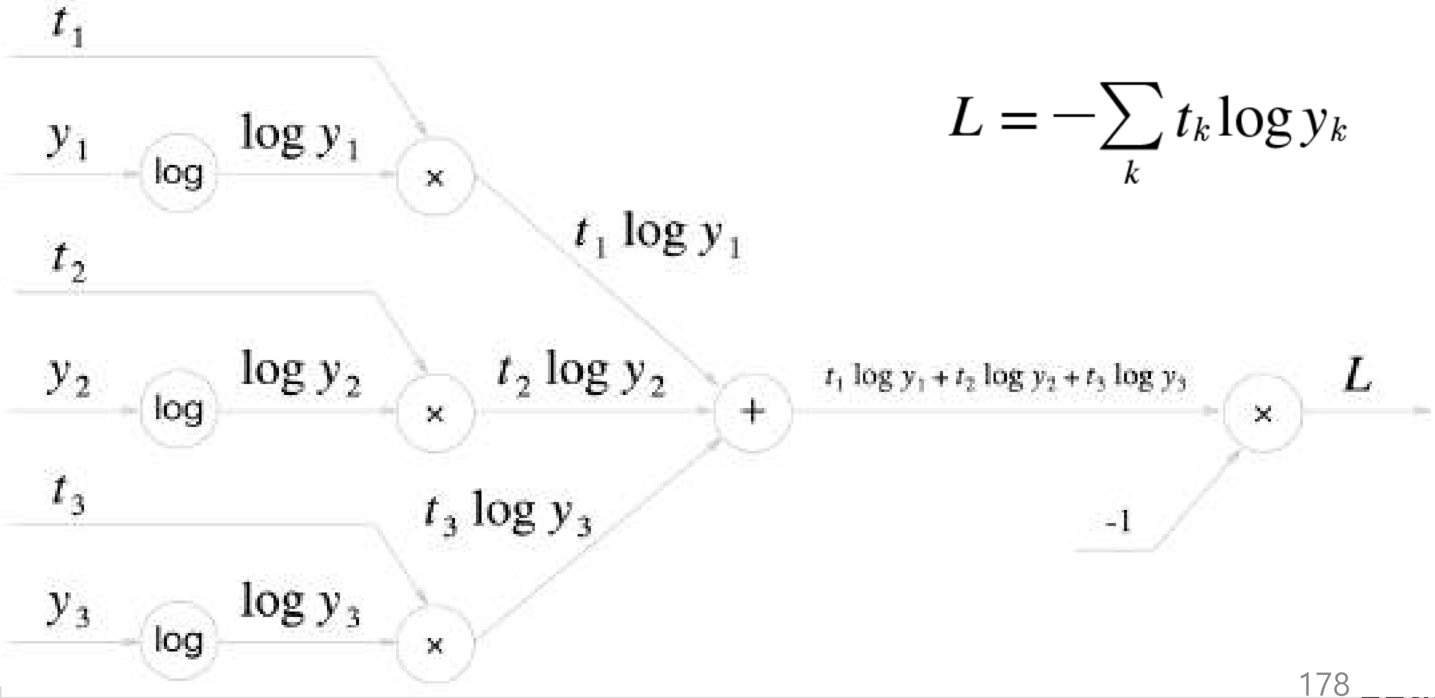
Softmax 계층 구현하기

모두의연구소

- Softmax-with-Loss 계층

Cross Entropy Error 계층 (순전파)

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



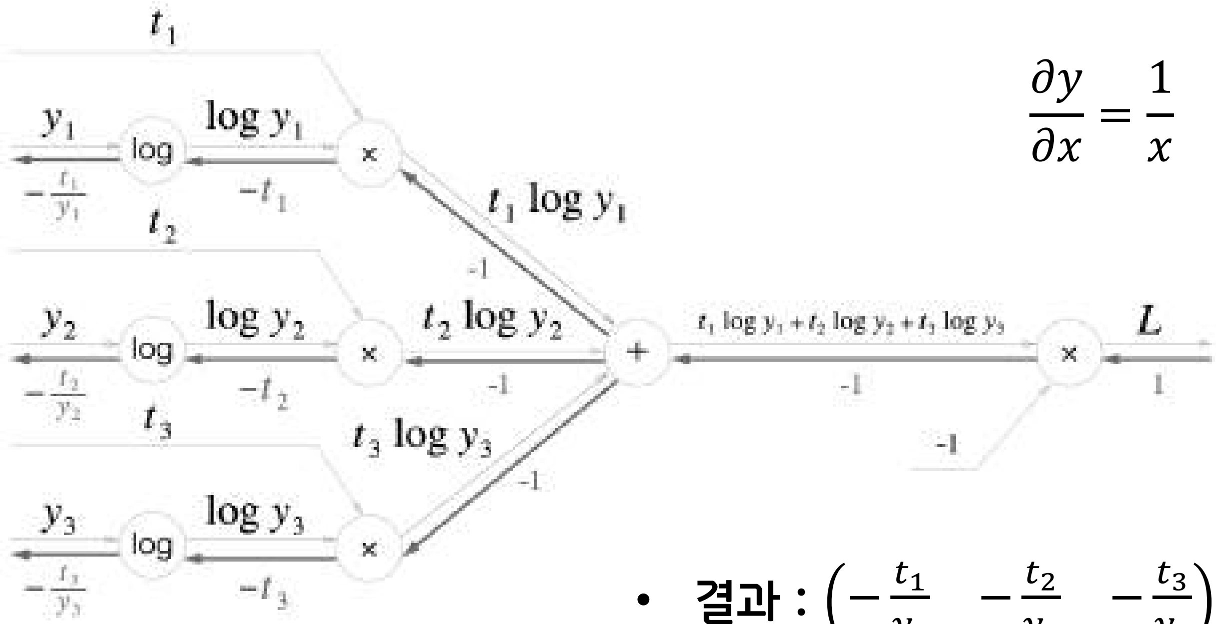
Softmax 계층 구현하기

- Softmax-with-Loss 계층

Cross Entropy Error 계층 (역전파)

$$y = \log x$$

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

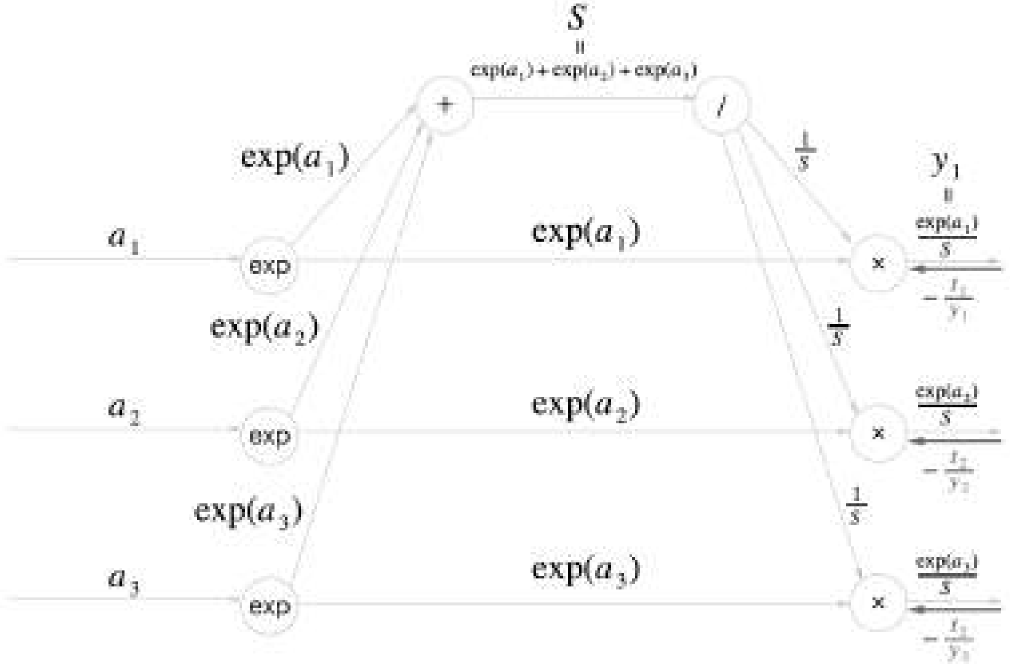


- 결과 : $\left(-\frac{t_1}{y_1} \quad -\frac{t_2}{y_2} \quad -\frac{t_3}{y_3}\right)$

Softmax 계층 구현하기

- Softmax-with-Loss 계층

Softmax 계층 (역전파)

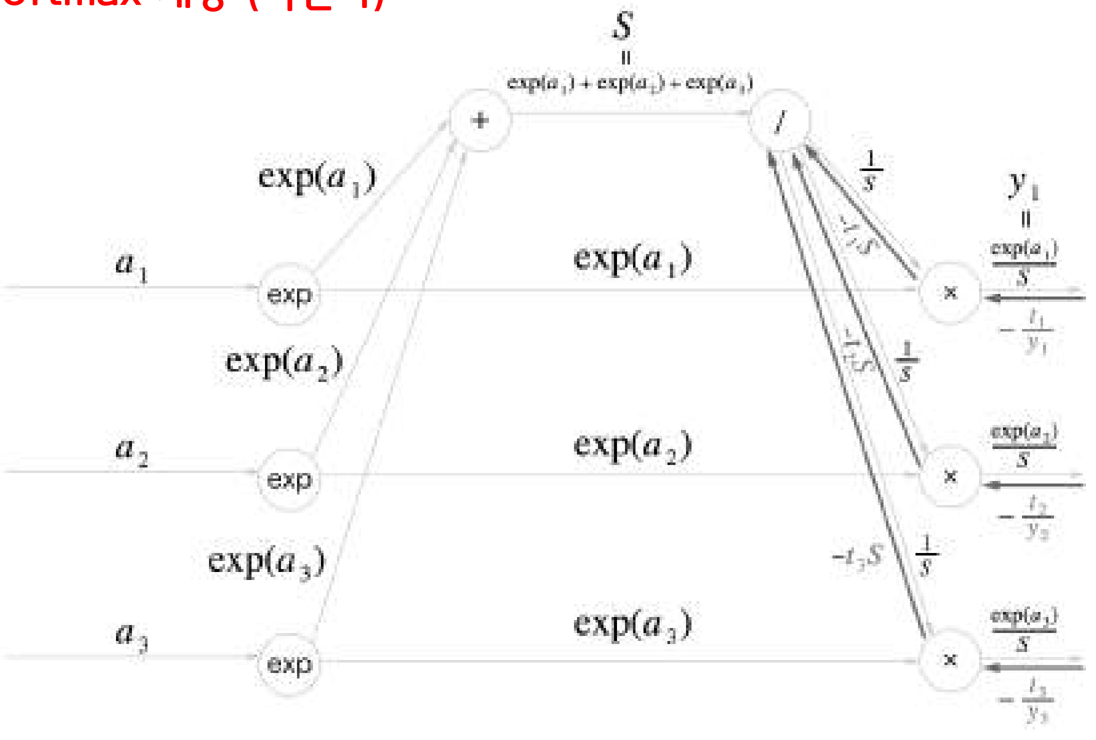


Softmax 계층 구현하기

- Softmax-with-Loss 계층

Softmax 계층 (역전파)

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



Softmax 계층 구현하기

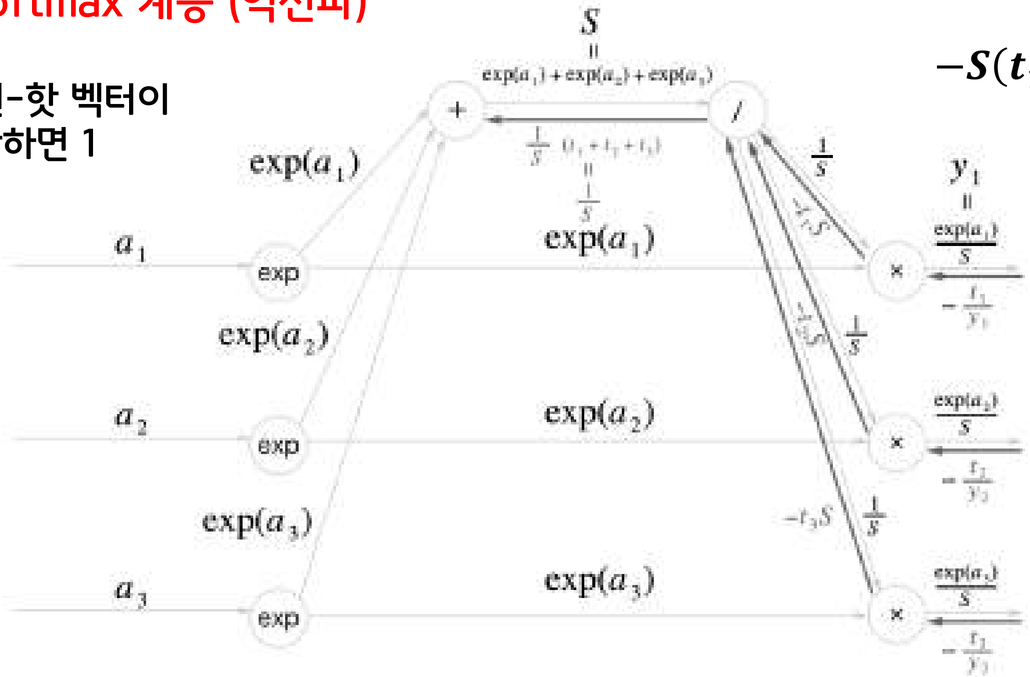
- Softmax-with-Loss 계층

Softmax 계층 (역전파)

$(-t_1S) + (-t_2S) + (-t_3S) = -S(t_1 + t_2 + t_3)$

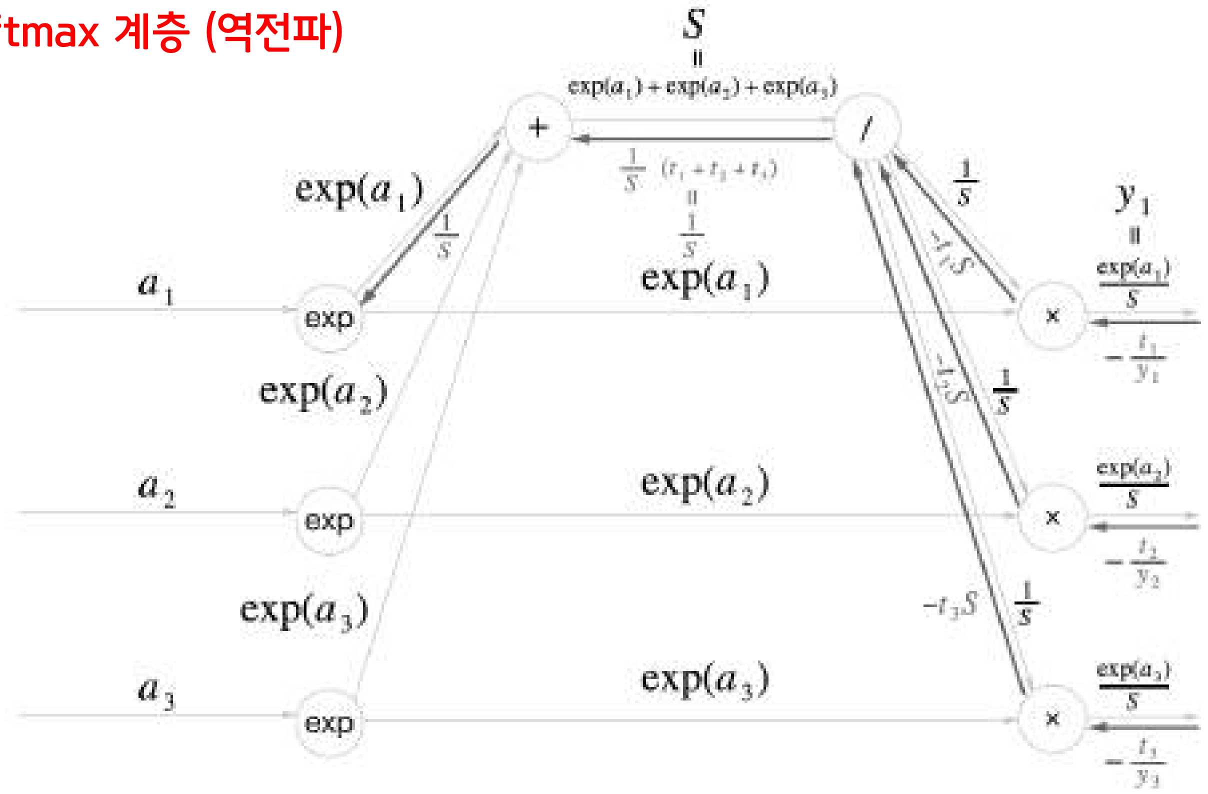
$$-S(t_1 + t_2 + t_3) * -\frac{1}{S^2}$$

$(t_1 + t_2 + t_3)$는 원-핫 벡터이기 때문에 계산하면 1



Softmax 계층 구현하기

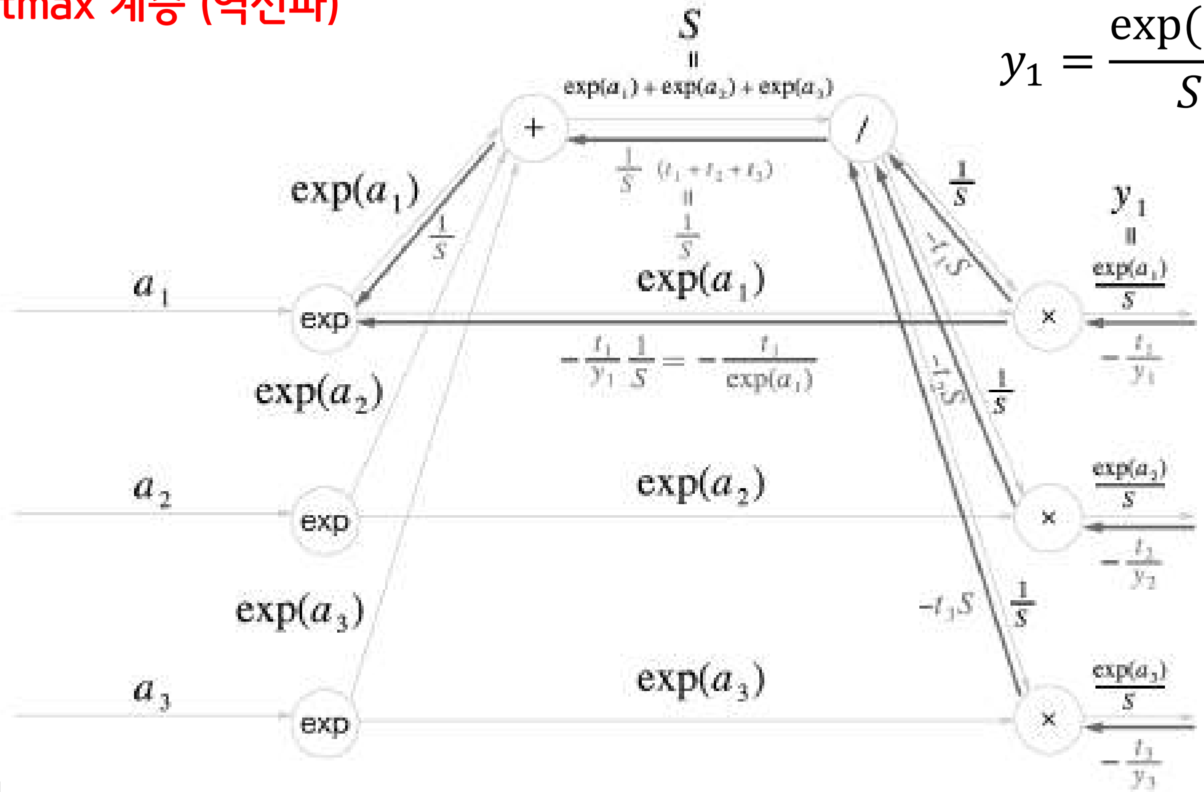
Softmax 계층 (역전파)



Softmax 계층 구현하기

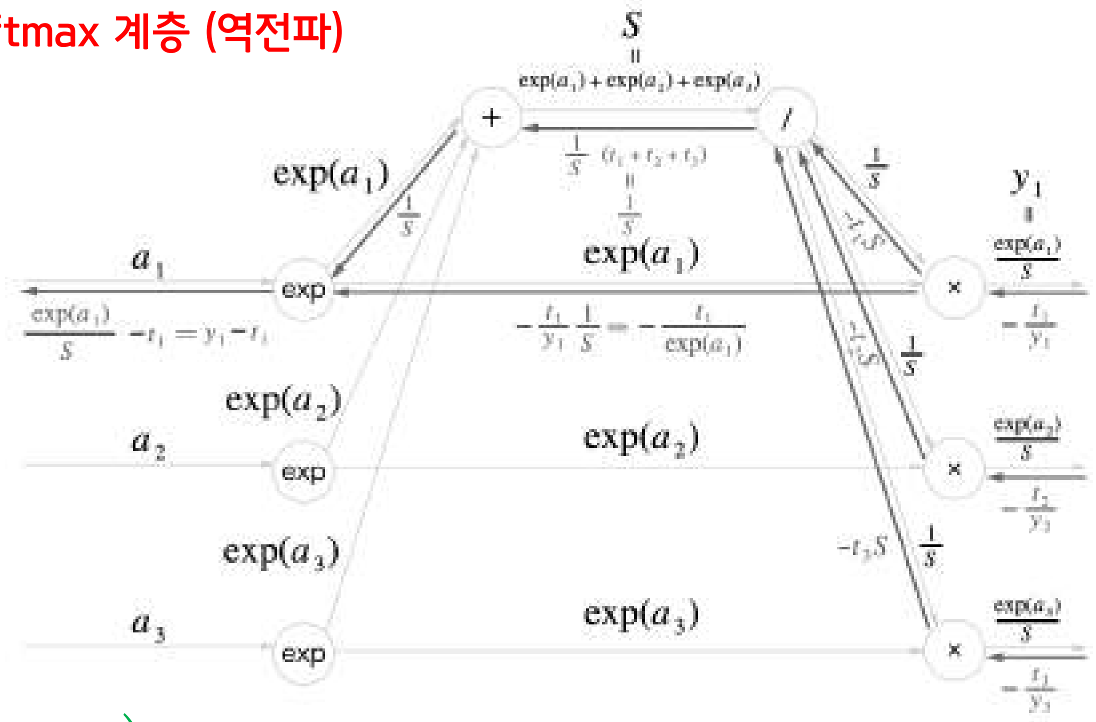
Softmax 계층 (역전파)

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



Softmax 계층 구현하기

Softmax 계층 (역전파)



$y = \exp(x)$

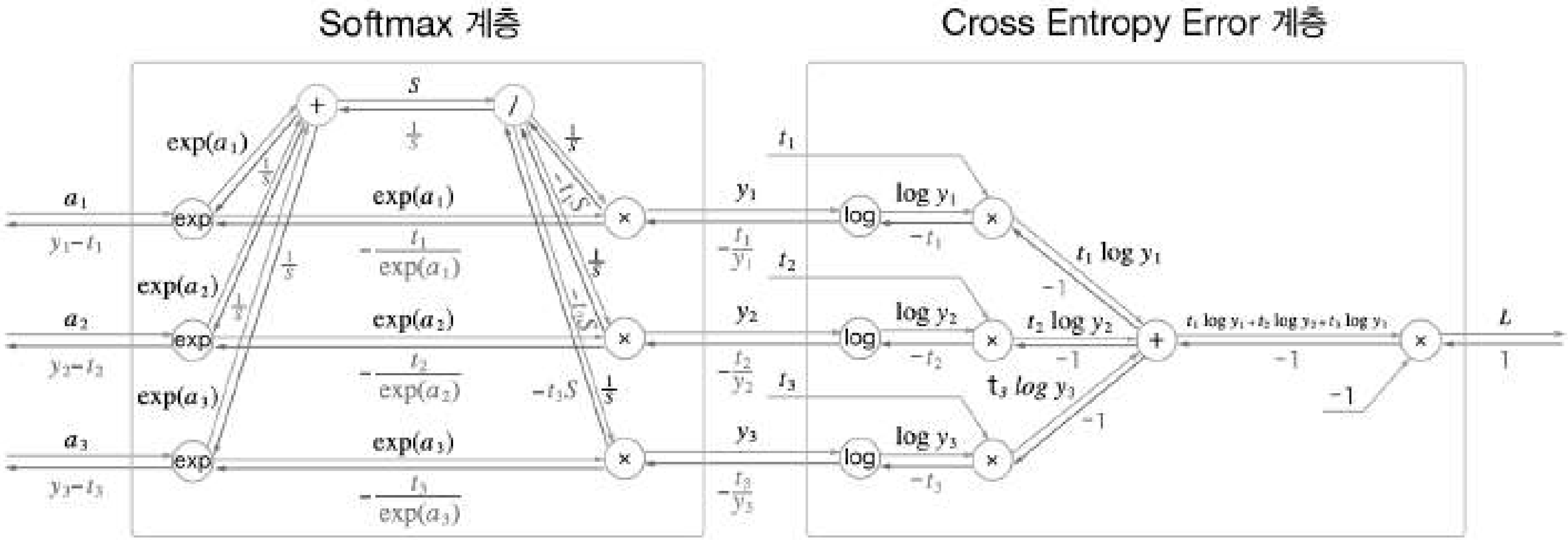
$\frac{\partial y}{\partial x} = \exp(x)$

Up-stream

$$\left(\frac{1}{S} - \frac{t_1}{\exp(a_1)}\right)\exp(a_1) = y_1 - t_1$$

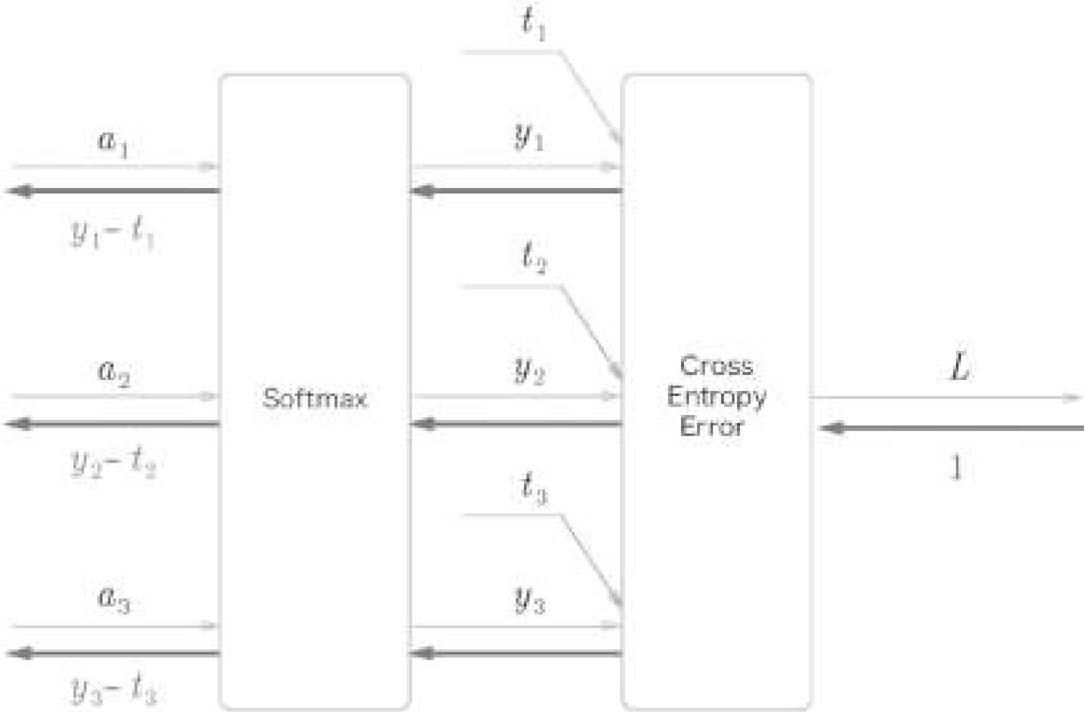
Softmax 계층 구현하기

Softmax-with-Loss 계층 정리



Softmax 계층 구현하기

간소화한 Softmax-with-Loss



Softmax 계층 구현하기

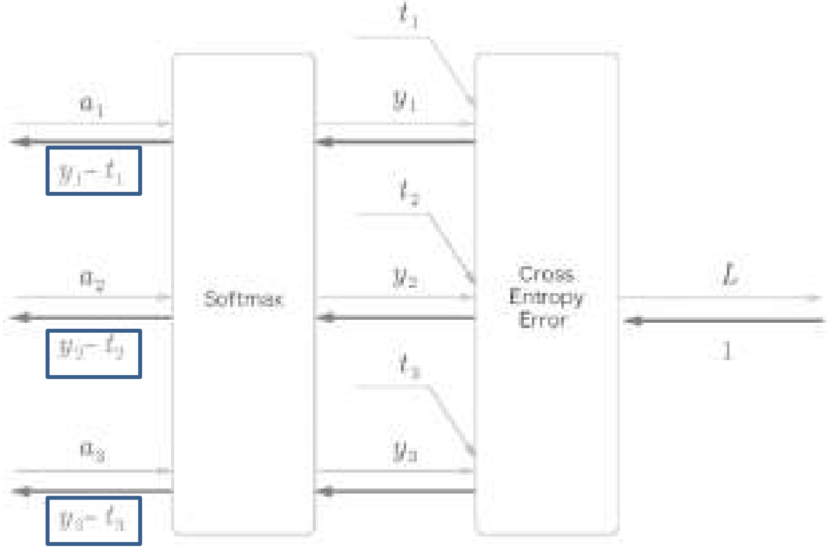
간소화한 Softmax-with-Loss

정답 : (0,1,0)
Softmax출력 : (0.3, 0.2, 0.5)
역전파 : (0.3, -0.8, .0.5)

- 커다란 오차를 전파함

정답: (0,1,0)
Softmax출력: (0.01, 0.99, 0)
역전파 : (0.01, -0.01, 0)

- 오차가 작고 학습하는 정도도 작아지게 됨



실습 1 : 2 레이어 네트워크 실습

모두의연구소

- 1) 실행 파일 (손글씨 인식) : 1_day/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.numerical_gradient(x_batch, t_batch)
    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 : 1_day/prac2_simple_two_Layer_net.py

```python
    def gradient(self, x, t):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']
        grads = {}

        batch_num = x.shape[0]

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

        # backward
        dy = (y - t) / batch_num
        ###################################
        ######### Write your codes #######
        ###################################
        grads['W2'] = None
        grads['b2'] = None

        da1 = None
        dz1 = None
        grads['W1'] = None
        grads['b1'] = None

        return grads
```

실습 1 : 2 레이어 네트워크 실습

모두의연구소

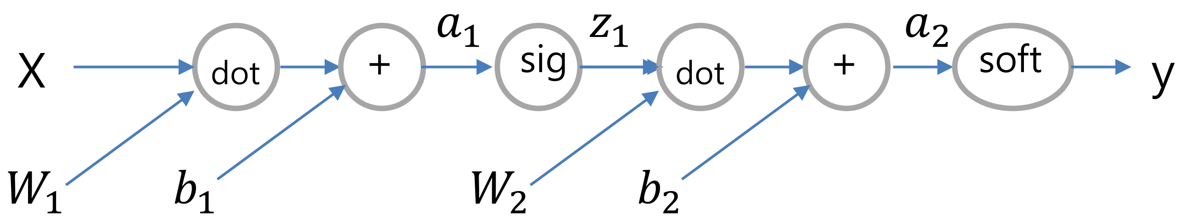
- 2) 구현할 함수 : 1_day/prac2_simple_two_Layer_net.py
  - 순전파를 계산 그래프로 표현해 봅시다

```
    def gradient(self, x, t):
        W1, W2 = self.params['W1'], self.params['W2']
        b1, b2 = self.params['b1'], self.params['b2']
        grads = {}

        batch_num = x.shape[0]

        # 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 : 1_day/prac2_simple_two_Layer_net.py

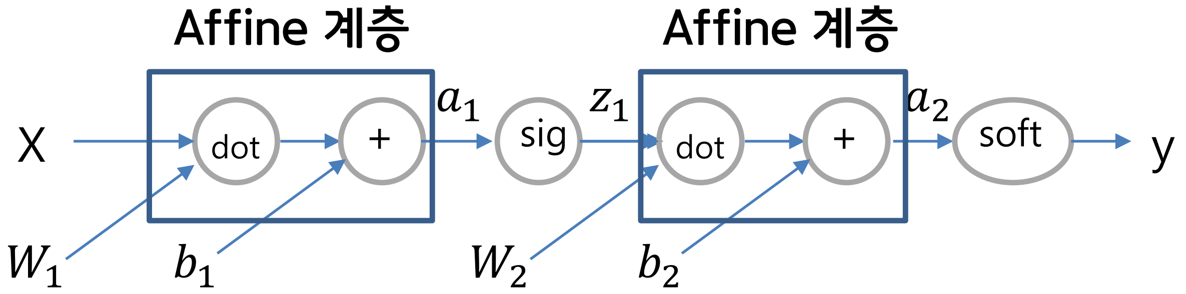
```python
        # forward
        a1 = np.dot(x, W1) + b1
        z1 = sigmoid(a1)
        a2 = np.dot(z1, W2) + b2
        y = softmax(a2)
```



실습 1 : 2 레이어 네트워크 실습

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

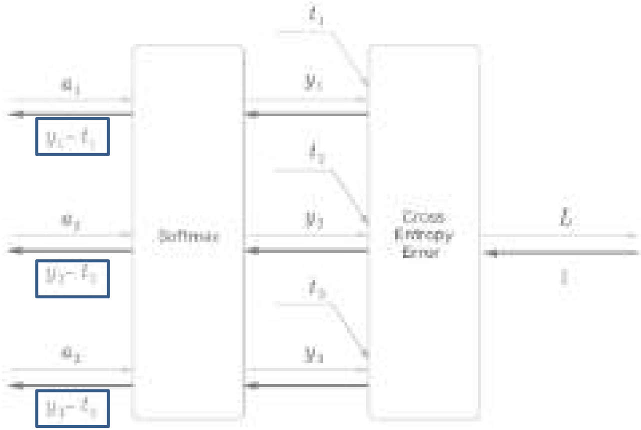
```
# forward
a1 = np.dot(x, W1) + b1
z1 = sigmoid(a1)
a2 = np.dot(z1, W2) + b2
y = softmax(a2)
```



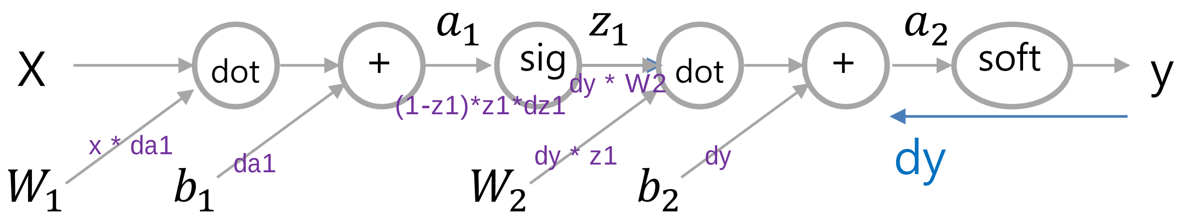
실습 1 : 2 레이어 네트워크 실습

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

```
    # backward
    dy = (y - t) / batch_num
    ####################################
    ######### Write your codes #######
    ####################################
    grads['W2'] = None
    grads['b2'] = None

    da1 = None
    dz1 = None
    grads['W1'] = None
    grads['b1'] = None

    return grads
```



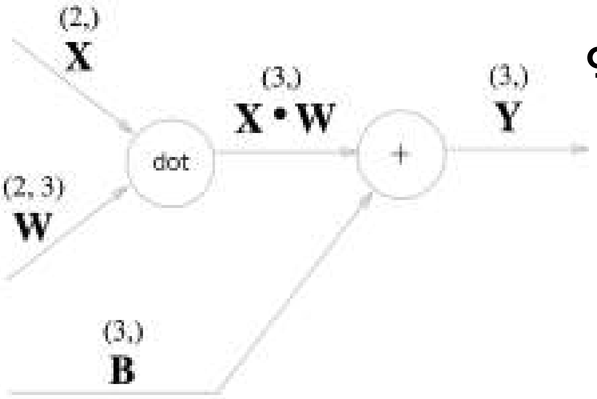
Softmax loss 부분은 이미 미분 해 두었습니다



실습 1 : 2 레이어 네트워크 실습

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

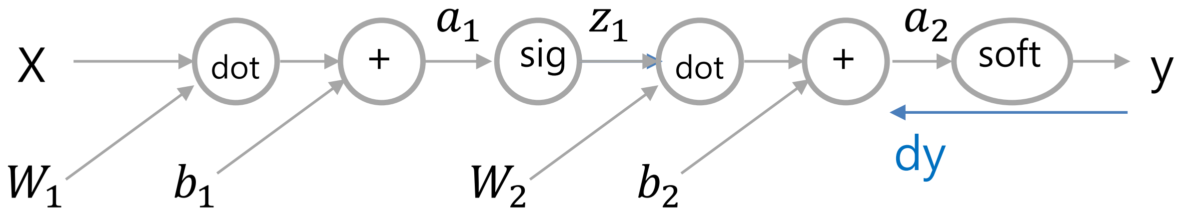
Affine 계층의 역전파를 기억해 봅시다



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

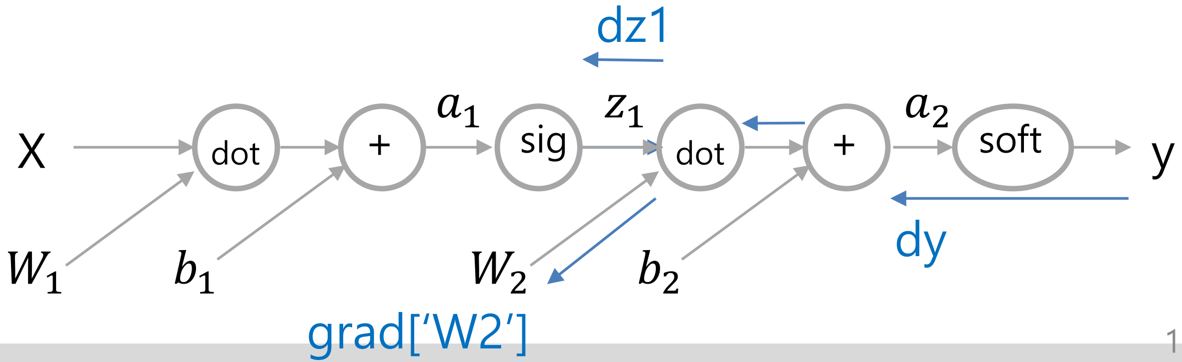
- 2) 구현할 함수 : 1_day/prac2_simple_two_Layer_net.py

Affine 계층의 역전파를 기억해 봅시다

```python
    # backward
    dy = (y - t) / batch_num
    ####################################
    ######### Write your codes #######
    ####################################
    grads['W2'] = None
    grads['b2'] = None

    dz1 = None
    da1 = None
    grads['W1'] = None
    grads['b1'] = None
```

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

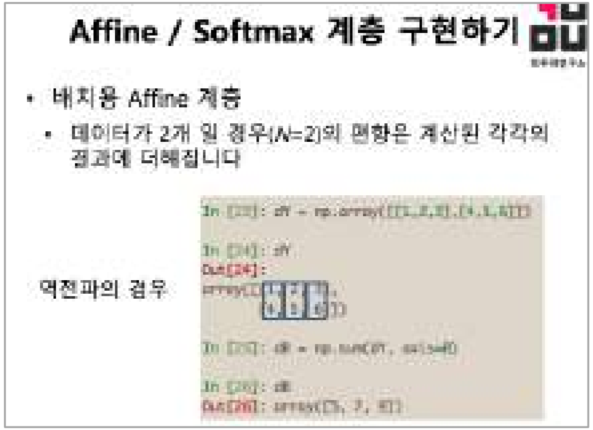
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^{\mathrm{T}} \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

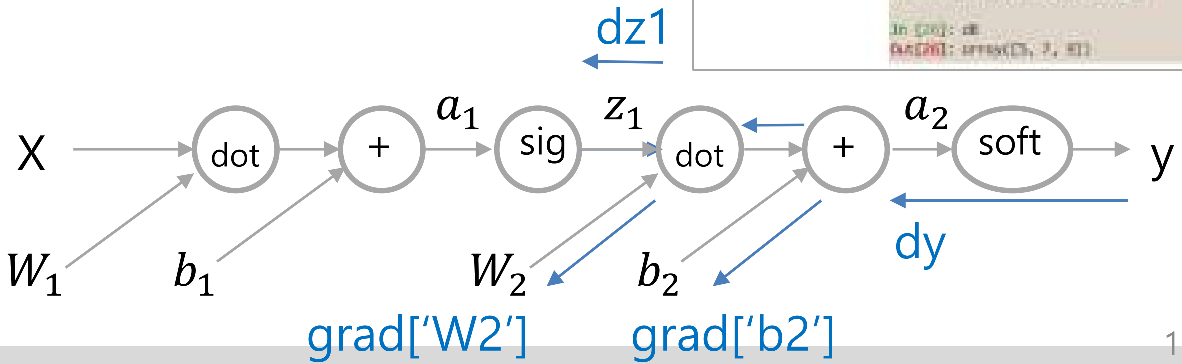


실습 1 : 2 레이어 네트워크 실습

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

```python
    # backward
    dy = (y - t) / batch_num
    ##################################
    ######### Write your codes #######
    ##################################
    grads['W2'] = None
    grads['b2'] = None

    dz1 = None
    da1 = None
    grads['W1'] = None
    grads['b1'] = None
```

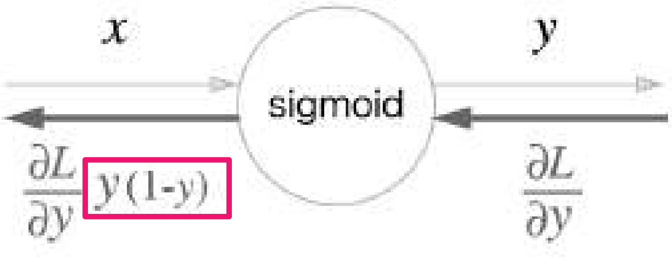


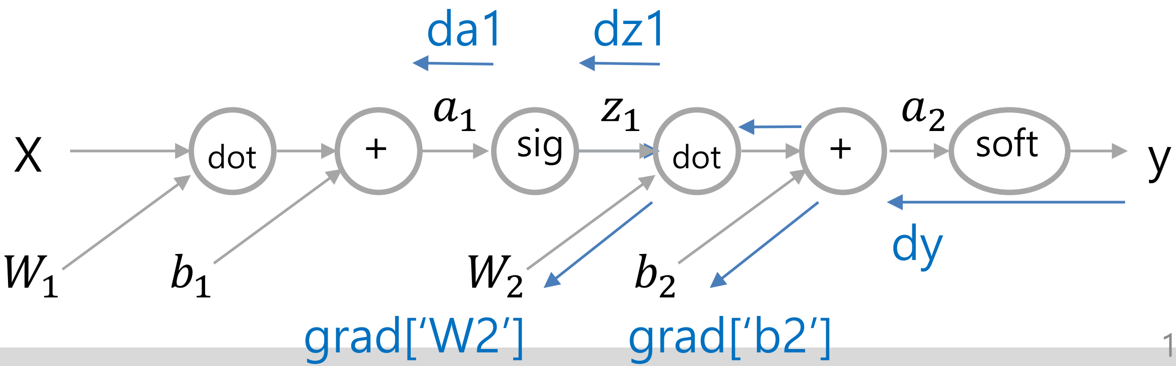


실습 1 : 2 레이어 네트워크 실습

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

```
# backward
dy = (y - t) / batch_num
####################################
######### Write your codes #######
####################################
grads['W2'] = None
grads['b2'] = None

dz1 = None
da1 = None
grads['W1'] = None
grads['b1'] = None
```





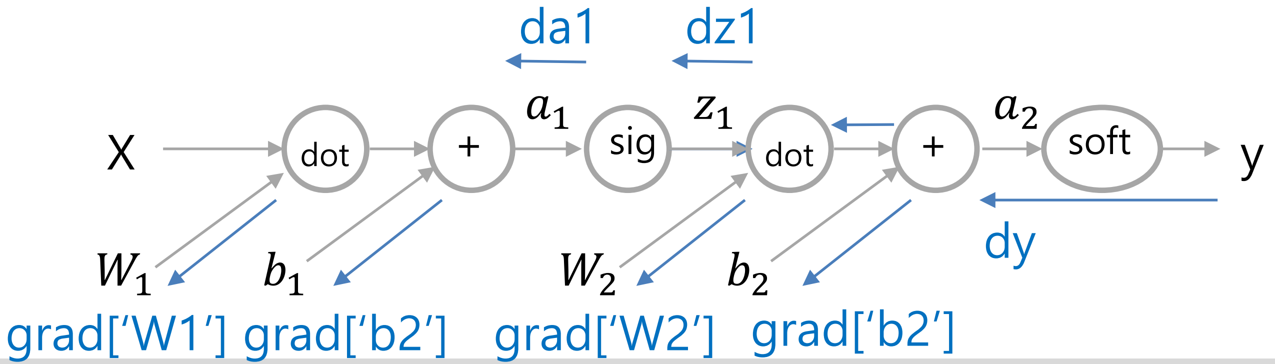
실습 1 : 2 레이어 네트워크 실습

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

```python
    # backward
    dy = (y - t) / batch_num
    ####################################
    ######### Write your codes #######
    ####################################
    grads['W2'] = None
    grads['b2'] = None

    dz1 = None
    da1 = None
    grads['W1'] = None
    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 : 1_day/prac3_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

- 2) 구현파일 : 1_day/prac3_two_layer_net.py

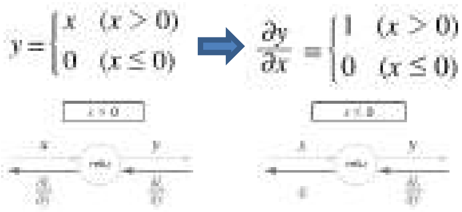
```python
# coding: utf-8
import sys, os
sys.path.append(os.pardir)  # 부모 디렉터리의 파일을 가져올 수 있도록 설정
import numpy as np
from common.layers import *
from common.gradient import numerical_gradient
from collections import OrderedDict


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

- 2) 구현파일 : 1_day/prac3_two_layer_net.py

  # common/layers.py




```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
```

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

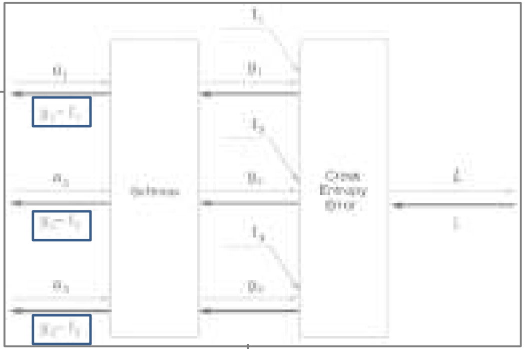
$$\frac{\partial L}{\partial \mathbf{W}} = \mathbf{X}^T \cdot \frac{\partial L}{\partial \mathbf{Y}}$$

실습 2 : 2 레이어 네트워크 모듈화

모두의연구소

- 2) 구현파일 : 1_day/prac3_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

모두의연구소

- 2) 구현파일 : 1_day/prac3_two_layer_net.py

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

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

        layers = list(self.layers.values())
        layers.reverse()
        ####################################
        #########  Write you code #########
        ####################################
        for layer in layers:
            dout = None
        #########  end of you code #########
        # 결과 저장
        grads = {}
        grads['W1'], grads['b1'] = self.layers['Affine1'].dW, self.layers['Affine1'].db
        grads['W2'], grads['b2'] = self.layers['Affine2'].dW, self.layers['Affine2'].db

        return grads
```

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

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

모두의연구소

박 은 수 Research Director

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