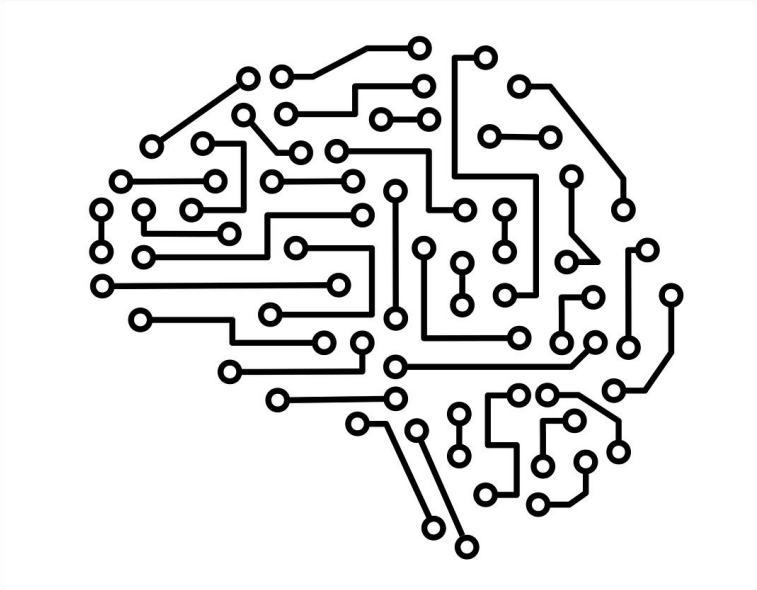
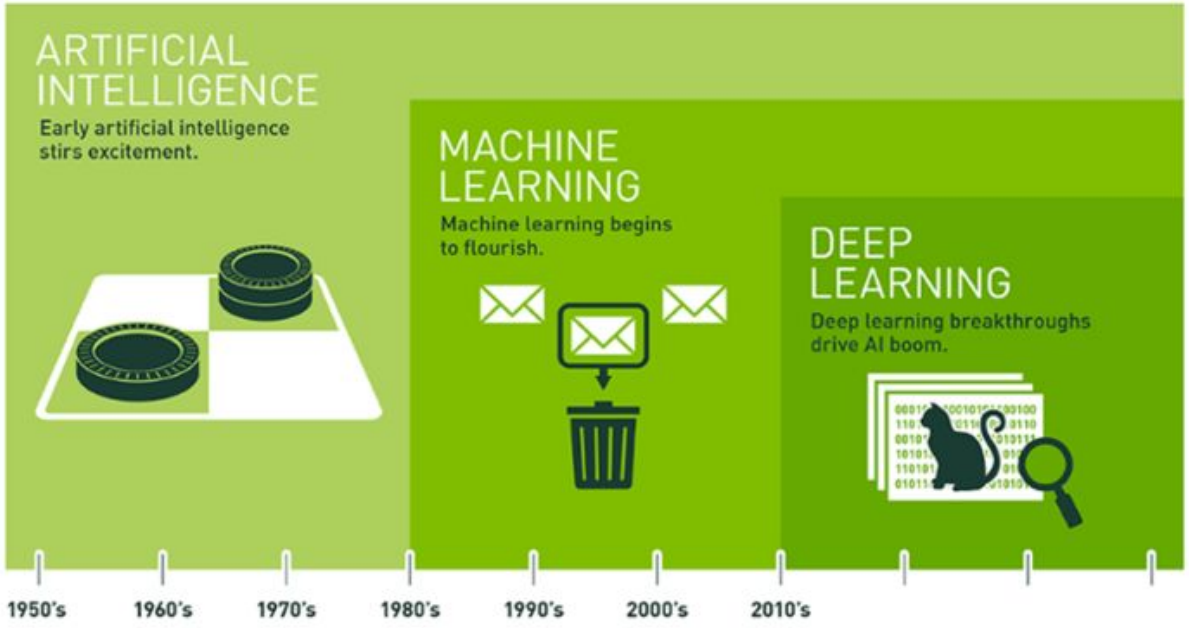
Deep Learning



Contents

- Deep Learning?
- ML? DL? AI?
- Deep Learning 적용 사례
- How to start learning Deep Learning?
- Neural Network?
- Keywords

Machine Learning? Deep Learning? A.I.?



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

https://blogs.nvidia.co.kr/2016/08/03/difference_ai_learning_machinelearning/

Machine Learning? Deep Learning? A.I.?

- 인공 지능 (Artificial Intelligence)

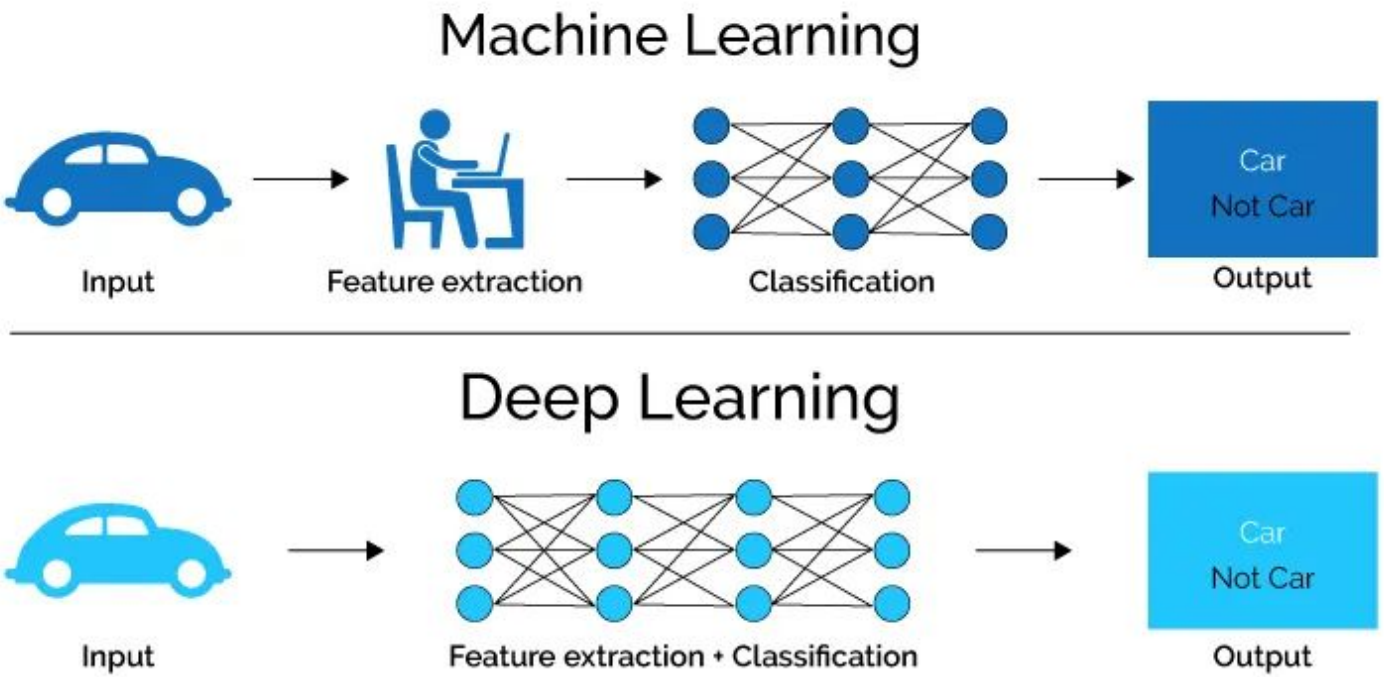
  기계로 부터 만들어진 지능
- 기계 학습 (Machine Learning)

  통계적 기법을 이용해 데이터로부터 학습
- 딥러닝 (Deep Learning)

  인공신경망을 기반으로 함

https://www.hohyeonmoon.com/blog/ai-ml-dl-relation/
https://blog.naver.com/PostView.nhn?blogId=dsjang650628&logNo=221864626337
https://towardsdatascience.com/understanding-the-difference-between-ai-ml-and-dl-cceb63252a6c

Why Deep Learning?

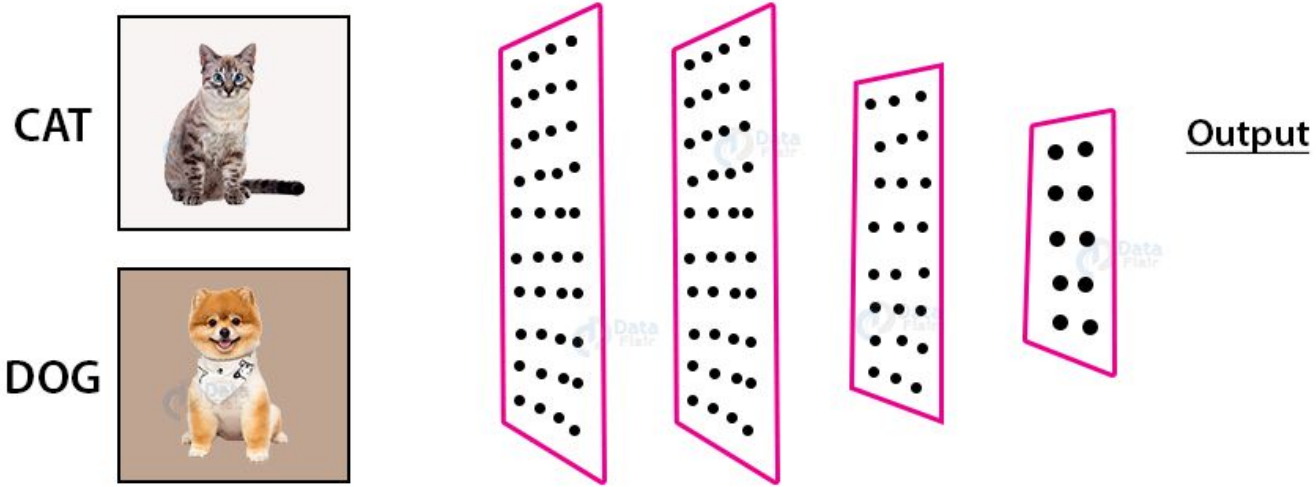


https://semiengineering.com/deep-learning-spreads/

Deep Learning 적용 사례

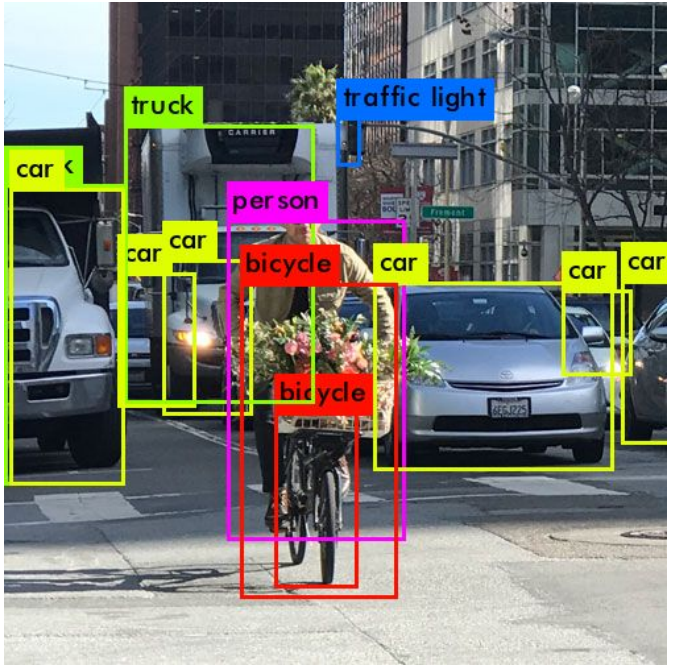
- Classification
- Object Detection & Segmentation
- Generation (Generative Adversarial Network)
- Reinforcement Learning
- 다른 적용 사례들

Deep Learning - Classification

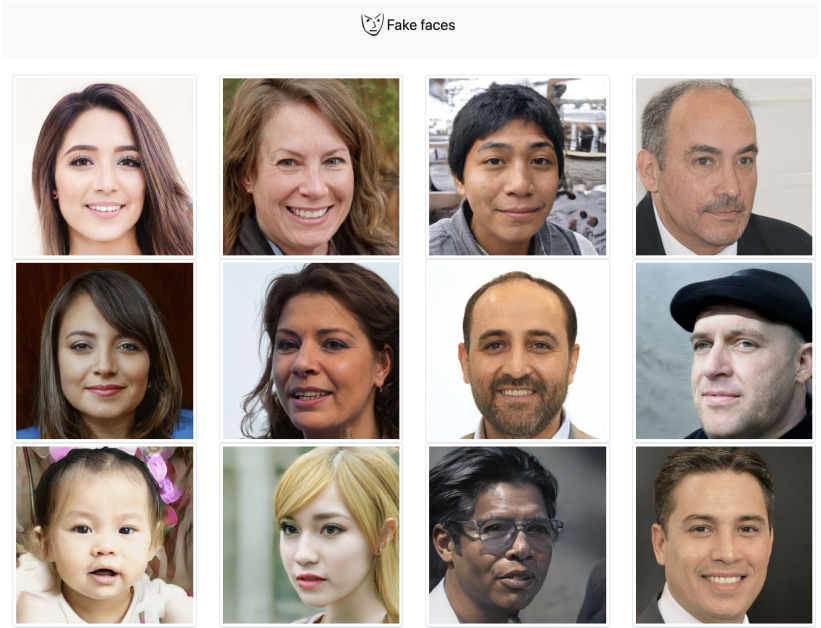


https://data-flair.training/blogs/cats-dogs-classification-deep-learning-project-beginners/

Deep Learning - Detection & Segmentation



Deep Learning - Generation (GAN)



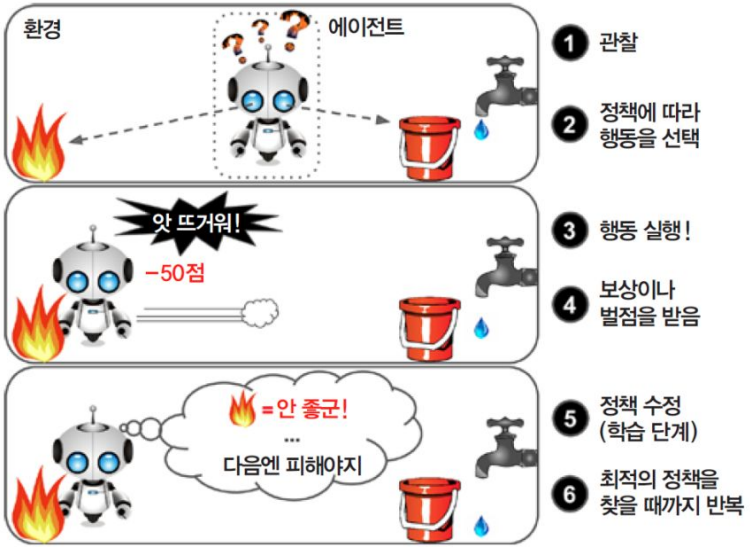
https://fakeface.co/

https://thispersondoesnotexist.com/

이게 다 가짜라니..

Deep Learning - Reinforcement Learning

그림 1-12 강화 학습



출처: tensorflow.blog

Deep Learning 적용 사례

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

딥러닝이 적용된 연구 사례, 논문, 프로젝트 등을 소개한 블로그 포스트를 요약 소개함

딥러닝 적용 사례 카테고리

- 컴퓨터 비전과 패턴 인식
- 컴퓨터 게임, 자율 주행, 로봇
- 작곡, 음성합성, 입모양 인식
- 딥러닝이 만들어내는 예술 작품
- 컴퓨터 환각, 예측 및 기타 사례 들

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

사례 - 컴퓨터 비전과 패턴 인식

정치인 재연

워싱턴 대학의 한 연구 그룹이 오디오 사운드를 이용하여 입술 모양을 재현하는 프로젝트를 수행함

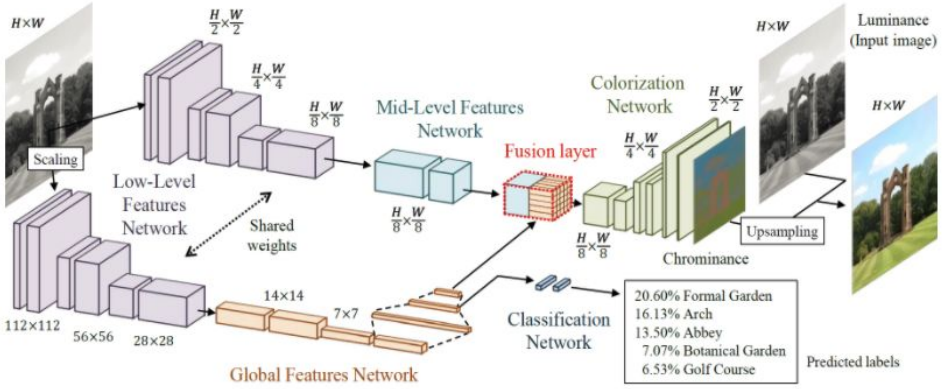
논문 링크

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

사례 - 컴퓨터 비전과 패턴 인식

흑백 사진 색상 복원 (사진, 영상)

Feature network와 colorization network를 동시에 학습하여 흑백 사진 복원 기술을 구현함



논문 링크

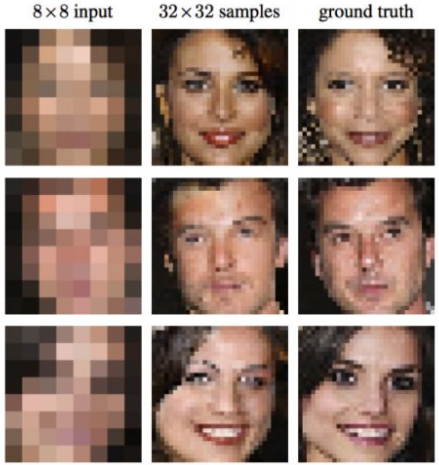
결과 예시들

http://www.yaronhadad.com/deep-learning-most-amazing-applications/

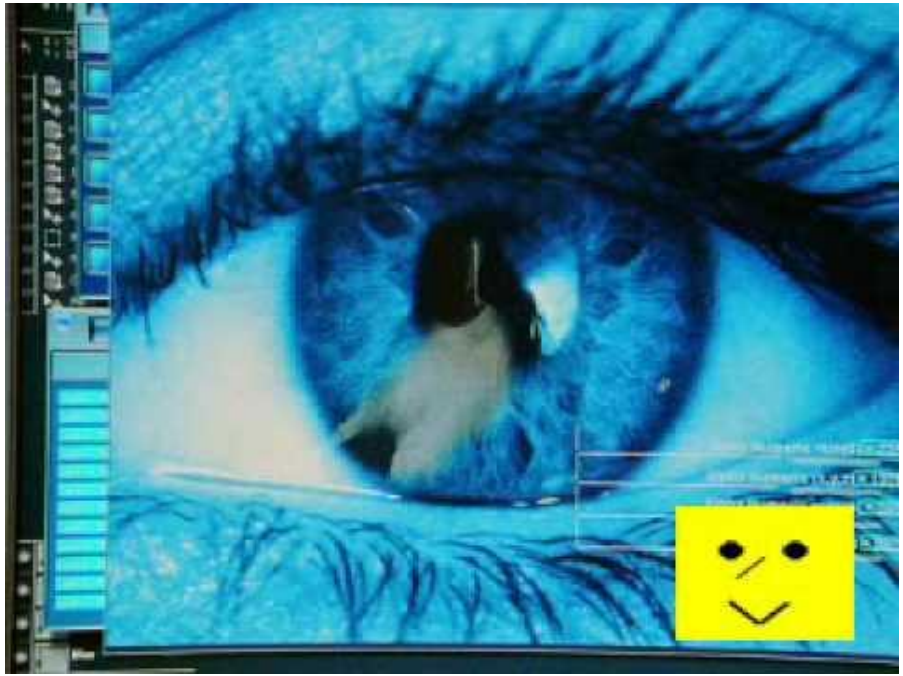
사례 - 컴퓨터 비전과 패턴 인식

픽셀 복원, 해상도 복원

저해상도 이미지에서 고해상도 이미지를 복원함



논문 링크



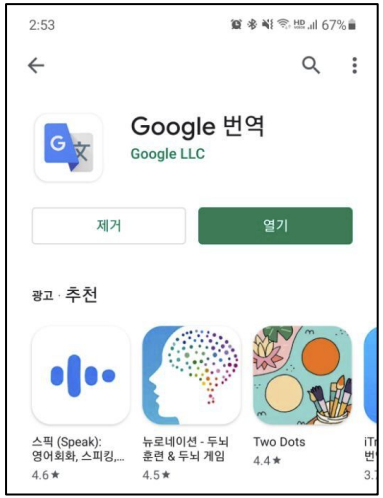
http://www.yaronhadad.com/deep-learning-most-amazing-applications/

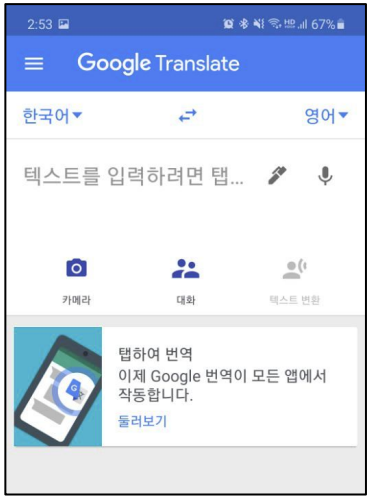
사례 - 컴퓨터 비전과 패턴 인식

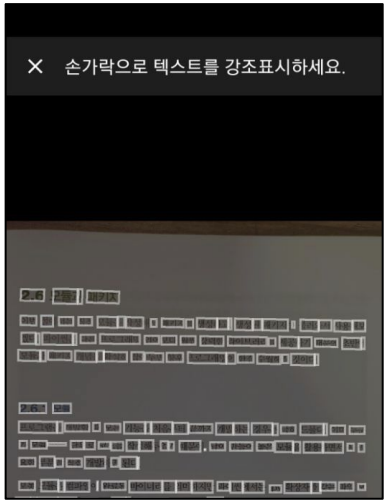
글자 번역 (Google Translator)

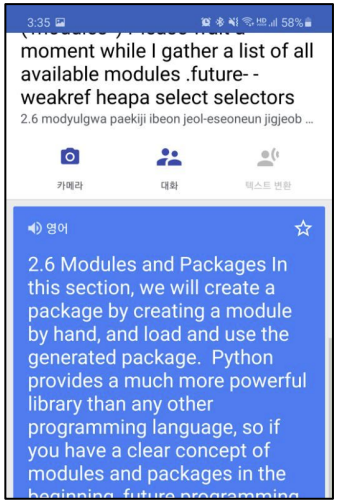
블로그 링크

이미지에서 글자를 인식하여 번역함









http://www.yaronhadad.com/deep-learning-most-amazing-applications/

사례 - 컴퓨터 비전과 패턴 인식

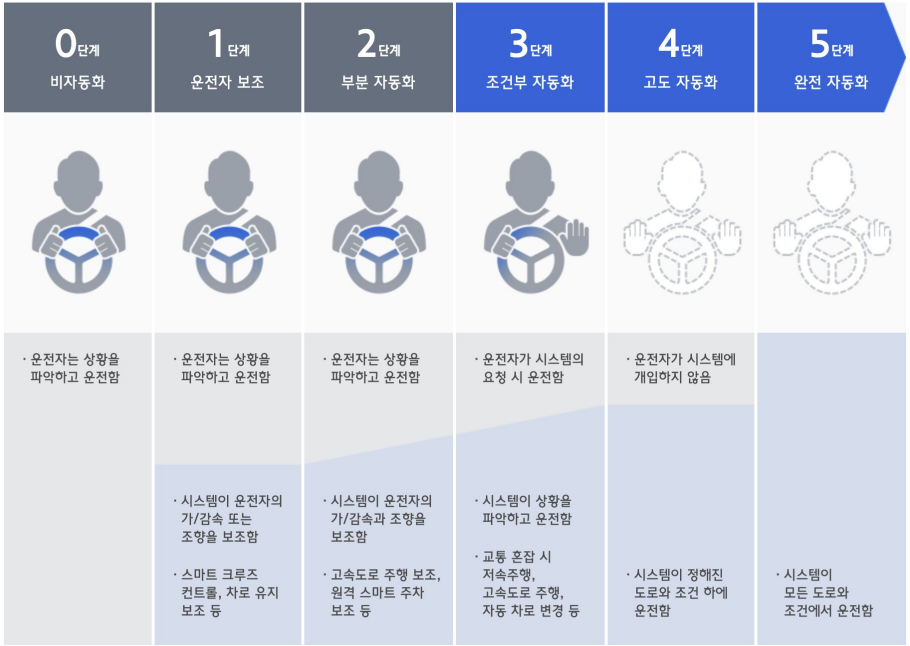
새로운 이미지 생성

이미지의 특성을 학습하여 새로운 이미지를
생성하거나 주어진 이미지를 재창조함

이미지 재생성 페이지 링크

사례 - 자율 주행 자동차

자동차에 부착된 센서에서 획득 할 수 있는 정보를 이용하여 자율 주행 기능을 수행함

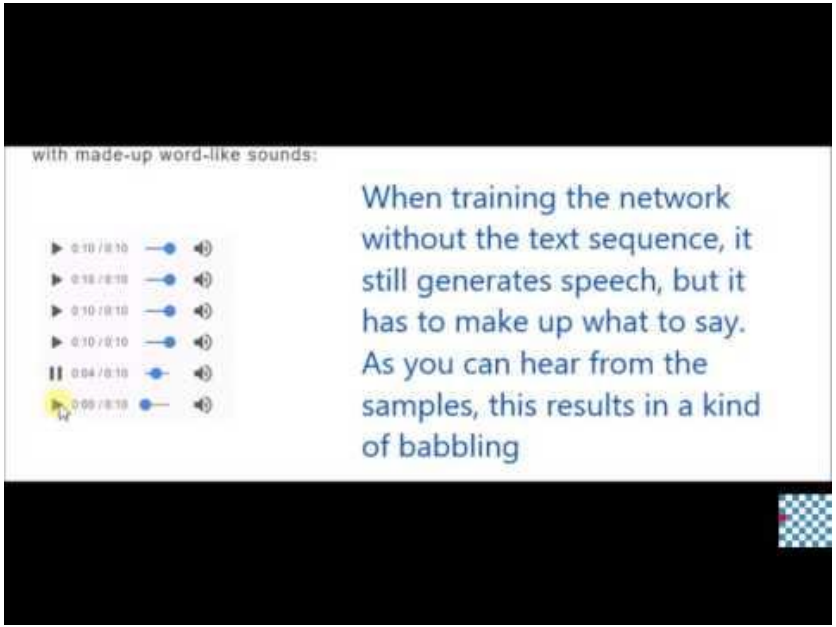


https://www.hyundai.co.kr/TechInnovation/Autonomous/Roadmap.hub

사례 - 소리 데이터

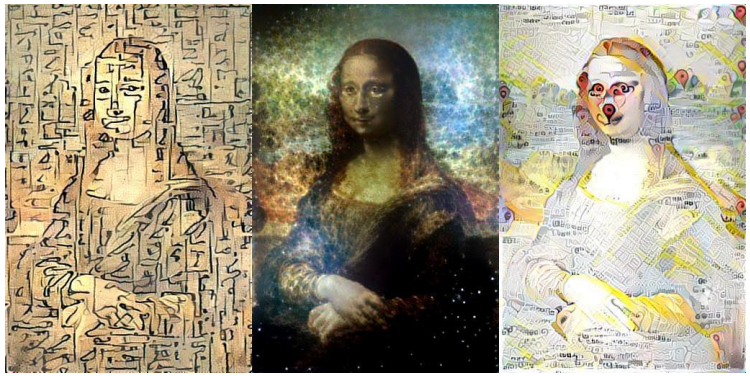
- 음성 합성 기술
- 작곡하는 인공지능

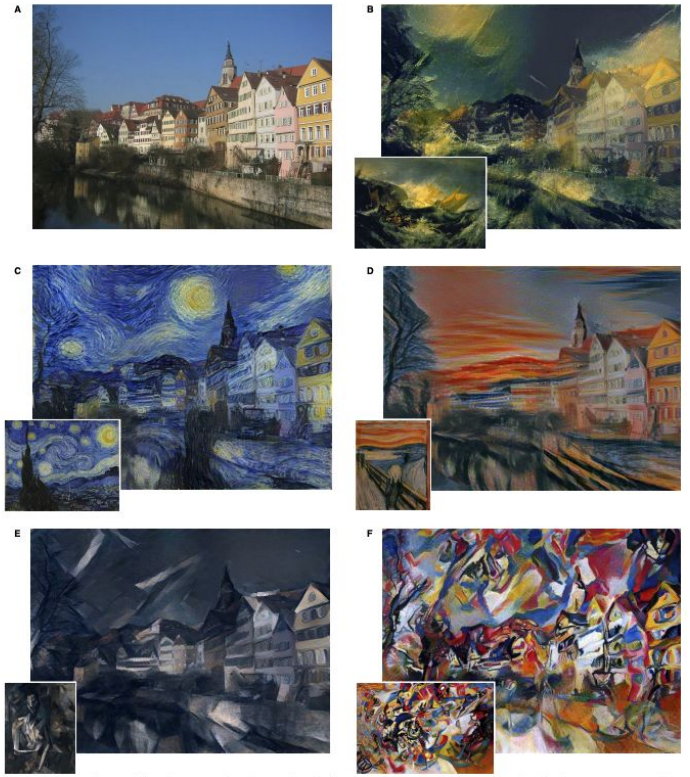
https://openai.com/blog/musenet/



사례 - 예술 작품

- 화풍을 따라하는 인공지능





관련 논문 페이지 링크

딥러닝 사업화 사례


vFlat - 내 손 안에 책 스캐너, 브이플랫
VoyagerX 생산성
6,554
일부 기기와 호환되는 앱입니다.
이 항목을 가족과 공유할 수 있습니다. 가족 콘텐츠 라이브러리
자세히 알아보기
설치됨
vFlat
터치 한번으로
빠르고 쉬운 스캔
2019 Google Play 올해의 숨은 보석 앱으로 선정!
vFlat은 문서, 책, 메모 등 핸드폰으로 촬영한 이미지를 고화질 PDF 또는 JPG 이미지로 만들어주는 스캐닝 앱입니다. vFlat은 문서의 테두리를 자동으로 인식하여 자르고 보정합니다. 또한 촬영한 이미지를 텍스트로 변환하여 원하는 키워드를 검색하거나 복사할 수 있습니다.


내가 쓰고, 인공지능이 채워줍니다.
한글 손글씨 폰트에 들어가는 글자는 총 11,172자,
이를 일일이 적는 것은 무척 힘든 일입니다.
이제, 당신은 300자만 쓰세요.
나머지는 온글잎의 인공지능이 꽉 채워드리겠습니다.
온글잎
OWNGLYPH
OWNGLYPH

Deep Learning History

- 딥러닝 역사 - 타임라인
- 딥러닝 타임라인 - 위키피디아
- 한국테크노파크진흥회 요약자료

How to start learning Deep Learning?

1. 개발환경 세팅 (pycharm, anaconda, colab, 등)
2. Framework 선택 (tensorflow, pytorch, 등)
3. 문제 정의 (classification, regression, 등)
4. 풀이 방법 설계
5. 데이터 분석, 전처리 (data-EDA, preprocessing)
6. Coding
7. 결과 분석 (hyper parameter tuning 등)
8. 성능 향상을 위한 추가적인 작업들

How to start learning Deep Learning?

Dataset

- MNIST
- Fashion-MNIST
- CelebA
- LSUN
- Kaggle
- Dacon
- +Custom data..

Tools & Frameworks


Models

- Classification
- Regression
- Generative
- Reinforcement
- NLP

딥러닝 프레임워크

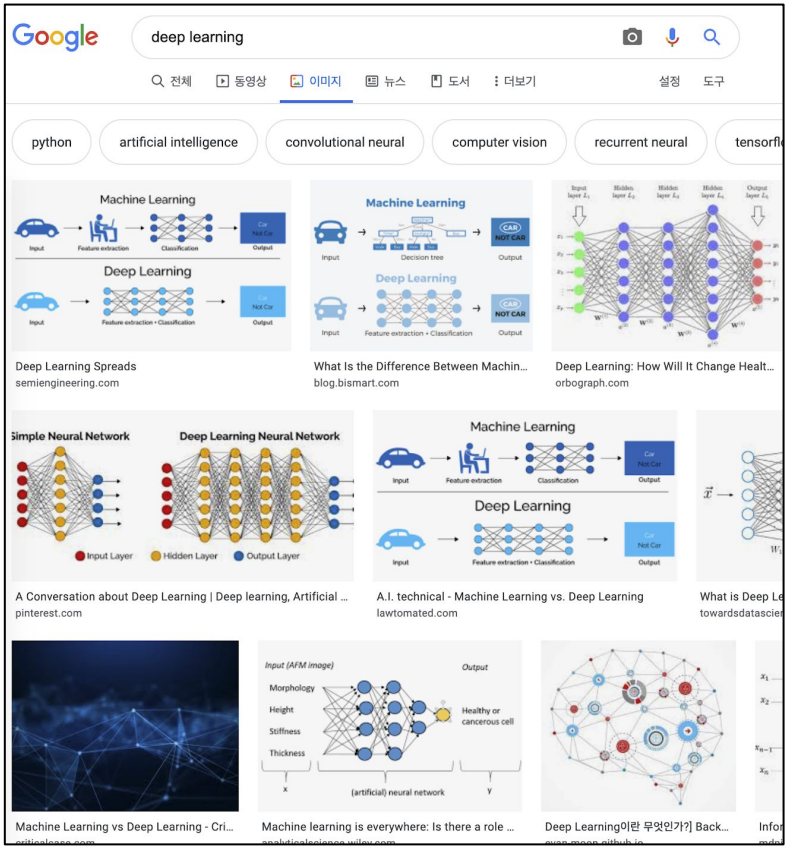
vs

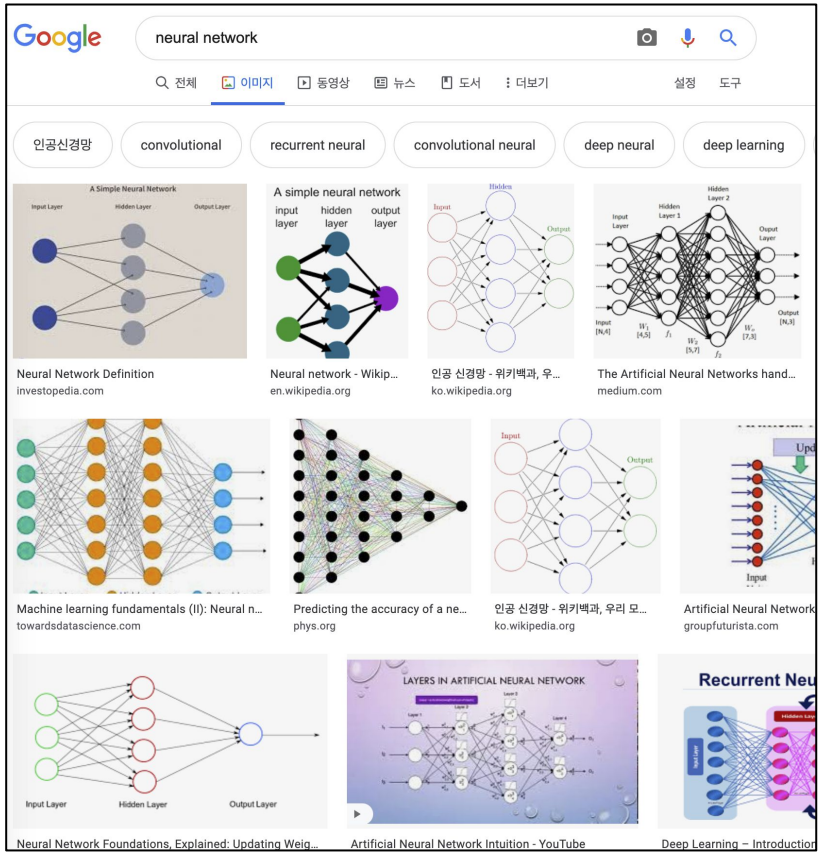
How to start learning Deep Learning?

1. 개발환경 세팅 (pycharm, anaconda, colab, 등) ➔ **Google Colab** Colab Tutorial
2. Framework 선택 (tensorflow, pytorch, 등) ➔ **Tensorflow (Keras)** TF공식 튜토리얼
3. 문제 정의 (classification, regression, 등) ➔ **분석하려는 데이터에 따라 결정**
4. 풀이 방법 설계
5. 데이터 분석, 전처리 (data-EDA, preprocessing)
6. Coding ➔ **Coding (python)**
7. 결과 분석 (hyper parameter tuning 등)
8. 성능 향상을 위한 추가적인 작업들 ➔ **데이터셋 정제, 네트워크 고도화 등**

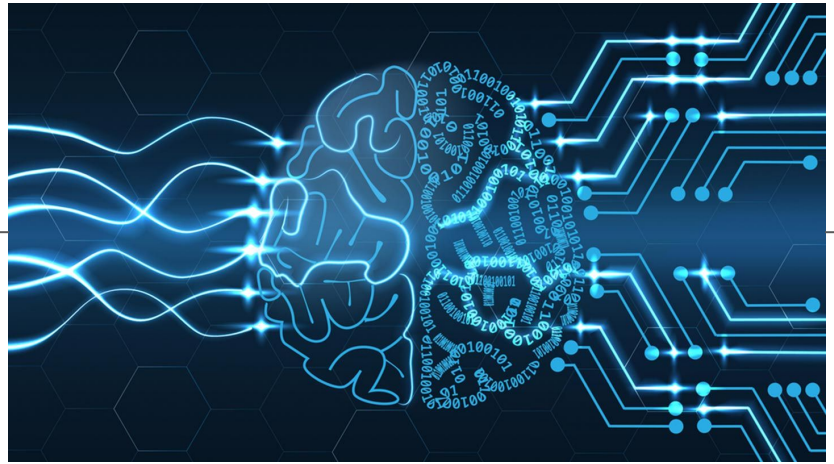
필요에 따라 다른 방향으로 분석하는 것도 당연히 가능하다!

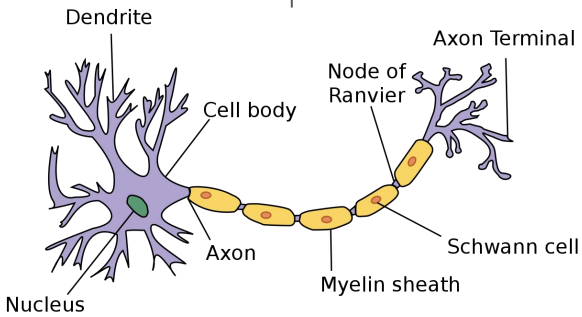
Deep Learning, Neural Network

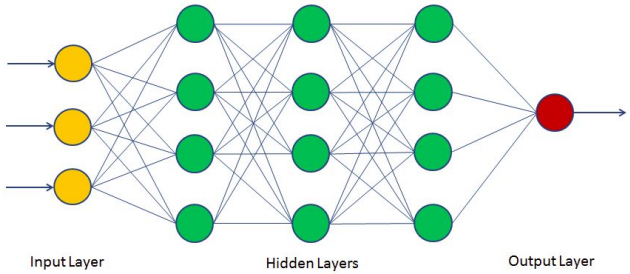




Deep Learning, Neural Network



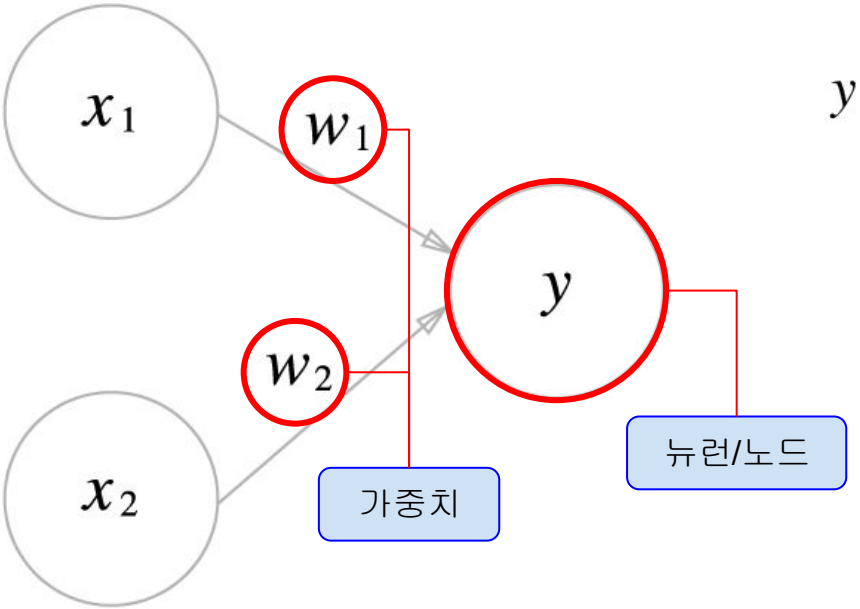




https://www.datacamp.com/community/tutorials/neural-network-models-r

Perceptron (퍼셉트론)

입력이 2개인 퍼셉트론



수식으로 나타내면..

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

임계값

입력신호가 **뉴런(노드)**에 보내질 때 고유한 **가중치**가 곱해짐.
뉴런에서 보내온 신호의 총 합이 **임계값**을 넘을 때만 1을 출력(**활성화**)함

단순한 논리 회로

- AND
- NAND (Not AND)
- OR

단순한 논리 회로

AND GATE

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

NAND GATE

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

OR GATE

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

퍼셉트론이 위의 **논리표** 대로 작동하게 하려면 어떻게 해야할까?

- x1, x2 입력에 대한 가중치 w1, w2 와 임계값 θ를 결정
- w1, w2, θ 는 여러가지 조합이 존재할 수 있음

단순한 논리 회로

AND GATE

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leqq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$$y = \begin{cases} 0 & (0.5 x_1 + 0.5 x_2 \leqq 0.7) \\ 1 & (0.5 x_1 + 0.5 x_2 > 0.7) \end{cases}$$

단순한 논리 회로

AND GATE

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leqq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$$y = \begin{cases} 0 & (0.5 x_1 + 0.5 x_2 \leqq 0.7) \\ 1 & (0.5 x_1 + 0.5 x_2 > 0.7) \end{cases}$$

x_1	x_2	y	$\theta = 0.7$
0	0	$0.5 \times 0 + 0.5 \times 0 = 0$	0
1	0	$0.5 \times 1 + 0.5 \times 0 = 0.5$	0
0	1	$0.5 \times 0 + 0.5 \times 1 = 0.5$	0
1	1	$0.5 \times 1 + 0.5 \times 1 = 1$	1

단순한 논리 회로

AND GATE

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$(w_1, w_2, \theta) = (0.5, 0.5, 0.7)$

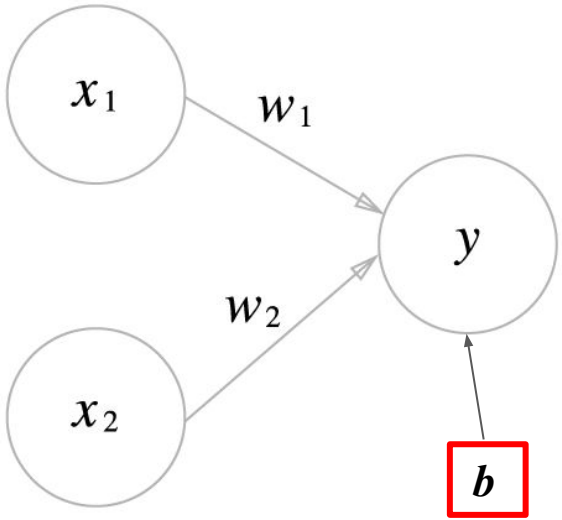
$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases}$$

$$y = \begin{cases} 0 & (0.5 x_1 + 0.5 x_2 \leq 0.7) \\ 1 & (0.5 x_1 + 0.5 x_2 > 0.7) \end{cases}$$

X_1	X_2	y	$\theta = 0.7$
0	0	$0.5 \times 0 + 0.5 \times 0 = 0$	0
1	0	$0.5 \times 1 + 0.5 \times 0 = 0.5$	0
0	1	$0.5 \times 0 + 0.5 \times 1 = 0.5$	0
1	1	$0.5 \times 1 + 0.5 \times 1 = 1$	1

가중치와 편향

$$y = \begin{cases} 0 & (w_1 x_1 + w_2 x_2 \leq \theta) \\ 1 & (w_1 x_1 + w_2 x_2 > \theta) \end{cases} \Rightarrow y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \leq 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$



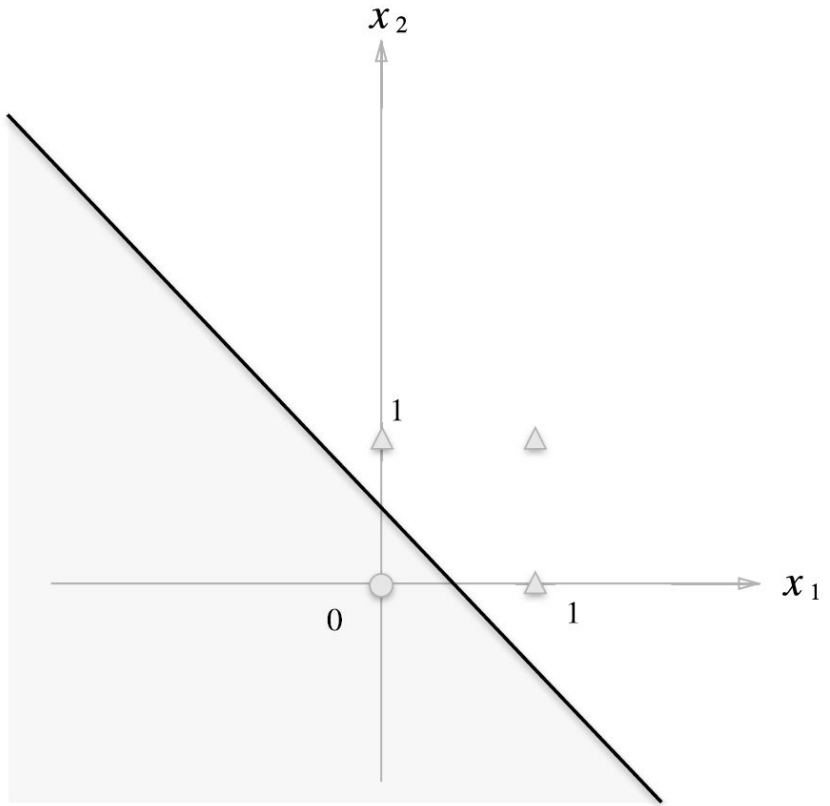
퍼셉트론의 시각화

$$y = \begin{cases} 0 & (-0.5 + x_1 + x_2 \leq 0) \\ 1 & (-0.5 + x_1 + x_2 > 0) \end{cases}$$

OR 게이트

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

회색 영역은 0을 출력하는 영역,
전체 영역은 OR 게이트의 성질을 만족함



퍼셉트론 구현하기

앞에서 학습한 간단한 논리 회로를 python 코드로 학습

- AND
- NAND
- OR

Colab에서 확인하기

퍼셉트론 구현하기

앞에서 학습한 간단한 논리 회로를 python 코드로 학습

```python
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

```python
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

```python
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

구조는 모두 같고 가중치나 편향이 다른 것을 확인할 수 있음

퍼셉트론의 한계

XOR GATE

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

(w1, w2, θ) = (? , ? , ?)

가중치와 임계값(혹은 편향)이 어떤 값을 가져야

XOR 게이트의 진리표를 충족시킬 수 있을까?

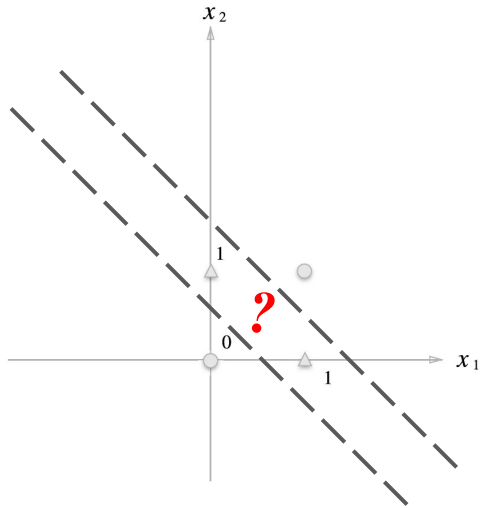
(퍼셉트론으로 XOR 게이트를 구현하려면 어떻게 해야할까?)

퍼셉트론의 한계

XOR GATE

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

퍼셉트론으로 XOR 게이트를 구현하려면 어떻게 해야할까?

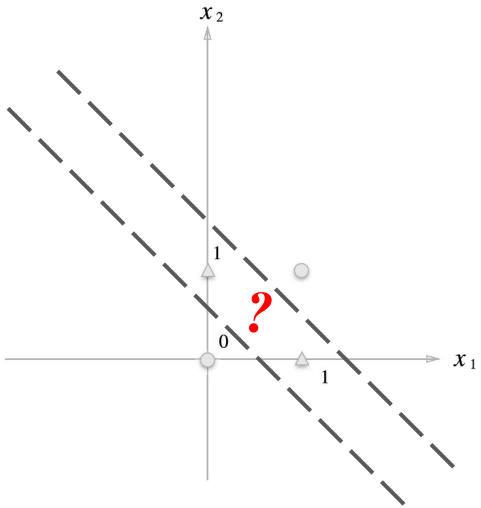


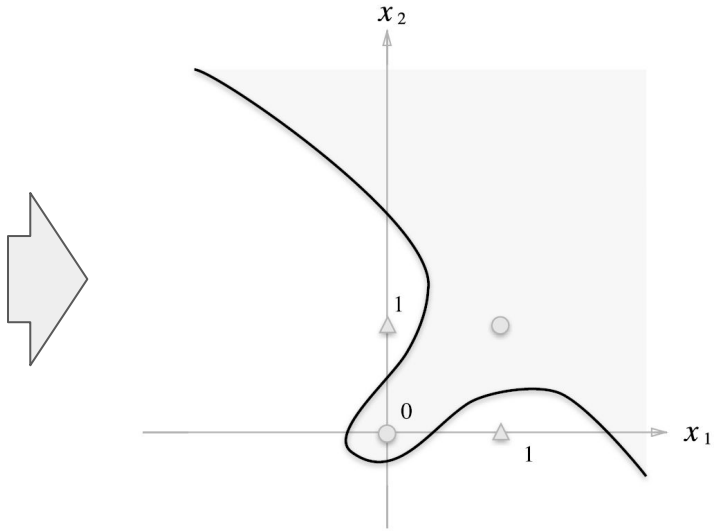
퍼셉트론의 한계

XOR GATE

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

퍼셉트론으로 XOR 게이트를 구현하려면 어떻게 해야할까?





퍼셉트론은 직선 하나로 나눈 영역만 표현할 수 있다는 한계가 있음

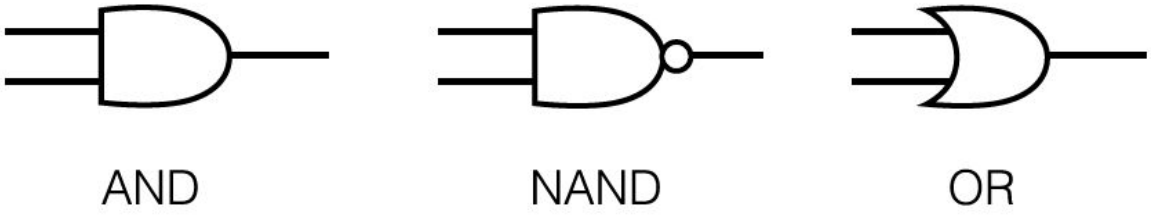
➔ **비선형** 영역의 필요성

다중 퍼셉트론

- 기존 게이트 조합하기
- XOR 게이트 구현하기

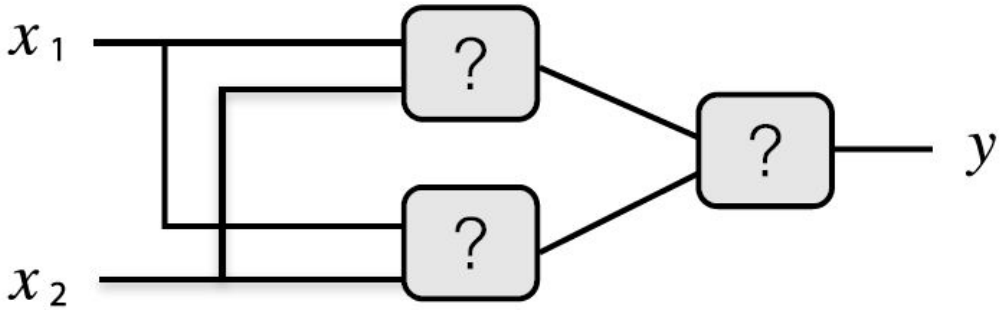
다중 퍼셉트론

게이트 기호

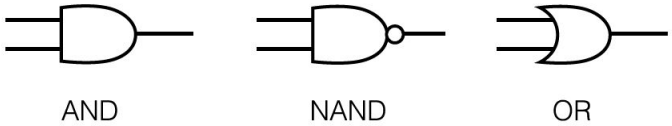


퍼셉트론 여러개를 조합하여 XOR 게이트를 구현할 수 있음

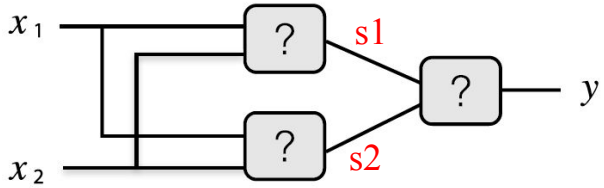
(게이트의 output들이 다른 게이트의 input으로 사용될 수 있음)

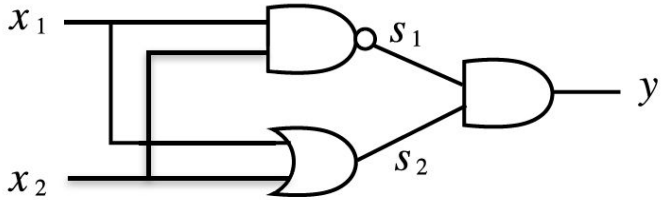


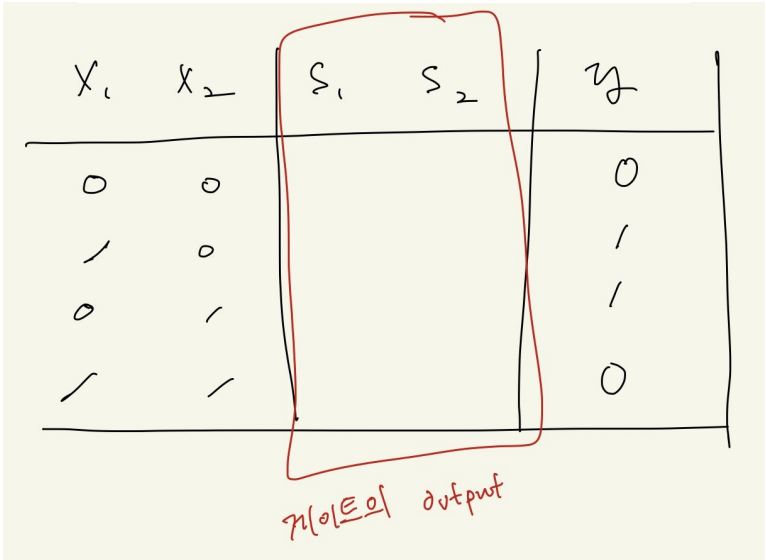
다중 퍼셉트론



퍼셉트론 여러개를 조합하여 XOR 게이트를 구현할 수 있음





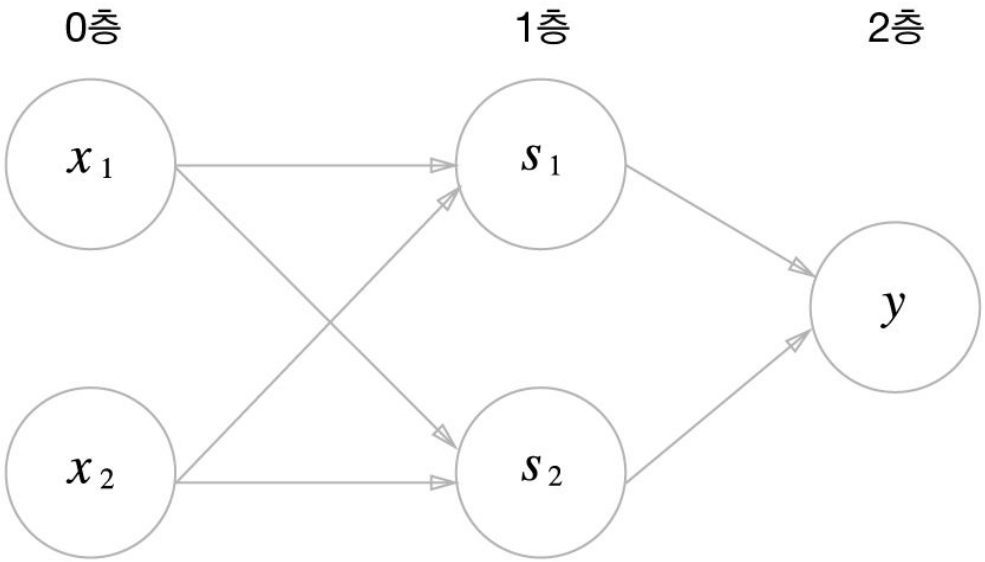


x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

Colab에서 확인하기

다중 퍼셉트론

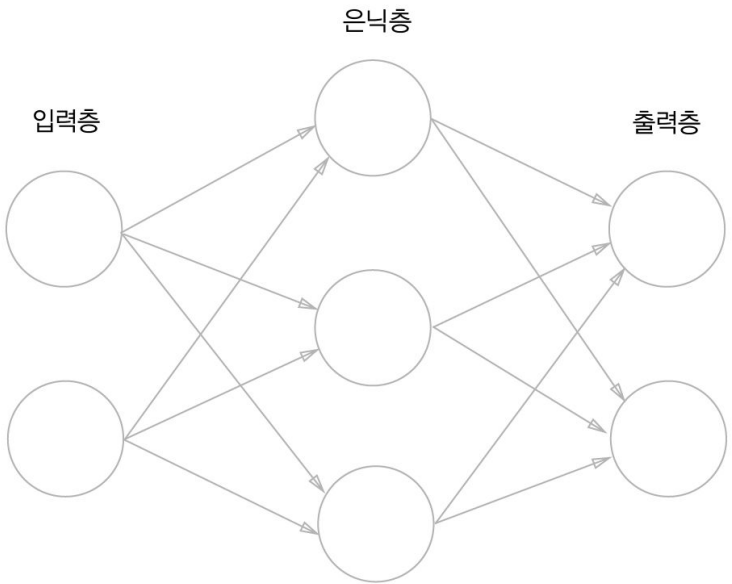
XOR 게이트를 퍼셉트론으로 표현하면 다음과 같음



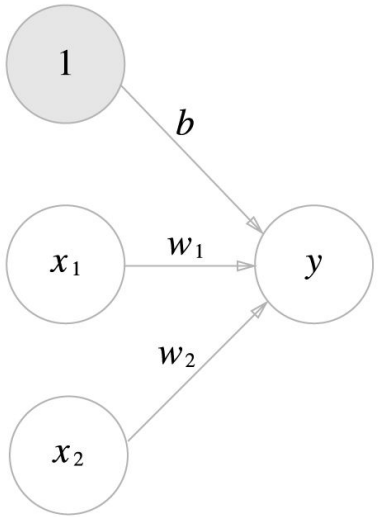
퍼셉트론 요약

- 퍼셉트론은 입출력을 갖춘 알고리즘이다. 입력을 주면 정해진 규칙에 따른 값을 출력한다.
- 퍼셉트론에서는 **'가중치'와 '편향'**을 매개변수로 설정한다.
- 퍼셉트론으로 AND, OR 게이트 등의 **논리 회로를 표현**할 수 있다.
- **XOR 게이트**는 단층 퍼셉트론으로는 표현할 수 없다.
- 2층 퍼셉트론을 이용하면 XOR 게이트를 표현할 수 있다.
- 단층 퍼셉트론은 직선형 영역만, 다층 퍼셉트론은 비선형 영역도 표현할 수 있다.

퍼셉트론에서 신경망으로



신경망 예시



편향이 포함된 퍼셉트론

활성화 함수 (Activation Function)

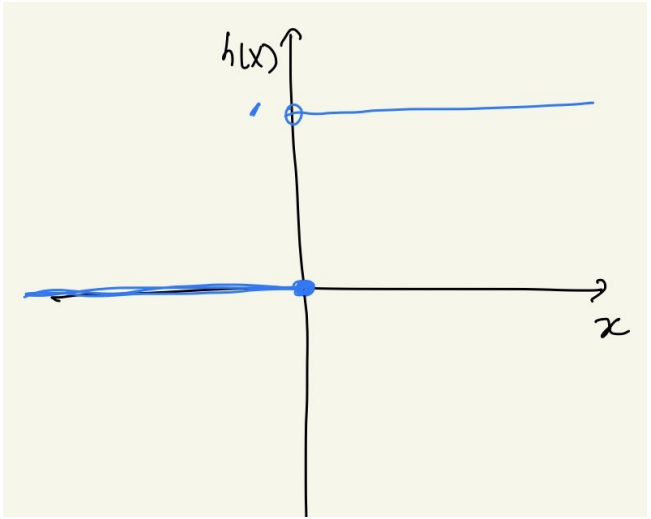
$$y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \leq 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

활성화 함수 (Activation Function)

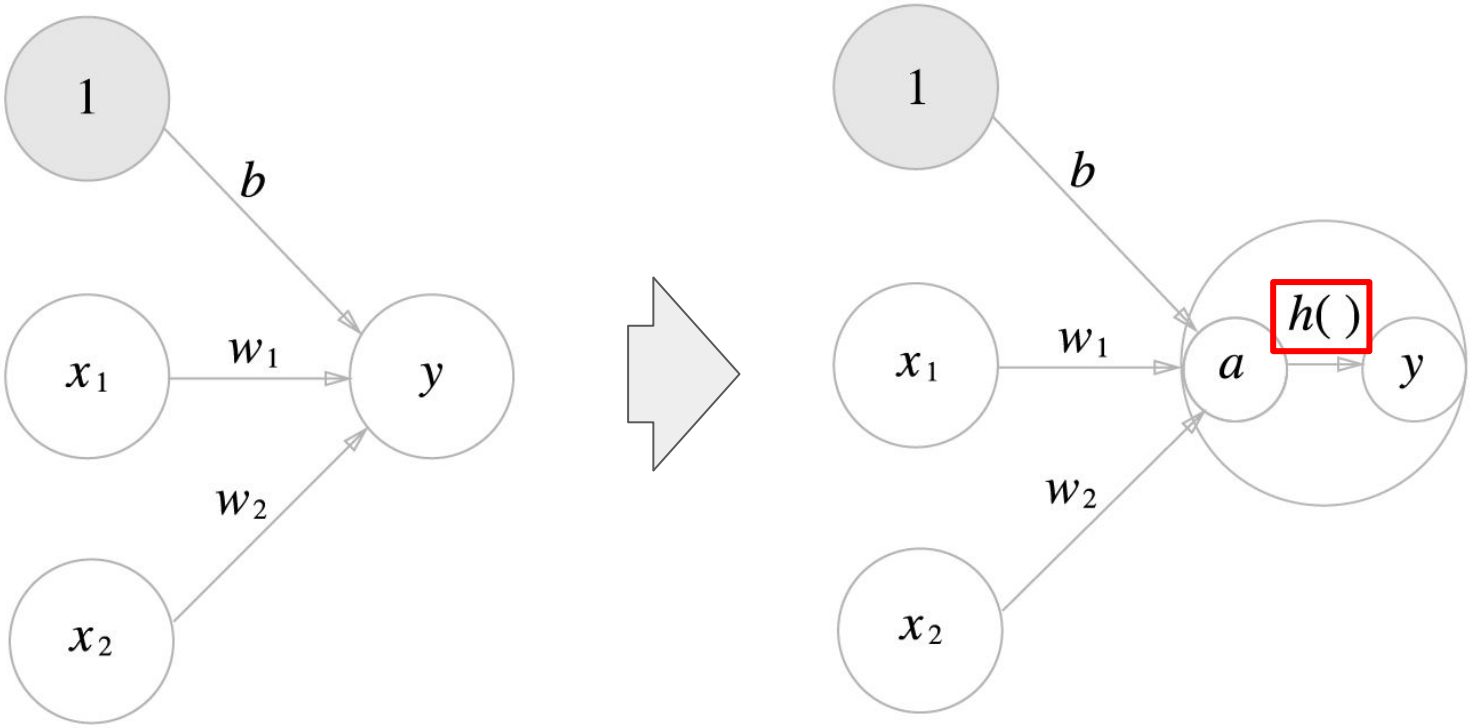
$$y = \begin{cases} 0 & (b + w_1 x_1 + w_2 x_2 \leq 0) \\ 1 & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases}$$

$$y = h(b + w_1 x_1 + w_2 x_2)$$

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



활성화 함수 (Activation Function)



Colab에서 확인하기

활성화 함수 (Activation Function)

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

```
def step_function(x):
  if x > 0:
    return 1
  else:
    return 0
```

```
1 def step_function(x):
2   y = x > 0
3   return y.astype(np.int)
```

```
1 import numpy as np
```

```
1 x = np.array([-1.0, 1.0, 2.0])
```

```
1 x
```

```
array([-1.,  1.,  2.])
```

```
1 y = x > 0
```

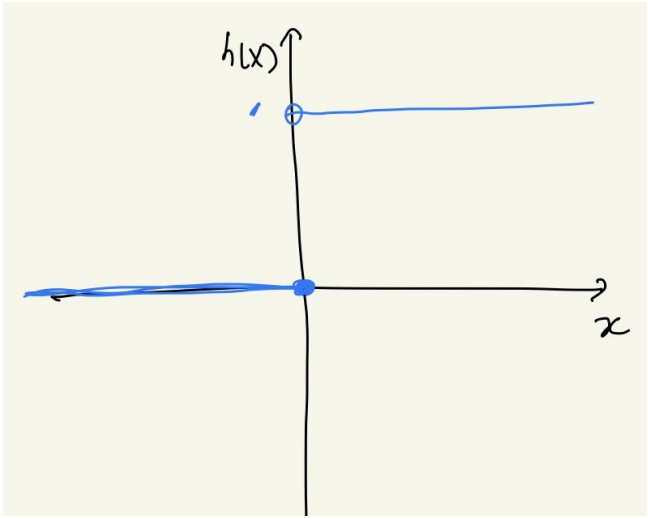
```
1 y
```

```
array([False,  True,  True])
```

```
1 y = y.astype(np.int)
```

```
1 y
```

```
array([0, 1, 1])
```



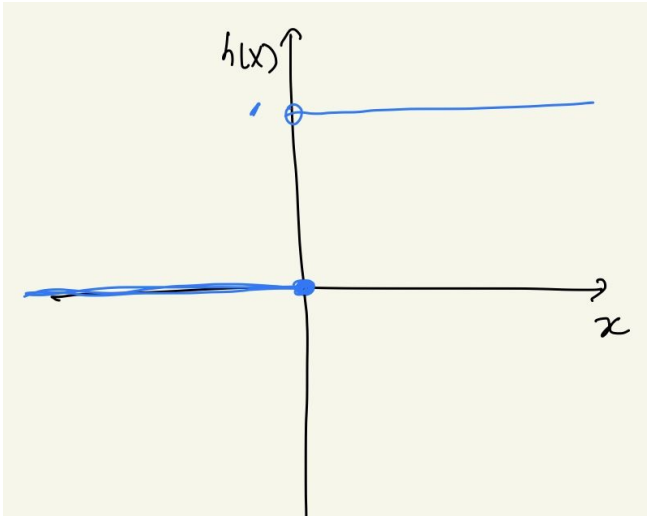
임계값을 경계로 출력이 바뀌는
함수를 **계단 함수 (step function)**
이라고 함

Colab에서 확인하기

활성화 함수 (Activation Function)

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

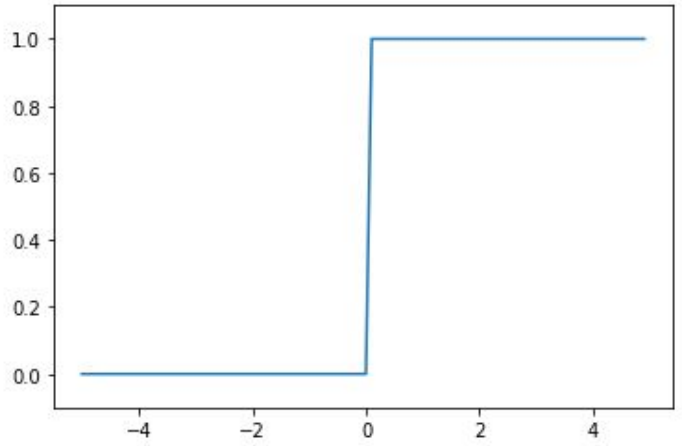
```python
def step_function(x):
  if x > 0:
    return 1
  else:
    return 0
```



임계값을 경계로 출력이 바뀌는
함수를 **계단 함수 (step function)**
이라고 함

```python
import numpy as np
import matplotlib.pylab as plt
```

```python
x = np.arange(-5.0, 5.0, 0.1)
y = step_function(x)
plt.plot(x,y)
plt.ylim(-0.1, 1.1)
plt.show()
```

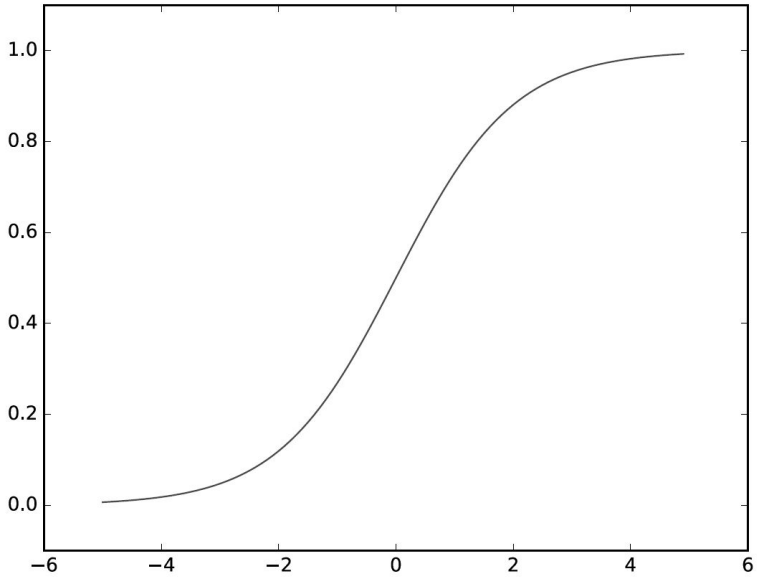


Colab에서 확인하기

활성화 함수 (Activation Function)

시그모이드 함수
(Sigmoid function)

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



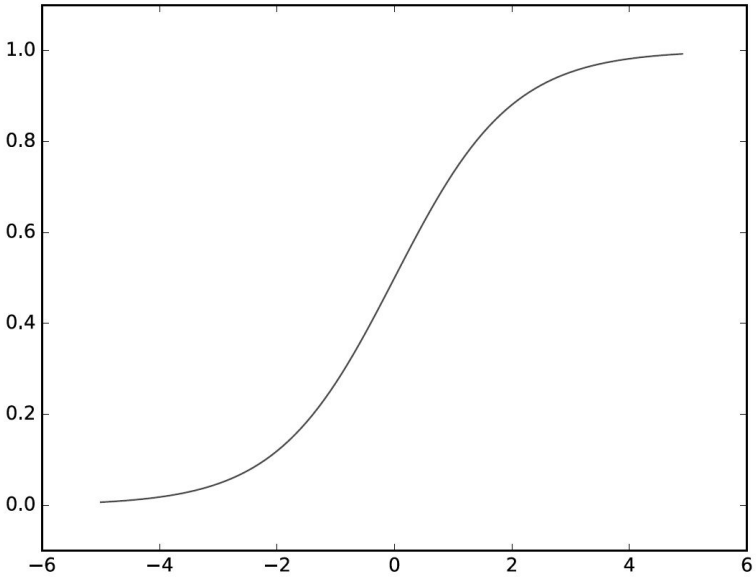
Colab에서 확인하기

활성화 함수 (Activation Function)

시그모이드 함수
(Sigmoid function)

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

```
def sigmoid(x):
  return 1 / (1 + np.exp(-x))
```



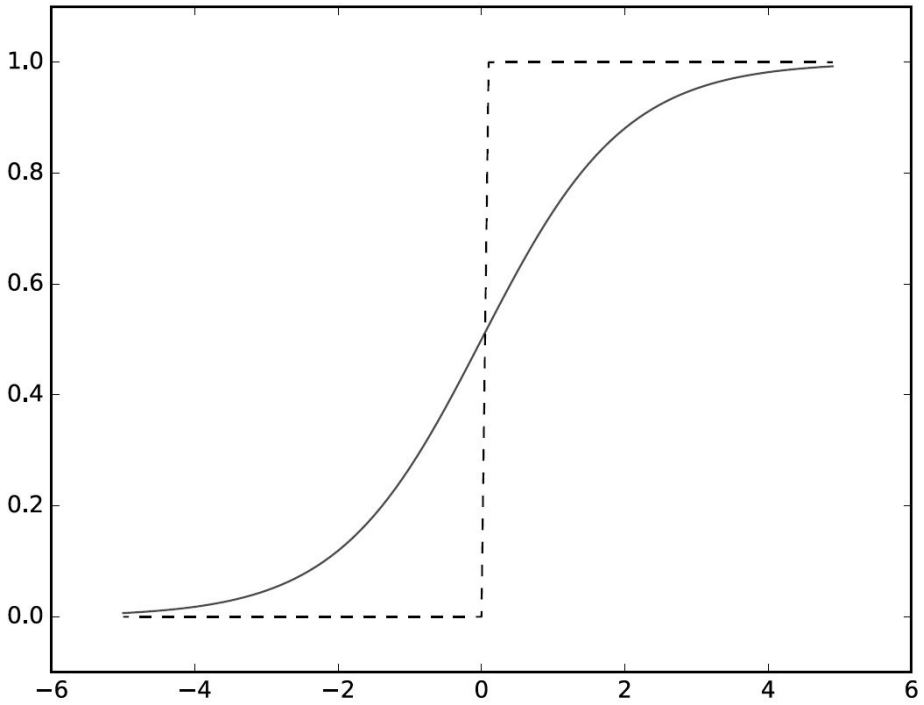

```
1 x = np.array([-1.0, 1.0, 2.0])
2 y = sigmoid(x)
3 print(y)
```

```
[0.26894142 0.73105858 0.88079708]
```

Colab에서 확인하기

활성화 함수 (Activation Function)

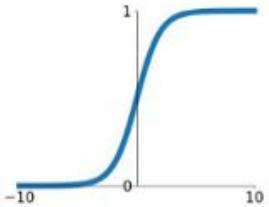
시그모이드 함수 (Sigmoid function) & 계단 함수 (Step function)



활성화 함수 (Activation Function)

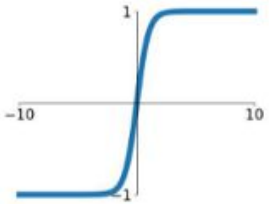
Sigmoid

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



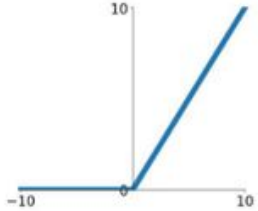
tanh

$$\tanh(x)$$



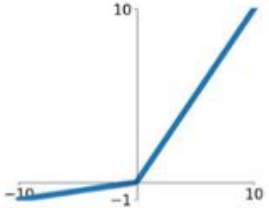
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

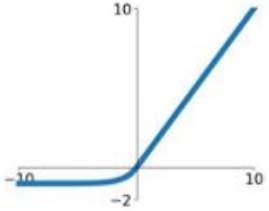


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

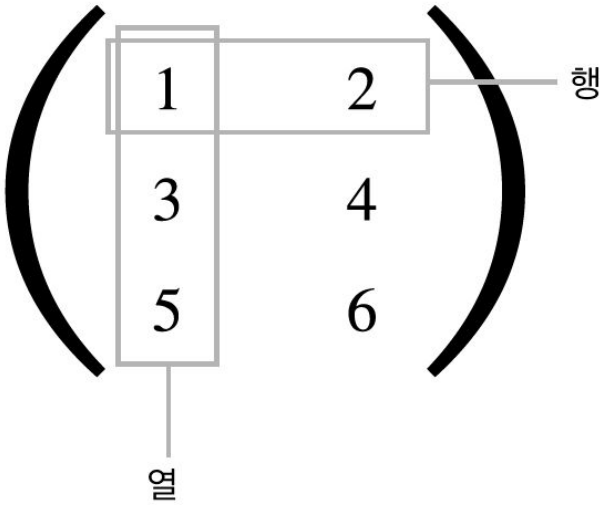
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



https://mc.ai/complete-guide-of-activation-functions/

Colab에서 확인하기

다차원 배열의 계산



2차원 배열의 행과 열

```
1 import numpy as np
2 A = np.array([1,2,3,4])
3 print(A)
```

```
[1 2 3 4]
```

```
1 B = np.array([[1,2], [3,4], [5,6]])
```

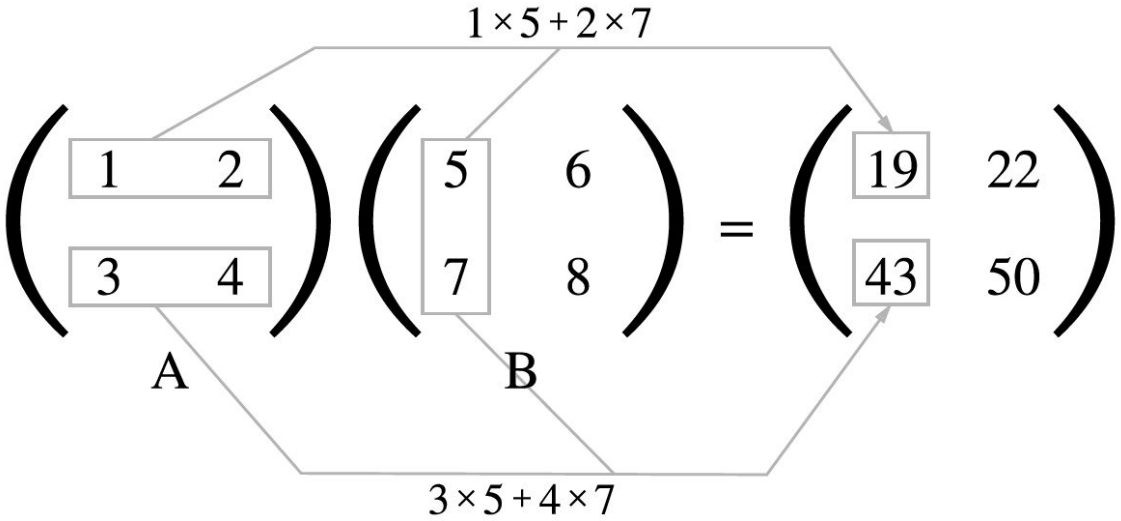
```
1 print(B)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

Colab에서 확인하기

다차원 배열의 계산

행렬의 내적 계산 방법

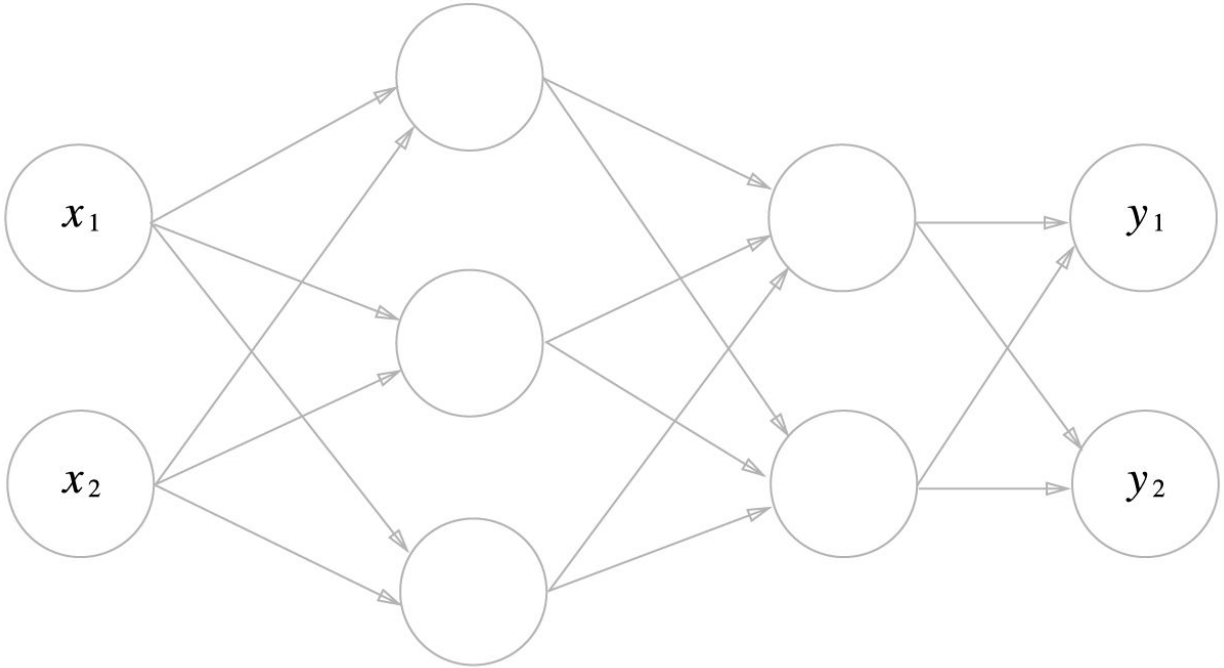



```
1 A = np.array([[1,2], [3,4]])
2 B = np.array([[5,6], [7,8]])
```

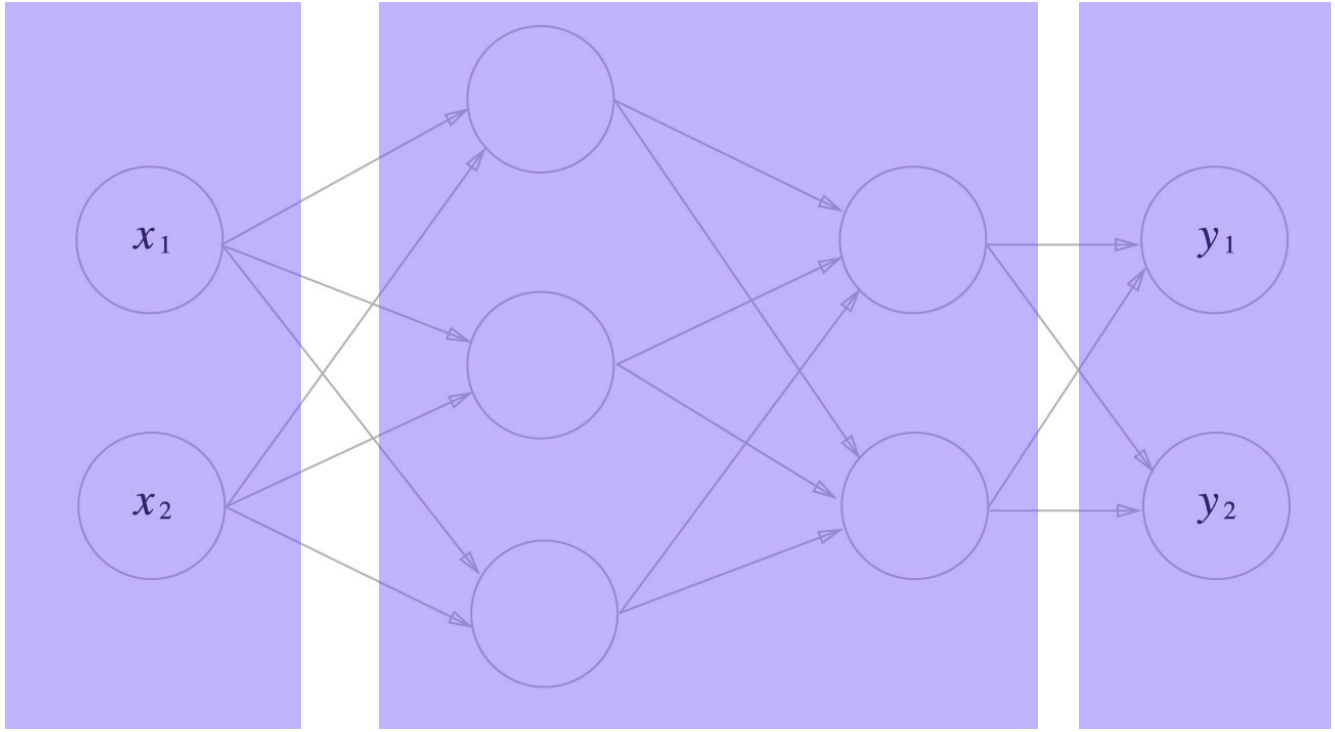
```
1 np.dot(A, B)
```

```
array([[19, 22],
       [43, 50]])
```

신경망 구현하기



신경망 구현하기



입력층
(Input layer)

은닉층
(Hidden layer)

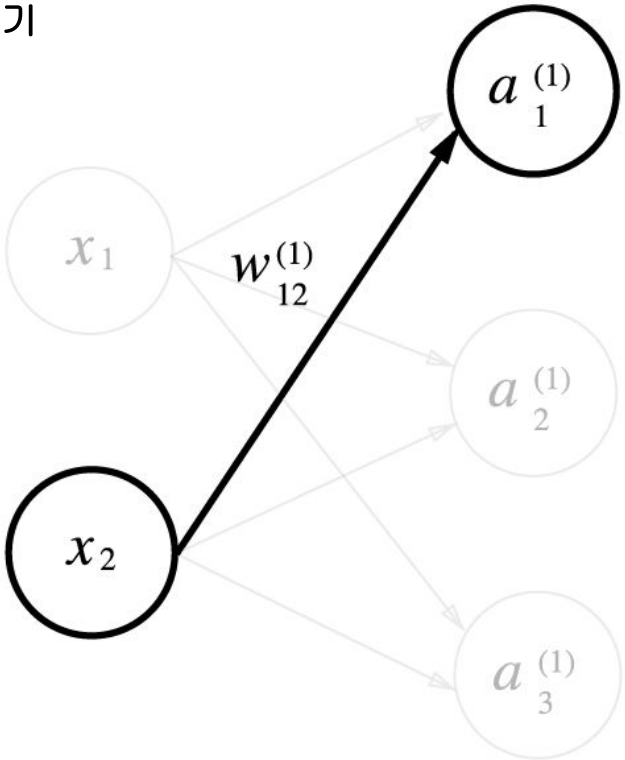
출력층
(Output layer)

지금까지 학습한 내용

- 퍼셉트론
- 활성화 함수
- 다차원 배열의 계산

신경망 구현하기

중요한 표기

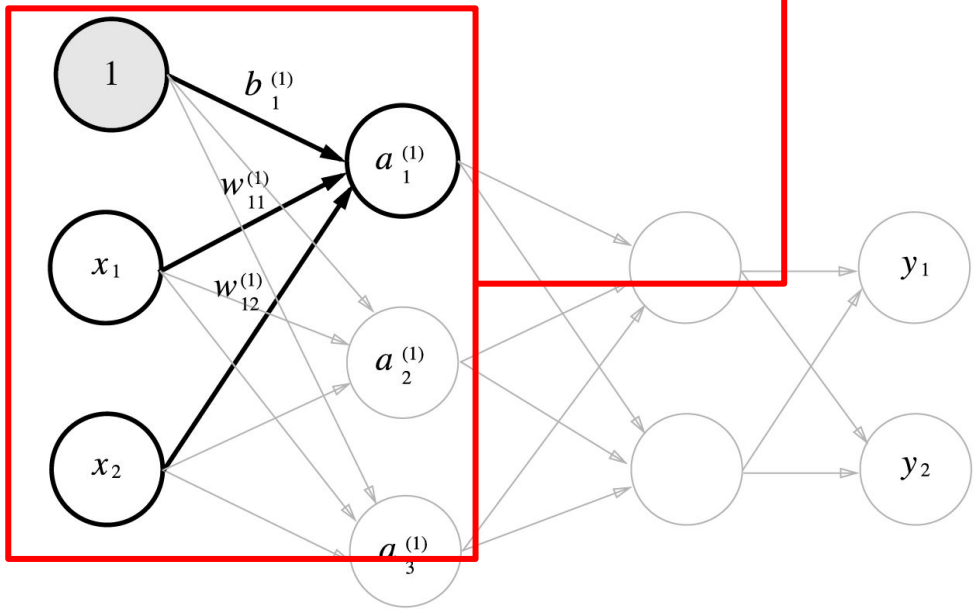


$w_{12}^{(1)}$

- (1): 1층의 가중치
- 1: 다음 층의 1번째 뉴런
- 2: 앞 층의 2번째 뉴런

신경망 구현하기

신호 전달 구현하기



$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

$$A^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)})$$

$$X = (x_1, x_2)$$

$$B^{(1)} = (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

신경망 구현하기

신호 전달 구현하기

$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

$$A^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)})$$

$$X = (x_1, x_2)$$

$$B^{(1)} = (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

```python
import numpy as np

X = np.array([1.0, 0.5])
W1 = np.array([[0.1, 0.3, 0.5],
               [0.2, 0.4, 0.6]])
B1 = np.array([0.1, 0.2, 0.3])

print(W1.shape)
print(X.shape)
print(B1.shape)

A1 = np.dot(X, W1) + B1
```

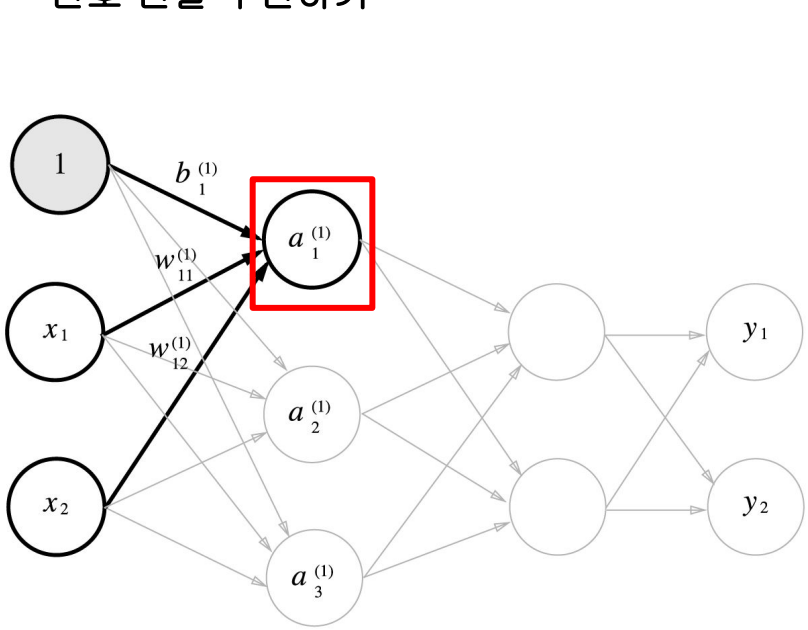
```
(2, 3)
(2,)
(3,)
```

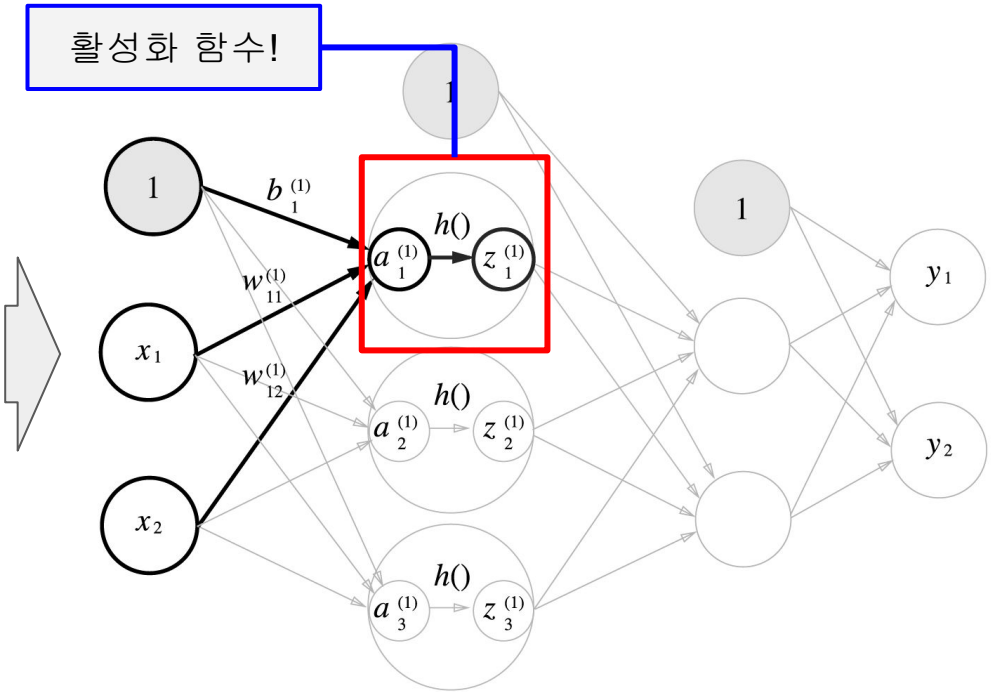
```python
print(A1)
```

```
[0.3 0.7 1.1]
```

신경망 구현하기

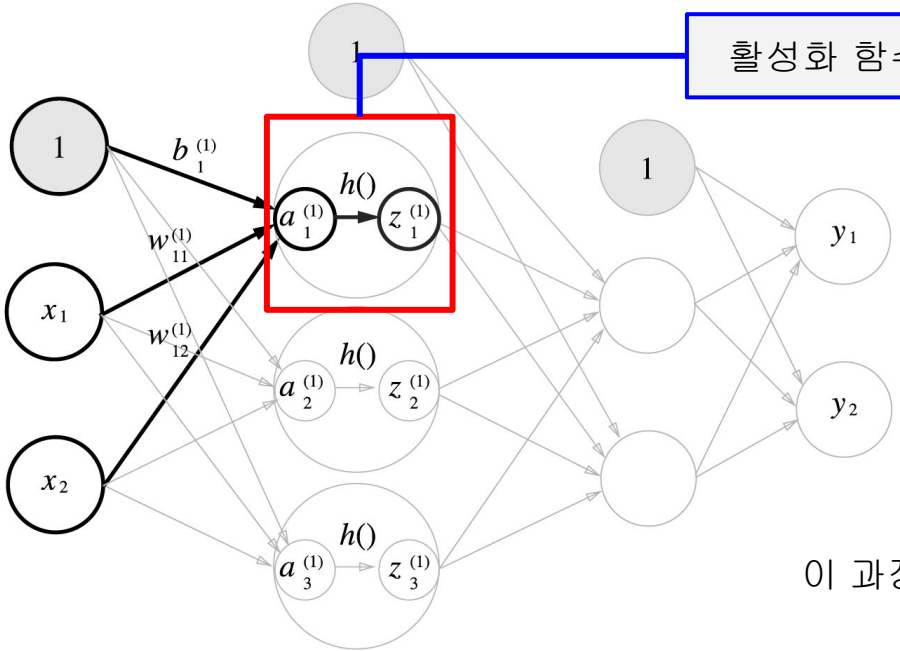
신호 전달 구현하기





신경망 구현하기

신호 전달 구현하기



활성화 함수!

```
1 Z1 = sigmoid(A1)
```

```
1 print(A1)
2 print(Z1)
```

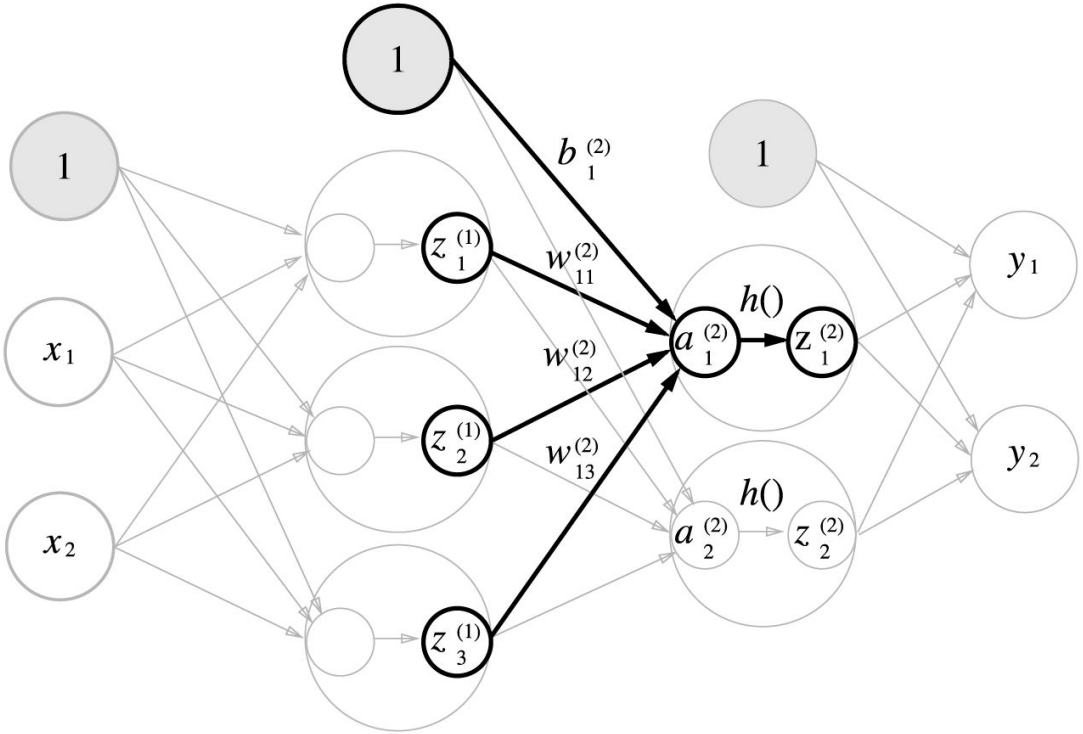
```
[0.3 0.7 1.1]
[0.57444252 0.66818777 0.75026011]
```

이 과정을 $A^{(1)} = (a^{(1)}_1, a^{(1)}_2, a^{(1)}_3)$ 에 대해 모두 수행

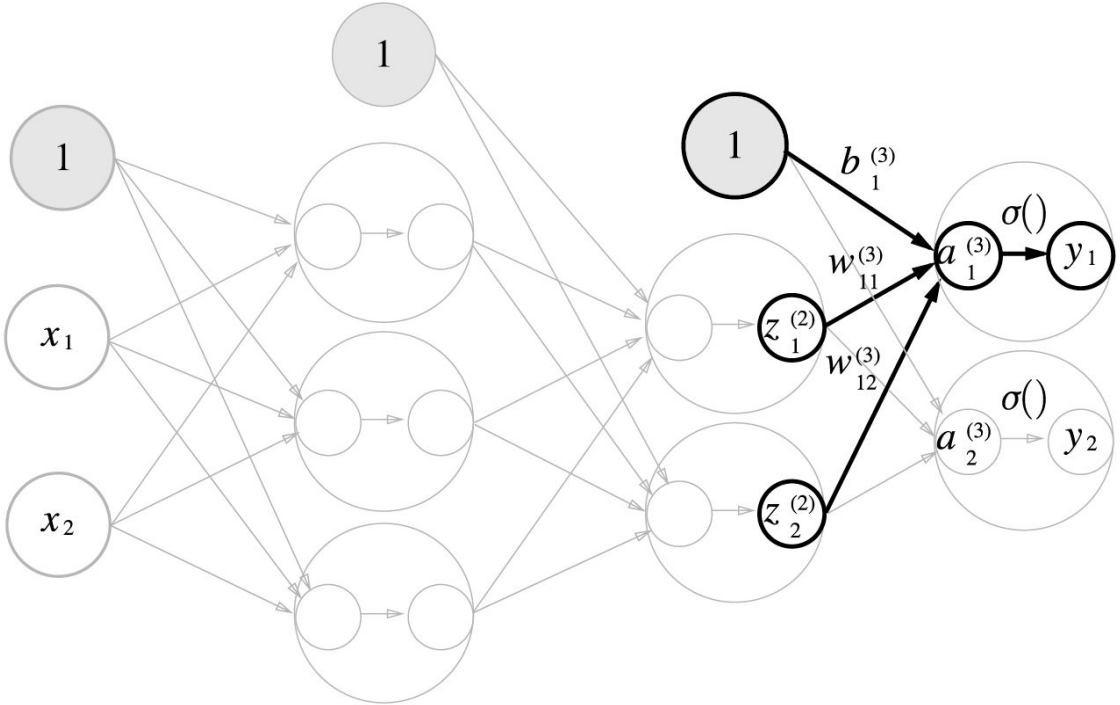
A1은 두번째 은닉층의 입력으로 전달됨

신경망 구현하기

신호 전달 구현하기

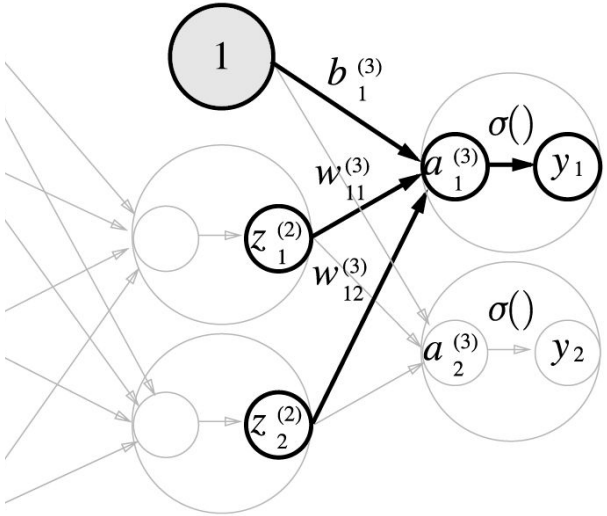


신경망 구현하기

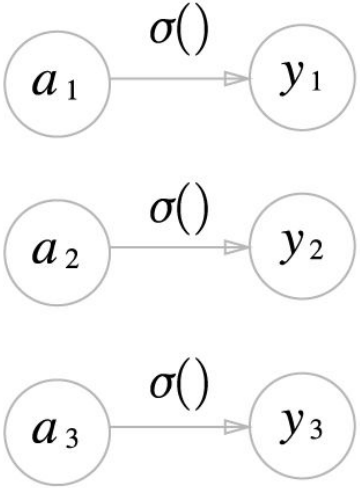


출력층 설계하기

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

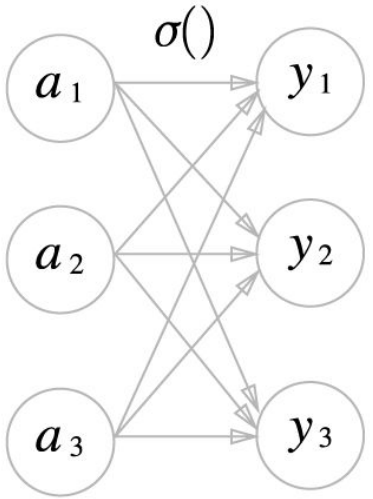


항등 함수



일반적으로
회귀 문제에 사용

소프트맥스 함수

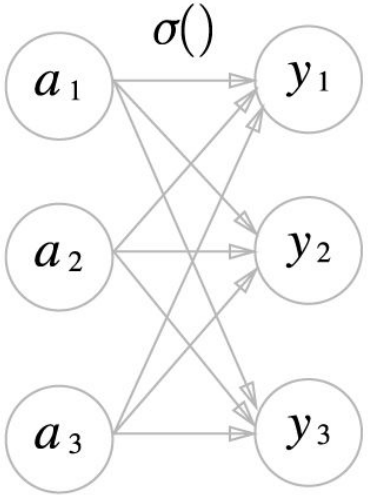


일반적으로
분류 문제에 사용

출력층 설계하기

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

소프트맥스 함수




```
1 a = np.array([0.3, 2.9, 4.0])
2
3 exp_a = np.exp(a)
4 print(exp_a)
```

```
[ 1.34985881 18.17414537 54.59815003]
```

```
1 sum_exp_a = np.sum(exp_a)
2 print(sum_exp_a)
```

```
74.1221542101633
```

```
1 y = exp_a / sum_exp_a
2 print(y)
```

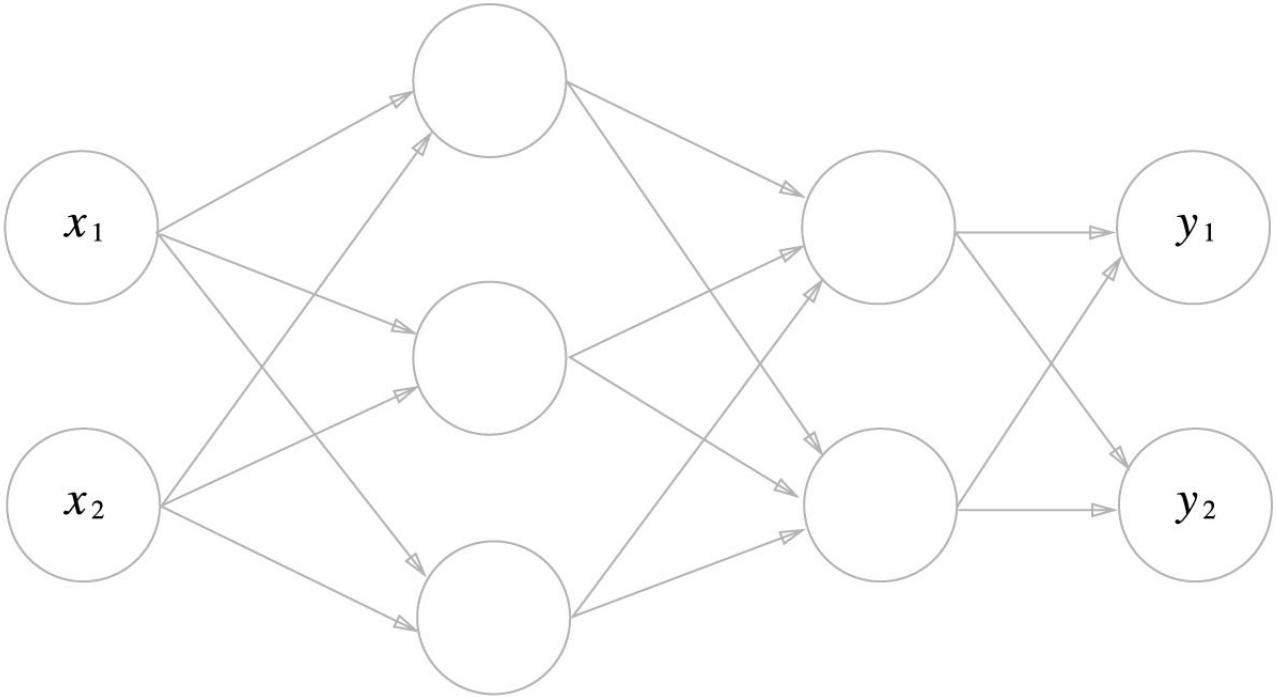
```
[0.01821127 0.24519181 0.73659691]
```

```
def softmax(a):
  exp_a = np.exp(a)
  sum_exp_a = np.sum(exp_a)
  y = exp_a / sum_exp_a

  return y
```

- 마지막 단계의 출력값을 정규화 시킴
- Softmax output의 합은 항상 1임
- Output Node의 개수는 예측하려는 Class 수

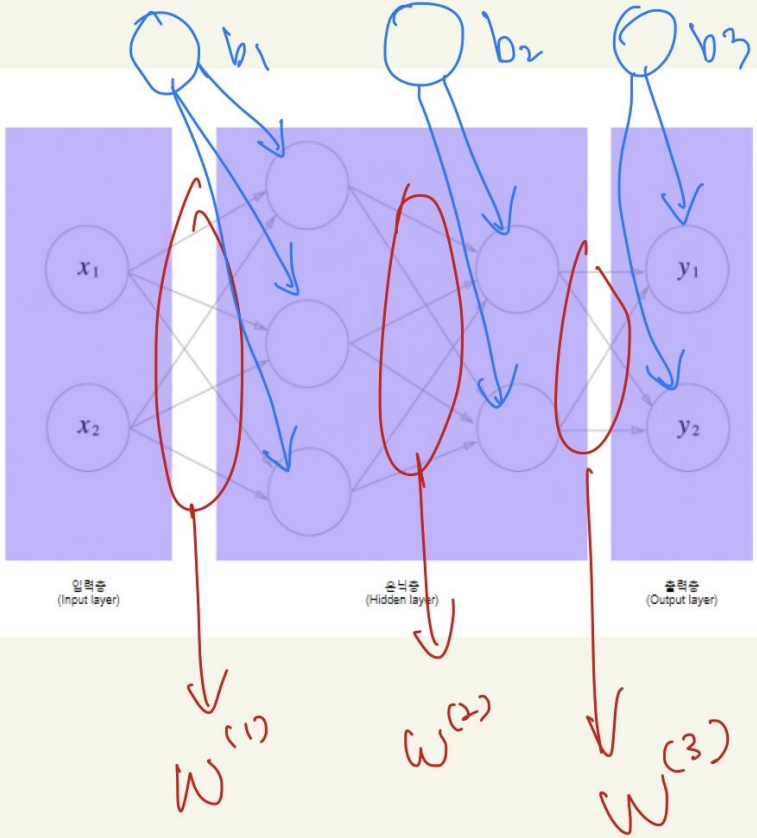
Colab에서 확인하기

구현 내용 정리



Colab에서 확인하기

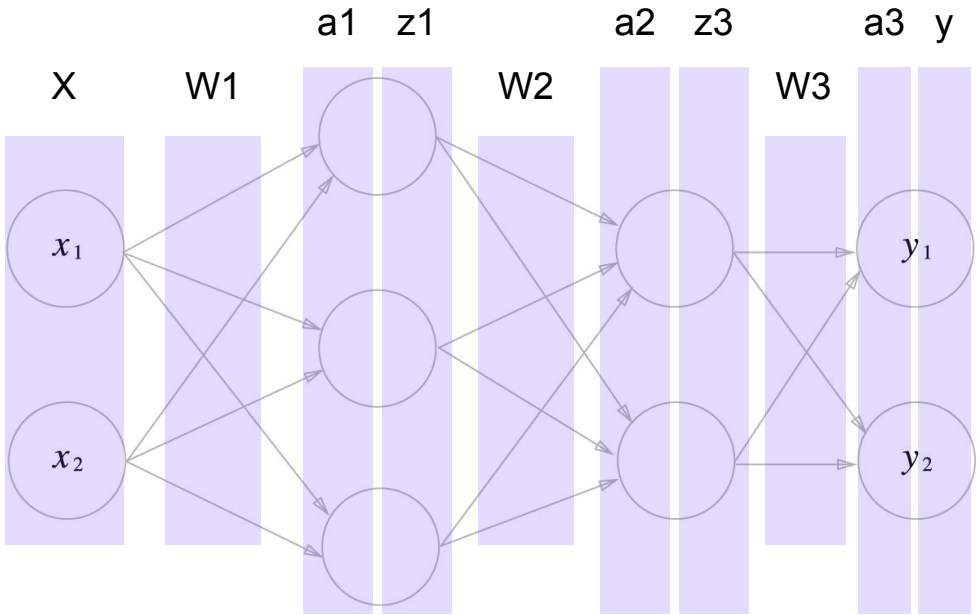
구현 내용 정리




```python
def init_network():
  network = {}
  network['W1'] = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
  network['b1'] = np.array([0.1, 0.2, 0.3])
  network['W2'] = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
  network['b2'] = np.array([0.1, 0.2])
  network['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])
  network['b3'] = np.array([0.1, 0.2])

  return network
```

Colab에서 확인하기

구현 내용 정리

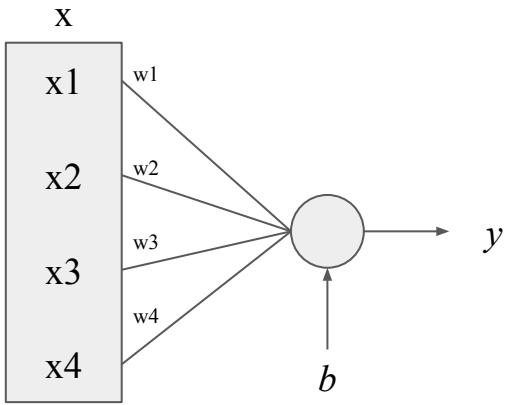



```
def forward(network, x):
  W1, W2, W3 = network['W1'], network['W2'], network['W3']
  b1, b2, b3 = network['b1'], network['b2'], network['b3']

  a1 = np.dot(x, W1) + b1
  z1 = sigmoid(a1)
  a2 = np.dot(z1, W2) + b2
  z2 = sigmoid(a2)
  a3 = np.dot(z2, W3) + b3
  y = identity_function(a3)

  return y
```

지금까지 학습한 내용

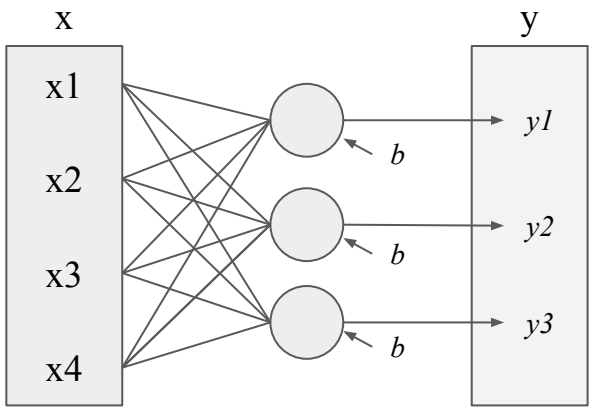
- 신경망 신호 전달 방식 구현
- 출력층 설계

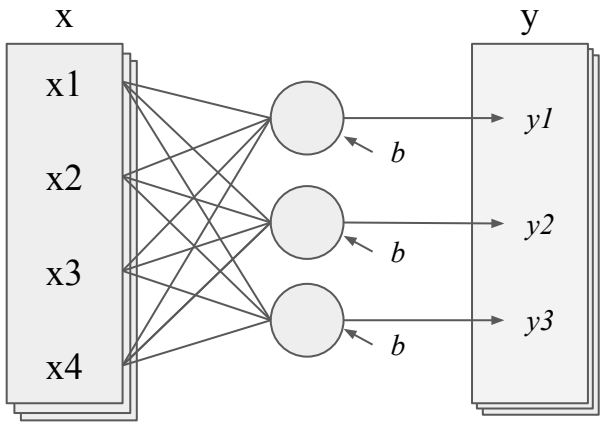
Perceptron - single layer



가중치 벡터	***w*** = (w1, w2, w3, w4)
입력 벡터	***x*** = (x1, x2, x3, x4)
가중치 편향	*b*
스칼라 출력	*y* = ***xw*** + *b*

Perceptron - multi layer

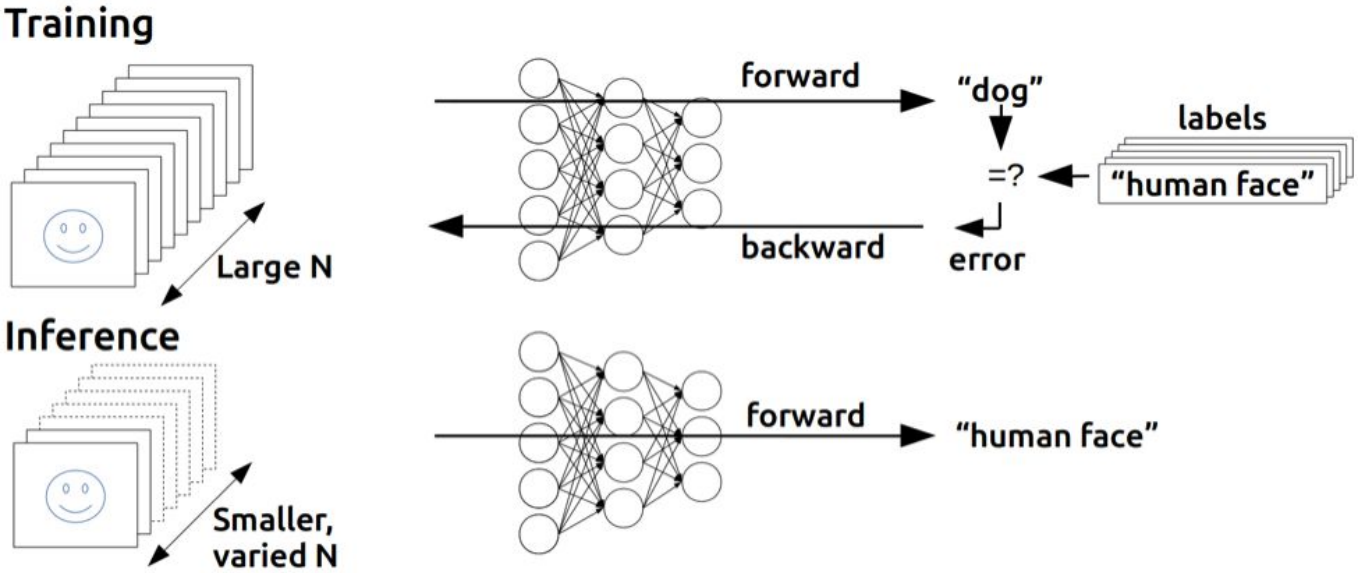




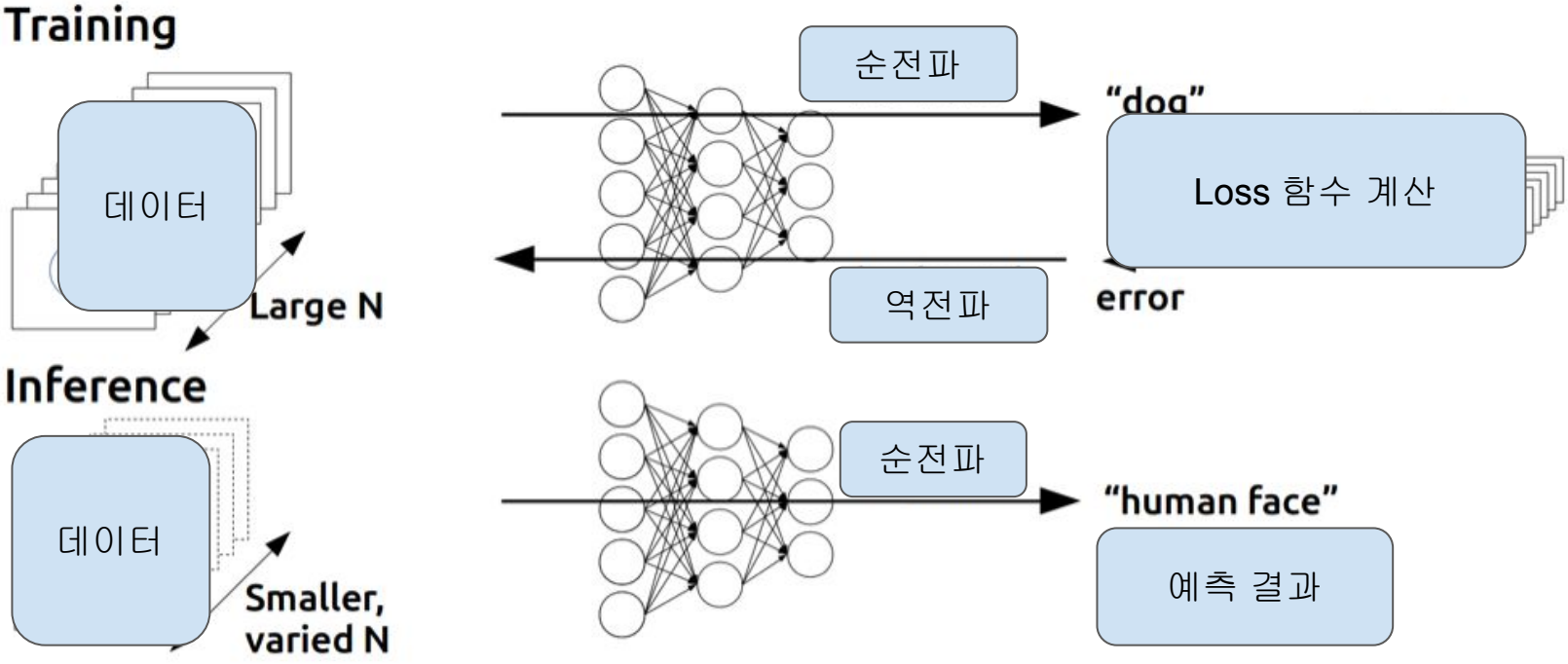
미니배치 데이터 처리

가중치 벡터 ***W*** = (w11, w12, … , w43, w44)
입력 벡터 ***x*** = (x1, x2, x3, x4)
가중치 편향 ***b*** = (b1, b2, b3)
출력 벡터 ***y = xW + b***

인공신경망을 학습시키는 방법

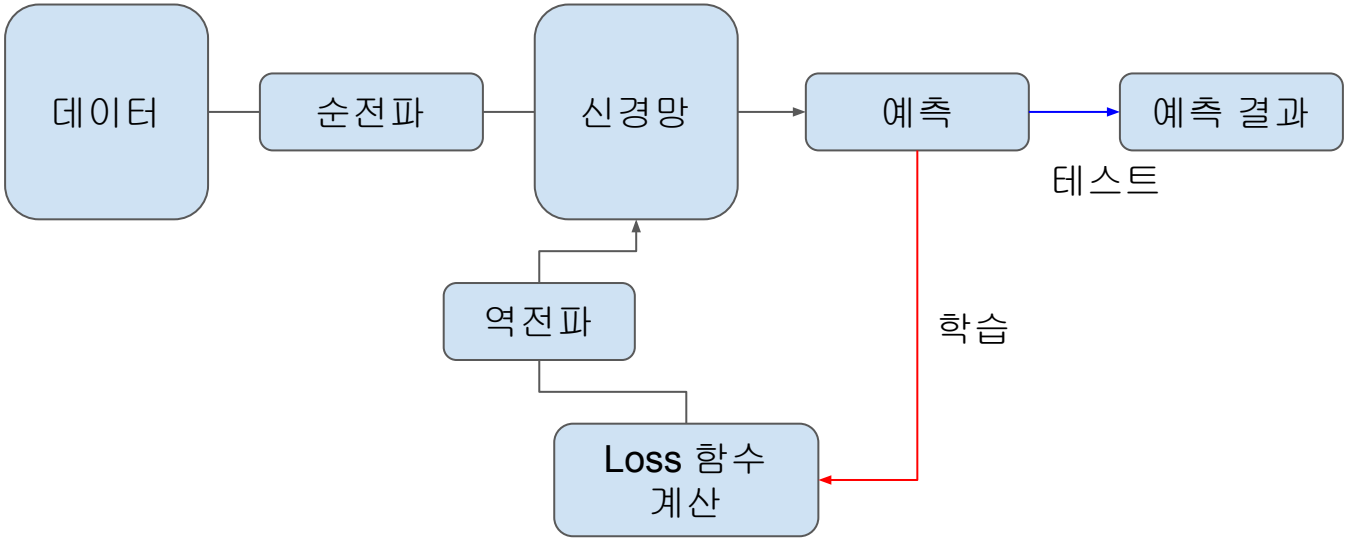


인공신경망을 학습시키는 방법



https://sungsoo.github.io/2016/05/07/inference-the-next-step-in-gpu-accelerated-deep-learning.html

Training Neural Network

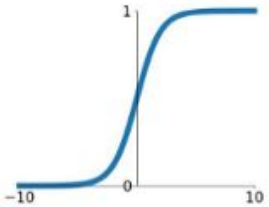


https://sungsoo.github.io/2016/05/07/inference-the-next-step-in-gpu-accelerated-deep-learning.html

Activation function

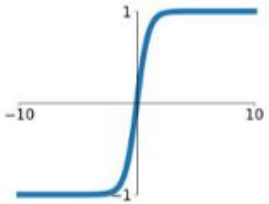
Sigmoid

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



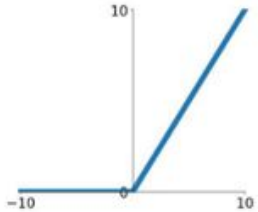
tanh

$$\tanh(x)$$



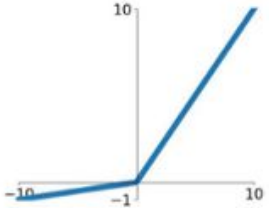
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

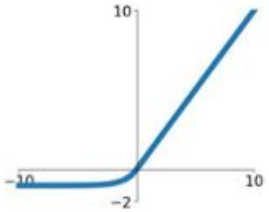


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

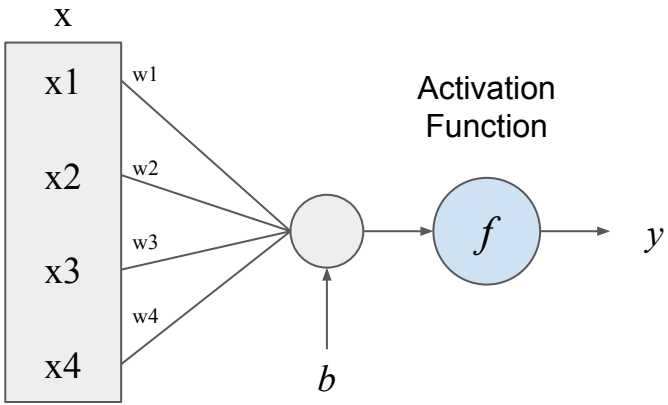
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



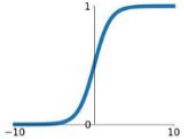
https://mc.ai/complete-guide-of-activation-functions/

Activation function



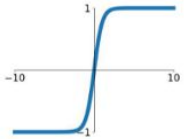
Sigmoid

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



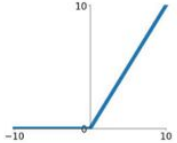
tanh

$\tanh(x)$



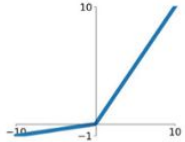
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

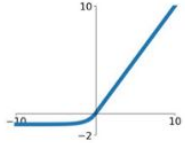


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

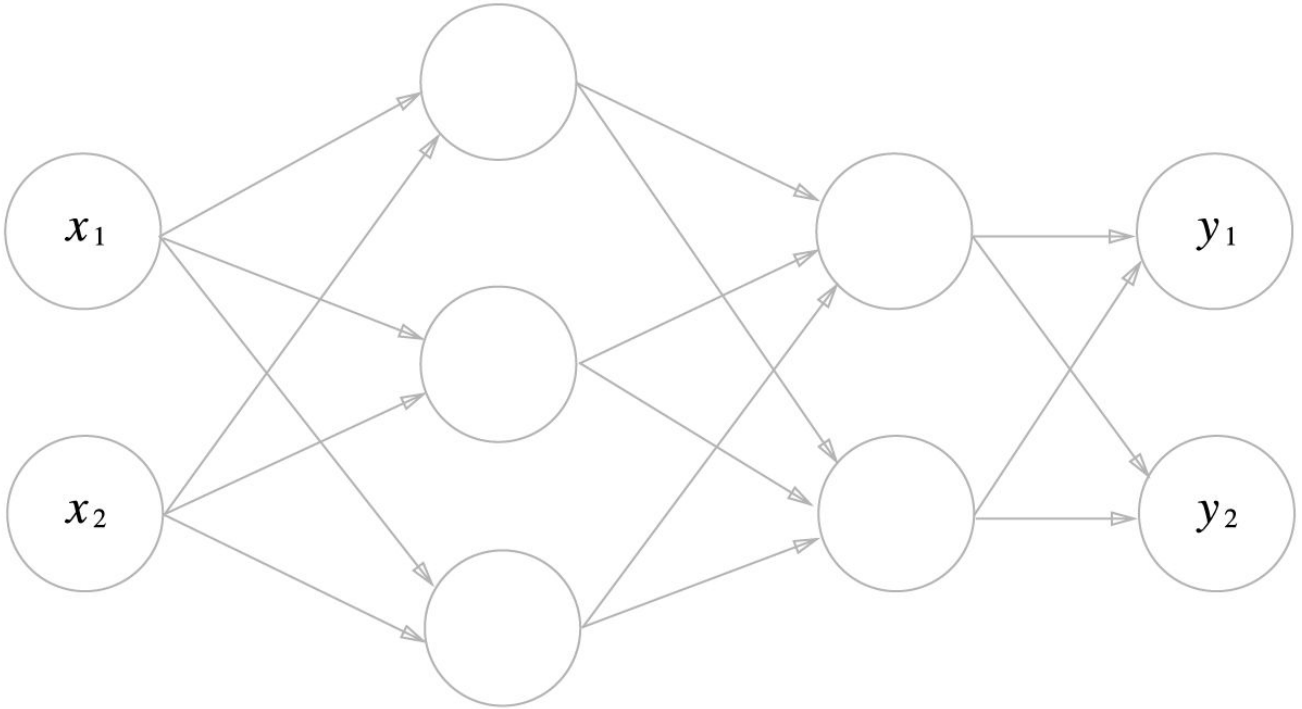
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

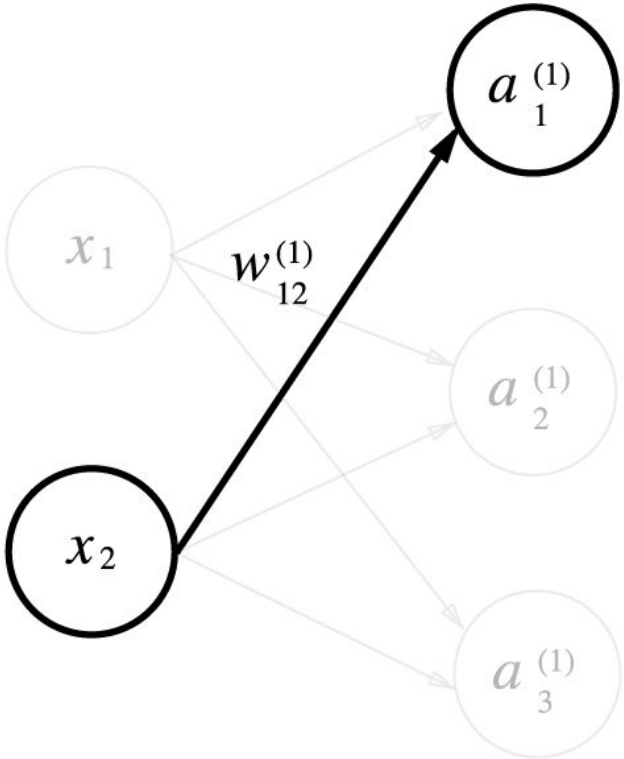


node에 계산된 값을 바로 다음 layer (output)으로 전달하지 않고,
비선형 함수 (activation function)을 통과 시킨 후 전달함
이 과정을 통해 layer를 깊게 쌓을 때 이점이 생김

Neural Network - 순전파 (Feed forward)



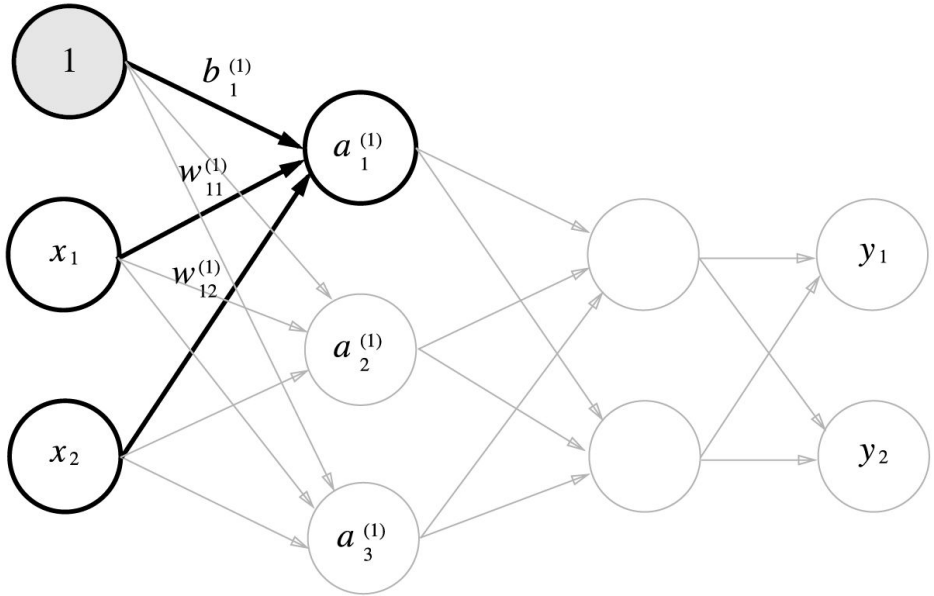
Neural Network



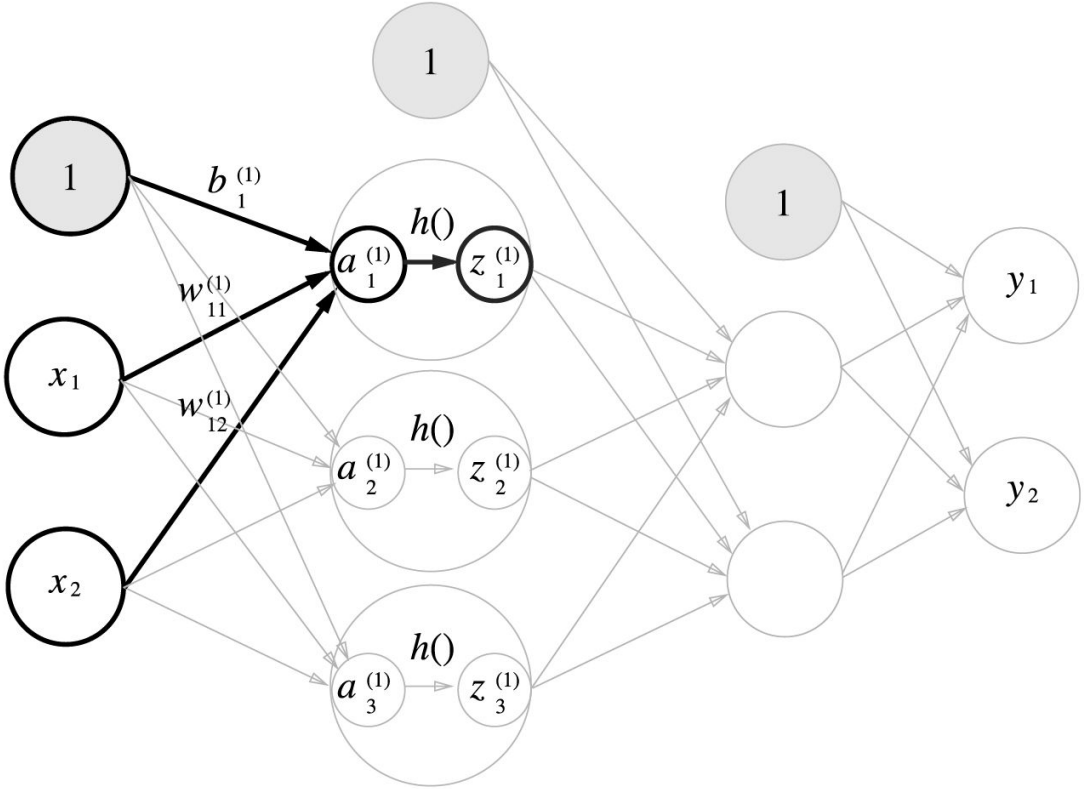
$$w_{12}^{(1)}$$

- (1): 1층의 가중치
- 2: 앞 층의 2번째 뉴런
- 1: 다음 층의 1번째 뉴런

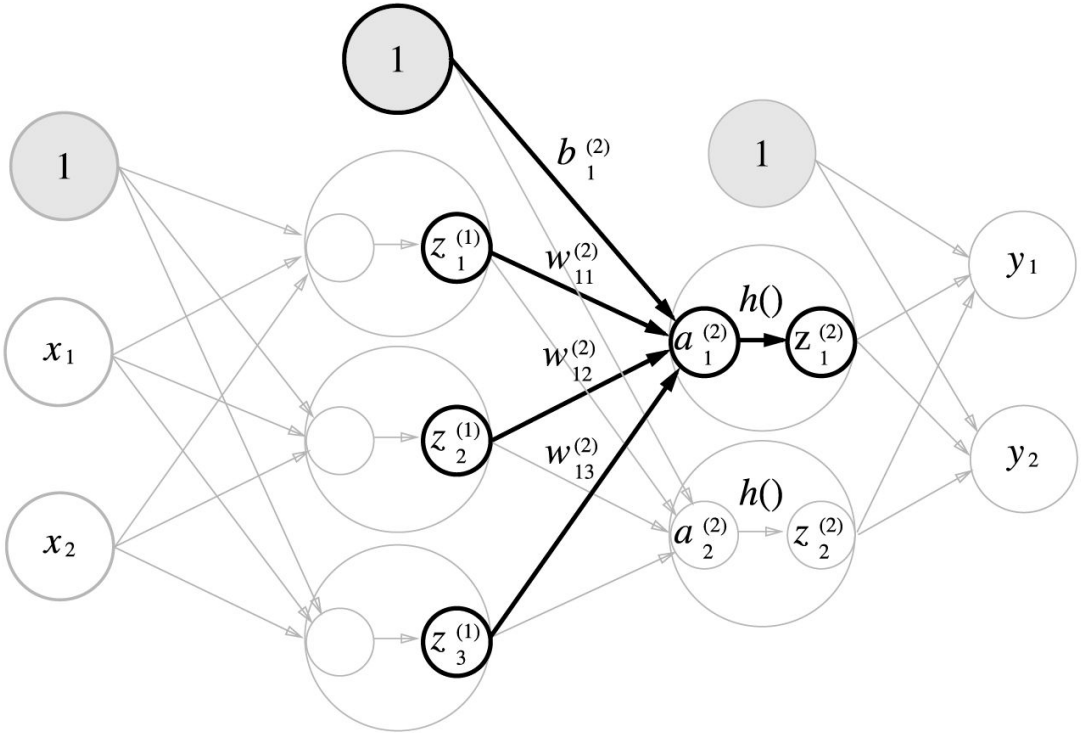
Neural Network



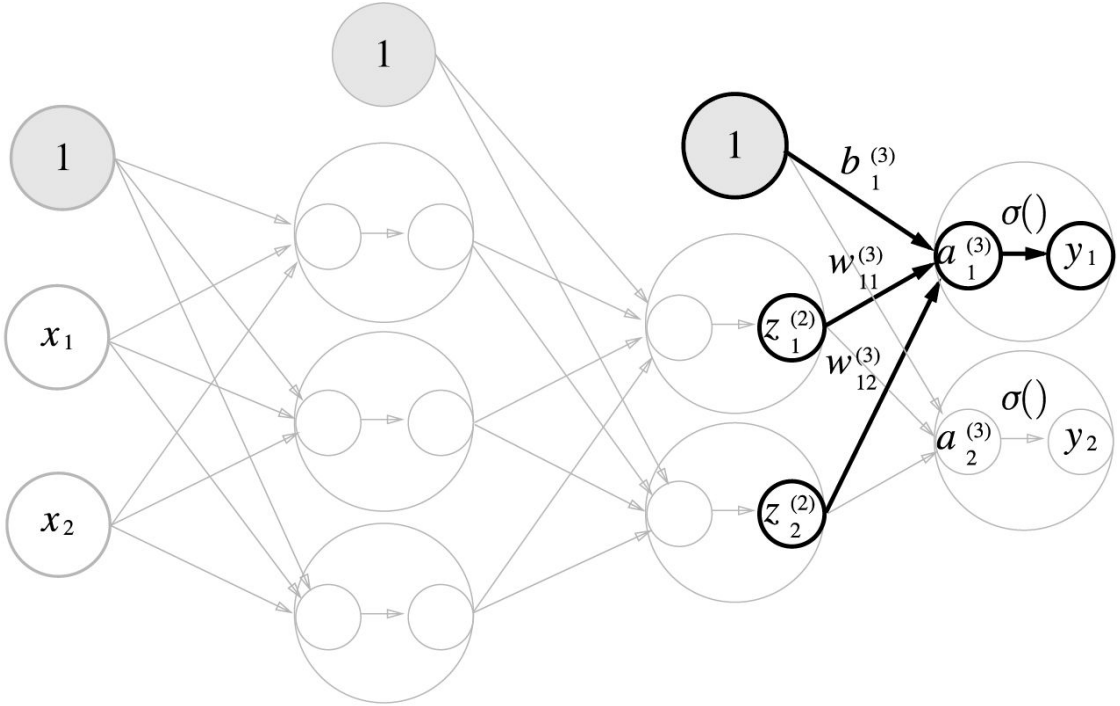
Neural Network



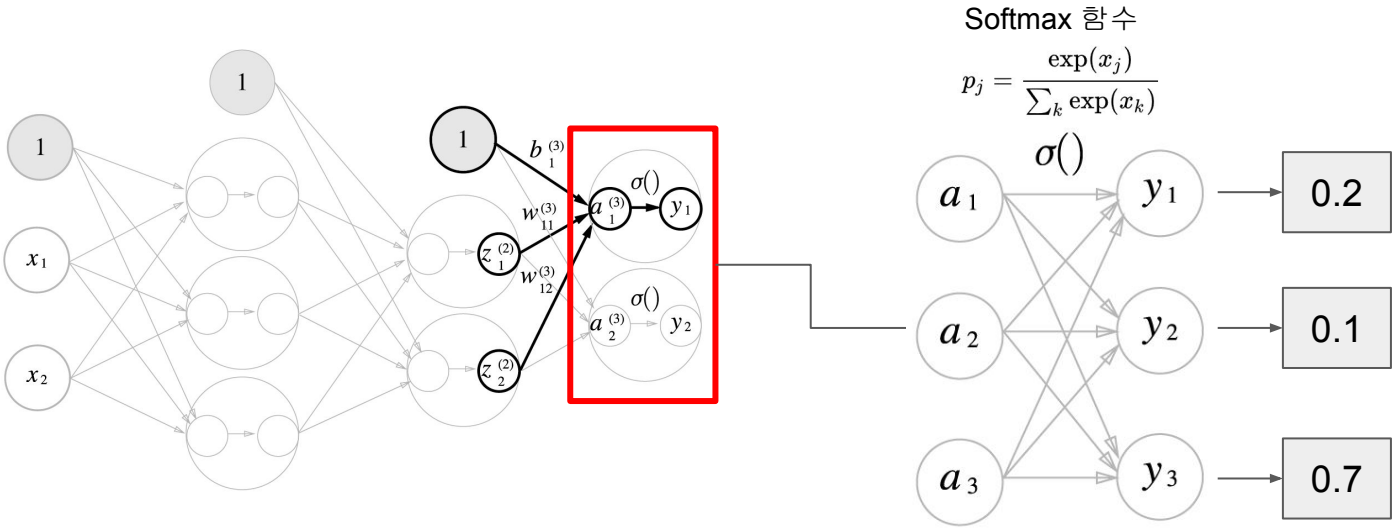
Neural Network



Neural Network



Neural Network - Softmax function



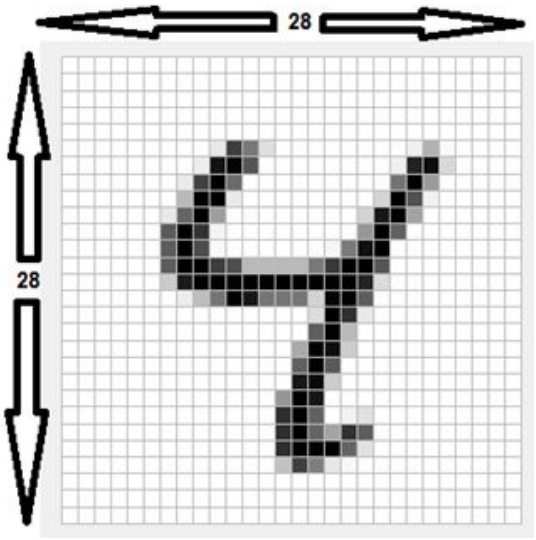
- 마지막 단계의 출력값을 정규화 시킴
- Softmax output의 합은 항상 1임
- Output Node의 개수는 예측하려는 Class 수

앞으로 학습 할 내용

- 가중치는 어떻게 결정되는가?
- 신경망에 실제 데이터를 학습하는 방식
- 신명망으로 분석할 수 있는 데이터들

데이터 추론 과정 (신경망 계산 과정 예시)

손글씨 숫자 데이터 (MNIST dataset)

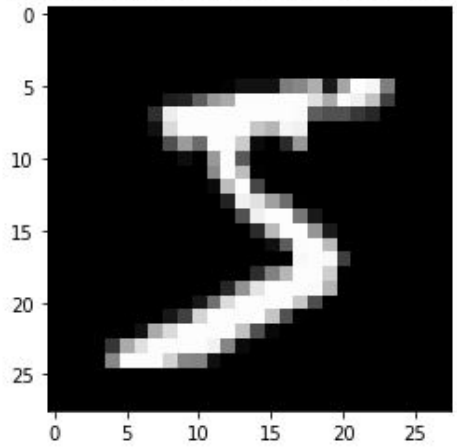


데이터 추론 과정

```
[ ]  1 import tensorflow as tf
     2 mnist = tf.keras.datasets.mnist
     3 (x_train, y_train),(x_test, y_test) = mnist.load_data()
```

```
1 plt.imshow(x_train[0], cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f4313a36358>
```



데이터
확인하기

```
1 for i in range(len(x_train[0])):
2   for j in range(len(x_train[0][i])):
3     if x_train[0][i][j] > 0:
4       print(1, end=' ')
5     else:
6       print(0, end=' ')
7   print('')
```

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

데이터 추론 과정

원-핫 인코딩One-hot Encoding

[1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 1, 0, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]

[0, 0, 0, 0, 1, 0, 0, 0, 0, 0]

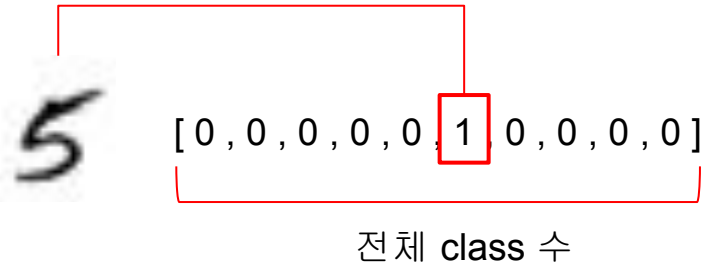
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 1, 0, 0, 0]

[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]

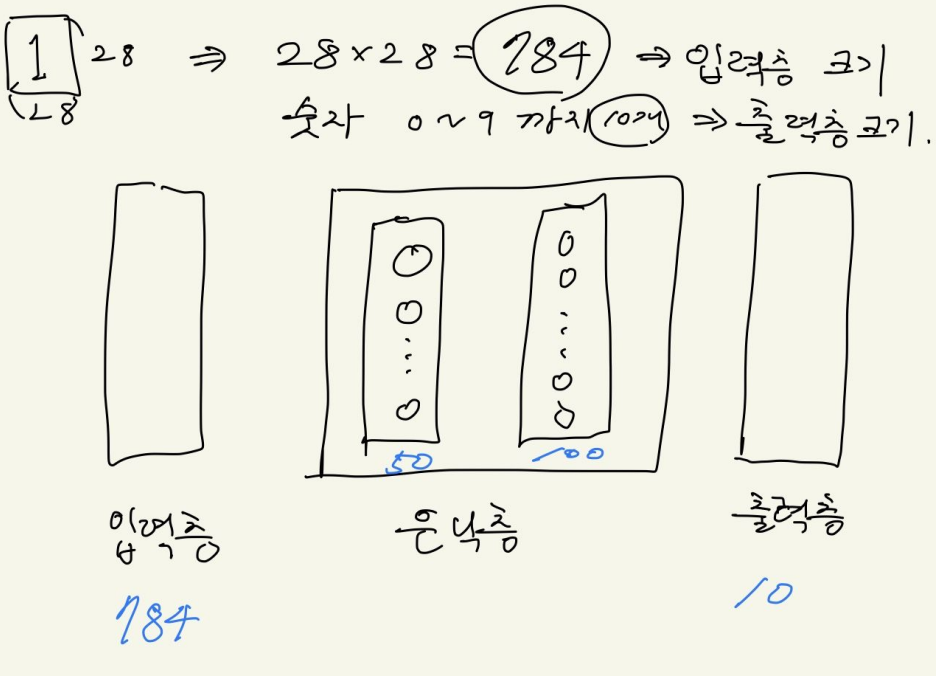
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0]

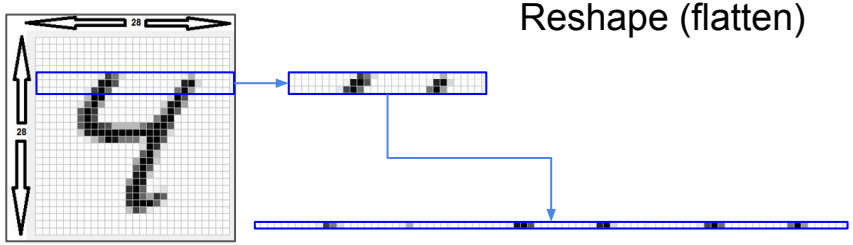
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1]



데이터 추론 과정

Colab에서 확인하기





Reshape (flatten)

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 픽셀 값을 0~1 사이로 정규화합니다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
x_test_flatten = x_test.reshape([x_test.shape[0], -1])
```

데이터 추론 과정

Colab에서 확인하기

```python
def init_network():
    with open("sample_weight.pkl", 'rb') as f:
        network = pickle.load(f)
    return network


def predict(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = softmax(a3)

    return y
```

앞서 학습한 **predict** 코드와 같은 코드

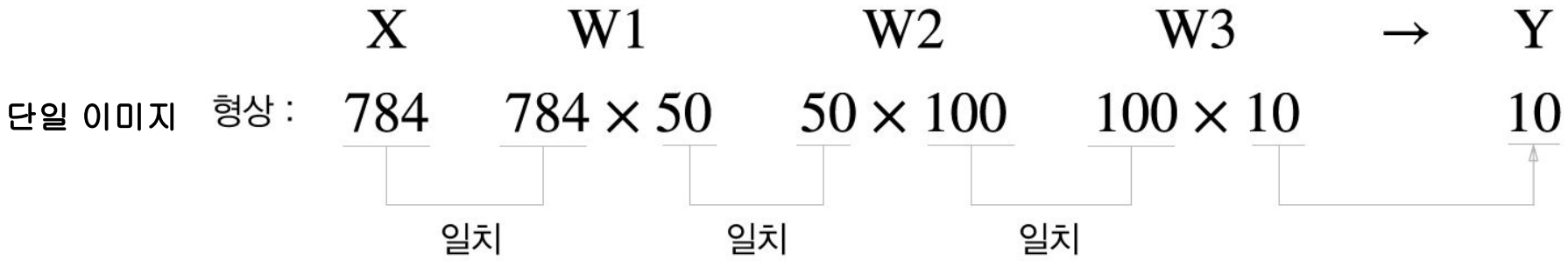
- MNIST 분류에 최적화 돼 있는 가중치

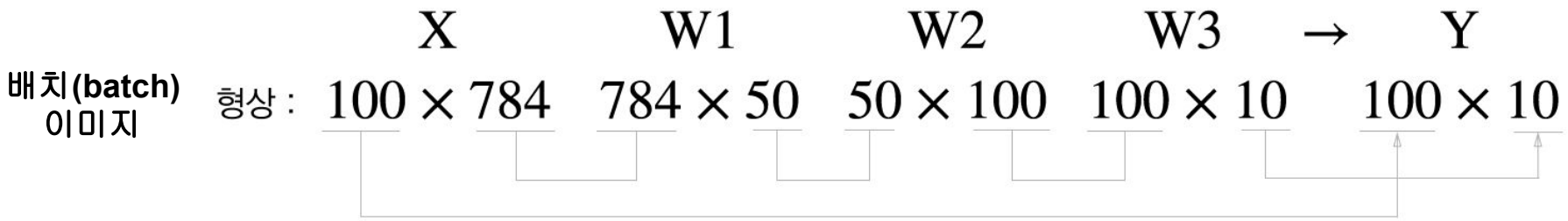
```
1 network
```

```
{'W1': array([[-0.00741249, -0.00790439, -0.01307499, ...,  0.01978721,
         -0.04331266, -0.01350104],
        [-0.01029745, -0.01616653, -0.01228376, ...,  0.01920228,
          0.02809811,  0.01450908],
        [-0.01309184, -0.00244747, -0.0177224 , ...,  0.00944778,
          0.01387301,  0.03393568],
        ...,
        [ 0.02242565, -0.0296145 , -0.06326169, ..., -0.01012643,
          0.01120969,  0.01027199],
        [-0.00761533,  0.02028973, -0.01498873, ...,  0.02735376,
         -0.01229855,  0.02407041],
        [ 0.00027915, -0.06848375,  0.00911191, ..., -0.03183098,
          0.00743086, -0.04021148]], dtype=float32),
 'W2': array([[-0.10694039,  0.01591247, -0.44349867, ...,  0.03561032,
          0.14045963,  0.03964241],
        [ 0.29911557, -0.03322235, -0.08902215, ..., -0.04722451,
         -0.0972147 ,  0.2950258 ],
        [ 0.06576645,  0.6330455 ,  0.02325344, ...,  0.05046809,
          0.26831996, -0.13252524],
        ...,
        [-0.1839421 , -0.10925075,  0.25180233, ...,  0.06017017,
          0.11689074,  0.28868544],
        [ 0.10001627,  0.0899286 , -0.03874066, ...,  0.15217757,
         -0.05744234, -0.00713823],
        [-0.02220659, -0.05105179,  0.00777963, ..., -0.531206  ,
         -0.4042084 ,  0.0090801 ]], dtype=float32),
```

데이터 추론 과정

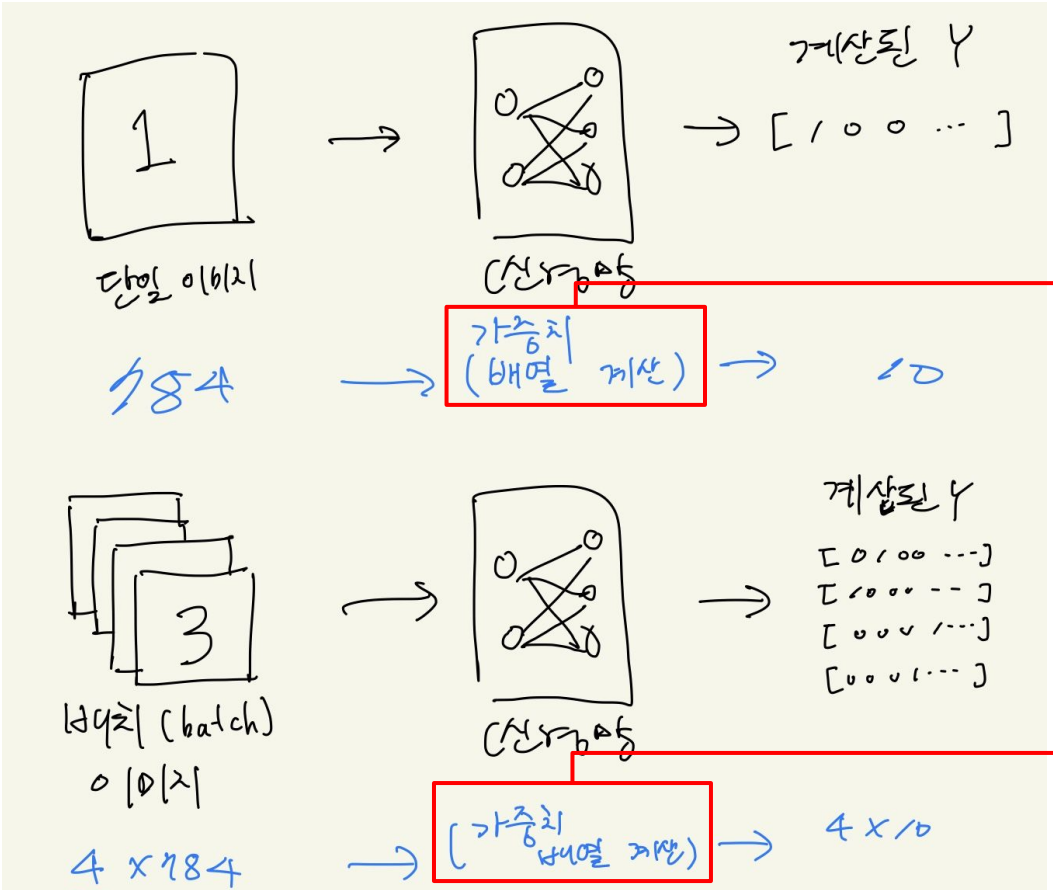
신경망 각 층의 형상(shape) 추이





데이터 추론 과정

Colab에서 확인하기



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

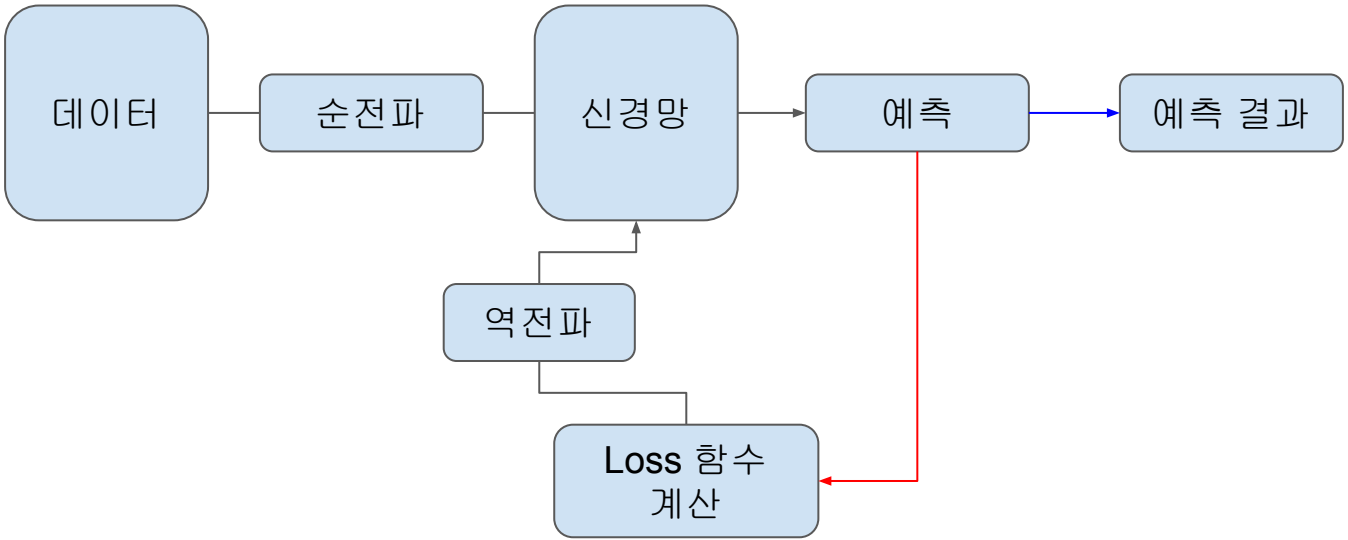
	X	W1	W2	W3	→	Y
형상 :	100 × 784	784 × 50	50 × 100	100 × 10		100 × 10

데이터 추론 과정

- 신경망에서는 활성화 함수로 **시그모이드 함수**와 ReLU 함수 같은 **매끄럽게 변화하는 함수**를 이용한다.
- 넘파이의 **다차원 배열**을 잘 사용하면 신경망을 효율적으로 구현할 수 있다.
- 기계학습 문제는 크게 **회귀와 분류**로 나눌 수 있다.
- 출력층의 활성화 함수로는 **회귀**에서는 주로 **항등 함수**를, **분류**에서는 주로 **소프트맥스 함수**를 이용한다.
- **분류**에서는 **출력층의 뉴런 수**를 분류하려는 **클래스 수와 같게** 설정한다.
- 입력 데이터를 묶은 것을 **배치**라 하며, 추론 처리를 이 **배치 단위로 진행**하면 결과를 훨씬 빠르게 얻을 수 있다.

신경망 학습

신경망 학습



손실함수 Loss functions

손실함수란?

신경망 성능이 **얼마나 나쁜지** 나타내는 지표

일반적으로 신경망이 예측한 결과와 정답을 비교하여 얼마나 차이가 있는지 수치화 함

차이를 계산하는 방식은 여러가지가 있음

대표적인 손실 함수

평균 제곱 오차
(Mean Squared Error, MSE)

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

교차 엔트로피 오차
(Cross Entropy Error, CEE)

$$E = -\sum_{k} t_k \log y_k$$

손실함수 Loss functions

평균 제곱 오차 (Mean Squared Error, MSE) $E = \frac{1}{2}\sum_k (y_k - t_k)^2$

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

Label	예측	정답	차이^2
0	0.1	0	0.01
1	0.05	0	0.0025
2	0.6	1	0.16
3	0	0	0
4	0.05	0	0.0025
5	0.1	0	0.01
6	0	0	0
7	0.1	0	0.01
9	0	0	0
		합계	0.195

$E = \frac{1}{2}\sum_k (y_k - t_k)^2$ 0.0975

Label	예측	정답	차이^2
0	0.1	0	0.01
1	0.05	0	0.0025
2	0.6	0	0.36
3	0	0	0
4	0.05	0	0.0025
5	0.1	0	0.01
6	0	0	0
7	0.1	1	0.81
9	0	0	0
		합계	1.195

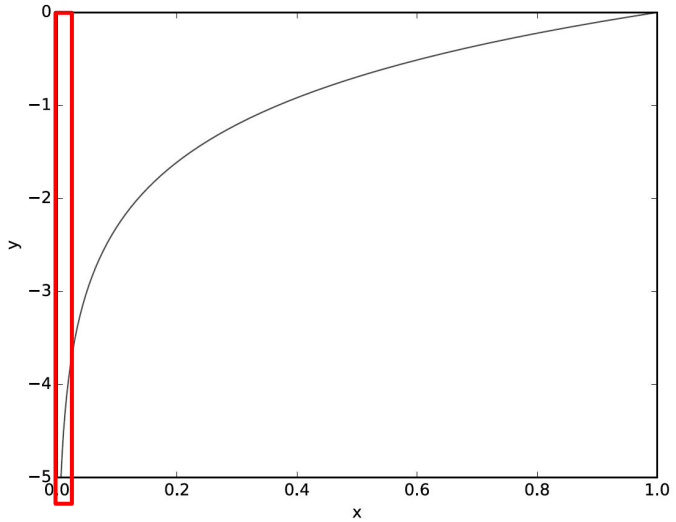
$E = \frac{1}{2}\sum_k (y_k - t_k)^2$ 0.5975

정답을 잘 맞추지 못할수록 손실 함수 계산 값은 커짐

손실함수 Loss functions

교차 엔트로피 오차 (Cross Entropy Error, CEE) $E = -\sum_k t_k \log y_k$

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



Label	예측	예측+1e-7	정답	t_k * log y_k
0	0.1	0.1000001	0	0.00
1	0.05	0.0500001	0	0.00
2	0.6	0.6000001	1	-0.51
3	0	0.0000001	0	0.00
4	0.05	0.0500001	0	0.00
5	0.1	0.1000001	0	0.00
6	0	0.0000001	0	0.00
7	0.1	0.1000001	0	0.00
9	0	0.0000001	0	0.00
			합계	0.51

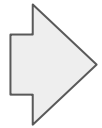
$E = -\sum_k t_k \log y_k$

자연로그에서 x가 0을 가질땐 음의 무한대로 수렴하므로
아주작은 값을 예측값에 더해줘서 계산 가능한 형태로 만들어
주어야 함

미니배치 학습에서 손실함수

교차 엔트로피 오차 (Cross Entropy Error, CEE)

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



$$E = -\frac{1}{N}\sum_n \sum_k t_{nk} \log y_{nk}$$

n개의 배치에 대한 평균 손실 함수를 구하는 것

```
# 배치용 교차 엔트로피 오차
def cross_entropy_error(y, t):
  if y.ndim == 1:
    t = t.reshape(1, t.size)
    y = y.reshape(1, y.size)

  batch_size = y.shape[0]
  return -np.sum(t * np.log(y)) / batch_size
```

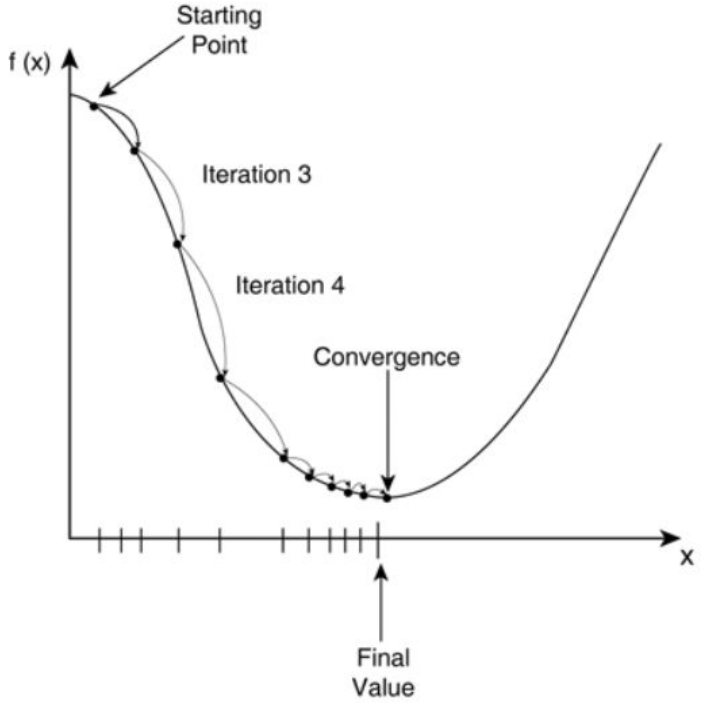
손실 함수 (Loss function)

- MSE (Mean Squared Error) - L2 Loss
- MAE (Mean Absolute Error) - L1 Loss
- CEE (Cross Entropy Error)
- ...

symbol	name	equation
$\mathcal{L}_1$	L_1 loss	$\|\mathbf{y}-\mathbf{o}\|_1$
$\mathcal{L}_2$	L_2 loss	$\|\mathbf{y}-\mathbf{o}\|_2^2$
$\mathcal{L}_1 \circ \sigma$	expectation loss	$\|\mathbf{y}-\sigma(\mathbf{o})\|_1$
$\mathcal{L}_2 \circ \sigma$	regularised expectation loss[1]	$\|\mathbf{y}-\sigma(\mathbf{o})\|_2^2$
$\mathcal{L}_\infty \circ \sigma$	Chebyshev loss	$\max_j \lvert\sigma(\mathbf{o})^{(j)}-\mathbf{y}^{(j)}\rvert$
hinge	hinge [13] (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})$
hinge^2	squared hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^2$
hinge^3	cubed hinge (margin) loss	$\sum_j \max(0, \frac{1}{2}-\hat{\mathbf{y}}^{(j)}\mathbf{o}^{(j)})^3$
log	log (cross entropy) loss	$-\sum_j \mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}$
log^2	squared log loss	$-\sum_j [\mathbf{y}^{(j)} \log \sigma(\mathbf{o})^{(j)}]^2$
tan	Tanimoto loss	$\frac{-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2^2+\|\mathbf{y}\|_2^2-\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}$
D_{CS}	Cauchy-Schwarz Divergence [3]	$-\log \frac{\sum_j \sigma(\mathbf{o})^{(j)}\mathbf{y}^{(j)}}{\|\sigma(\mathbf{o})\|_2\|\mathbf{y}\|_2}$

경사 하강법

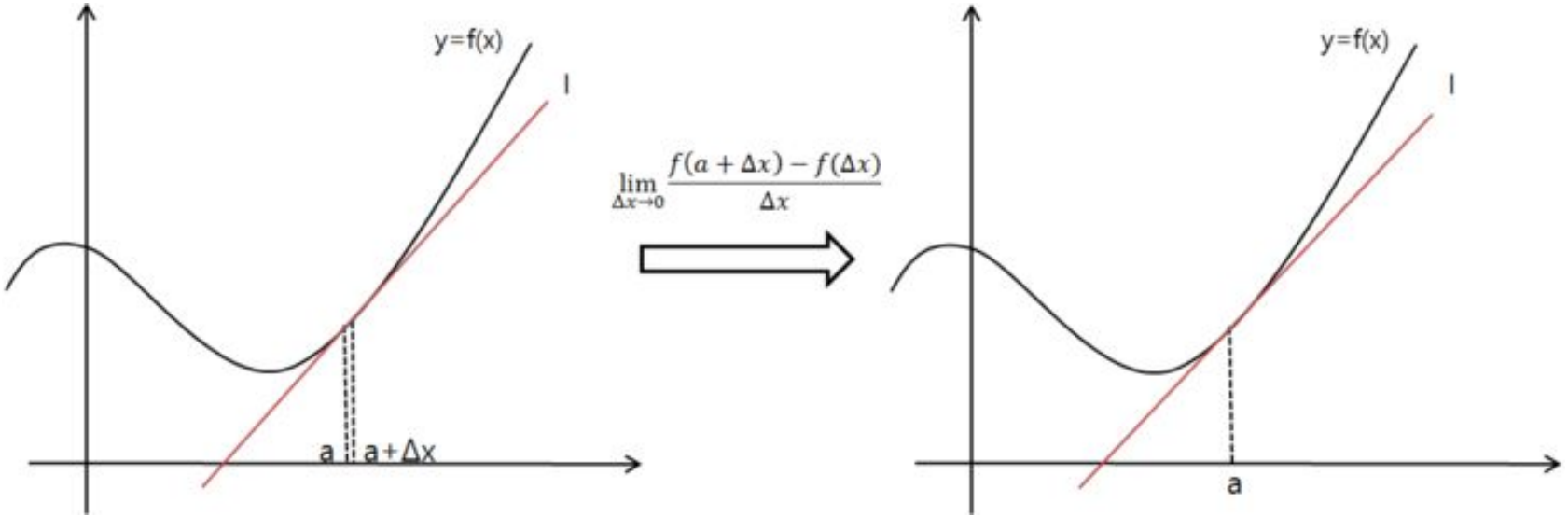
- 초기값 설정
- Cost function 계산
- 파라미터 값 업데이트
- 과정 반복



경사 하강법을 학습하기전에...

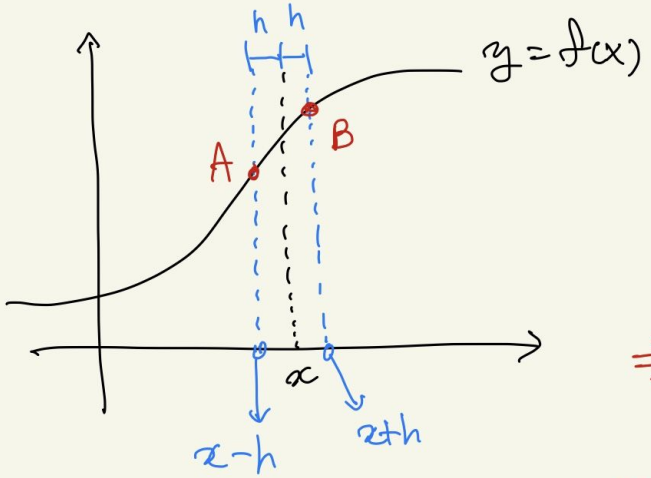
미분, 편미분, 기울기 등 개념 복습

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



경사 하강법을 학습하기전에...

"수치 미분"



A, B 에 대한 기울기는?

$$\frac{y\text{증가분}}{x\text{증가분}}$$

$$\Rightarrow \frac{f(x+h)-f(x-h)}{(x+h)-(x-h)}$$

$$= \frac{f(x+h)-f(x-h)}{2h}$$

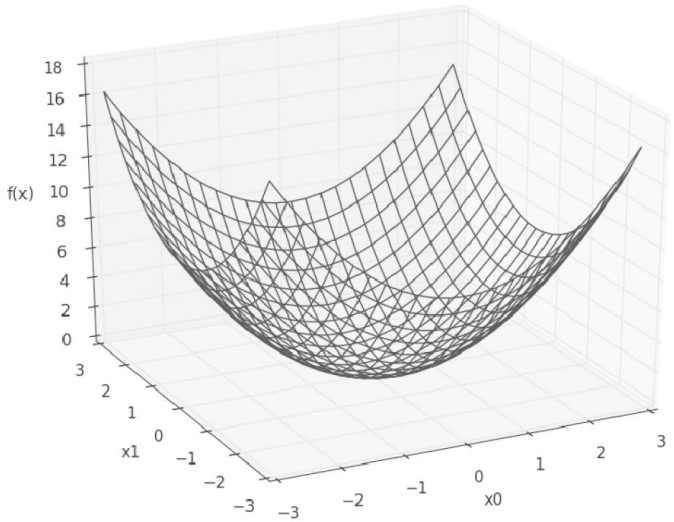
- "h"가 매우 작은 값으로 설정하면 x에서의 실제 기울기와 유사한 값을 얻을수 있다!

```
def numerical_diff(f, x):
  h = 1e-4 # 0.0001
  return (f(x+h) - f(x-h)) / (2*h)
```

경사 하강법을 학습하기전에...

편미분

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



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

"x_0 에 대해 편미분" : $\frac{\partial f}{\partial x_0}$

"x_1 " " : $\frac{\partial f}{\partial x_1}$

수치 미분을 편미분에 적용한다면?

미분하려는 변수를 제외한 다른 값들은 고정한 뒤 수치 미분을 적용함

경사 하강법

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

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

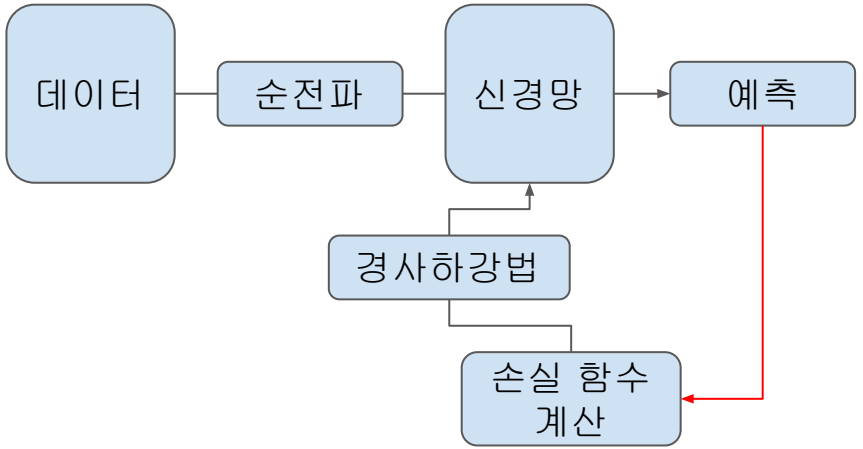
학습률 (Learning rate)

```
def gradient_descent(f, init_x, lr=0.01, step_num=100):
  x = init_x

  for i in range(step_num):
    grad = numerical_gradient(f, x)
    x -= lr * grad

  return x
```

- 경사 하강법은 최적(최소)값을 보장하지는 않지만, 함수의 값을 줄일 수 있는 방법임
- 현 위치에서 기울어진 방향으로 일정거리 이동
- 이동한 곳에서 다시 경사 하강법 반복
- 학습률이 너무 크거나 작으면 효과가 없음

현재까지 학습한 내용

- 퍼셉트론
- 활성화 함수
- 다차원 배열의 계산
- 신경망 구현하기, 출력층 설계하기
- 데이터 추론
- 손실함수
- 미니배치
- 경사 하강법 (수치 미분)



현재까지 학습한 내용 구현하기

- 퍼셉트론
- **활성화 함수**
- 다차원 배열의 계산
- **신경망** 구현하기, **출력층** 설계하기
- 데이터 추론
- **손실함수**
- **미니배치**
- 경사 하강법 (수치 미분)

현재까지 학습한 내용 구현하기

1. 데이터 로딩
2. 하이퍼파라미터 설정하기
3. 신경망 & 출력층 설계 및 구현
4. 미니배치 구성 & 학습
5. 기울기 (gradient) 계산
6. 가중치 업데이트
7. 학습 경과 기록하기 (loss function)

현재까지 학습한 내용 구현하기

1. 데이터 로딩 - MNIST 데이터

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 픽셀 값을 0~1 사이로 정규화합니다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

TensorFlow > API > TensorFlow Core v2.3.0 > Python

tf.keras.datasets.mnist.load_data

공식 문서 링크

Returns

Tuple of Numpy arrays: `(x_train, y_train), (x_test, y_test)`.

x_train, x_test: uint8 arrays of grayscale image data with shapes (num_samples, 28, 28).

y_train, y_test: uint8 arrays of digit labels (integers in range 0-9) with shapes (num_samples,).

현재까지 학습한 내용 구현하기

1. 데이터 로딩 - MNIST 데이터

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 픽셀 값을 0~1 사이로 정규화합니다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
x_test_flatten = x_test.reshape([x_test.shape[0], -1])
```

신경망 입력층으로 전달하기위해 데이터 형상을 28x28에서 784로 맞춤

```
1 print(x_test.shape)
2 print(x_test_flatten.shape)
```

```
(10000, 28, 28)
(10000, 784)
```

공식 문서 링크

TensorFlow > API > TensorFlow Core v2.3.0 > Python

tf.keras.datasets.mnist.load_data

Returns

Tuple of Numpy arrays: `(x_train, y_train), (x_test, y_test)`.

x_train, x_test: uint8 arrays of grayscale image data with shapes (num_samples, 28, 28).

y_train, y_test: uint8 arrays of digit labels (integers in range 0-9) with shapes (num_samples,).

현재까지 학습한 내용 구현하기

2. 하이퍼파라미터 설정하기

하이퍼파라미터란? 모델링할 때 **사용자가 직접 설정해주는 값**을 뜻함

딥러닝 모델링 시에 설정하는 하이퍼파라미터로는 대표적으로 아래와 같은 값들이 있음

- 학습률 (Learning rate)
- 미니배치 크기
- 학습 반복 횟수

이 외에도 모델에 따라 여러가지 하이퍼파라미터가 있을 수 있음

현재까지 학습한 내용 구현하기

2. 하이퍼파라미터 설정하기

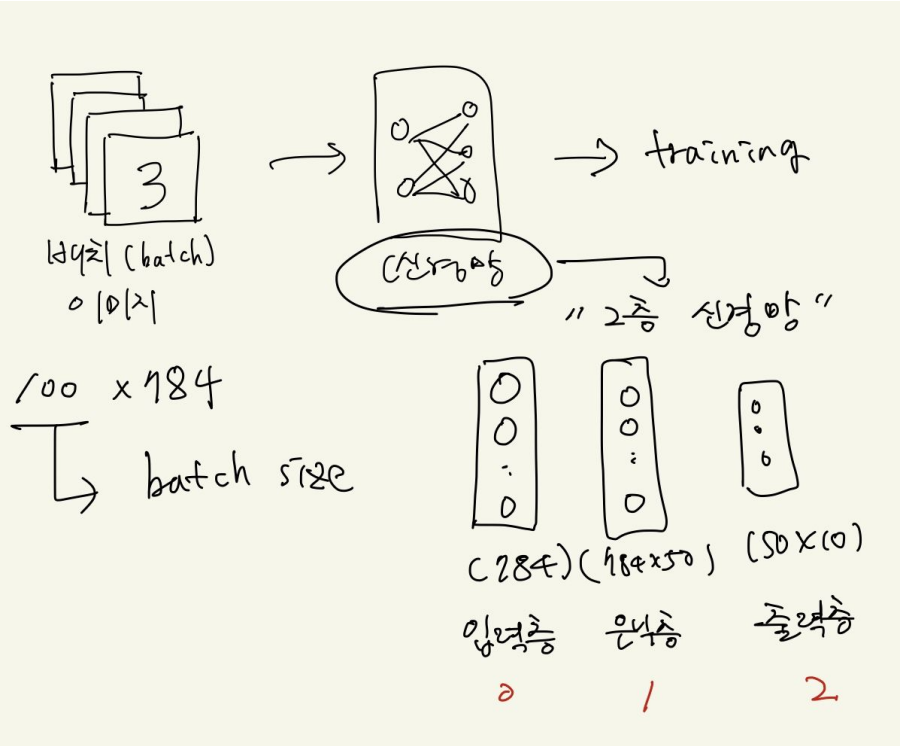
본 예시에서 사용하는 하이퍼파라미터

```
# 하이퍼파라미터 설정
iters_num = 10000 # 반복 횟수
train_size = x_train_flatten.shape[0]
batch_size = 100 # 미니배치 크기
learning_rate = 0.1
```

현재까지 학습한 내용 구현하기

[Colab에서 확인하기]

3. 신경망 & 출력층 설계 및 구현




```python
def predict(self, x):
  W1, W2= self.params['W1'], self.params['W2']
  b1, b2 = self.params['b1'], self.params['b2']

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

  return y
```

(code 일부)

현재까지 학습한 내용 구현하기

4. 미니배치 구성 & 학습

```
# 미니배치 획득
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train_flatten[batch_mask]
t_batch = t_train[batch_mask]
```

```
1 batch_mask = np.random.choice(train_size, batch_size)
```

```
1 batch_mask
```

```
array([52561, 50710, 53084, 18135, 51108, 16164, 52756,  1943, 50638,
         955, 32013, 28355, 21440, 14731,   313, 13573, 15709, 12626,
       26219, 17018, 37639,  8130, 37994, 12281, 58532, 19161, 28458,
       38488, 50212, 26498, 14359,  8035, 56998, 55129, 12071, 10960,
       48525,  4079,   141, 47901, 41360, 59310, 38015,  6440, 26912,
       58357, 55789, 25486,  9456, 24933, 31985, 35151, 54278, 58009,
       10787, 16678, 49943, 35562, 23786, 46182, 37909, 38026, 32125,
       55127, 24962, 50218, 17726, 16019, 37506, 26506, 34762, 58135,
        9124, 33258, 55016, 23428,  9496, 57777,  2554,  5584, 30748,
        3769, 11875, 33923, 47622, 54920, 30565, 23118, 31595, 53873,
       18641, 41440, 18178, 23042,  4248, 33978, 31902, 11985, 39362,
       51771])
```

현재까지 학습한 내용 구현하기

5. 기울기 (gradient) 계산

```python
def numerical_gradient(f, x):
  h = 1e-4
  grad = np.zeros_like(x) # x와 형상이 같은 배열을 생성

  for idx in range(x.shape[0]):
    tmp_val = x[idx]

    # f(x+h) 계산
    x[idx] = tmp_val + h
    fxh1 = f(x)

    # f(x-h) 계산
    x[idx] = tmp_val - h
    fxh2 = f(x)

    grad[idx] = (fxh1 - fxh2) / (2*h)
    x[idx] = tmp_val

  return grad
```

TwoLayerNet class의 함수

```python
# x : 입력 데이터, t : 정답 레이블
def numerical_gradient(self, x, t):
  loss_W = lambda W: self.loss(x, t)

  grads = {}
  grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
  grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
  grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
  grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

  return grads
```

현재까지 학습한 내용 구현하기

6. 가중치 업데이트

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

앞 단계에서 계산한 gradient

7. 학습 경과 기록하기

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

오차역전파법 (backpropagation)

오차역전파법이란?

오차를 역으로 전파하는 방법 (backward propagation of errors)를 줄여서 ‘backpropagation’이라고 함

앞서 학습한 **수치 미분**은 단순하고 구현하기도 쉽지만 **계산 시간이 오래 걸린다는 단점**이 있음

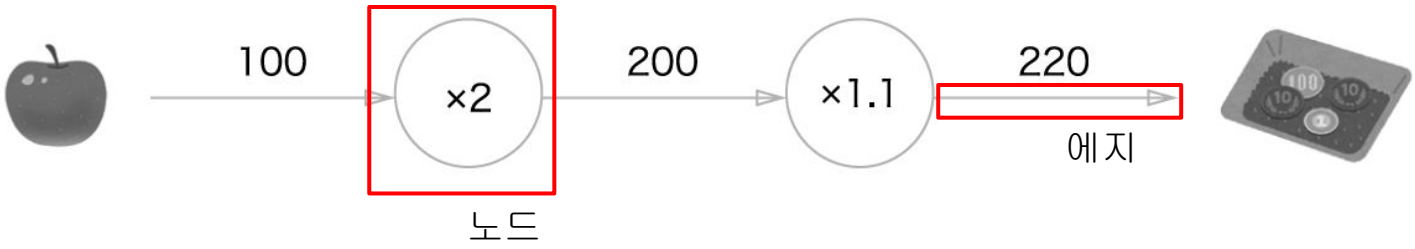
오차역전파법을 이용하면 가중치 매개변수의 기울기를 **효율적으로 계산**할 수 있음

Computational graph (계산 그래프)

계산그래프란?

계산 과정을 그래프로 나타낸 것

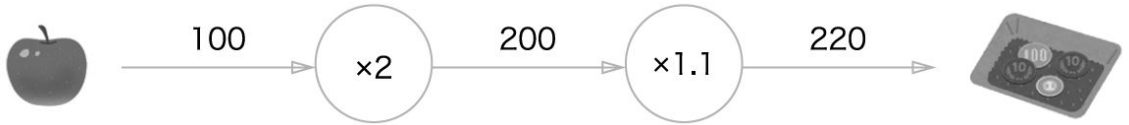
노드 (node)와 에지 (edge)로 표현되는 그래프 자료구조임



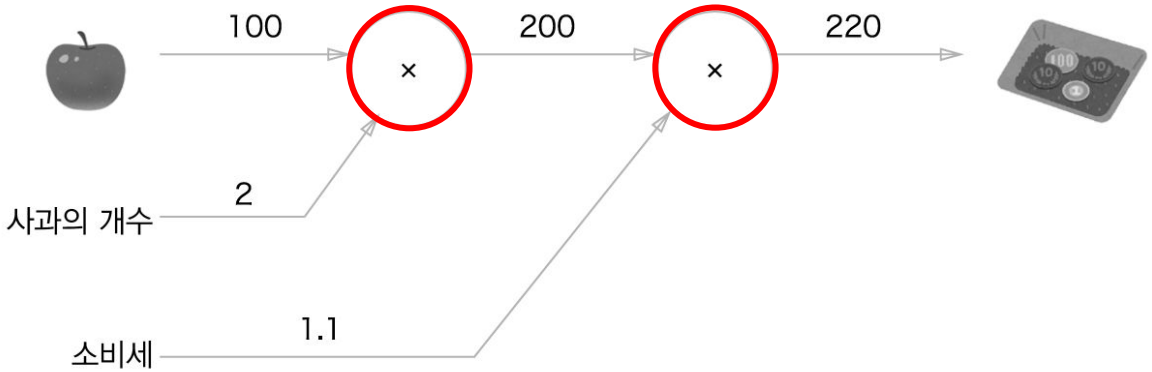
Computational graph (계산 그래프)

문제1 : 현빈 군은 슈퍼에서 1개에 **100원**인 사과를 **2개** 샀습니다.
이때 지불 금액을 구하세요. 단, 소비세가 **10%** 부과됩니다.

계산 그래프

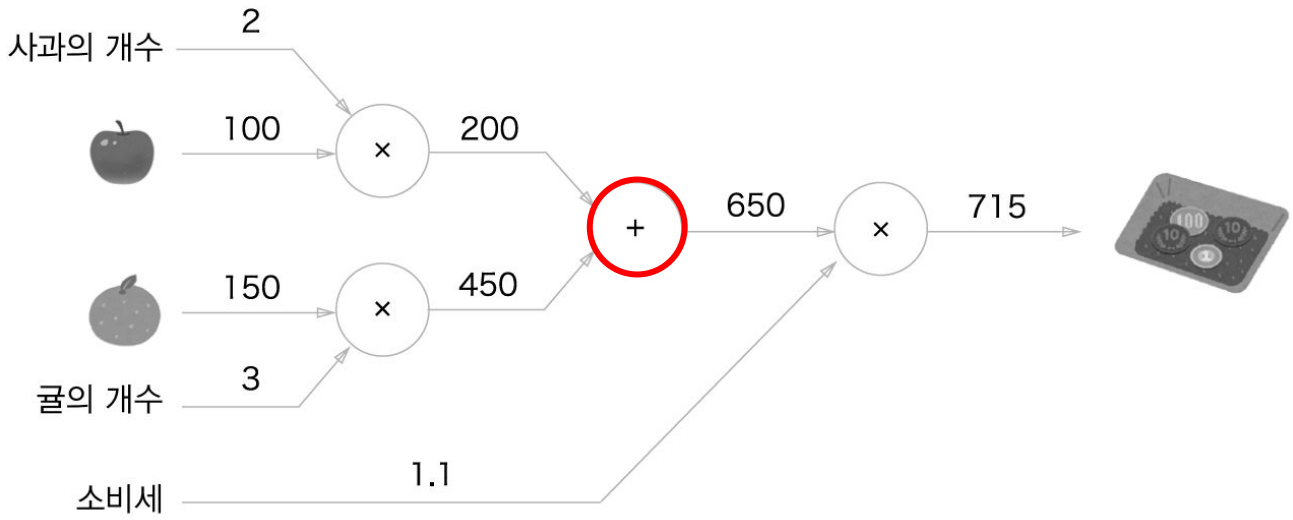


계산 그래프
(변수 취급)



Computational graph (계산 그래프)

문제2 : 현빈 군은 슈퍼에서 **사과**를 **2개**, **귤을 3개** 샀습니다. 사과는 1개에 **100원**, 귤은 1개에 **150원**입니다. 소비세가 **10%**일 때 지불 금액을 구하세요.

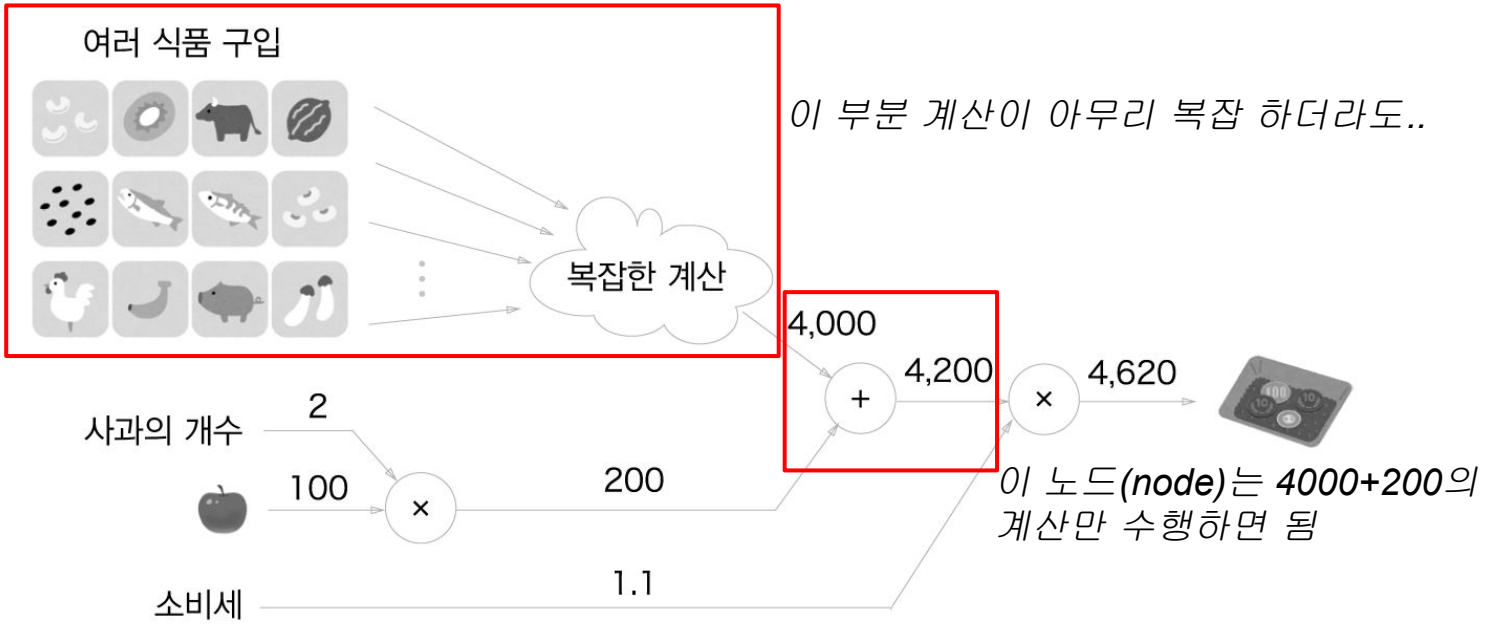


Computational graph (계산 그래프)

계산 그래프는 '국소적 계산'을 전파하여 최종 결과를 얻는 다는 특징이 있음

국소적이란 '자신과 직접 관계된 작은 범위' 라는 뜻

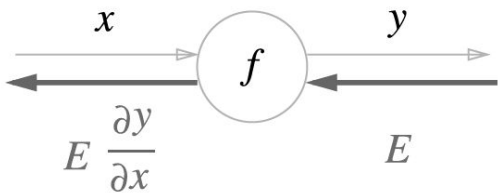
전체 계산 결과와 상관없이, 자신과 관계된 정보만으로 결과를 출력할 수 있다는 것

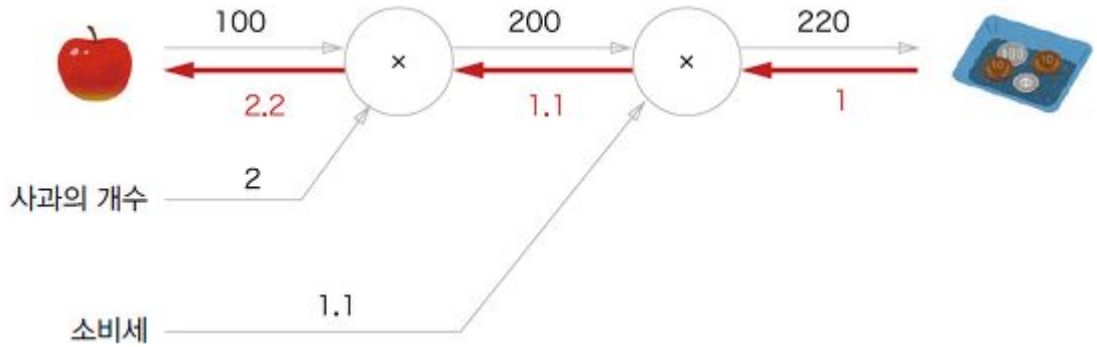


Backpropagation (역전파)

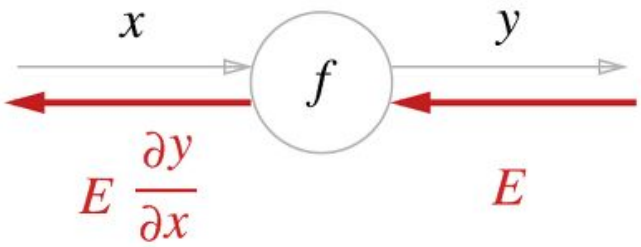
문제1 : 현빈 군은 슈퍼에서 1개에 **100원**인 사과를 **2개** 샀습니다.
이때 지불 금액을 구하세요. 단, 소비세가 **10%** 부과됩니다.

사과 가격이 오르면 최종 금액에 어떤 영향을 끼치는가?
=> '사과 가격에 대한 지불 금액의 미분'을 구하는 문제!

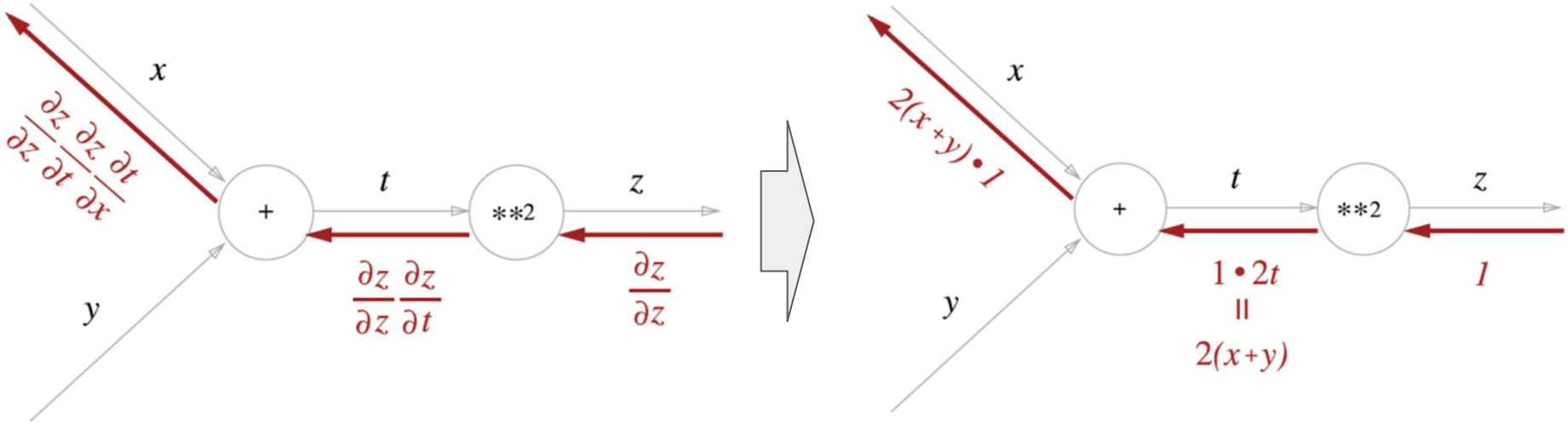




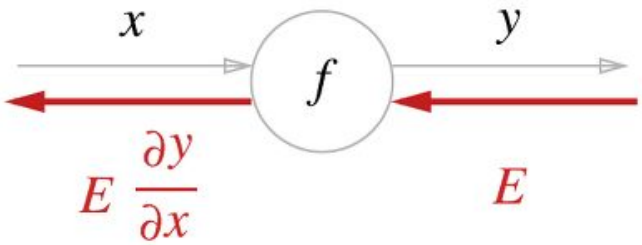
연쇄 법칙



역전파시 계산해야 하는 값 : x에대한 y의 미분
(x가 변할 때 y는 얼만큼 변하는지?)



연쇄 법칙



수식보다 의미를 이해하자

x 가 1만큼 변할 때 y는 어느정도 변하는지 계산한 것

예를 들어 x가 사과의 개수,

y가 전체 가격이라면,

$E\frac{\partial y}{\partial x}$ 사과 개수 1개당 가격의 변동 정도로 해석할 수 있음

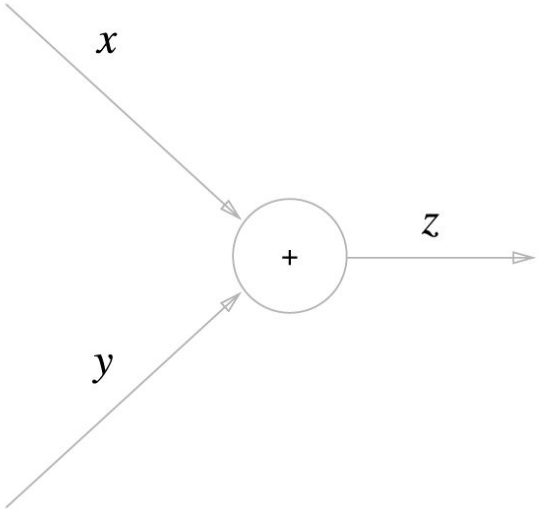
역전파

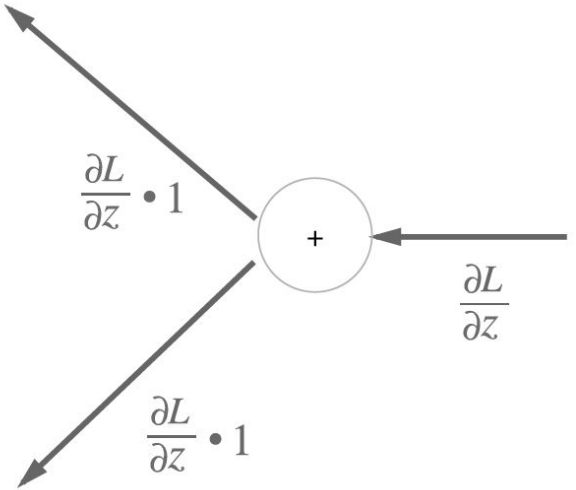
덧셈 노드의 역전파

z = x + y

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

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





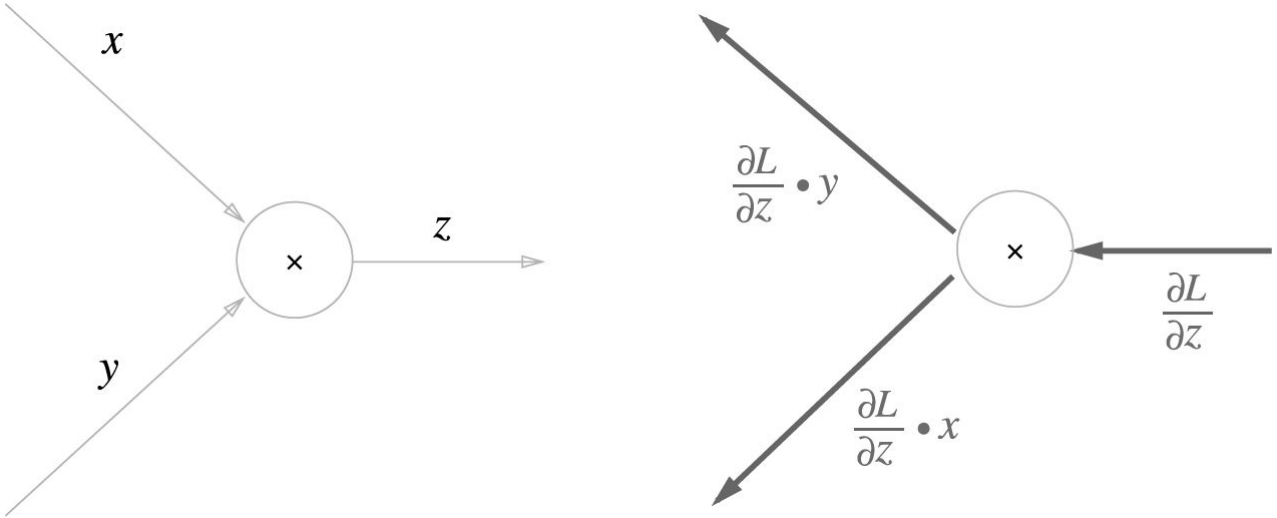
역전파

곱셈 노드의 역전파

z = xy

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

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



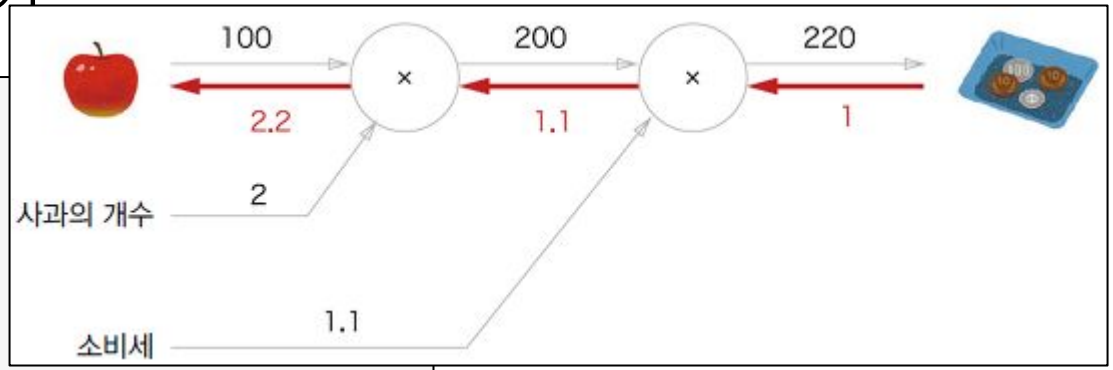
단순한 계층 구현하기

계산 그래프에서 사용 되는 곱셉 노드와 덧셈 노드를 구현해보자

```python
class MulLayer:
  def __init__(self):
    self.x = None
    self.y = None

  def forward(self, x, y):
    self.x = x
    self.y = y
    out = x * y

    return out

  def backward(self, dout):
    dx = dout * self.y
    dy = dout * self.x

    return dx, dy
```

```python
class AddLayer:
  def __init__(self):
    pass

  def forard(self, x, y):
    out = x + y
    return out

  def backward(self, dout):
    dx = dout * 1
    dy = dout * 1
    return dx, dy
```

단순한 계층 구현하기

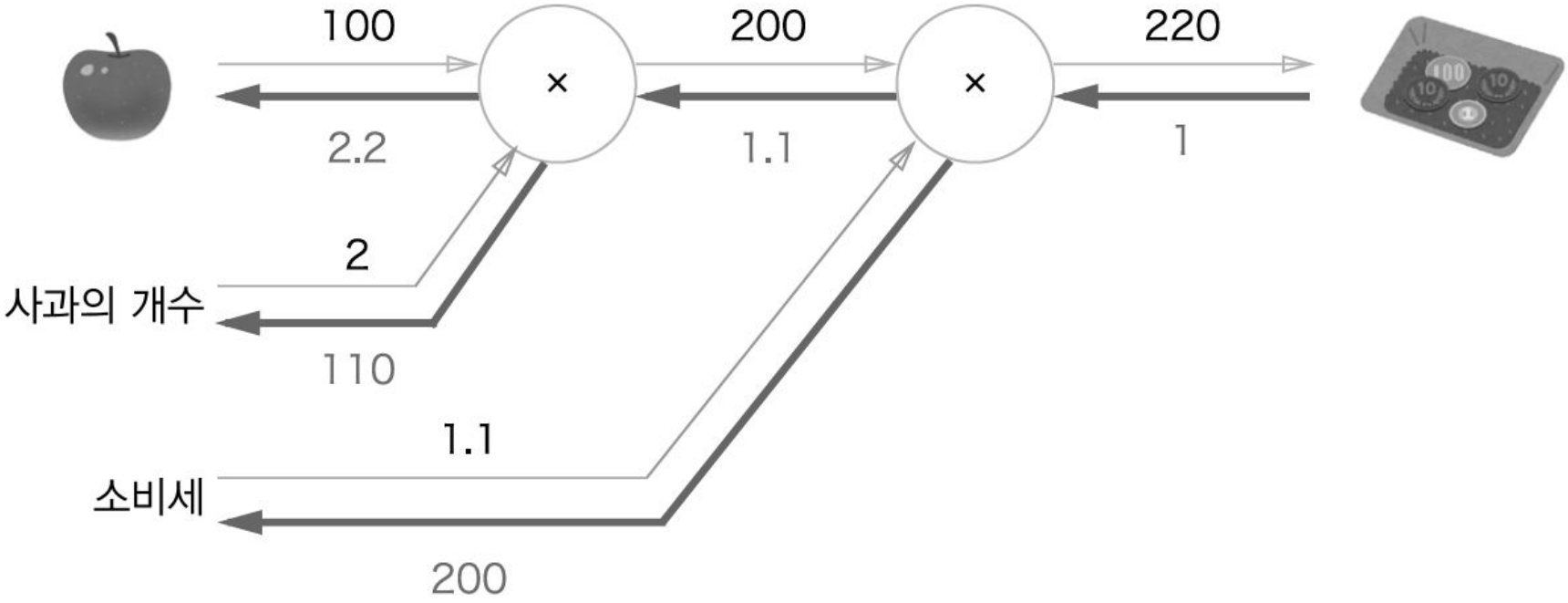



```
apple = 100
apple_num = 2
tax = 1.1

mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

# backward
dprice = 1
dapple_price, dtax = mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dTax:", dtax)
```

```
price: 220
dApple: 2.2
dApple_num: 110
dTax: 200
```

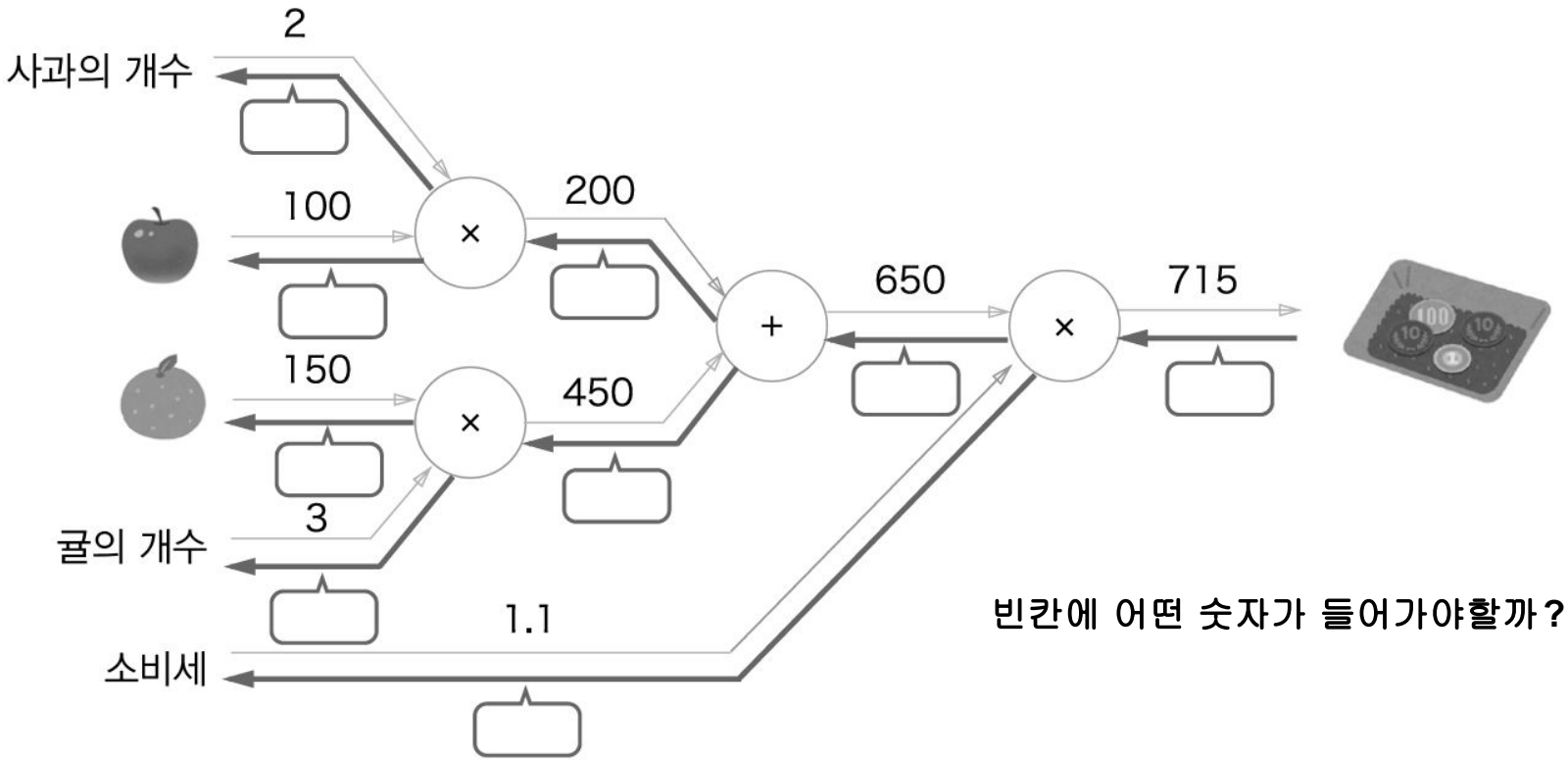
역전파 예시

사과 쇼핑의 역전파 예



역전파 예시

사과와 귤 쇼핑의 역전파 예



빈칸에 어떤 숫자가 들어가야할까?

활성화 함수 계층 구현하기

계산 그래프에 활성화 함수 구현 (ReLU)

ReLU 수식

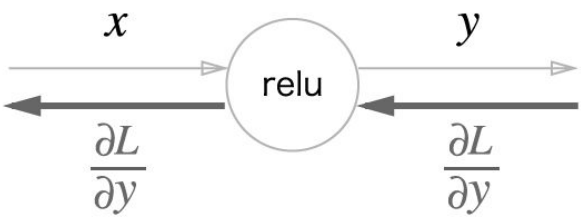
$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

x에 대한 y의 미분

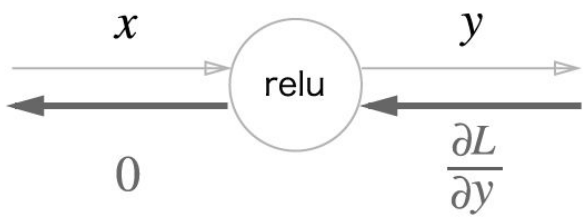
$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

입력인 x가 0보다 크면 역전파는 상류의 값을 그대로 전달
x가 0이하면, 하류로 신호를 보내지 않음

$x > 0$



$x \leq 0$



활성화 함수 계층 구현하기

계산 그래프에 활성화 함수 구현 (ReLU)

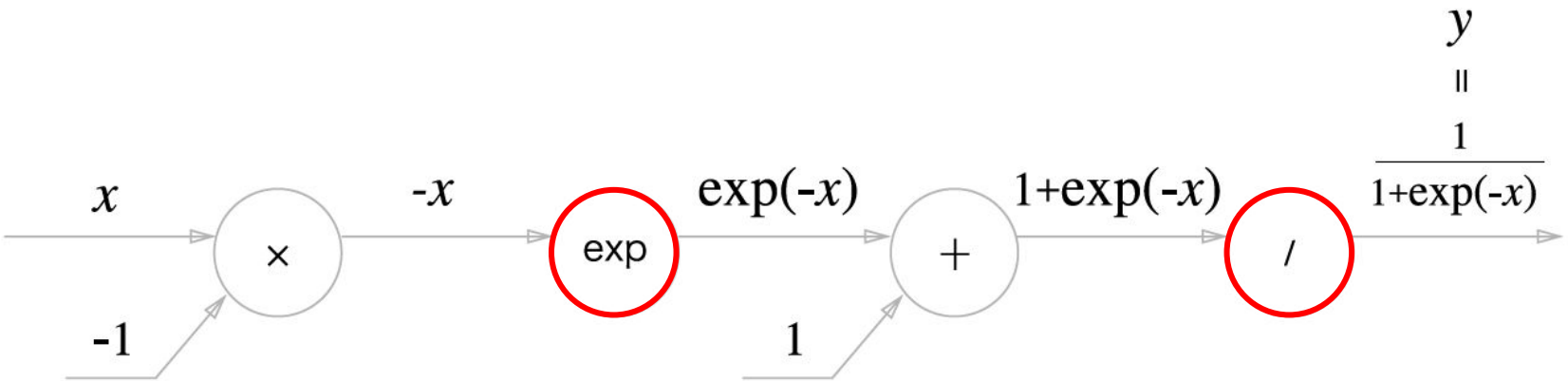
```
class Relu:
  def __init__(self):
    self.mask = None

  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
```

활성화 함수 계층 구현하기

계산 그래프에 활성화 함수 구현 (Sigmoid)

Sigmoid 수식

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



활성화 함수 계층 구현하기

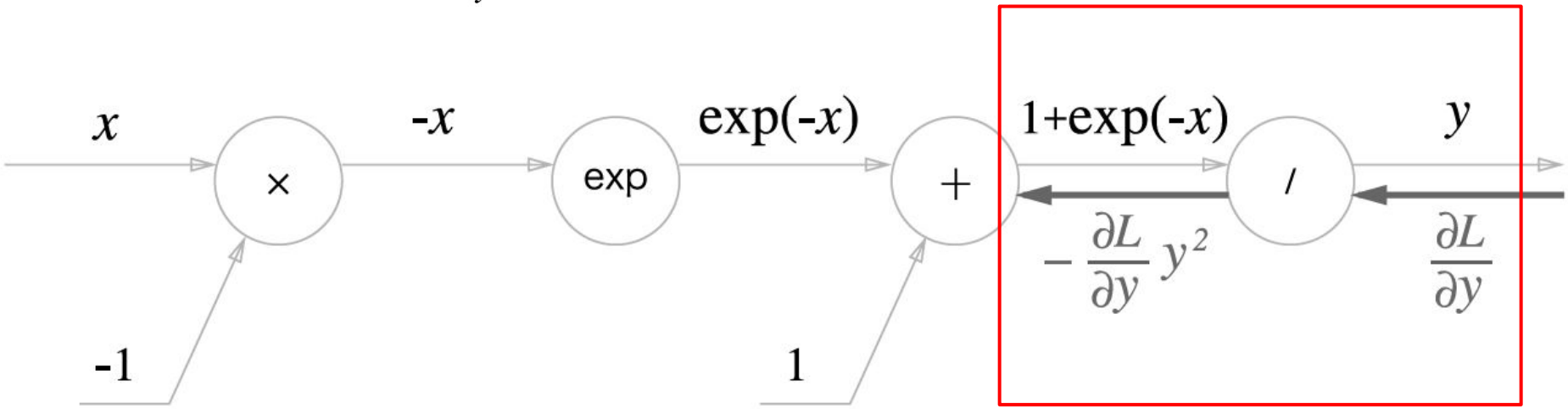
계산 그래프에 활성화 함수 구현 (Sigmoid)

‘/’ 노드

$$\frac{\partial y}{\partial x} = -\frac{1}{x^2}$$

y = 1/x 의 미분

$$= -y^2$$

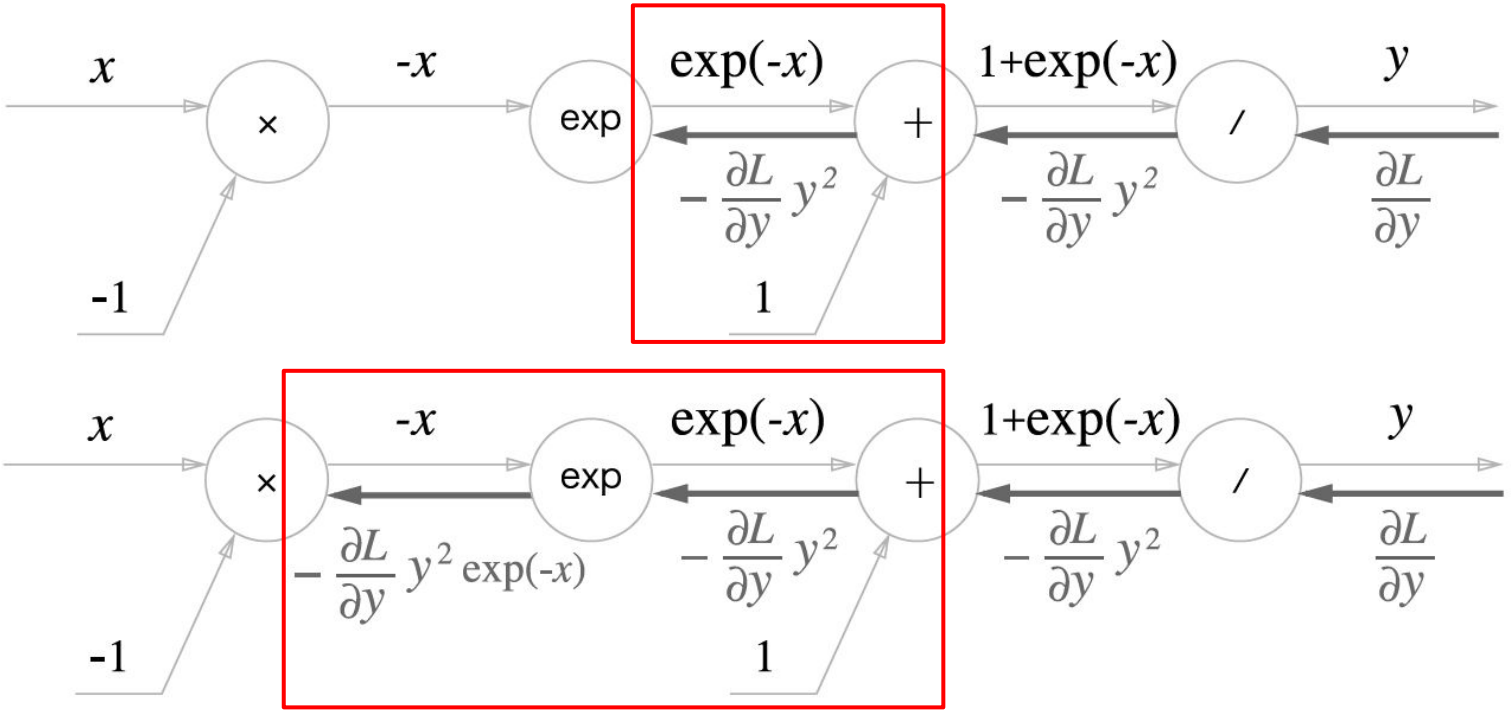


활성화 함수 계층 구현하기

y = exp(x)

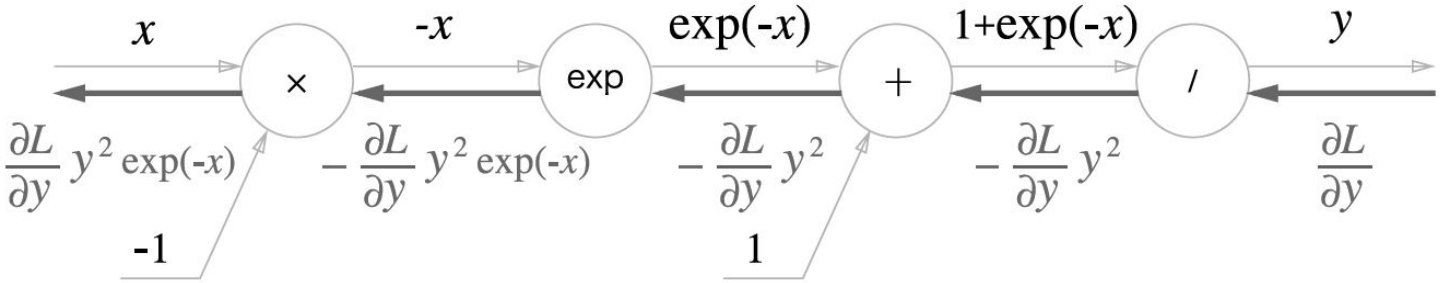
$$\frac{\partial y}{\partial x} = \exp(x)$$

계산 그래프에 활성화 함수 구현 (Sigmoid)

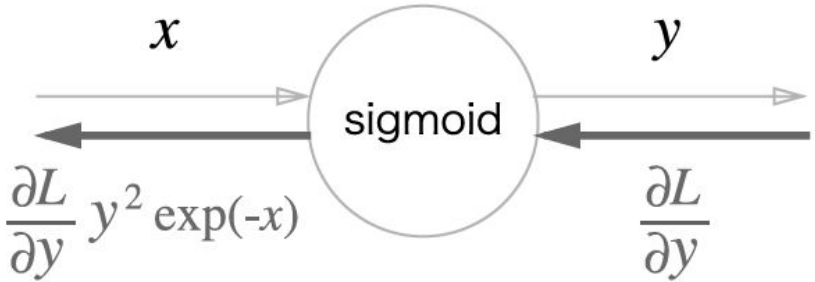


활성화 함수 계층 구현하기

계산 그래프에 활성화 함수 구현 (Sigmoid)



위 계산 과정을 ‘sigmoid’ 노드로 축약!



활성화 함수 계층 구현하기

Sigmoid 수식

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

계산 그래프에 활성화 함수 구현 (Sigmoid)

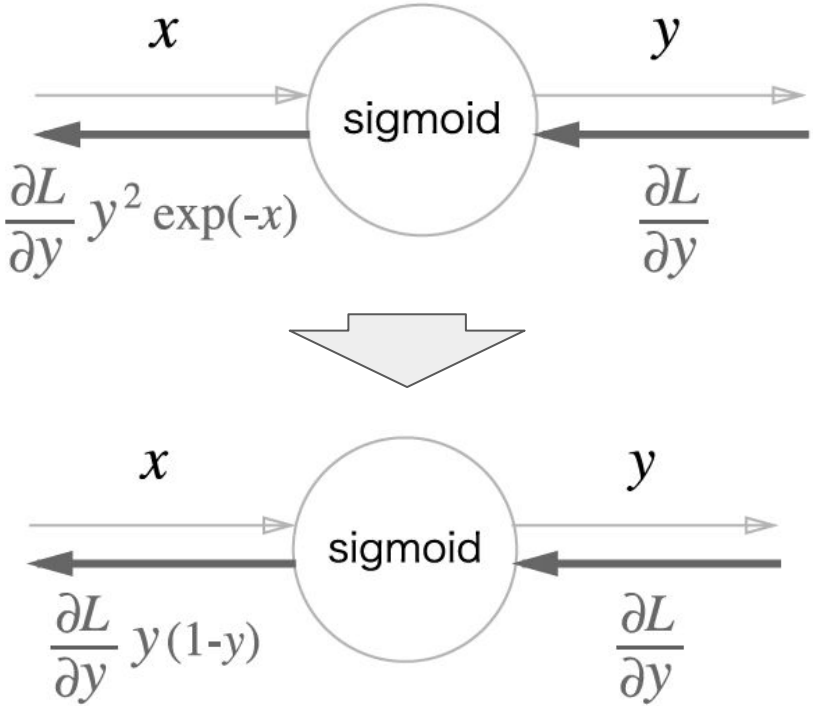
$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1 + \exp(-x))^2} \exp(-x)$$

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

$$= \frac{\partial L}{\partial y} y(1 - y)$$

활성화 함수 계층 구현하기

계산 그래프에 활성화 함수 구현 (Sigmoid)




```
class Sigmoid:
  def __init__(self):
    self.out = None

  def forward(self, x):
    out = 1 / (1 + np.exp(-x))
    self.out = out

    return out

  def backward(self, dout):
    dx = dout * (1.0 - self.out) * self.out

    return dx
```

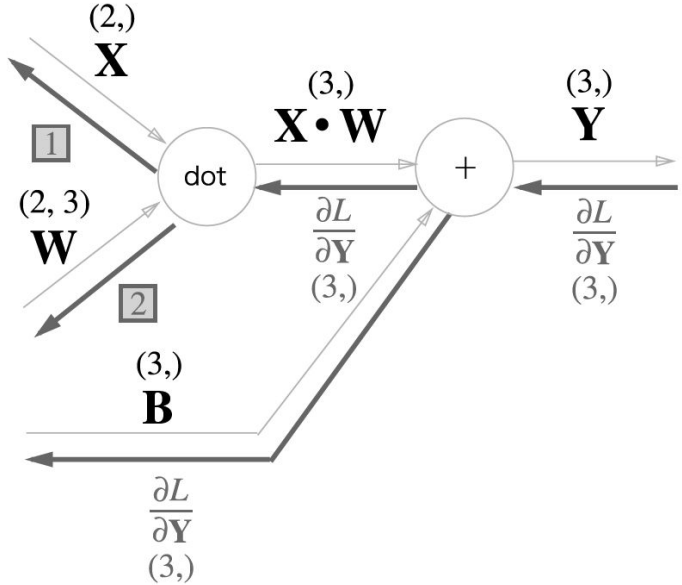
Backpropagation (역전파)

배치용 Affine 계층

신경망의 순전파 때 수행하는 행렬의 내적은 기하학에서 ‘Affine transformation’ (어파인 변환) 이라고 함
어파인 변환을 수행하는 계층을 ‘Affine 계층’ 이라고 함

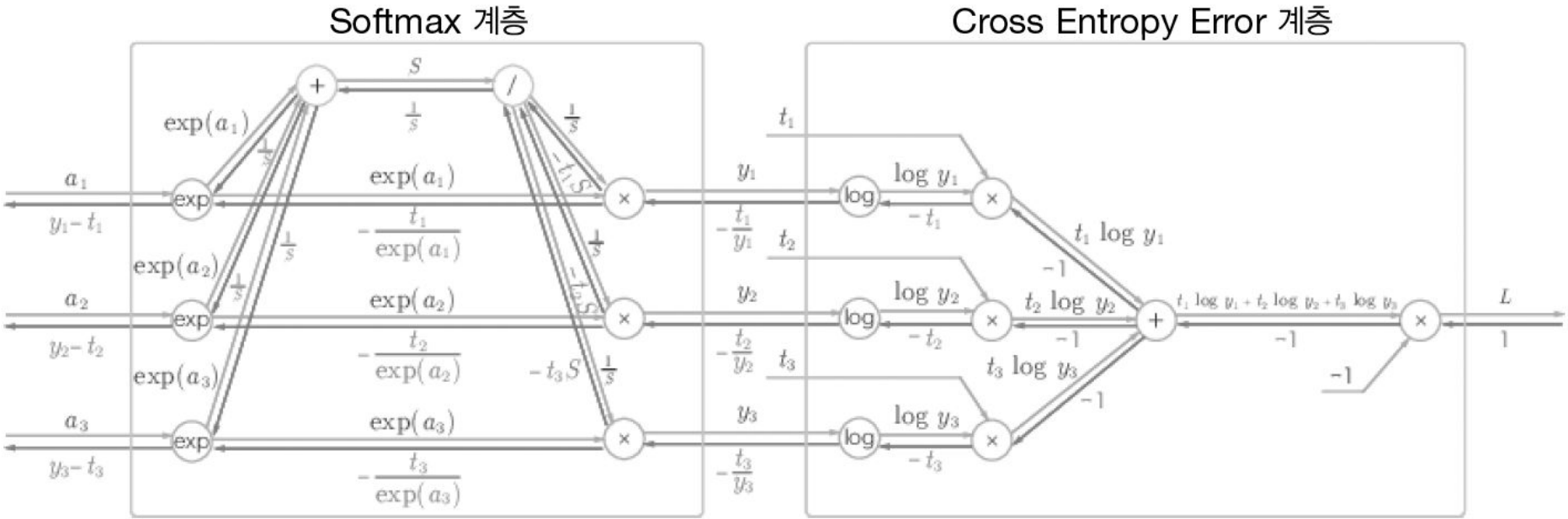
$$\boxed{1}\quad \underset{(2,)}{\frac{\partial L}{\partial \mathbf{X}}} = \underset{(3,)}{\frac{\partial L}{\partial \mathbf{Y}}}\ \underset{(3,\,2)}{\mathbf{W}^{\mathrm{T}}}$$

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



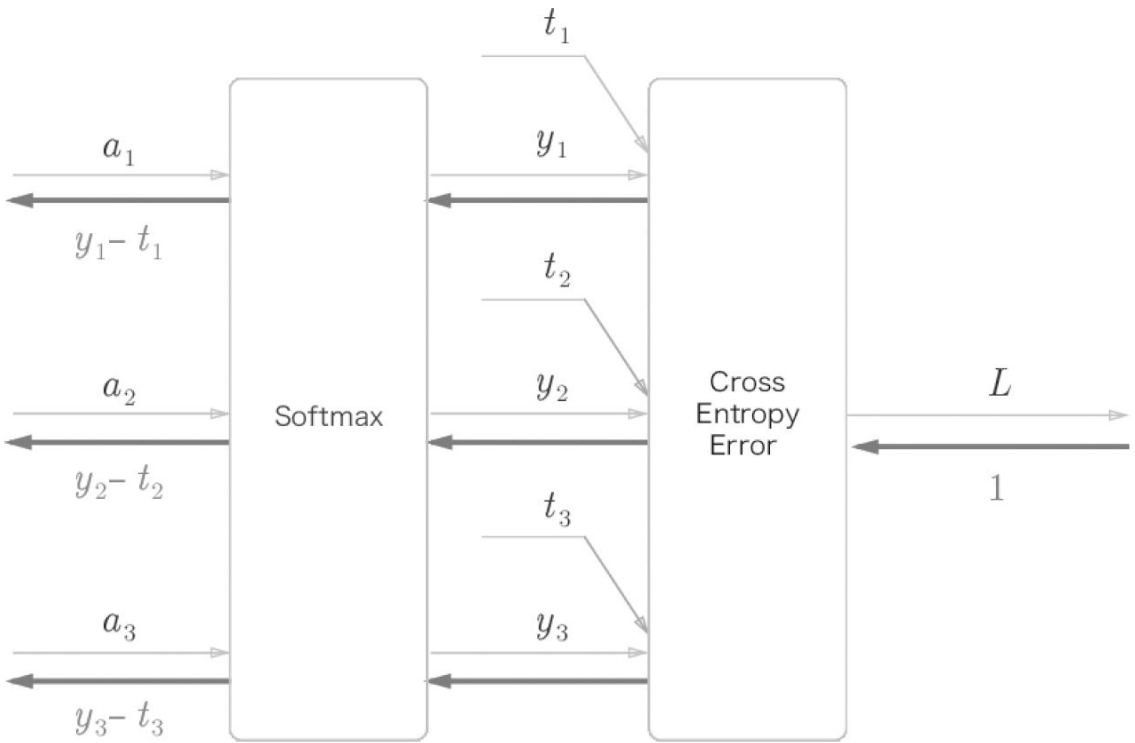
Backpropagation (역전파)

Softmax-with-Loss 계층



Backpropagation (역전파)

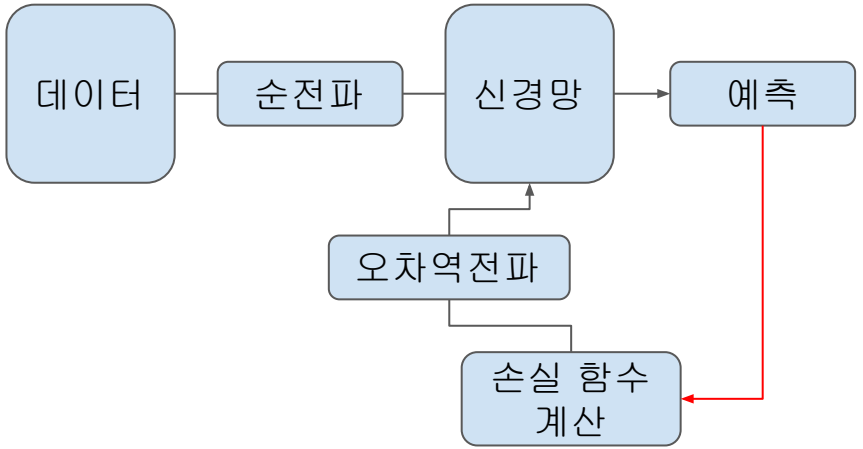
Softmax-with-Loss 계층



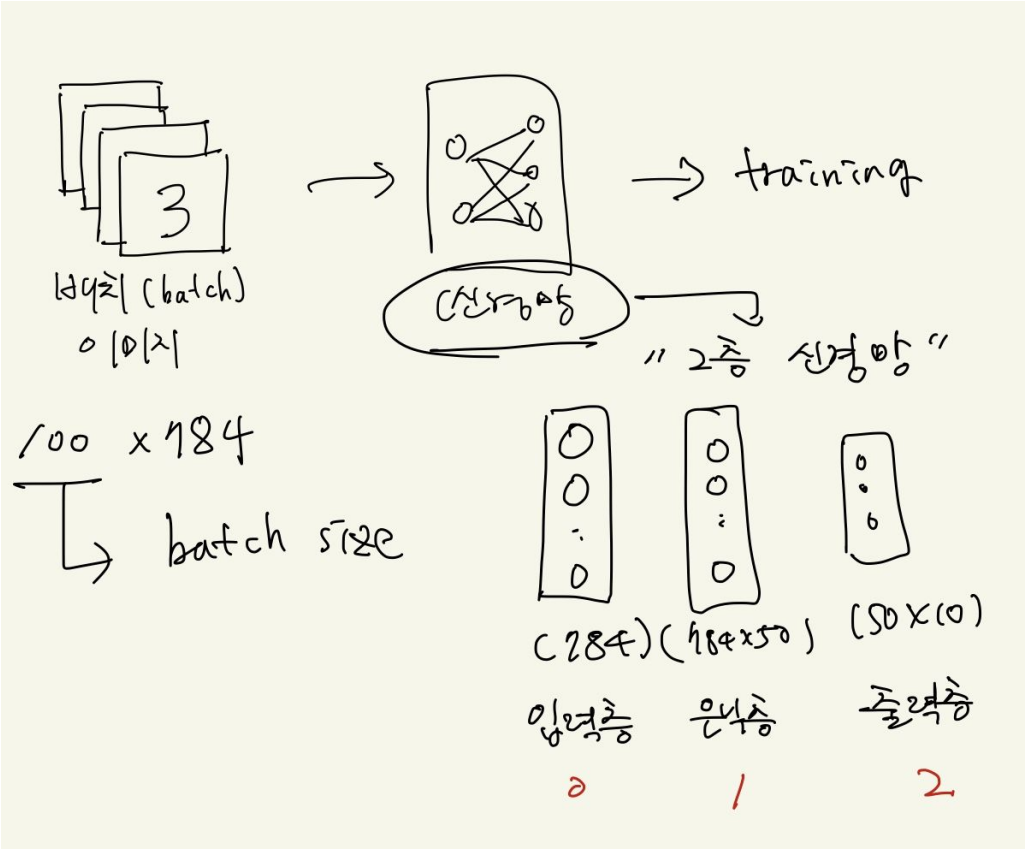
현재까지 학습한 내용

- 퍼셉트론
- 활성화 함수
- 다차원 배열의 계산
- 신경망 구현하기, 출력층 설계하기
- 데이터 추론
- 손실함수
- 미니배치
- ~~경사 하강법 (수치 미분)~~

  => 오차역전파법



오차역전파법 구현하기



앞서 경사하강법에서 구현했던 모델을
오차역전파법으로 변경해보자

오차역전파법 구현하기

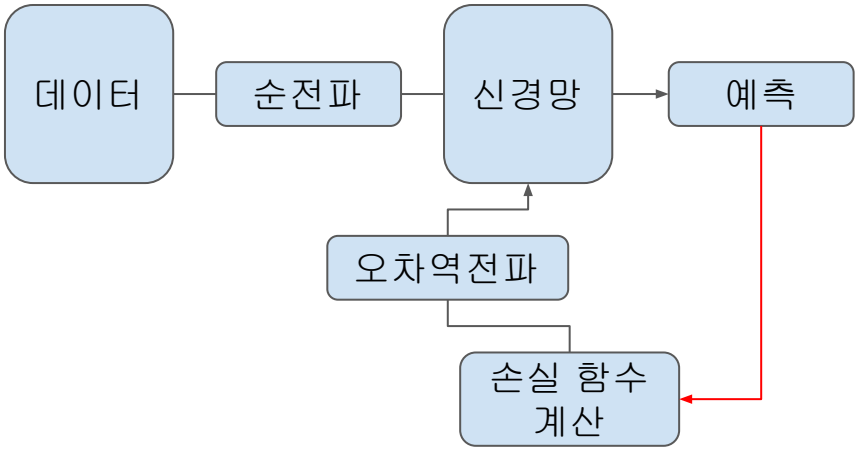
Colab에서 확인하기

지금까지 학습한 내용들

- **계산 그래프**를 이용하면 계산 과정을 시각적으로 파악할 수 있다.
- 계산 그래프의 노드는 **국소적 계산**으로 구성된다. 국소적 계산을 조합해 전체 계산을 구성한다.
- 계산 그래프의 **순전파는 통상의 계산**을 수행한다. 한편, 계산 그래프의 **역전파로는 각 노드의 미분**을 구할 수 있다.
- 신경망의 구성 요소를 **계층으로 구현하여 기울기를 효율적**으로 계산할 수 있다(**오차역전파법**).

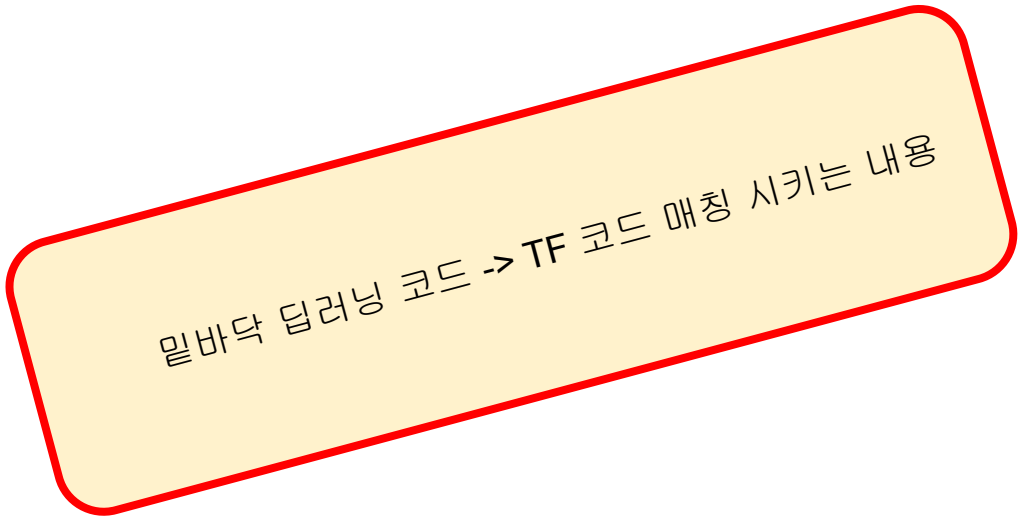
데이터 학습 단계 요약

1. 데이터 읽기 & 전처리
2. 하이퍼파라미터 설정
3. 학습 반복문
   a. 미니배치 획득
   b. 기울기 계산 (오차역전파법)
   c. 가중치 갱신 (learning rate, gradient)
   d. loss 값 계산, 저장



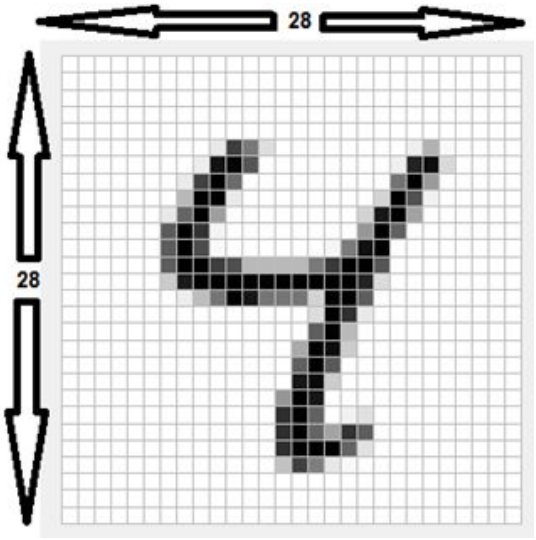
코드로 다시 보는 Neural Network

코드로 다시 보는 Neural Network



Training Neural Network

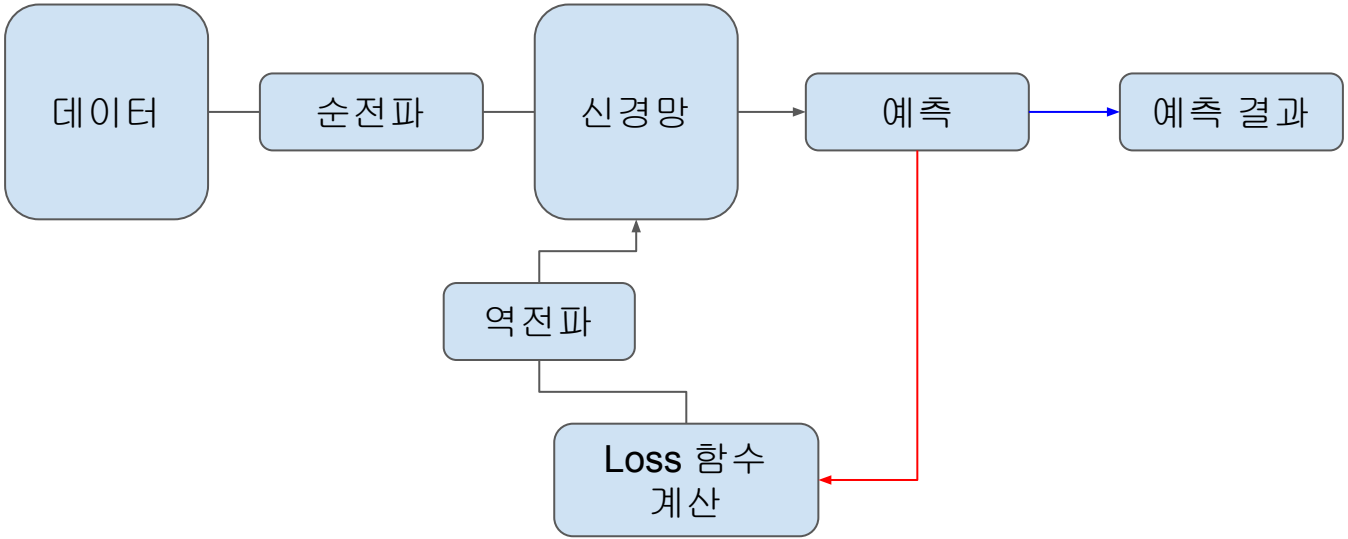
MNIST dataset



Training MNIST data

Training MNIST data

Colab에서 학습해보기



Training MNIST data

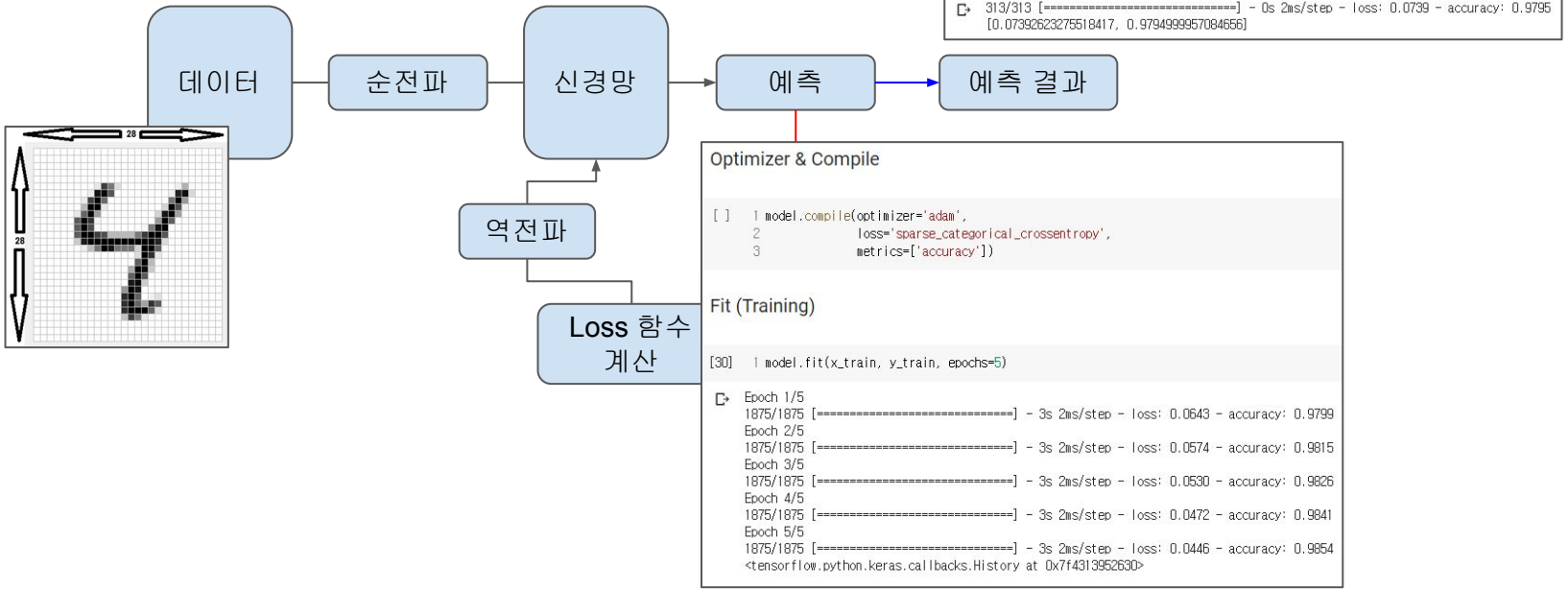
[Colab에서 학습해보기]

Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)

      313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
      [0.07392623275518417, 0.9794999957084656]
```



Optimizer & Compile

```
[ ]  1 model.compile(optimizer='adam',
     2              loss='sparse_categorical_crossentropy',
     3              metrics=['accuracy'])
```

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)

      Epoch 1/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
      Epoch 2/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
      Epoch 3/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
      Epoch 4/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
      Epoch 5/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
      <tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Training MNIST data

Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

```
[32]  1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```

Optimizer & Compile

```
[ ]  1 model.compile(optimizer='adam',
     2               loss='sparse_categorical_crossentropy',
     3               metrics=['accuracy'])
```

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
<tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
[0.07392623275518417, 0.9794999957084656]
```

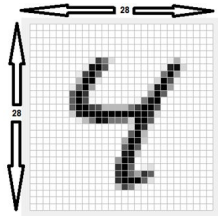
Training MNIST data

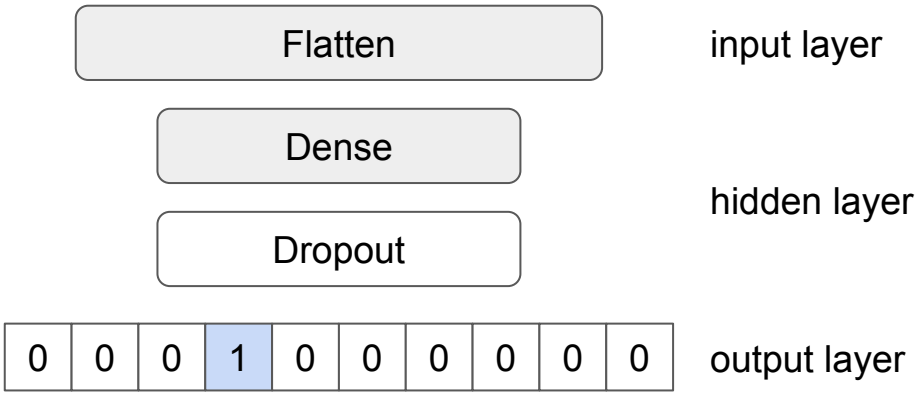
Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

```
[32]  1 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
_________________________________________________________________
```





Training MNIST data

- Dense? Flatten?
- Dropout?
- softmax? relu?
- loss?
- optimizer?
- epochs?

epoch / batch / iterations

In the neural network terminology:

one **epoch** = one forward pass and one backward pass of *all* the training examples

batch size = the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need.

number of **iterations** = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

epoch / batch / iterations

In the neural network terminology:

one **epoch** = one forward pass and one backward pass of *all* the training examples

전체 Training example 단위

batch size = the number of training e gher the batch size, the more memory space you'll need.

한번에 학습 하는 단위

number of **iterations** = number of passes ples. To be clear, one pass = one forward pass + one backward pass (w pass as two different passes).

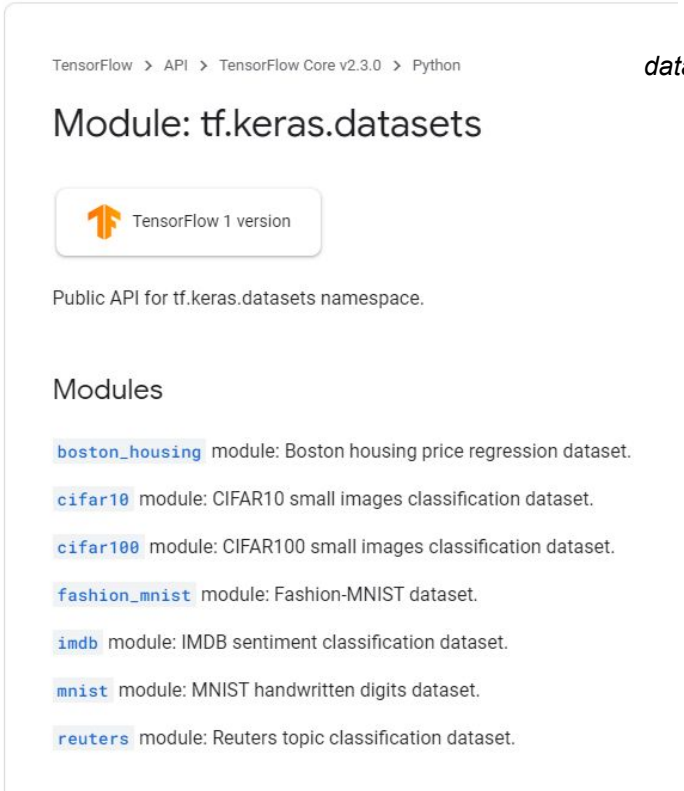
반복하는 회수

Example: if you have *1000 training examples*, and your *batch size is 500*, then it will take 2 iterations to complete 1 epoch.

1000개의 Training Example이 있고, **batch size**가 500이면,
1 **epoch**을 위해 2의 **iteration**이 필요하다

tf.keras.datasets

https://www.tensorflow.org/api_docs/python/tf/keras/datasets

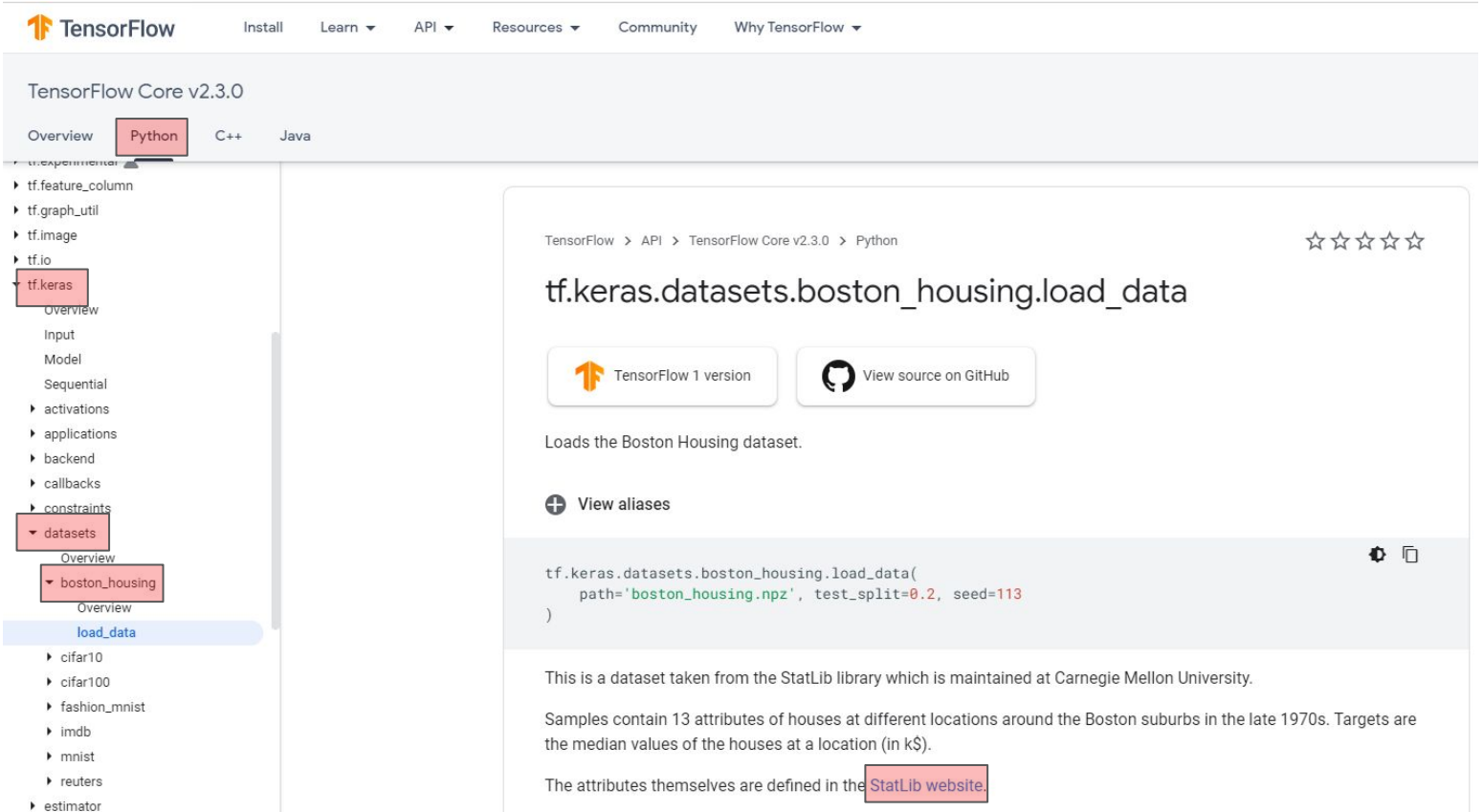


dataframe	`boston_housing` module: Boston housing price **regression** dataset.
image	`cifar10` module: CIFAR10 small **images classification** dataset.
image	`cifar100` module: CIFAR100 small **images classification** dataset.
image	`fashion_mnist` module: Fashion-MNIST dataset.
text	`imdb` module: IMDB **sentiment** classification dataset.
image	`mnist` module: MNIST handwritten **digits** dataset.
text	`reuters` module: Reuters **topic** classification dataset.

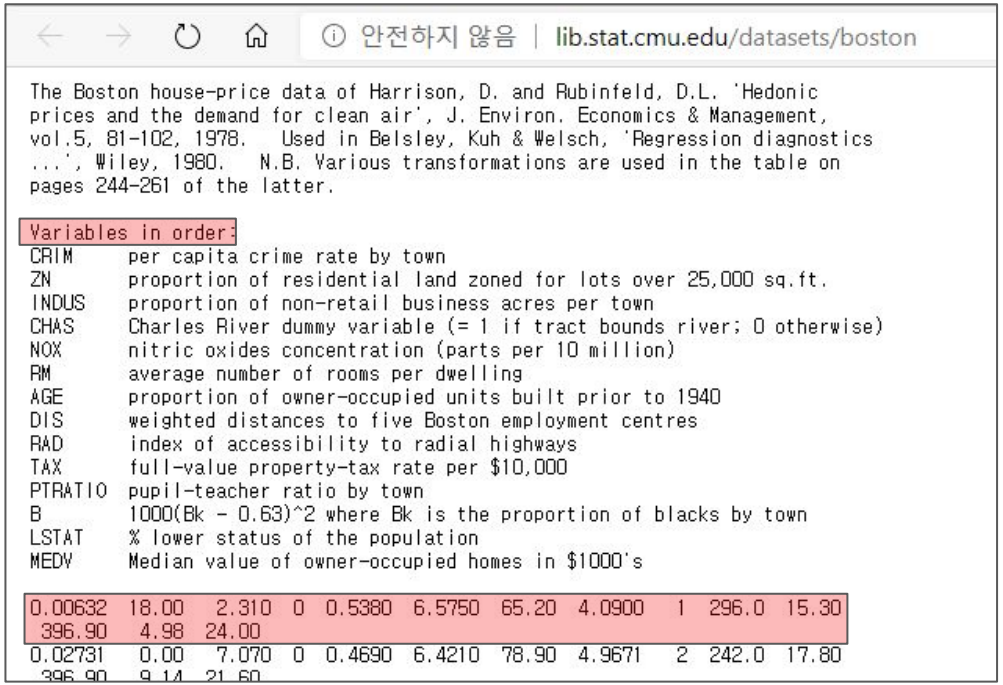
```
1 boston_data = tf.keras.datasets.
2 (x_train, y_train),(x_test, y_test) = boston_data.load_data()
```

module name

dataset 자세히 보기 - boston_housing 예시



dataset 자세히 보기 - boston_housing 예시



안전하지 않음 | lib.stat.cmu.edu/datasets/boston

```
The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

Variables in order:
CRIM     per capita crime rate by town
ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
INDUS    proportion of non-retail business acres per town
CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
NOX      nitric oxides concentration (parts per 10 million)
RM       average number of rooms per dwelling
AGE      proportion of owner-occupied units built prior to 1940
DIS      weighted distances to five Boston employment centres
RAD      index of accessibility to radial highways
TAX      full-value property-tax rate per $10,000
PTRATIO  pupil-teacher ratio by town
B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
LSTAT    % lower status of the population
MEDV     Median value of owner-occupied homes in $1000's

0.00632  18.00   2.310  0  0.5380  6.5750  65.20  4.0900   1  296.0  15.30
 396.90   4.98  24.00
0.02731   0.00   7.070  0  0.4690  6.4210  78.90  4.9671   2  242.0  17.80
 396.90   9.14  21.60
```

```
1 boston_data = tf.keras.datasets.boston_housing
2 (x_train, y_train),(x_test, y_test) = boston_data.load_data()
```

```
1 x_train[0]
```

```
array([  1.23247,   0.     ,   8.14   ,   0.     ,   0.538  ,   6.142  ,
        91.7    ,   3.9769 ,   4.     , 307.     ,  21.     , 396.9    ,
        18.72   ])
```

Training CIFAR data

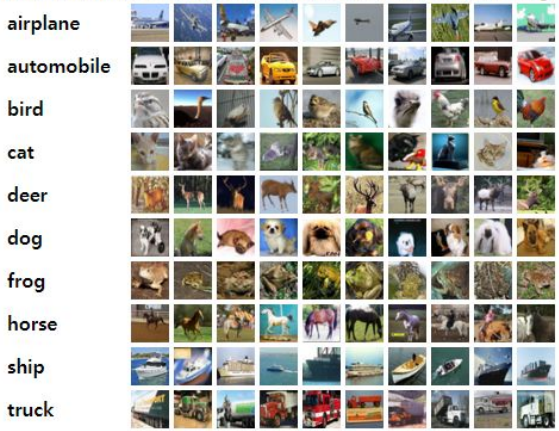
https://www.cs.toronto.edu/~kriz/cifar.html

The CIFAR-10 dataset

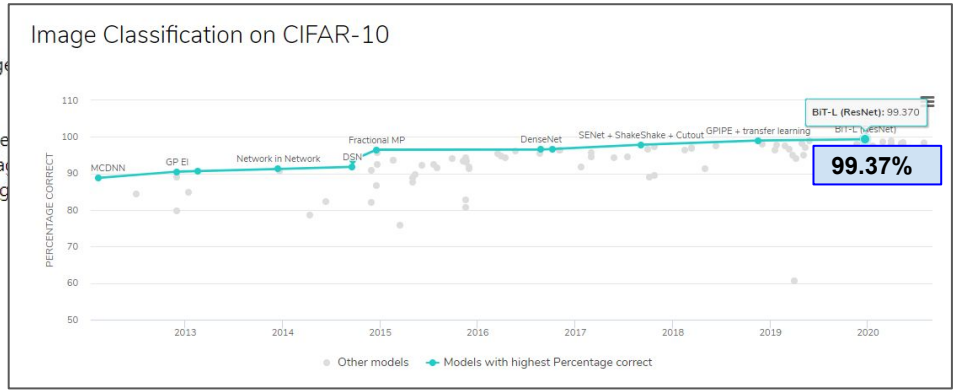
The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 image
images and 10000 test images.

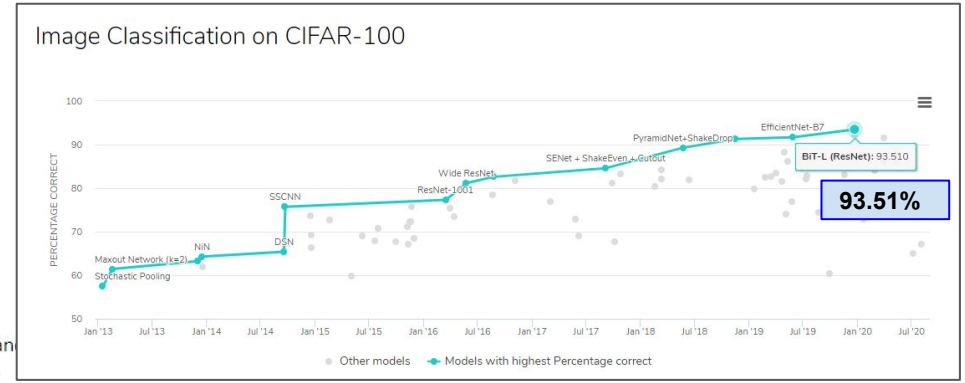
The dataset is divided into five training batches and one test batch, each with 10000 image
randomly-selected images from each class. The training batches contain the remaining ima
batches may contain more images from one class than another. Between them, the training
from each class.

Here are the classes in the dataset, as well as 10 random images from each:



The classes are completely mutually exclusive. There is no overlap between automobiles an
SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.





Training CIFAR-10 data

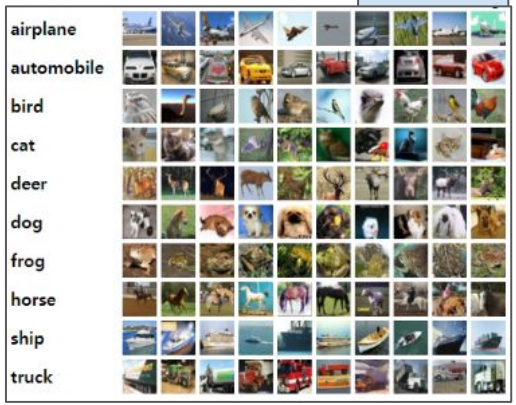
Model

```
[ ]  1 model = tf.keras.models.Sequential([
     2   tf.keras.layers.Flatten(input_shape=(28, 28)),
     3   tf.keras.layers.Dense(128, activation='relu'),
     4   tf.keras.layers.Dropout(0.2),
     5   tf.keras.layers.Dense(10, activation='softmax')
     6 ])
```

Evaluate (Testing)

```
[31]  1 model.evaluate(x_test, y_test)

      313/313 [==============================] - 0s 2ms/step - loss: 0.0739 - accuracy: 0.9795
      [0.07392623275518417, 0.9794999957084656]
```

데이터 → 순전파 → 신경망 → 예측 → 예측 결과



역전파

Loss 함수 계산

Optimizer & Compile

```
[ ]  1 model.compile(optimizer='adam',
     2              loss='sparse_categorical_crossentropy',
     3              metrics=['accuracy'])
```

Fit (Training)

```
[30]  1 model.fit(x_train, y_train, epochs=5)

      Epoch 1/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0643 - accuracy: 0.9799
      Epoch 2/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0574 - accuracy: 0.9815
      Epoch 3/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0530 - accuracy: 0.9826
      Epoch 4/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0472 - accuracy: 0.9841
      Epoch 5/5
      1875/1875 [==============================] - 3s 2ms/step - loss: 0.0446 - accuracy: 0.9854
      <tensorflow.python.keras.callbacks.History at 0x7f4313952630>
```

Training CIFAR-10 data

MNIST 학습에 사용했던 모델을 그대로 사용해본다

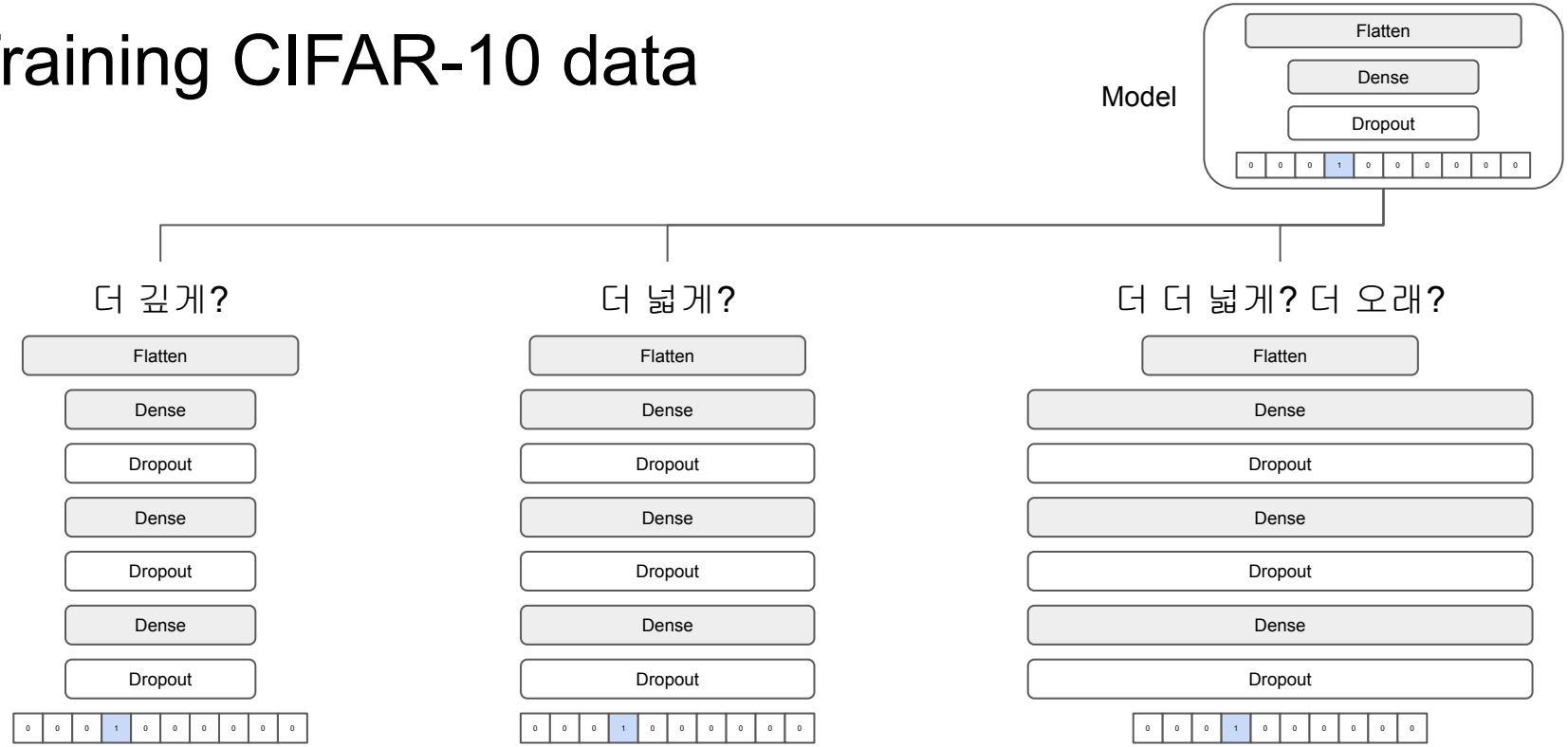
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=x_shape),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_4 (Flatten)          (None, 3072)              0
_________________________________________________________________
dense_13 (Dense)             (None, 128)               393344
_________________________________________________________________
dropout_9 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_14 (Dense)             (None, 10)                1290
=================================================================
Total params: 394,634
Trainable params: 394,634
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
1563/1563 [==============================] - 3s 2ms/step - loss: 2.0212 - accuracy: 0.2469
Epoch 2/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.9198 - accuracy: 0.2915
Epoch 3/5
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8924 - accuracy: 0.3037
Epoch 4/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8731 - accuracy: 0.3095
Epoch 5/5
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8685 - accuracy: 0.3129
313/313 [==============================] - 1s 2ms/step - loss: 1.7859 - accuracy: 0.3615
[1.7859476804733276, 0.36149999499320984]
```

```
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8540 - accuracy: 0.3254
Epoch 16/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8519 - accuracy: 0.3254
Epoch 17/20
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8470 - accuracy: 0.3269
Epoch 18/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8527 - accuracy: 0.3254
Epoch 19/20
1563/1563 [==============================] - 3s 2ms/step - loss: 1.8473 - accuracy: 0.3274
Epoch 20/20
1563/1563 [==============================] - 4s 2ms/step - loss: 1.8416 - accuracy: 0.3273
313/313 [==============================] - 1s 2ms/step - loss: 1.7571 - accuracy: 0.3696
[1.7570570707321167, 0.36959999799728394]
```

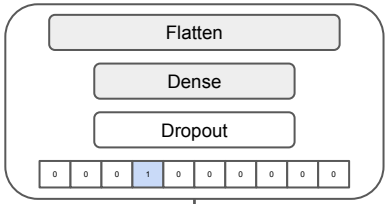
Training CIFAR-10 data



Colab에서 확인하기

Training CIFAR-10 data

Model



더 깊게?

Layer (type)	Output Shape	Param #
flatten_10 (Flatten)	(None, 3072)	0
dense_34 (Dense)	(None, 128)	393344
dropout_21 (Dropout)	(None, 128)	0
dense_35 (Dense)	(None, 128)	16512
dropout_22 (Dropout)	(None, 128)	0
dense_36 (Dense)	(None, 128)	16512
dropout_23 (Dropout)	(None, 128)	0
dense_37 (Dense)	(None, 10)	1290

Total params: 427,658
Trainable params: 427,658
Non-trainable params: 0

427,658

accuracy: 0.**3549**

더 넓게?

Layer (type)	Output Shape	Param #
flatten_11 (Flatten)	(None, 3072)	0
dense_38 (Dense)	(None, 256)	786688
dropout_24 (Dropout)	(None, 256)	0
dense_39 (Dense)	(None, 256)	65792
dropout_25 (Dropout)	(None, 256)	0
dense_40 (Dense)	(None, 256)	65792
dropout_26 (Dropout)	(None, 256)	0
dense_41 (Dense)	(None, 256)	65792
dropout_27 (Dropout)	(None, 256)	0
dense_42 (Dense)	(None, 10)	2570

Total params: 986,634
Trainable params: 986,634
Non-trainable params: 0

986,634

accuracy: 0.**3814**

더 더 넓게? & 더 오래?

Layer (type)	Output Shape	Param #
flatten_12 (Flatten)	(None, 3072)	0
dense_43 (Dense)	(None, 1024)	3146752
dropout_28 (Dropout)	(None, 1024)	0
dense_44 (Dense)	(None, 1024)	1049600
dropout_29 (Dropout)	(None, 1024)	0
dense_45 (Dense)	(None, 1024)	1049600
dropout_30 (Dropout)	(None, 1024)	0
dense_46 (Dense)	(None, 1024)	1049600
dropout_31 (Dropout)	(None, 1024)	0
dense_47 (Dense)	(None, 10)	10250

Total params: 6,305,802
Trainable params: 6,305,802
Non-trainable params: 0

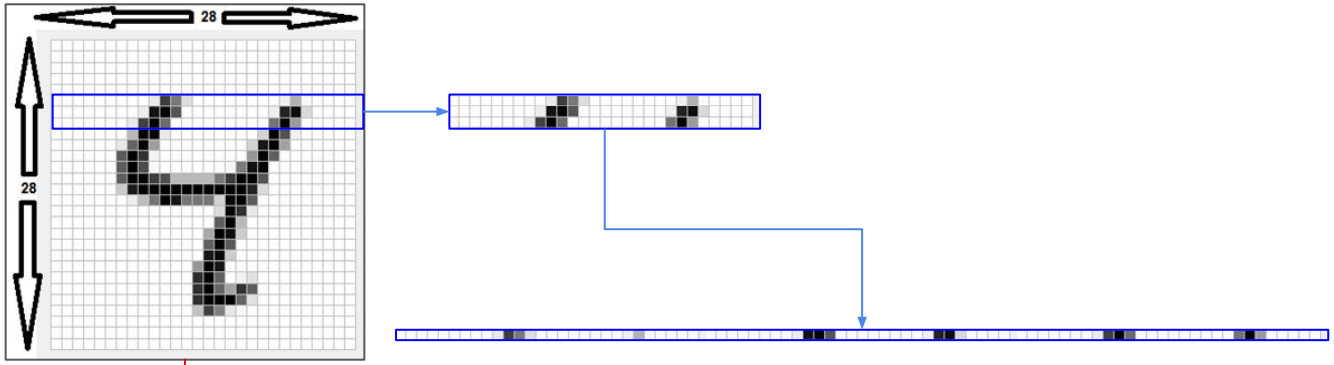
6,305,802

accuracy: 0.**4127** / accuracy: 0.**4251**

20 epoch 100 epoch

성능이 개선되지 않는 이유?

Fully connected layer (Dense layer)의 한계



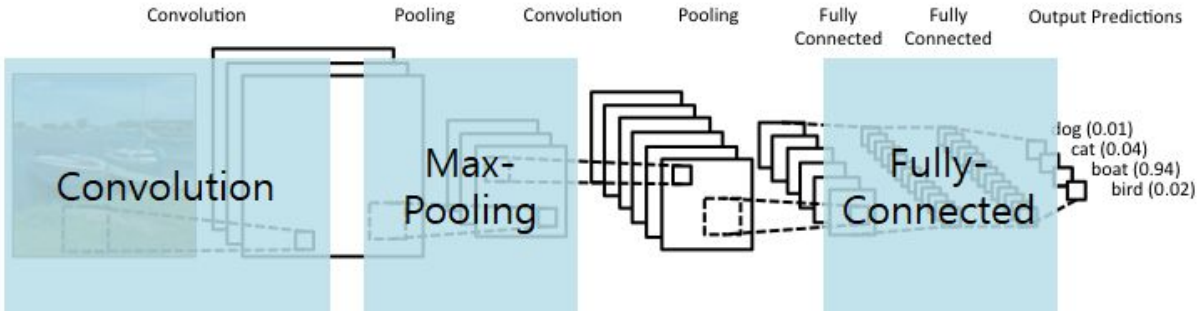

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Flatten data

데이터의 형상이 무시되는 문제점이 있음

Convolutional Neural Network (합성곱 신경망)

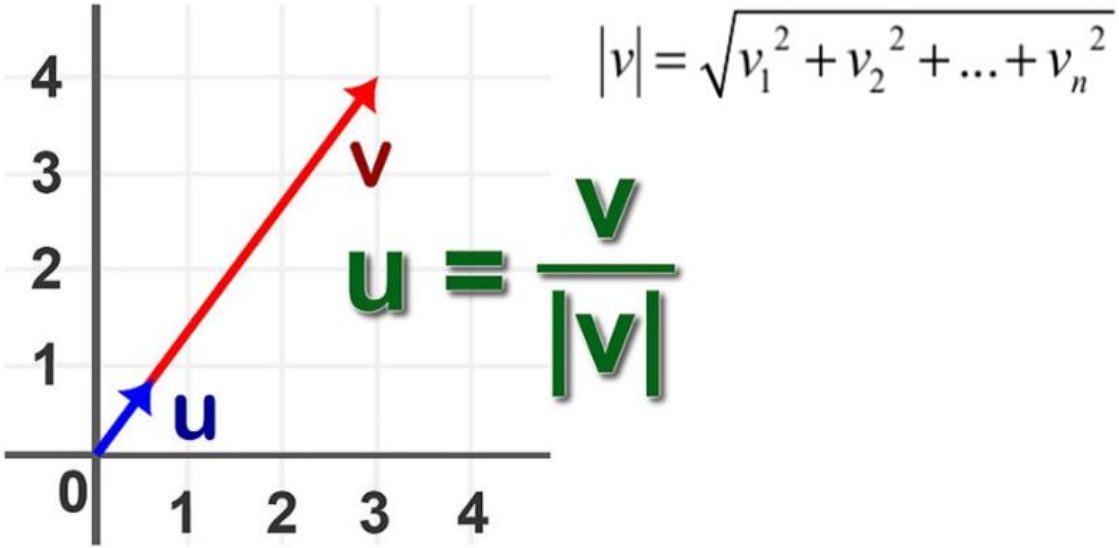
- **Layer의 구성**
  - Convolution Layer – Feature 추출
  - Max-Pooling Layer – Local Search
  - Fully-Connected Layer – 최종 분류



Convolution Layer

- 정규화 (Normalization)

- 벡터를 자신의 길이로 나누어 주면 길이가 1이고 방향만 남은 단위벡터가 됨
- 벡터의 길이는 벡터의 norm 이라고도 함

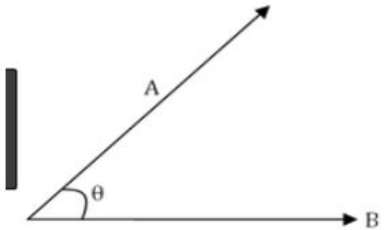


$$|v| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

Convolution Layer

- 벡터의 내적 (Inner Product)

- 내적의 기하학적 의미는 A를 B로 사영시킨 선분의 길이와 B의 길이의 곱



$$\vec{A} \cdot \vec{B} = \overrightarrow{\|A\|}\,\overrightarrow{\|B\|}\cos\theta$$

$$\cos\theta = \frac{\vec{A} \cdot \vec{B}}{\overrightarrow{\|A\|}\,\overrightarrow{\|B\|}}$$

$$\theta = \cos^{-1}\left(\frac{\vec{A} \cdot \vec{B}}{\overrightarrow{\|A\|}\,\overrightarrow{\|B\|}}\right)$$

- 식으로 나타내면 다음과 같다

$$= x_1 y_1 + x_2 y_2 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i$$

$$= \mathbf{x}^T \mathbf{y} = \mathbf{y}^T \mathbf{x}$$

Convolution Layer

- **정규화 + 내적 = cos(θ)**

- 앞선 내적 공식에서 cos(θ)값은 다음과 같이 구할 수 있다

$$\cos\theta = \frac{\overline{A} \cdot \overline{B}}{\overline{\|A\|}\,\overline{\|B\|}}$$

- 정규화를 시키면 분모의 A,B벡터의 길이가 1이므로 A,B의 내적이 cos(θ) 값이 됨

- 두 벡터의 cos(θ)값은 두 벡터의 유사성을 비교하는 척도로 사용 가능 (높을수록 유사, 낮을수록 이질)

Convolution Layer

- **Concatenate**
- 2개의 열벡터 A,B가 있다면

A	B
1	2
2	4
3	6

- concat(A,B) by x 축

A	B
1	2
2	4
3	6

- concat(A,B) by y축

A,B
1
2
3
2
4
6

- concat(A,B) by z축

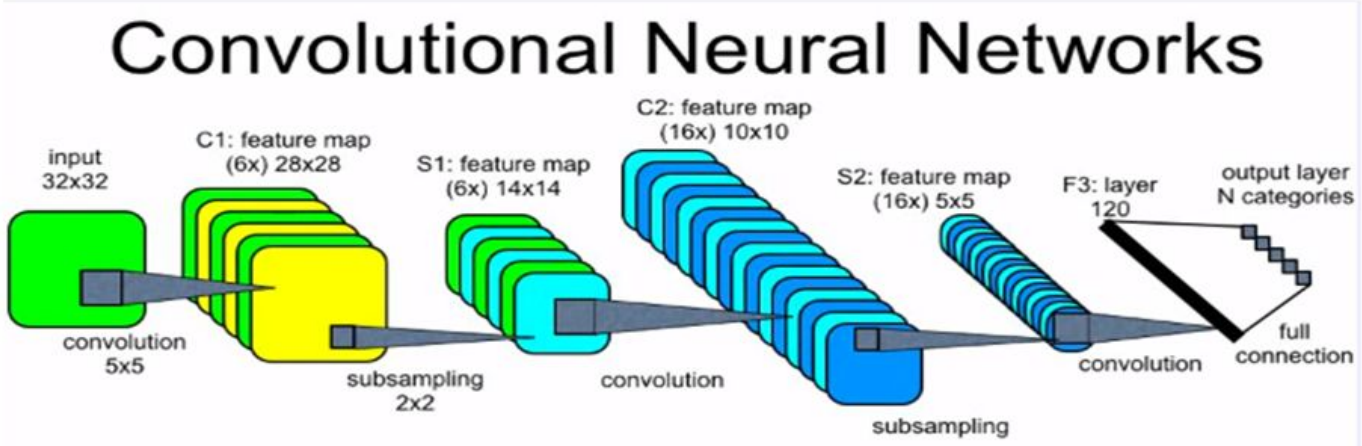
A
1
2
3

벡터가 아닌 행렬 또는 n차원 tensor에도 동일하게 적용

Convolutional Neural Network (합성곱 신경망)

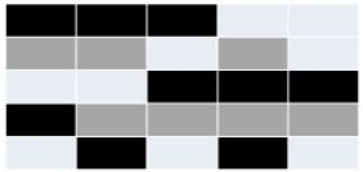
음성인식, 이미지 인식과 같은 2차원 데이터를 처리하기 좋은 딥 러닝 알고리즘

1. Feature 추출 (Convolution Stage, Cn)
2. Subsampling, Sn
3. 위 두 stage를 1차원 벡터 (그림에서는 F3)가 나올 때까지 반복
4. Classification



Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

필터A

=

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

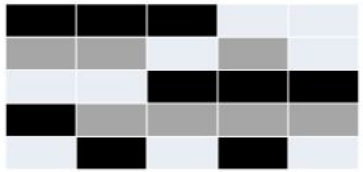
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

필터A

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

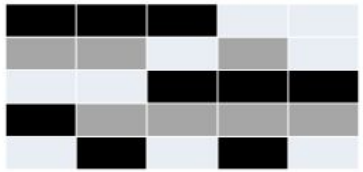
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

=

필터A

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

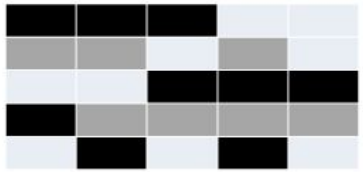
정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

흰색을 0, 회색을 0.5, 검정을 1이라고 했을 때 아래 이미지에서 오른쪽과 같은 특징을 찾고 싶다고 한다면



원래 이미지

=

1	1	1
0.5	0.5	0.5
0	0	0

필터A

=

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

필터A

중앙 파란색은 원래 이미지,
주변의 옅은 색은 차원을 유지하고 경계를 구분하는 역할 (선택)

정규화 + Convolution!! (합성곱)

$$\begin{matrix}1 & 2\\ 3 & 4\end{matrix} * \begin{matrix}5 & 6\\ 7 & 8\end{matrix} = 1x5+2x6+3x7+4x8 = 70$$

Convolution은 지정된 범위에서의 벡터 내적이며 정규화가 되어 있으므로 코사인유사도와 의미가 정확히 같음

Convolution Layer

- Filter의 연산을 직접 해 봅시다!

0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

*

1	1	1
0.5	0.5	0.5
0	0	0

필터A

- **1. 정규화를 거쳐**

0.471	0.471	0.471
0.236	0.236	0
0	0	0.471

*

0.516	0.516	0.516
0.258	0.258	0.258
0	0	0

=

2. Convolution을 하면

0.471*0.516+0.471*0.516+
...+0.471*0=**0.852**

- 수학적 의미를 엄밀하게 따지지 않는다면 해당 영역이 필터와 85.2% 유사하다고 생각하여도 무방

Convolution Layer

- 이 과정을 모든 영역에 적용하면?

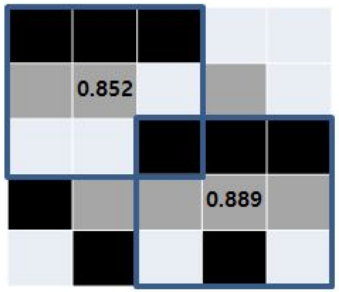
0	0	0	0	0	0	0
0	1	1	1	0	0.5	0
0	0.5	0.5	0	0.5	0	0
0	0	0	1	1	1	0
0	1	0.5	0.5	0.5	0.5	0
0	0	1	0	1	0	0
0	0	0	0	0	0	0

*

1	1	1
0.5	0.5	0.5
0	0	0

=

0.327	0.414	0.327	0.316	0.183
0.816	0.852	0.609	0.426	0.245
0.390	0.447	0.573	0.516	0.467
0.258	0.552	0.652	0.889	0.690
0.689	0.816	0.778	0.781	0.632



원래 이미지

필터A

원래 이미지에서 필터A와 가장 유사한 부분은 파란 테두리 부분!

이런 필터를 여러개 사용하여 Concatenate

Convolution Layer

- Convolution filter의 예시

- Sharpen

0	0	0	0	0
0	0	-1	0	0
0	-1	5	-1	0
0	0	-1	0	0
0	0	0	0	0

- Edge Detect

	0	1	0	
	1	-4	1	
	0	1	0	

- Blur

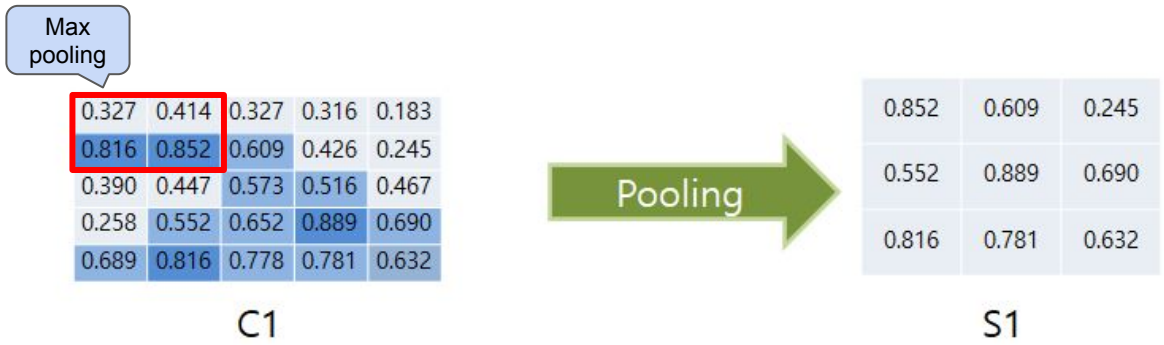
0	0	0	0	0
0	1	1	1	0
0	1	1	1	0
0	1	1	1	0
0	0	0	0	0

- Emboss

	-2	-1	0	
	-1	1	1	
	0	1	2	

Pooling Layer

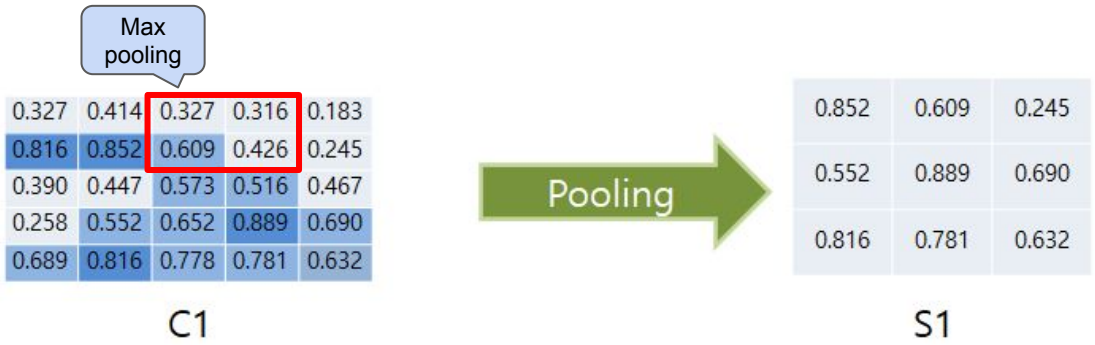
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Pooling Layer

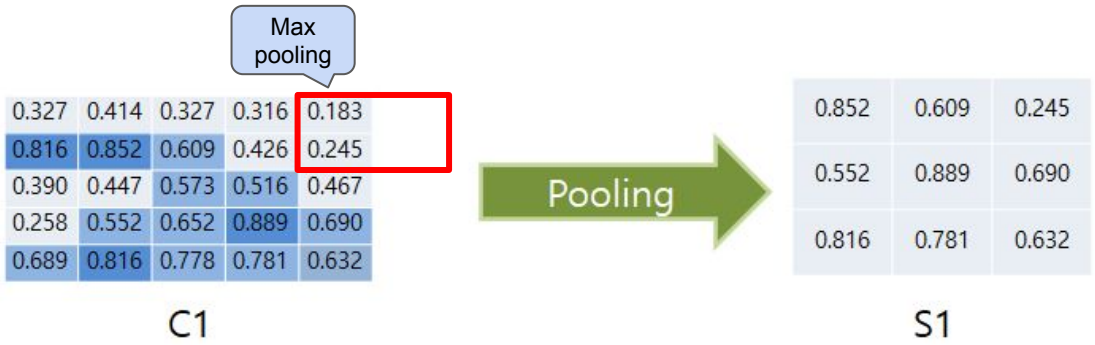
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Pooling Layer

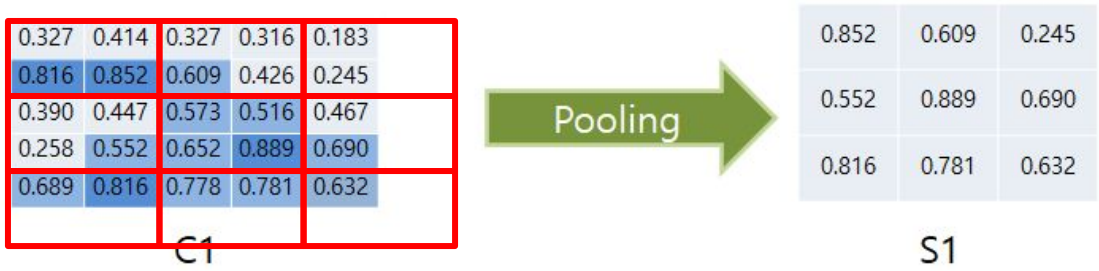
- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

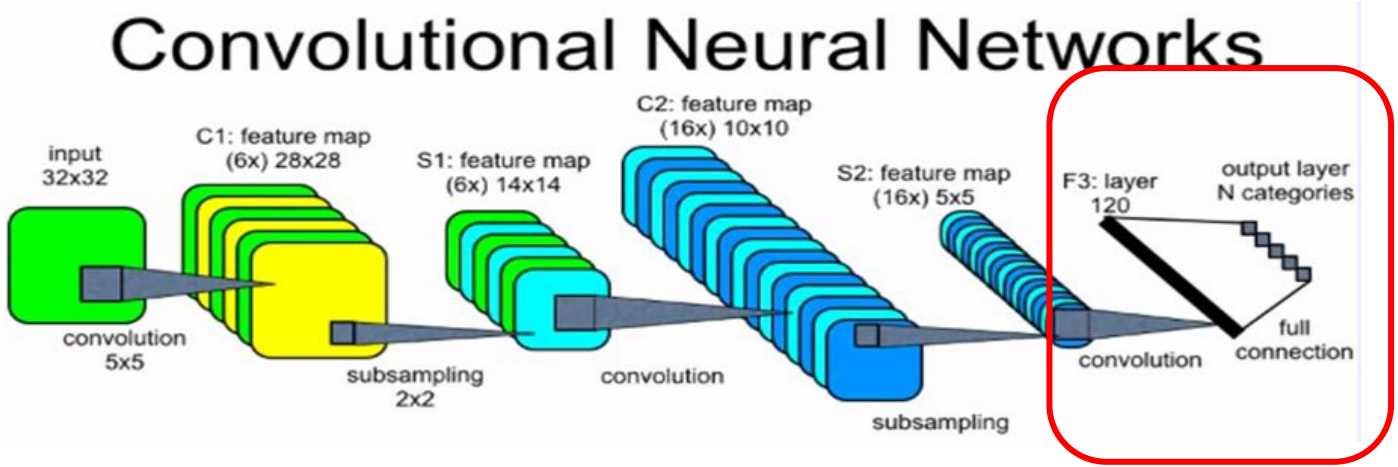
Pooling Layer

- Convolution을 실행한 C1에서 특징 (통상적으로 2x2 4칸 중 가장 큰 값을 고르는 Max Pooling을 사용)을 추려 냄



- S1을 새로운 Input처럼 생각하며 Convolution과 Subsampling을 반복

Fully Connected Layer



- MLP (Multi Layer Perceptron)에서 사용된 구조와 동일함
- 일반적으로 여러 필터를 통과한 특징들을 **flatten** 시켜
  하나의 벡터로 만든 후 최종 **output** 예측 직전단계에 활용함

Some Techniques

Data Augmentation

- Random cropping to 224 X 224 images
- Altering RGB/Grayscale
- Flipping
- ...

Some Techniques

- Input 데이터에 이미지 변형 과정을 거쳐 데이터 자체의 특징은 유지하면서 데이터의 양은 늘려 학습의 효율을 크게 증가시킬 수 있는 기법. 데이터의 수가 적을 때 효과가 좋음.

Rotation

Zoom

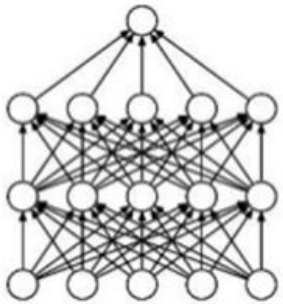
Flip (Horizontal, Vertical)

Shift (Width, Height)

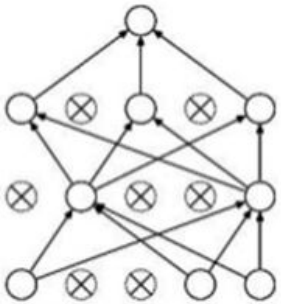
Some Techniques

Dropout – Avoiding overfitting

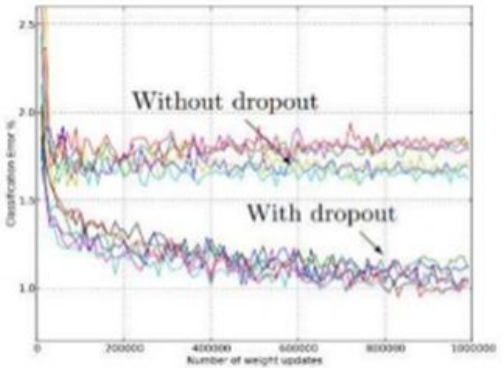
- Fully-connected layers Node의 0.0~0.5를 데이터에 따라 off함
- Mini-batch마다 사용하는 neuron의 종류가 달라짐
- 여러 개의 architecture를 ensemble하는 효과가 생김
- Better generalization



(a) Standard Neural Net



(b) After applying dropout.



Training CIFAR-10 data

합성곱 신경망 (Convolutional Neural Network)

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_6 (Conv2D)            (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_8 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_30 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_31 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/5
1563/1563 [==============================] - 5s 3ms/step - loss: 1.4835 - accuracy: 0.4626
Epoch 2/5
1563/1563 [==============================] - 4s 3ms/step - loss: 1.1241 - accuracy: 0.6044
Epoch 3/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.9768 - accuracy: 0.6571
Epoch 4/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.8794 - accuracy: 0.6931
Epoch 5/5
1563/1563 [==============================] - 4s 3ms/step - loss: 0.8053 - accuracy: 0.7190
313/313 [==============================] - 1s 2ms/step - loss: 0.8871 - accuracy: 0.6957
[0.887057900428772, 0.6956999897956848]
```

훨씬 적은 파라미터 수로 좋은 결과를 얻었음

Training CIFAR-10 data

epoch 5 -> 50

```
Epoch 45/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1037 - accuracy: 0.9643
Epoch 46/50
1563/1563 [==============================] - 5s 3ms/step - loss: 0.1198 - accuracy: 0.9587
Epoch 47/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1102 - accuracy: 0.9617
Epoch 48/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1116 - accuracy: 0.9626
Epoch 49/50
1563/1563 [==============================] - 4s 3ms/step - loss: 0.1026 - accuracy: 0.9651
Epoch 50/50
1563/1563 [==============================] - 5s 3ms/step - loss: 0.1105 - accuracy: 0.9628
313/313 [==============================] - 1s 2ms/step - loss: 2.5728 - accuracy: 0.6903
[2.5728492736816406, 0.6902999877929688]
```

학습 epoch를 50으로 늘렸지만 test accuracy에서 개선이 없었음

Training CIFAR-10 data

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_11 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
flatten_1 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 486,794
Trainable params: 486,794
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 47/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0700 - accuracy: 0.9794
Epoch 48/50
1563/1563 [==============================] - 11s 7ms/step - loss: 0.0782 - accuracy: 0.9768
Epoch 49/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0722 - accuracy: 0.9784
Epoch 50/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.0712 - accuracy: 0.9786
313/313 [==============================] - 1s 4ms/step - loss: 3.1726 - accuracy: 0.6930
[3.172563076019287, 0.6930000185966492]
```

accuracy: 0.6930

layer를 추가했으나 비슷한 성능

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_14 (Conv2D)           (None, 30, 30, 64)        1792
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 15, 15, 64)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 13, 13, 128)       73856
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 4, 4, 128)         147584
_________________________________________________________________
flatten_2 (Flatten)          (None, 2048)              0
_________________________________________________________________
dropout (Dropout)            (None, 2048)              0
_________________________________________________________________
dense_4 (Dense)              (None, 128)               262272
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1290
=================================================================
Total params: 486,794
Trainable params: 486,794
Non-trainable params: 0
_________________________________________________________________
```

```
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1414 - accuracy: 0.9532
Epoch 48/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1410 - accuracy: 0.9519
Epoch 49/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1336 - accuracy: 0.9546
Epoch 50/50
1563/1563 [==============================] - 12s 7ms/step - loss: 0.1382 - accuracy: 0.9543
313/313 [==============================] - 1s 4ms/step - loss: 1.5392 - accuracy: 0.7249
[1.539198875427246, 0.7249000072479248]
```

accuracy: 0.7249

Dropout을 적용한 결과 약간의 개선

Fashion-MNIST data

레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Fashion-MNIST data

데이터 불러오기 & 정규화

```
import tensorflow as tf
fashion_mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()

x_train = x_train.reshape((60000, 28, 28, 1))
x_test = x_test.reshape((10000, 28, 28, 1))

# 픽셀 값을 0~1 사이로 정규화합니다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Fashion-MNIST data

모델 정의하기

CNN?

```
from tensorflow.keras import datasets, layers, models

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

```

MLP?

```
from tensorflow import keras
# layer를 하나씩 추가하는 방식
model = keras.models.Sequential()
model.add(keras.layers.Flatten(input_shape=[28,28]))
model.add(keras.layers.Dense(300, activation="relu"))
model.add(keras.layers.Dense(100, activation="relu"))
model.add(keras.layers.Dense(10, activation="softmax"))

# 층 리스트로 전달하는 방식
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28,28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax"),
])
```

Fashion-MNIST data

모델 컴파일 & 모델 학습 및 평가

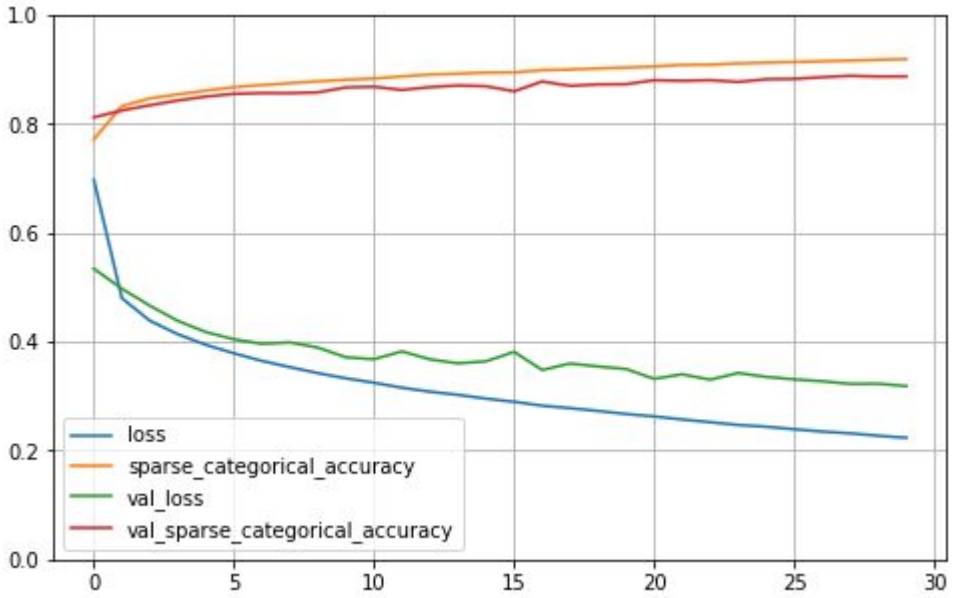
```
model.compile(loss="sparse_categorical_crossentropy", optimizer="sgd", metrics=["accuracy"])
history = model.fit(x_train, y_train, epochs=30, validation_data=(x_test, y_test))
```

```
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2521 - sparse_categorical_accuracy: 0.9088 - val_loss: 0.3303 - val_sparse_categorical_accuracy: 0.8804
Epoch 24/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2471 - sparse_categorical_accuracy: 0.9111 - val_loss: 0.3425 - val_sparse_categorical_accuracy: 0.8771
Epoch 25/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2438 - sparse_categorical_accuracy: 0.9126 - val_loss: 0.3354 - val_sparse_categorical_accuracy: 0.8821
Epoch 26/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2392 - sparse_categorical_accuracy: 0.9137 - val_loss: 0.3308 - val_sparse_categorical_accuracy: 0.8827
Epoch 27/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2350 - sparse_categorical_accuracy: 0.9150 - val_loss: 0.3274 - val_sparse_categorical_accuracy: 0.8858
Epoch 28/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2317 - sparse_categorical_accuracy: 0.9164 - val_loss: 0.3225 - val_sparse_categorical_accuracy: 0.8885
Epoch 29/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2273 - sparse_categorical_accuracy: 0.9179 - val_loss: 0.3228 - val_sparse_categorical_accuracy: 0.8870
Epoch 30/30
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2234 - sparse_categorical_accuracy: 0.9190 - val_loss: 0.3186 - val_sparse_categorical_accuracy: 0.8873
```

Fashion-MNIST data

학습 곡선 확인하기

```
import pandas as pd
import matplotlib.pyplot as plt

pd.DataFrame(history.history).plot(figsize=(8,5))
plt.grid(True)
plt.gca().set_ylim(0, 1)
plt.show()
```



Fashion-MNIST data

모델 평가

```
1 model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.3247 - sparse_categorical_accuracy: 0.8860
[0.32465073466300964, 0.8859999775886536]
```

데이터 예측하기

```
1 X_new = x_test[:5]
2 y_proba = model.predict(X_new)
3 y_proba.round(2)
```

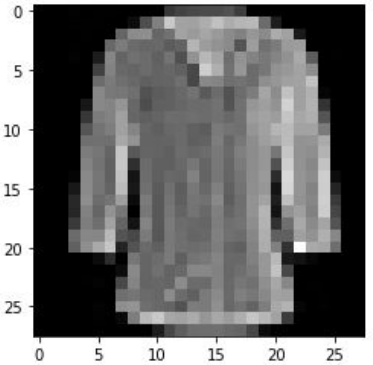
```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.49, 0.  , 0.02, 0.  , 0.01, 0.  , 0.48, 0.  , 0.  , 0.  ]],
```

T-shirt/top

Shirt

```
1 plt.imshow(X_new[4].reshape(28,28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7f31d03fea90>
```



Fashion-MNIST data

모델 저장 & 복원

```
1 model.save('fashion_mnist_mlp_model.h5')
```

```
1 model_loaded = keras.models.load_model('fashion_mnist_mlp_model.h5')
```

```
1 y_proba = model_loaded.predict(X_new)
2 y_proba.round(2)
```

```
array([[0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  , 0.98],
       [0.  , 0.  , 0.98, 0.  , 0.02, 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.49, 0.  , 0.02, 0.  , 0.01, 0.  , 0.48, 0.  , 0.  , 0.  ]],
```

복원된 모델의 예측결과가
저장한 모델과 같은 것을 확인함