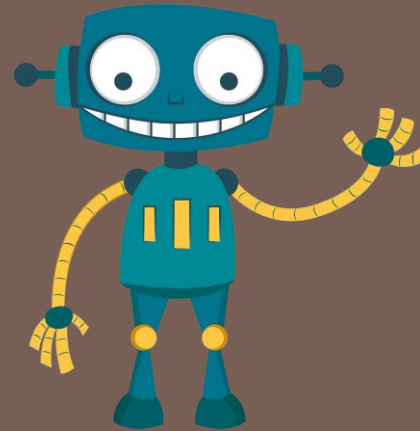
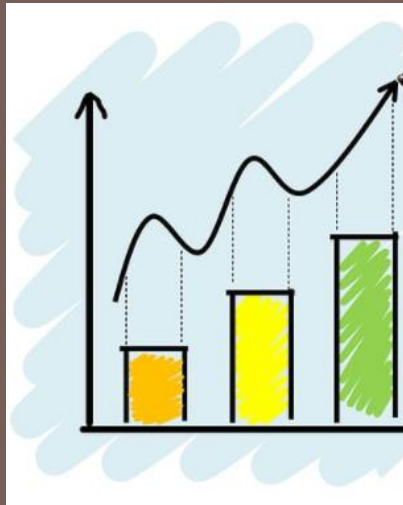


파이썬 익스프레스



16장 기계학습

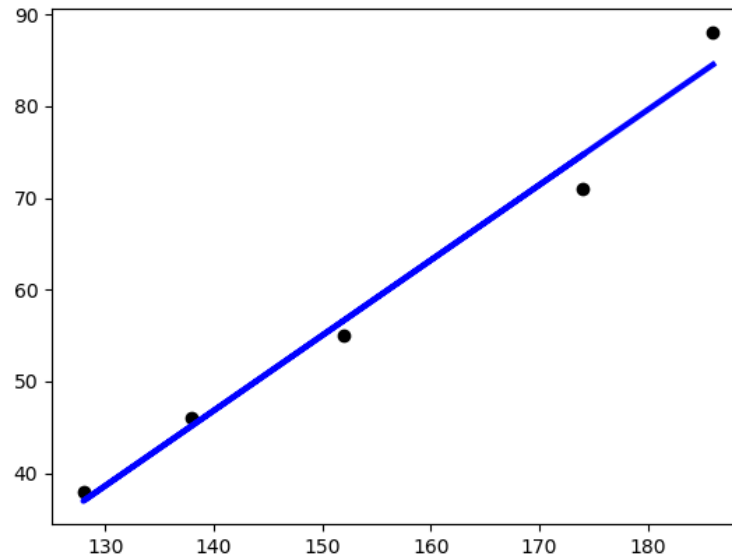
학습 목표

- 기계 학습의 개념에 대하여 살펴본다.
- 선형 회귀 문제를 **sklearn** 라이브러리를 이용하여 실습해본다.
- XOR 문제를 케라스 라이브러리를 이용하여 실습해본다.
- 숫자 인식 프로그램을 케라스 라이브러리를 이용하여 실습해본다.



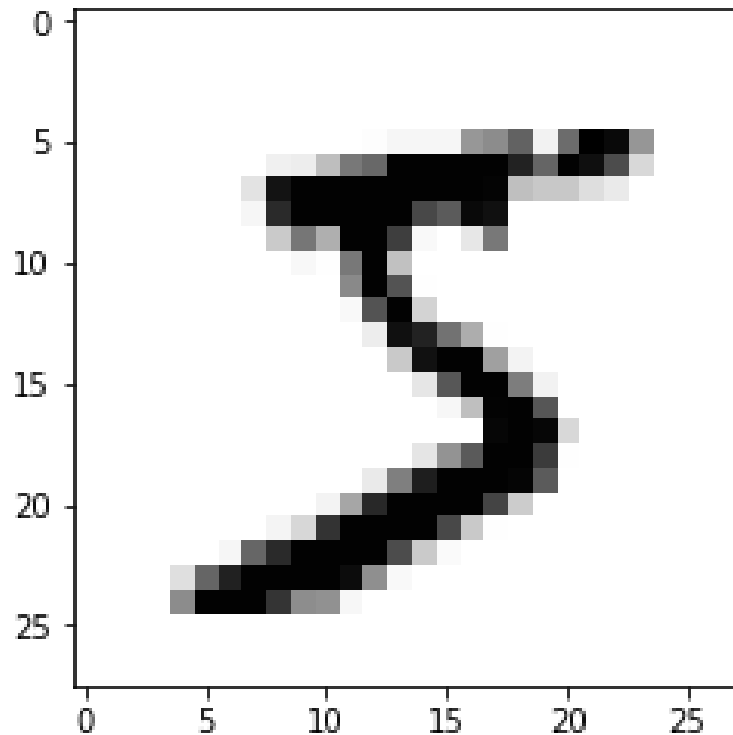
이번 장에서 만들 프로그램

- 선형 회귀 분석 프로그램을 sklearn 라이브러리를 이용하여 작성해 보자.



이번 장에서 만들 프로그램

- 필기체 숫자 인식 프로그램을 케라스 라이브러리를 이용하여 작성해 보자.

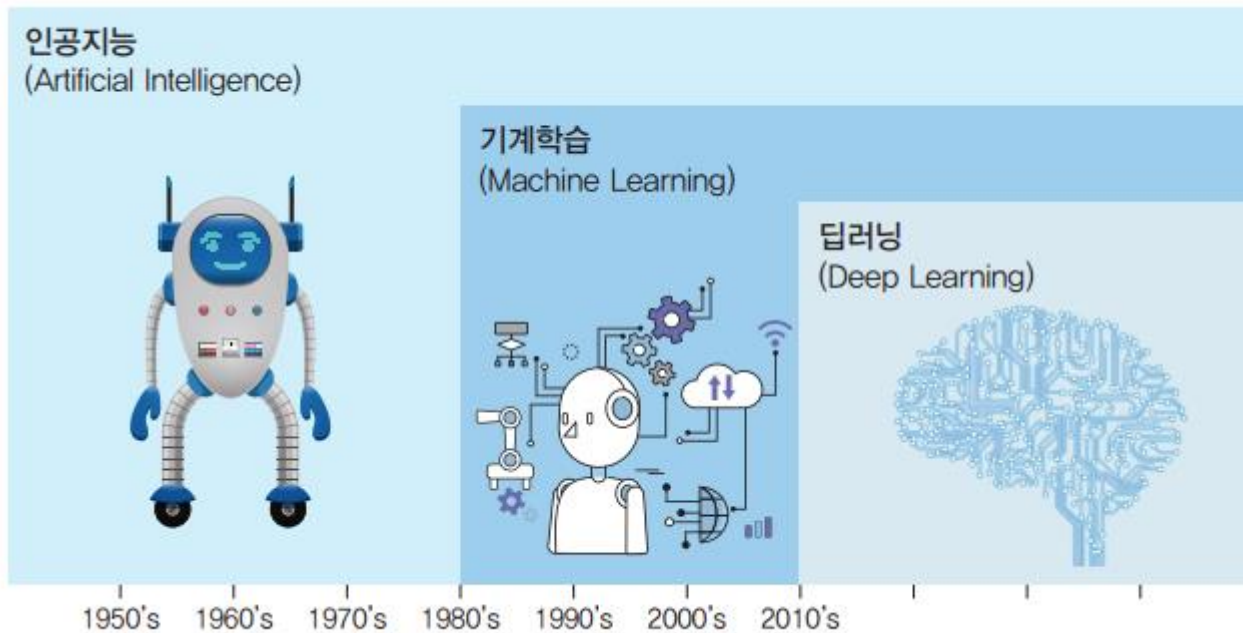


기계학습

- 기계 학습(machine learning)은 인공 지능의 한 분야로, 컴퓨터에 학습 기능을 부여하기 위한 연구 분야이다.
- 이들 알고리즘은 항상 고정적인 의사 결정을 하는 프로그램과는 다르게, 데이터 중심의 예측 또는 결정을 내릴 수 있다.
- 기계 학습은 어떤 문제에 대하여 명시적 알고리즘을 설계하고 프로그래밍하는 것이 어렵거나 불가능한 경우에 주로 사용된다.



인공지능, 기계학습, 딥러닝



기계 학습이 중요하게 사용되는 분야

- 기계 학습은 문제를 해결하는데, 많은 경우가 있어서, 각각의 경우를 정확하게 처리하는 것이 불가능한 경우에 필요하다.

기계학습을 사용하는 않는 경우



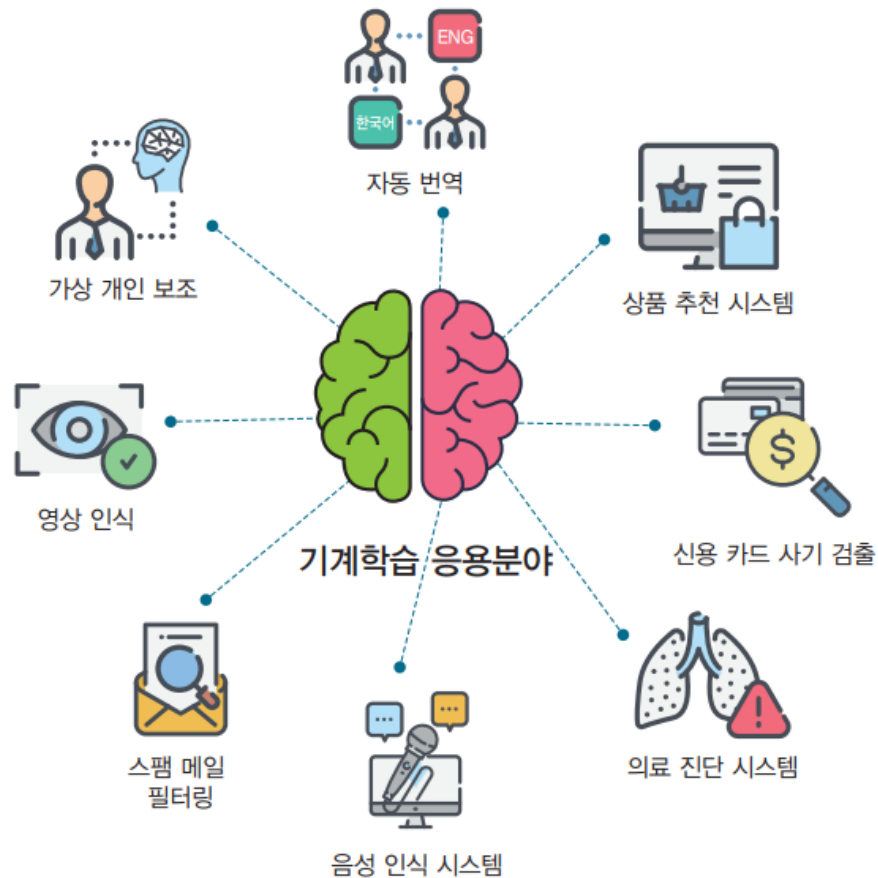
기계학습을 사용하는 경우



기계 학습이 중요하게 사용되는 분야

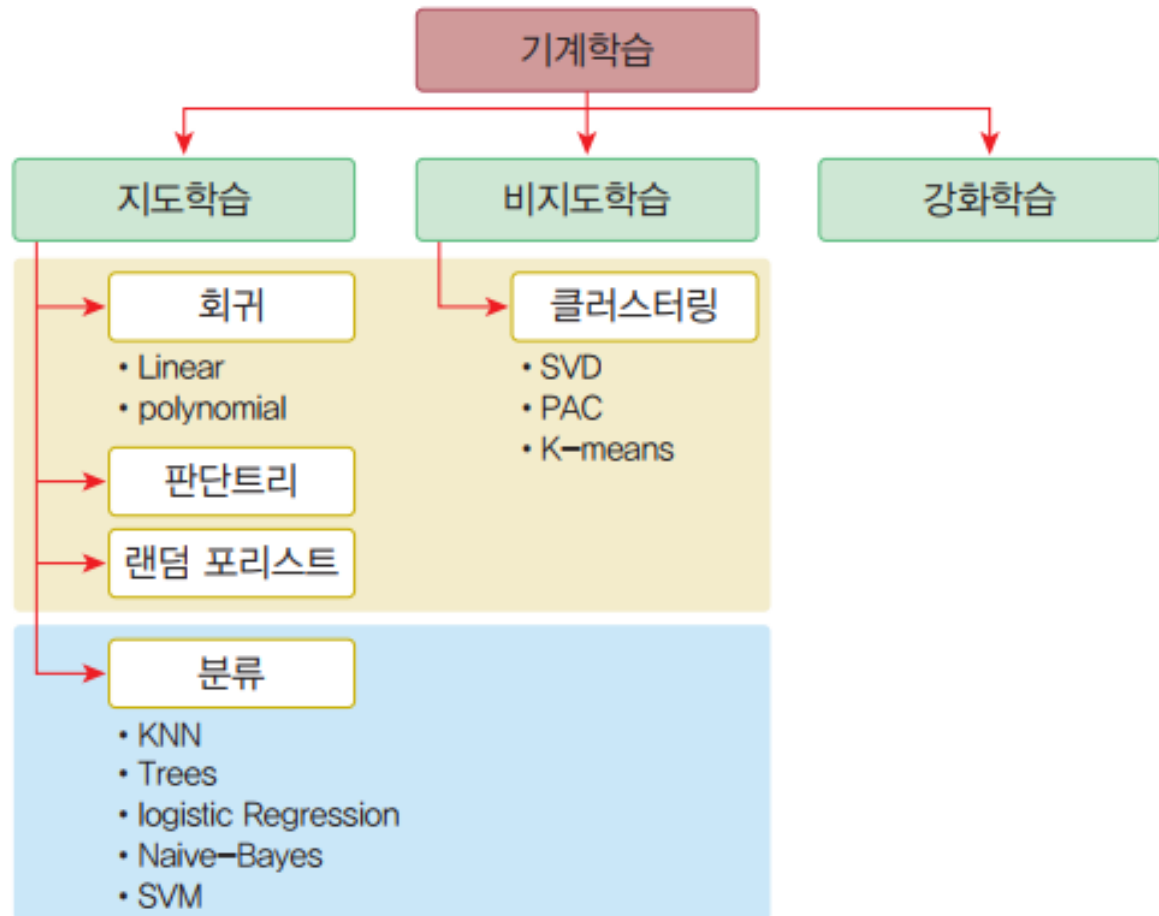
- 음성 인식처럼 프로그램으로 작성하기에는 규칙과 공식이 너무 복잡할 때(인간도 잘 모르는 분야이다.)
- 컴퓨터를 사용한 금융 - 신용 평가, 주식 거래: 보안 시스템에서 침입을 탐지하거나 신용 카드 거래 기록에서 사기를 감지하는 경우처럼 작업 규칙이 지속적으로 바뀌는 상황일 때
- 주식 거래나 에너지 수요 예측, 쇼핑 추세 예측의 경우처럼 데이터 특징이 계속 바뀌고 프로그램을 계속해서 변경해야 하는 상황일 때

기계 학습이 중요하게 사용되는 분야



기계학습의 분류

신경망은 모든 기계학습
분야에서 사용될 수
있습니다!!!



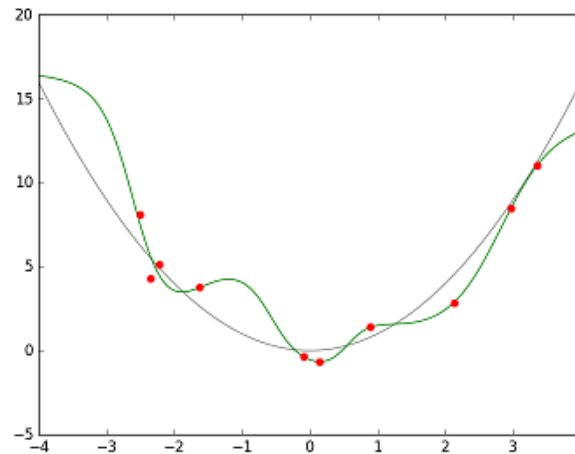
기계 학습

- 기계 학습은 항상 입력을 받아서 출력하는 함수 $y=f(x)$ 를 학습한다고 생각할 수 있다. (함수 근사)

우리는 함수를
학습합니다.



입력

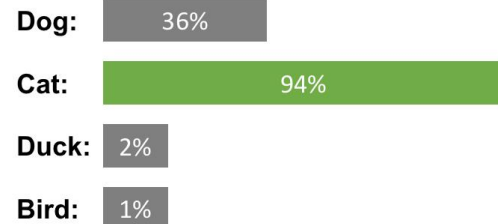
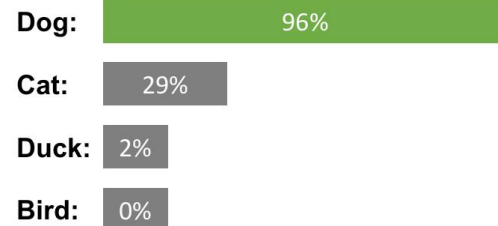


출력

기계 학습(machine learning) == 함수 근사(function approximation)

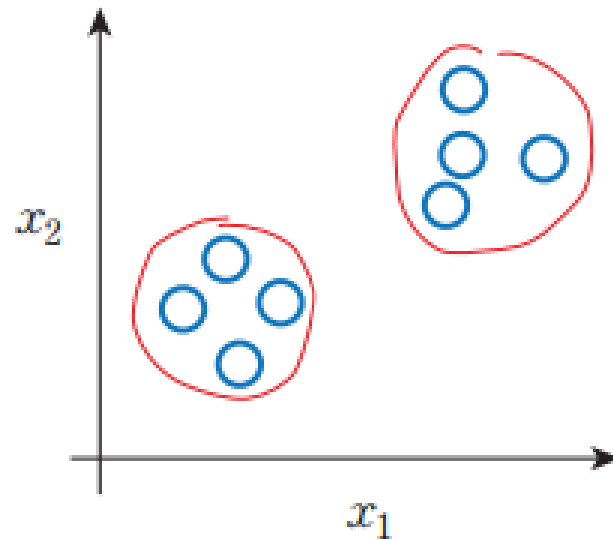
지도학습

- 컴퓨터는 "교사"에 의해 주어진 예제와 정답(레이블)을 제공받는다. 지도 학습의 목표는 입력을 출력에 매핑하는 일반적인 규칙을 학습하는 것이다.



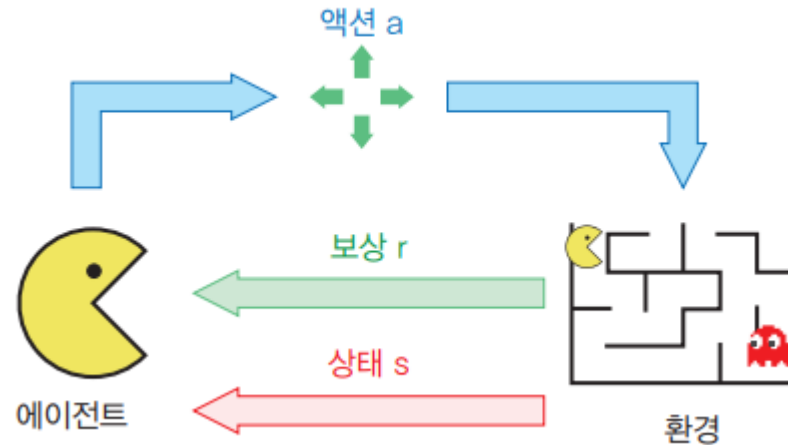
비지도학습

- 외부에서 정답(레이블)이 주어지지 않고 학습 알고리즘이 스스로 입력에서 어떤 구조를 발견하는 학습이다.



강화 학습

- 보상이나 처벌 형태의 피드백으로 학습이 이루어지는 기계 학습이다. 주로 차량 운전이나 상대방과의 경기 같은 동적인 환경에서 프로그램의 행동에 대한 피드백만 제공된다.



지도학습

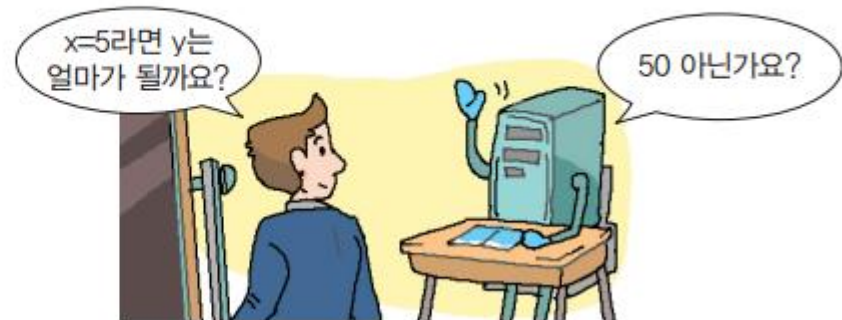
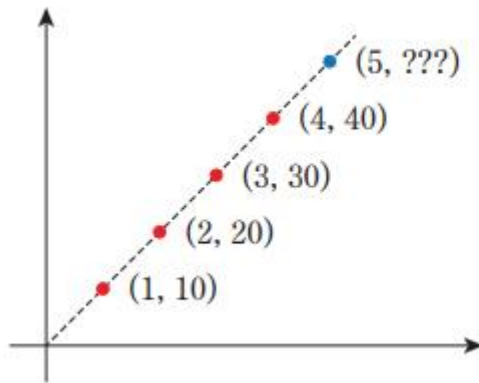
- 지도 학습은 입력(x)과 출력(y)이 주어질 때, 입력에서 출력으로의 매핑 함수를 학습하는 것이라 할 수 있다.

$$y = f(x)$$



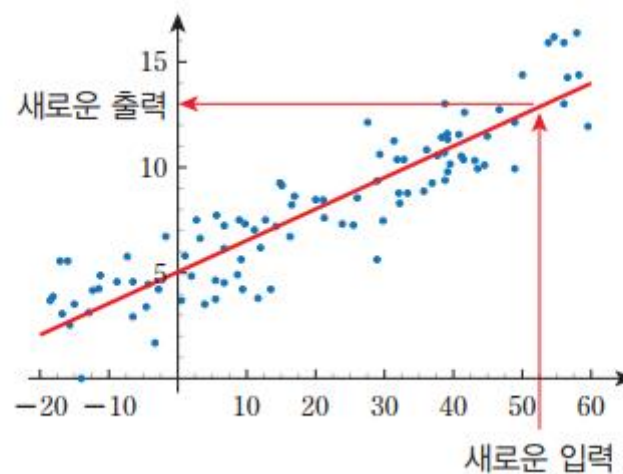
지도학습의 예

- 입력 데이터로 직선 $y=10x$ 위에 있는 점 $(1, 10)$, $(2, 20)$, $(3, 30)$, $(4, 40)$ 들이 주어졌다고 하자.
- 학습이 끝난 후에 $x=5$ 를 입력하면 컴퓨터가 $y=50$ 이라는 답을 할 수 있도록 만드는 것이 지도 학습이다.



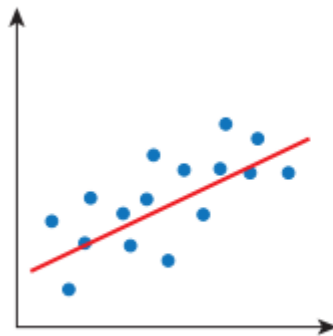
지도 학습: 회귀(regression)

- 회귀(regression)란 일반적으로 예제 데이터들을 2차원 공간에 찍은 후에, 이들 데이터들을 가장 잘 설명하는 직선이나 곡선을 찾는 문제라고 할 수 있다.

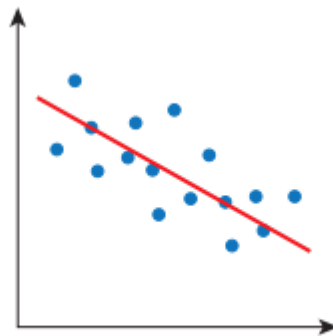


지도 학습: 회귀(regression)

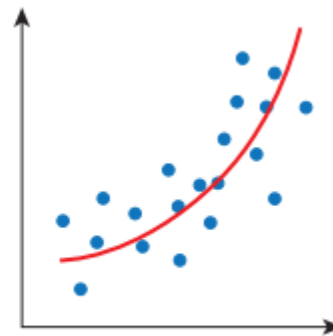
- 회귀에서는 출력(y)의 형태가 이산적이 아니라 연속적이다.
- $y = f(x)$ 에서 입력 x 와 출력 y 가 모두 실수이다.



선형



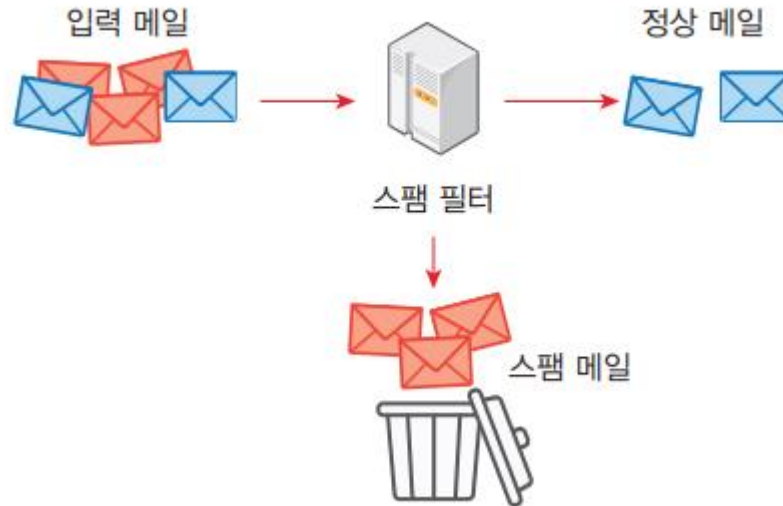
선형



비선형

지도 학습: 분류(classification)

- 식 $y = f(x)$ 에서 출력 y 가 이산적(discrete)인 경우에 이것을 분류 문제 (또는 인식 문제)라고 부른다. 분류는 입력을 2개 이상의 클래스로 나눈다.



비지도학습

- 비지도 학습(unsupervised Learning)은 “교사” 없이 컴퓨터가 스스로 입력들을 분류하는 것을 의미한다. 식 $y = f(x)$ 에서 정답인 레이블 y 가 주어지지 않는 것이다.



비지도학습

- 가장 대표적인 비지도 학습이 클러스터링(군집화, **clustering**)이다. 클러스터링이란 입력 데이터 간의 거리를 계산하여서 입력을 몇 개의 군집으로 나누는 방법이다. **K-means** 클러스터링이 가장 고전적인 클러스터링 방법이다.



특징(features)

- 특징이란 우리가 학습 모델에게 공급하는 입력이다. 가장 간단한 경우에는 입력 자체가 특징이 된다.
- (예)
 - 이메일에 “검찰”이라는 문자 포함 여부(yes 또는 no)
 - 이메일에 “광고”, “선물 교환권”이나 “이벤트 당첨” 문자열 포함 여부(yes 또는 no)
 - 이메일의 제목이나 본문에 있는 ‘★’과 같은 특수 기호의 개수(정수)
 - ...



학습데이터와 테스트데이터

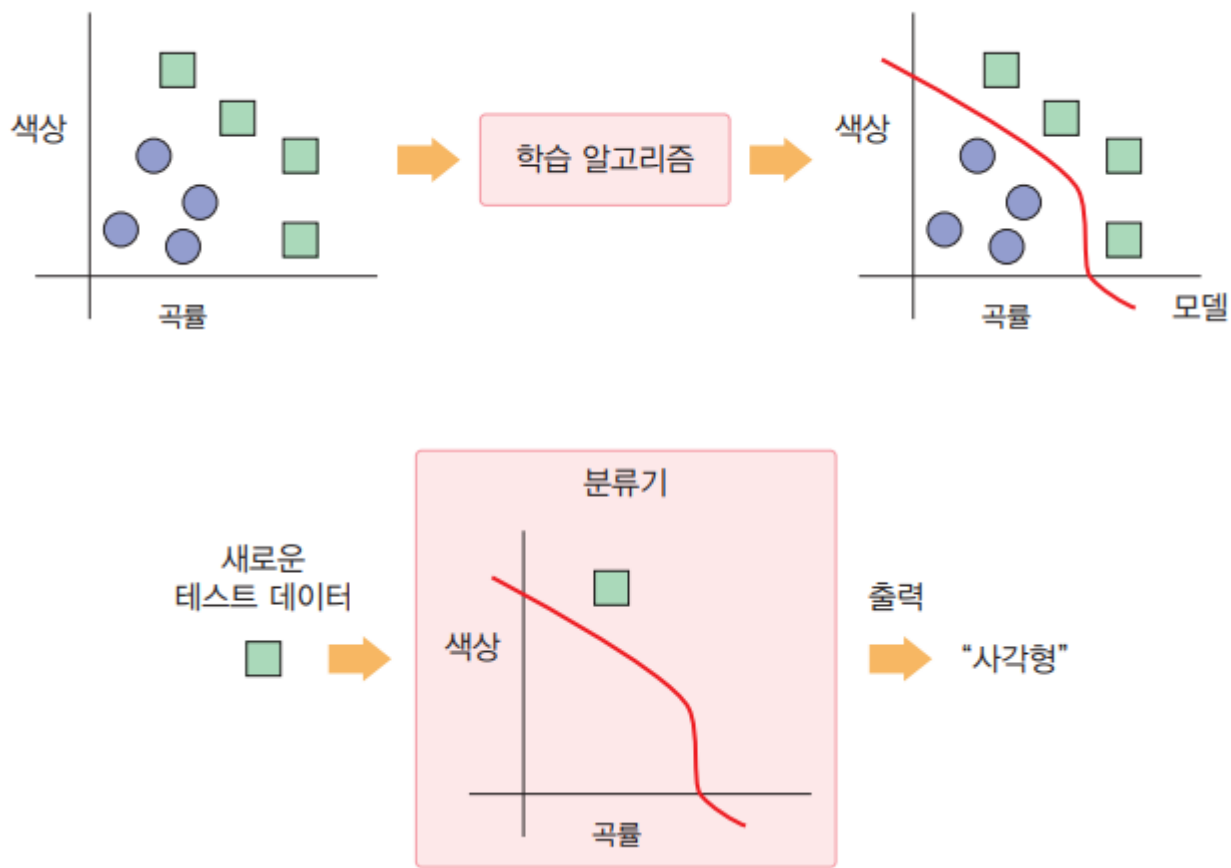
- 데이터를 학습시킬 때, 데이터를 원형 그대로 사용하는 때도 있지만 일반적으로는 데이터에서 어떤 특징을 추출하여 이것으로 학습시키고 테스트하게 된다.
- 이메일에 “검찰”이라는 단어 포함 여부(yes 또는 no)
- 이메일에 “광고”, “선물 교환권”이나 “이벤트 당첨” 단어 포함 여부(yes 또는 no)
- 이메일 발신자의 도메인(문자열)
- 이메일의 제목이나 본문에 있는 ‘★’과 같은 특수 기호의 개수(정수)



레이블과 샘플

- 레이블(label)
 - $y = f(X)$ 에서 y 변수에 해당한다.
 - 예를 들어서 농작물의 향후 가격, 사진에 표시되는 동물의 종류, 동영상의 의미 등 무엇이든지 레이블이 될 수 있다.
- 샘플, 또는 예제
 - 샘플은 기계 학습에 주어지는 특정한 예이다. $y = f(X)$ 에서 X 에 해당한다. 레이블이 있는 샘플도 있고 레이블이 없는 샘플도 있다. 지도 학습을 시키려면 레이블이 있어야 한다.

학습 데이터와 테스트 데이터

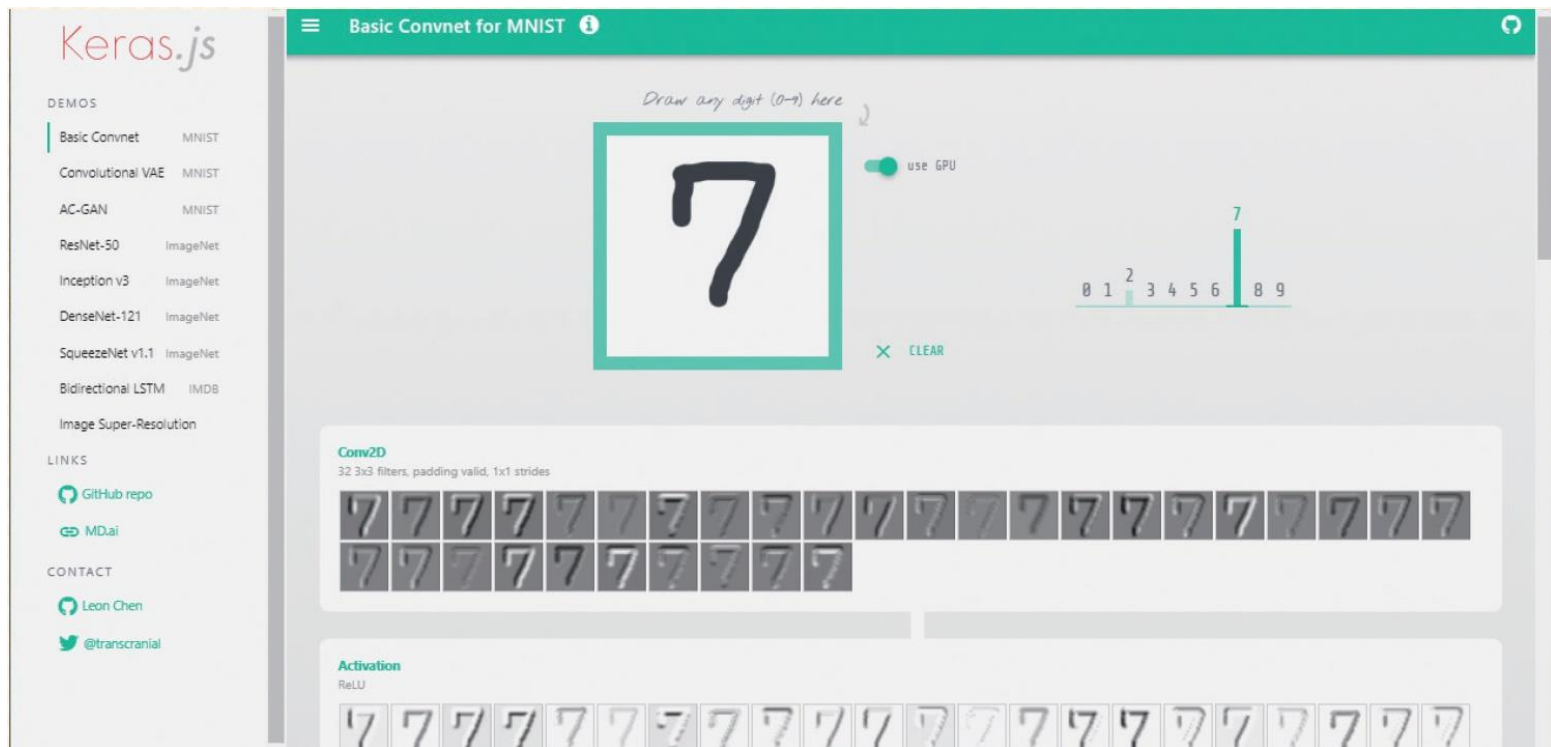


학습과 예측

- 학습(learning)은 모델을 만들거나 배우는 것을 의미한다.
- 예측(prediction)은 학습된 모델을 레이블이 없는 샘플에 적용하는 것을 의미한다. 즉 학습된 모델을 사용하여 유용한 예측(y')을 해내는 것이다.

Lab: 기계 학습 체험하기

- <https://transcranial.github.io/keras-js/#/>



Lab: 기계 학습 체험하기

□ <https://transcranial.github.io/keras-js/#/>

Keras.js

DEMOS

- Basic Convnet MNIST
- Convolutional VAE MNIST
- AC-GAN MNIST
- ResNet-50 ImageNet**
- Inception v3 ImageNet
- DenseNet-121 ImageNet
- SqueezeNet v1.1 ImageNet
- Bidirectional LSTM IMDB
- Image Super-Resolution

LINKS

- GitHub repo
- MDai

CONTACT


- Leon Chen
- @transcranial

50-layer Residual Network, trained on ImageNet

Enter a valid image URL or select an image from the dropdown:

enter image url: or select image: ☒ use GPU

visualization:



inference time: 352.5 ms (2.8 fps)

Class	Percentage
Newfoundland	77%
Tibetan mastiff	21%
Leonberg	0%
groenendael	0%
chow	0%

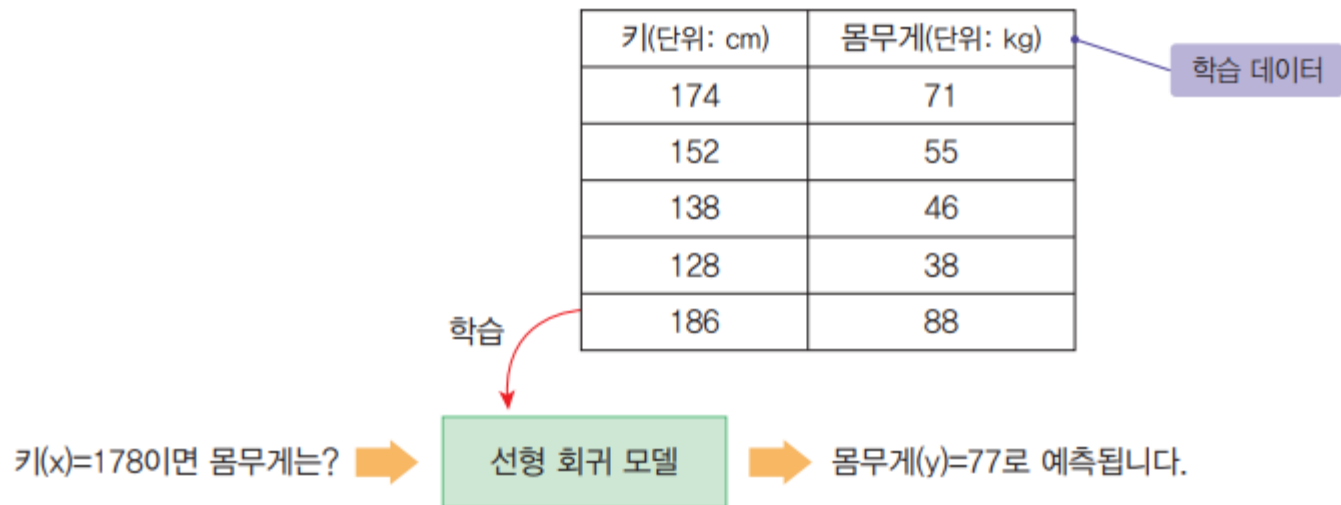
Network Architecture:

- InputLayer: shape: [224, 224, 3]
- Conv2D: 64 7x7 filters, 2x2 striding, pad to same borders
- BatchNormalization
- Activation: relu

선형 회귀 소개

- 직선의 방정식: $f(x) = mx+b$
- 선형 회귀는 입력 데이터를 가장 잘 설명하는 기울기와 절편값을 찾는 문제이다
- 선형 회귀의 기본식: $f(x) = Wx+b$
 - 기울기->가중치
 - 절편->바이어스

선형 회귀 예제



선형 회귀 예제

- 이번 절에서는 아나콘다에 포함되어 있는 **Scikit-Learn** 라이브러리를 사용하여 회귀 함수를 구현하는 방법을 살펴본다.

```
import matplotlib.pyplot as plt
from sklearn import linear_model
```

```
# 선형 회귀 모델을 생성한다.
```

```
reg = linear_model.LinearRegression()
```

```
# 데이터는 파이썬의 리스트로 만들어도 되고 아니면 넘파이의 배열로 만들어도 됨
```

```
X = [[174], [152], [138], [128], [186]]
```

```
# 학습 예제
```

```
y = [71, 55, 46, 38, 88]
```

```
# 정답
```

```
reg.fit(X, y)
```

```
# 학습 함수
```

학습 데이터 만들기

- 학습 데이터는 반드시 2차원 배열이어야 한다(한 열만 있어도 반드시 2차원 배열 형태로 만들어야 한다). 따라서 리스트의 리스트를 만들어서 다음과 같은 2차원 배열을 생성한다.

	키				몸무게
샘플 #1	174				71
샘플 #2	152				55
샘플 #3	138				46
샘플 #4	128				38
샘플 #5	186				88

선형 회귀 예제

```
>>> reg.coef_           # 직선의 기울기
array([0.82021132])
>>> reg.intercept_      # 직선의 절편
-68.0248807089298
>>> reg.score(X, y)     # 학습 점수
0.9812769231994423

>>> reg.predict([[178]])
array([77.97273347])
```

선형 회귀 예제

학습 데이터를 산포도로 그린다.

```
plt.scatter(X, y, color='black')
```

학습 데이터를 입력으로 하여 예측값을 계산한다. 직선을 가지고 예측하기 때문에 직선 상의 점이 된다.

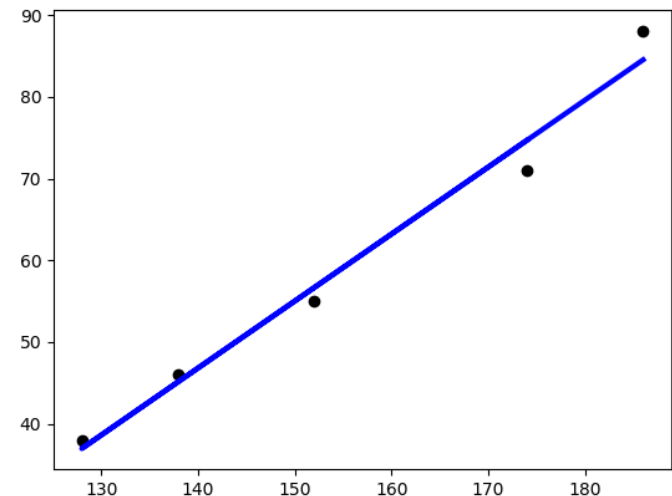
```
y_pred = reg.predict(X)
```

예측값으로 선그래프를 그린다.

직선이 그려진다.

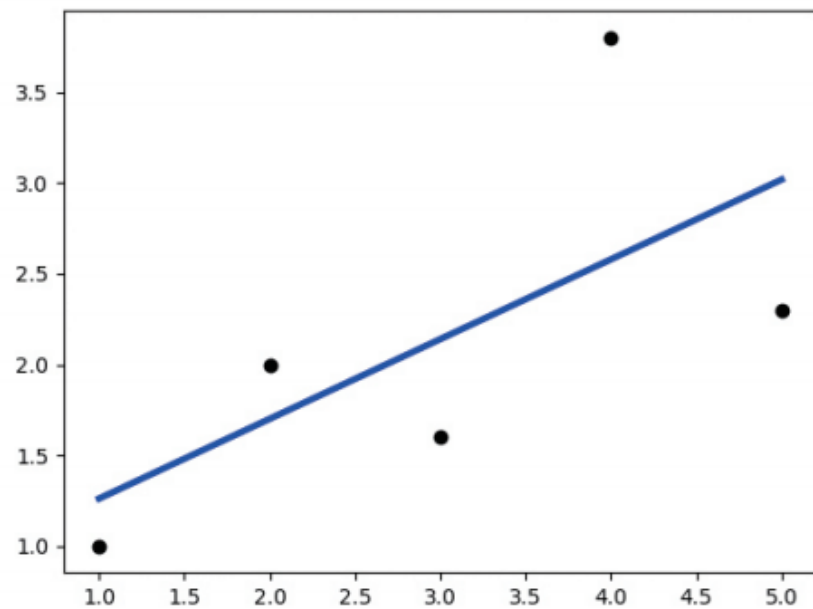
```
plt.plot(X, y_pred, color='blue', linewidth=3)
```

```
plt.show()
```



Lab: 선형 회귀 실습

X	y
1.0	1.0
2.0	2.0
3.0	1.6
4.0	3.8
5.0	2.3



선형 회귀 예제

```
import matplotlib.pyplot as plt
from sklearn import linear_model

reg = linear_model.LinearRegression()

X = [[1.0], [2.0], [3.0], [4.0], [5.0]]
y = [1.0, 2.0, 1.6, 3.8, 2.3]
reg.fit(X, y)                                     # 학습

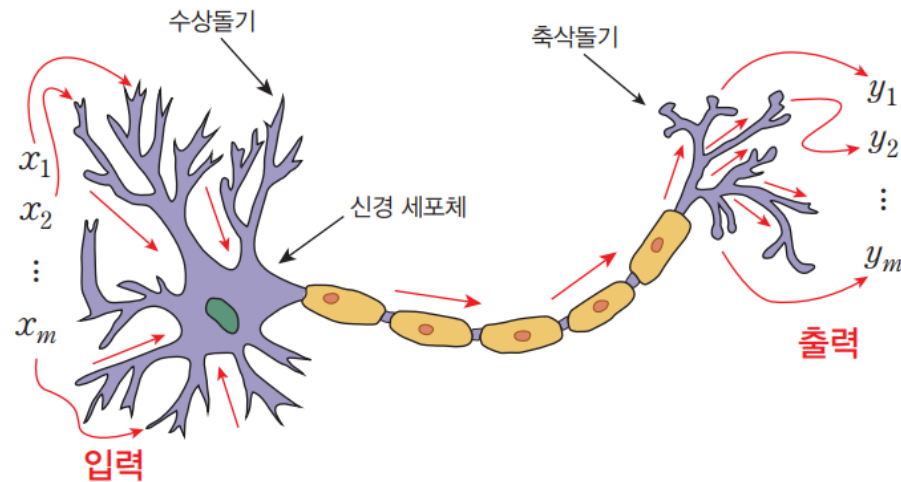
# 학습 데이터와 y 값을 산포도로 그린다.
plt.scatter(X, y, color='black')

# 학습 데이터를 입력으로 하여 예측값을 계산한다.
y_pred = reg.predict(X)

# 학습 데이터와 예측값으로 선그래프로 그린다.
# 계산된 기울기와 y 절편을 가지는 직선이 그려진다.
plt.plot(X, y_pred, color='blue', linewidth=3)
plt.show()
```

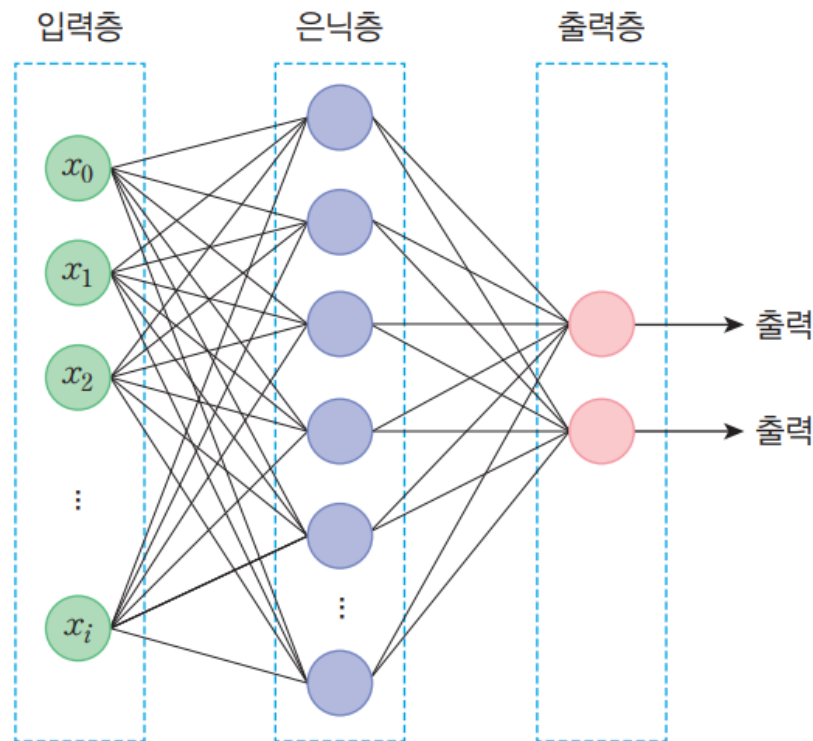
신경망

- 최근에 많은 인기를 끌고 있는 딥러닝(deep learning)의 시작은 1950년대부터 연구되어 온 인공 신경망(artificial neural network: ANN)이다. 인공신경망은 생물학적인 신경망에서 영감을 받아서 만들어진 컴퓨팅 구조이다. “스스로 생각하는 기계”는 항상 인간의 꿈이었고 1950년대에 사람들은 인간의 두뇌를 본떠서 기계로 만들려고 시도하였다.



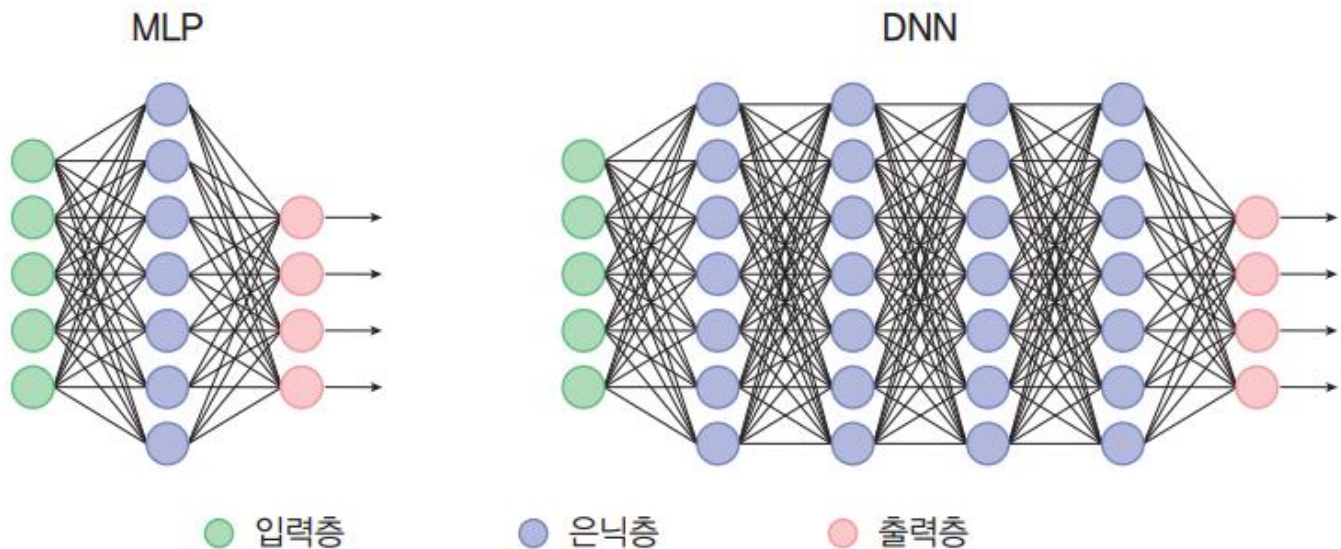
신경망

- 구체적으로 다음과 같이 입력층과 출력층 사이에 은닉층(hidden layer)을 가지고 있는 신경망을 생각할 수 있다. 아래와 같은 구조의 신경망을 다층 퍼셉트론(multilayer perceptron: MLP)이라고 부른다.



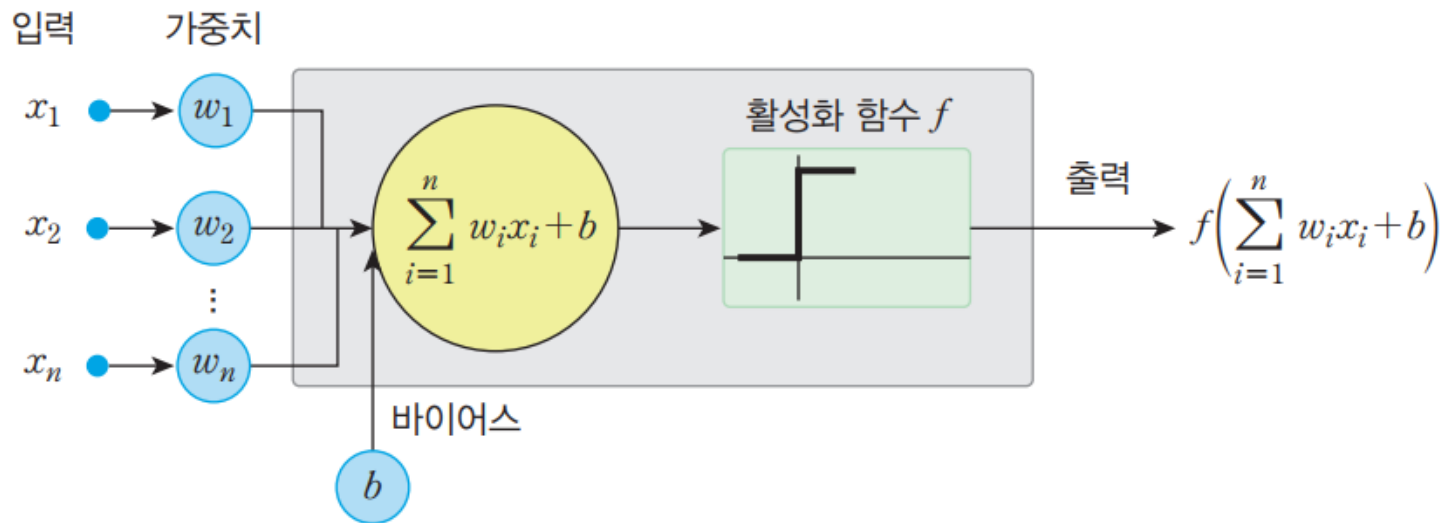
딥러닝

- "딥(deep)"이라는 용어가 은닉층이 깊다는 것을 의미한다. 최근에 딥러닝은 컴퓨터 시각, 음성 인식, 자연어 처리, 소셜 네트워크 필터링, 기계 번역 등에 적용되어서 인간 전문가에 필적하는 결과를 얻고 있다.

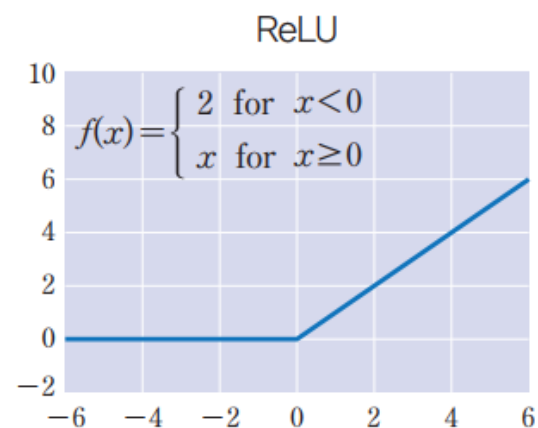
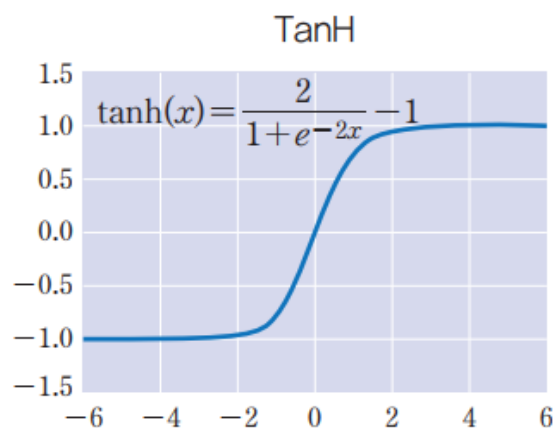
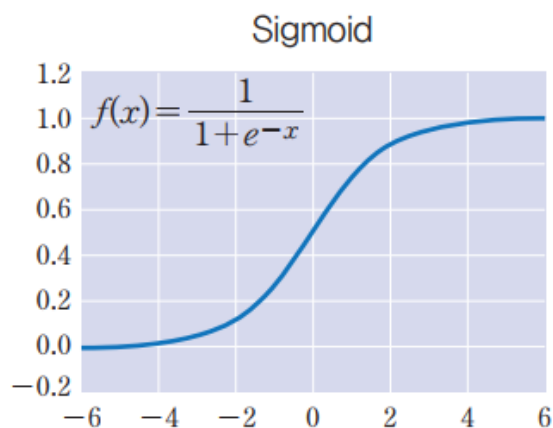


뉴런 모델

- 신경망에서는 하나의 뉴론을 다음과 같이 모델링한다.

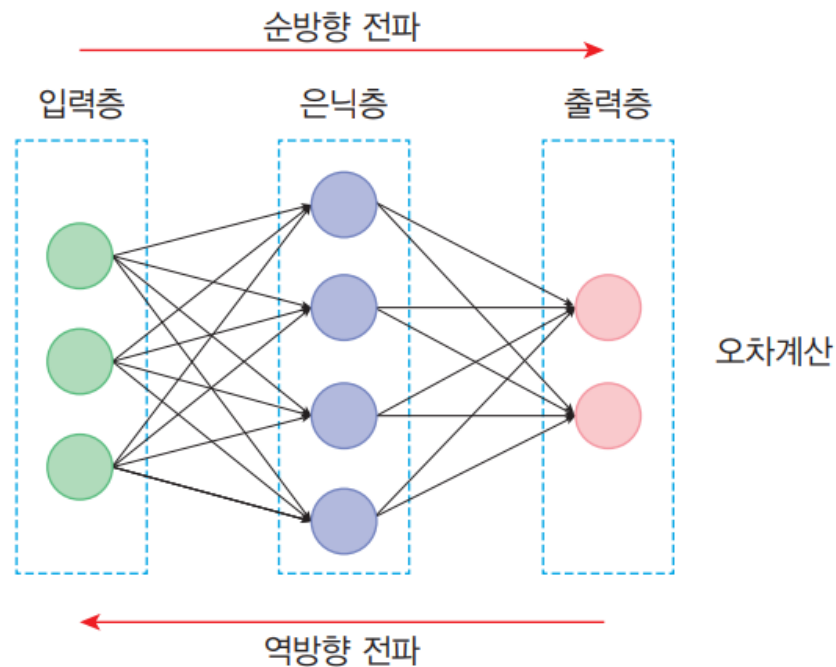


화성화 함수



역전파 학습 알고리즘

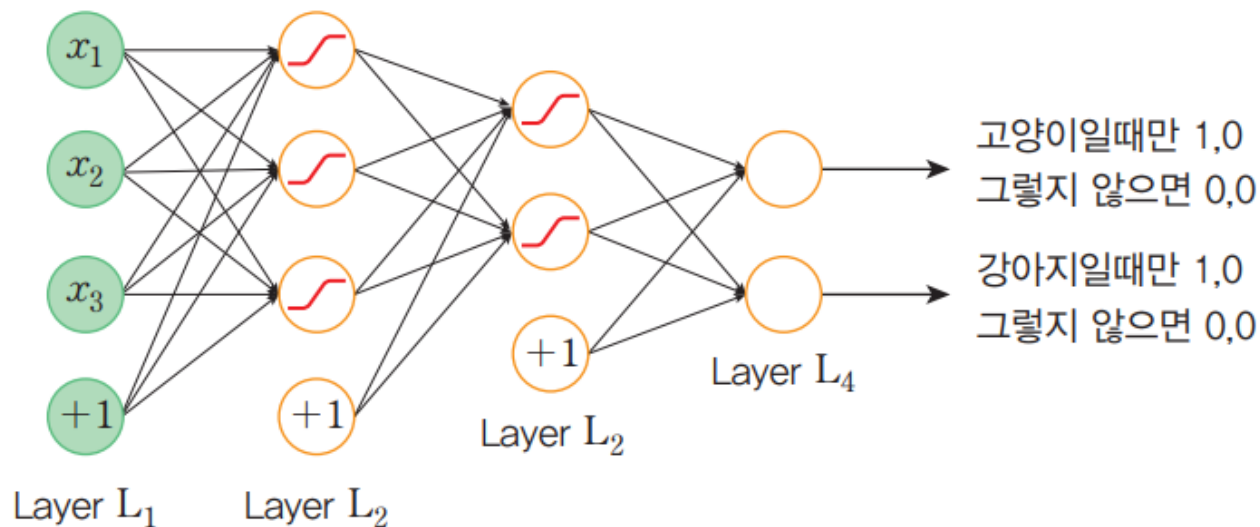
- 역전파 알고리즘은 입력이 주어지면 순방향으로 계산하여 출력을 계산한 후에 실제 출력과 우리가 원하는 출력 간의 오차를 계산한다. 이 오차를 역방향으로 전파하면서 오차를 줄이는 방향으로 가중치를 변경한다.



손실 함수란 무엇인가?

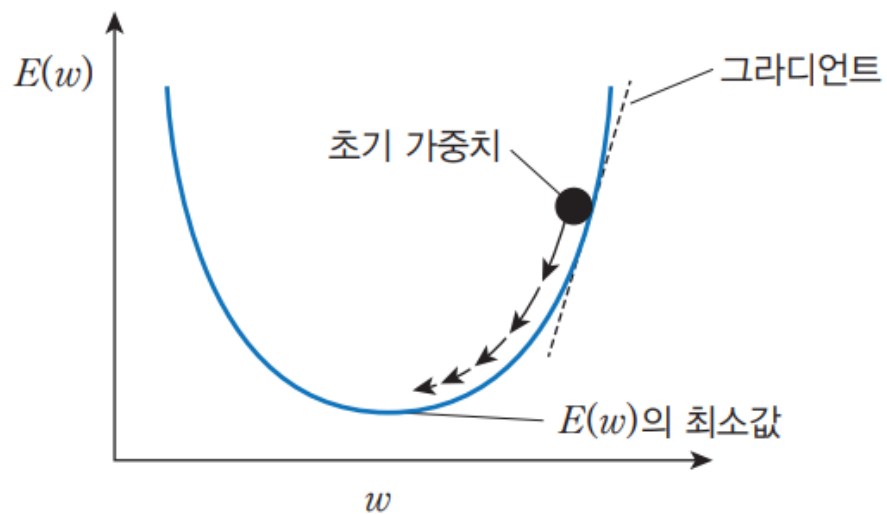


vs.



$$E(w) = \frac{1}{2} \sum_i (t_i - o_i)^2$$

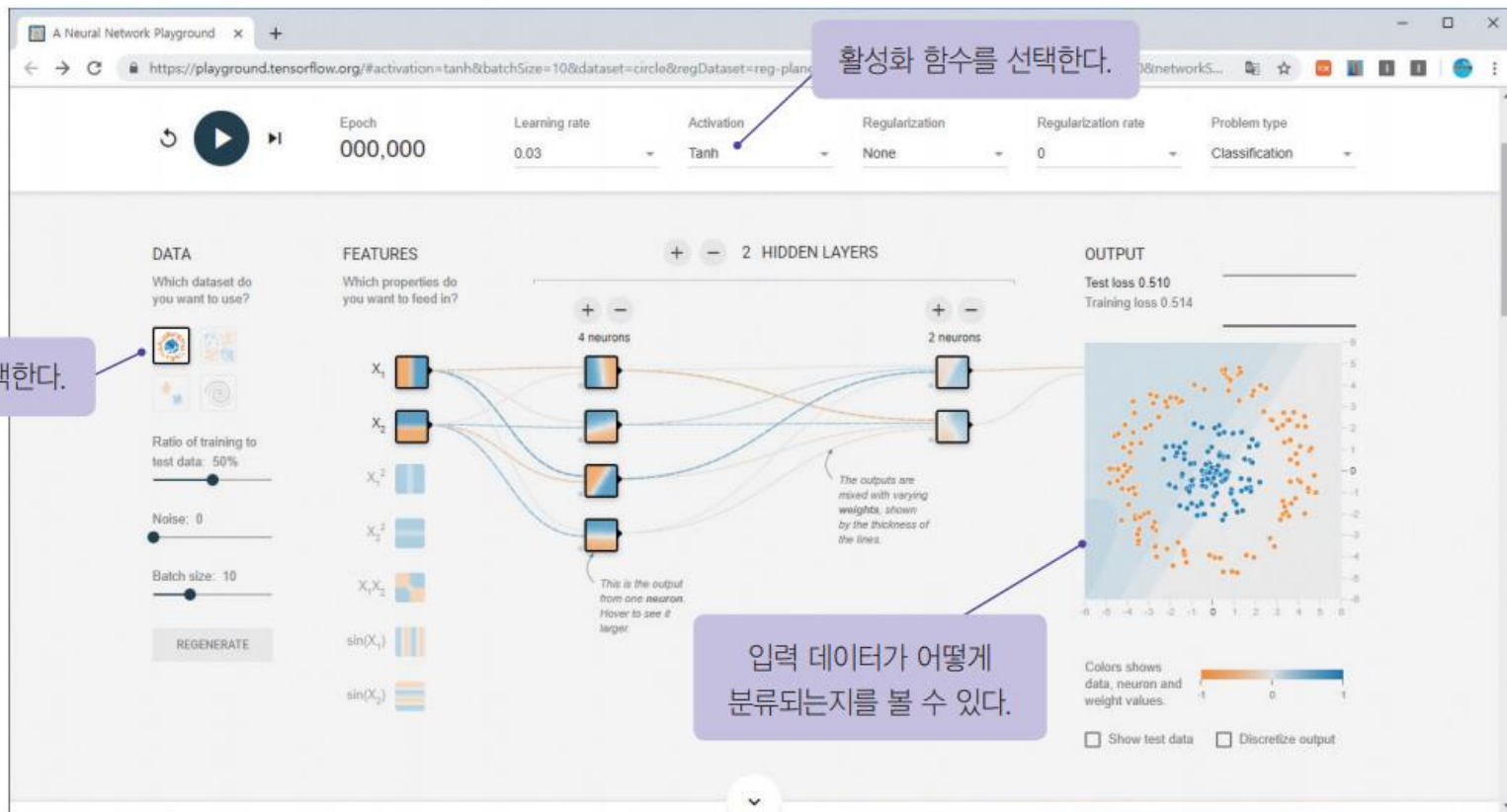
경사하강법



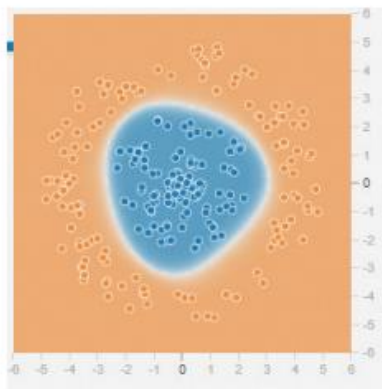
그라디언트는 접선의 기울기로 이해해도 됩니다. 접선의 기울기가 양수이면 반대로 w 를 감소시킵니다.



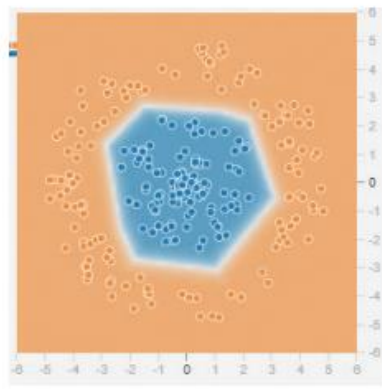
Lab: 활성화 함수 실험



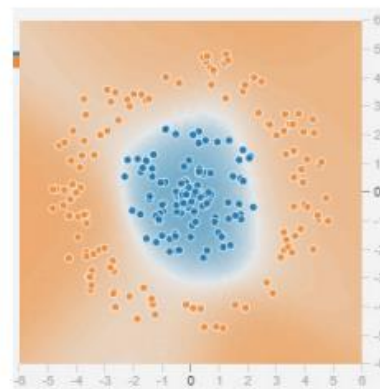
Lab: 활성화 함수 실험



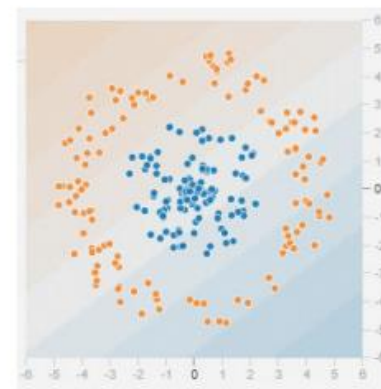
tanh



ReLU



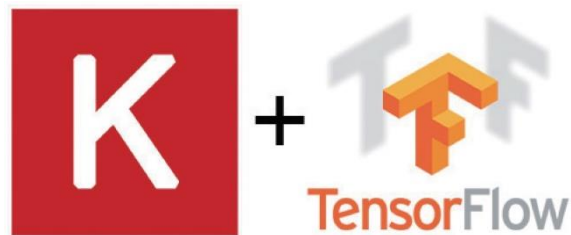
Sigmoid



Linear

Keras

- Keras는 Python으로 작성되었으며 TensorFlow , CNTK 또는 Theano에서 실행할 수 있는 고수준 딥러닝 API이다.
 - 쉽고 빠른 프로토타이핑이 가능하다.
 - 순방향 신경망, 컨볼루션 신경망과 반복적인 신경망은 물론 물론 여러 가지의 조합도 지원한다.
 - CPU 및 GPU에서 원활하게 실행된다.



텐서플로우 설치

- 텐서플로우는 아나콘다에 포함되지 않아서 **pip**로 설치
- 최신의 파이썬 버전을 아직 지원하지 못해서 아나콘다에서 가상환경을 만들어서 파이썬 **3.7** 버전으로 설정한 후에 텐서플로우 설치
- 아나콘다 프롬프트를 열어서 다음 명령 입력

```
(base) C:\Users\chun> conda create -n deep python=3.7
```

여기서 **python=3.7**이란 파이썬 버전을 **3.7**으로 설정하라는 의미이다. 이어서 다음과 같은 명령으로 “**deep**”이라는 가상환경을 활성화 상태로 만든다.

```
(base) C:\Users\chun > conda activate deep
```

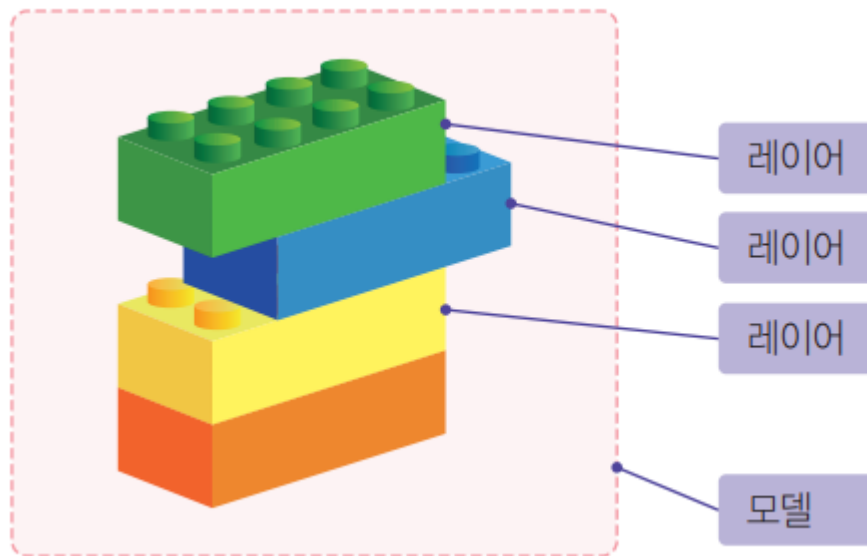
위의 명령이 실행되고 나면 **base**가 **deep** 으로 변경된다. 여기에다가 스파이더와 텐서플로를 설치해보자.

```
(deep) C:\Users\chun> conda install spyder
```

```
(deep) C:\Users\chun> conda install tensorflow
```


Keras

- Keras는 신경망을 레고 조립하듯이 만들 수 있다.



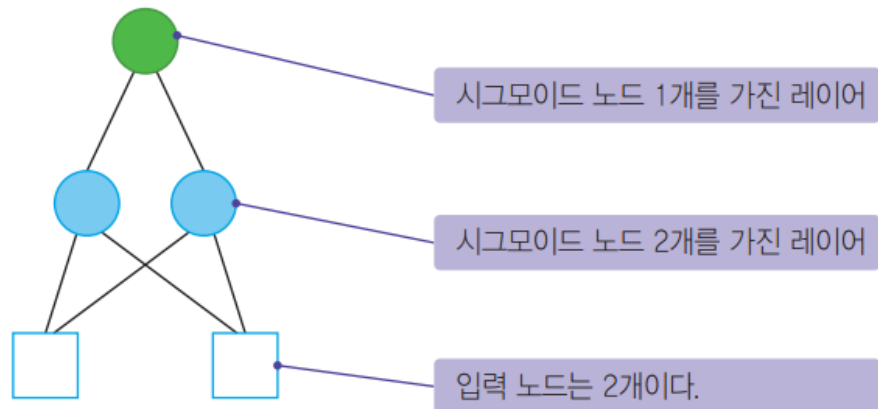
모델 작성

```
from tf.keras.models import Sequential

model = Sequential()

from tf.keras.layers import Dense

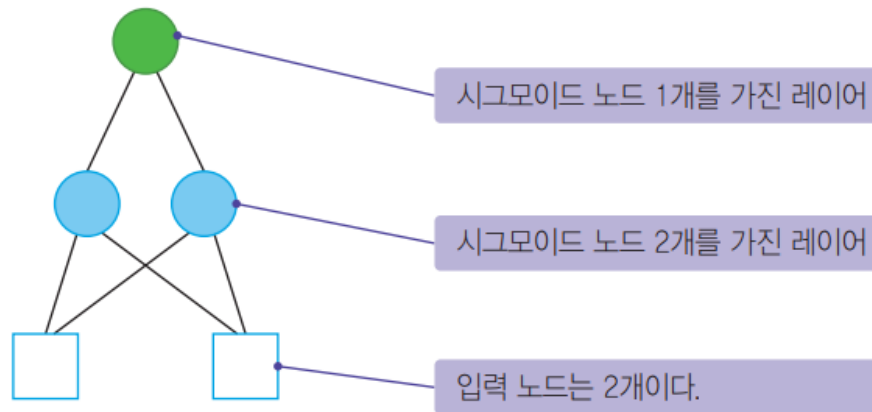
model.add(Dense(units=2, activation='sigmoid', input_dim=2)) # ①
model.add(Dense(units=1, activation='sigmoid'))              # ②
```



모델 작성

```
sgd = tf.keras.optimizers.SGD(lr=0.1)
```

```
model.compile(loss='mean_squared_error', optimizer=sgd)
```




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

model.fit(X, y, batch_size=1, epochs=10000)
```

	x1	x2		y
샘플 #1	0	0	→	0
샘플 #2	0	1		1
샘플 #3	1	0		1
샘플 #4	1	1		0

```
>>> print( model.predict(X) )  
...  
Epoch 10000/10000  
4/4 [=====] - 0s 748us/sample - loss: 9.7796e-04  
[[0.02736092]  
 [0.9704443 ]  
 [0.9701309 ]  
 [0.03734875]]
```

Lab: 논리적인 OR 학습

	x1	x2		y
샘플 #1	0	0		0
샘플 #2	0	1		1
샘플 #3	1	0		1
샘플 #4	1	1		1

...

Epoch 10000/10000

4/4 [=====] - 0s 2ms/sample - loss: 2.1323e-04

[[0.02099779]

[0.9857609]

[0.9858792]

[0.9968916]]

Sol:

```
import tensorflow as tf
import numpy as np

model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Dense(units=2, input_dim=2, activation='sigmoid')) # ①
model.add(tf.keras.layers.Dense(units=1, activation='sigmoid')) # ②

sgd = tf.keras.optimizers.SGD(lr=0.1)
model.compile(loss='mean_squared_error', optimizer=sgd)

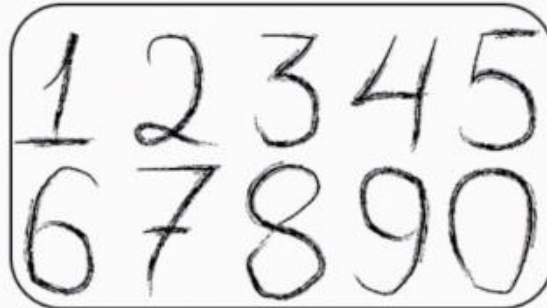
X = np.array([[0, 0],[0, 1],[1, 0],[1, 1]])
y = np.array([[0], [1], [1], [1]])

model.fit(X, y, batch_size=1, epochs=10000)
print( model.predict(X) )
```

MLP를 사용한 MNIST 숫자 인식

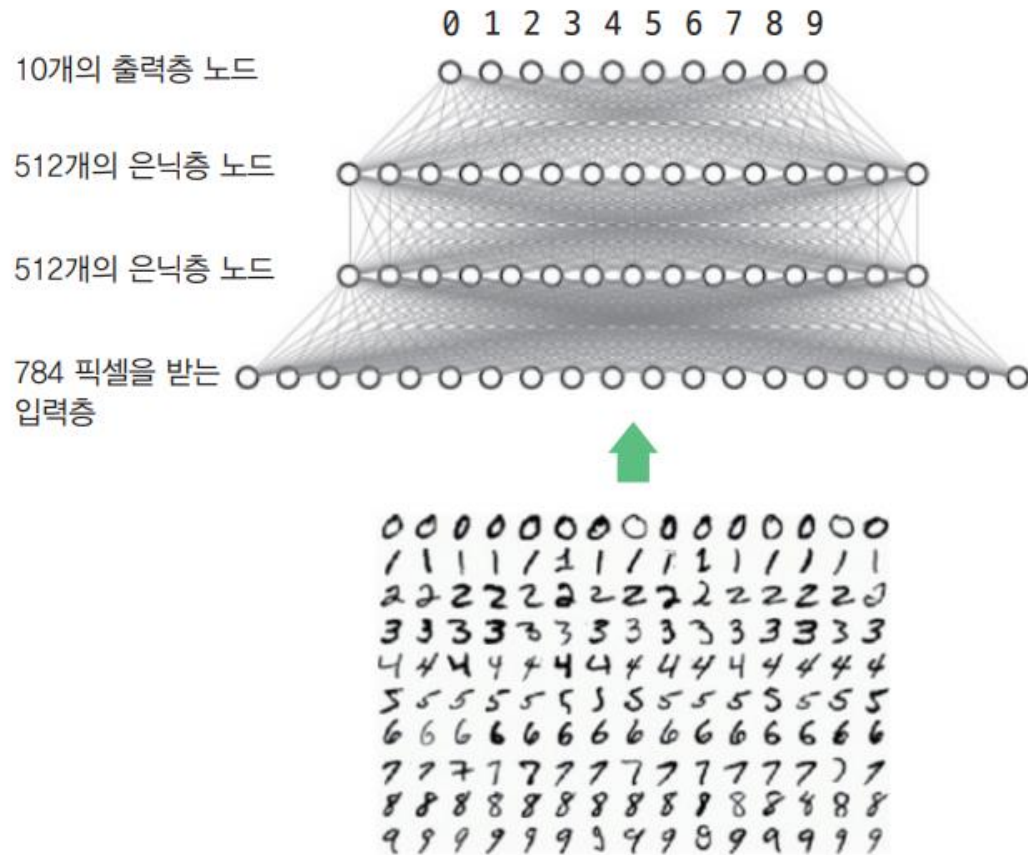
MNIST 데이터셋

* 55,000개의 학습 이미지



10,000개의 테스트 이미지

MLP를 사용한 MNIST 숫자 인식



MLP를 사용한 MNIST 숫자인식

```
import matplotlib.pyplot as plt
import tensorflow as tf

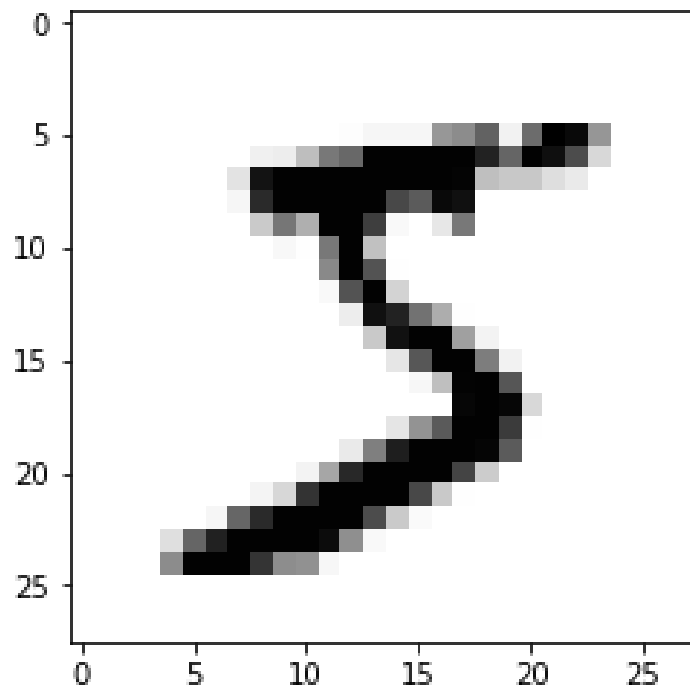
mnist = tf.keras.datasets.mnist

# 훈련 데이터와 테스트 데이터를 가져온다.
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 넘파이를 사용하여 입력을 0.0에서 1.0 사이로 만든다.
x_train, x_test = x_train / 255.0, x_test / 255.0
```

입력 이미지 불러

```
plt.imshow(x_train[0], cmap="Greys");
```



모델 구축하기

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
model.add(tf.keras.layers.Dense(512, activation='relu'))
model.add(tf.keras.layers.Dropout(0.2))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

- 현재 옵티마이저는 “adam”으로 지정되었다. ICLR 2015 학술대회에서 처음으로 발표된 “adam”은 학습 도중에 학습률을 적응적으로 변경시키는 최적화 알고리즘이다. 손실 함수는 'sparse_categorical_crossentropy'로 지정되었다. 이것은 교차 엔트로피 값을 손실 함수로 지정한다.

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

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

```
Epoch 1/5
60000/60000 [=====] - 7s 116us/sample - loss: 0.2205 - acc:
0.9348
Epoch 2/5
60000/60000 [=====] - 7s 110us/sample - loss: 0.0969 - acc:
0.9700
Epoch 3/5
60000/60000 [=====] - 7s 109us/sample - loss: 0.0678 - acc:
0.9785
Epoch 4/5
60000/60000 [=====] - 6s 108us/sample - loss: 0.0529 - acc:
0.9834
Epoch 5/5
60000/60000 [=====] - 7s 108us/sample - loss: 0.0428 - acc:
0.9859
```

타이타닉 생존자 예측하기



Second



Third



Crew

생존자를 예측해봅시다. 어떤 부류의 사람들의 생존률이 높았을까요?
우리는 어떤 속성을 이용하여 이것을 예측할 수 있을까요?



라이브러리 적재

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
```


하스 데이터 다운로드

```
>>> train = pd.read_csv("train.csv", sep=',')
>>> test = pd.read_csv("test.csv", sep=',')
```

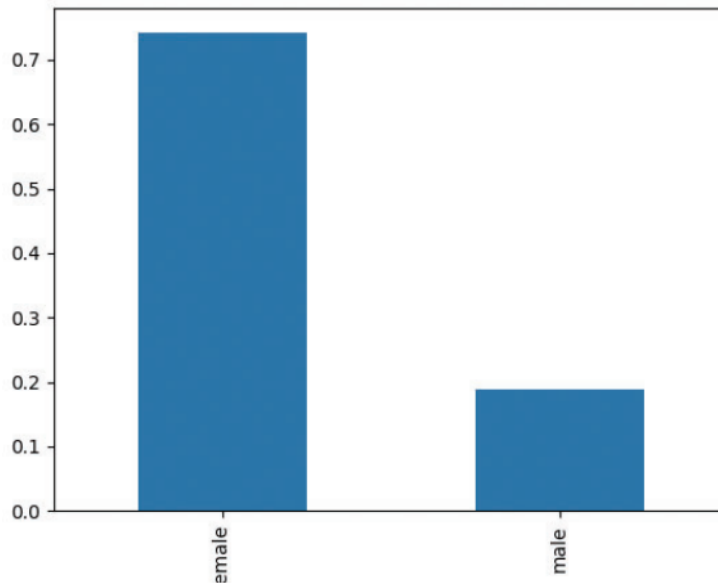
작업을 하기 전에는 항상 데이터가 어떻게 생겼는지 보는 것이 바람직하다.

```
>>> train.head()
```

	PassengerId	Survived	Pclass	...	Fare	Cabin	Embarked
0	1	0	3	...	7.2500	NaN	S
1	2	1	1	...	71.2833	C85	C
2	3	1	3	...	7.9250	NaN	S
3	4	1	1	...	53.1000	C123	S
4	5	0	3	...	8.0500	NaN	S

시각화

```
>>> df = train.groupby('Sex').mean()["Survived"]  
>>> df.plot(kind='bar')  
>>> plt.show()
```



여성의 생존률이 높은
것을 알 수 있네요!



하스 데이터 정제

```
>>> train.drop(['SibSp', 'Parch', 'Ticket', 'Embarked', 'Name',\
                'Cabin', 'PassengerId', 'Fare', 'Age'], inplace=True, axis=1)
```

```
>>> train.head()
   Survived  Pclass    Sex
0         0       3  male
1         1       1 female
2         1       3 female
3         1       1 female
4         0       3  male
```

성별을 숫자로 변환

```
for ix in train.index:  
    if train.loc[ix, 'Sex']=="male":  
        train.loc[ix, 'Sex']=1  
    else:  
        train.loc[ix, 'Sex']=0
```

컴퓨터는 숫자만 처리할 수 있다. 딥러닝은 0부터 1 사이의 실수만 처리 가능

케라스 모델 구축

```
model = tf.keras.models.Sequential()

model.add(tf.keras.layers.Dense(16, activation='relu', input_shape=(2,)))
model.add(tf.keras.layers.Dense(8, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

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

model.fit(train, target, epochs=30, batch_size=1, verbose=1)
```

실행결과

...

Epoch 29/30

891/891 [=====] - 1s 753us/sample - loss: 0.4591 - acc: 0.7677

Epoch 30/30

891/891 [=====] - 1s 753us/sample - loss: 0.4547 - acc: 0.7789

약 78% 정확도

이번 장에서 배운 것

- 기계 학습(machine learning)은 인공지능의 한 분야로, 컴퓨터에 학습 기능을 부여하기 위한 연구 분야이다. 인공지능의 한 분야가 기계 학습이라고 할 수 있고 기계학습 중에서 하나의 알고리즘이 딥러닝이라고 할 수 있다.
- 기계 학습은 “교사”의 존재 여부에 따라 크게 지도 학습과 비지도 학습으로 나뉘어진다. 또 강화학습도 있다. 지도 학습은 "교사"에 의해 주어진 예제(샘플)와 정답(레이블)을 제공받는다. 비지도 학습은 외부에서 정답(레이블)이 주어지지 않고 학습 알고리즘이 스스로 입력에서 어떤 구조를 발견하는 학습이다.
- 지도 학습은 크게 회귀와 분류로 나눌 수 있다. 회귀는 주어진 입력-출력 쌍을 학습한 후에 새로운 입력값이 들어왔을 때, 합리적인 출력값을 예측하는 것이다. 분류(classification): 입력을 두 개 이상의 유형으로 분할하는 것이다. 학습 시에는 교사가 있어서 입력의 올바른 유형을 알려준다.



Q & A

