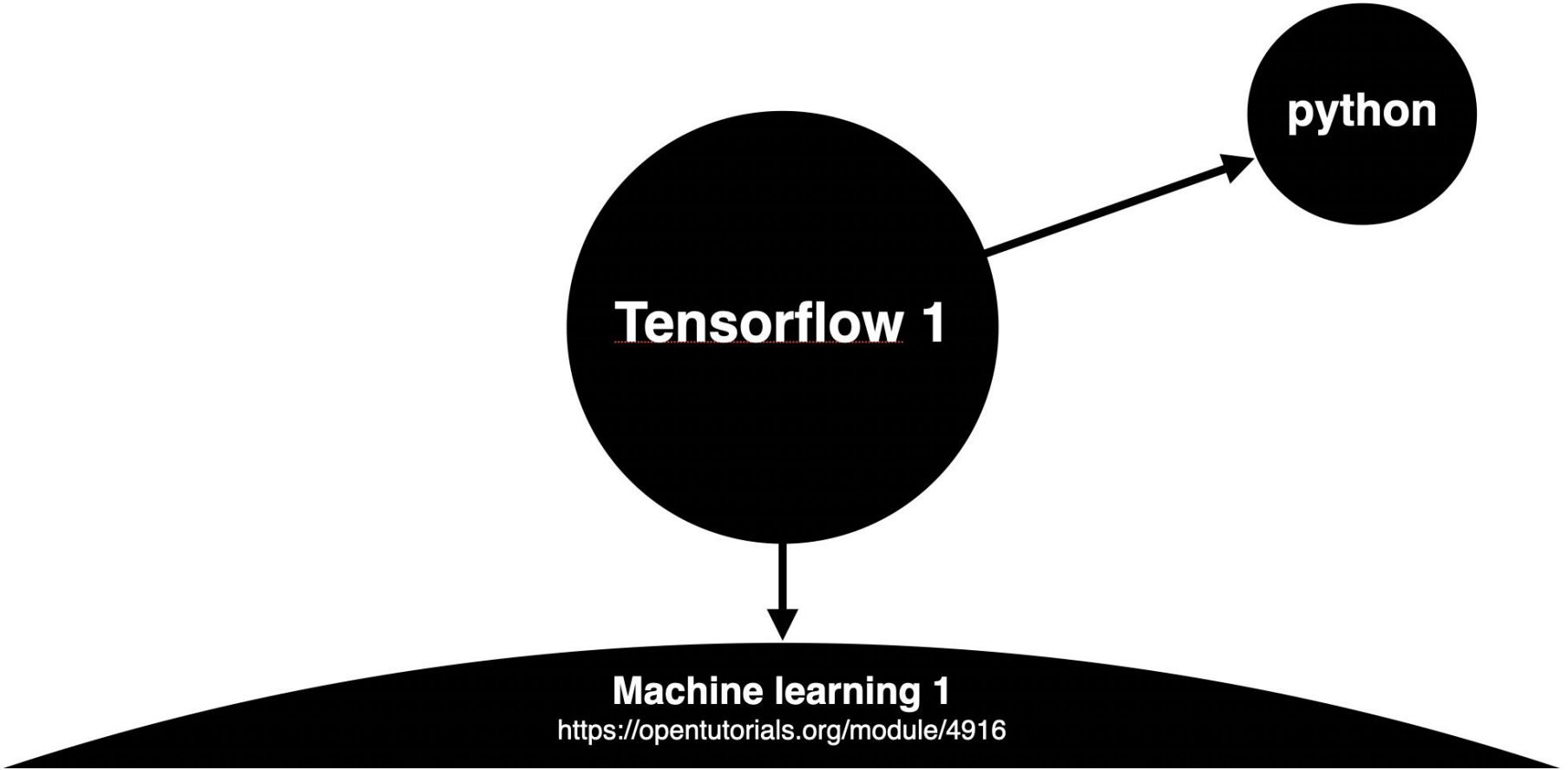
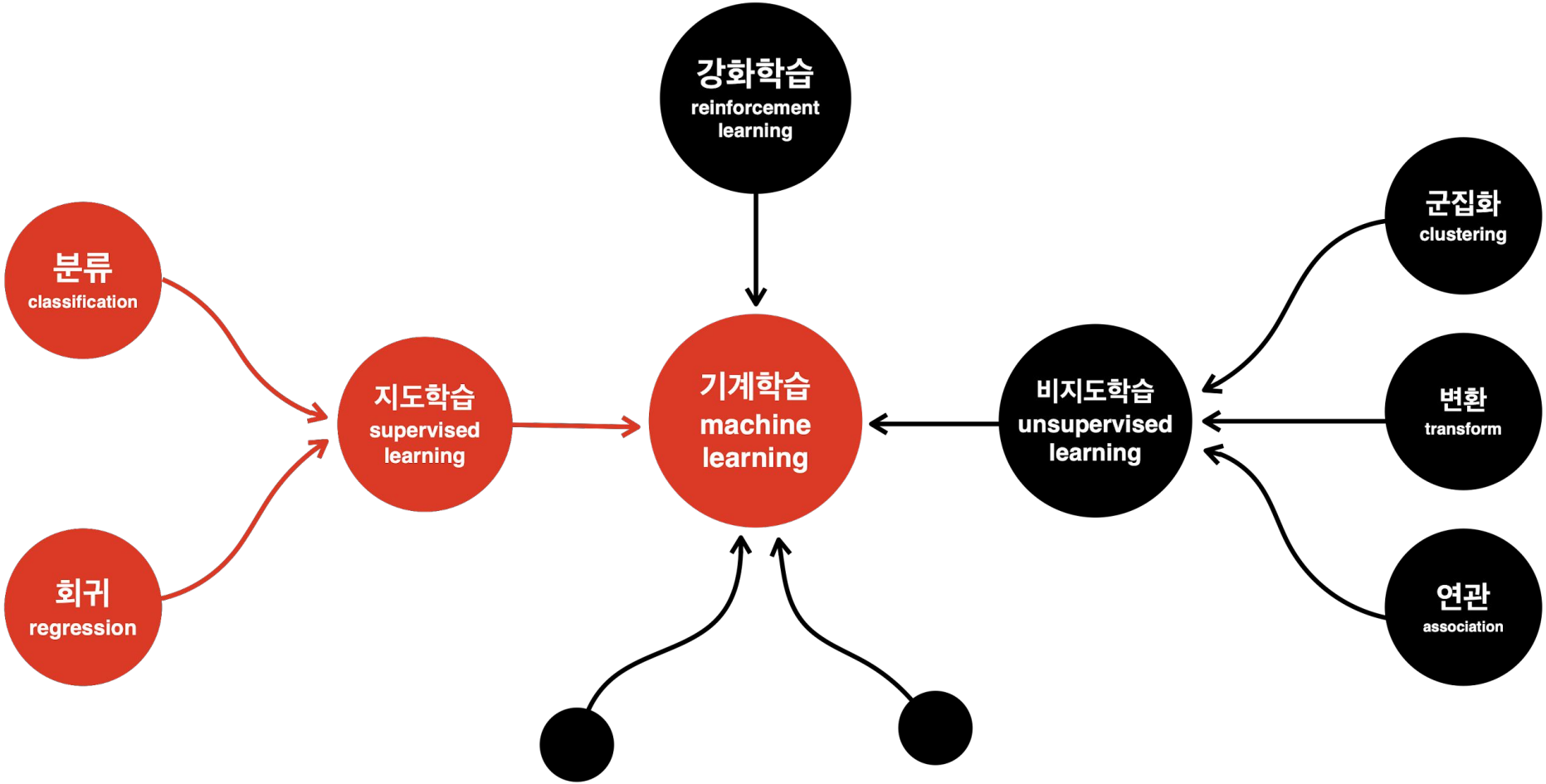

python
Tensorflow 1
Machine learning 1
https://opentutorials.org/module/4916


강화학습
reinforcement learning
분류
classification
회귀
regression
지도학습
supervised learning
기계학습
machine learning
비지도학습
unsupervised learning
군집화
clustering
변환
transform
연관
association

회귀
regression

1
2

분류
classification

양성
음성

Decision Tree

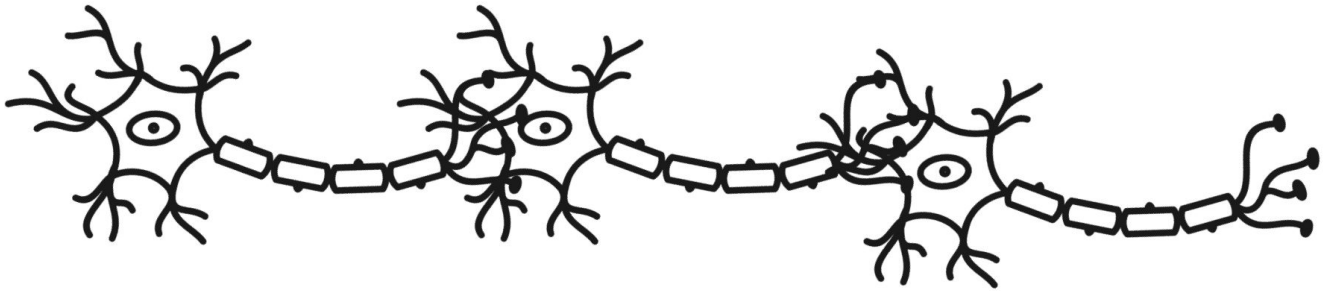
Random Forest

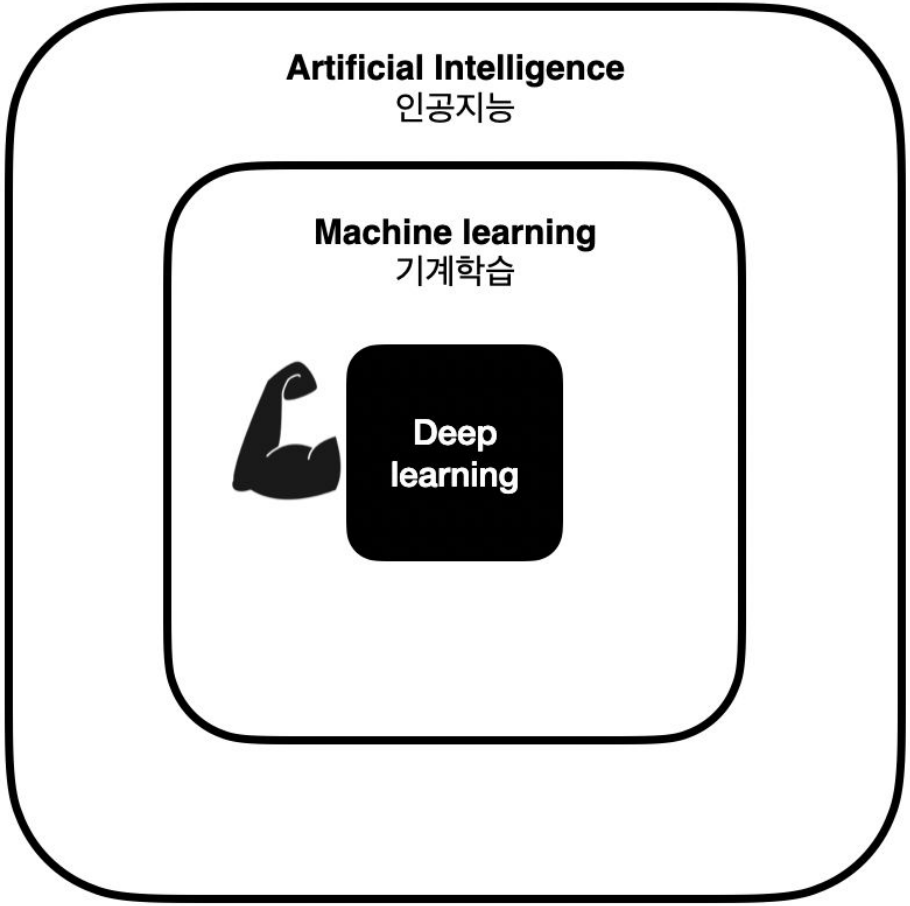
KNN

SVM

Neural Network

Neural Network = 인공 신경망 = Deep learning




Artificial Intelligence
인공지능
Machine learning
기계학습
Deep
learning

Deep learning

라이브러리

Decision Tree

Random Forest

KNN

SVM

Neural Network
Deep Learning

알고리듬

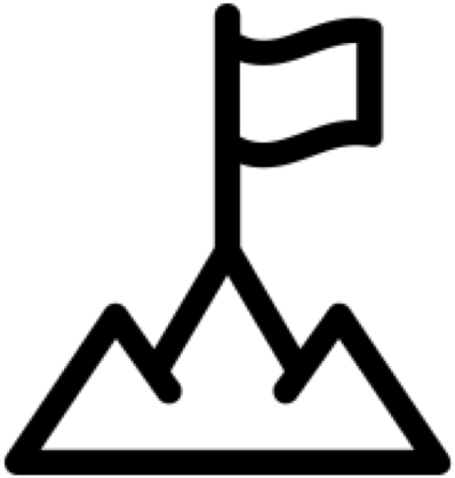
회귀

분류

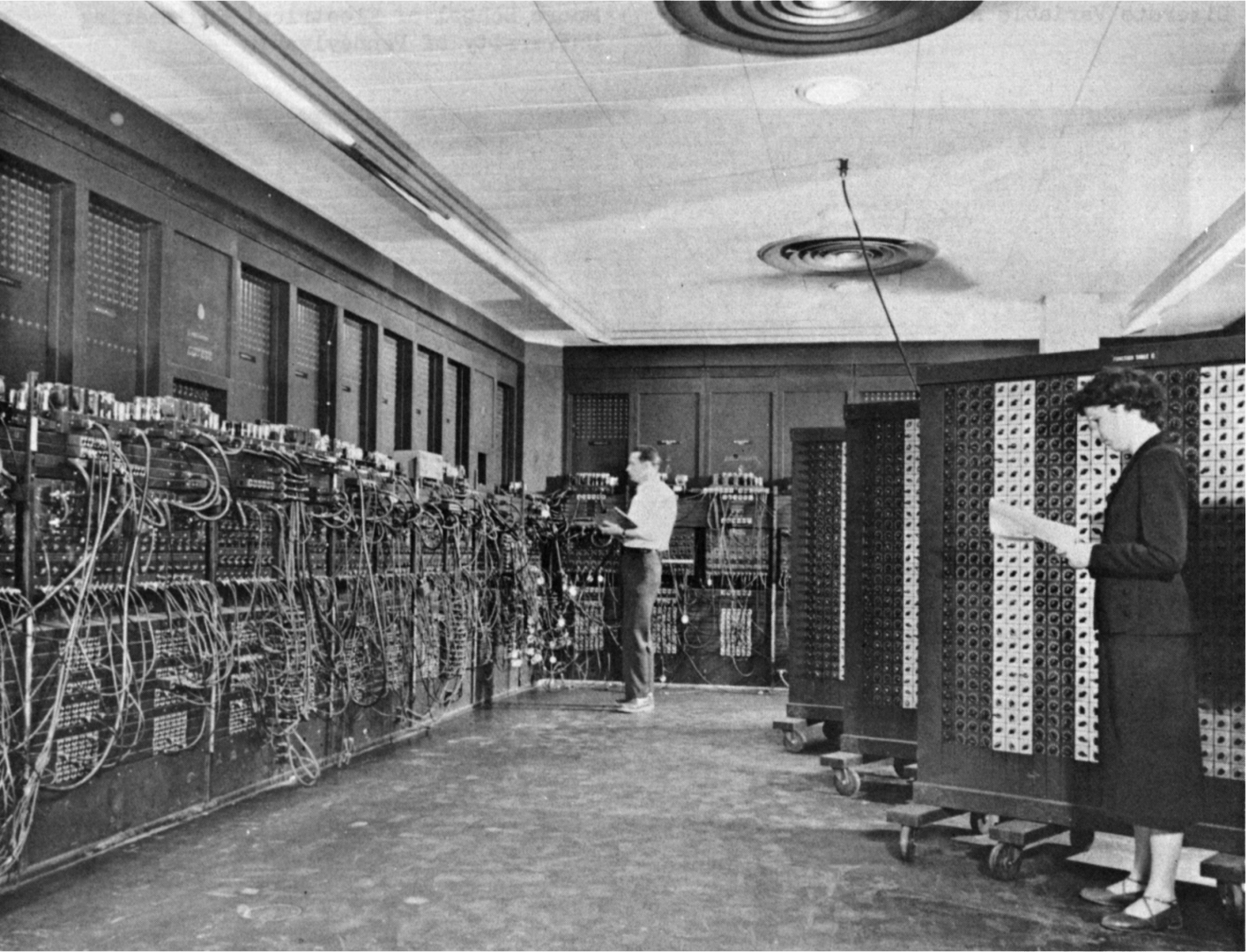
문제
지도학습

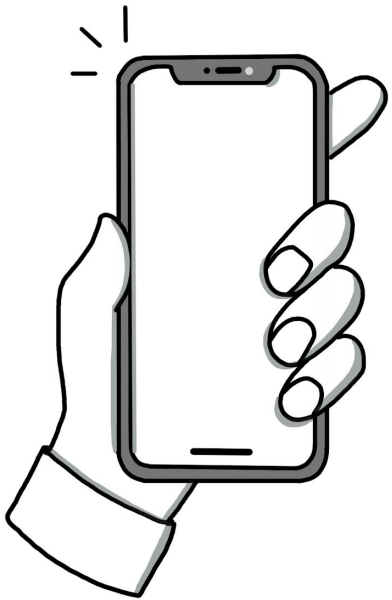
machine learning

AI



목표와 전략





오늘 우리는 다음과 같은 방법으로

머신러닝 코드를 배울 것입니다.

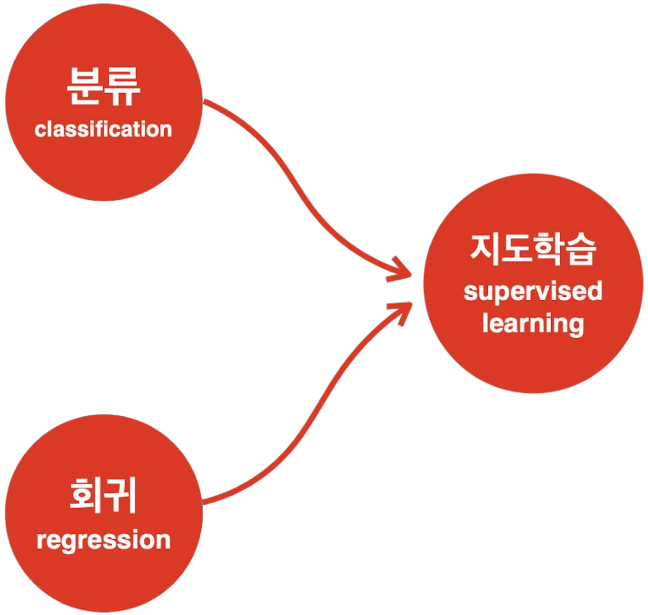
1. 간단한 코드를 구경하고
2. 코드가 동작하는 것을 경험하고
3. 해당 코드를 어떻게 이용하면 될지 추측합니다.

```
import tensorflow as tf

# 모델 구조 생성
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')

# 모델 학습
model.fit(x_train, y_train, epochs=1000, verbose=0)
model.fit(x_train, y_train, epochs=10)
```

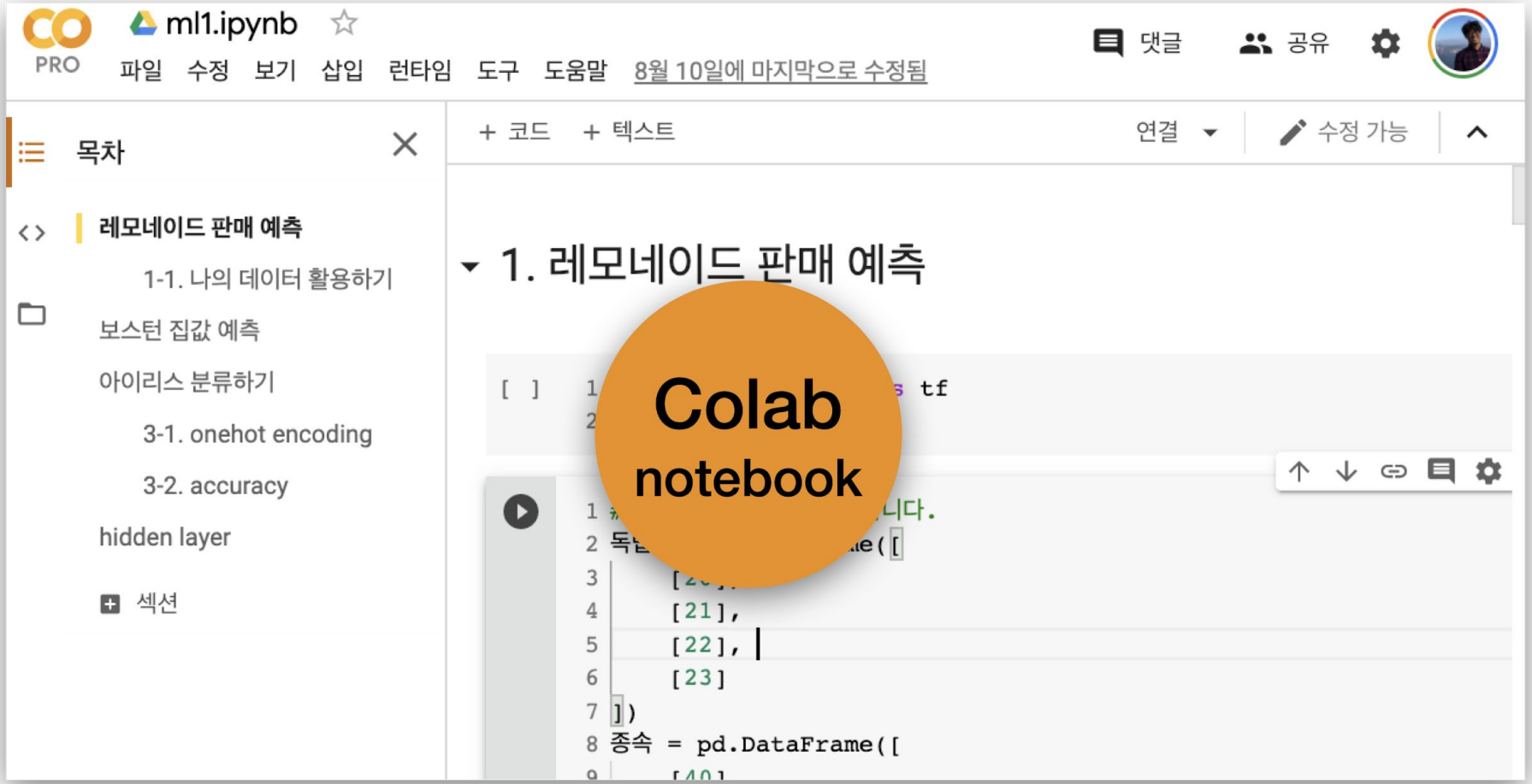
```
Epoch 1/10
1/1 [==============================] - 0s 2ms/step - loss: 0.0111
Epoch 2/10
1/1 [==============================] - 0s 2ms/step - loss: 0.0111
Epoch 3/10
1/1 [==============================] - 0s 2ms/step - loss: 0.0110
Epoch 4/10
1/1 [==============================] - 0s 3ms/step - loss: 0.0110
Epoch 5/10
1/1 [==============================] - 0s 891us/step - loss: 0.0110
Epoch 6/10
1/1 [==============================] - 0s 2ms/step - loss: 0.0110
```

```
# 모델 구조 생성
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation="softmax")(X)
model = tf.keras.models.Model(X, Y)
model.compile(
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)

# 모델 학습
model.fit(x_train, y_train, epochs=100, verbose=0)
model.fit(x_train, y_train, epochs=10)
```

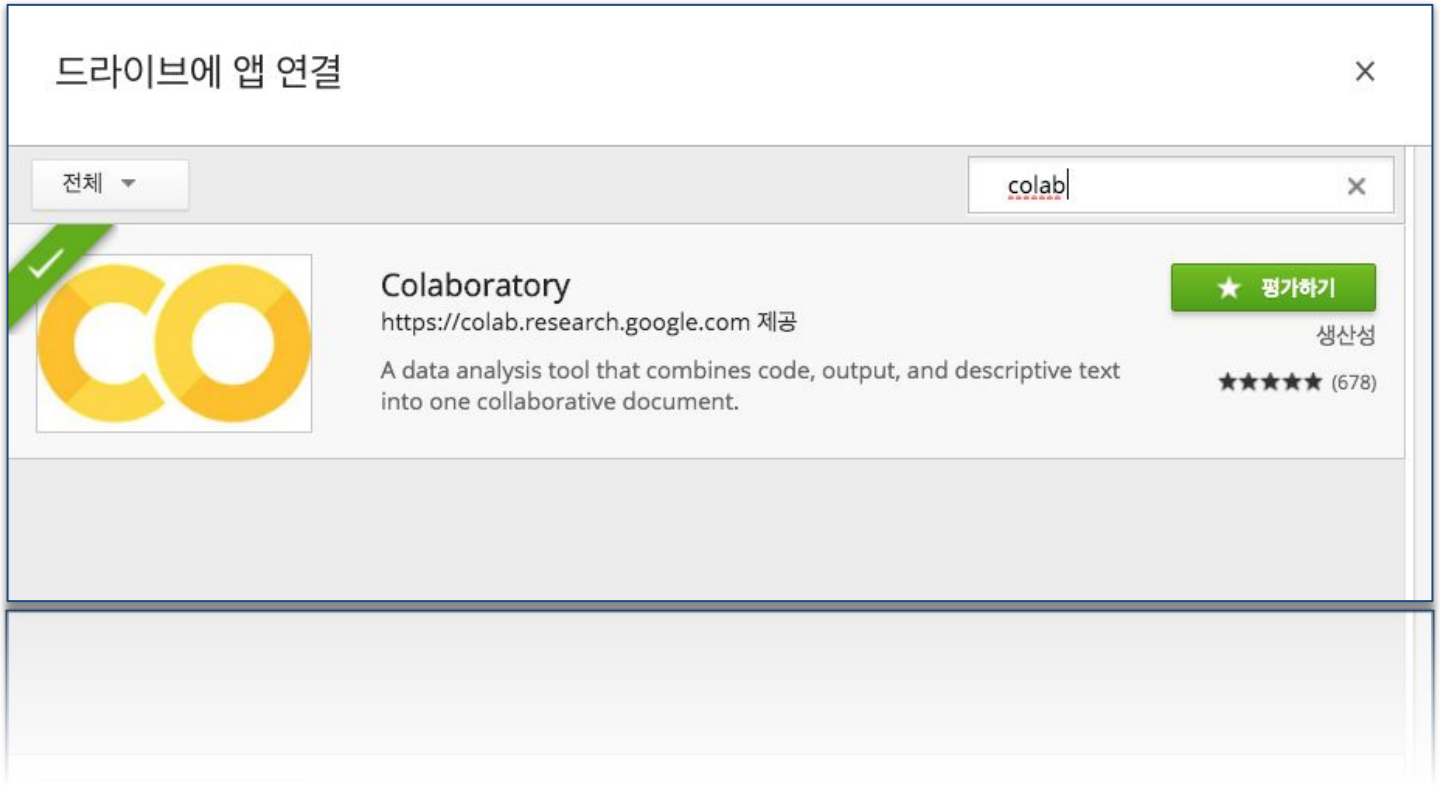
```
Epoch 1/10
5/5 [==============================] - 0s 3ms/step - loss: 0.7484 - accuracy: 0.6867
Epoch 2/10
5/5 [==============================] - 0s 2ms/step - loss: 0.7448 - accuracy: 0.6867
Epoch 3/10
5/5 [==============================] - 0s 3ms/step - loss: 0.7405 - accuracy: 0.6933
Epoch 4/10
5/5 [==============================] - 0s 3ms/step - loss: 0.7360 - accuracy: 0.7000
Epoch 5/10
5/5 [==============================] - 0s 3ms/step - loss: 0.7318 - accuracy: 0.6933
Epoch 6/10
5/5 [==============================] - 0s 3ms/step - loss: 0.7289 - accuracy: 0.6933
```


분류
classification
회귀
regression
지도학습
supervised
learning

실습환경


ml1.ipynb
PRO
파일 수정 보기 삽입 런타임 도구 도움말 8월 10일에 마지막으로 수정됨
댓글
공유
목차
+ 코드 + 텍스트
연결
수정 가능
레모네이드 판매 예측
1-1. 나의 데이터 활용하기
보스턴 집값 예측
아이리스 분류하기
3-1. onehot encoding
3-2. accuracy
hidden layer
섹션
1. 레모네이드 판매 예측
Colab
notebook
[ ]
tf
[21],
[22],
[23]
])
종속 = pd.DataFrame([

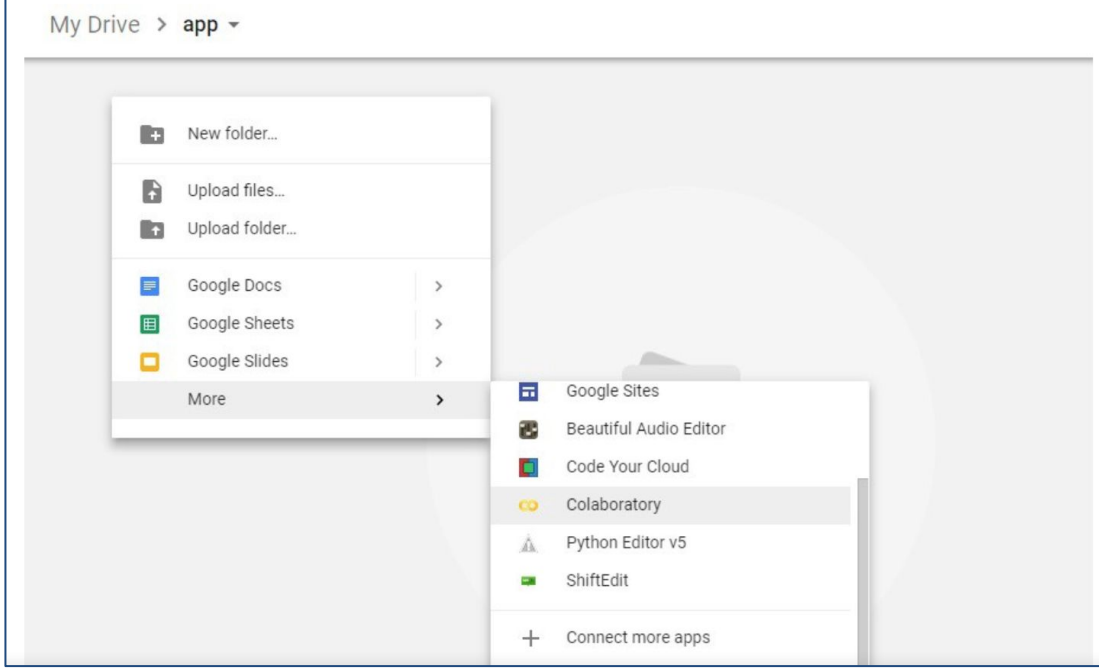
Google Colaboratory를 사용합니다.



새 Colaboratory 문서 생성

Creating New Colab Notebook

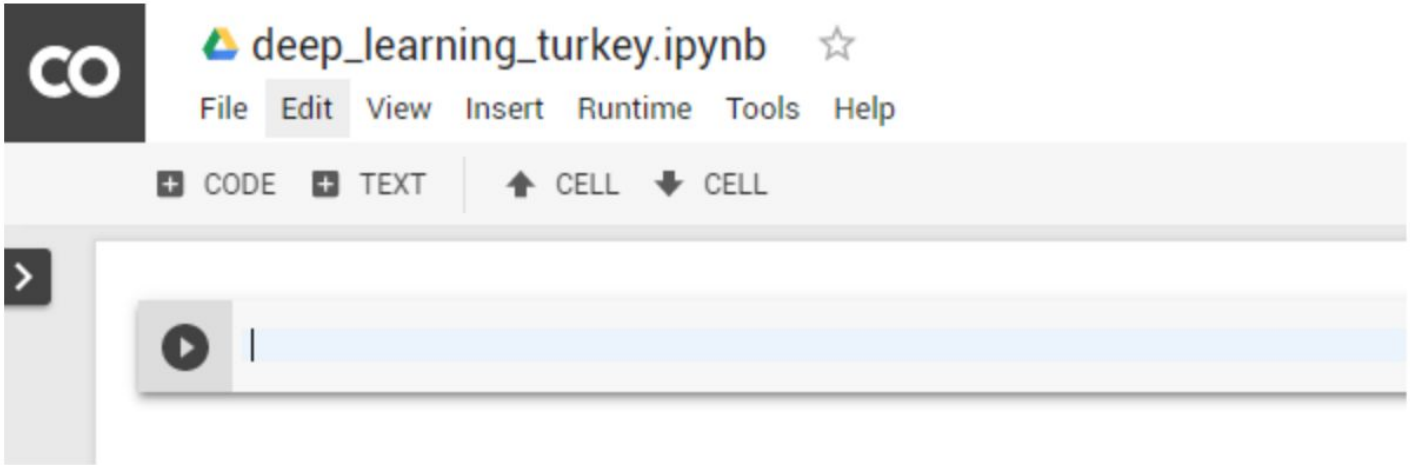
Create a new notebook via **Right click > More > Colaboratory**



Google Colab 시작

Running Basic Python Codes with Google Colab

Now we can start using **Google Colab**.



X = 1
X = 2

변수

variable

X = 1

X = 2

```
# 데이터 불러오기
파일경로 = 'lemonade.csv'
데이터 = pd.read_csv(파일경로)

파일경로 = 'boston.csv'
데이터 = pd.read_csv(파일경로)

파일경로 = 'iris.csv'
데이터 = pd.read_csv(파일경로)

```

날짜	요일	온도	판매량
2020.1.3	금	20	40
2020.1.4	토	21	42
2020.1.5	일	22	44

온도 = 20

온도 = 21

온도 = 22

날짜	요일	온도	판매량
2020.1.3	금	20	40
2020.1.4	토	21	42
2020.1.5	일	22	44

```
import pandas as pd

```

```
# 데이터 불러오기
파일경로 = 'lemonade.csv'
데이터 = pd.read_csv(파일경로)

# 독립변수와, 종속변수의 분리
독립 = 데이터[['온도']]
종속 = 데이터[['판매량']]

# 데이터 모양 확인
print(독립.shape, 종속.shape)

```

레모네이드 판매 예측

주석과 코드


지도학습
supervised
learning

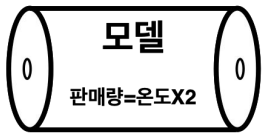
독립변수 (원인)	종속변수 (결과)
온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

3.데이터로 모델을 학습(FIT)합니다.

2. 모델의 구조를 만듭니다

15도
원인



30개
결과

4. 모델을 이용합니다

1.과거의 데이터를 준비합니다.

2. 모델의 구조를 만듭니다

3.데이터로 모델을 학습(FIT)합니다.

4. 모델을 이용합니다

1.과거의 데이터를 준비합니다.

```
레모네이드 = pd.read_csv('lemonade.csv')
독립 = 레모네이드[['온도']]
종속 = 레모네이드[['판매량']]

print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

4. 모델을 이용합니다

```
print("Predictions: ", model.predict([[15]]))
```

독립변수 **종속변수**

원인 결과

온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

```
레모네이드 = pd.read_csv('lemonade.csv')
독립 = 레모네이드[['온도']]
종속 = 레모네이드[['판매량']]

print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다


```
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

독립변수 종속변수
원인 결과

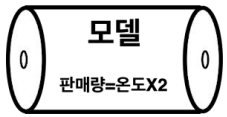
온도	판매량
20	40
21	42
22	44
23	46

3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

```
print("Predictions: ", model.predict([[15]]))
```

15도
원인



30개
결과

4. 모델을 이용합니다

1.과거의 데이터를 준비합니다.

```
레모네이드 = pd.read_csv('lemonade.csv')
독립 = 레모네이드[['온도']]
종속 = 레모네이드[['판매량']]

print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

4. 모델을 이용합니다

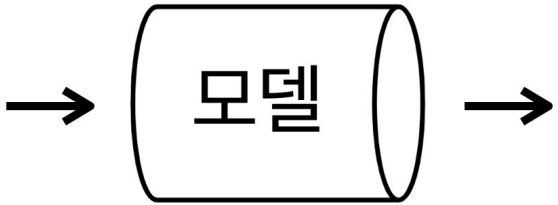
```
print("Predictions: ", model.predict([[15]]))
```

model.fit(독립, 종속, epochs=10)

```
Epoch 1/10
1/1 [==============================] - 0s 1ms/step - loss: 2548.6941
Epoch 2/10
1/1 [==============================] - 0s 1ms/step - loss: 2541.5061
Epoch 3/10
1/1 [==============================] - 0s 2ms/step - loss: 2536.3013
Epoch 4/10
1/1 [==============================] - 0s 2ms/step - loss: 2531.9492
Epoch 5/10
1/1 [==============================] - 0s 939us/step - loss: 2528.0898
Epoch 6/10
1/1 [==============================] - 0s 1ms/step - loss: 2524.5566
Epoch 7/10
1/1 [==============================] - 0s 1ms/step - loss: 2521.2566
Epoch 8/10
1/1 [==============================] - 0s 802us/step - loss: 2518.1318
Epoch 9/10
1/1 [==============================] - 0s 1ms/step - loss: 2515.1431
Epoch 10/10
1/1 [==============================] - 0s 888us/step - loss: 2512.2637
```

독립 종속

온도	판매량
20	40
21	42
22	44
23	46



예측
35
40
45
50

판매량
40
42
44
46

예측
35
40
45
50

(예측 - 결과)^2

Error^2
25
4
1
16

LOSS

평균: 11.5

보스턴 집값 예측

수식과 퍼셉트론

$$y = 2x$$

온도	판매량
20	40
21	42
22	44
23	46

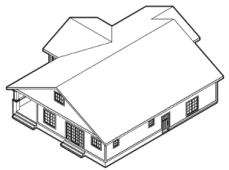
$$y = -0.09832304x_1 + 0.088935934x_2 + -0.060747888x_3 + 4.444701x_4 + 0.95923173x_5 + 3.4496038x_6 + 0.03198209x_7 + -0.8730943x_8 + 0.17615545x_9 + -0.00806713x_{10} + 0.14275435x_{11} + 0.017301293x_{12} + -0.6197083x_{13} + 2.0836691856384277$$

Boston Housing Price

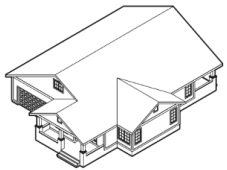
1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
													집값
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
1.23247	0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21	396.9	18.72	15.2
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15
8.98296	0	18.1	1	0.77	6.212	97.4	2.1222	24	666	20.2	377.73	17.6	17.8

median value

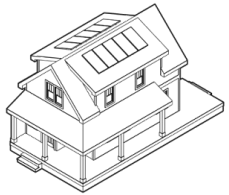
$10000



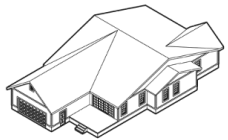
$20000



$40000



$30000



$60000

Boston Housing Price

$$
\begin{aligned}
19.13 = {} & -0.09832304 \times 1.23247 \\
& + 0.088935934 \times 0 \\
& + -0.060747888 \times 8.14 \\
& + 4.444701 \times 0 \\
& + 0.95923173 \times 0.538 \\
& + 3.4496038 \times 6.142 \\
& + 0.03198209 \times 91.7 \\
& + -0.8730943 \times 3.9769 \\
& + 0.17615545 \times 4 \\
& + -0.00806713 \times 307 \\
& + 0.14275435 \times 21 \\
& + 0.017301293 \times 396.9 \\
& + -0.6197083 \times 18.72 \\
& + 2.083669185638427 7
\end{aligned}
$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
범죄율			강변		평균 방 수	노후주택 비율			재산세 세율	학생/교사 비율		하위계층 비율	집값
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.9	5.33	36.2
0.02985	0	2.18	0	0.458	6.43	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.6	12.43	22.9
0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.9	19.15	27.1
1.23247	0	8.14	0	0.538	6.142	91.7	3.9769	4	307	21	396.9	18.72	15.2
0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.1	18.9
0.22489	12.5	7.87	0	0.524	6.377	94.3	6.3467	5	311	15.2	392.52	20.45	15
8.98296	0	18.1	1	0.77	6.212	97.4	2.1222	24	666	20.2	377.73	17.6	17.8

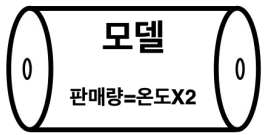
독립변수 (원인)	종속변수 (결과)
온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

3.데이터로 모델을 학습(FIT)합니다.

2. 모델의 구조를 만듭니다

15도
원인



30개
결과

4. 모델을 이용합니다

1.과거의 데이터를 준비합니다.

```
보스턴 = pd.read_csv('boston.csv')
독립 = 보스턴[['crim', 'zn', 'indus', 'chas',
              'nox', 'rm', 'age', 'dis',
              'rad', 'tax', 'ptratio', 'b',
              'lstat']]
종속 = 보스턴[['medv']]
print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[13])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

3.데이터로 모델을 학습(FIT)합니다.

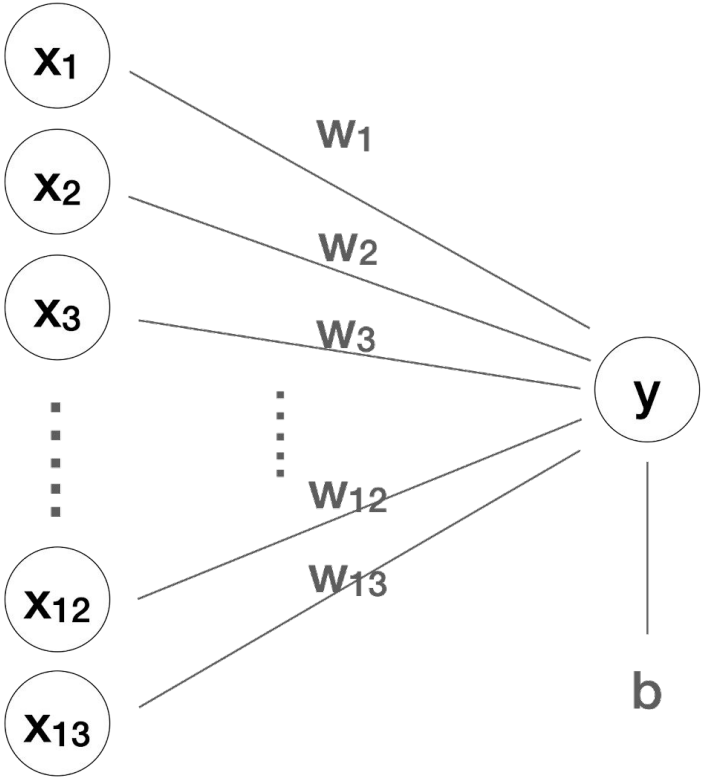
```
model.fit(독립, 종속, epochs=1000)
```

4. 모델을 이용합니다

```
print("Predictions: ", model.predict(독립[0:5]))
```

```
X = tf.keras.layers.Input(shape=[13])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4

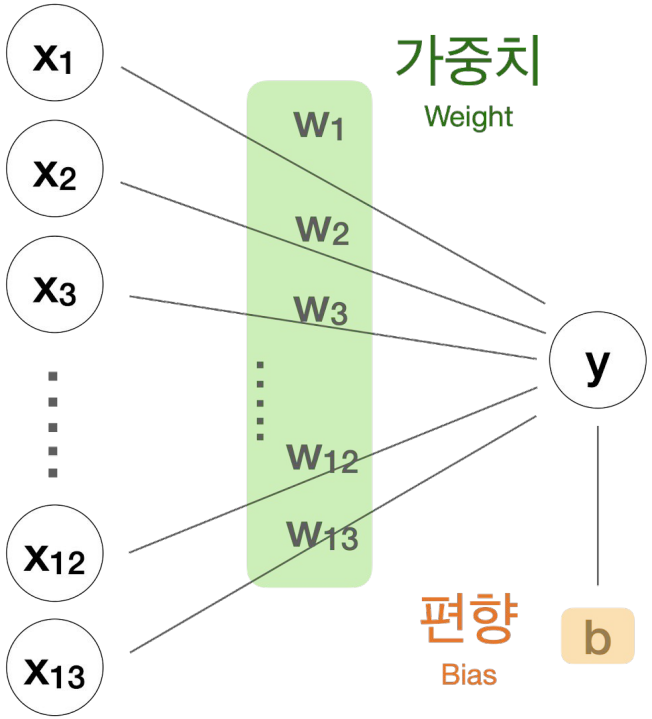


$$y = w_1x_1 + w_2x_2 + \cdots\cdots + w_{13}x_{13} + b$$

$$y = w_1x_1 + w_2x_2 + \cdots\cdots + w_{13}x_{13} + b$$

퍼셉트론

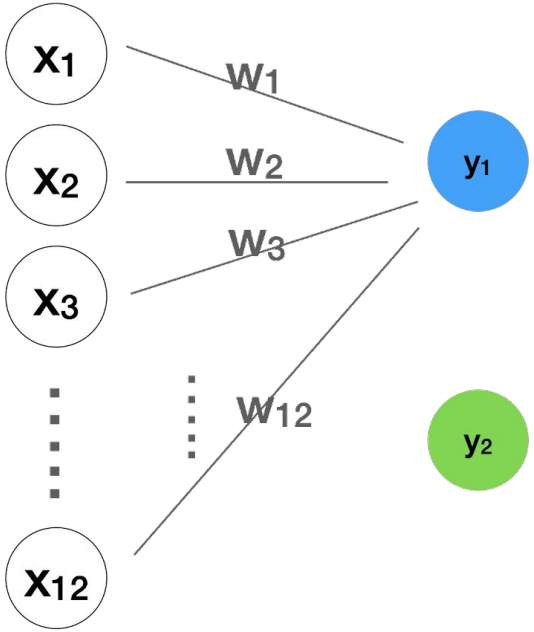
Perceptron



```
X = tf.keras.layers.Input(shape=[12])
Y = tf.keras.layers.Dense(2)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

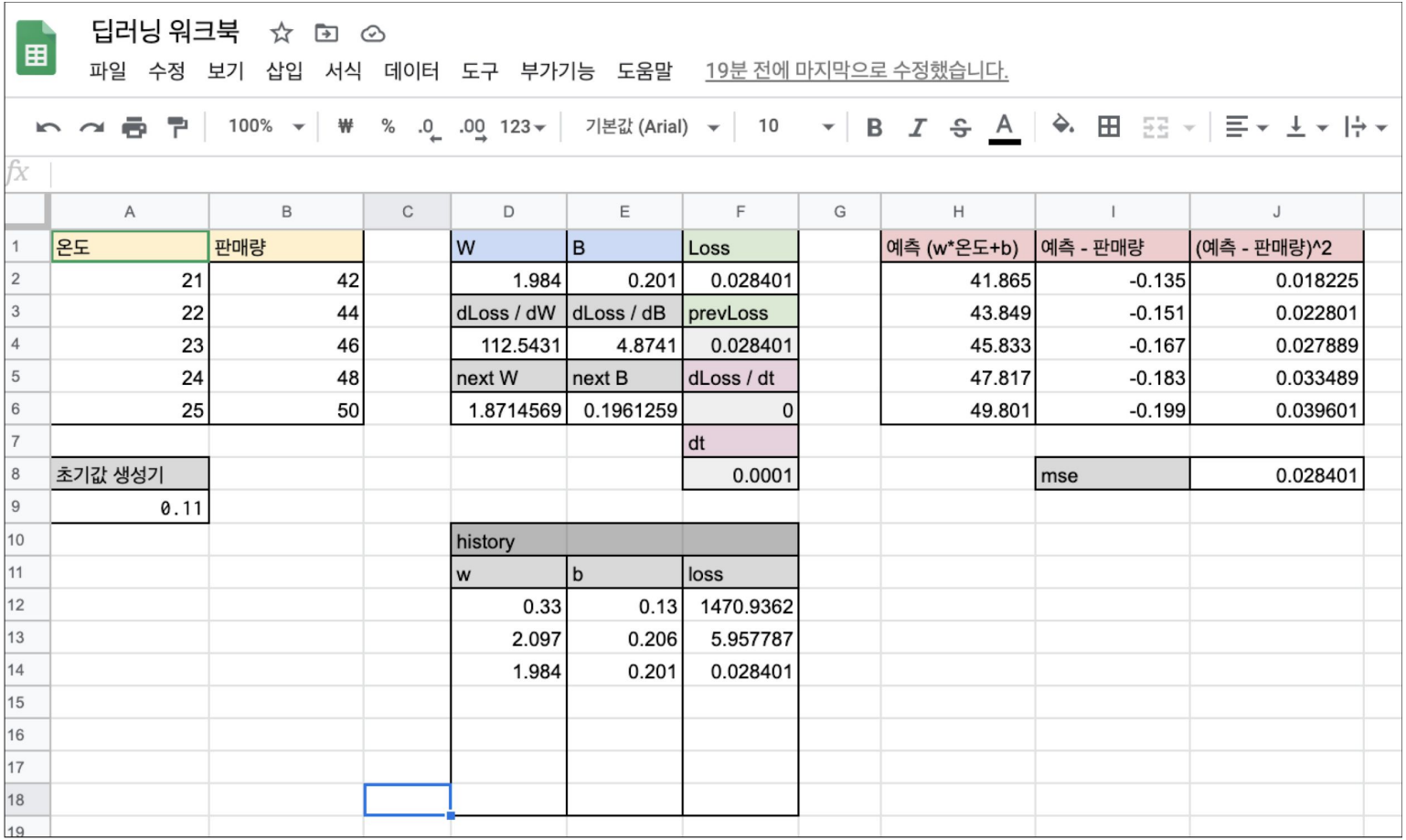
1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4

$$y_1 = w_1x_1 + w_2x_2 + \cdots\cdots + w_{12}x_{12} + b$$



$$y_2 = w_1x_1 + w_2x_2 + \cdots\cdots + w_{12}x_{12} + b$$

학습의 실제



	온도	판매량
	21	42
	22	44
	23	46
	24	48
	25	50
	초기값 생성기	
	0.11	

W	B	Loss
1.984	0.201	0.028401
dLoss / dW	dLoss / dB	prevLoss
112.5431	4.8741	0.028401
next W	next B	dLoss / dt
1.8714569	0.1961259	0
		dt
		0.0001

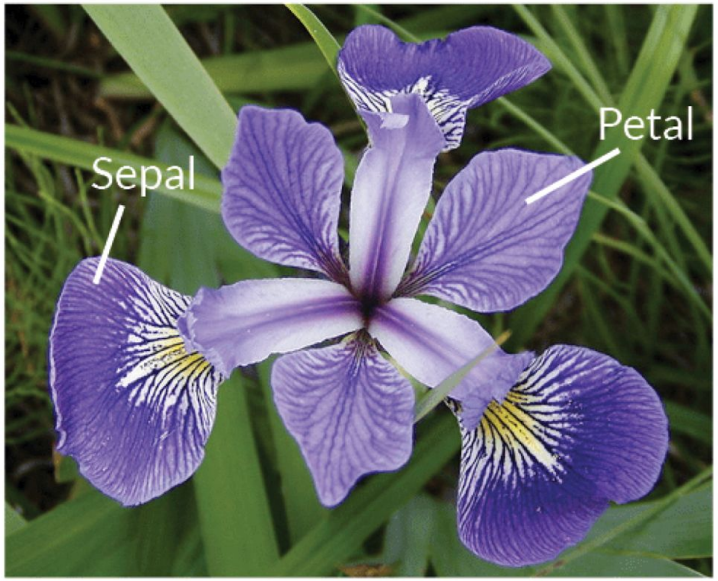
history		
w	b	loss
0.33	0.13	1470.9362
2.097	0.206	5.957787
1.984	0.201	0.028401

예측 (w*온도+b)	예측 - 판매량	(예측 - 판매량)^2
41.865	-0.135	0.018225
43.849	-0.151	0.022801
45.833	-0.167	0.027889
47.817	-0.183	0.033489
49.801	-0.199	0.039601
	mse	0.028401

https://bit.ly/2DEBlPd

아이리스 분류하기

분류는 확률 예측



Iris Versicolor

Iris Setosa

Iris Virginica

https://www.datacamp.com/community/tutorials/machine-learning-in-r

 **blackdew** 칼럼명 Latest commit `042fcba` 4 days ago **History**

1 contributor

Executable File | 151 lines (151 sloc) | 3.77 KB Raw Blame

Search this file...

	꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종
2	5.1	3.5	1.4	0.2	setosa
3	4.9	3.0	1.4	0.2	setosa
4	4.7	3.2	1.3	0.2	setosa
5	4.6	3.1	1.5	0.2	setosa
6	5.0	3.6	1.4	0.2	setosa
7	5.4	3.9	1.7	0.4	setosa
8	4.6	3.4	1.4	0.3	setosa
9	5.0	3.4	1.5	0.2	setosa

온도	판매량
20	40
21	42
22	44
23	46

양적 → **회귀**
regression

공부시간	시험결과
20	불합격
21	불합격
22	합격
23	합격

범주형 → **분류**
classification

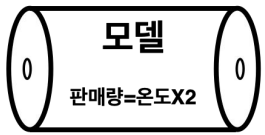
독립변수 (원인)	종속변수 (결과)
온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

3.데이터로 모델을 학습(FIT)합니다.

2. 모델의 구조를 만듭니다

15도
원인



30개
결과

4. 모델을 이용합니다

1.과거의 데이터를 준비합니다.

```
아이리스 = pd.read_csv('iris.csv')
아이리스 = pd.get_dummies(아이리스)

독립 = 아이리스[['꽃잎길이', '꽃입폭',
                 '꽃받침길이', '꽃받침폭']]
종속 = 아이리스[['품종']]
print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

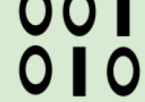
```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy')
```

3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

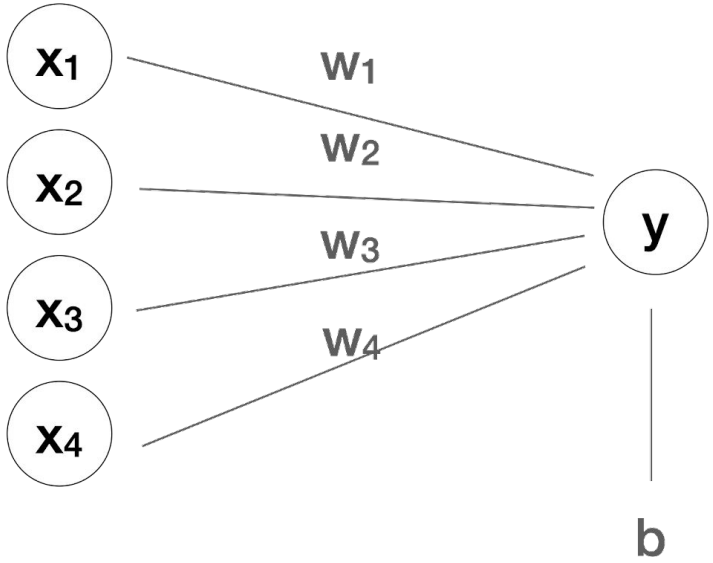
4. 모델을 이용합니다

```
print("Predictions: ", model.predict(독립[0:5]))
```



원핫인코딩

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	versicolor
4.7	3.2	1.3	0.2	virginica
4.6	3.1	1.5	0.2	setosa



$$y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b$$

원핫인코딩 onehot-encoding

품종
setosa
virginica
versicolor
setosa
versicolor

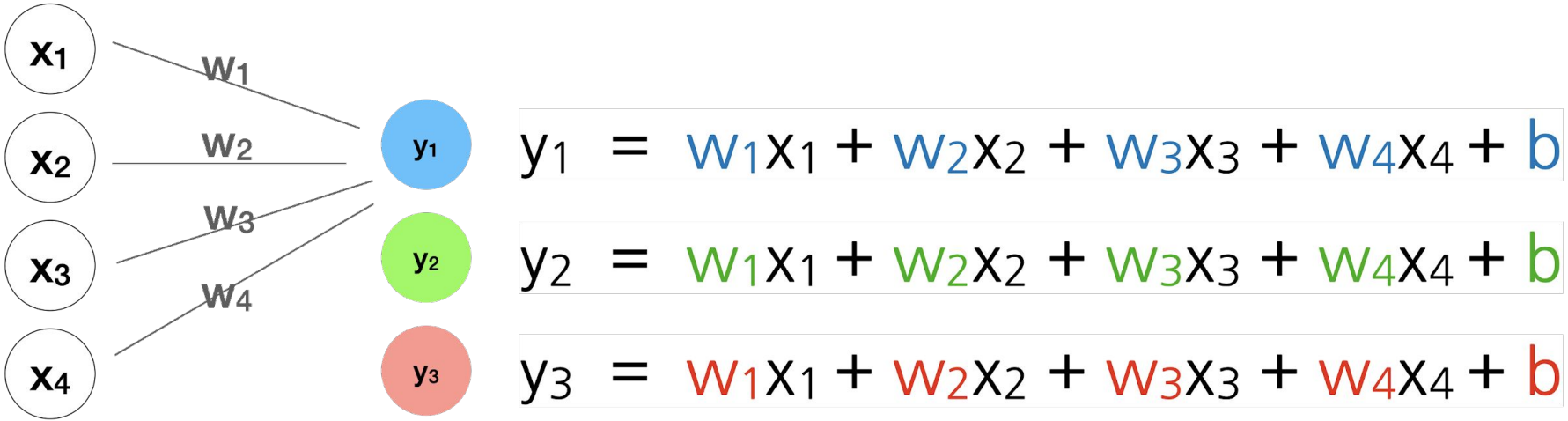
setosa	virginica	versicolor
1	0	0
0	**1**	0
0	0	**1**
1	0	0
0	0	**1**

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	versicolor
4.7	3.2	1.3	0.2	virginica
4.6	3.1	1.5	0.2	setosa

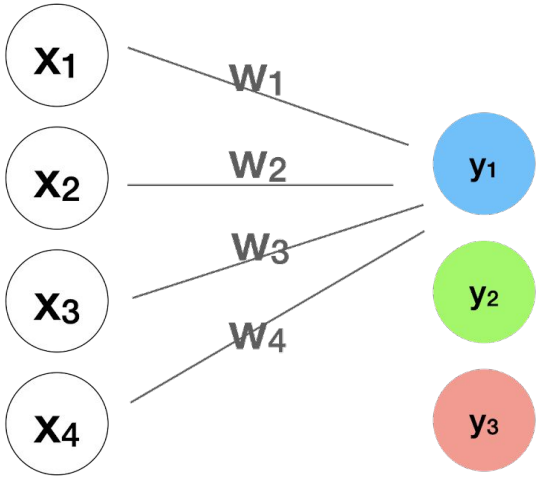
```
# 학습할 데이터를 준비합니다.
아이리스 = pd.read_csv('iris.csv')

# 원핫인코딩
아이리스 = pd.get_dummies(아이리스)
```

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0	1	0
4.7	3.2	1.3	0.2	0	0	1
4.6	3.1	1.5	0.2	1	0	0

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0	1	0
4.7	3.2	1.3	0.2	0	0	1
4.6	3.1	1.5	0.2	1	0	0



꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0	1	0
4.7	3.2	1.3	0.2	0	0	1
4.6	3.1	1.5	0.2	1	0	0

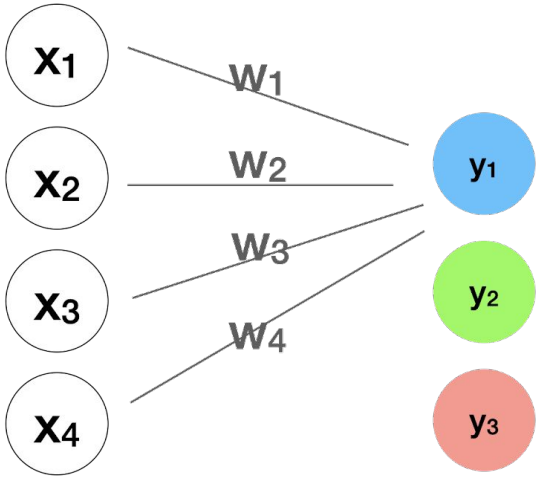


2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy')
```

softmax & crossentropy

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0	1	0
4.7	3.2	1.3	0.2	0	0	1
4.6	3.1	1.5	0.2	1	0	0



2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy')
```

분류 예측

30%

99%

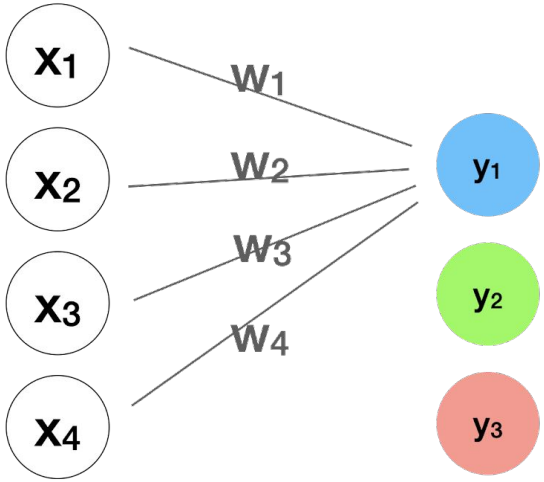
50%

0% ~ 100%

Sigmoid

Softmax

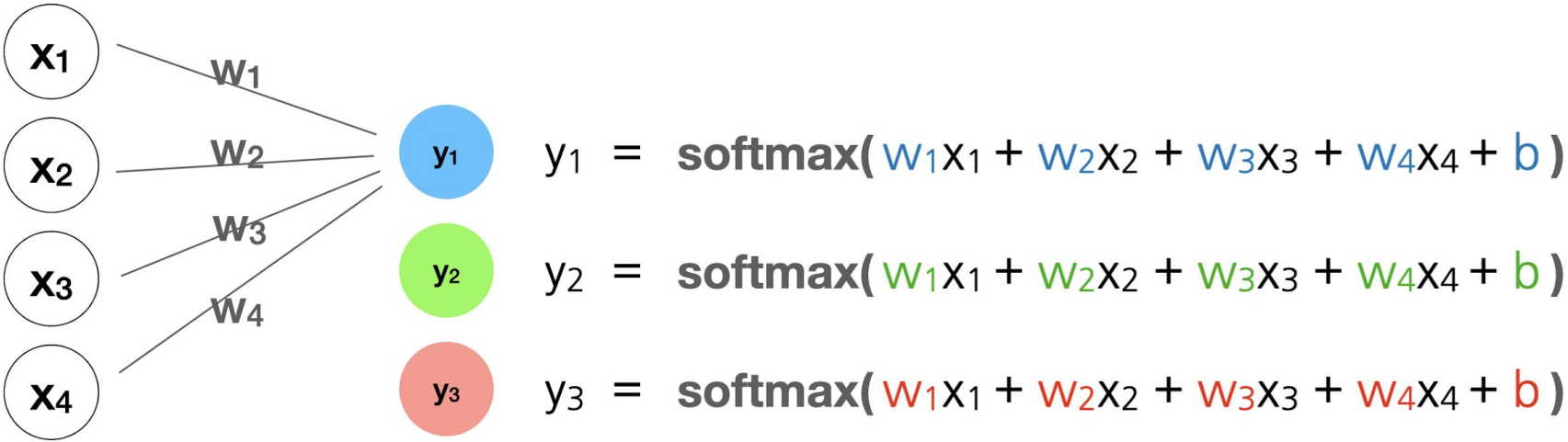
꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0.7	0.3	0
4.7	3.2	1.3	0.2	0.2	0.3	0.5
4.6	3.1	1.5	0.2	1	0	0



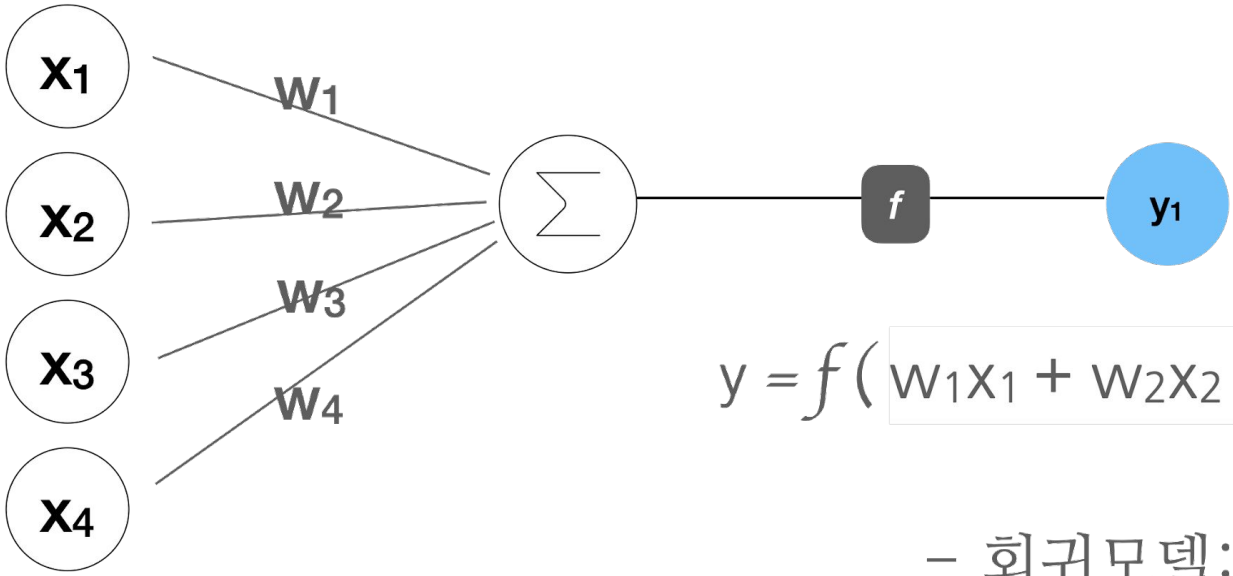
2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy')
```

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭	품종.setosa	품종.virginica	품종.versicolor
5.1	3.5	1.4	0.2	1	0	0
4.9	3	1.4	0.2	0.7	0.3	0
4.7	3.2	1.3	0.2	0.2	0.3	0.5
4.6	3.1	1.5	0.2	1	0	0



Activation
활성화 함수



$$y = f\left(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + b \right)$$

- 회귀모델: Identity (y=x)
- 분류모델: Softmax

2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy')
```

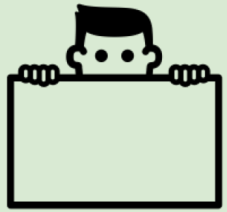
2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[13])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

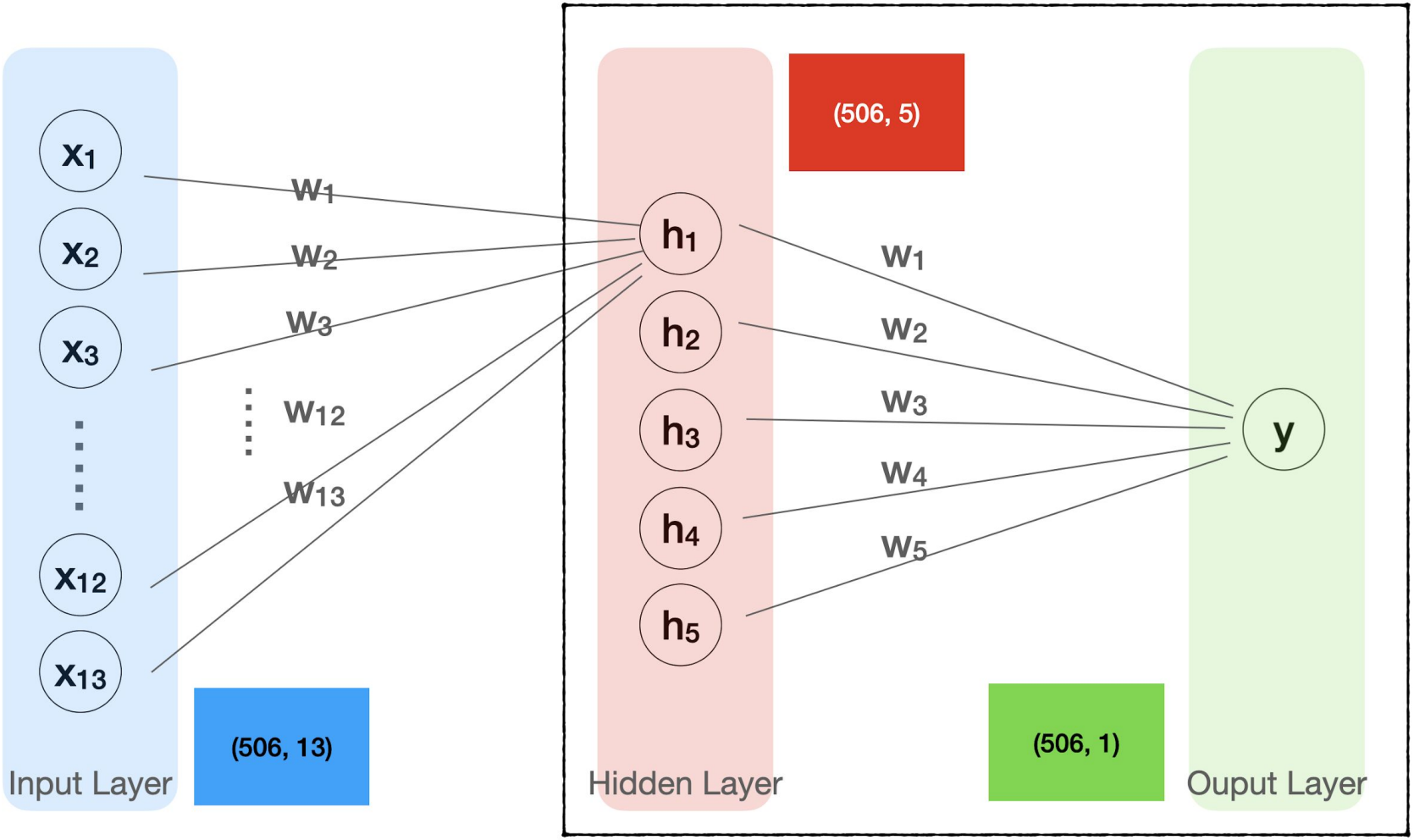
2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[4])
Y = tf.keras.layers.Dense(3, activation='softmax')(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy',
              metrics='accuracy')
```

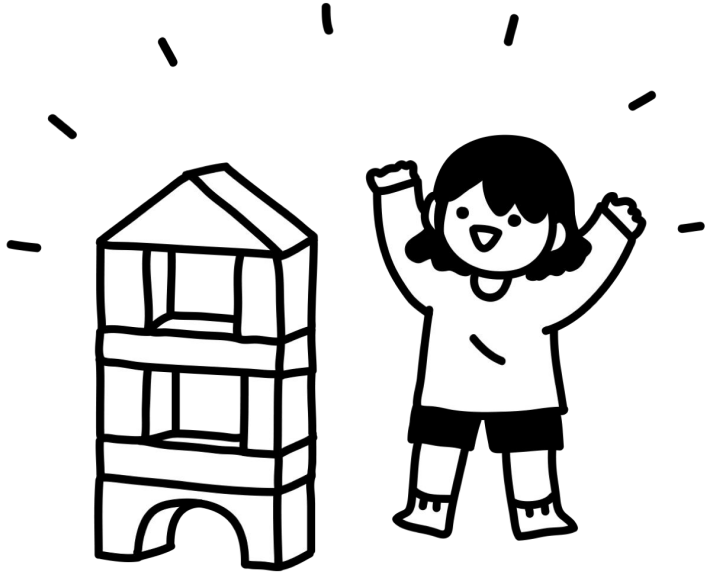
```
Epoch 1/5
5/5 [==============================] - 0s 2ms/step - loss: 0.1584 - accuracy: 0.9667
Epoch 2/5
5/5 [==============================] - 0s 1ms/step - loss: 0.1587 - accuracy: 0.9733
Epoch 3/5
5/5 [==============================] - 0s 2ms/step - loss: 0.1579 - accuracy: 0.9667
Epoch 4/5
5/5 [==============================] - 0s 1ms/step - loss: 0.1580 - accuracy: 0.9733
Epoch 5/5
5/5 [==============================] - 0s 1ms/step - loss: 0.1579 - accuracy: 0.9667
```



hidden layer


x1
x2
x3
x12
x13
w1
w2
w3
w12
w13
(506, 13)
Input Layer
h1
h2
h3
h4
h5
(506, 5)
Hidden Layer
w1
w2
w3
w4
w5
y
(506, 1)
Ouput Layer

```
# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[13])
H = tf.keras.layers.Dense(5, activation='swish')(X)
H = tf.keras.layers.Dense(3, activation=‘swish’)(H)
H = tf.keras.layers.Dense(3, activation=‘swish')(H)
Y = tf.keras.layers.Dense(1)(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```



Linear Regression
Logistic Regression

평균과 분산

$$s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

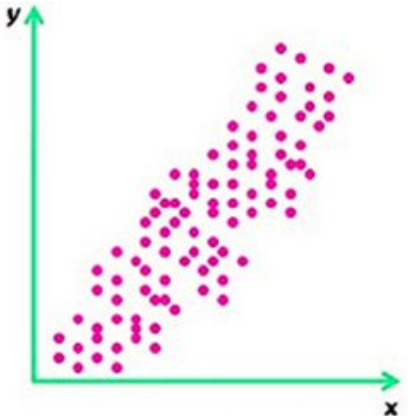
- 평균: 자료의 중심값으로 자료를 대표하는 값
- 분산: 자료들이 평균을 중심으로 퍼져있는 평균적인 거리
- 분산 = 편차제곱합의 평균
- 평균 = 편차제곱합이 최소가 되도록 하는 값
- 평균과 표준편차는 이상치에 취약하다.

"데이터가 정규분포를 따를 때, 최소제곱법이 가장 좋은 추정방법이다."

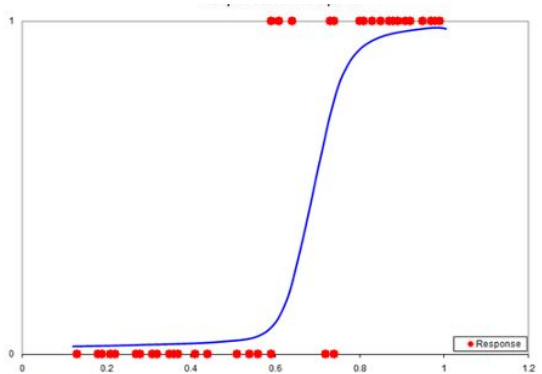
– 가우스

선형회귀와 로지스틱회귀

- 최적의 그래프를 어떻게 찾을까?
  a. 그래프와 각 데이터의 오차를 구한다.
  b. 오차를 모두 더한다.
  c. 오차의 합이 최소가 되게 한다.

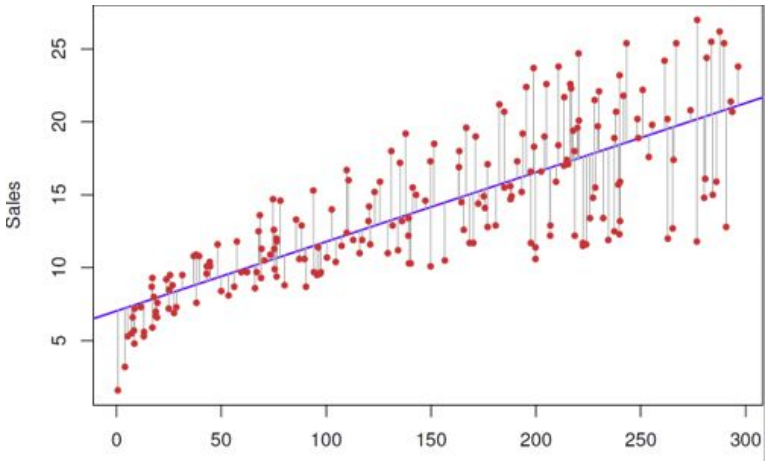


$$y = w_1 x_1 + w_0$$



$$y = class(w_1 x_1 + w_0)$$

선형회귀와 최소제곱추정



$$\hat{y} = f_w(x_i)$$

$$loss_i = (y_i - \hat{y}_i)^2$$

$$sum\ of\ loss = \sum_i^n (y_i - \hat{y}_i)^2$$

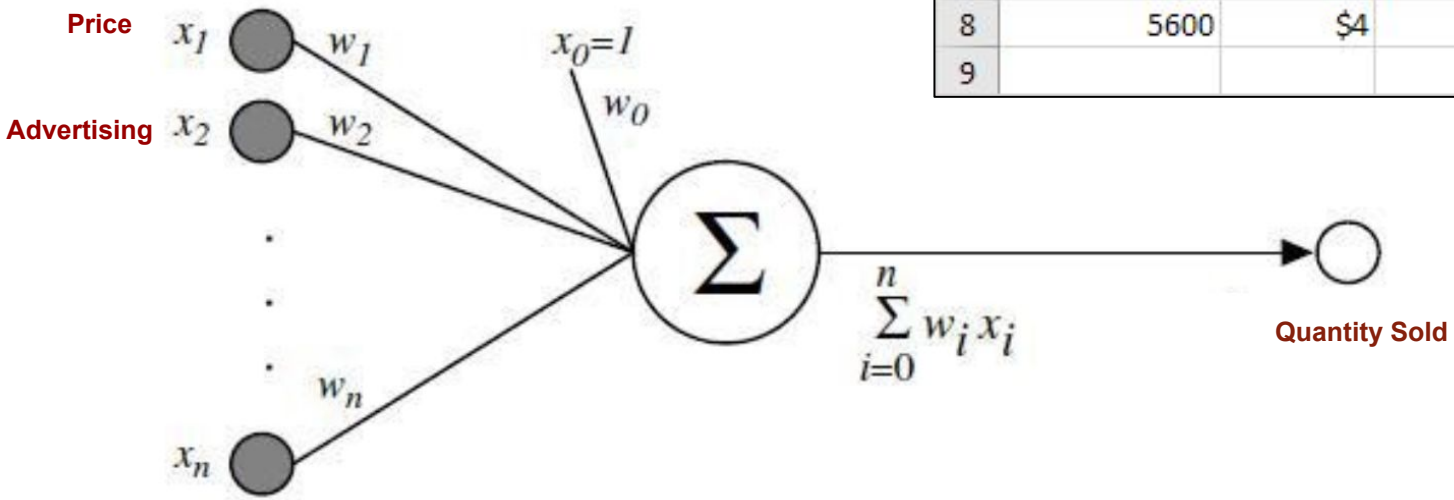
$$\underset{w}{\operatorname{argmin}} \sum_i^n (y_i - f_w(x_i))^2$$

분산의 수식과 비교해보자.

$$s^2 = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{n-1}$$

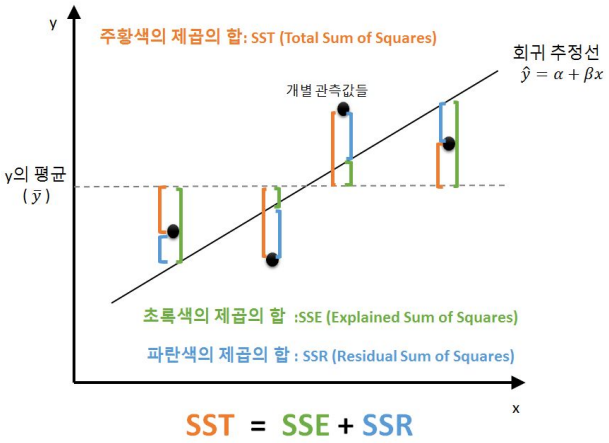
```python
# 모델 생성 or 선택
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(2, )))
model.add(tf.keras.layers.Dense(1))
model.compile(
    optimizer='adam',
    loss='mse'
)
```

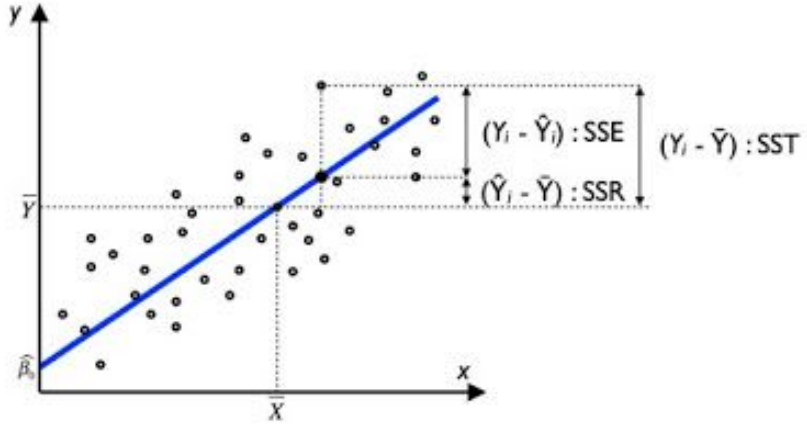
	A	B	C	D
1	Quantity Sold	Price	Advertising	
2	8500	$2	$2,800	
3	4700	$5	$200	
4	5800	$3	$400	
5	7400	$2	$500	
6	6200	$5	$3,200	
7	7300	$3	$1,800	
8	5600	$4	$900	
9				



선형회귀와 최소제곱합

- SST: Y의 평균값으로 설명할 수 없는 분산
- SSR: SST 중 선형회귀식을 통해 얼마나 설명해낸 분산
- SSE: 선형회귀식이 설명할 수 없는 분산



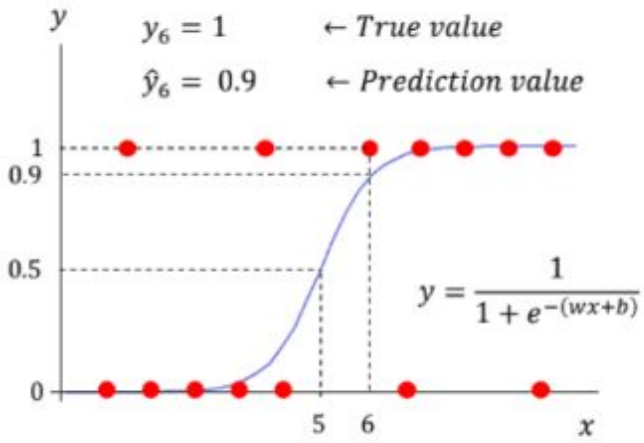


$$SST = \sum_{i=1}^{n} (Y_i - \bar{Y})^2$$

$$SSR = \sum_{i=1}^{n} (\hat{Y}_i - \bar{Y})^2$$

$$SSE = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2$$

로지스틱회귀와 크로스 엔트로피



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$\text{Loss} = -\sum y_i \cdot \log \hat{y}_i$$

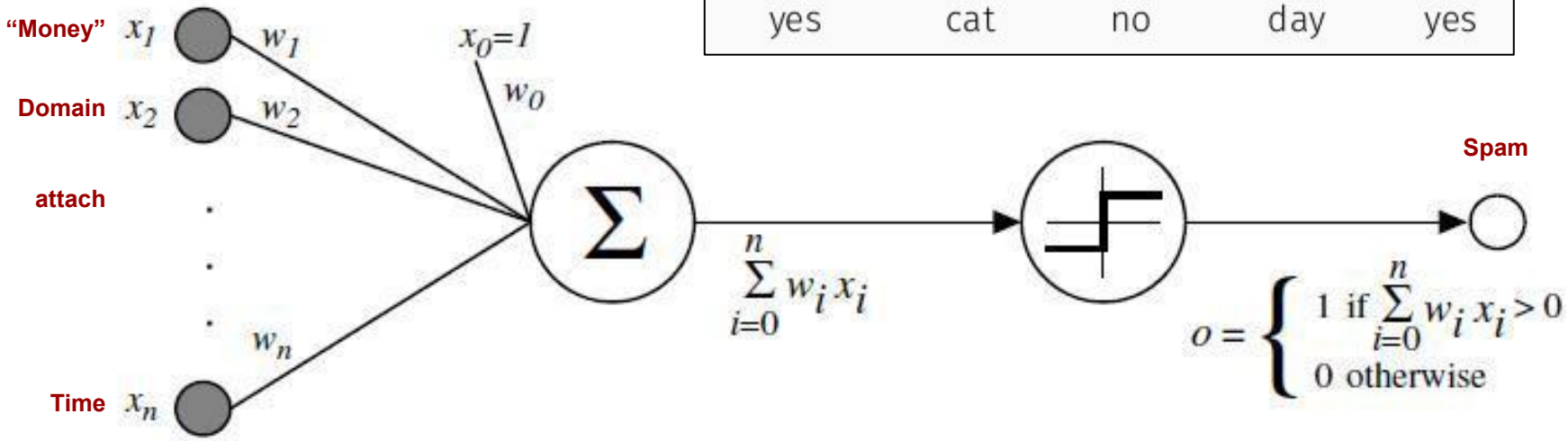
$$\hat{y} = f_w(x_i)$$

$$loss_i = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

$$\underset{w}{\text{argmin}} \sum_{i}^{n} loss_i$$

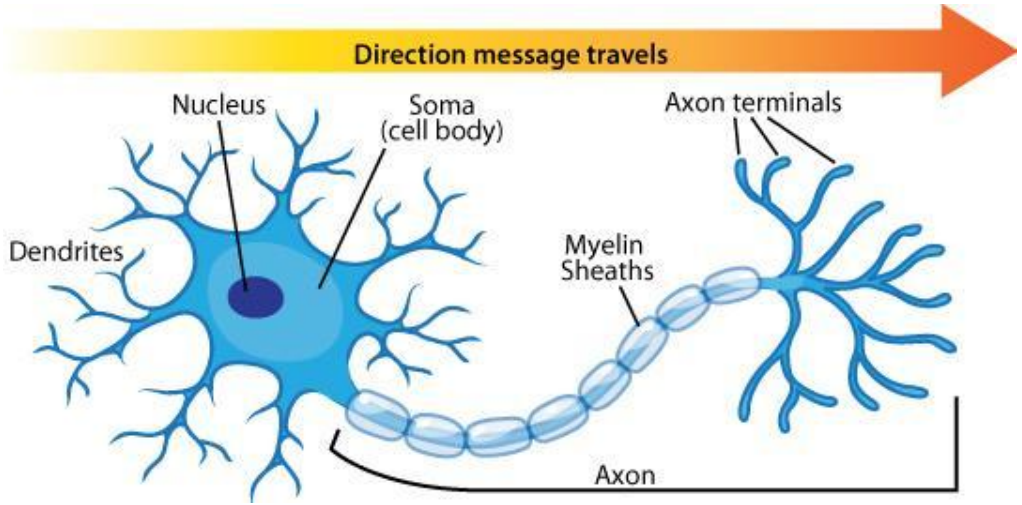
```python
# 모델 생성 or 선택
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(2, )))
model.add(tf.keras.layers.Dense(1))
model.add(tf.keras.layers.Activation("sigmoid"))
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=["accuracy"]
)
```

Contains "Money"	Domain type	Has attach.	Time received	spam
yes	com	yes	night	yes
yes	edu	no	night	yes
no	com	yes	night	yes
no	edu	no	day	no
no	com	no	day	no
yes	cat	no	day	yes



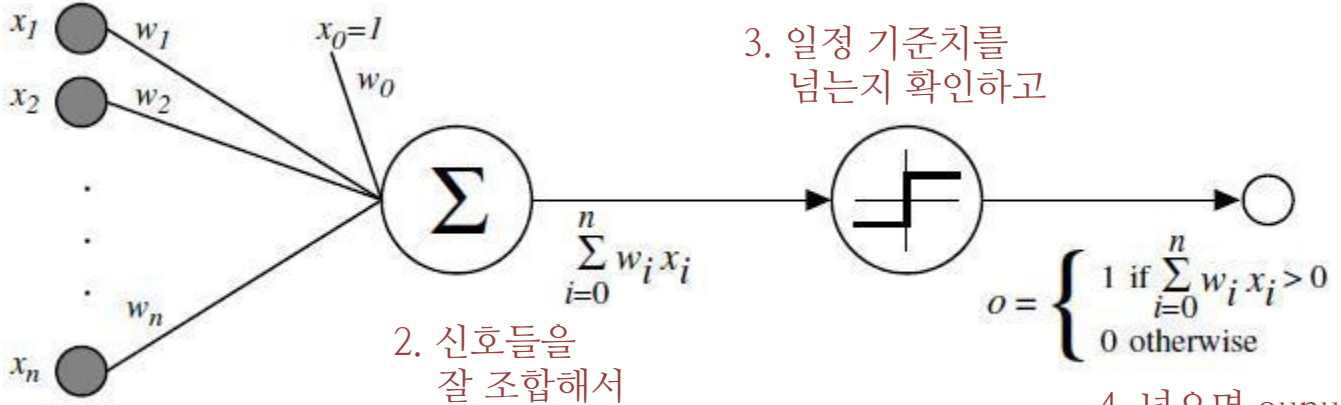
Neuron

1. 근처 뉴런들에게서 이진 신호를 받고
2. 그 신호들을 잘 조합해서
3. 일정 기준치가 넘으면 흥분을 하고
4. 흥분을 하면 terminal로 신호를 보냄!



Perceptron

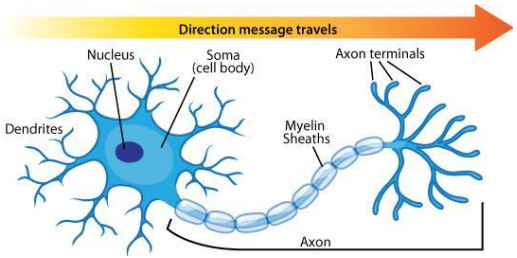
1. 신호들을 잘 받아서



2. 신호들을
잘 조합해서

3. 일정 기준치를
넘는지 확인하고

4. 넘으면 ouput :1
안 넘으면 output : 0



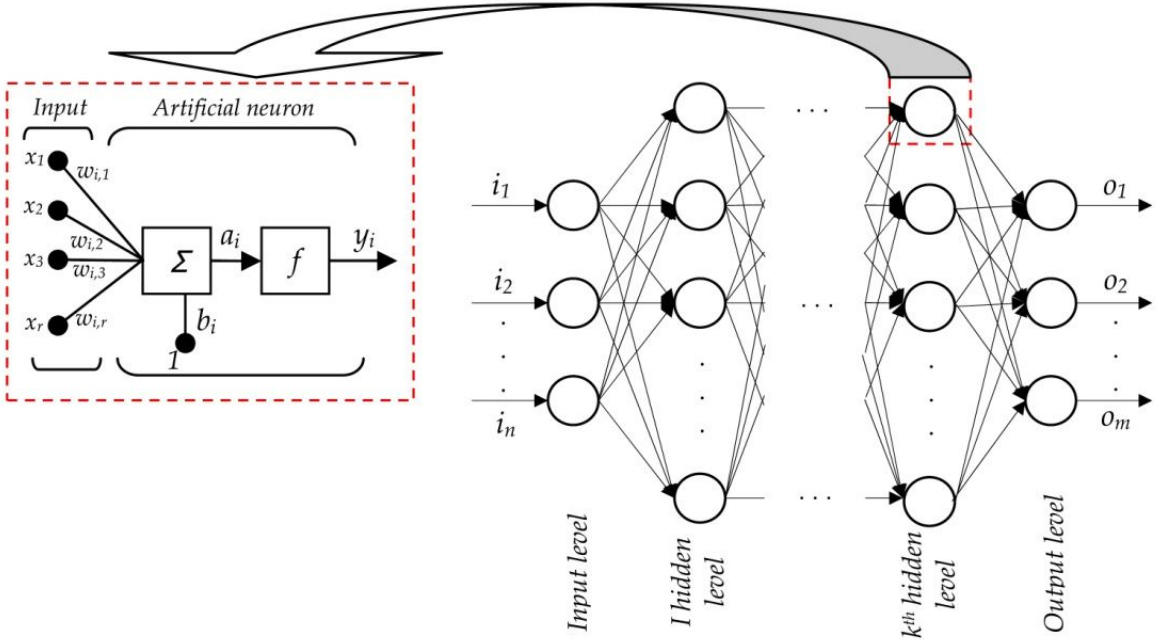
Neuron의 수학적인 모델링!

인공신경망

Perceptron 이 모여 Layer를 이루는 것

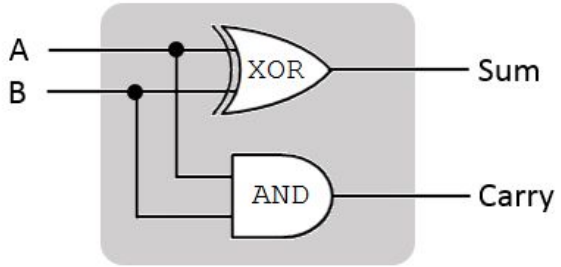
X와 Y사이에 특정 함수가 존재한다면, Multi Layer는 인공신경망이 어떤 함수든 근사(Approximate)할 수 있다.

Multilayer feedforward networks are universal approximators (1989)

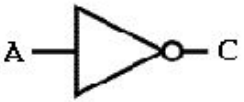


Computer?

- 컴퓨터는 논리 Gate들의 연결을 통해 구성
- Not, And, Or Gate를 쌓아 연산을 만듬

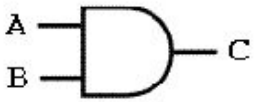


NOT



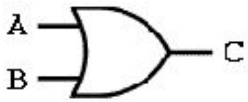
Input	Output
A	C
0	1
1	0

AND



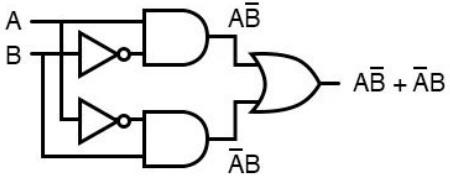
Inputs		Output
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

OR



Inputs		Output
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

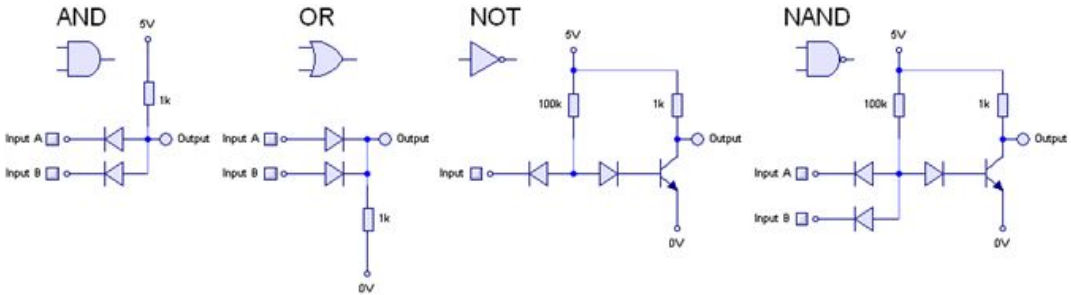
. . . is equivalent to . .

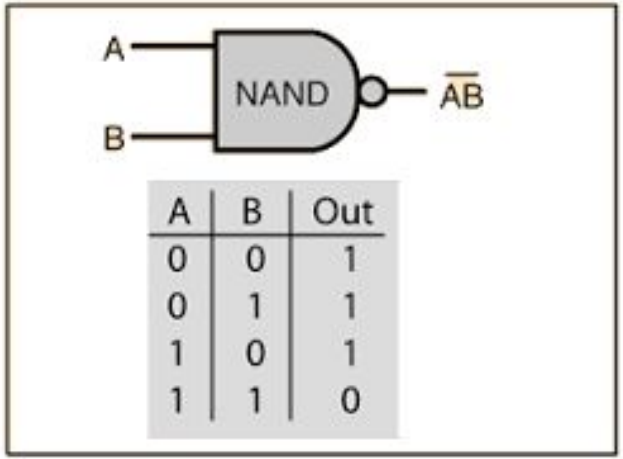


$A \oplus B = A\bar{B} + \bar{A}B$

Computer?

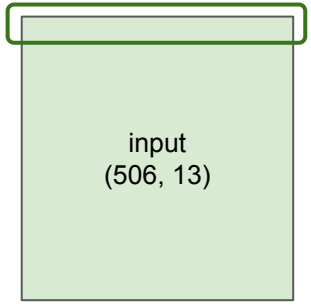
- Nand Gate를 쌓아 Not, And, Or Gate를 만듬.
- Nand만으로 컴퓨터를 완성할 수 있음.
- Nand Gate는 Universal Gate라고도 불림.

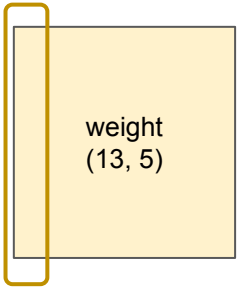


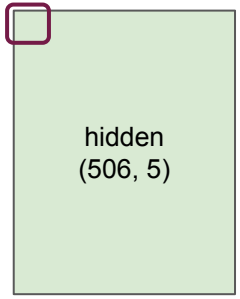


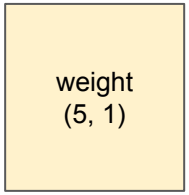
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

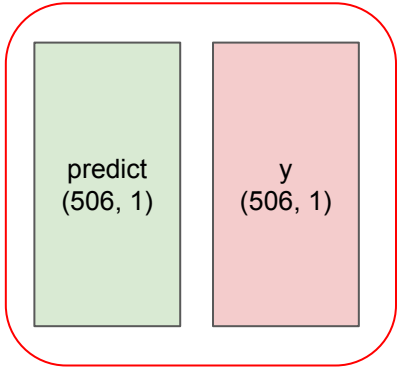
Universal Gate

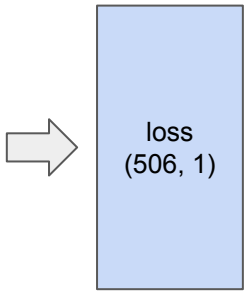






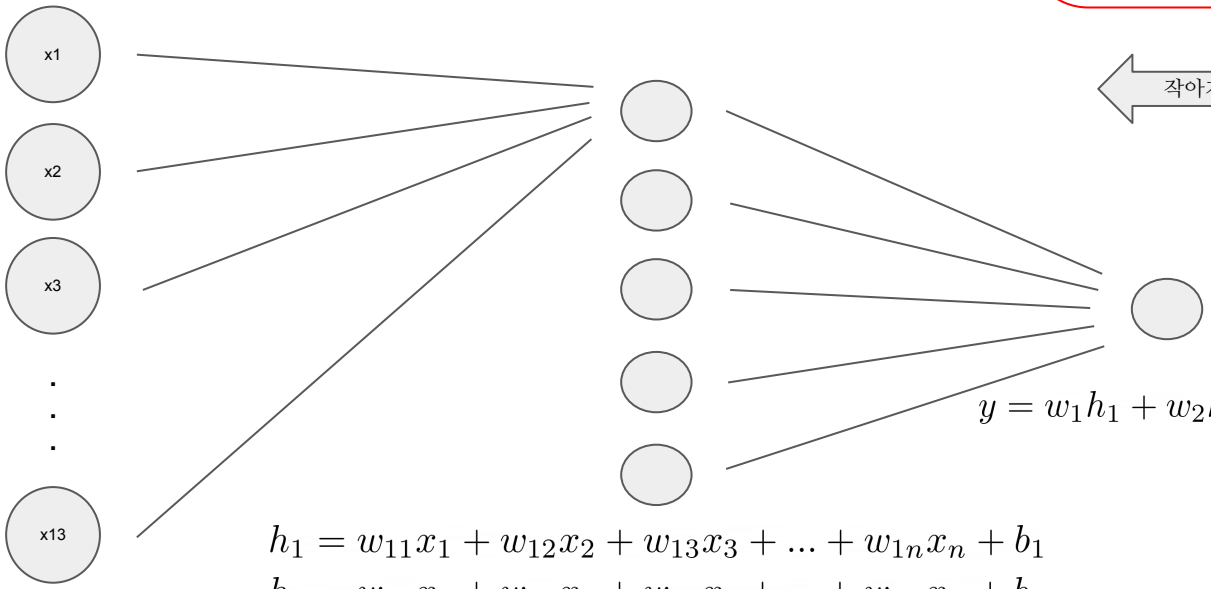






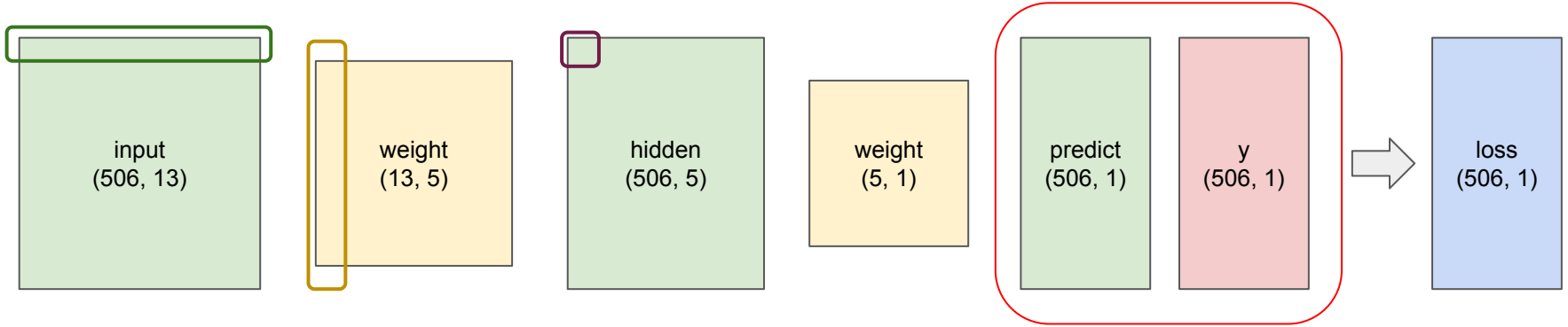
$\sum$

작아지는 방향으로 weight 조정



$$y = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_3 + w_5h_5 + b$$

$$h_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + ... + w_{1n}x_n + b_1$$
$$h_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + ... + w_{2n}x_n + b_2$$



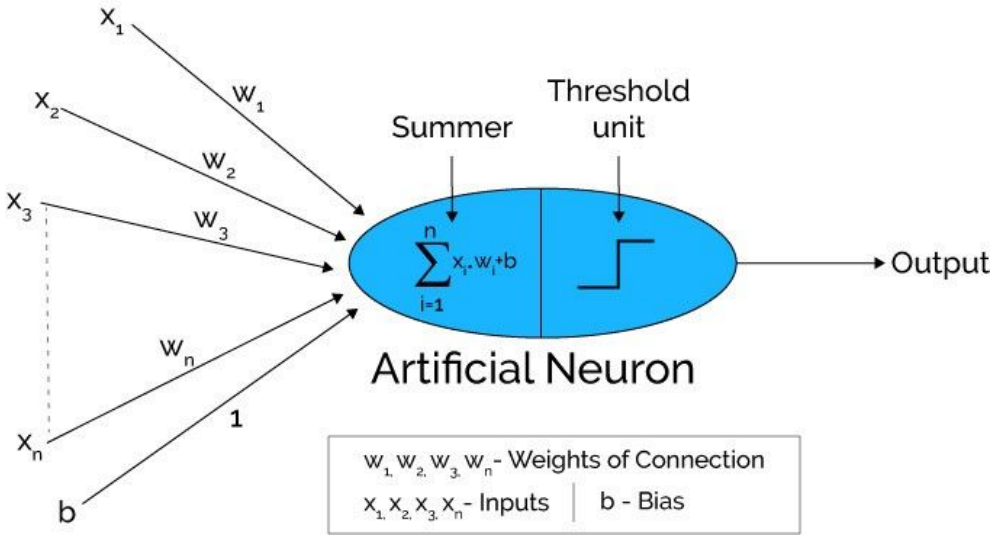
$$h_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + ... + w_{1n}x_n + b_1$$
$$h_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + ... + w_{2n}x_n + b_2$$
$$h_3 = w_{31}x_1 + w_{32}x_2 + w_{33}x_3 + ... + w_{3n}x_n + b_3$$
$$h_4 = w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + ... + w_{4n}x_n + b_4$$
$$h_5 = w_{51}x_1 + w_{52}x_2 + w_{53}x_3 + ... + w_{5n}x_n + b_5$$

$$y = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_3 + w_5h_5 + b$$

$$H = W_1X + B_1 \qquad Y = W_2H + B_2$$

$$Y = WX + b$$

$$y = w_1x_1 + w_2x_2 + \cdots\cdots + w_nx_n + b$$

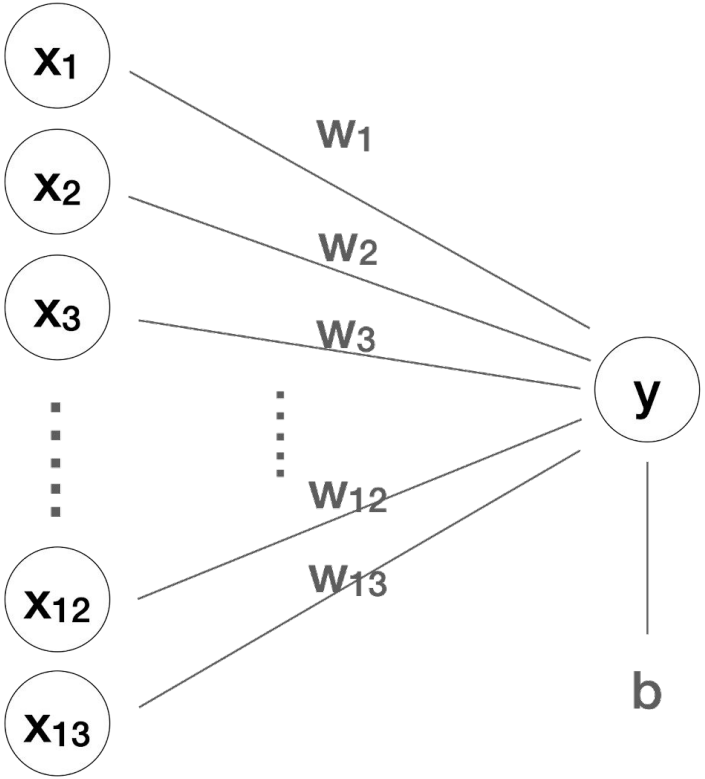


x1	x2	y
0	0	1
0	1	0
1	0	0
1	1	0

```
# 모델 생성 or 선택
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.Input(shape=(2, )))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=["accuracy"])

```

```
X = tf.keras.layers.Input(shape=[13])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14
CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6
0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4



$$y = w_1x_1 + w_2x_2 + \cdots\cdots + w_{13}x_{13} + b$$

온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

```
레모네이드 = pd.read_csv('lemonade.csv')
독립 = 레모네이드[['온도']]
종속 = 레모네이드[['판매량']]

print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

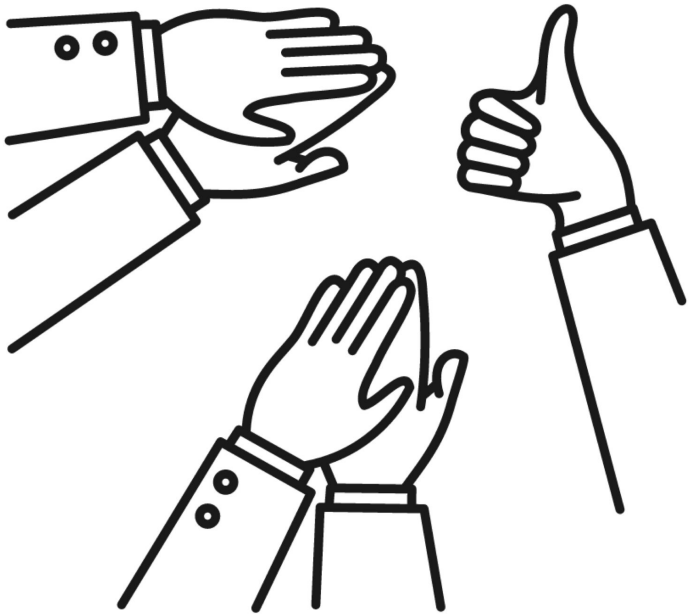
```
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

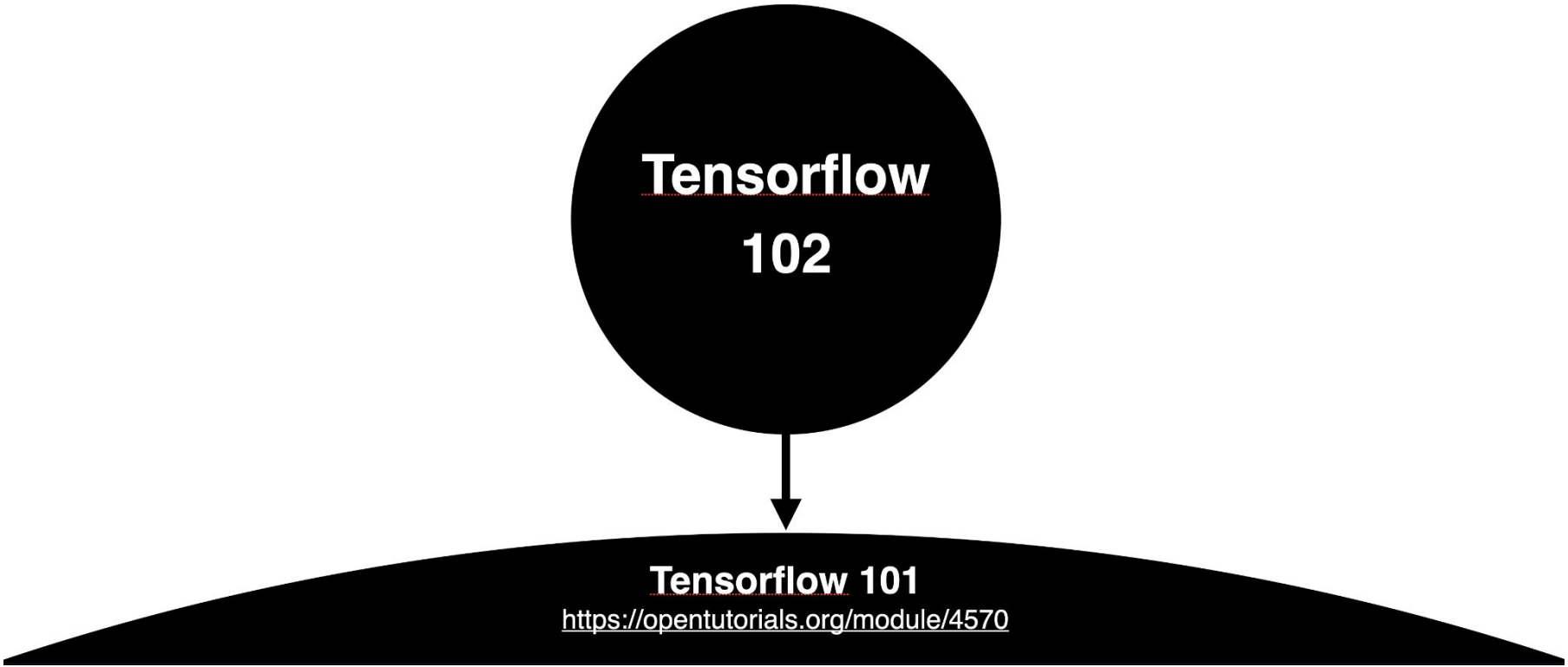
3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

4. 모델을 이용합니다

```
print("Predictions: ", model.predict([[15]]))
```




Tensorflow
102
Tensorflow 101
https://opentutorials.org/module/4570

날짜	요일	온도	판매량
2020.1.3	금	20	40
2020.1.4	토	21	42
2020.1.5	일	22	44

1.과거의 데이터를 준비합니다.

```
레모네이드 = pd.read_csv('lemonade.csv')
독립 = 레모네이드[['온도']]
종속 = 레모네이드[['판매량']]
print(독립.shape, 종속.shape)
```

2. 모델의 구조를 만듭니다

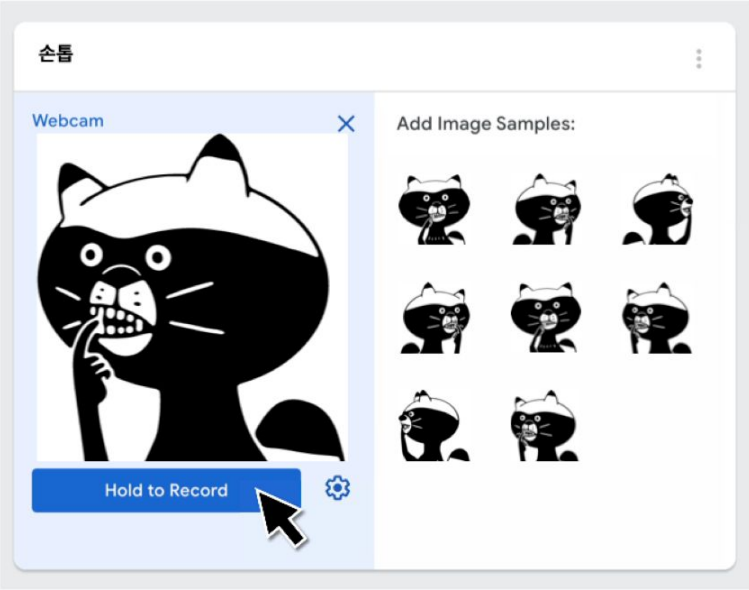
```
X = tf.keras.layers.Input(shape=[1])
Y = tf.keras.layers.Dense(1)(X)
model = tf.keras.models.Model(X, Y)
model.compile(loss='mse')
```

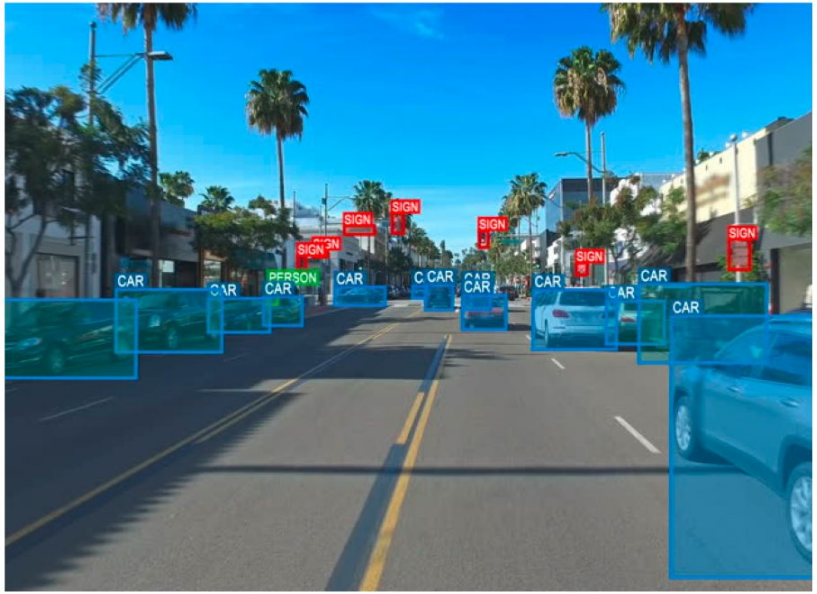
3.데이터로 모델을 학습(FIT)합니다.

```
model.fit(독립, 종속, epochs=1000)
```

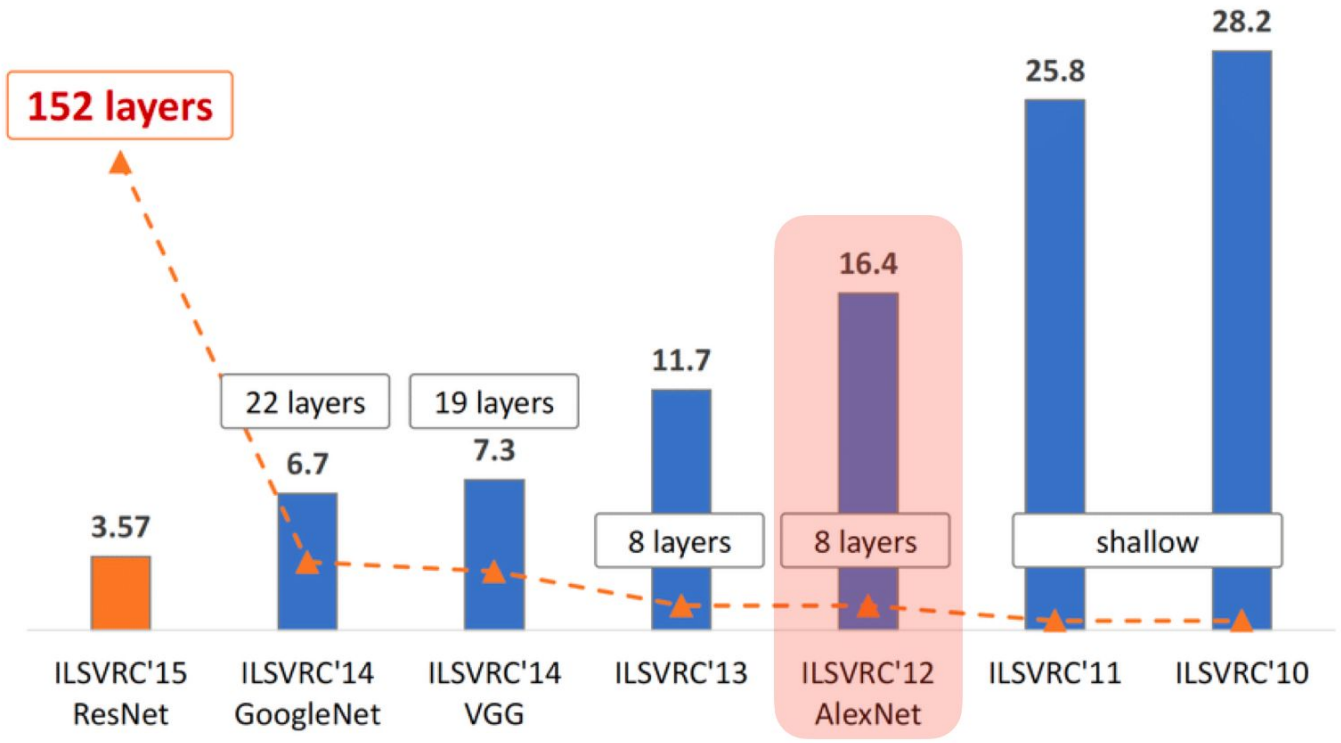
4. 모델을 이용합니다

```
print("Predictions: ", model.predict([[15]]))
```


손톱
Webcam
Add Image Samples:
Hold to Record

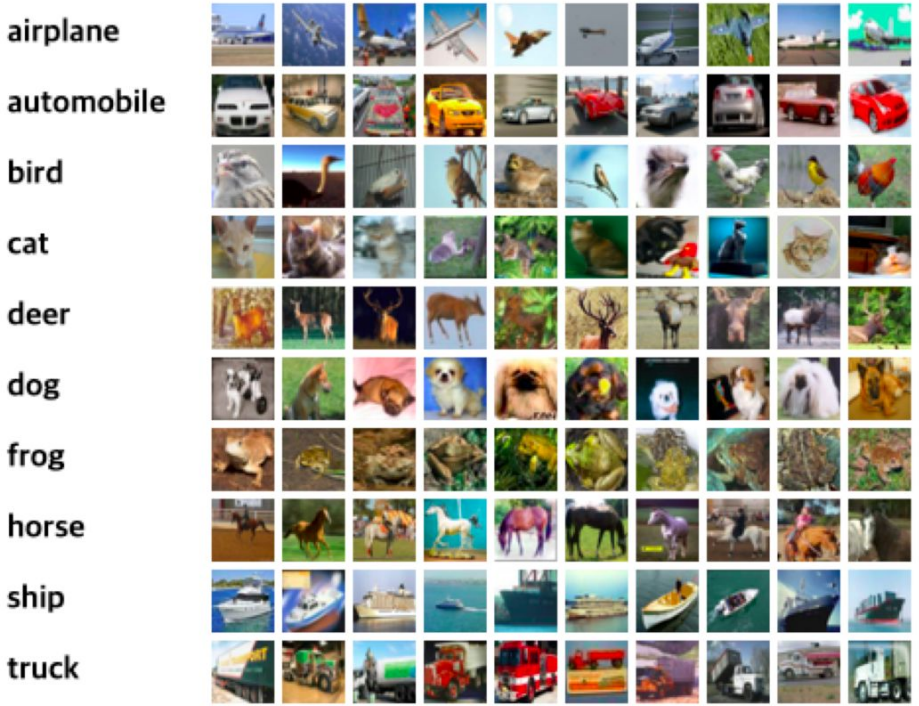

CAR
PERSON
SIGN


WALL


152 layers
22 layers
19 layers
8 layers
8 layers
shallow
3.57
6.7
7.3
11.7
16.4
25.8
28.2
ILSVRC'15 ResNet
ILSVRC'14 GoogleNet
ILSVRC'14 VGG
ILSVRC'13
ILSVRC'12 AlexNet
ILSVRC'11
ILSVRC'10

MNIST

CIFAR10



```
# 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
종속 = tf.keras.utils.to_categorical(종속)
print(독립.shape, 종속.shape)
```

```
# 모델을 만듭니다.
X = tf.keras.layers.Input(shape=[28, 28])
H = tf.keras.layers.Flatten()(X)
H = tf.keras.layers.Dense(120, activation='swish')(H)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy',
              metrics='accuracy')
```

```
# 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 28, 28, 1)
종속 = tf.keras.utils.to_categorical(종속)
print(독립.shape, 종속.shape)
```

```
# 모델을 만듭니다.
X = tf.keras.layers.Input(shape=[28, 28, 1])

H = tf.keras.layers.Conv2D(6, kernel_size=5, padding="same")(X)
H = tf.keras.layers.Activation('swish')(H)
H = tf.keras.layers.MaxPool2D()(H)

H = tf.keras.layers.Conv2D(16, kernel_size=5)(H)
H = tf.keras.layers.Activation('swish')(H)
H = tf.keras.layers.MaxPool2D()(H)

H = tf.keras.layers.Flatten()(H)
H = tf.keras.layers.Dense(120, activation='swish')(H)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)

model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy', metrics='accuracy')
```

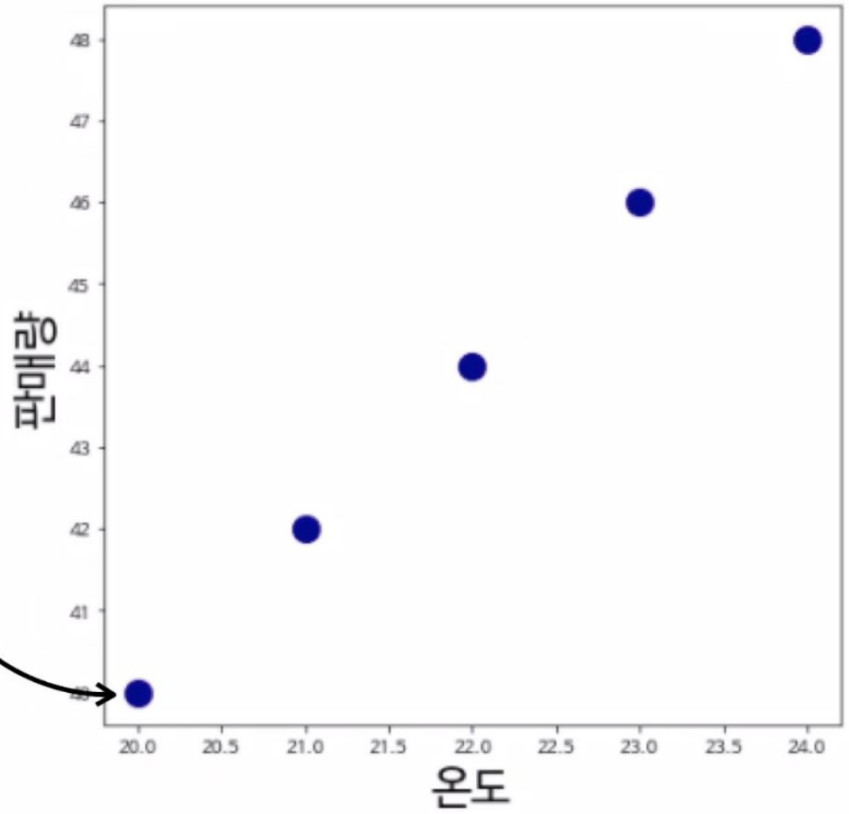
차원

표의 열 vs 포함 관계

표의 열 vs 포함 관계

5

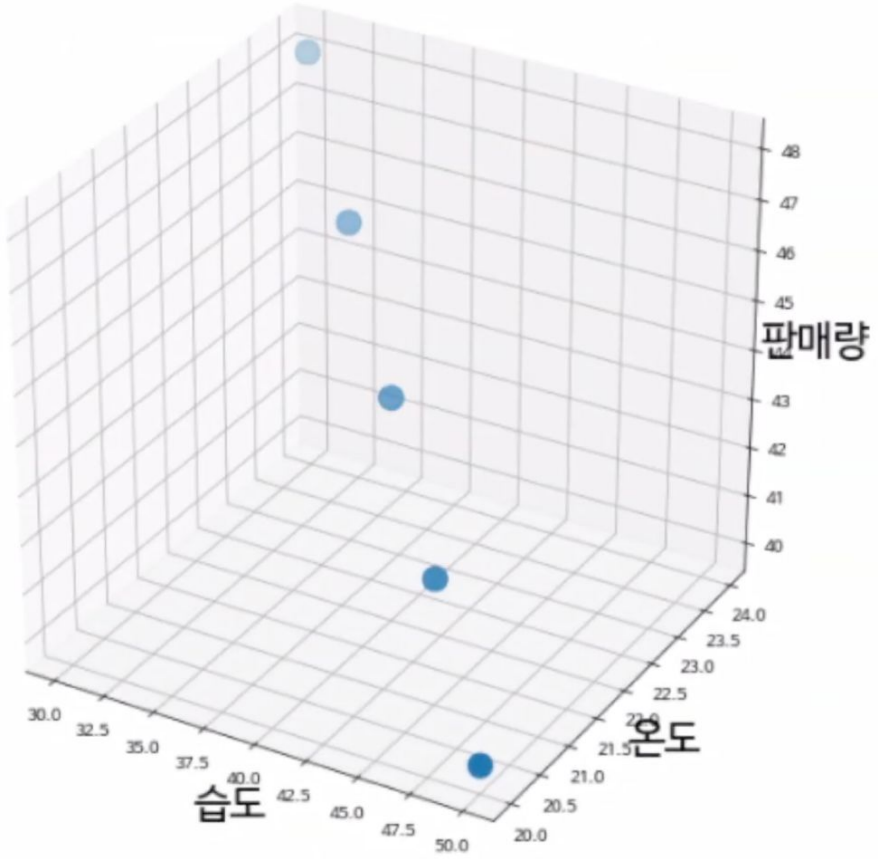
온도	판매량
20	40
21	42
22	44
23	46
24	48



표의 열 vs 포함 관계

5

습도	온도	판매
30	20	40
35	21	42
40	22	44
45	23	46
50	24	48

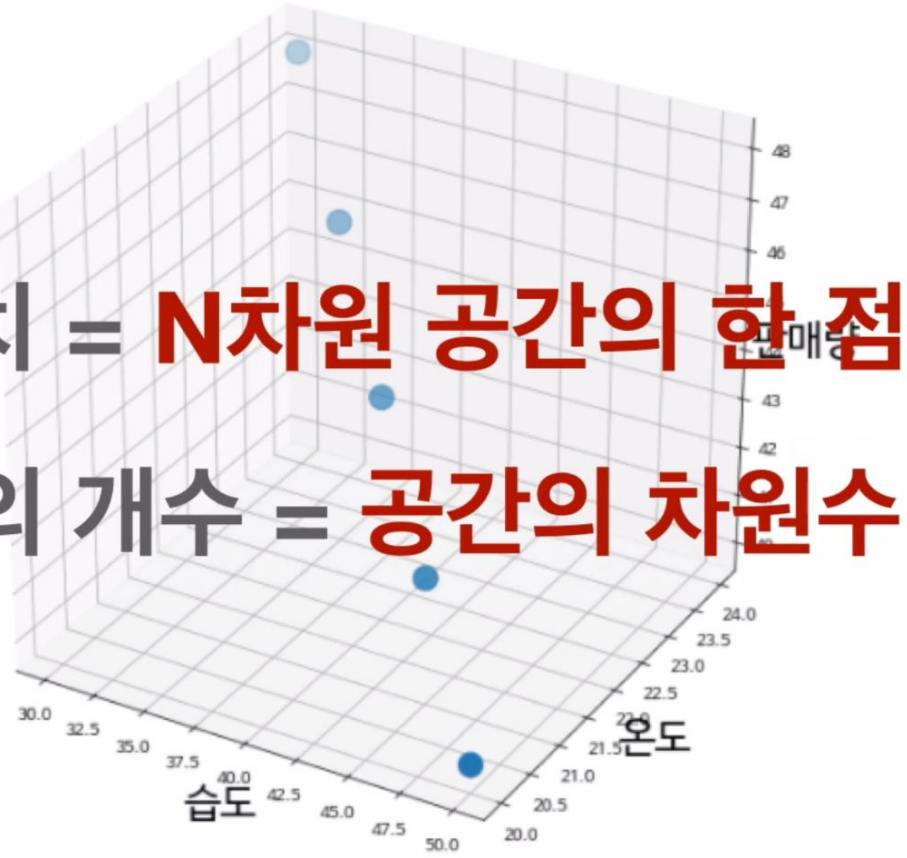


표의 열 vs 포함 관계

습도	온도	판매	…	N
30	20	40	…	
35	21	42	…	
40	22	44	…	
45	23	46	…	
50	24	48	…	

관측치 = N차원 공간의 한 점

변수의 개수 = 공간의 차원수



표의 열 vs 포함 관계

꽃잎길이	꽃잎폭	꽃받침길이	꽃받침폭
5.1	3.5	1.4	0.2
4.9	3.0	1.5	0.2
4.7	3.2	1.3	0.2

4차원 공간에 표현되는 관측치

배열의 깊이 = “차원수”

```
x1 = np.array( [5.1, 3.5, 1.4, 0.2] )
print(x1.ndim, x1.shape)
# 1차원 형태,  (4, )

x2 = np.array( [4.9, 3.0, 1.4, 0.2] )
x3 = np.array( [4.7, 3.2, 1.3, 0.2] )

아이리스 = np.array( [x1, x2, x3] )
print( 아이리스.ndim, 아이리스.shape )
# 2차원 형태,  (3, 4)
```

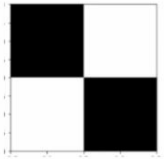
표의 열 vs 포함 관계

```
x1 = np.array( [5.1, 3.5, 1.4, 0.2] )
print(x1.ndim, x1.shape)
# 1차원 형태,  (4, )
```

```
x2 = np.array( [4.9, 3.0, 1.4, 0.2] )
x3 = np.array( [4.7, 3.2, 1.3, 0.2] )
```

```
아이리스 = np.array( [x1, x2, x3] )
print( 아이리스.ndim, 아이리스.shape )
# 2차원 형태,  (3, 4)
```

```
img1 = np.array( [
  [  0,  255],
  [255,    0]
] )
print(img1.ndim, img1.shape)
# 2차원 형태,  (2, 2)
```



```
img2 = np.array( [[255, 255], [255, 255]] )
img3 = np.array( [[0, 0], [0, 0]] )
```

```
이미지셋 = np.array( [img1, img2, img3] )
print( 이미지셋.ndim, 이미지셋.shape )
# 3차원 형태,  (3, 2, 2)
```

표의 열 vs 포함 관계

```
d1 = np.array( [1, 2, 3] )
print( d1.ndim, d1.shape)
# 1차원 형태,  (3, )
```

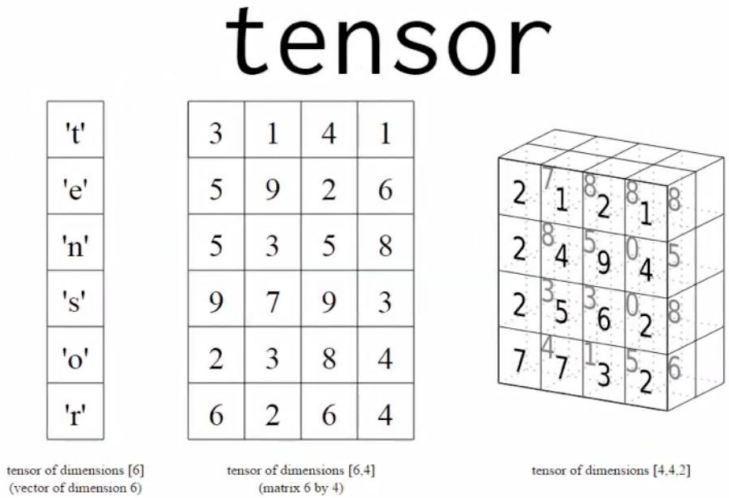
```
d2 = np.array( [d1, d1, d1, d1, d1] )
print( d2.ndim, d2.shape)
# 2차원 형태,  (5, 3)
```

```
d3 = np.array( [d2, d2, d2, d2] )
print( d3.ndim, d3.shape)
# 3차원 형태,  (4, 5, 3)
```

```
d4 = np.array( [d3, d3] )
print( d4.ndim, d4.shape)
# 4차원 형태,  (2, 4, 5, 3)
```

배열의 깊이 = 차원수

표의 열 vs 포함 관계



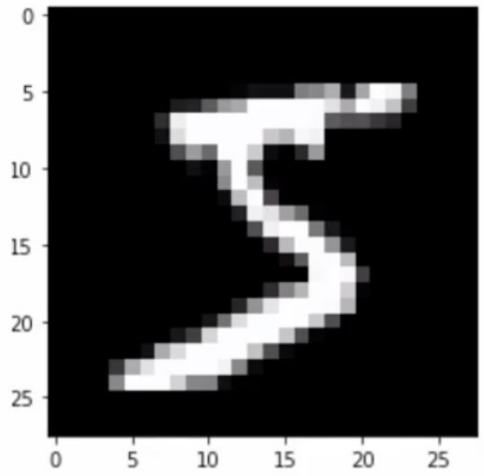

```
d2 = np.array( [d1, d1, d1, d1, d1] )
print( d2.ndim, d2.shape)
# 2차원 형태,  (5, 3)
```

```
d4 = np.array( [d3, d3] )
print( d4.ndim, d4.shape)
# 4차원 형태,  (2, 4, 5, 3)
```

배열의 깊이 = 차원수

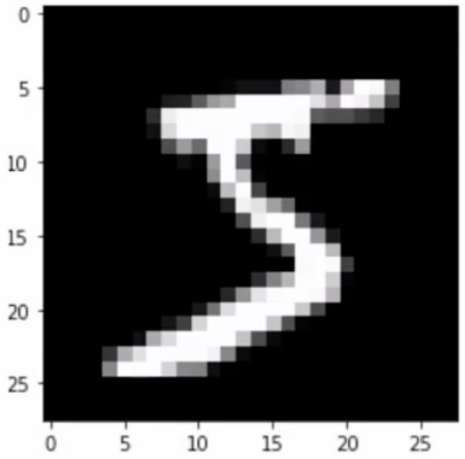
- 데이터 **공간**의 맥락
  차원수 = **변수의 개수**

- 데이터 **형태**의 맥락
  차원수 = **배열의 깊이**



개별 Mnist - (28, 28)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
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	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
2	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
3	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
4	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
5	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0	0
6	0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
7	0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0
8	0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	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
26	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
27	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



개별 Mnist - (28, 28)

Mnist 셋: (60000, 28, 28)

28

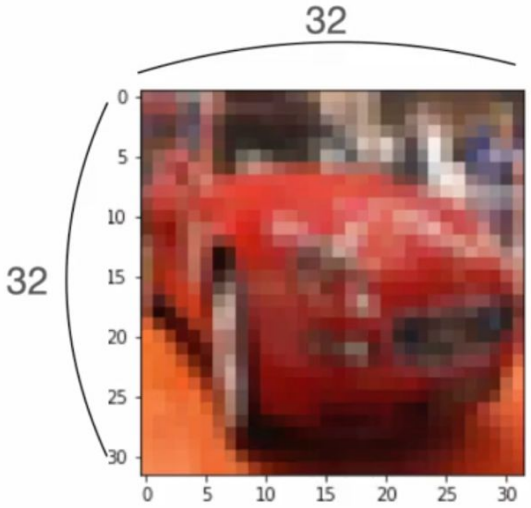
28

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
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	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
2	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
3	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
4	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
5	0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0
6	0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0
7	0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0
8	0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	253	250	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	23	66	213	253	253	253	253	198	81	2	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	18	171	219	253	253	253	253	195	80	9	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	0	0	55	172	226	253	253	253	253	244	133	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	0	0	0	136	253	253	253	212	135	132	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	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
26	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
27	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

2차원 형태

784차원 공간의 한 점

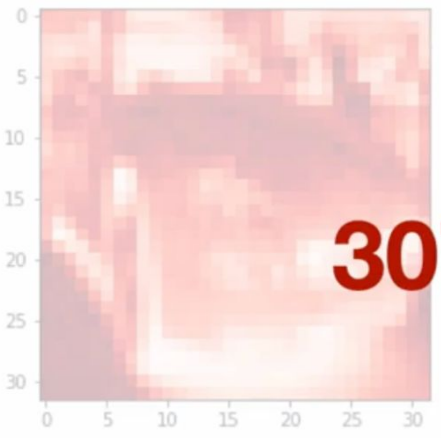
784개의 숫자 (= 28 * 28)

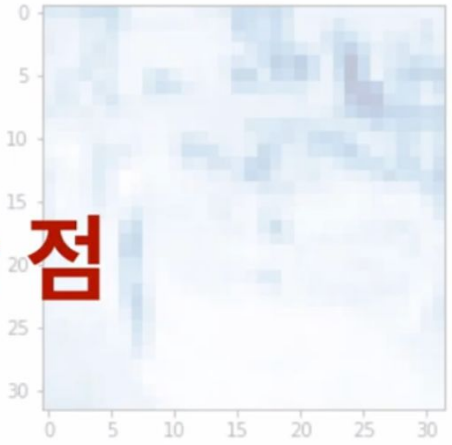


개별 cifar10 - (32, 32, 3)

cifar10 셋: (5000, 32, 32, 3)

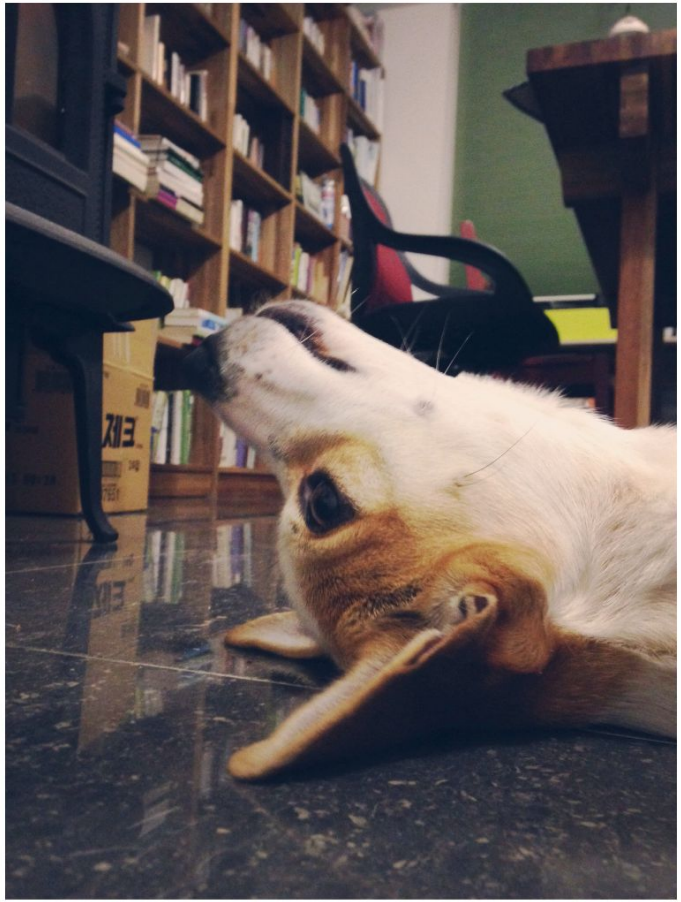
3072개의 숫자 (= 32 * 32 * 3)





3차원 형태

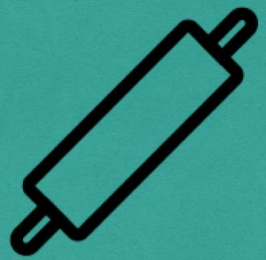
3072차원 공간의 한 점



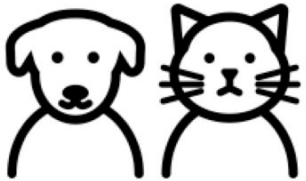
2,448 * 3,264 * 3 = 23,970,816

주전: Tensorfl

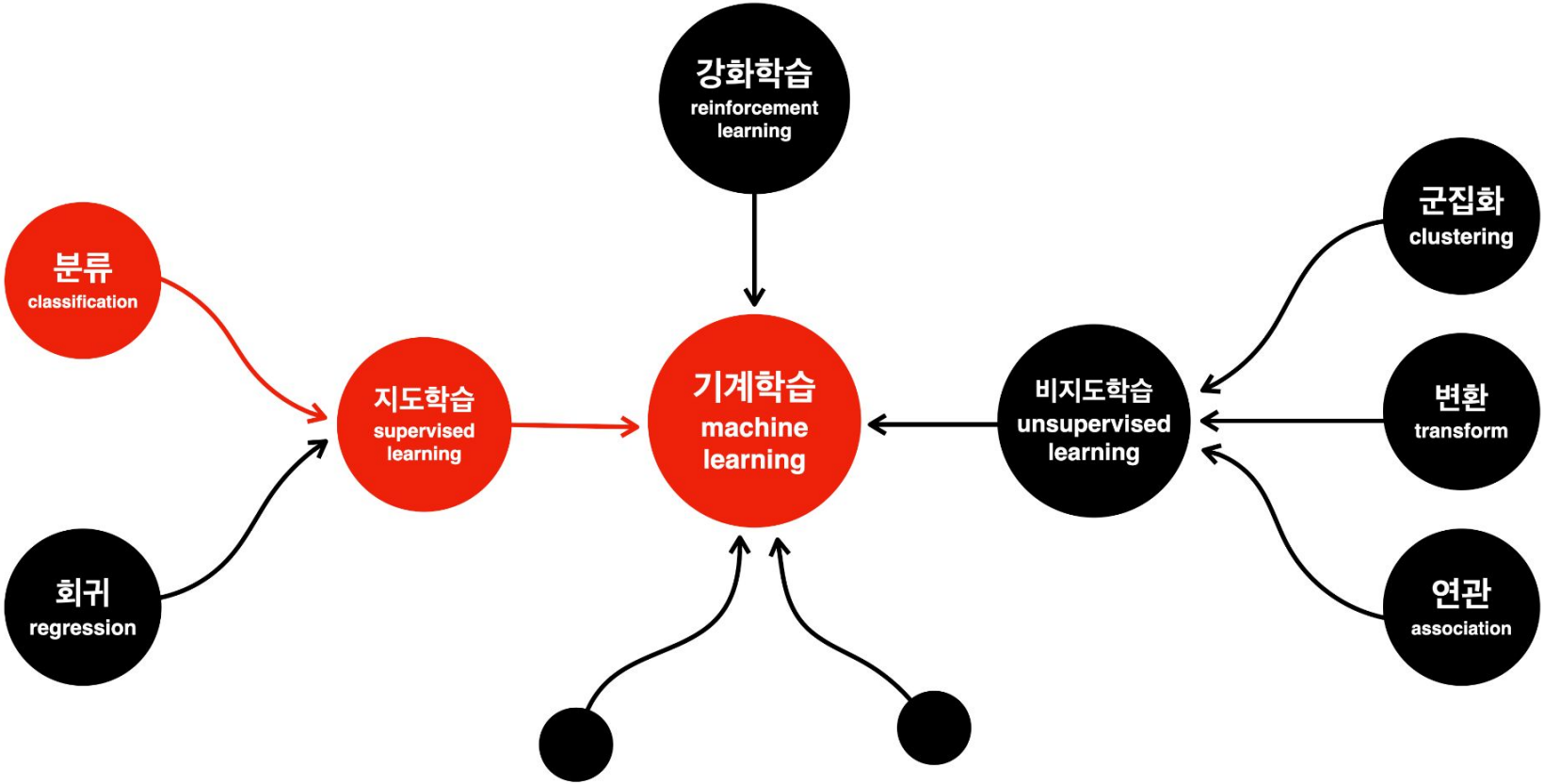
```
# MNIST
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
print(독립.shape, 종속.shape)
# (60000, 28, 28) (60000,)

# CIFAR10
(독립, 종속), _ = tf.keras.datasets.cifar10.load_data()
print(독립.shape, 종속.shape)
# (50000, 32, 32, 3) (50000, 1)
```

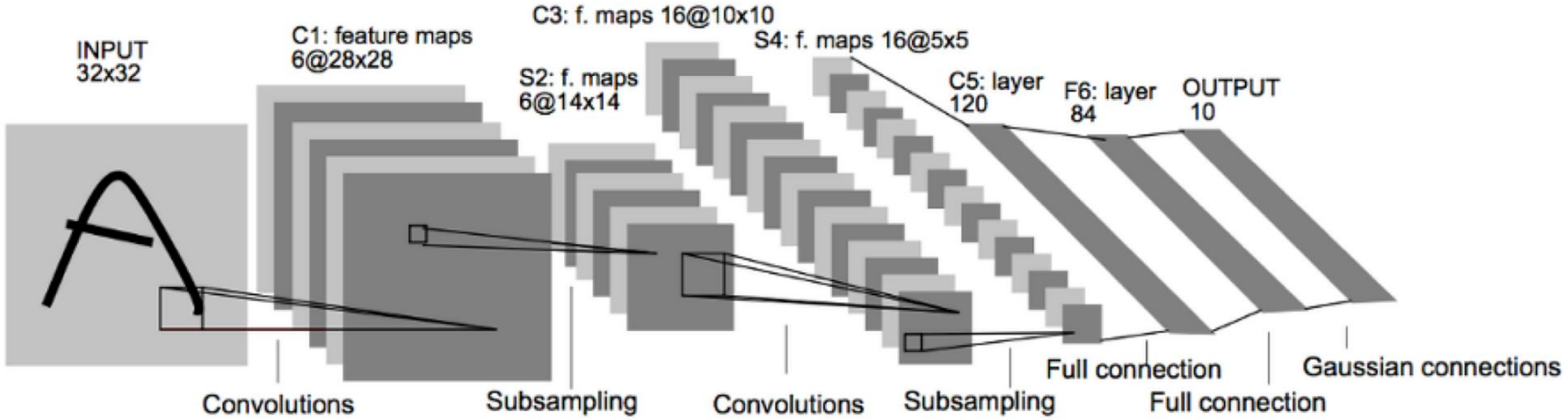


다섯번째 딥러닝 1 - Flatten



이미지 분류기


강화학습
reinforcement learning
분류
classification
지도학습
supervised learning
기계학습
machine learning
비지도학습
unsupervised learning
군집화
clustering
변환
transform
연관
association
회귀
regression



CNN called LeNet by Yann LeCun (1998)

독립변수 종속변수

원인 결과

온도	판매량
20	40
21	42
22	44
23	46

1.과거의 데이터를 준비합니다.

3.데이터로 모델을 학습(FIT)합니다.

2. 모델의 구조를 만듭니다

15도

원인

모델

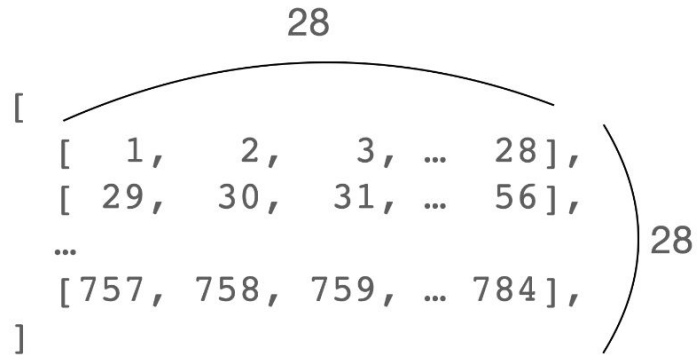
판매량=온도X2

30개

결과

4. 모델을 이용합니다

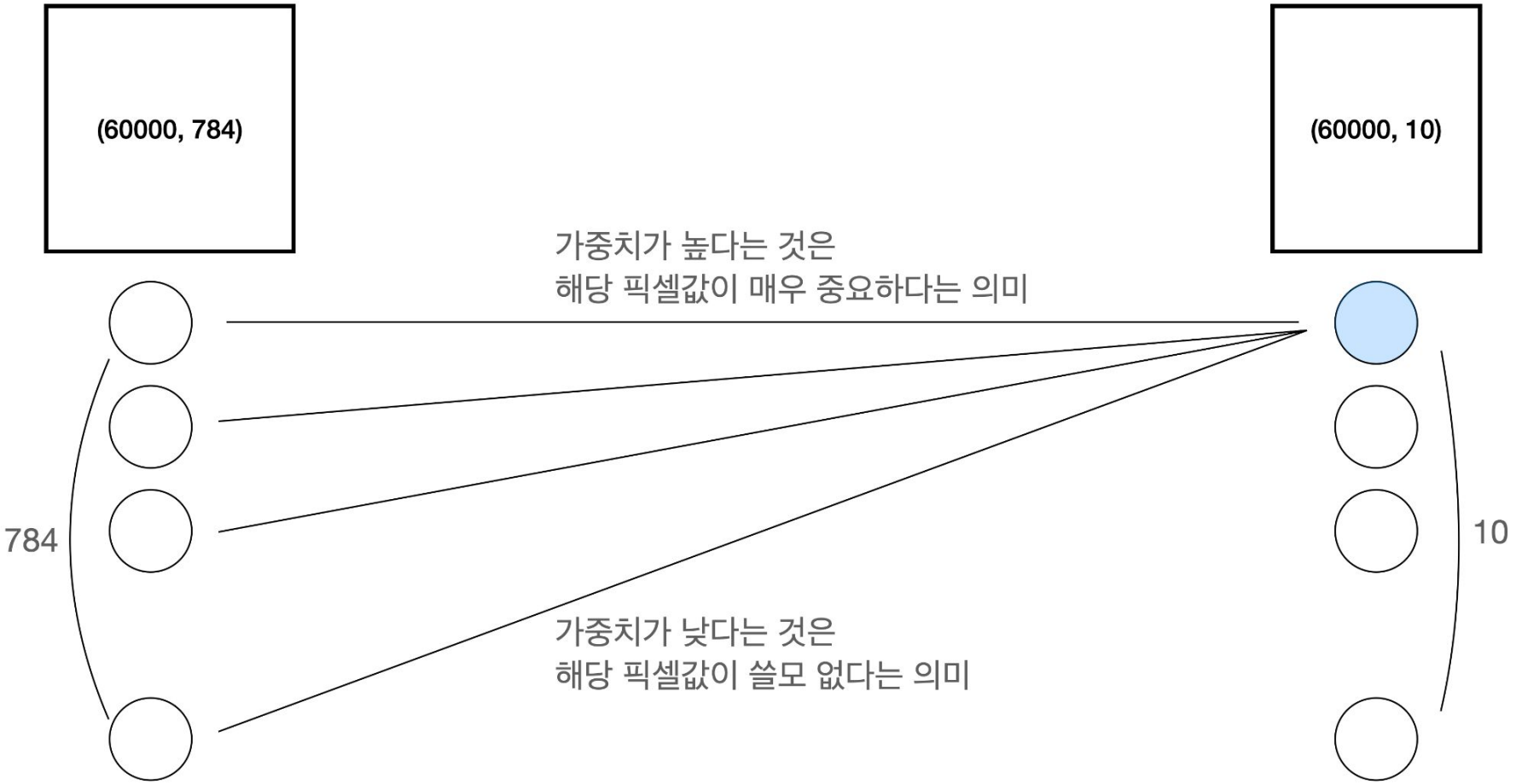
```
# 1.과거의 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 784)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)

# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[784])
H = tf.keras.layers.Dense(84, activation='swish')(X)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='categorical_crossentropy', metrics='accuracy')

# 3.데이터로 모델을 학습(FIT)합니다.
model.fit(독립, 종속, epochs=10)

# 4. 모델을 이용합니다
print("Predictions: ", model.predict(독립[0:5]))
```

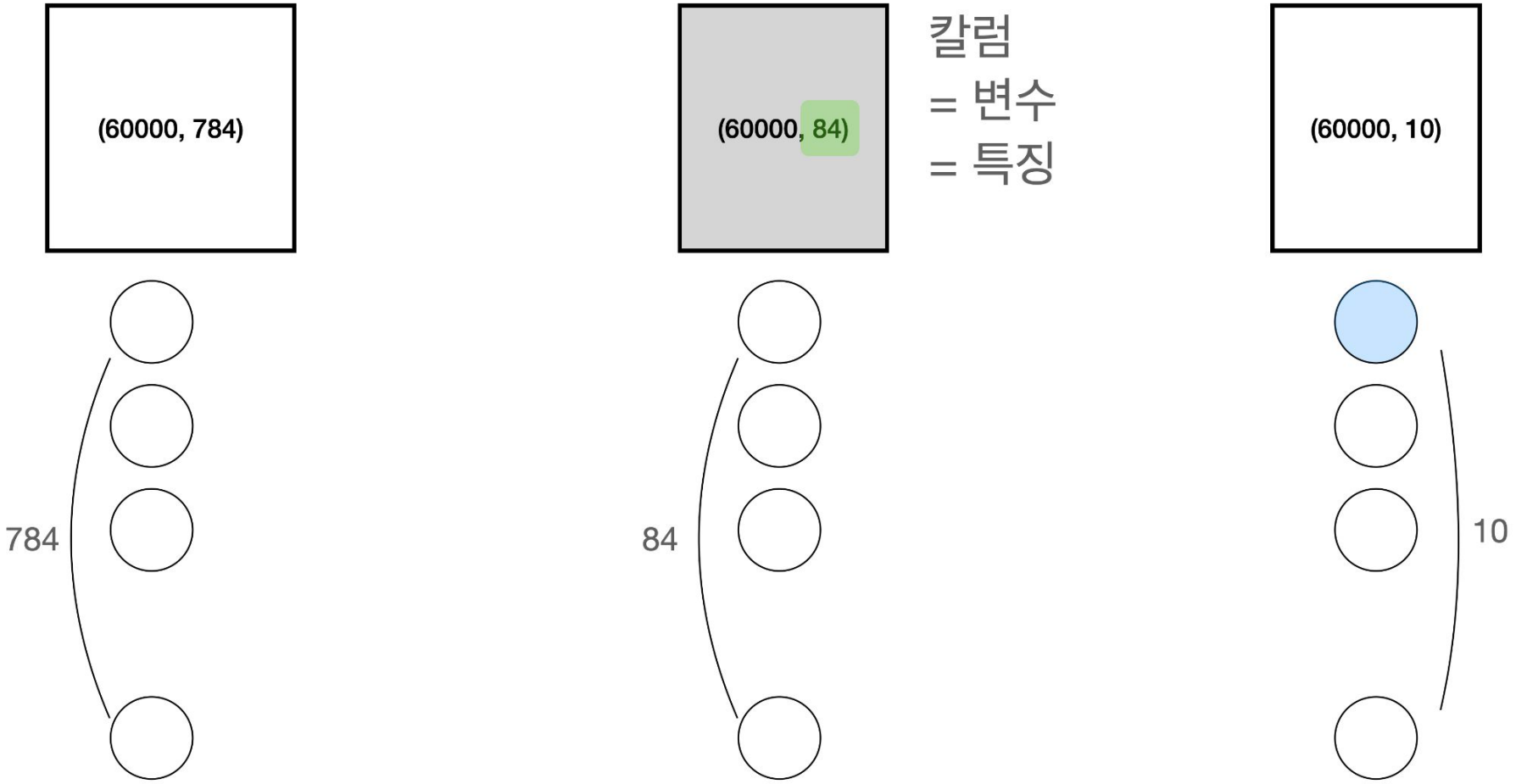


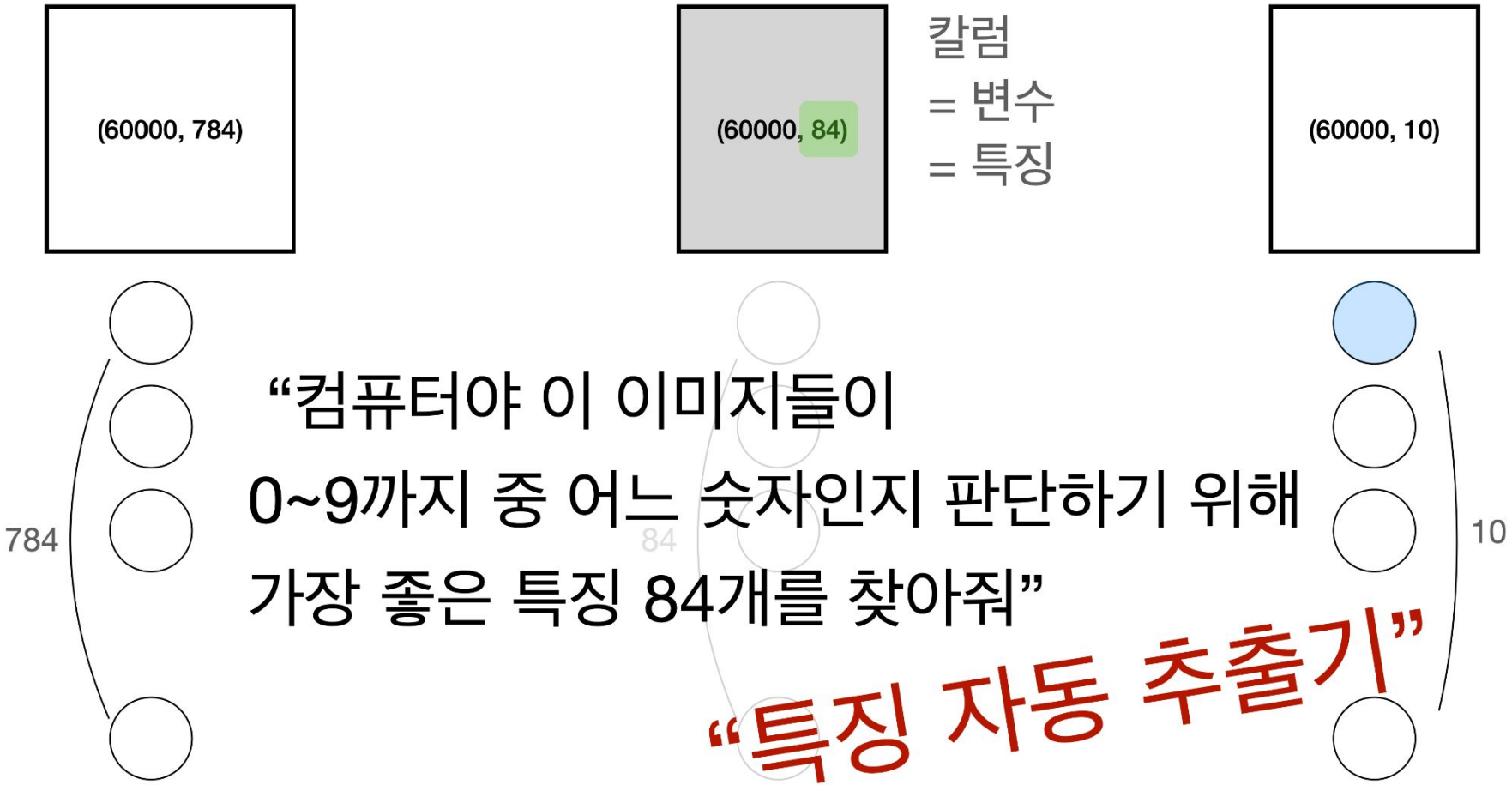

```
print(독립.shape)
# (60000, 28, 28)

독립.reshape(60000, 784)
print(독립.shape)
# (60000, 784)
```

784

1	2	3	…	28	29	30	31	…	56		…		757	758	759	…	784


(60000, 784)
(60000, 10)
가중치가 높다는 것은
해당 픽셀값이 매우 중요하다는 의미
784
10
가중치가 낮다는 것은
해당 픽셀값이 쓸모 없다는 의미

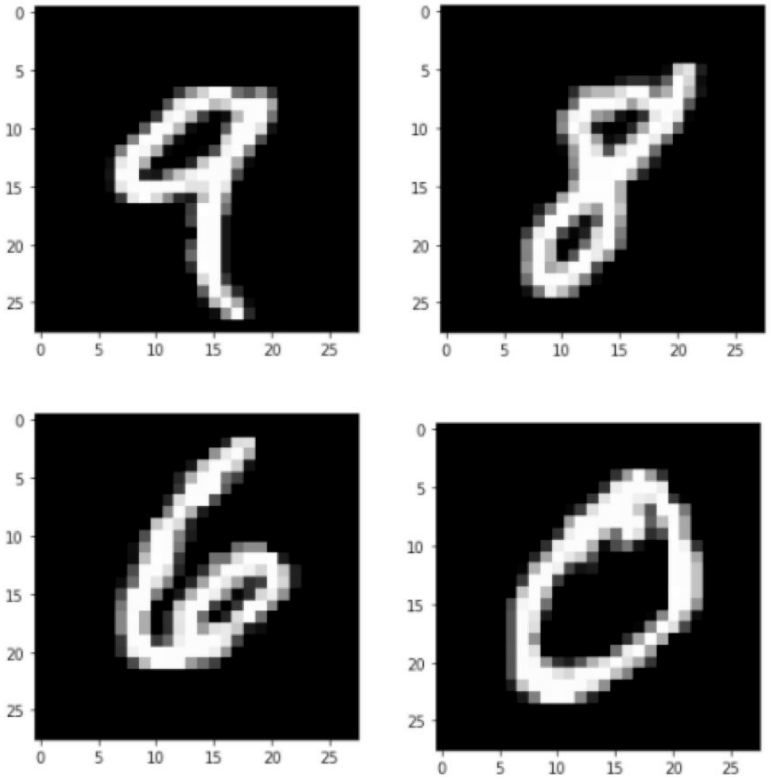

(60000, 784)
784
(60000, 84)
84
칼럼
= 변수
= 특징
(60000, 10)
10


(60000, 784)
(60000, 84)
칼럼
= 변수
= 특징
(60000, 10)
784
84
10
"컴퓨터야 이 이미지들이
0~9까지 중 어느 숫자인지 판단하기 위해
가장 좋은 특징 84개를 찾아줘"
"특징 자동 추출기"

```
# 1.과거의 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 784)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)

# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[ 784 ])
H = tf.keras.layers.Dense(84, activation='swish')(X)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')

# 3.데이터로 모델을 학습(FIT)합니다.
model.fit(독립, 종속, epochs=10)

# 4. 모델을 이용합니다
print("Predictions: ", model.predict(독립[0:5]))
```

```
# 1.과거의 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)
# (60000, 28, 28), (60000, 1)

# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[ 28, 28 ])
H = tf.keras.layers.Flatten()(X)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')

# 3.데이터로 모델을 학습(FIT)합니다.
model.fit(독립, 종속, epochs=10)

# 4. 모델을 이용합니다
print("Predictions: ", model.predict(독립[0:5]))
```

다섯번째 딥러닝 2 - Conv2D



숫자	전체	위	아래
9	1	1	0
8	2	1	1
6	1	0	1
0	1	0	0

특정한 패턴의 특징이

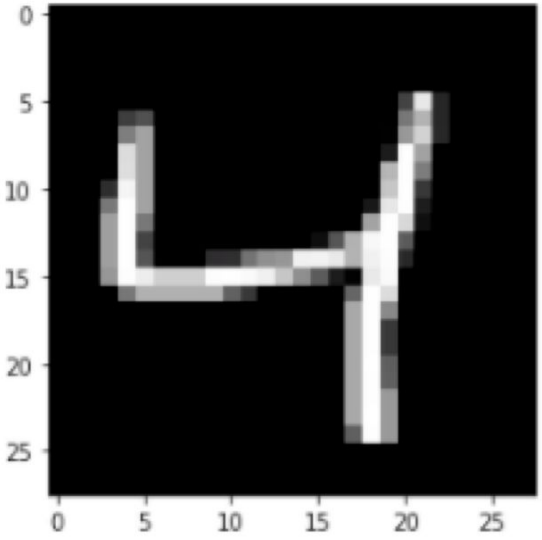
어디서 나타나는지를 확인하는 도구

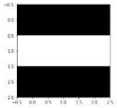
“Convolution”

특정한 패턴의 특징이

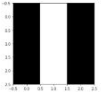
어디서 나타나는지를 확인하는 도구

“Convolution”

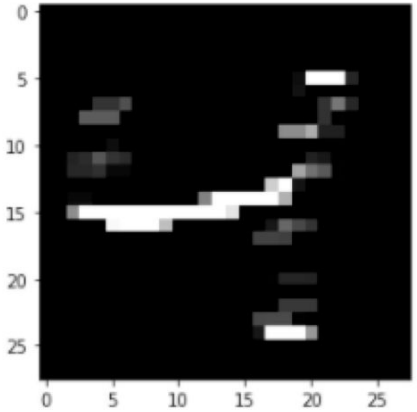


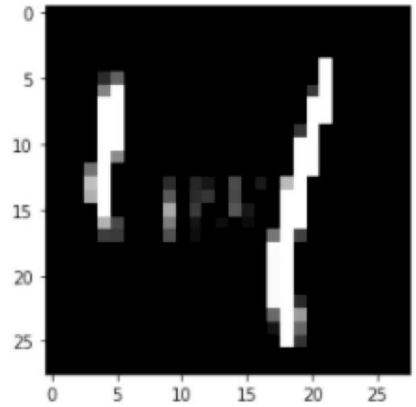


필터



특징맵 feature map





```
# 1.과거의 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 28, 28, 1)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)
```

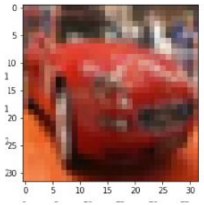
1. 필터를 몇 개 사용할 것인가
2. 필터의 사이즈를 얼마로 할 것인가

```
# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[28, 28, 1])
H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X)   # 3개의 특징맵 = 3채널의 특징맵
H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H)   # 6개의 특징맵 = 6채널의 특징맵
H = tf.keras.layers.Flatten()(H)   # 표로 만든다.
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')
```

필터의 이해

1. 필터는 3차원 형태로 된 가중치의 모음
2. 필터 하나는 앞선 레이어의 결과인 “특징맵” 전체를 본다.
3. 필터 개수 만큼 특징맵을 만든다.

```
Conv2D(3, kernel_size=5, activation='swish')
Conv2D(6, kernel_size=5, activation='swish')
```



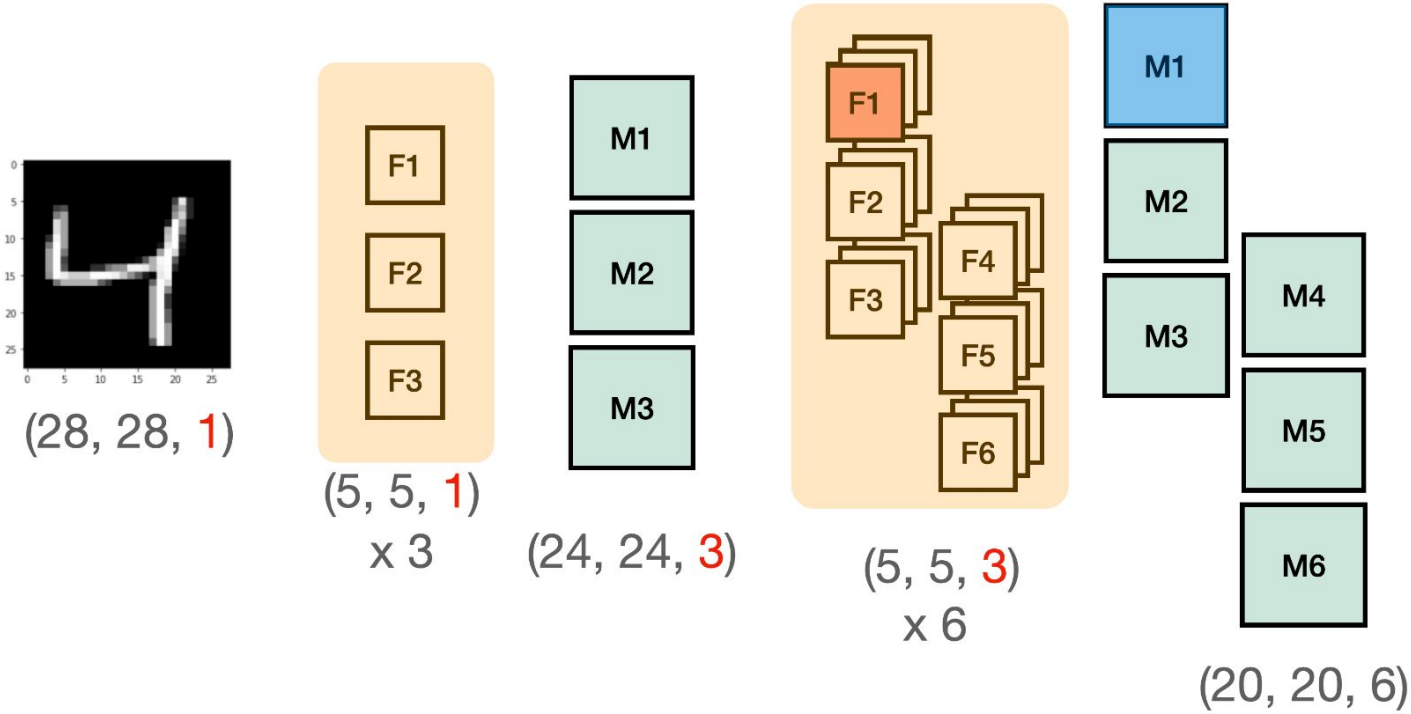
(32, 32, 3)

(5, 5, 3)

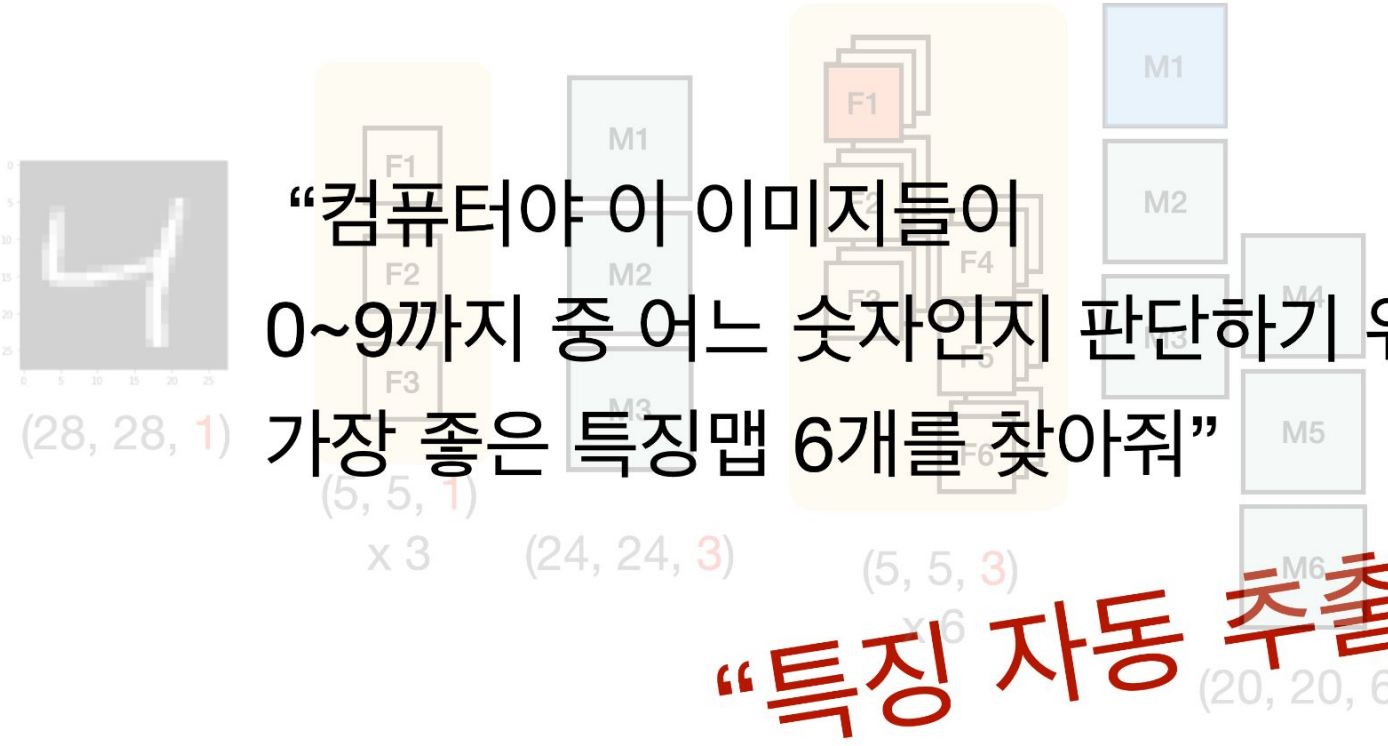
(3, 5, 5, ?)

(6, 5, 5, ?)

```
Conv2D(3, kernel_size=5, activation='swish')
Conv2D(6, kernel_size=5, activation='swish')
```



```
Conv2D(3, kernel_size=5, activation='swish')
Conv2D(6, kernel_size=5, activation='swish')
```



"컴퓨터야 이 이미지들이
0~9까지 중 어느 숫자인지 판단하기 위해
가장 좋은 특징맵 6개를 찾아줘"

"특징 자동 추출기"

(8, 8, 1)

x1	x2	x3	x4	x5	x6	x7	x8
x9	x10	x11	x12	x13	x14	x15	x16
x17	x18	x19	…				

(3, 3, 1)

w1	w2	w3
w4	w5	w6
w7	w8	W9

필터

(6, 6)

y1	y2	y3	y4	y5	y6
y7	y8	y9	y10	…	

```
y1 = x1*w1 + x2*w2 + x3*w3 +
     x9*w4 + x10*w5 + x11*w6 +
     x17*w7 + x18*w8 + x19*w9
```

(8, 8, 1)

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

(3, 3, 1)

-1	-1	-1
2	2	2
-1	-1	-1

필터

(6, 6)

1					

= 0 * -1 + 0 * -1 + 0 * -1 +
0 * 2 + 1 * 2 + 0 * 2 +
0 * -1 + 1 * -1 + 0 * -1

(8, 8, 1)

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

(3, 3, 1)

-1	-1	-1
2	2	2
-1	-1	-1

필터

(6, 6)

1	1				

= 0 * -1 + 0 * -1 + 0 * -1 +
1 * 2 + 0 * 2 + 0 * 2 +
1 * -1 + 0 * -1 + 0 * -1

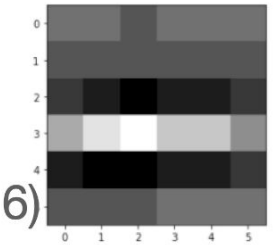
(8, 8, 1)

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

(3, 3, 1)

-1	-1	-1
2	2	2
-1	-1	-1

필터



(6, 6)

1	1	0	1	1	1
0	0	0	0	0	0
-1	-2	-3	-2	-2	-1
3	5	6	4	4	2
-2	-3	-3	-2	-2	-1
0	0	0	1	1	1

y = * -1 + * -1 + * -1 +
* 2 + * 2 + * 2 +
* -1 + * -1 + * -1

다섯번째 딥러닝 3 - MaxPool2D

```
X = tf.keras.layers.Input(shape=[28, 28])
H = tf.keras.layers.Flatten()(X)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')
```

$84 * (784 + 1) = 65940$

$10 * (84 + 1) = 850$

`model.summary()`

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28)]          0
_________________________________________________________________
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 84)                65940
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850
=================================================================
Total params: 66,790
Trainable params: 66,790
Non-trainable params: 0
_________________________________________________________________
```

```
X = tf.keras.layers.Input(shape=[28, 28, 1])
H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X)
H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H)
H = tf.keras.layers.Flatten()(H)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_cross
```

```
model.summary()
```

$84 * (2400 + 1) = 201{,}684$

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 24, 24, 3)         78
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 20, 20, 6)         456
_________________________________________________________________
flatten (Flatten)            (None, 2400)              0
_________________________________________________________________
dense (Dense)                (None, 84)                201684
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850
=================================================================
Total params: 203,068
Trainable params: 203,068
Non-trainable params: 0
_________________________________________________________________
```

```
X = tf.keras.layers.Input(shape=[28, 28, 1])

H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X)
H = tf.keras.layers.MaxPool2D()(H)
H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H)
H = tf.keras.layers.MaxPool2D()(H)

H = tf.keras.layers.Flatten()(H)
H = tf.keras.layers.Dense(84, activation='swis
Y = tf.keras.layers.Dense(10, activation='soft
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy'
```

```
model.summary()
```

WOW!

$84 * (96 + 1) = 8{,}148$

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 3)         78
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 3)         0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 6)           456
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 6)           0
_________________________________________________________________
flatten_1 (Flatten)          (None, 96)                0
_________________________________________________________________
dense_2 (Dense)              (None, 84)                8148
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850
=================================================================
Total params: 9,532
Trainable params: 9,532
Non-trainable params: 0
_________________________________________________________________
```

10	7	2	42	30	25
1	16	33	8	7	6
9	22	11	45	10	22
13	15	31	27	12	41
7	20	19	30	23	6
3	24	15	27	14	40

16		

10	7	2	42	30	25
1	16	33	8	7	6
9	22	11	45	10	22
13	15	31	27	12	41
7	20	19	30	23	6
3	24	15	27	14	40

16	42	

나는 특징맵

10	7	2	42	30	25
1	16	33	8	7	6
9	22	11	45	10	22
13	15	31	27	12	41
7	20	19	30	23	6
3	24	15	27	14	40

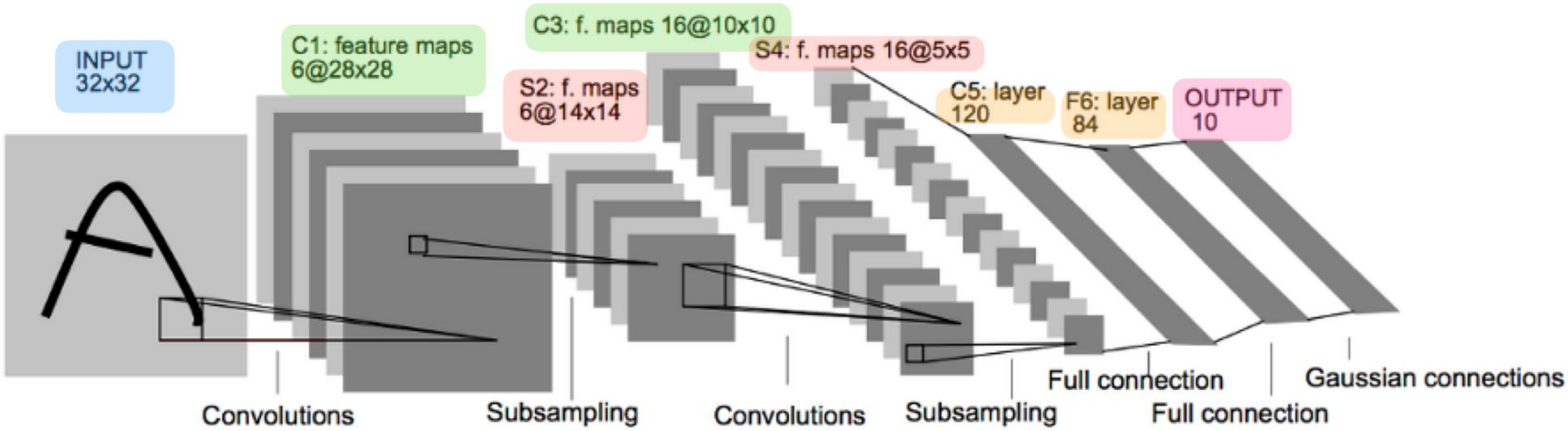
MaxPooling

16	42	30
22	45	41
24	30	40

값이 크다 = 필터로 찾으려는 특징이 많이 나타난 부분

```
# 1.과거의 데이터를 준비합니다.
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 28, 28, 1)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)

# 2. 모델의 구조를 만듭니다
X = tf.keras.layers.Input(shape=[28, 28, 1])

H = tf.keras.layers.Conv2D(3, kernel_size=5, activation='swish')(X)
H = tf.keras.layers.MaxPool2D()(H)
H = tf.keras.layers.Conv2D(6, kernel_size=5, activation='swish')(H)
H = tf.keras.layers.MaxPool2D()(H)

H = tf.keras.layers.Flatten()(H)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')
```

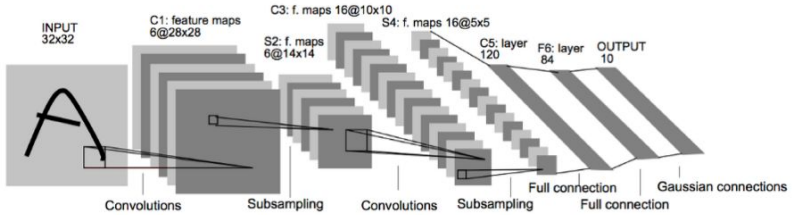
다섯번째 딥러닝 완성 - LeNet-5



CNN called LeNet by Yann LeCun (1998)

1.과거의 데이터를 준비합니다.

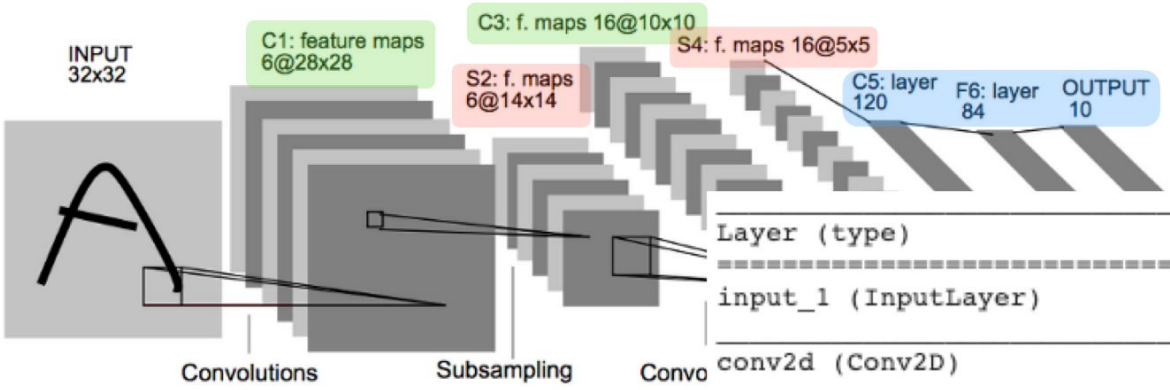
```
(독립, 종속), _ = tf.keras.datasets.mnist.load_data()
독립 = 독립.reshape(60000, 28, 28, 1)
종속 = pd.get_dummies(종속)
print(독립.shape, 종속.shape)
```



CNN called LeNet by Yann LeCun (1998)

2. 모델의 구조를 만듭니다

```
X = tf.keras.layers.Input(shape=[28, 28, 1])
H = tf.keras.layers.Conv2D(6, kernel_size=5, padding='same', activation='swish')(X)
H = tf.keras.layers.MaxPool2D()(H)
H = tf.keras.layers.Conv2D(16, kernel_size=5, activation='swish')(H)
H = tf.keras.layers.MaxPool2D()(H)

H = tf.keras.layers.Flatten()(H)
H = tf.keras.layers.Dense(120, activation='swish')(H)
H = tf.keras.layers.Dense(84, activation='swish')(H)
Y = tf.keras.layers.Dense(10, activation='softmax')(H)
model = tf.keras.models.Model(X, Y)
model.compile(loss='scategorical_crossentropy', metrics='accuracy')
```



CNN called LeNet by

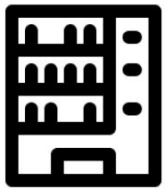
Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 28, 28, 1)]	0
conv2d (Conv2D)	(None, 28, 28, 6)	156
max_pooling2d (MaxPooling2D)	(None, 14, 14, 6)	0
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48120
dense_1 (Dense)	(None, 84)	10164
dense_2 (Dense)	(None, 10)	850

Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0

차원

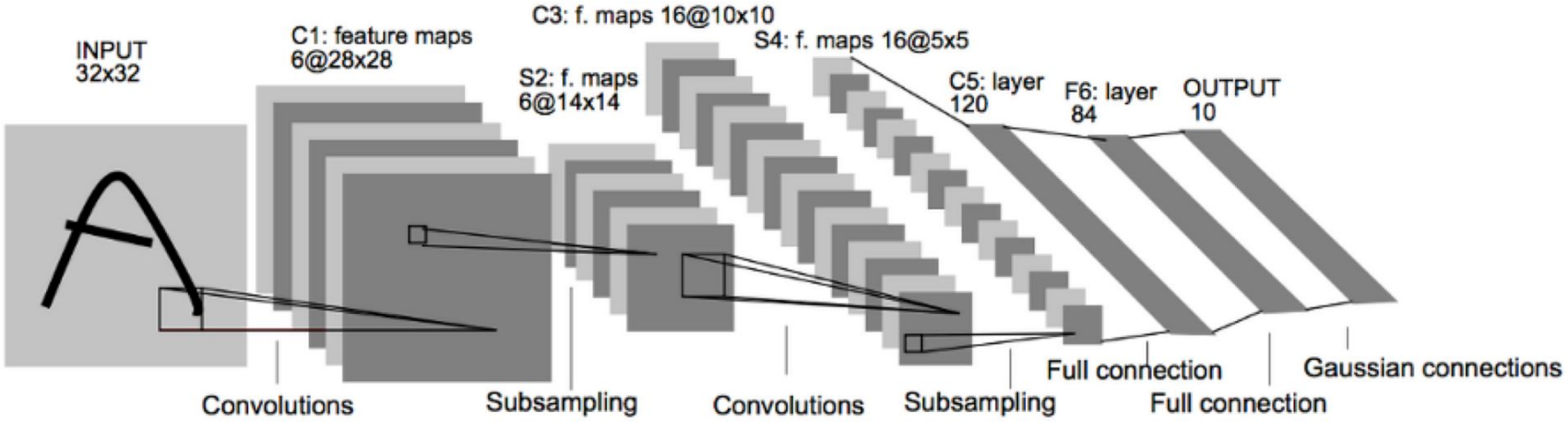
표의 열 vs 포함 관계

- 데이터 공간의 맥락
  차원수 = 변수의 개수
- 데이터 형태의 맥락
  차원수 = 배열의 깊이



“특징 자동 추출기”

- **기본 딥러닝 모델**: weight를 학습해서 최적의 특징을 만든다.
- **CNN 구조 딥러닝 모델**: filter를 학습해서 최적의 특징맵을 만든다.



CNN called LeNet by Yann LeCun (1998)