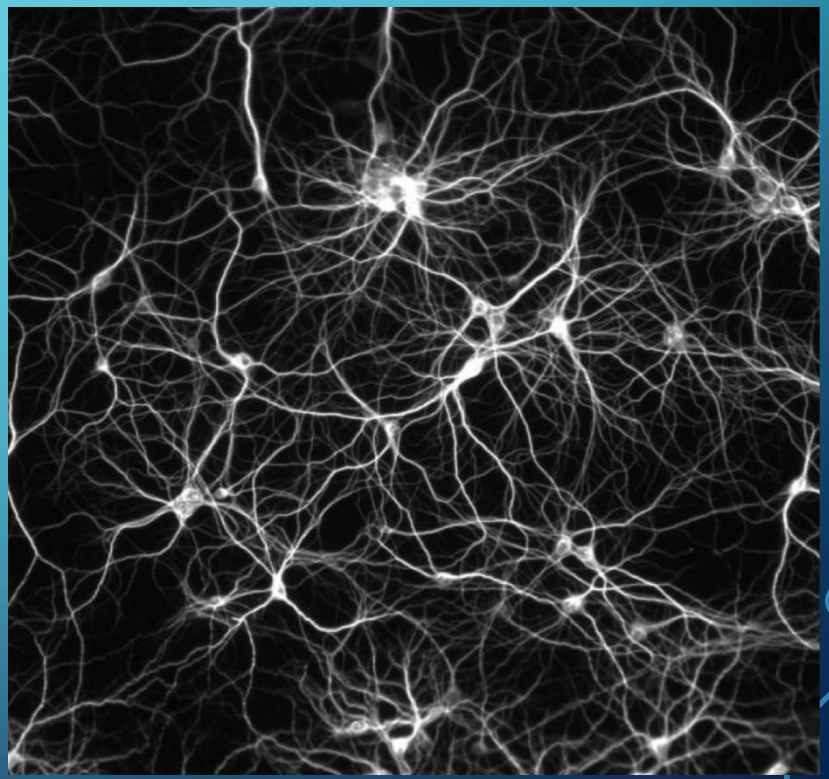
DEEP LEARNING WITH KERAS

DNN – BOSTON HOUSING

HUMAN BRAIN

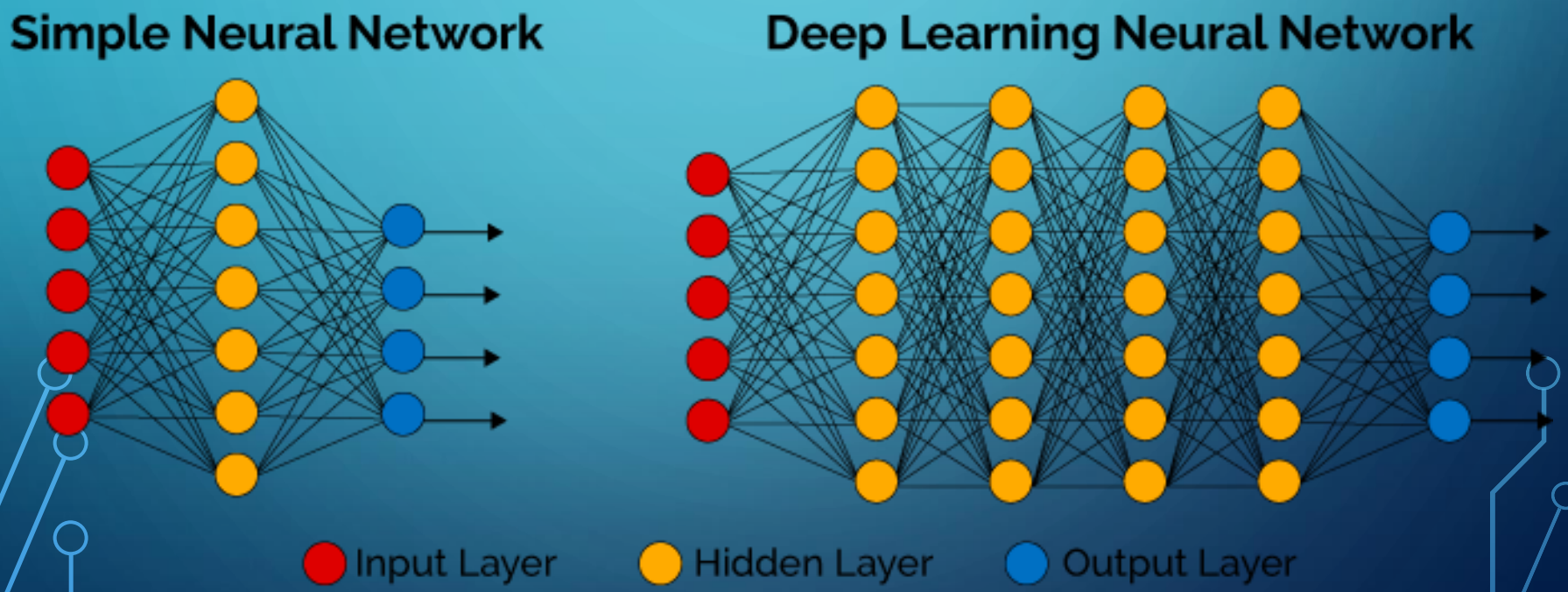
- Amazing structure



천억개의 뉴런들이 백조개의 시냅스로 연결

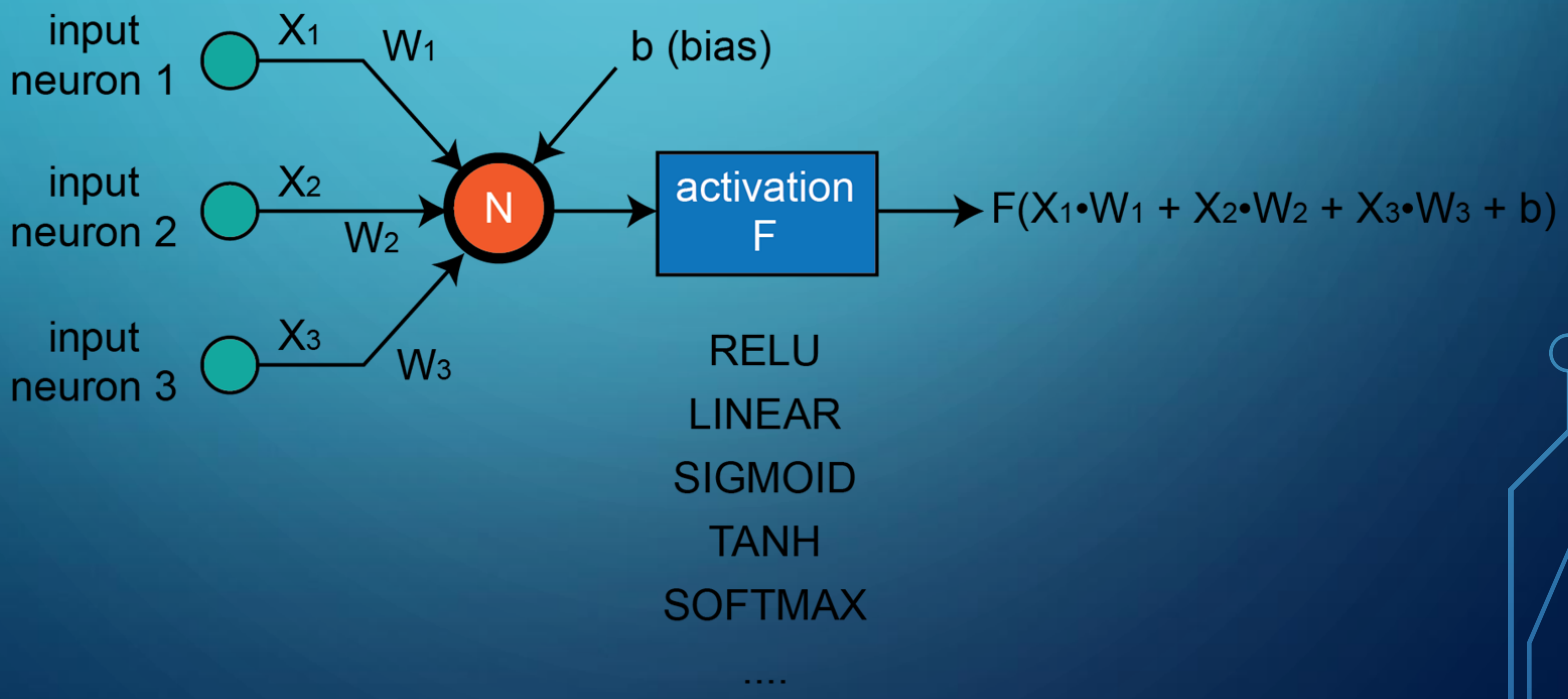
DEEP NEURAL NETWORK

- DNN simply has more hidden layers than ANN
  - More weights to tune
  - Becomes more powerful



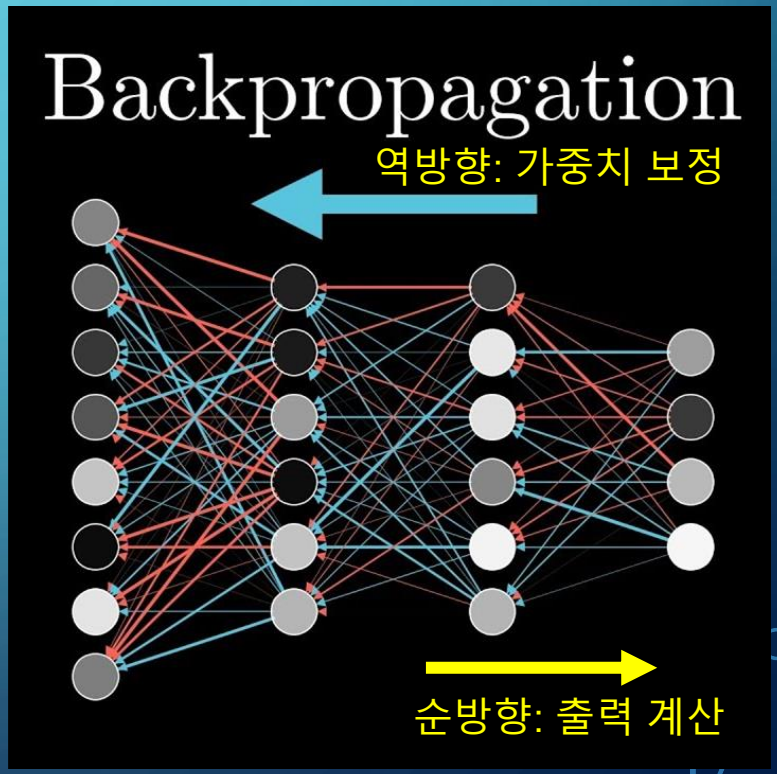
HOW TO CALCULATE OUTPUT OF NEURON

- 사용된 변수
  - X_K: 입력 뉴런 K의 출력값
  - W_K: 시냅스 K의 가중치
  - b: 편향값



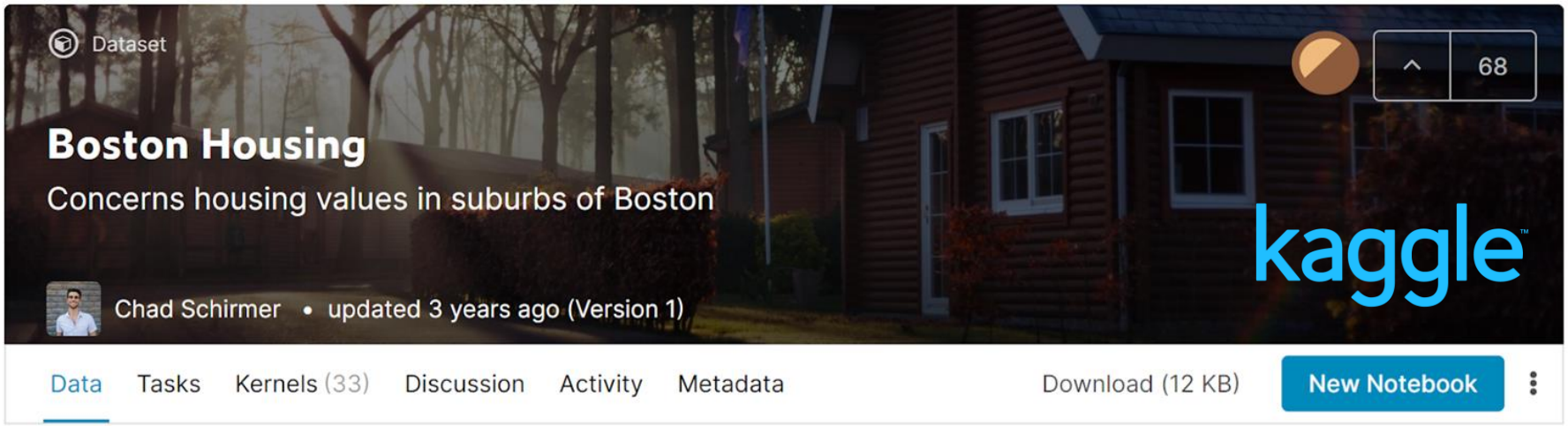
BACKPROPAGATION ALGORITHM

- 보편적인 인공신경망 학습 알고리즘
  - 지도 학습에 사용
- 순방향(출력 계산)
  - 입력값이 주어지면 출력값 계산은 왼쪽에서 오른쪽으로 (= forward)
  - 각각의 뉴런에 계산값 저장
- 역방향(가중치 보정)
  - 출력값이 주어지면 신경망의 가중치 보정은 오른쪽에서 왼쪽으로 (= backward)
  - 순방향 계산 결과를 활용



BOSTON MONTHLY RENT

BOSTON HOUSING DATASET



- 보스톤 집값 정보 데이터
  - Kaggle에서 제공 (www.kaggle.com)
  - 1978년 보스톤 교외 506 동네의 평균 집값 정보
  - 각 동네 마다 집값에 영향을 미치는 14개의 요소를 제공

BOSTON HOUSING DATASET

- 506 rows and 14 columns
  - CRIM: 범죄율
  - ZN: 25,000 평방피트를 (약 700평) 초과하는 거주지역 비율
  - INDUS: 비소매 상업지역 면적 비율
  - CHAS: 찰스강의 경계에 위치한 경우는 1, 아니면 0
  - NOX: 일산화질소 농도
  - RM: 주택당 방 수
  - AGE: 1940년 이전에 건축된 주택의 비율

집 값

0.58	20	3.9	0	0.57	8.29	67	2.42	5	264	13	384	7.44	50
38.3	0	18.1	0	0.69	5.45	100	1.48	24	666	20.2	396	30.5	5

BOSTON HOUSING DATASET

- 506 rows and 14 columns
  - DIS: 직업센터의 거리
  - RAD: 방사형 고속도로까지의 거리
  - TAX: 재산세율
  - PTRATIO: 학생/교사 비율
  - B: 인구 중 흑인 비율
  - LSTAT: 인구 중 하위 계층 비율
  - MEDV: 주택 가격 ($1,000)

집 값

0.58	20	3.9	0	0.57	8.29	67	2.42	5	264	13	384	7.44	50
38.3	0	18.1	0	0.69	5.45	100	1.48	24	666	20.2	396	30.5	5

HOUSING.PY (1/11)

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

from keras.models import Sequential
from keras.layers import Dense

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

HOUSING.PY (2/11)

```
# hyper-parameters
MY_EPOCH = 500
MY_BATCH = 32



# read DB file
# it returns a pandas data frame (= 2 dimensional array)
heading = ['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age
', 'dis', 'rad', 'tax', 'ptratio', 'black', 'lstat', 'medv'
]

file_name = "housing.csv"
raw_DB = pd.read_csv(file_name, delim_whitespace = True,
        names = heading)
```

HOUSING.PY (3/11)

```
# print raw data stats with describe()
print('\n== FIRST 20 RAW DATA ==')
print(raw_DB.head(5))
print(raw_DB.describe())
```

HOUSING.PY (4/11)

```
# scaling with z-score: z = (x - u) / s
# mean becomes 0, and standard deviation 1
# it returns a numpy array
scaler = StandardScaler()
scaled_DB = scaler.fit_transform(raw_DB)

# framing numpy array into pandas data frame
# this is needed after scaling to use describe()
scaled_DB = pd.DataFrame(scaled_DB, columns = heading)
print(raw_DB.head(5))
print(raw_DB.describe())
```

Z-SCORE NORMALIZATION

- Z-점수 = (X – 평균) / 표준편차
  - 모든 Z-점수에 대해 평균은 0, 표준편차는 1(=표준정규분포)

- 활용 예
  - A는 SAT 시험에서 1800 (out of 2400) 을 받음
  - B는 ACT 시험에서 24 (out of 36) 을 받음
  - SAT 평균은 1500, 표준편차는 300
  - ACT 평균은 18, 표준편차는 5
  - A의 Z-점수는 1, B는 1.2. 결국 B가 더 잘했음.

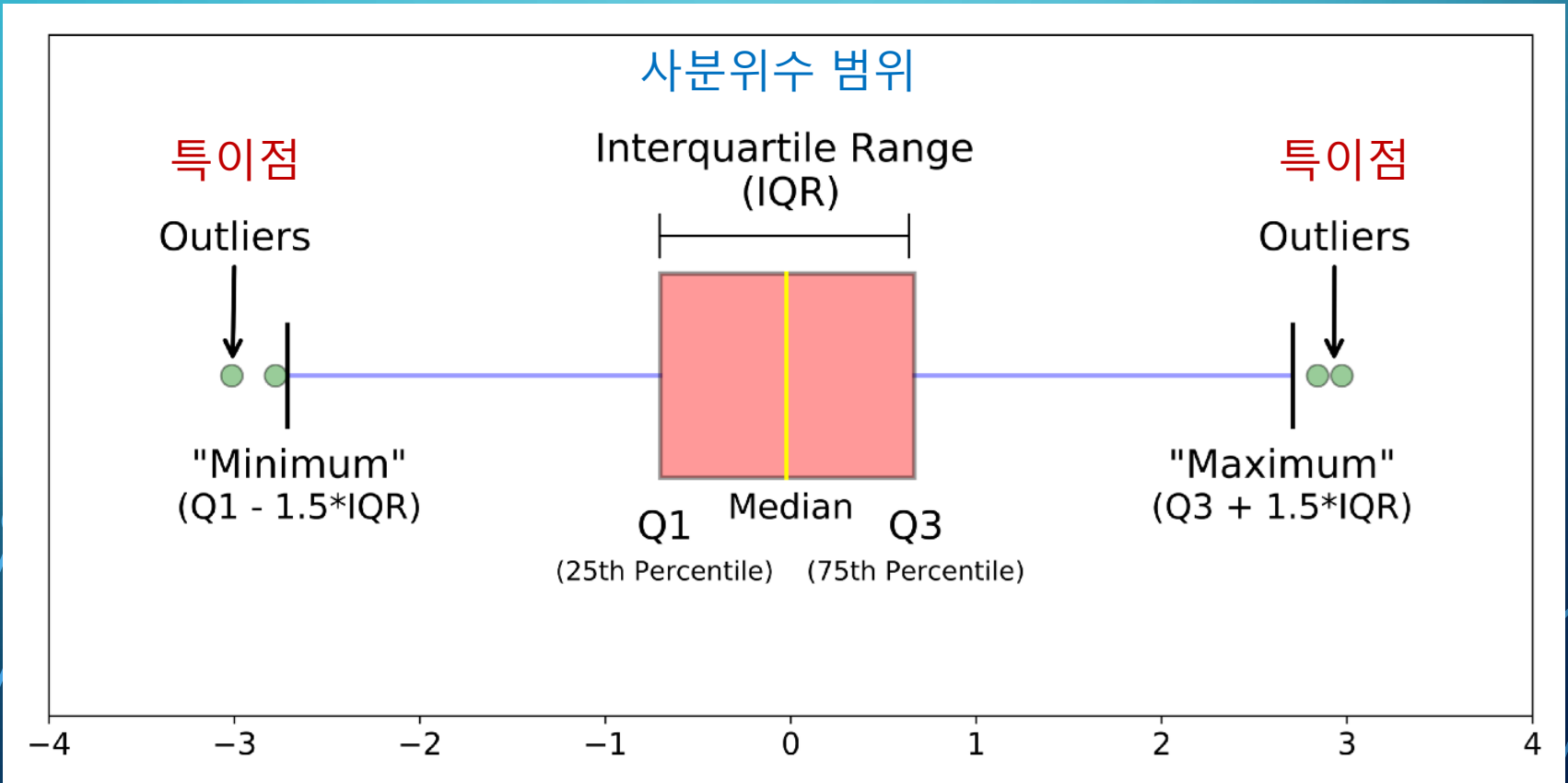
- fit_transform() 함수는 사용된 평균과 표준편차를 기억
  - 나중에 Z-점수를 원래 점수로 역전환 할때 사용

HOUSING.PY (5/11)

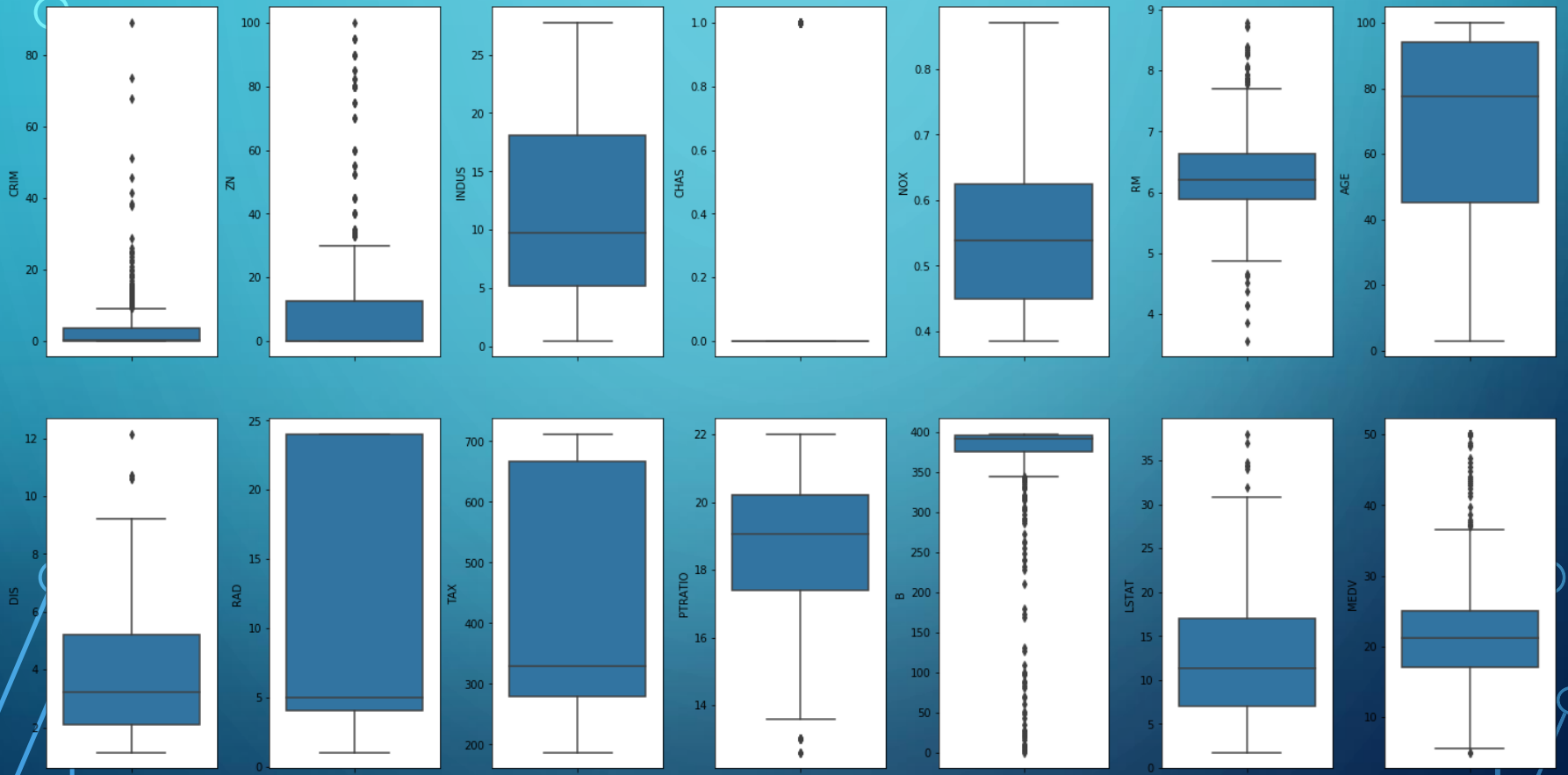
```
# display box plot of scaled DB
boxplot = scaled_DB.boxplot(column = heading)
print('\n== BOX PLOT OF SCALED DATA ==')
plt.show()
```

BOX PLOT

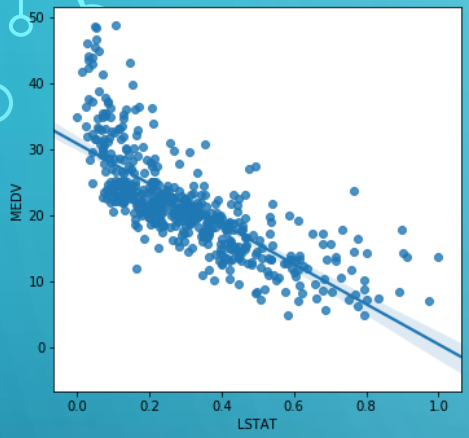
- 데이터의 주요 특징 값을 시각적으로 표현
  - 사분위수 (Q1, 중앙값, Q3)와 최소값, 최대값

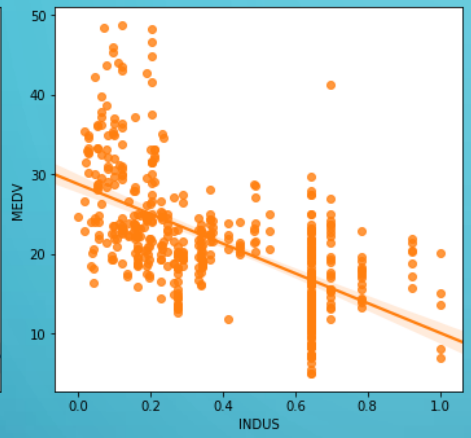


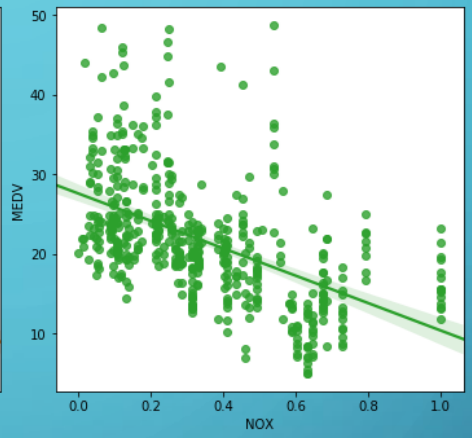
DISTRIBUTIONS

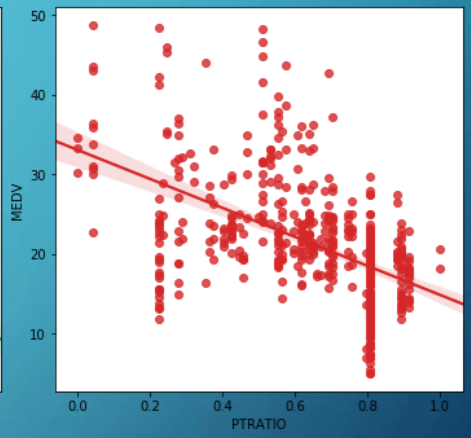


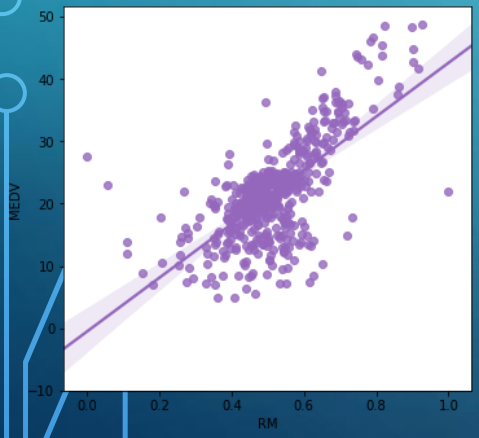
SCATTER PLOT

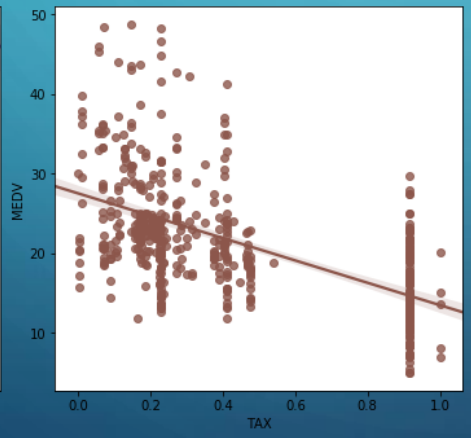


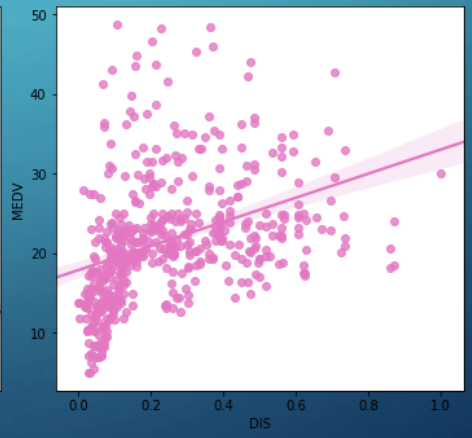


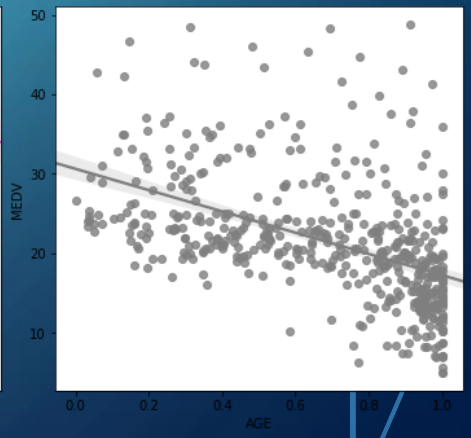












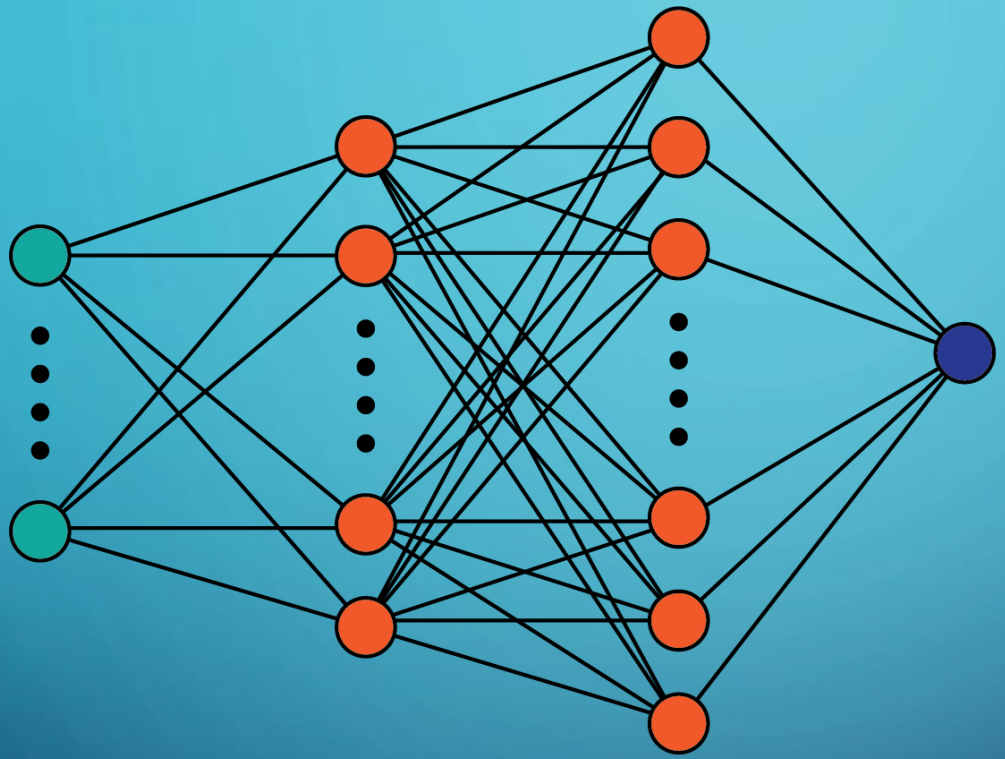
HOUSING.PY (6/11)

```
# split the DB into inputs and label
# (506, 14) becomes (506, 13) and (506,)
print('\n== DB SHAPE INFO ==')
print('DB shape = ', scaled_DB.shape)

X = scaled_DB.drop('medv', axis = 1)
print('X (= input) shape = ', X.shape)

Y = scaled_DB['medv']
print('Y (= output) shape = ', Y.shape)
```

HOUSING.PY (7/11)

```
# split the DB into training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y,
        test_size = 0.30, random_state = 5)

print('\nX train shape = ', X_train.shape)
print('X test shape = ', X_test.shape)
print('Y train shape = ', Y_train.shape)
print('Y test shape = ', Y_test.shape)
```

DNN 설계도



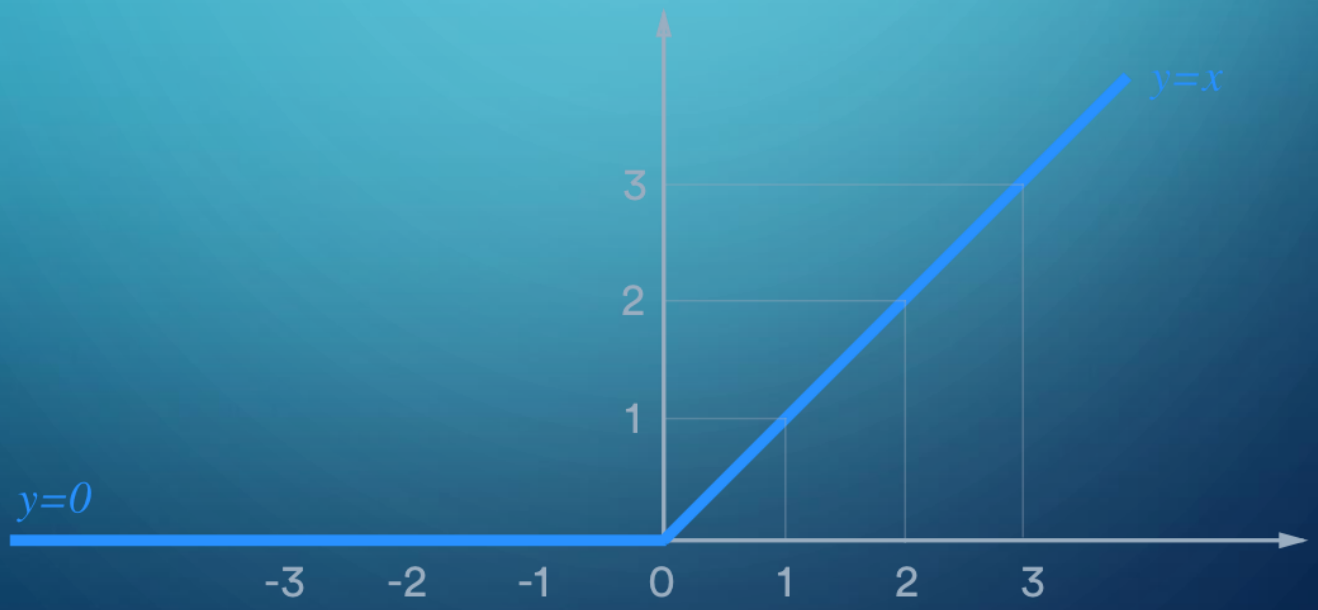
- 4층짜리 DNN이 Dense Layer로 연결된 구조
- 총 1214개 뉴런
- 총 204,801개의 보정치

HOUSING.PY (8/11)

```
    ###################################
    # MODEL BUILDING AND TRAINING #
    ###################################


# build a keras sequential model of our DNN
model = Sequential()
model.add(Dense(200, input_dim = 13, activation = 'relu'))
model.add(Dense(1000, activation = 'relu'))
model.add(Dense(1, activation = 'linear'))
model.summary()
```

RELU ACTIVATION

- Rectified linear unit (ReLU)
  - F(x) = max(0, x)
  - Most commonly used activation function, especially in CNNs
  - No vanishing gradient problem as in sigmoid or tanh
  - Many neurons are not activated at all: this is often desirable

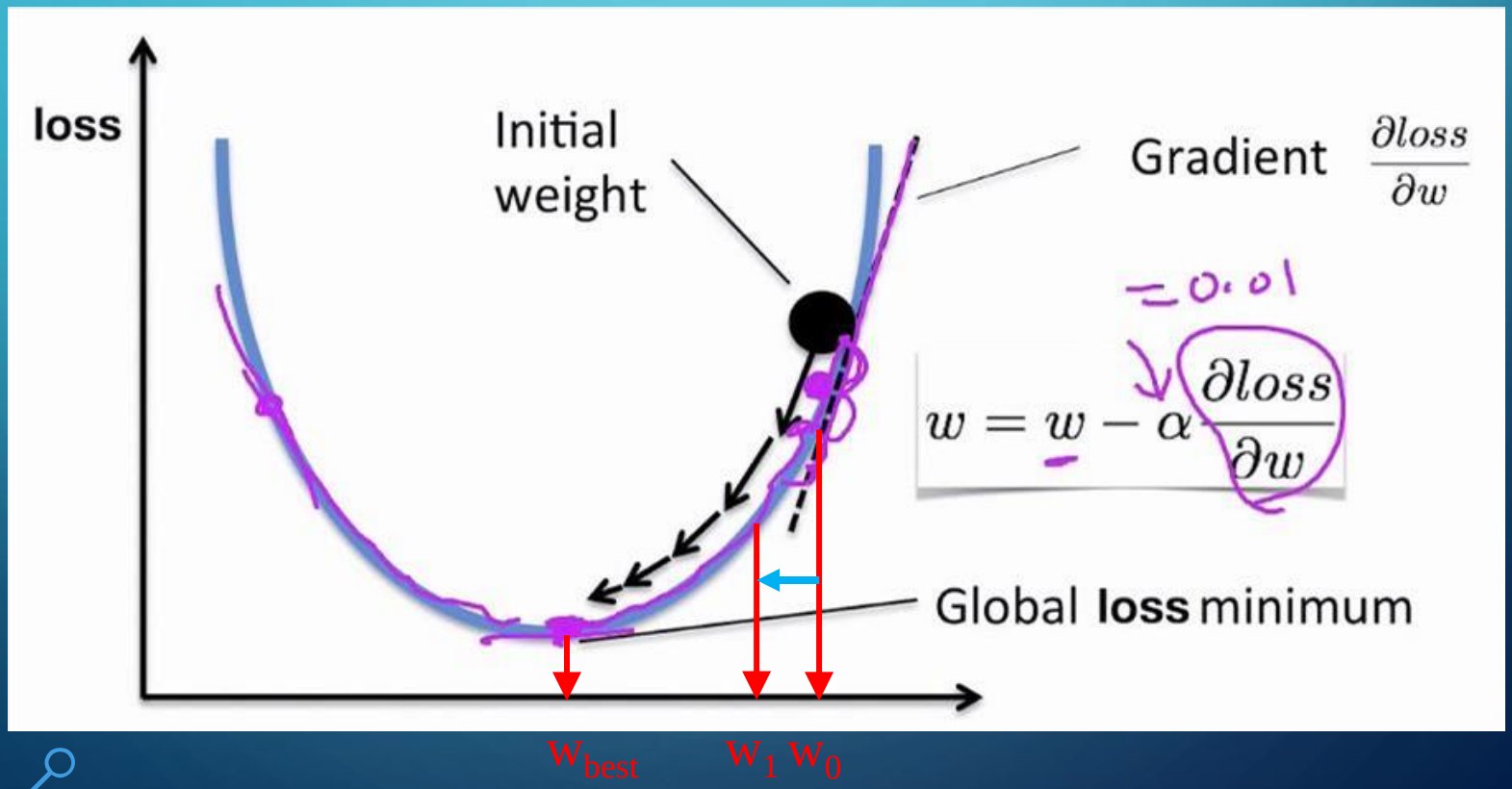


HOUSING.PY (9/11)

```
# model training and saving
model.compile(optimizer = 'sgd', loss = 'mse',
        metrics = ['accuracy'])

model.fit(X_train, Y_train, epochs = MY_EPOCH,
        batch_size = MY_BATCH, verbose = 1)
```

STATISTICAL GRADIENT DESCENT

- Popular way to update W and b in ML
  - We move towards the negative of the gradient of the loss at the current po int



HOUSING.PY (10/11)

```
    ########################
    # MODEL EVALUATION #
    ########################


# model evaluation
loss, acc = model.evaluate(X_test, Y_test, verbose = 1)
print('\nMSE of DNN model', loss)
print('Model accuracy', acc)
```

HOUSING.PY (11/11)

```
# plot keras DNN modeling result
pred = model.predict(X_test)
plt.scatter(Y_test, pred)
plt.xlabel("Actual values")
plt.ylabel("Predicted values")
plt.title("Keras DNN Model")
plt.show()
```

PREDICTION VS. ACTUAL VALUES