07_01_TensorFlow

1. TensorFlow ?

- Google이 만든 Machine Learning Library
- Open Source Library
- 수학적 계산을 하기 위한 Library
- Data Flow Graph를 이용
- TensorFlow는 Node와 Edge로 구성된 방향성 있는 Graph
  - Node : 데이터의 입출력과 수학적 계산
  - Edge : Tensor를 Node로 실어 나르는 역할
  - Tensor : 동적 크기의 다차원 배열을 지칭

1.1 TensorFlow 설치

1.1.1 CPU 버전

- Python에서 설치

```
$ pip install tensorflow==2.0
```

- Anaconda에서 설치

```
$ conda install tesnsorflow==2.0
```

1.1.2 GPU 버전

1. https://developer.nvidia.com/cuda-toolkit-archive에서 CUDA 10.0 다운로드 및 설치
2. https://developer.nvidia.com/rdp/cudnn-download에서 for CUDA 10.0 다운로드 및 압축 해제
   - 압축 해제된 CUDA에서 lib, include, bin 폴더 등의 파일을 전체 복사
   - 1.에서 설치된 경로에 붙여넣기
3. 환경 변수 확인
   - 잘 되어있지 않다면 아까 추가한 lib, include, bin 폴더의 경로 추가
4. tensorflow-gpu 설치
   - Python에서

```
$ pip install tensorflow-gpu==2.0

# 업그레이드 시
$ pip install --upgrade tensorflow-gpu==2.0
```

   - Anaconda에서

```
$ conda install tensorflow-gpu==2.0
```

2. TensorFlow 기초

2.1 출력

- Node는 숫자 연산과 데이터 입출력을 담당

```
my_node = tf.constant("Hello World")
print(my_node.numpy().decode())   # 입력한 데이터 출력 .decode()
```

2.2 constant

- 선언과 동시에 초기화

```
node1 = tf.constant(10, dtype = tf.float32)
node2 = tf.constant(20, dtype = tf.float32)

node3 = node1 + node2

print(node3.numpy())
print([node1.numpy(), node2.numpy(), node3.numpy()])
```

```
30.0
[10.0, 20.0, 30.0]
```

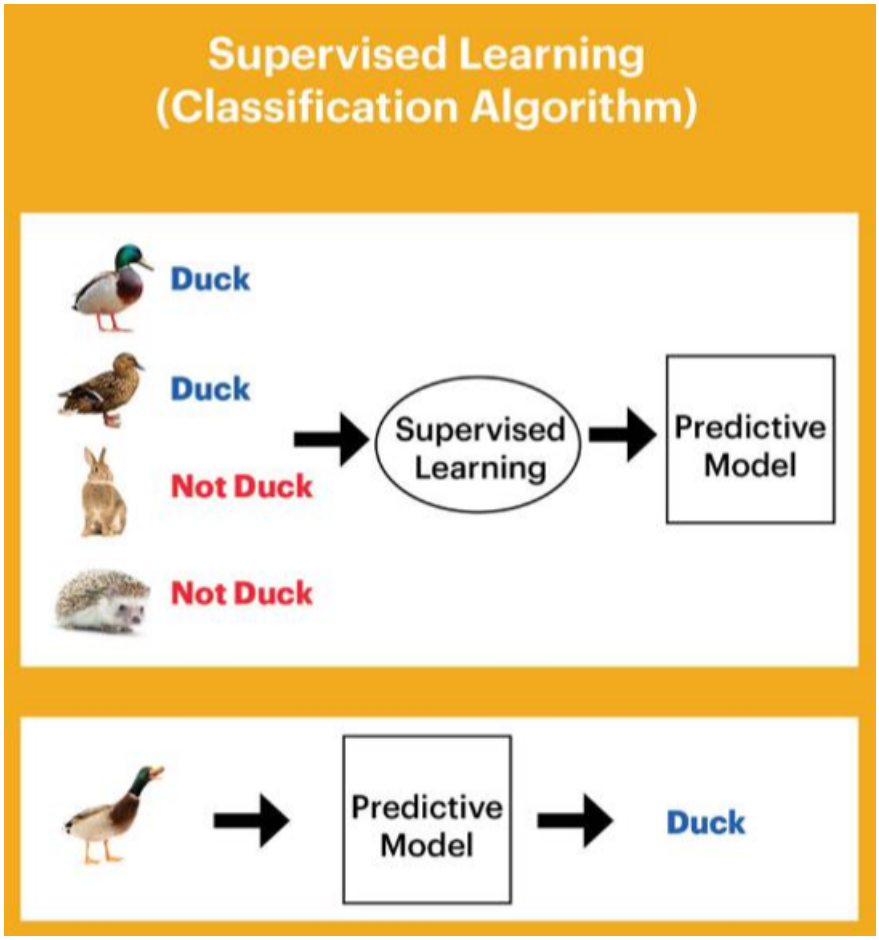
3. Machine Learning

- 프로그램 자체가 **데이터를 기반으로 학습**을 통해 배우는 능력을 가지는 프로그래밍

3.1 Learning의 종류

3.1.1 Supervised Learning(지도 학습)

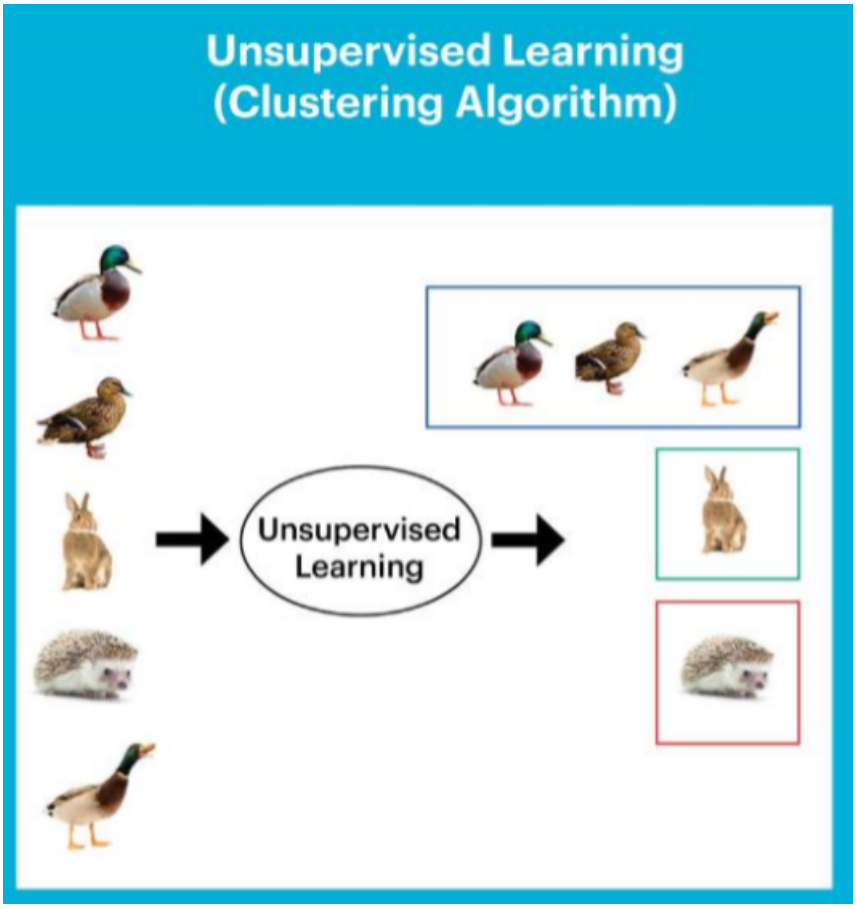
- Training Set이라고 불리는 Label화 된 데이터를 통해 학습



- Linear Regression(선형 회귀) - 공부 시간 : 시험 점수
- Logistic Regression(로지스틱 회귀)
    - Binary Classification(이항 분류) - 공부 시간 : 합격/불합격
    - Multinomial Classification(다향 분류) - 공부 시간 : 학점

3.1.2 Unsupervised Learning(비지도 학습)

- Label화 되지 않은 데이터를 통해 학습
- 데이터를 이용해 스스로 학습



4. Linear Regression

- Linear Regression의 가장 큰 목표는 가설의 완성

$$가설(Hypothesis) = Wx + b$$

4.1 Training Data Set 준비

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random

x = [1, 2, 3]
y = [2, 4, 6]
```

4.2 Weight(W) & Bias(b) 준비

```
W = tf.Variable(random.random(), name="weight")
b = tf.Variable(random.random(), name="bias")
```

- Hypothesis(가설)
  - 최종 목적은 Training Data에 가장 근접한 Hypothesis를 만드는 것(W와 b를 결정)
  - 잘 만들어진 가설은 W가 1에 b가 0에 가까워야 함

4.3 Cost(loss) Function

$$cost(W, b) = \frac{1}{n}\sum_{i=1}^{n}(H(x^i) - y^i)^2$$

- cost 함수 선언

```
def compute_cost():
    H = W * x + b
    cost = tf.reduce_mean(tf.square(H - y))
    return cost
```

- **Cost Function Minimize**

```
optimizer = tf.optimizers.Adam(learning_rate = 0.01)
```

4.4 Training

```
# 학습 진행
for step in range(3000):
    optimizer.minimize(compute_cost, var_list=[W, b])
    if step % 300 == 0:
        print("{}, {}, {}".format(W.numpy(), b.numpy(), compute_cost().numpy()))
```

4.5 Prediction

```
feed_x = 8
predict_y = W * feed_x + b

print(predict_y.numpy())
```

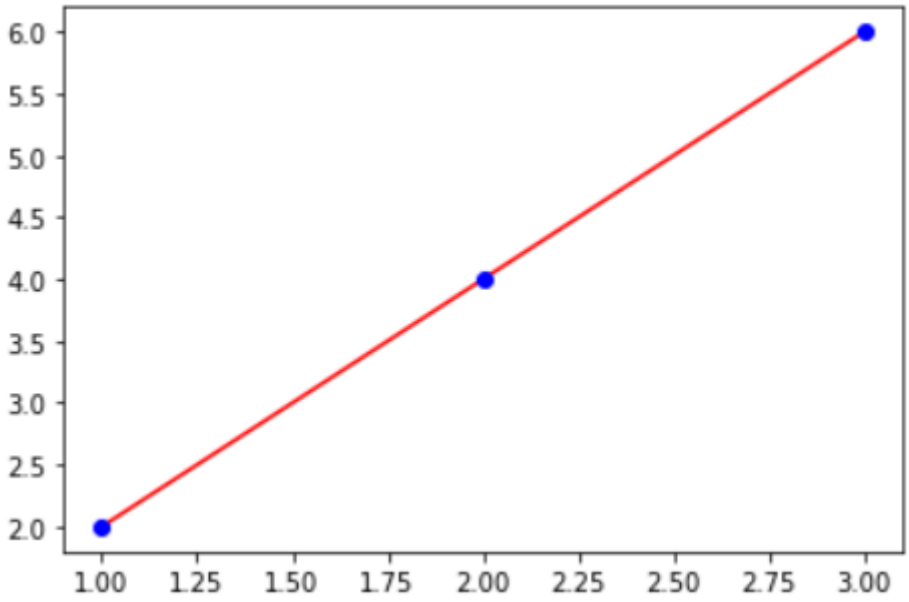
4.5 그래프

- 그래프 범위

```
line_x = np.arrange(min(x), max(x), 0.01)
line_y = W * line_x + b
```

- 그래프 그리기

```
plt.plot(line_x, line_y, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```



4.5 소스

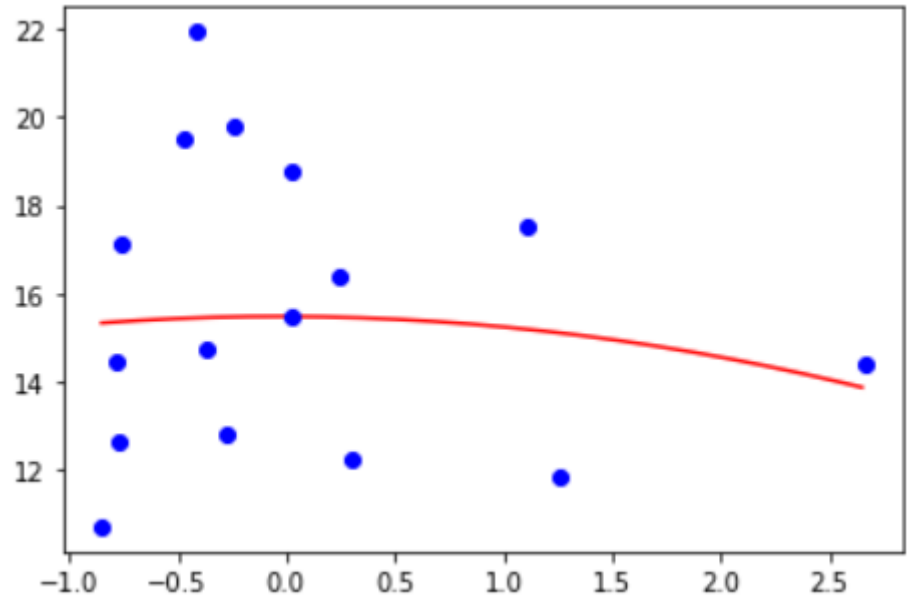
```
x = [1, 2, 3]
y = [2, 4, 6]

W = tf.Variable(random.random(), name="weight")
b = tf.Variable(random.random(), name="bias")

def compute_cost():
    H = W * x + b
    cost = tf.reduce_mean(tf.square(H - y))
    return cost

optimizer = tf.optimizers.Adam(learning_rate = 0.01)

for step in range(3000):
    optimizer.minimize(compute_cost, var_list=[W, b])
    if step % 300 == 0:
        print("{}, {}, {}".format(W.numpy(), b.numpy(), compute_cost().numpy()))

line_x = np.arrange(min(x), max(x), 0.01)
line_y = W * line_x + b

plt.plot(line_x, line_y, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```

5. 다항 회귀

5.1 2차

```
x = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85,
-0.41, -0.27, 0.02, -0.76, 2.66]
y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74,
10.72, 21.94, 12.83, 15.51, 17.14, 14.42]

W = tf.Variable(random.random(), name="weight")
```

```
b = tf.Variable(random.random(), name="bias")
c = tf.Variable(random.random(), name="c")

def compute_cost():
    H = W * x*x + b * x + c
    cost = tf.reduce_mean(tf.square(H - y))
    return cost

optimizer = tf.optimizers.Adam(learning_rate = 0.01)

for step in range(3000):
    optimizer.minimize(compute_cost, var_list=[W, b, c])
    if step % 300 == 0:
        print("{}, {}, {}, {}".format(W.numpy(), b.numpy(), c.numpy(),
compute_cost().numpy()))

line_x = np.arange(min(x), max(x), 0.01)
line_y = W * line_x**2 + b * line_x + c

plt.plot(line_x, line_y, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```

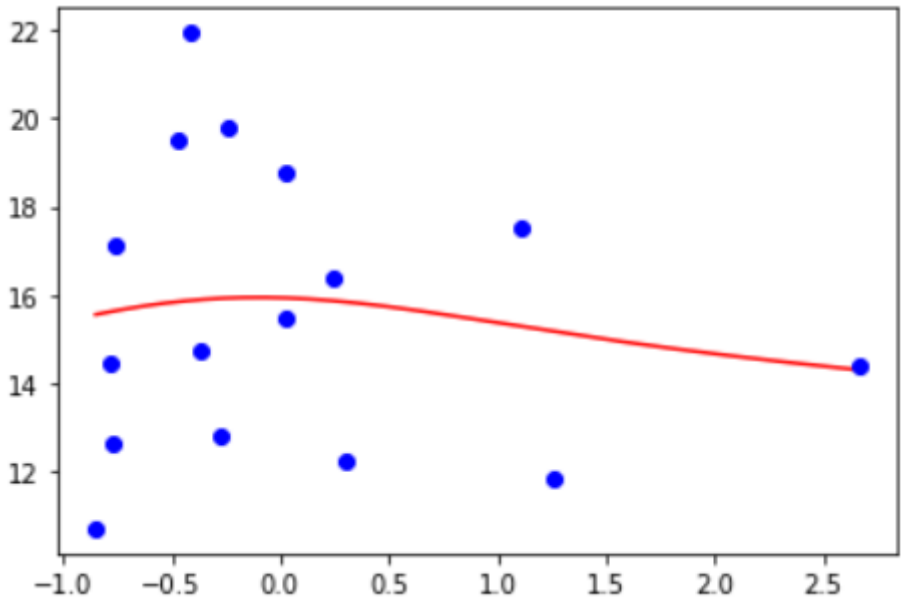


6. 딥러닝을 활용한 회귀

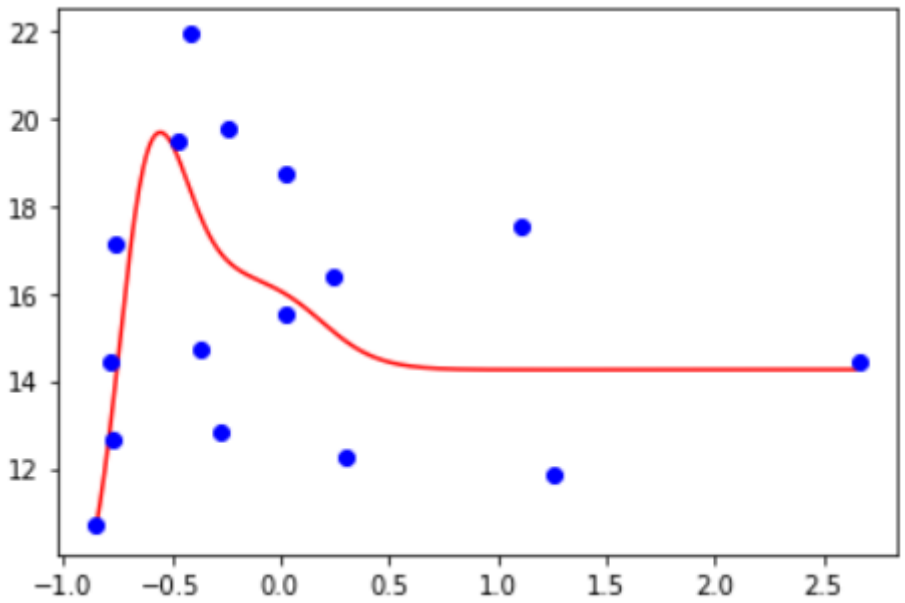
```
x = [0.3, -0.78, 1.26, 0.03, 1.11, 0.24, -0.24, -0.47, -0.77, -0.37, -0.85,
-0.41, -0.27, 0.02, -0.76, 2.66]
y = [12.27, 14.44, 11.87, 18.75, 17.52, 16.37, 19.78, 19.51, 12.65, 14.74,
10.72, 21.94, 12.83, 15.51, 17.14, 14.42]

model = tf.keras.Sequential([
   tf.keras.layers.Dense(units=6, activation='tanh', input_shape=(1,)),
    tf.keras.layers.Dense(units=1)
])

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.1), loss='mse')

model.summary()
```

- 위의 딥러닝 모델은 2개의 Dense 레이어로 구성
  - 첫번째 레이어는 활성화 함수로 'tanh'를 사용함으로써 실수를 입력 받아 -1~1 사이의 값을 반환
  - 또한 6개의 뉴런을 할당, 너무 많이 할당 되면 **Overfitting** 문제 발생 가능
  - 두번째 레이어는 x에 입력값에 대한 하나의 y값만 출력되어야 하기 때문에 뉴런 수가 1개
- optimizer의 손실은 MSE(Mean Squared Error)로, 잔차의 제곱의 평균을 구함
  - 때문에 손실을 줄이는 쪽으로 학습

```
model.fit(x, y, epochs=10)  // 학습
model.predict(x)            // 예측

plt.plot(line_x, line_y, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```



- 과적합 예시 (1000번 학습시)



7. 더하기

7.1 데이터 셋 준비

- 데이터 셋 다운로드

```
from tensorflow.keras.datasets import boston_housing
(train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()
```

- 데이터 셋 확인

```
print(len(train_X), len(test_X))
print(train_X[0])
print(train_Y[0])
```

```
404 102     // 훈련데이터, 테스트 데이터
[  1.23247   0.        8.14      0.        0.538     6.142    91.7
   3.9769    4.      307.       21.      396.9      18.72   ]
15.2
```

7.2 정규화 & 표준화

- 정규화
  - 데이터를 0~1 사이의 값으로 변환
  - 이상치가 있을 경우 최소, 최대값에 많은 영향을 받으므로 주의 필요

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

- 표준화
  - 0이 평균인 표준정구분포화 시킴
  - 평균을 기준으로 어느 정도 떨어져 있는지를 나타낼 때 사용
  - 이상치는 잘 처리하지만, 정확히 동일한 척도로 정규화 데이터를 생성하지는 않음

$$x = \frac{x - \mu}{\sigma}$$

- 딥러닝에서는 데이터를 전처리해서 정규화해야 학습 효율이 좋음

```
x_mean = train_X.mean()
x_std = train_X.std()

# 표준화
train_X -= x_mean
train_X /= x_std

test_X -= x_mean
test_X /= x_std

y_mean = train_Y.mean()
y_std = train_Y.std()
train_Y -= y_mean
train_Y /= y_std
test_Y -= y_mean
test_Y /= y_std
```

```
# 정규화 확인
print(train_X[0])
print(train_Y[0])
```

```
[-0.47482083 -0.48335641 -0.42698208 -0.48335641 -0.47963044 -0.44081941
  0.15172056 -0.45581402 -0.45565404  1.64280094 -0.33791894  2.26541184
 -0.35370929]
-0.7821526033779157
```

7.3 모델 생성

```
import tensorflow as tf
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)
])
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')
model.summary()
```

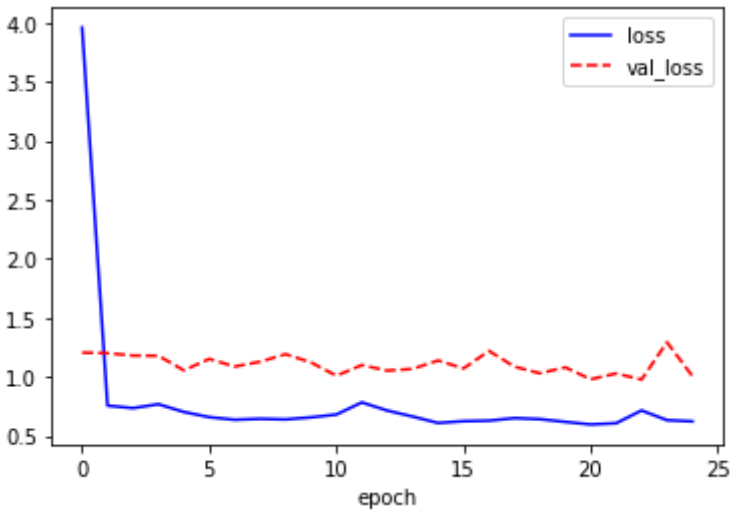
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 52)                728
_________________________________________________________________
dense_1 (Dense)              (None, 39)                2067
_________________________________________________________________
dense_2 (Dense)              (None, 26)                1040
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 27
=================================================================
Total params: 3,862
Trainable params: 3,862
Non-trainable params: 0
_________________________________________________________________
```

7.4 학습

```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32,
validation_split=0.25)
```

- `validation_split` : 훈련 데이터 검증률
- 시각화

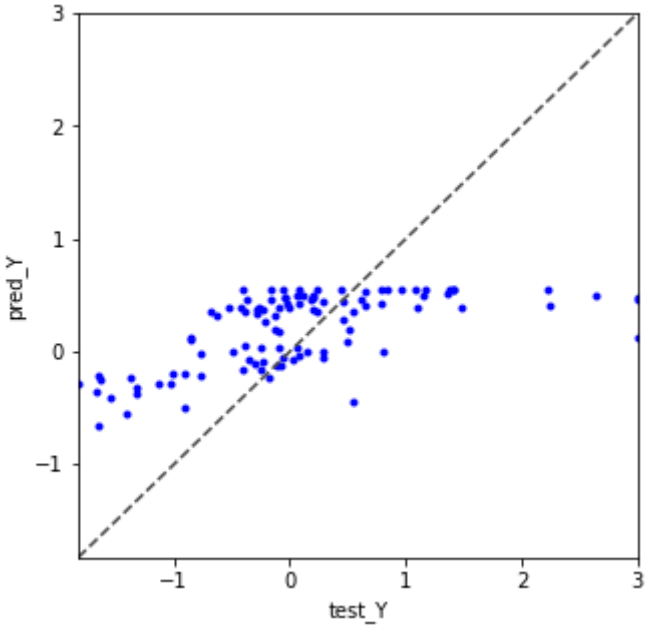
```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--', label='val_loss')
plt.xlabel('epoch')
plt.legend()
plt.show()
```



7.5 비교

- 실제 값과 예측 값을 비교 시각화

```
pred_Y = model.predict(test_X)

plt.figure(figsize=(5, 5))
plt.plot(test_Y, pred_Y, 'b.')
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])

plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--",
c=".3")
plt.xlabel('test_Y')
plt.ylabel('pred_Y')

plt.show()
```

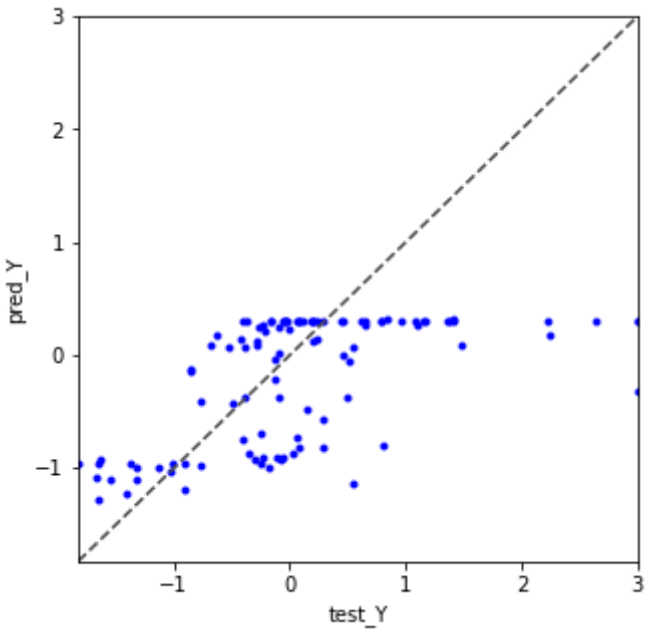


- 이상적이라면 대각선 위에 모든 점이 위치해야 함

7.6 모델 및 학습 수정

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)
])
model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')

history = model.fit(train_X, train_Y, epochs=25, batch_size=32,
validation_split=0.25, callbacks=[tf.keras.callbacks.EarlyStopping(patience=3,
monitor='val_loss')])
```

- 콜백함수 추가하여 학습 도중 epoch가 끝날 때마다 호출
- 위에서는 val_loss가 3회의 epoch를 수행하는 동안 최고 기록으 갱신하지 못한다면 학습을 중단
- 7.5의 코드르 복사하여 다시 실행하면



- 조금 더 다양한 값을 예측하는 것으로 볼 수 있다.
- EarlyStopping을 이용하면 과적합되지 않도록 도중에 학습 중단 가능