

2018년 대한민국 인구증가율과 고령인구비율

실습 파일

- 파일 생성
 - tf2_ch04.ipynb

인구증가율과 고령인구비율

- 교재 p74

```
# 인구증가율과 고령인구비율
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]
```

활성화 함수 tanh

- 하이퍼볼릭 타젠트
 - S자 곡선
 - $(-1, 1)$ 사이의 값

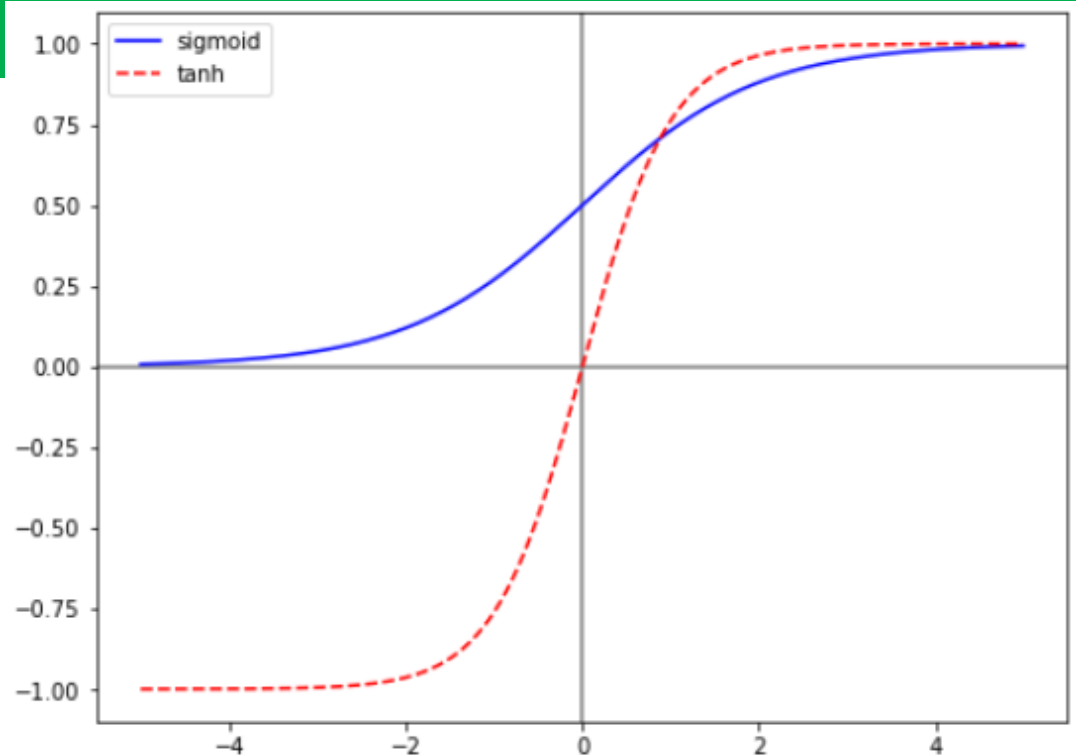
그림 4.2 출력 코드

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

```
x = np.arange(-5, 5, 0.01)
sigmoid_x = [sigmoid(z) for z in x]
tanh_x = [math.tanh(z) for z in x]
```

```
plt.figure(figsize=(8, 6))
```

```
plt.axhline(0, color='gray')
plt.axvline(0, color='gray')
plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
plt.plot(x, tanh_x, 'r--', label='tanh')
plt.legend()
plt.show()
```



$$\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \sigma(x) = \frac{1}{1 + e^{-x}}.$$

딥러닝 모델

- **중간층**
 - 뉴런 6 개
- **출력층**
 - 뉴런 1 개

4.7 딥러닝 네트워크를 이용한 회귀

```
import tensorflow as tf
import numpy as np
```

인구증가율과 고령인구비율

```
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, 1
0.72, 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(lr=0.1), loss='mse')
model.summary()
```

학습

```
[10] 1 # 4.8 딥러닝 네트워크의 학습
      2 model.fit(X, Y, epochs=10)
```

```
↳ Epoch 1/10
1/1 [=====] - 0s 1ms/step - loss: 256.3840
Epoch 2/10
1/1 [=====] - 0s 824us/step - loss: 116.0593
Epoch 3/10
1/1 [=====] - 0s 926us/step - loss: 9.6935
Epoch 4/10
1/1 [=====] - 0s 952us/step - loss: 9.3609
Epoch 5/10
1/1 [=====] - 0s 2ms/step - loss: 9.2970
Epoch 6/10
1/1 [=====] - 0s 963us/step - loss: 9.2382
Epoch 7/10
1/1 [=====] - 0s 991us/step - loss: 9.1770
Epoch 8/10
1/1 [=====] - 0s 908us/step - loss: 9.1145
Epoch 9/10
1/1 [=====] - 0s 911us/step - loss: 9.0526
Epoch 10/10
1/1 [=====] - 0s 899us/step - loss: 8.9924
<tensorflow.python.keras.callbacks.History at 0x7f9ab36daa58>
```

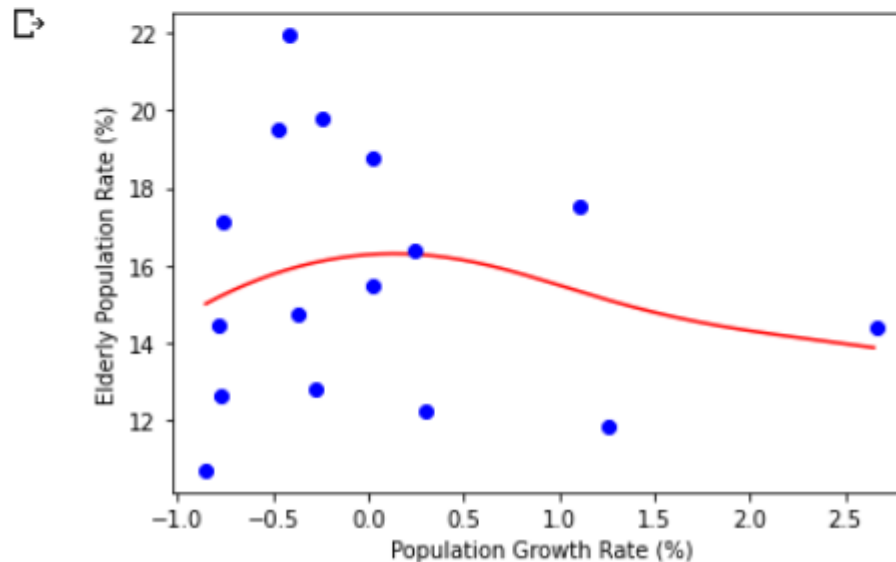
예측

```
[11] 1 # 4.9 딥러닝 네트워크의 Y값 예측  
      2 model.predict(X)
```

```
↳ array([[16.27147 ],  
          [15.190466],  
          [15.10627 ],  
          [16.290047],  
          [15.327049],  
          [16.291916],  
          [16.117289],  
          [15.818182],  
          [15.214785],  
          [15.965492],  
          [15.012625],  
          [15.909767],  
          [16.086227],  
          [16.287073],  
          [15.238832],  
          [13.875053]], dtype=float32)
```

결과 시각화

```
[12] 1 # 4.10 딥러닝 네트워크의 회귀선 확인
      2 import matplotlib.pyplot as plt
      3
      4 line_x = np.arange(min(X), max(X), 0.01)
      5 line_y = model.predict(line_x)
      6
      7 plt.plot(line_x, line_y, 'r-')
      8 plt.plot(X, Y, 'bo')
      9
     10 plt.xlabel('Population Growth Rate (%)')
     11 plt.ylabel('Elderly Population Rate (%)')
     12 plt.show()
```



텐서플로만을 이용한 회귀 분석

- **optimizer**

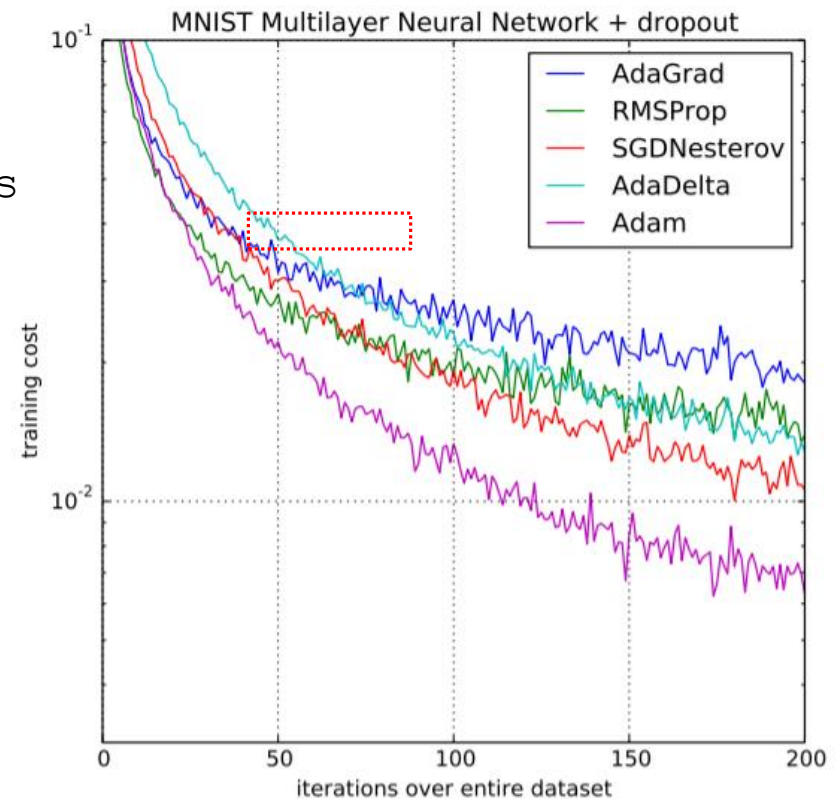
- 최적화 과정(복잡한 미분 계산 및 가중치 수정)을 자동으로 진행

- **SGD, adam**

- **학습률(learning rate)**

- 보통 0.1 ~ 0.0001

```
optimizer = tf.keras.optimizers
.Adam(lr=0.07)
```



변수 Variables

- **딥러닝 학습에서 최적화 과정**
 - 모델의 매개변수(parameters) 즉, 가중치(및 편향)를 조정하는 것
- **변수 tf.Variable**
 - 프로그램에 의해 변화하는 공유된 지속 상태를 표현하는 가장 좋은 방법
 - 하나의 텐서를 표현
 - 텐서 값은 텐서에 연산을 수행하여 변경 가능
 - 모델 파라미터를 저장하는데 tf.Variable을 사용
- **변수 생성**
 - 변수를 생성하려면 단순히 초기값을 설정

```
# a와 b를 랜덤한 값으로 초기화합니다.  
# a = tf.Variable(random.random())  
# b = tf.Variable(random.random())  
a = tf.Variable(tf.random.uniform([1], 0, 1))  
b = tf.Variable(tf.random.uniform([1], 0, 1))
```

메소드 `minimize()`

- 첫 번째 인자
 - 최소화할 손실 함수
- 두 번째 인자 `var_list`
 - 학습시킬 변수 리스트, 가중치와 편향
- 1000번의 학습을 거쳐
 - 잔차의 제곱 평균을 최소화하는 적절한 값 `a`, `b`에 도달을 기대

```
for i in range(1000):  
    # 잔차의 제곱의 평균을 최소화(minimize) 합니다.  
    optimizer.minimize(compute_loss, var_list=[a,b])
```

텐서플로만을 이용한 회귀 분석 소스

4.4 텐서플로를 이용해서 회귀선 구하기

```
import tensorflow as tf
import numpy as np
# import random
```

```
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]
```

a와 b를 랜덤한 값으로 초기화합니다.

```
# a = tf.Variable(random.random())
# b = tf.Variable(random.random())
a = tf.Variable(tf.random.uniform([1], 0, 1))
b = tf.Variable(tf.random.uniform([1], 0, 1))
```

잔차의 제곱의 평균을 반환하는 함수입니다.

```
def compute_loss():
    y_pred = a * X + b
    loss = tf.reduce_mean((Y - y_pred) ** 2)
    return loss
```

```
optimizer = tf.keras.optimizers.Adam(lr=0.07)
for i in range(1000):
    # 잔차의 제곱의 평균을 최소화(minimize)합니다.
    optimizer.minimize(compute_loss, var_list=[a,b])
```

```
if i % 100 == 99:
    print(i, 'a:', a.numpy(), 'b:', b.numpy(), 'loss:', compute_loss().numpy())
```



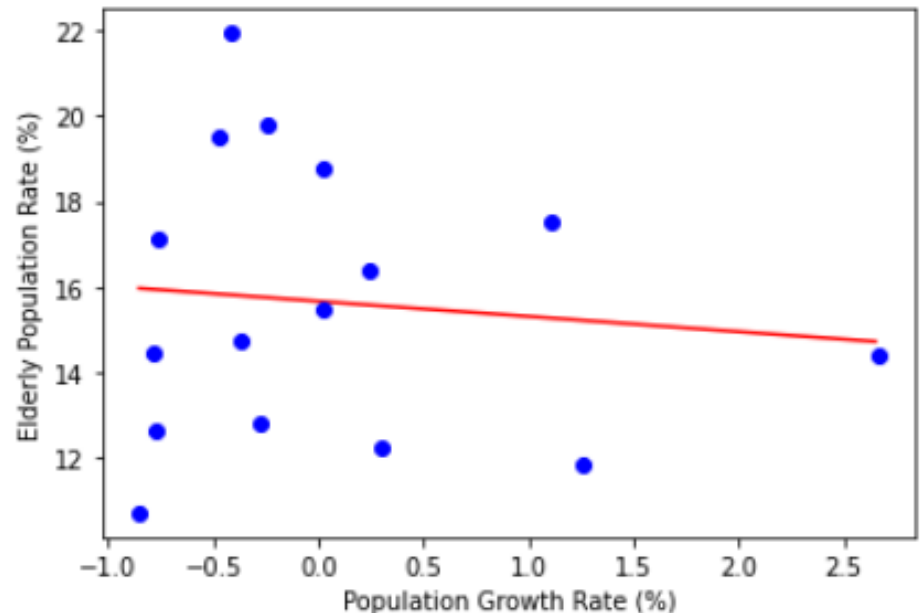
```
99 a: [0.09661794] b: [7.1642118] loss: 81.94983
199 a: [-0.13748273] b: [11.560307] loss: 26.625612
299 a: [-0.2691387] b: [14.037779] loss: 12.436548
399 a: [-0.32794657] b: [15.1445] loss: 10.055599
499 a: [-0.34860265] b: [15.533232] loss: 9.79928
599 a: [-0.35433018] b: [15.641014] loss: 9.781603
699 a: [-0.35558546] b: [15.664643] loss: 9.780827
799 a: [-0.35580176] b: [15.668713] loss: 9.780804
899 a: [-0.355831] b: [15.66926] loss: 9.780804
999 a: [-0.3558332] b: [15.669303] loss: 9.780804
```

텐서플로만을 이용한 회귀 분석 회귀선 그리기

```
import matplotlib.pyplot as plt

line_x = np.arange(min(X), max(X), 0.01)
line_y = a * line_x + b

# 그래프를 그립니다.
plt.plot(line_x, line_y, 'r-')
plt.plot(X, Y, 'bo')
plt.xlabel('Population Growth Rate (%)')
plt.ylabel('Elderly Population Rate (%)')
plt.show()
```

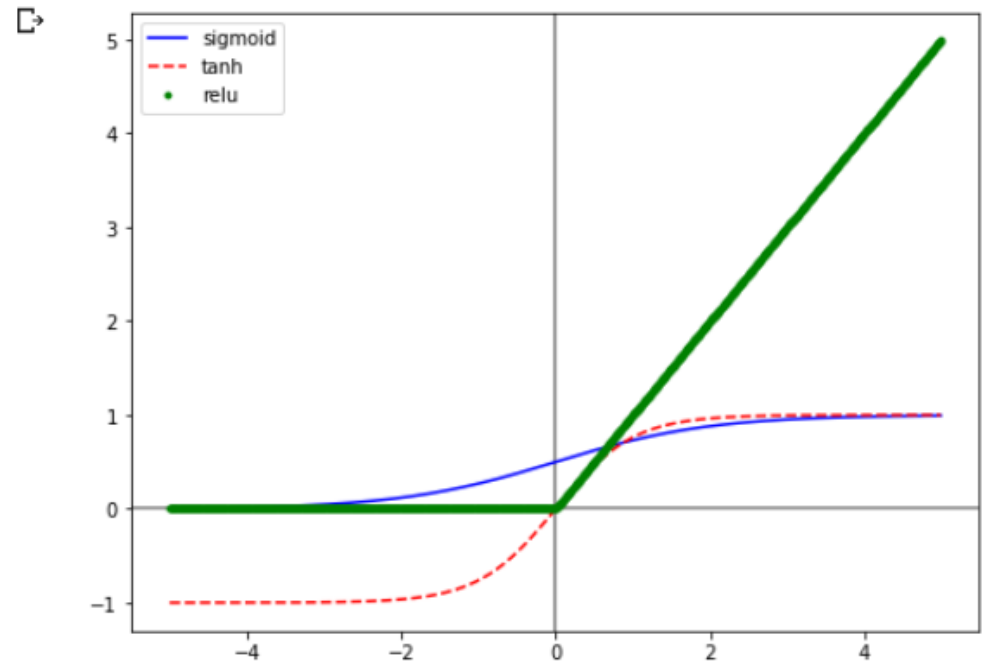


보스턴 주택 가격 예측

주요 활성화 함수

- ReLU
- Sigmoid
- Tanh

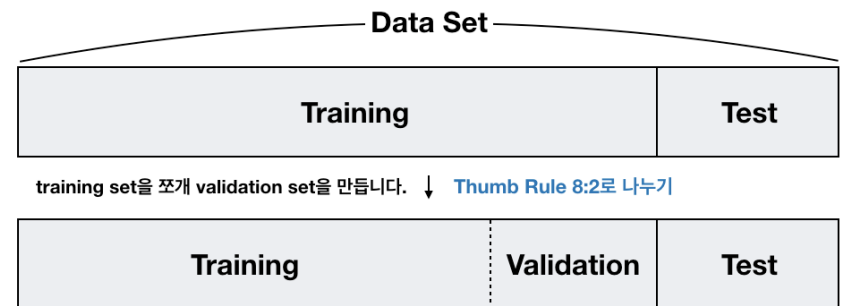
```
[22] 1 # 활성화 함수
      2 import math
      3 def sigmoid(x):
      4     return 1 / (1 + math.exp(-x))
      5
      6 x = np.arange(-5, 5, 0.01)
      7 sigmoid_x = [sigmoid(z) for z in x]
      8 tanh_x = [math.tanh(z) for z in x]
      9 relu = [0 if z < 0 else z for z in x]
     10
     11 plt.figure(figsize=(8, 6))
     12
     13 plt.axhline(0, color='gray')
     14 plt.axvline(0, color='gray')
     15 plt.plot(x, sigmoid_x, 'b-', label='sigmoid')
     16 plt.plot(x, tanh_x, 'r--', label='tanh')
     17 plt.plot(x, relu, 'g.', label='relu')
     18 plt.legend()
     19 plt.show()
```



보스톤 주택 가격 예측

• 1978년 보스톤 지역 주택 가격 데이터 셋

- 506 개타운의 주택 가격 중앙 값, 천 달러 단위
- 범죄율, 방 수, 고속도로 까지 거리 등 13가지 특성, p93
- 학습 데이터
 - 404개
- 테스트 데이터
 - 102개



```
[17] 1 # 4.11 데이터 불러오기
      2 from tensorflow.keras.datasets import boston_housing
      3 (train_X, train_Y), (test_X, test_Y) = boston_housing.load_data()
      4
      5 print(train_X.shape, test_X.shape)
      6 print(train_X[0])
      7 print(train_Y[0])
```

```
↳ (404, 13) (102, 13)
   [ 1.23247  0.         8.14      0.         0.538      6.142     91.7
     3.9769   4.        307.      21.        396.9     18.72   ]
   15.2
```


자료의 정규화

정규화의 필요

- 특성의 단위가 다름
 - 비율, 0/1, 양수 등
- 정규화(standardization)가 학습 효율에 좋음

정규화 방법

- 학습 데이터:
 - $(\text{train_X}_i - \text{학습데이터평균}) / \text{학습데이터 표준편차}$
 - 정규 분포를 가정
- 테스트 데이터
 - $(\text{test_X}_i - \text{학습데이터평균}) / \text{학습데이터 표준편차}$
 - 테스트데이터가 정규 분포를 가정할 수 없으므로

4.12 데이터 전처리 (정규화)

```
x_mean = train_X.mean(axis=0)
x_std = train_X.std(axis=0)
train_X -= x_mean
train_X /= x_std
test_X -= x_mean
test_X /= x_std
```

```
y_mean = train_Y.mean(axis=0)
y_std = train_Y.std(axis=0)
train_Y -= y_mean
train_Y /= y_std
test_Y -= y_mean
test_Y /= y_std
```

```
print(train_X[0])
print(train_Y[0])
```

```
[-0.27224633 -0.48361547 -0.43576161 -0.25683275 -0.1652266 -0.1764426
 0.81306188 0.1166983 -0.62624905 -0.59517003 1.14850044 0.44807713 0.8252202]
-0.7821526033779157
```

딥러닝 모델

- 총 4개 층
 - 출력 층은 회귀 모델, 주택 가격이므로 1
- 최적화
 - 학습률
 - $lr=0.07$
 - 손실 함수
 - mse

```
[20] 1
      2 # 4.13 Boston Housing Dataset 회귀 모델 생성
      3 model = tf.keras.models.Sequential([
      4     tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
      5     tf.keras.layers.Dense(units=39, activation='relu'),
      6     tf.keras.layers.Dense(units=26, activation='relu'),
      7     tf.keras.layers.Dense(units=1)
      8 ])
      9
     10 model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')
     11
     12 model.summary()
```

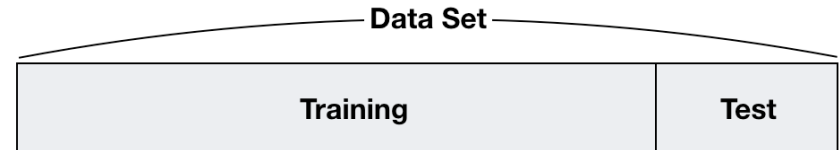
Model: "sequential_4"

Layer (type)	Output Shape	Param #
dense_10 (Dense)	(None, 52)	728
dense_11 (Dense)	(None, 39)	2067
dense_12 (Dense)	(None, 26)	1040
dense_13 (Dense)	(None, 1)	27
Total params: 3,862		
Trainable params: 3,862		
Non-trainable params: 0		

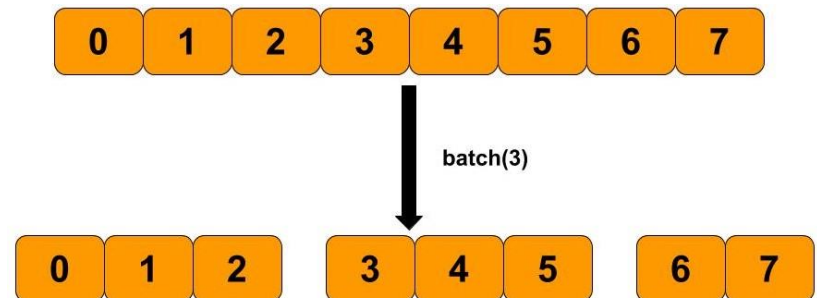
학습

• 배치 사이즈와 검증 데이터

- 훈련과 검증 분리, 훈련 데이터 404개 중 일부를 검증 데이터로 사용
 - **Validation_split:**
 - 검증 용 데이터의 비율
 - **만일 .2면**
 - 훈련:검증 == 80%:20% 비준으로 준비
- Batch_size
 - **훈련에서 가중치와 편향의 패러미터를 수정하는 데이터 단위 수**
- train_size
 - **훈련 데이터 수**



training set을 쪼개 validation set을 만듭니다. ↓ Thumb Rule 8:2로 나누기



4.14 회귀 모델 학습

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

훈련과정 시각화와 평가

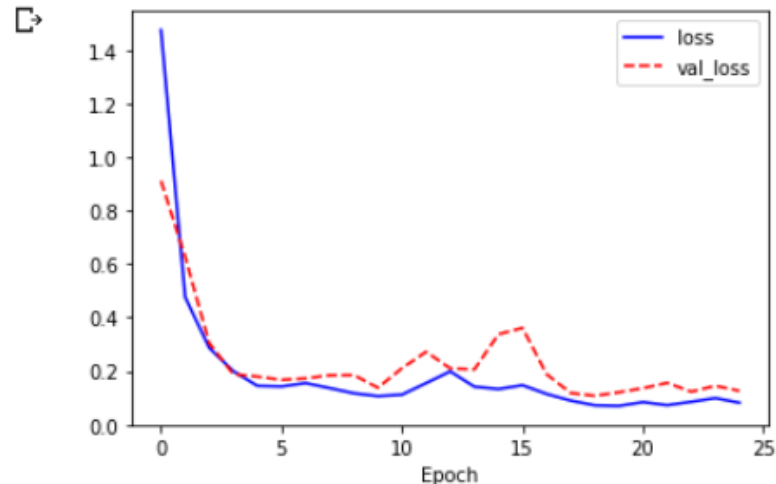
검증 데이터 손실

- 일반적으로 loss는 꾸준히 감소
- val_loss
 - 일반적으로 loss 보다 높음
 - 항상 감소하지도 않음

평가 결과

- 손실 값
 - 작을수록 좋은 결과
 - 검증 손실이 적을수록 테스트 평가의 손실도 적음

```
[ ] 1 # 4.15 회귀 모델 학습 결과 시각화
2 import matplotlib.pyplot as plt
3 plt.plot(history.history['loss'], 'b-', label='loss')
4 plt.plot(history.history['val_loss'], 'r--', label='val_loss')
5 plt.xlabel('Epoch')
6 plt.legend()
7 plt.show()
```



```
[60] 1 # 4.16 회귀 모델 평가
2 model.evaluate(test_X, test_Y)
```

```
4/4 [=====] - 0s 2ms/step - loss: 0.1999
0.19986297190189362
```

예측 시각화

- 테스트 데이터의 예측과 실제 주택 가격 비교
 - 각 점들이 점선의 대각선에 있어야 좋은 예측

4.17 실제 주택 가격과 예측 주택 가격 시각화

```
import matplotlib.pyplot as plt
```

```
pred_Y = model.predict(test_X)
```

```
plt.figure(figsize=(8,8))
```

```
plt.plot(test_Y, pred_Y, 'b.')
```

```
plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])
```

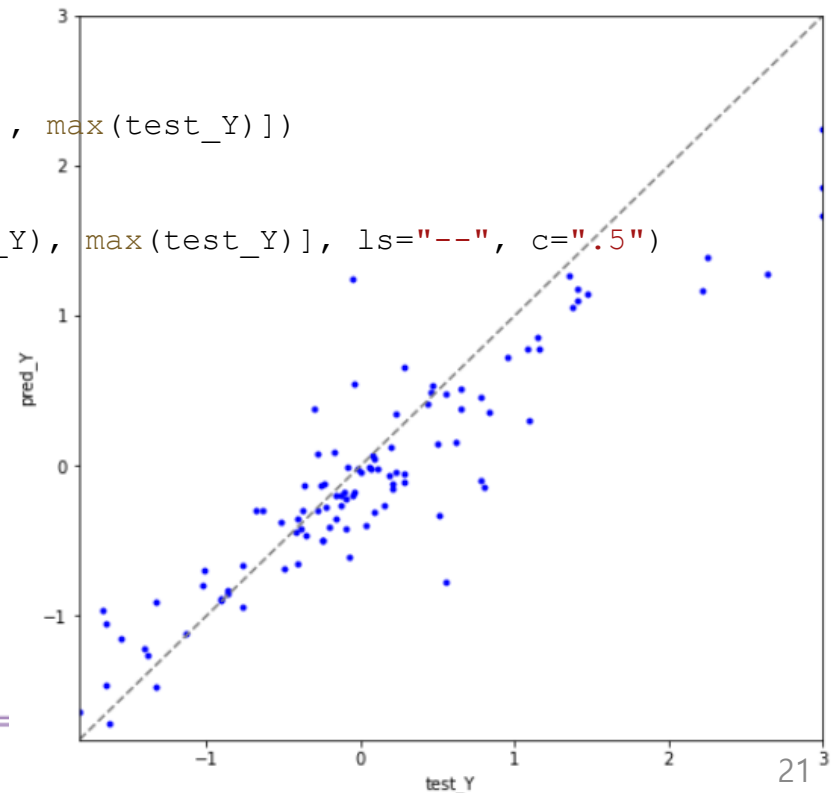
$y=x$ 에 해당하는 대각선

```
plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".5")
```

```
plt.xlabel('test_Y')
```

```
plt.ylabel('pred_Y')
```

```
plt.show()
```



자동으로 학습 중단

- 검증 손실(val_loss)이 적을수록 테스트 평가의 손실도 적음
- 검증 데이터에 대한 성적이 좋도록 유도
 - 과적합에 의해 검증 손실이 증가하면 학습을 중단되도록 지정
 - 함수 callbacks 사용

```
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')])
```


- 일찍 멈춤 기능
 - tf.keras.callbacks.EarlyStopping
 - monitor='val_loss'
 - 지켜볼 기준 값이 검증 손실
 - patience=3
 - 3회의 epochs를 실행하는 기준 값이 동안 최고 기록을 갱신하지 못하면(더 낮아지지 않으면) 학습을 멈춤

EarlyStopping

- 10 에폭의 기록인 .1748 이후
 - 13 에폭에서도 그 기록을 갱신하지 못했으므로 학습을 중단

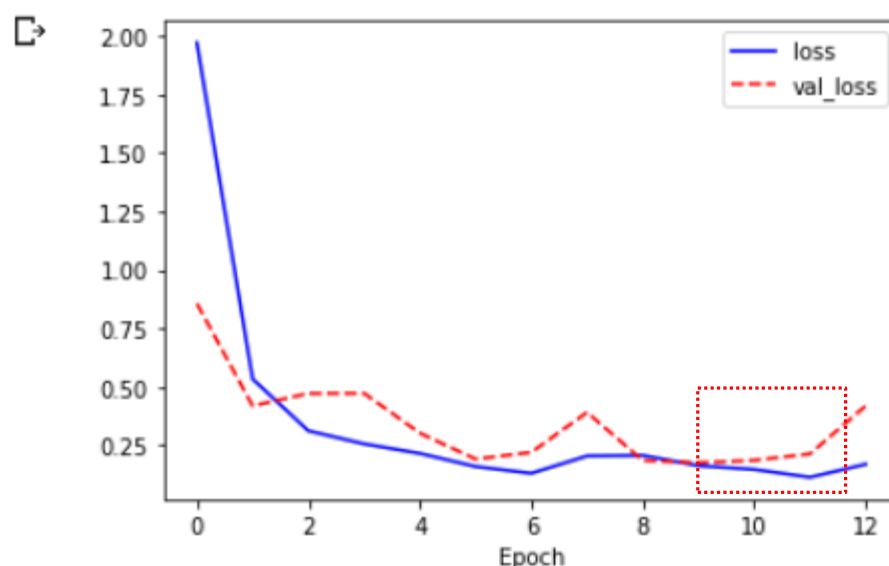
```
history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
                    callbacks=[tf.keras.callbacks.EarlyStopping(patience=3, monitor='val_loss')])
```

```
Epoch 9/25
10/10 [=====] - 0s 4ms/step - loss: 0.2058 - val_loss: 0.1839
Epoch 10/25
10/10 [=====] - 0s 5ms/step - loss: 0.1620 - val_loss: 0.1748
Epoch 11/25
10/10 [=====] - 0s 5ms/step - loss: 0.1459 - val_loss: 0.1848
Epoch 12/25
10/10 [=====] - 0s 4ms/step - loss: 0.1124 - val_loss: 0.2126
Epoch 13/25
10/10 [=====] - 0s 4ms/step - loss: 0.1683 - val_loss: 0.4185
```



자동 중단 시각화

```
[73] 1 # 4.19 회귀 모델 학습 결과 시각화
      2 import matplotlib.pyplot as plt
      3 plt.plot(history.history['loss'], 'b-', label='loss')
      4 plt.plot(history.history['val_loss'], 'r--', label='val_loss')
      5 plt.xlabel('Epoch')
      6 plt.legend()
      7 plt.show()
```

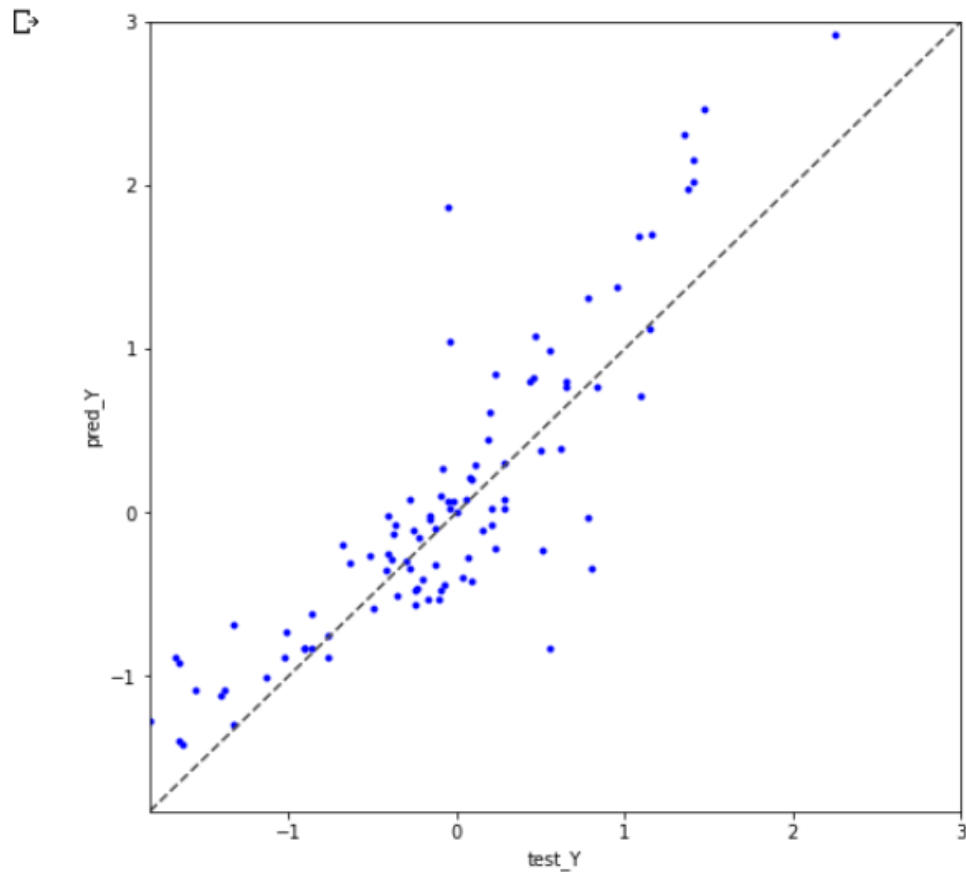


```
[74] 1 # 4.20 회귀 모델 평가
      2 model.evaluate(test_X, test_Y)
```

```
4/4 [=====] - 0s 2ms/step - loss: 0.2536
0.2535804212093353
```


예측 시각화

```
[75] 1 # 4.21 실제 주택 가격과 예측 주택 가격 시각화
      2 import matplotlib.pyplot as plt
      3
      4 pred_Y = model.predict(test_X)
      5
      6 plt.figure(figsize=(8,8))
      7 plt.plot(test_Y, pred_Y, 'b.')
      8 plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])
      9
     10 plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
     11 plt.xlabel('test_Y')
     12 plt.ylabel('pred_Y')
     13
     14 plt.show()
```



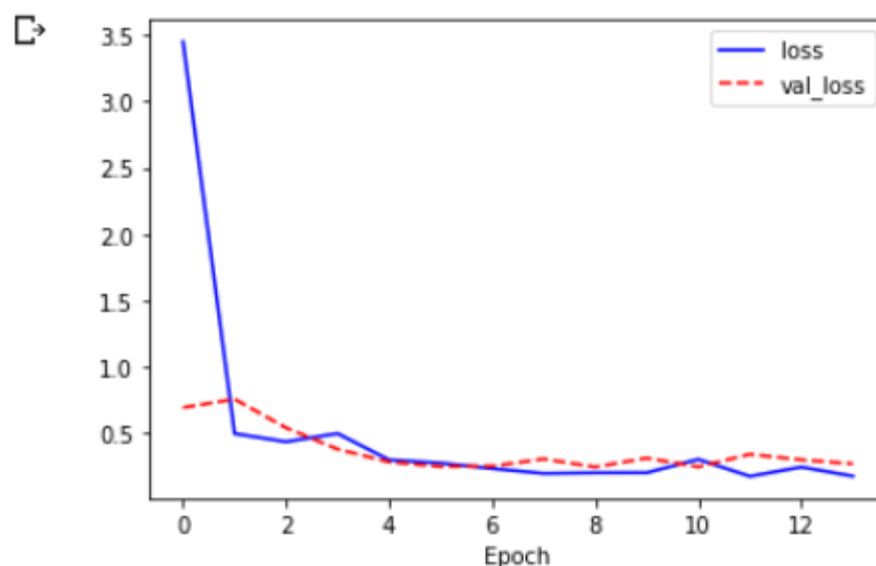
Dropout

```
[77] 1 # 모델 재정의 및 학습, dropout 사용
      2 model = tf.keras.Sequential([
      3     tf.keras.layers.Dense(units=52, activation='relu', input_shape=(13,)),
      4     tf.keras.layers.Dense(units=39, activation='relu'),
      5     tf.keras.layers.Dense(units=26, activation='relu'),
      6     tf.keras.layers.Dropout(.1),
      7     tf.keras.layers.Dense(units=1)
      8 ])
      9
     10 model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.07), loss='mse')
     11
     12 history = model.fit(train_X, train_Y, epochs=25, batch_size=32, validation_split=0.25,
     13                     callbacks=[tf.keras.callbacks.EarlyStopping(patience=5, monitor='val_loss')])
```

```
↳ Epoch 1/25
10/10 [=====] - 0s 10ms/step - loss: 3.4517 - val_loss: 0.6883
Epoch 2/25
10/10 [=====] - 0s 4ms/step - loss: 0.4919 - val_loss: 0.7523
Epoch 3/25
10/10 [=====] - 0s 5ms/step - loss: 0.4304 - val_loss: 0.5356
Epoch 4/25
10/10 [=====] - 0s 4ms/step - loss: 0.4928 - val_loss: 0.3748
Epoch 5/25
10/10 [=====] - 0s 4ms/step - loss: 0.2932 - val_loss: 0.2776
Epoch 6/25
10/10 [=====] - 0s 6ms/step - loss: 0.2694 - val_loss: 0.2432
Epoch 7/25
10/10 [=====] - 0s 4ms/step - loss: 0.2281 - val_loss: 0.2464
Epoch 8/25
10/10 [=====] - 0s 5ms/step - loss: 0.1911 - val_loss: 0.2999
Epoch 9/25
10/10 [=====] - 0s 5ms/step - loss: 0.1959 - val_loss: 0.2399
Epoch 10/25
10/10 [=====] - 0s 5ms/step - loss: 0.1966 - val_loss: 0.3071
Epoch 11/25
10/10 [=====] - 0s 5ms/step - loss: 0.2973 - val_loss: 0.2406
Epoch 12/25
10/10 [=====] - 0s 5ms/step - loss: 0.1697 - val_loss: 0.3360
Epoch 13/25
10/10 [=====] - 0s 4ms/step - loss: 0.2392 - val_loss: 0.2938
Epoch 14/25
10/10 [=====] - 0s 4ms/step - loss: 0.1710 - val_loss: 0.2645
```

학습 시각화

```
[78] 1 # 4.19 회귀 모델 학습 결과 시각화
      2 import matplotlib.pyplot as plt
      3 plt.plot(history.history['loss'], 'b-', label='loss')
      4 plt.plot(history.history['val_loss'], 'r--', label='val_loss')
      5 plt.xlabel('Epoch')
      6 plt.legend()
      7 plt.show()
```



```
[79] 1 # 4.20 회귀 모델 평가
      2 model.evaluate(test_X, test_Y)
```

```
4/4 [=====] - 0s 2ms/step - loss: 0.2792
0.27918723225593567
```

예측 시각화

```
[80] 1 # 4.21 실제 주택 가격과 예측 주택 가격 시각화
      2 import matplotlib.pyplot as plt
      3
      4 pred_Y = model.predict(test_X)
      5
      6 plt.figure(figsize=(8,8))
      7 plt.plot(test_Y, pred_Y, 'b.')
      8 plt.axis([min(test_Y), max(test_Y), min(test_Y), max(test_Y)])
      9
     10 plt.plot([min(test_Y), max(test_Y)], [min(test_Y), max(test_Y)], ls="--", c=".3")
     11 plt.xlabel('test_Y')
     12 plt.ylabel('pred_Y')
     13
     14 plt.show()
```

