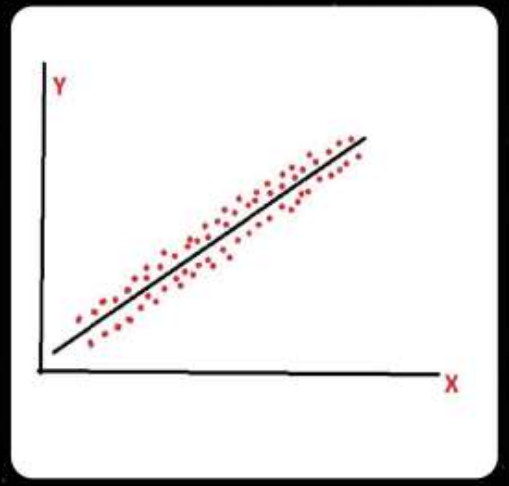
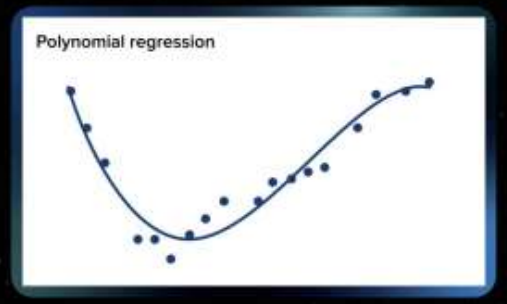
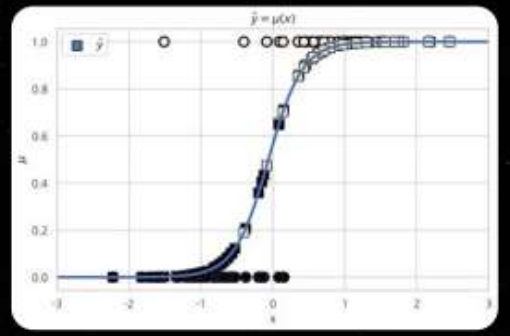
회귀 분석 모델의 종류

회귀 분석 모델의 종류

- 회귀 분석 모델은 선형 회귀, 다항식 회귀, 로지스틱 회귀 등 다양한 유형이 있습니다.
- 선형 회귀는 데이터 간의 선형 관계를 모델링하며, 다항식 회귀는 비선형 관계를 모델링합니다.
- 로지스틱 회귀는 이진 분류 문제에 사용되며, 데이터가 특정 카테고리에 속할 확률을 예측합니다.

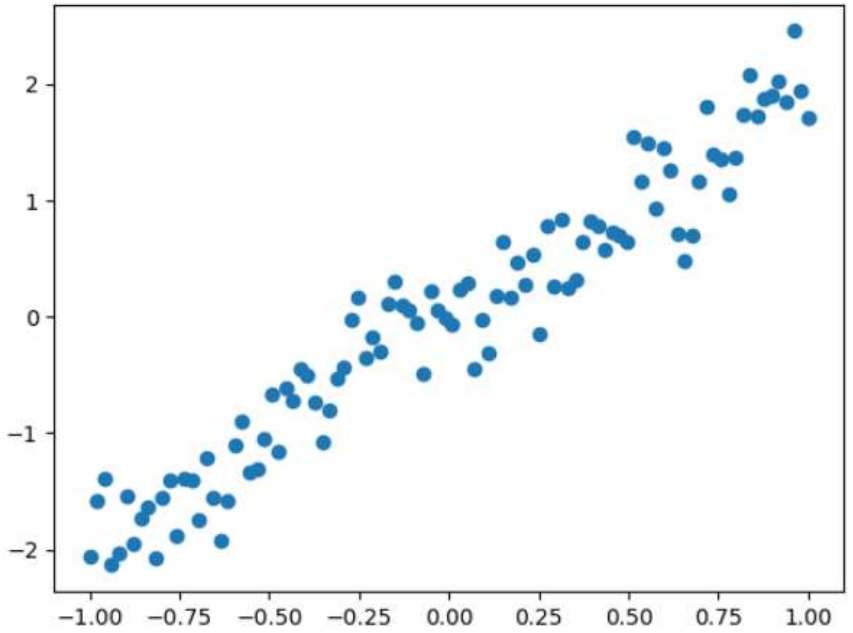






실습I[데이터 시각화]

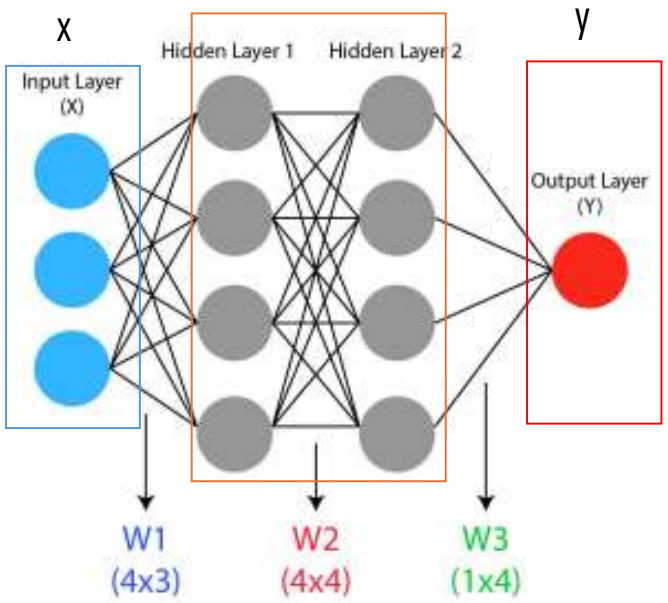
```python
# -*- coding: utf-8 -*-
"""
Created on Sun Sep 10 22:02:01 2023

@author: ajm
"""

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

#데이터를 임의로 생성
x_train = np.linspace(-1, 1, 100)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

#Numpy에서 tensor로 변환
x_train = tf.convert_to_tensor(x_train,dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train,dtype=tf.float32)

plt.scatter(x_train, y_train)
plt.show()
```



최

모델 만드는 방법

모델 = x,y 관계를 잘 나타내는 함수




```
model = models.Sequential([
    layers.Input(shape=(3,)),        # 입력 데
    layers.Dense(4, activation='relu'),  # 첫
    layers.Dense(4, activation='relu'),  # 두
    layers.Dense(1, activation='linear') # 출
])
```

Input Layer

Hidden Layer

Ouput Layer

Input Layer :layers.**Input(shape)** : 입력 데이터 1개의 차원

Hidden Layer : layers.**Dense(노드의 수,activation)**

OutPut Layer : layers.Dense(출력 차원)

실습2[선형 회귀 코드]

```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import Adam

x_train = np.linspace(-1, 1, 1000)
y_train = 2 * x_train + np.random.randn(*x_train.shape) * 0.33

model = models.Sequential([
    layers.Input(shape=(1,)),
    layers.Dense(1)
])

x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.3)

x_train = tf.convert_to_tensor(x_train,dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train,dtype=tf.float32)
x_test = tf.convert_to_tensor(x_test,dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test,dtype=tf.float32)

# Compile the model
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mse')
model.summary()
history = model.fit(x_train, y_train, epochs=100, batch_size=16)#, validation_data=(
y_pred = model.predict(x_test)

plt.scatter(x_test, y_test, color='blue', s=5, label='Test Data (with noise)')
plt.scatter(x_test, y_pred, color='green', linewidth=1, label='Model Prediction')
plt.legend()
plt.show()
```

입력 data (-1~1 까지 1000개의 data)

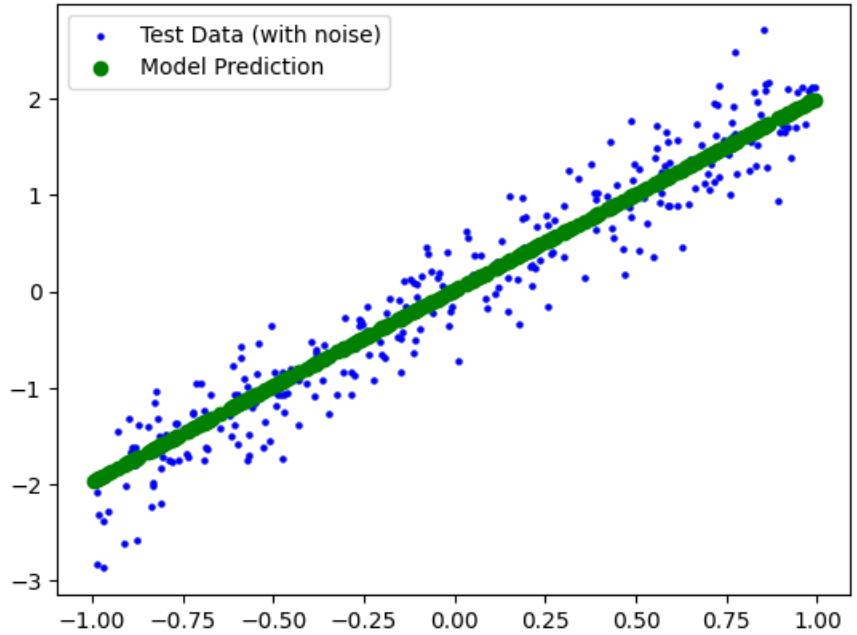
Model 구축

test 30%
train 70 %
데이터 나눔

최적화

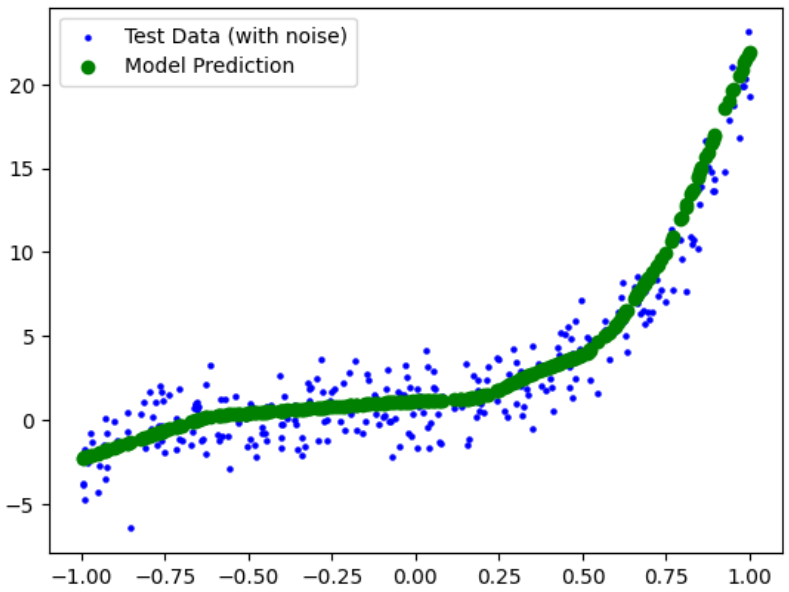
모델학습, 예측

시각화

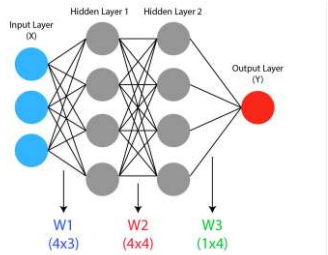


실습3[비선형 다항 회귀 코드]

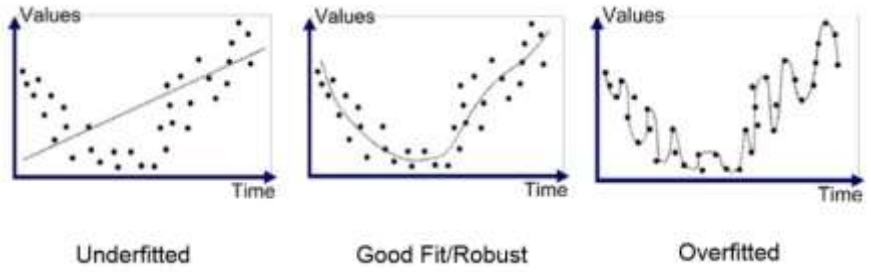
```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import SGD

learning_rate = 0.0085
training_epochs = 40
x_train = np.linspace(-1, 1, 1001)
iterations = 0
num_coeffs = 6
trY_coeffs = [1, 2, 3, 4, 5, 6]
y_train = 0

for i in range(num_coeffs):
    y_train += trY_coeffs[i] * np.power(x_train, i)
y_train += np.random.randn(*x_train.shape) * 1.5

model = models.Sequential([
    layers.Input(shape=(1,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(16, activation='relu'),
    layers.Dense(1)
])
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.3)

x_train = tf.convert_to_tensor(x_train,dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train,dtype=tf.float32)
x_test = tf.convert_to_tensor(x_test,dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test,dtype=tf.float32)

# Compile the model
optimizer = SGD(learning_rate=learning_rate,momentum = 0.5)
model.compile(optimizer=optimizer, loss='mse')
model.summary()
history = model.fit(x_train, y_train, epochs=training_epochs, batch_size=16)
y_pred = model.predict(x_test)
plt.scatter(x_test, y_test, color='blue', s=5, label='Test Data (with noise)')
plt.scatter(x_test, y_pred, color='green', linewidth=1, label='Model Prediction')
plt.legend()
plt.show()
```

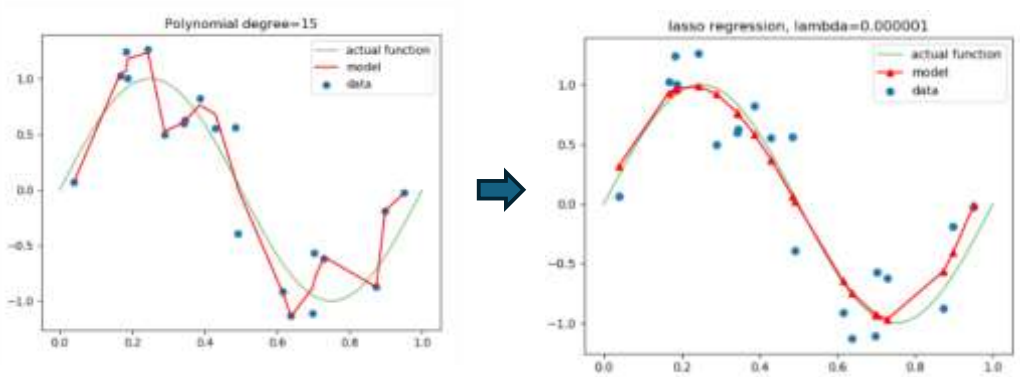


[문제 1] Dense가 1인 모델과 Dense 3인 모델 비교해 보기



과적합





$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

정규화 기법

1. **과적합 방지**
   정규화는 모델의 복잡도를 제한하여 과적합을 방지합니다.
2. **균형 유지**
   모델의 성능과 단순성 사이의 균형을 유지합니다.
3. **일반화 능력 향상**
   새로운 데이터에 대한 예측 성능을 향상시킵니다.
4. 릿지 회귀 (Ridge Regression): L2정규화, 비용함수 J(w)=MSE+α∑w2, 모든 회기계수의 제곱의 합합

   라쏘 회귀 (Lasso Regression): L1정규화, 비용함수 J(w)=MSE+α∑ | w | , 모든 회기계수의 절대값의 합

   엘라스틱 넷 회귀 (Elastic Net Regression): L1과 L2조

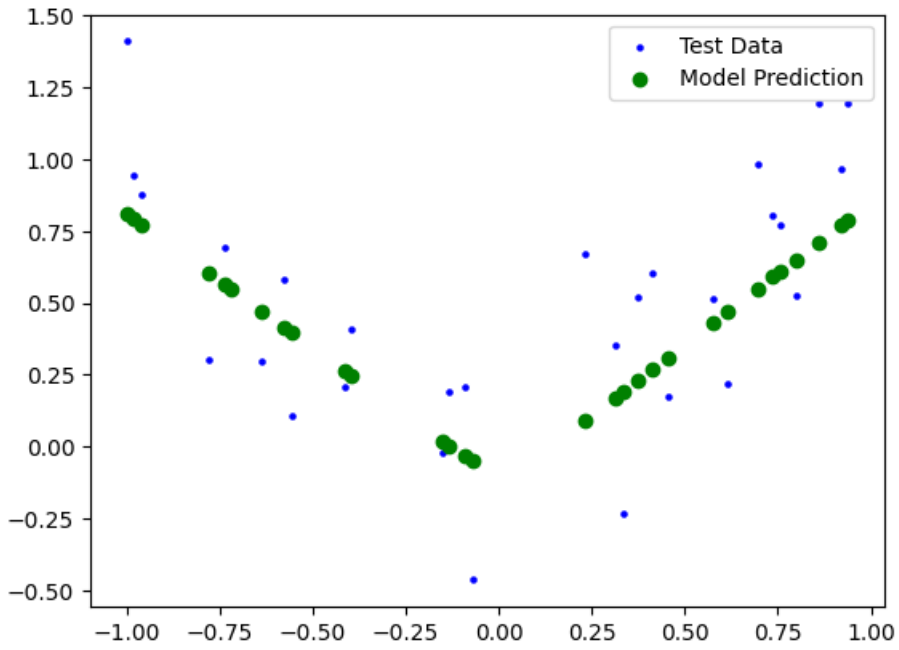
   비용함수 J(w)=MSE+α1∑ | w | +α2∑wi2

정규화는 모델의 복잡도를 제어하는 기법입니다. 이를 통해 과적합을 방지하고 모델의 일반화 능력을 향상시킬 수 있습니다.

실습4

L2 정규화는 선형 회귀 모델의 과적합을 방지하는 효과적인 방법입니다. L2 정규화는 비용 함수에 가중치의 제곱합을 추가하여 모델의 복잡성을 제한합니다. 이는 가중치 값을 작게 유지하는 효과를 가져와 과적합을 줄입니다. TensorFlow를 사용하여 L2 정규화를 적용한 선형 회귀 모델을 구현할 수 있습니다. 모델 학습 과정에서 L2 정규화를 추가하여 가중치가 너무 커지는 것을 방지할 수 있습니다.

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

learning_rate = 0.001
training_epochs = 3000
reg_lambda = 0.001
num_coeffs = 9
x_dataset = np.linspace(-1, 1, 100)
y_dataset_params = [0.] * num_coeffs
y_dataset_params[2] = 1
y_dataset = 0
for i in range(num_coeffs):
    y_dataset += y_dataset_params[i] * np.power(x_dataset, i)
y_dataset += np.random.randn(*x_dataset.shape) * 0.3
#plt.scatter(x_dataset, y_dataset)
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
from tensorflow.keras.optimizers import SGD
from tensorflow.keras import regularizers

model = models.Sequential([
    layers.Input(shape=(1,)),
    layers.Dense(64, activation='relu',kernel_regularizer=regularizers.l2(reg_lambda)),
    layers.Dense(1)
])
x_train, x_test, y_train, y_test = train_test_split(x_dataset, y_dataset, test_size=0.3)

x_train = tf.convert_to_tensor(x_train.dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train.dtype=tf.float32)
x_test = tf.convert_to_tensor(x_test.dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test.dtype=tf.float32)

optimizer = SGD(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='mse')
model.summary()
history = model.fit(x_train, y_train, epochs=training_epochs, batch_size=32)
y_pred = model.predict(x_test)
plt.scatter(x_test, y_test, color='blue', s=5, label='Test Data')
plt.scatter(x_test, y_pred, color='green', linewidth=1, label='Model Prediction')
plt.legend()
plt.show()
```



[문제2] 규제를 작게 준 것과 아주 크게 준 것과 적당히 준 것 아주 작게 준 것의 차이를 비교

실습4[다중 출력 회기 코드]

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

# 입력 데이터 X: 3개의 샘플, 2개의 특성
X = np.array([[1, 2], [3, 4], [5, 6]])
# 출력 데이터 y: 3개의 샘플, 2개의 출력
y = np.array([[3, 5], [7, 9], [11, 13]])

model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(units=5, input_shape=[2], activation='relu'),  # 은닉층
    tf.keras.layers.Dense(units=2)  # 출력층 (출력 수: 2)
])

model.compile(optimizer=tf.keras.optimizers.Adam(0.1), loss='mean_squared_error')
history = model.fit(X, y, epochs=100, verbose=0)

predictions = model.predict(X)

plt.scatter(y[:, 0], predictions[:, 0], color='blue', label='First Output')
plt.plot(y[:, 0], y[:, 0], 'r--', label='Ideal Prediction for First Output')
plt.scatter(y[:, 1], predictions[:, 1], color='black', label='Second Output')
plt.plot(y[:, 1], y[:, 1], 'r--', label='Ideal Prediction for Second Output')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Multi-output Regression Results')
plt.legend()

# 손실 값 그래프 그리기
plt.figure()
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```