

사물인터넷 (Internet of Things)

김태운

목차

- Sensing Layer: 센서 및 센싱/분석기술
 - 데이터 정제

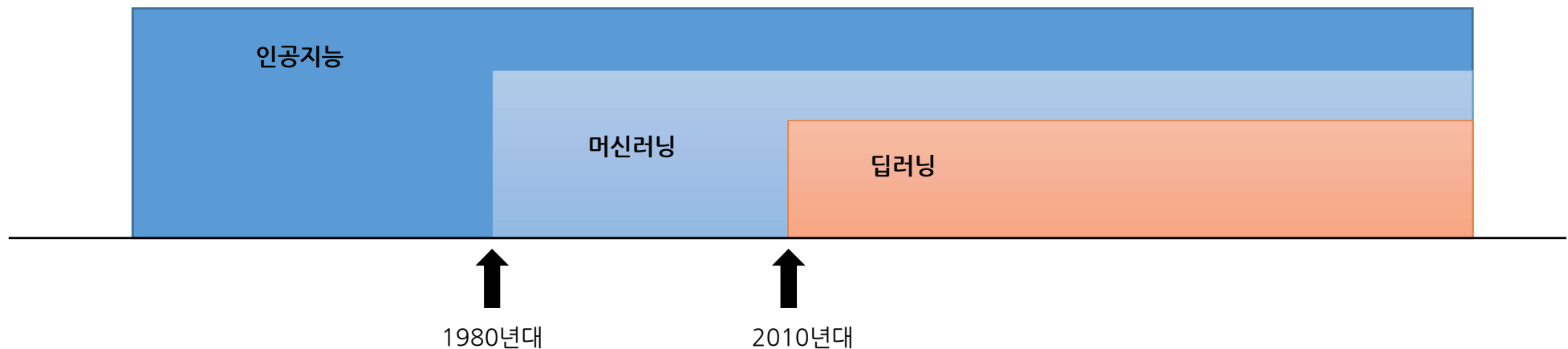
Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 측정 데이터에 포함된 노이즈를 처리하는 방법
 - ~~고가의 고정밀 센서를 사용 => 제품 단가 상승 => 시장 경쟁력 하락~~
 - 노이즈가 포함된 측정값을 그대로 사용하되, 노이즈의 영향을 상쇄시킬 수 있는 기법을 활용 => 지금까지 학습한 내용
 - **사전에 노이즈를 (최대한) 제거하고, 노이즈가 제거된 데이터를 사용**
 - 온도, 습도, 무게 등 저차원(예: 스칼라 값)의 측정값의 경우에는 LPF (Low-Pass Filter) 등을 사용해서 손쉽게 잡음의 영향을 제거할 수 있지만, 2D 이미지 또는 다수의 오디오 소스가 중첩된 음성신호와 같은 고차원의 데이터에서는 LPF 적용이 어려움
 - 따라서, 저 차원 및 고 차원의 측정 데이터에서 노이즈를 제거할 수 있는 기법이 필요!

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

- 딥러닝을 활용한 잡음 제거: 인공지능/딥러닝 개요
 - 인공지능이란(AI) 인간의 사고를 인공적으로 모방한 모든 것
 - 인공지능을 구현하는 방법은 다양하며, 머신러닝이 그 중 하나
 - 머신러닝은 주어진 데이터를 가지고 통계학적인 모델을 학습시켜 인공지능을 구현하는 방법이며, 딥러닝은 머신러닝의 수많은 학습법 중 하나



- 초기의 인공지능은 입력부터 출력까지 사람이 일일이 알려주는 방식으로 개발됨
- 머신러닝은 전문가가 추출한 특징(feature)을 기반으로, 입력 데이터와 정답을 알려주면 정답을 찾는 과정을 스스로 학습
- 딥러닝은 스스로 특징을 추출하고, 정답을 찾는 과정도 스스로 학습함

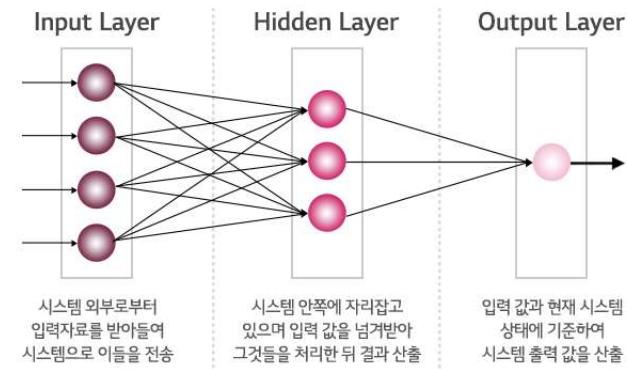
Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 딥러닝을 활용한 잡음 제거: 인공지능/딥러닝 개요
 - 머신러닝의 분류
 - 지도학습: 정답에 해당하는 레이블을 반복적으로 모델에 보여주고, 입력과 출력간 연관성을 학습함. 추후, 한번도 보지 못한 새로운 데이터를 보고 결과를 예측하는데 사용함
 - 비지도학습: 정답이 없는 데이터로 학습함. 군집 분석(cluster analysis), 데이터 표현(data representation), 차원 감소(dimensionality reduction) 등을 학습할 수 있고, 지도학습을 보조하는 용도로도 사용함
 - 강화학습: 주어진 환경에서 에이전트가 환경과 상호작용 하고(행동 및 보상), 높은 보상을 획득하는 방법을 스스로 학습
 - 그 외: 레이블이 부족한 데이터를 학습하는 준 지도학습(semi-supervised learning), 학습법 자체를 학습하는 메타학습(meta learning), 반복 없이 한 번 혹은 몇 번만 보고 학습하는 원샷학습(one-shot learning), 한 문제를 위해 학습한 후 다른 문제에도 바로 적용하는 전이학습(transfer learning) 등이 있음
 - 딥러닝 구현을 위해서는 인간의 신경망을 모방한 인공 신경망을 주로 사용함
 - 딥러닝 구현을 위한 프레임워크로는 TensorFlow, Keras, Pytorch 등이 널리 사용됨

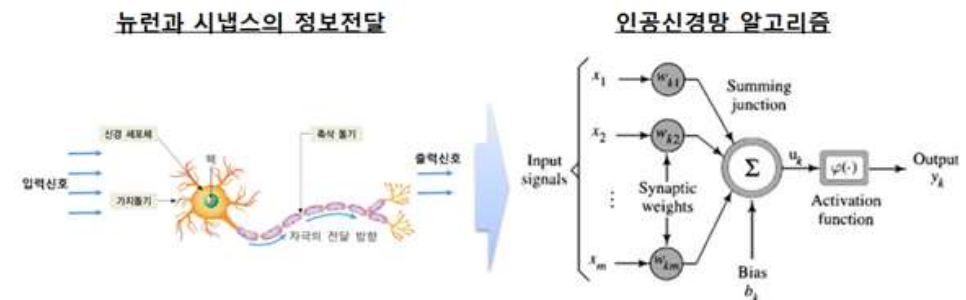
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

- 딥러닝을 활용한 잡음 제거: 인공지능/딥러닝 개요
 - 일반적인 인공신경망:
 - 입력층(1개), 은닉층(1개 또는 그 이상), 출력층(1개)으로 구성



- 각 Layer를 구성하는 노드는 직전 Layer로 부터의 입력 값에 가중치(weight)를 곱하고, 편향(bias)을 더한 결과를 활성화 함수(activation)로 전달하고, 그 결과를 다음 Layer로 전달함



- 오차함수와 역전파 알고리즘으로 인공신경망을 연결하는 가중치를 최적화하여 인공신경망의 성능을 최적화 함

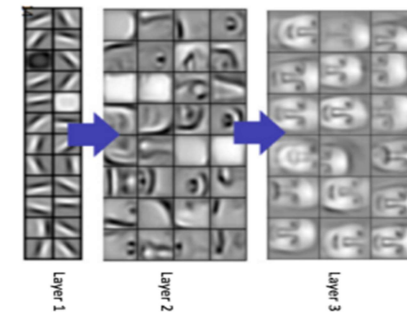
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

• 오토인코더(AutoEncoder, AE)

- 비지도학습(unsupervised learning)의 일종으로, 정답에 해당하는 레이블이 없음
- 입력 데이터의 효율적인 표현(representation)인 코딩(coding)을 학습할 수 있는 신경망
- 참고

- 인공신경망을 구성하는 각 Layer는 직전 Layer의 출력 값을 입력으로 해서, (오차 함수를 최소화 하는) 최적의 표현(representation)을 학습
- 예: 얼굴을 인식하고 분류하는 신경망은 각 계층별로 윤곽선 탐지(Layer 1), 눈/코/입/귀 등 얼굴을 구성하는 특징 탐지(Layer 2), 얼굴 전체를 탐지(Layer 3)

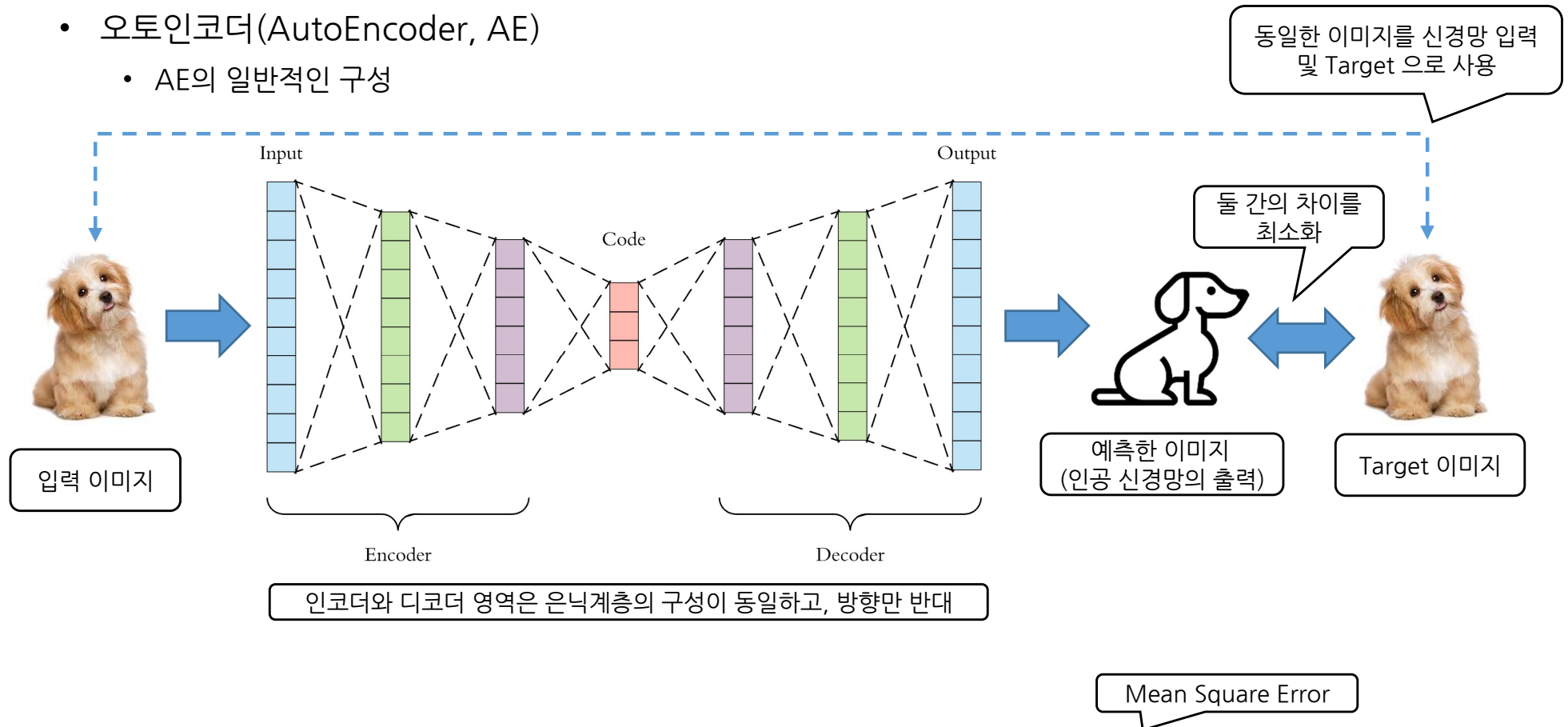


- 일반적으로 " 코딩 " 에 해당하는 표현은 입력보다 훨씬 낮은 차원을 가지므로 차원 축소에 유용하게 사용됨
 - 예를 들어, 100차원으로 구성된 데이터를 10차원으로 구성된 데이터로 차원을 축소하는 동시에 오차함수를 최소화 할 수 있음.
 - 이 경우, 100바이트를 전송하는 대신 10바이트만 전송해도 되고, 이를 통해 연산 효율성, 메모리 사용량 절약, 통신전력 사용량 감소, 네트워크 트래픽 감소 등의 장점을 얻을 수 있어, 자원 제약이 있는 사물인터넷 환경에서 큰 장점이 됨
 - 기존의 주성분 분석(Principal Component Analysis, PCA)을 활용한 일차원 데이터 처리 방식을 딥러닝 방식으로 확장한 것
- AE는 특성 추출기처럼 작동하기 때문에 심층 신경망의 비지도 사전훈련, 훈련 데이터와 유사한 새로운 데이터 생성, 차원 압축, 노이즈 제거 등의 분야에 활용할 수 있음

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

- 오토인코더(AutoEncoder, AE)
 - AE의 일반적인 구성



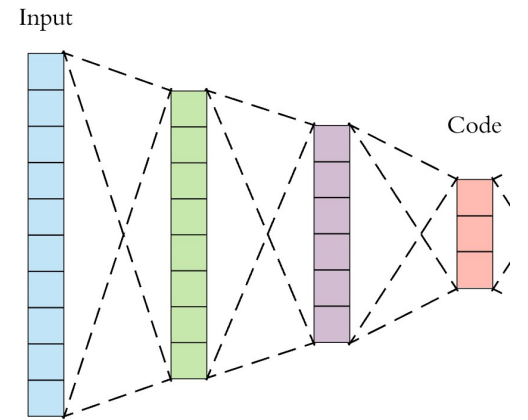
- AE 학습 목표는 오차함수(Input과 Output의 차이)를 최소화하는 것: $\text{minimize MSE}(\text{input}-\text{output})$
- 즉, 신경망 Input이 정답(레이블)으로 사용되며, 이 외에 별도의 정답이 주어지지 않으므로 비지도학습에 해당함

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE)

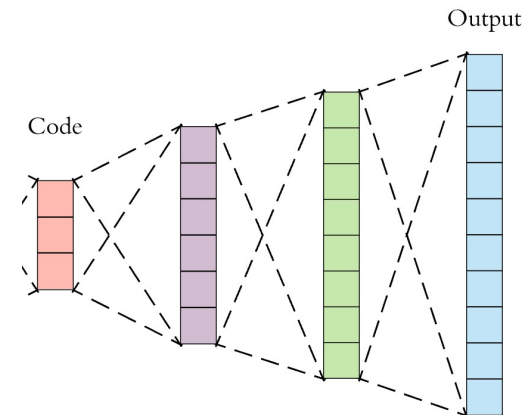
AE의 일반적인 구성:

- 인코더(앞부분): 각 계층의 크기(차원)가 갈수록 줄어듦
- 인코더의 다른 이름은 인지 네트워크 (recognition network)



공간의 크기(차원): 10 => 8 => 6 => 3

- 디코더(뒷부분): 인코더의 정 반대로 구성되어 있고, 입력과 동일한 결과를 출력하는 것을 목표로 함
- 디코더의 다른 이름은 생성 네트워크(generative network)



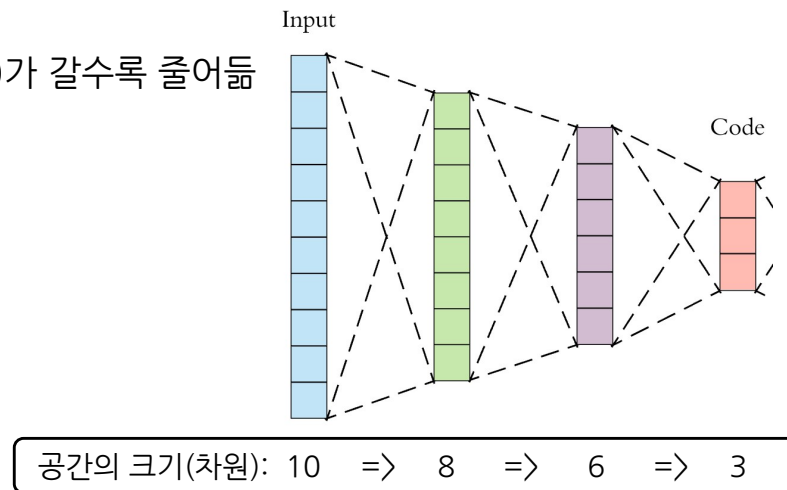
Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

- 오토인코더(AutoEncoder, AE)

AE의 일반적인 구성:

- 인코더(앞부분): 각 계층의 크기(차원)가 갈수록 줄어듦



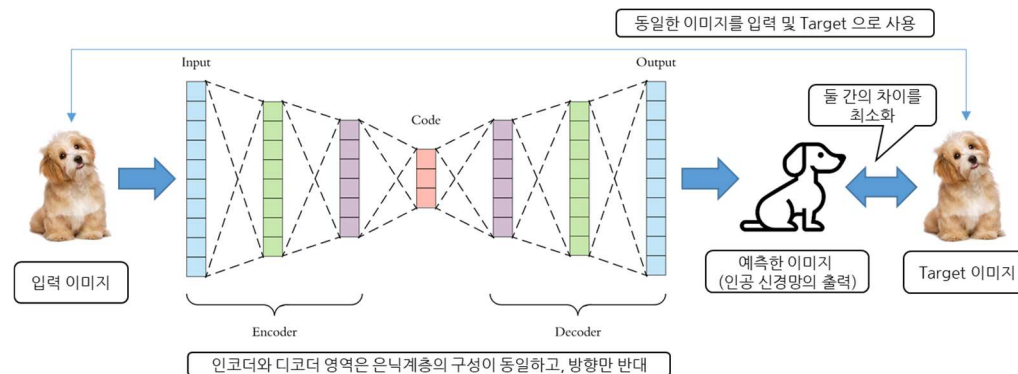
- 계층을 지날 수록 차원이 줄어든다는 것은?
 - 10개의 값으로 표현된 입력을 8개로 줄여서 표현하고, 그걸 다시 6개로 줄여서 표현하고, ..., 줄여서 3개로 표현
- 주어진 정보를 압축해서 표현한다는 것
 - 10Byte로 표현한 정보를 3Byte로 압축해서 표현하는 것과 유사(하지만, 일반적으로 사용하는 ZIP, RAR 등의 압축과는 다른 의미)
 - 예를 들어, 사람을 구분할 때, 체형, 키, 옷, 머리 색 등에 관한 정보는 버리고, 얼굴 생김새 정보만을 이용하는 것과 같은 원리
- 즉, 높은 차원으로 표현된 데이터를 낮은 차원의 잠재공간에서 높은 밀도로 표현하는 것

Sensing Layer: 데이터 정제

■ 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)

• 오토인코더(AutoEncoder, AE)

- AE의 앞부분 인코더는 입력된 정보를 저 차원의 정보로 압축하는 역할을 담당
- AE의 뒷부분 디코더는 압축된 정보를 입력으로, 원래의 정보로 복원하는 역할을 담당



• 정보의 손실

- 인공 신경망은 범용근사자(universal function approximator)로서, 근사치를 출력하기 때문에 AE의 출력이 입력 값과 완벽히 동일하기는 어려움
- 즉, 정보의 압축 및 복원 과정에서 정보의 손실이 발생할 수 있음: $\{(AE \text{의 입력}) - (AE \text{의 출력})\}^2 \geq 0$
- AE에서 오차값을 “복원오차” 또는 “정보손실(reconstruction loss)”이라고 표현함

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE)
 - 입력과 출력의 크기는 동일하고, 중간으로 갈수록 신경망의 차원이 줄어들어 정보의 압축 및 복원이 진행되고, 이 과정에서 정보의 손실이 발생함
 - AE 신경망 중간에서의 출력에 해당하는 “코드”는 최종적으로 압축된 정보를 나타냄
 - 정보의 손실이 발생할 때, AE는 “우선순위”를 두어, 덜 중요한 정보를 버리는 가공 작업을 수행함
 - 따라서, AE는 복잡한 데이터의 차원을 줄이는 목적으로도 사용하지만, 덜 중요한 잡음을 제거하는 목적으로도 사용함

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): 원본 이미지를 재현해내는 AE

```
"""  
참고: https://www.tensorflow.org/tutorials/generative/autoencoder?hl=ko  
"""
```

```
'\nhttps://www.tensorflow.org/tutorials/generative/autoencoder?hl=ko\n'
```

```
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import tensorflow as tf  
  
from sklearn.metrics import accuracy_score, precision_score, recall_score  
from sklearn.model_selection import train_test_split  
from tensorflow.keras import layers, losses  
from tensorflow.keras.datasets import fashion_mnist  
from tensorflow.keras.models import Model  
  
from keras.datasets import mnist
```

각 픽셀을 [0.0, 1.0] 범위로 스케일링

```
# MNIST 데이터 셋 다운로드  
# 패션 MNIST 데이터 셋  
(x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()  
# 손글씨 MNIST 데이터 셋  
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
  
# GrayScale 이미지를 [0.0, 1.0] 범위의 데이터로 변환  
x_train = x_train.astype('float32') / 255.  
x_test = x_test.astype('float32') / 255.  
  
# 데이터 수 및 차원 출력: 데이터 수(60,000 장), 데이터 차원(28 x 28 픽셀 이미지)  
print (x_train.shape)  
print (x_test.shape)
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): 원본 이미지를 재현해내는 AE

인코더 신경망

디코더 신경망

```
# AutoEncoder 신경망 구성하기

code_dimension = 12 # AE 중간 코드의 차원 정의
input_image_dim = 28*28
hidde1_layer_1_dim = 256
hidde1_layer_2_dim = 64

class Autoencoder(Model):
    def __init__(self):
        super(Autoencoder, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Flatten(),
            layers.Dense(hidde1_layer_1_dim, activation='relu'),
            layers.Dense(hidde1_layer_2_dim, activation='relu'),
            layers.Dense(code_dimension, activation='relu'),
        ])
        self.decoder = tf.keras.Sequential([
            layers.Dense(hidde1_layer_2_dim, activation='relu'),
            layers.Dense(hidde1_layer_1_dim, activation='relu'),
            layers.Dense(input_image_dim, activation='sigmoid'),
            layers.Reshape((28, 28))
        ])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

autoencoder = Autoencoder()
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): 원본 이미지를 재현해내는 AE

MSE 손실 함수

```
# optimizer와 loss function 정의
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
"""
x_train을 입력과 대상으로 사용하여 모델을 훈련합니다.
encoder는 데이터셋을 784차원에서 잠재 공간으로 압축하는 방법을 배우고,
decoder는 원본 이미지를 재구성하는 방법을 배웁니다.
"""
```

```
autoencoder.fit(x_train, x_train,
                epochs=10,
                shuffle=True,
                validation_data=(x_test, x_test))
```

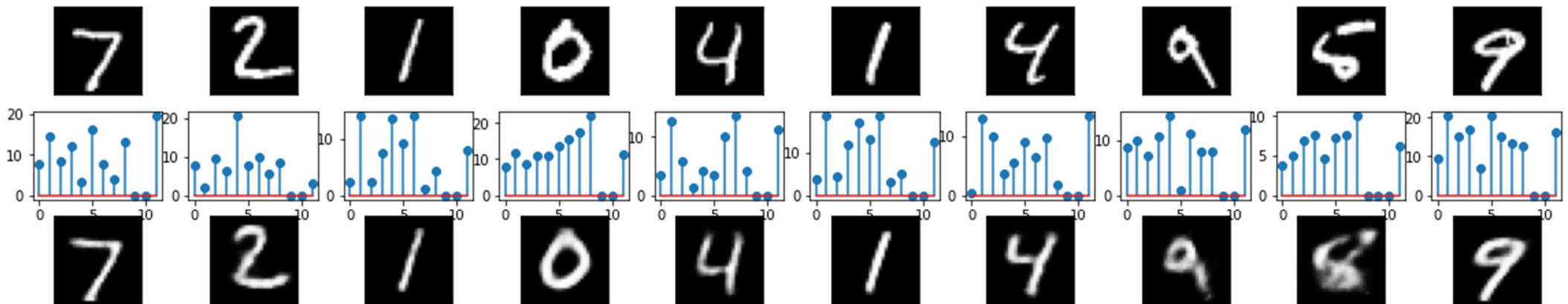
```
Epoch 1/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0332 - val_loss: 0.0222
Epoch 2/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0203 - val_loss: 0.0185
Epoch 3/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0180 - val_loss: 0.0170
Epoch 4/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0168 - val_loss: 0.0164
Epoch 5/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0161 - val_loss: 0.0161
Epoch 6/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0156 - val_loss: 0.0154
Epoch 7/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0152 - val_loss: 0.0150
Epoch 8/10
1875/1875 [=====] - 6s 3ms/step - loss: 0.0149 - val_loss: 0.0148
Epoch 9/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0146 - val_loss: 0.0146
Epoch 10/10
1875/1875 [=====] - 5s 3ms/step - loss: 0.0143 - val_loss: 0.0145
```


Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): 원본 이미지를 재현해내는 AE

```
# 모델이 훈련되었으므로 테스트 세트에서 이미지를 인코딩 및 디코딩하여 테스트
encoded_imgs = autoencoder.encoder(x_test).numpy()
decoded_imgs = autoencoder.decoder(encoded_imgs).numpy()
```

- 테스트 결과 출력: 원본(상), 인코딩 된 결과(중), 인공신경망 출력(하)



Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf

from sklearn.metrics import accuracy_score, precision_score, recall_score
from sklearn.model_selection import train_test_split
from tensorflow.keras import layers, losses
from tensorflow.keras.datasets import fashion_mnist
from tensorflow.keras.models import Model

# MNIST 패션 데이터셋 사용
(x_train, _), (x_test, _) = fashion_mnist.load_data()

x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.

print (x_train.shape)
print (x_test.shape)
```

Sensing Layer: 데이터 정제

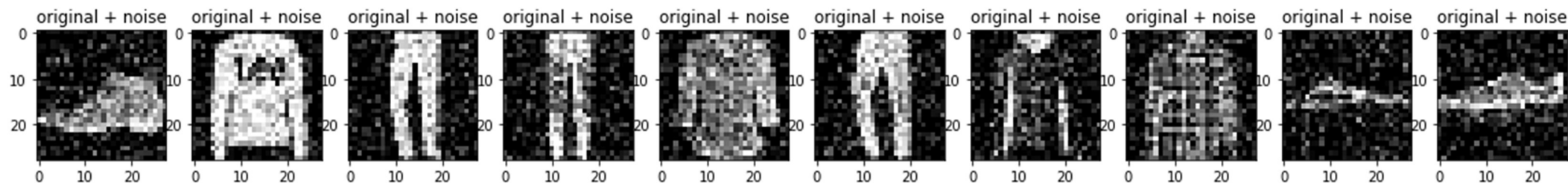
- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE

원본 이미지에 무작위로
노이즈를 추가

```
# 이미지에 임의의 노이즈를 추가
noise_factor = 0.2
x_train_noisy = x_train + noise_factor * tf.random.normal(shape=x_train.shape)
x_test_noisy = x_test + noise_factor * tf.random.normal(shape=x_test.shape)

x_train_noisy = tf.clip_by_value(x_train_noisy, clip_value_min=0., clip_value_max=1.)
x_test_noisy = tf.clip_by_value(x_test_noisy, clip_value_min=0., clip_value_max=1.)

# 노이즈가 있는 이미지를 출력
n = 10
plt.figure(figsize=(20, 2))
for i in range(n):
    ax = plt.subplot(1, n, i + 1)
    plt.title("original + noise")
    plt.imshow(tf.squeeze(x_test_noisy[i]))
    plt.gray()
plt.show()
```



노이즈가 추가된 이미지

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE

인코더 신경망

디코더 신경망

```
# CNN을 이용한 autoencoder 정의
class Denoise(Model):
    def __init__(self):
        super(Denoise, self).__init__()
        self.encoder = tf.keras.Sequential([
            layers.Input(shape=(28, 28, 1)),
            layers.Conv2D(16, (3,3), activation='relu', padding='same', strides=2),
            layers.Conv2D(8, (3,3), activation='relu', padding='same', strides=2)])

        self.decoder = tf.keras.Sequential([
            layers.Conv2DTranspose(8, kernel_size=3, strides=2, activation='relu', padding='same'),
            layers.Conv2DTranspose(16, kernel_size=3, strides=2, activation='relu', padding='same'),
            layers.Conv2D(1, kernel_size=(3,3), activation='sigmoid', padding='same')])

    def call(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

autoencoder = Denoise()
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE

```
autoencoder.compile(optimizer='adam', loss=losses.MeanSquaredError())
```

```
autoencoder.fit(x_train_noisy, x_train,  
                epochs=10,  
                shuffle=True,  
                validation_data=(x_test_noisy, x_test))
```

```
Epoch 1/10  
1875/1875 [=====] - 19s 4ms/step - loss: 0.0184 - val_loss: 0.0111  
Epoch 2/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0100 - val_loss: 0.0094  
Epoch 3/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0090 - val_loss: 0.0087  
Epoch 4/10  
1875/1875 [=====] - 7s 4ms/step - loss: 0.0085 - val_loss: 0.0084  
Epoch 5/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0083 - val_loss: 0.0082  
Epoch 6/10  
1875/1875 [=====] - 7s 4ms/step - loss: 0.0081 - val_loss: 0.0083  
Epoch 7/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0080 - val_loss: 0.0079  
Epoch 8/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0079 - val_loss: 0.0079  
Epoch 9/10  
1875/1875 [=====] - 7s 4ms/step - loss: 0.0078 - val_loss: 0.0079  
Epoch 10/10  
1875/1875 [=====] - 6s 3ms/step - loss: 0.0078 - val_loss: 0.0078
```

Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE

```
# encoder의 요약: 이미지가 28x28에서 7x7로 어떻게 다운샘플링되는지 확인
autoencoder.encoder.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 16)	160
conv2d_1 (Conv2D)	(None, 7, 7, 8)	1160

```
=====
Total params: 1,320
Trainable params: 1,320
Non-trainable params: 0
=====
```

```
# decoder의 요약: 이미지를 7x7에서 28x28로 다시 업샘플링함
autoencoder.decoder.summary()
```

```
Model: "sequential_1"
```

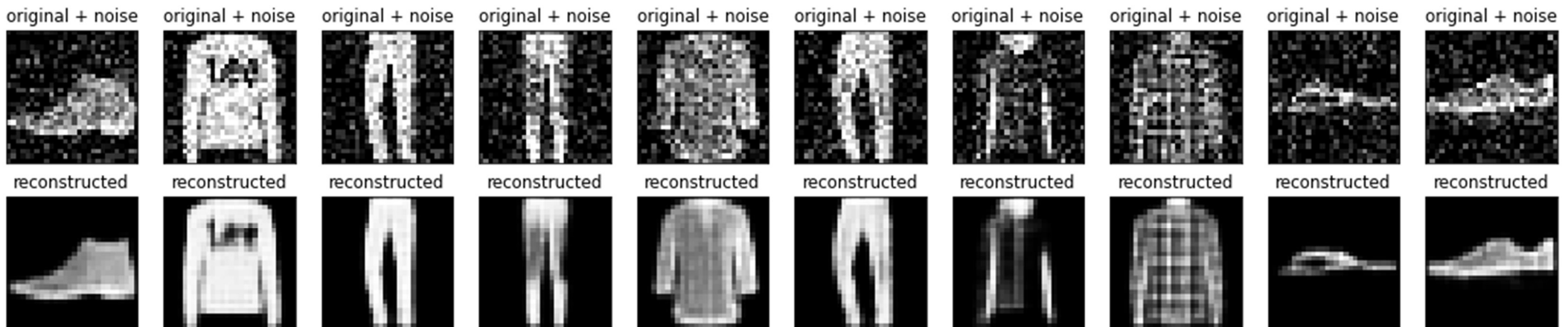
Layer (type)	Output Shape	Param #
conv2d_transpose (Conv2DTranspose)	(None, 14, 14, 8)	584
conv2d_transpose_1 (Conv2DTranspose)	(None, 28, 28, 16)	1168
conv2d_2 (Conv2D)	(None, 28, 28, 1)	145

```
=====
Total params: 1,897
Trainable params: 1,897
Non-trainable params: 0
=====
```


Sensing Layer: 데이터 정제

- 데이터 정제(Data cleaning/cleansing/scrubbing): 5. Noise reduction (잡음 제거)
 - 오토인코더(AutoEncoder, AE): CNN 기반의 잡음 제거 AE
 - 입력 이미지(원본+노이즈) 및 AE의 출력 이미지를 플롯

(위) 입력으로 사용한 Noisy Images



(아래) 잡음 제거 AE의 출력 이미지

추가 토픽: 센싱 값 분석 및 예측

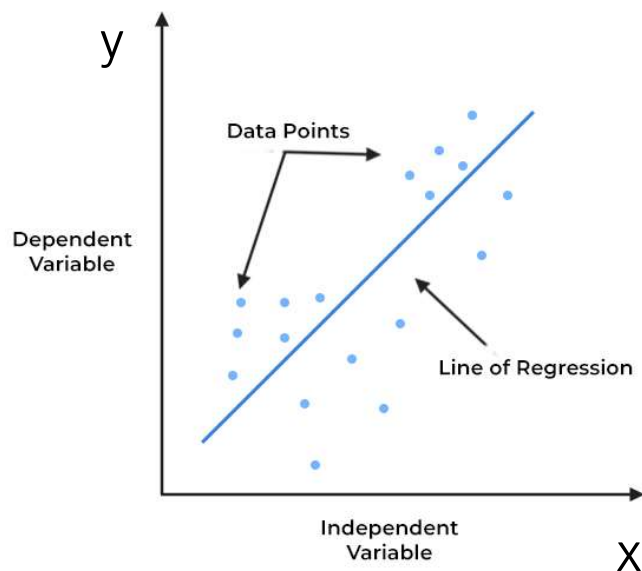
Sensing Layer: 데이터 정제

■ 추가 토픽: 센싱 값 분석/예측

- 수집한 센서 값을 기반으로 추세/트렌드를 분석하여 누락된 값을 예측하여 복원하거나, 가까운 미래의 센서 값을 예측하기
- 추세/트렌드를 분석하고 예측하는 기본적인 기법: 회귀(Regression)

■ 회귀/Regression

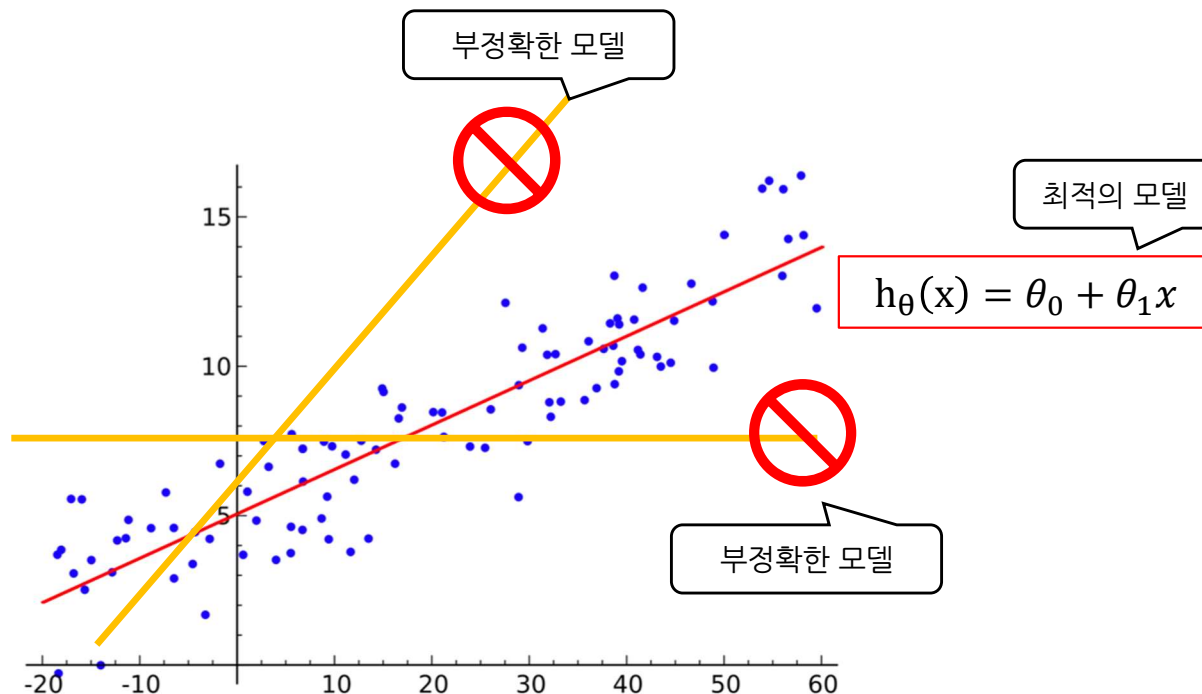
- 한 개 또는 여러 개의 독립변수 x 와 한 개의 종속변수 $y = f(x)$ 간의 상관관계를 모델링하는 기법
- 독립변수가 종속변수에 미치는 영향을 파악하고, 이를 통해 독립변수의 일정한 값에 대응하는 종속변수의 값을 예측하는 모형을 산출하는 방법
 - 예: 월세 {방 1, 화장실 1}=30, {방 2, 화장실 1}=40, {방 3, 화장실 2}=60, ... 일 때, {방 5, 화장실 3}의 월세는?



Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

- 선형 회귀 분석 기법 중에서 가장 기본이 되는 분석 법
- 단순 선형 회귀(Simple Linear Regression)
 - 하나의 독립변수(x)와 이에 대응하는 종속변수(y)간의 관계를 분석하는 기법
 - $h_{\theta}(x) = \theta_0 + \theta_1 x$ 로 표현되는 직선 그래프 모델의 파라미터(θ_0, θ_1)를 최적화하여 전체 데이터를 대표하는 직선 모델 찾기
 - 직선 모델은 절편(θ_0)과 기울기(θ_1)로 정의할 수 있고, 최적의 (θ_0, θ_1)를 찾는 것이 단순 선형 회귀 분석의 목표
 - 주의: (x, y)는 주어진 값이고, 최적화를 통해서 계산해야 하는 값은 모델 파라미터인 (θ_0, θ_1)임.



Sensing Layer: 데이터 정제

■ 회귀를 이용한 센싱 값 분석/예측

• 단순 선형 회귀(Simple Linear Regression)

- 데이터 준비: scikit-learn 패키지에서 제공하는 당뇨병 데이터 셋

```
from sklearn.datasets import load_diabetes

# 당뇨병 데이터 셋 불러오기
diab = load_diabetes()

# 데이터 셋 크기 확인
print(diab.data.shape, diab.target.shape)
```

```
(442, 10) (442,)
```

- 442개의 데이터 샘플로 구성
- 각 데이터 샘플은 10개의 feature로 구성
- 442개의 각 데이터 샘플에 대응하는 Target값도 442개

****Data Set Characteristics:****

:Number of Instances: 442

:Number of Attributes: First 10 columns are numeric predictive values

:Target: Column 11 is a quantitative measure of disease progression one year after baseline

:Attribute Information:

- age age in years
- sex
- bmi body mass index
- bp average blood pressure
- s1 tc, T-Cells (a type of white blood cells)
- s2 ldl, low-density lipoproteins
- s3 hdl, high-density lipoproteins
- s4 tch, thyroid stimulating hormone
- s5 ltg, lamotrigine
- s6 glu, blood sugar level

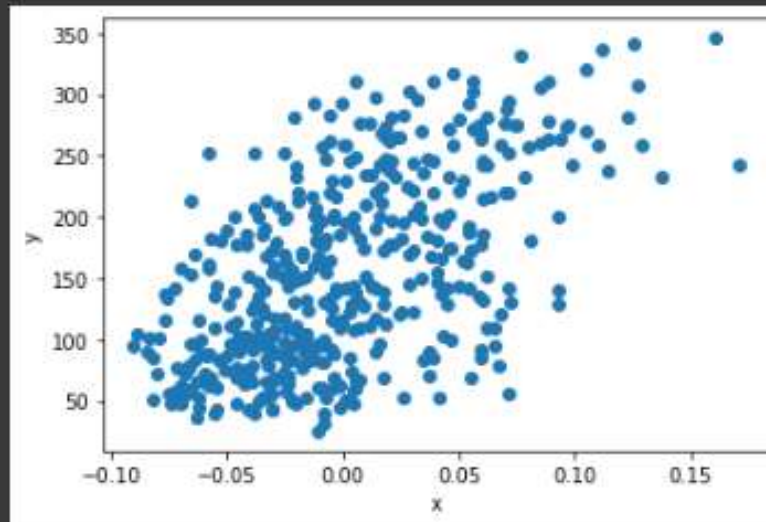
Note: Each of these 10 feature variables have been mean centered and scaled by the standard deviation times 'n_samples' (i.e. the sum of squares of each column totals 1).

Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 단순 선형 회귀(Simple Linear Regression)
 - 당뇨병 데이터 셋에서 3번째 feature 인 BMI(x 축) 와 target 값(y 축) plot 하기

```
# 데이터 셋의 일부를 plot 하기: body mass index vs target(y)
# 독립변수(x)와 종속변수(y)간에 정비례(선형) 관계가 있는 것을 알 수 있음
import matplotlib.pyplot as plt

plt.scatter(diab.data[:,2], diab.target[:])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

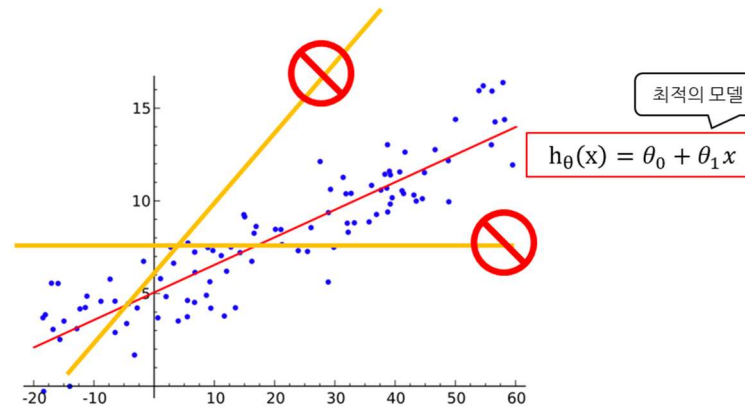
단순 선형 회귀(Simple Linear Regression)

데이터 셋 설명:

- x := 당뇨병 데이터 셋에서 3번째 feature 인 BMI 값
- y := 각 데이터 샘플에 대응하는 target 값

모델 정의하기: $h_{\theta}(x) = \theta_0 + \theta_1 x$

- 주어진 값: (x, y)
- 최적화를 통해서 우리가 찾고자 하는 값: (θ_0, θ_1)
- (θ_0, θ_1) 값에 따라 다양한 직선 그래프가 만들어지며, 이 중에서 최적의 모델을 구성하는 최적의 (θ_0, θ_1) 쌍을 찾아야 함



최적의 모델이란?

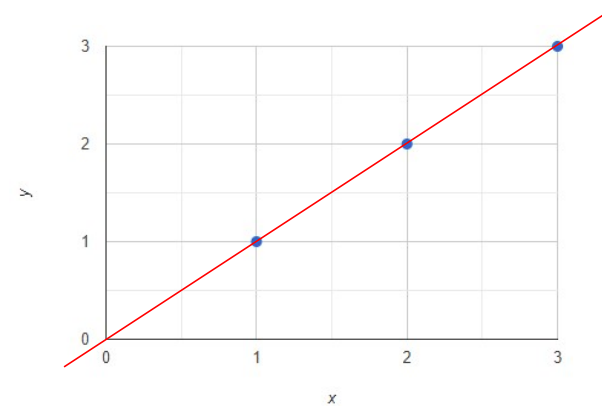
Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

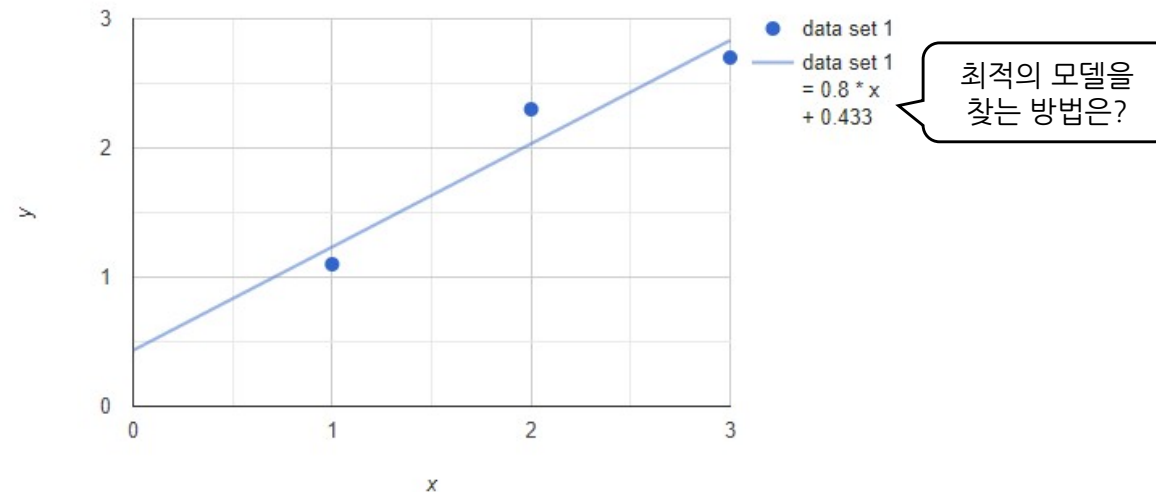
단순 선형 회귀(Simple Linear Regression)

최적 모델

- 주어진 데이터 분포를 최소한의 오차로 대표하는 직선 그래프
- 예:
 - (1,1), (2,2), (3,3) 데이터가 주어진 경우
 - 최적의 직선 그래프는 $h_{\theta}(x) = 0 + 1x$
 - 데이터를 대표하는 선형 모델이 필요한 이유?
 - 누락된 값 예측: 누락된 $x = 1.5$ 에 대해, y 는 얼마일지 예측
 - 미래의 값 예측: $x = 4$ 일 때 y 는 얼마일지 예측



- 하지만, 대부분의 경우 위의 그래프처럼 모든 데이터 포인트를 정확하게 지나가는 직선을 찾기 어려운 경우가 많음

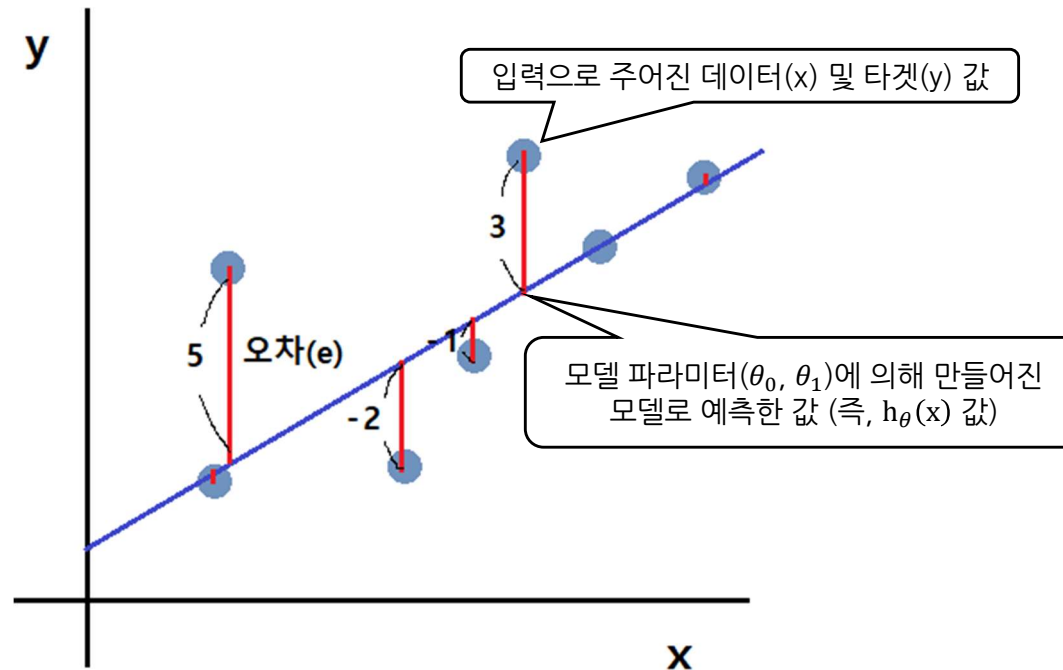


Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

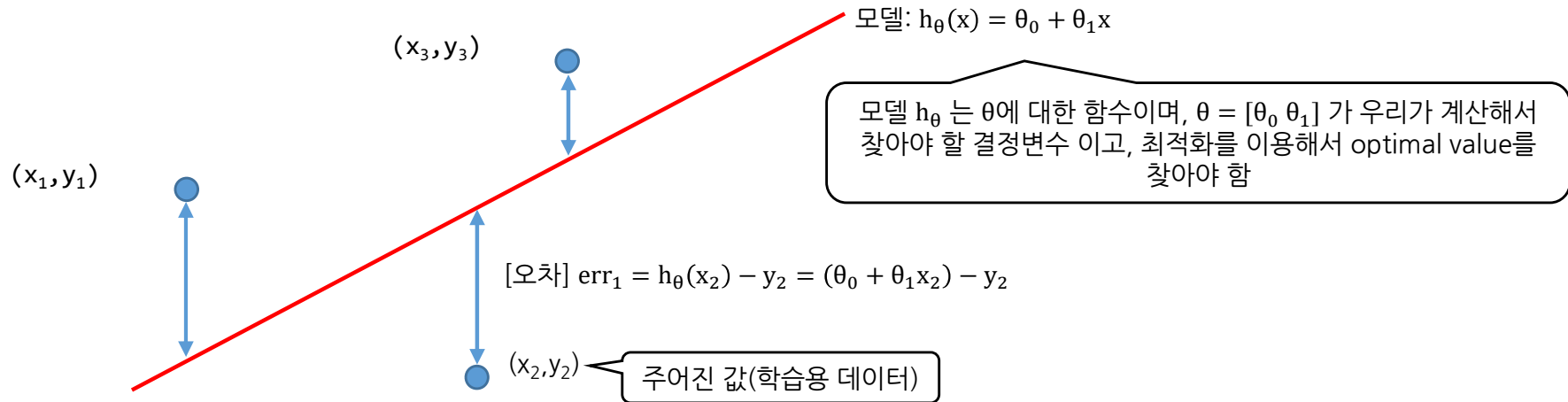
단순 선형 회귀(Simple Linear Regression)

- 최적 모델을 찾기 위해서는 오차 함수(Loss Function)를 정의하고, 오차 함수를 최소화 하는 모델을, 즉 (θ_0, θ_1) 를 찾는 방식으로 접근해야 함
- 오차: 실제 target 값과 모델이 예측한 값의 차이
 - x값에 대한 실제 target 값: x값에 대응하는 y값 (주어진 값)
 - x값에 대한 예측 값: $h_{\theta}(x) = \theta_0 + \theta_1 x$ (x: 주어진 데이터, $h_{\theta}(x)$ 는 모델로 예측한 값)
 - 오차: error $e = (y - h_{\theta}(x))^2$ # 오차가 항상 양의 값을 갖도록 오차의 제곱을 사용 (예: 오차 -5와 오차 +5의 합이 0이 되는 현상 방지)



Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 단순 선형 회귀(Simple Linear Regression)
 - 오차 정의하기



- 전체 m 개의 training data 에 대한 오차의 총합: $\sum_{i=1}^m err_i^2 = \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$ // $i := i$ 번째 학습 샘플 인덱스
- $\sqrt{z^2} = z$ 인 성질을 이용하면...

$$\sqrt{\sum_{i=1}^m err_i^2} = \sqrt{\sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2} = \left\| \begin{bmatrix} h_{\theta}(x_1) - y_1 \\ \vdots \\ h_{\theta}(x_m) - y_m \end{bmatrix} \right\|_2 = \left\| \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} - \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \right\|_2 = \|A\theta - b\|_2^2$$

벡터 θ

L2 norm (= convex function)

Row vector (열 벡터)

행렬 A

벡터 b

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

단순 선형 회귀(Simple Linear Regression)

최적의 모델 파라미터 θ 찾기

- Training data 에 주어진 (x, y) 값과의 차이(=오차)를 최소화 하는 직선 찾기
- 전체 m 개의 training data 에 대한 오차의 제곱의 총합을 최소화

- $\underset{\theta}{\text{minimize}} \|A\theta - b\|_2^2$

→ Mean Square Error로 변형: $\underset{\theta}{\text{minimize}} \frac{1}{m} \|A\theta - b\|_2^2$

→ 1/2을 곱한 형태로 변형: $\underset{\theta}{\text{minimize}} \frac{1}{2} \frac{1}{m} \|A\theta - b\|_2^2$

오차 함수 앞에 $1/m$, $1/2$ 등의 상수를 곱하더라도, 최적의 θ 값에는 아무런 영향이 없음.

- 오차함수 $J = \frac{1}{2} \frac{1}{m} \|A\theta - b\|_2^2$

- 최적의 모델 파라미터 θ 를 계산하는 방법은?

Sensing Layer: 데이터 정제

■ 회귀를 이용한 센싱 값 분석/예측

- 단순 선형 회귀(Simple Linear Regression)

최적의 모델 파라미터 θ 를 계산하는 방법 1: $A\theta - b$ 를 0으로 만드는 θ 를 직접 계산하기

- A 매트릭스가 invertible 인 경우:

- A가 square matrix 이고 (# rows = # columns) Full rank 일 때 (all m rows are independent)
- $\theta^* = A^{-1}b$

- A 매트릭스의 inverse를 구할 수 없는 경우:

- Pseudo inverse를 사용
- $\theta^* = (A^T A)^{-1} A^T b$

- 단, n-by-n 행렬의 역행렬을 계산하는 연산의 시간 복잡도는 $O(n^3)$ 으로, 시간 복잡도가 높음

- 시간 복잡도가 더 낮은 방법은?

Sensing Layer: 데이터 정제

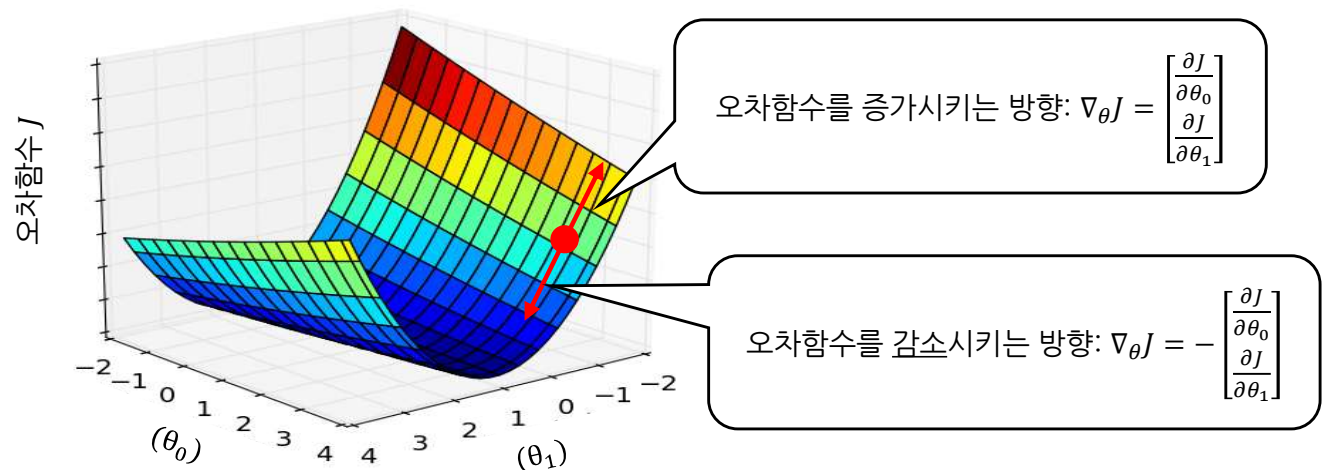
회귀를 이용한 센싱 값 분석/예측

• 단순 선형 회귀(Simple Linear Regression)

최적의 모델 파라미터 θ 를 계산하는 방법 2: Gradient Descent 기법으로 점진적으로 최적의 파라미터 찾기

- 모델 파라미터 (θ_0, θ_1) 의 변화에 따른 오차함수 (J) 값의 변화

L2 norm은 convex function 이므로,
local optimum = global optimum



- 오차함수를 최소화 하는 모델 파라미터 (θ_0, θ_1) 를 찾기 위해서는, 점진적으로 오차함수를 감소시키는 방향으로 모델 파라미터 (θ_0, θ_1) 를 이동시켜야 함
- 즉, J 의 기울기의 반대 방향으로 모델 파라미터 (θ_0, θ_1) 를 점진적으로 이동시켜야 함

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

단순 선형 회귀(Simple Linear Regression)

최적의 모델 파라미터 θ 를 계산하는 방법 2: Gradient Descent 기법으로 점진적으로 최적의 파라미터 찾기

• 목적 함수 : $\min_{\theta} \frac{1}{2m} \|A\theta - b\|_2^2$

• 목적 함수의 gradient:

$$\nabla_{\theta} L(\theta) = \begin{bmatrix} \frac{\partial L(\theta)}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) \\ \frac{\partial L(\theta)}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i \end{bmatrix}$$

Decision variable θ update 규칙


$$\theta_0 = \theta_0 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)$$

$$\theta_1 = \theta_1 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$$

• 초기 θ 값은 0으로 설정하거나, 임의의 난수로 설정함.

• α 는 step-size로서, (0,1)사이에서 작은 상수 값을 사용하거나, $\alpha^t = \frac{1+k}{t+k}$, for $k \in \mathbb{R}_+$ 형태의 점진적으로 감소하는 값을 사용함(t: iteration counter)

• 주의 : 두개의 θ 를 동시에 업데이트 해야 함

Correct	Wrong
$t_0 = \theta_0 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)$ $t_1 = \theta_1 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$ $\theta_0 = t_0$ $\theta_1 = t_1$	$\theta_0 = \theta_0 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)$ $\theta_1 = \theta_1 - \alpha \times \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i) x_i$ 

θ_1 을 업데이트 할 때, 직전에 업데이트 한 θ_0 값에 영향을 받음

Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 단순 선형 회귀(Simple Linear Regression)
 - 구현하기: 사이킷런의 LinearRegression

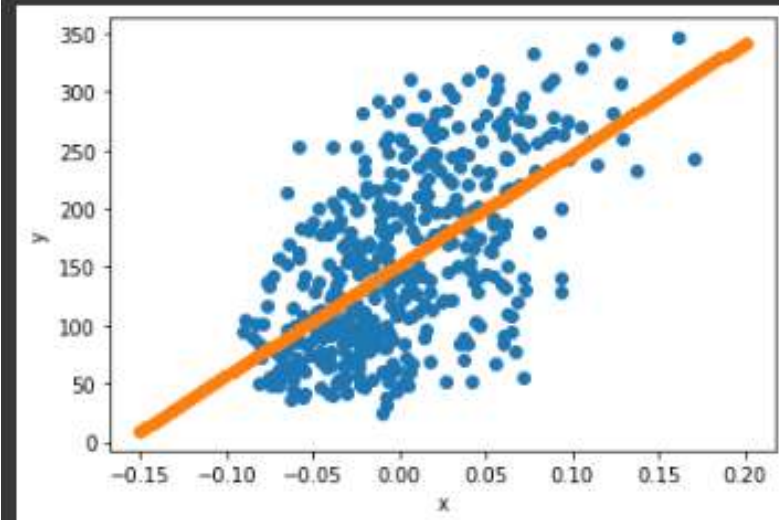
```
# BMI와 Target 간의 분석을 위한 데이터 준비
x = diab.data[:,2]
y = diab.target
```

```
# 사이킷런의 LinearRegression
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(x.reshape(-1,1),y.reshape(-1,1))
print("Model: optimal slope is ", model.coef_)
print("Model: optimal y-intercept is ", model.intercept_)

Model: optimal slope is [[949.43526038]]
Model: optimal y-intercept is [152.13348416]
```

```
# 모델에 해당하는 직선 그래프 그리기
import numpy as np
x_values = np.linspace(-0.15, 0.20, 100)
y_pred = model.predict(x_values.reshape(-1,1))
```

```
plt.scatter(diab.data[:,2], diab.target[:])
plt.scatter(x_values, y_pred)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



```
# 최적화된 모델로, 마지의 값 예측하기
prediction = model.predict(np.array([0.20]).reshape(-1,1))
print('x=0.20에 대한 y의 예측값: %2.2f' % prediction[0][0])
```

Sensing Layer: 데이터 정제

■ 회귀를 이용한 센싱 값 분석/예측

• D차원 선형 회귀

- 지금까지의 모델은 $h_{\theta}(x) = \theta_0 + \theta_1 x$ 로, 단일 차원의 스칼라(scalar) 독립변수(x)만을 사용해서 정의했음
- 하지만, D개의 독립변수의 선형 조합으로 회귀 모델을 생성하는 것이 필요할 수 있음

- $h_{\theta}(\underline{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_D x_D$

- 단순 선형 회귀에서 x는 스칼라 값(\mathbb{R}^1)이었지만, D차원 선형회귀에서는 $x \in \mathbb{R}^D$
- θ_0 는 절편이며, x 변수가 곱해지지 않음

- 수식 전개 편의성을 위해 절편(θ_0)을 제거한 형태를 고려함: $h_{\theta}(\underline{x}) = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_D x_D$

- 절편이 없으므로, $h_{\theta}(x)$ 가 형성하는 D차원 평면은 원점을 지나는 평면이 됨
- 즉, $x = [0, 0, \dots, 0]$ 을 대입하면 $h_{\theta}(x) = 0$

- 행렬 표기법으로 바꿔 쓰면, $h_{\theta}(x) = \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_D x_D = \underline{\theta}^T \underline{x} = [\theta_1 \quad \dots \quad \theta_D] \begin{bmatrix} x_1 \\ \dots \\ x_D \end{bmatrix}$

- 최적의 모델 파라미터 벡터 θ 를 구하기 위해서, 오차 함수를 정의: $J(\theta) = \frac{1}{N} \sum_{n=0}^{N-1} (h_{\theta}(\underline{x}_n) - y_n)^2 = \frac{1}{N} \sum_{n=0}^{N-1} (\underline{\theta}^T \underline{x}_n - y_n)^2$

- N : 샘플의 수
- \underline{x}_n : n 번째 샘플에 해당하는 x 벡터 (\mathbb{R}^D)
- $x_{n,i}$: \underline{x}_n 벡터 내의 i 번째 값
- y_n : n 번째 샘플에 대한 y 값 (label 또는 target 값)

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

D차원 선형 회귀

- 최적의 모델 파라미터 벡터 θ 를 구하기 위해서, 오차 함수를 정의: $J(\theta) = \frac{1}{N} \sum_{n=0}^{N-1} (h_{\theta}(\underline{x}_n) - y_n)^2 = \frac{1}{N} \sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n)^2$
- 오차 함수를 θ_i 로 편미분: $\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{N} \sum_{n=0}^{N-1} \frac{\partial}{\partial \theta_i} (\theta^T \underline{x}_n - y_n)^2 = \frac{2}{N} \sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) x_{n,i}$
 - 참고로, $\theta^T \underline{x}_n = [\theta_1 x_{n,1} + \theta_2 x_{n,2} + \dots + \theta_D x_{n,D}]$ 을 θ_i 로 편미분하면 $x_{n,i}$ 만 남음
- 오차 함수 $J(\theta)$ 를 최소화 하는 θ 는 모든 θ_i 방향에 대한 기울기가 0인 경우, 즉 편미분 식이 모든 i 에 대해서 0이 되는 경우
이므로, 다음의 조건을 만족: $i = 1, 2, \dots, D$ 에 대하여 $\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) x_{n,i} = 0$ (양변에 $\frac{N}{2}$ 를 곱함)

즉,

$$\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) x_{n,0} = 0$$

$$\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) x_{n,1} = 0$$

...

$$\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) x_{n,D} = 0$$

- 이는, $\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) [x_{n,0}, x_{n,1}, \dots, x_{n,D}] = [0, 0, \dots, 0]$ 과 같고, $\sum_{n=0}^{N-1} (\theta^T \underline{x}_n - y_n) \underline{x}_n^T = [0, 0, \dots, 0]$ 로 쓸 수 있음
- $(a + b)c = ac + bc$ 와 같이 \underline{x}_n^T 에 대해서 분배법칙을 적용하면 $\sum_{n=0}^{N-1} (\theta^T \underline{x}_n \underline{x}_n^T - y_n \underline{x}_n^T)$ 이고, 이를 분해하면

$$\theta^T \sum_{n=0}^{N-1} \underline{x}_n \underline{x}_n^T - \sum_{n=0}^{N-1} y_n \underline{x}_n^T = [0, 0, \dots, 0]$$

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

D차원 선형 회귀

- $X = \begin{bmatrix} x_{0,1} & x_{0,2} & \dots & x_{0,D} \\ x_{1,1} & x_{1,2} & \dots & x_{1,D} \\ \dots & \dots & \dots & \dots \\ x_{N-1,1} & x_{N-1,2} & \dots & x_{N-1,D} \end{bmatrix}$ 로 정의하면, $\sum_{n=0}^{N-1} \underline{x}_n \underline{x}_n^T = X^T X$ 이고, $\sum_{n=0}^{N-1} y_n \underline{x}_n^T = \underline{y}^T X$ 이므로,
- $\underline{\theta}^T \sum_{n=0}^{N-1} \underline{x}_n \underline{x}_n^T - \sum_{n=0}^{N-1} y_n \underline{x}_n^T = \underline{\theta}^T X^T X - \underline{y}^T X = [0, 0, \dots, 0]$ 를 얻음
- 양변을 전치(transpose)하면 $(\underline{\theta}^T X^T X - \underline{y}^T X)^T = [0, 0, \dots, 0]^T$
- $(A^T)^T = A$, $(AB)^T = B^T A^T$ 이고, $\underline{\theta}^T = A$, $X^T X = B$ 로 두면, $(X^T X)^T (\underline{\theta}^T)^T - X^T \underline{y} = [0, 0, \dots, 0]^T$ 이므로, 아래와 같이 쓸 수 있음: $(X^T X) \underline{\theta} = X^T \underline{y}$
- 마지막으로, 양 변에 $(X^T X)^{-1}$ 을 곱하면, $\underline{\theta} = (X^T X)^{-1} X^T \underline{y}$ 이고, 이것은 오차 함수를 최소화 하는 최적의 모델 파라미터에 해당함
- 절편을 추가하기 위해서는 $\theta^T = [\theta_0, \theta_1, \dots, \theta_D]$, $x^T = [1, \theta_1, \dots, \theta_D]$ 로 확장하면 됨

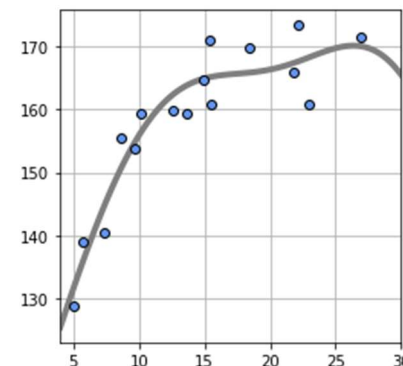
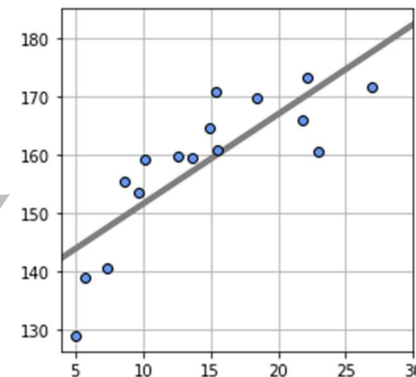
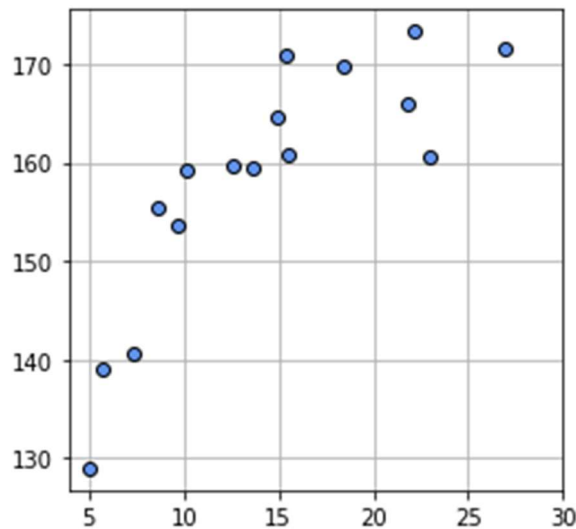
- 단순 선형 회귀 예제에서 사용한 사이킷런의 LinearRegression은 D차원 선형회귀에도 사용할 수 있음

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

선형 기저 함수 모델

- D차원 선형 회귀 모델은 단순 선형 회귀 모델에 비해, feature 와 target 간 복잡한 관계를 학습할 수 있음.
- 하지만, D차원 선형 회귀에서 생성하는 모델은 여전히 평면(hyperplane) 모델이며, 이를 통해 표현할 수 있는 함수에는 제약이 있음
- 따라서, 직선 또는 평면 형태 이외의 비선형의 형태를 가진 추세/트렌드를 분석하기에는 부적합함



Sensing Layer: 데이터 정제

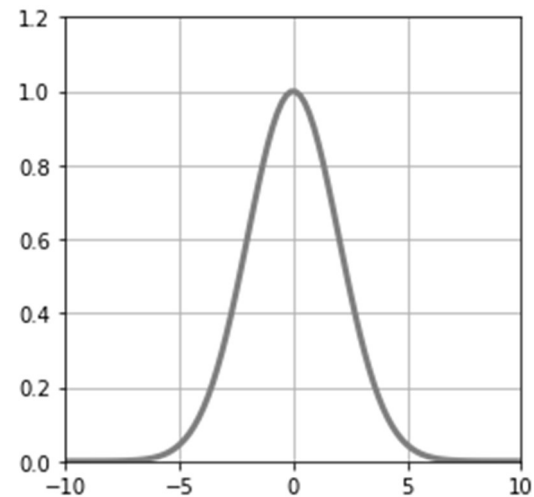
회귀를 이용한 센싱 값 분석/예측

선형 기저 함수 모델

- 비 선형(예: 곡선)으로 표현되는 추세를 나타내기 위한 방법 중 하나
- 기저 함수(basis function)라는 것은 기본이 되는 함수를 말하고, 이러한 기저 함수의 선형 조합을 통해 곡선 등 비선형 모델을 생성할 수 있음
- 가우스 함수를 기저 함수로 사용하고, 서로 다른 (평균, 표준편차)를 가지는 가우스 함수를 선형으로 조합함
- j 번째 기저 함수는 $\phi_j(x)$ 로 나타내고, 평균 μ_j 및 표준편차 s 를 가지는 가우스 함수의 경우 아래와 같이 표현함

$$\phi_j(x) = \exp \left\{ -\frac{(x - \mu_j)^2}{2s^2} \right\}$$

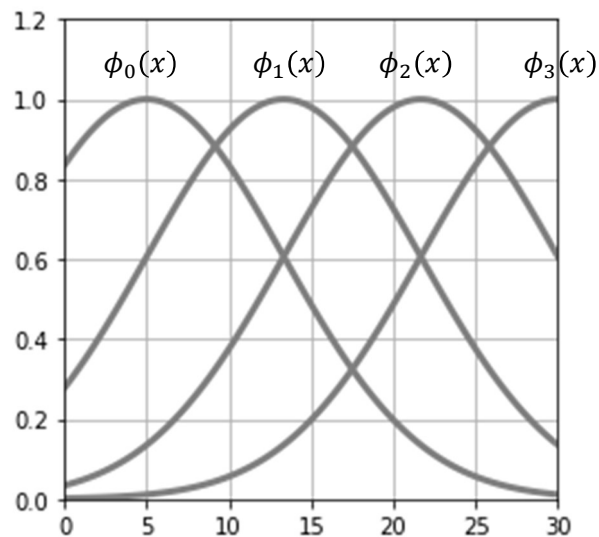
- 예) mean=0, std_dev=2 인 가우스 함수



Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

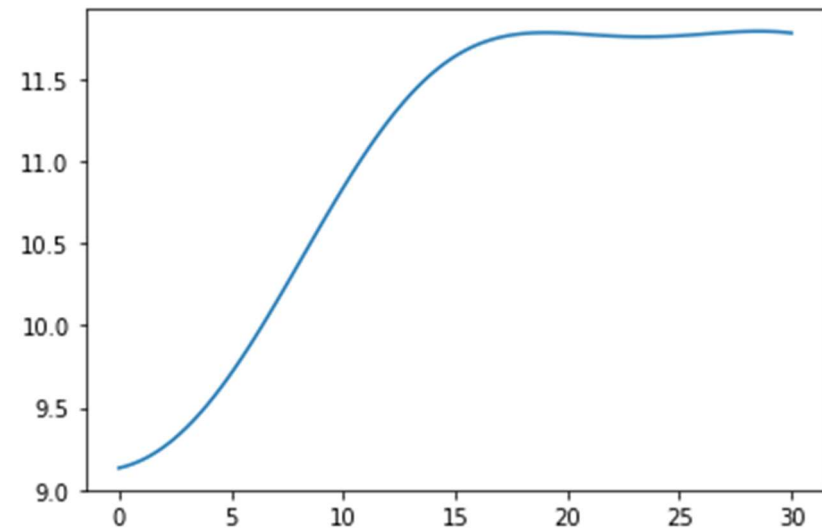
- 선형 기저 함수 모델
 - 기저 함수의 선형 조합을 통한 곡선 비선형 모델 생성 예시



4개의 가우스 기저 함수



- $y = w_0\phi_0(x) + w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x) + w_4$
- 이 때, $w_0 = -2, w_1 = 3, w_2 = -1, w_3 = 2, w_4 = 10$ 으로 설정



4개의 가우스 기저 함수를 조합하여 새로 만든 곡선 모델

Sensing Layer: 데이터 정제

■ 회귀를 이용한 센싱 값 분석/예측

• 선형 기저 함수 모델

- M개의 기저 함수로 선형 조합 한 새로운 함수를 y 라 할 때, $y = w_0\phi_0(x) + w_1\phi_1(x) + \dots + w_{M-1}\phi_{M-1}(x) + w_M$ 와 같음
- 또한, $\phi_M(x) = 1$ 이라고 하는 더미(dummy) 기저 함수를 사용하여 $y = \sum_{j=0}^M w_j\phi_j(x)$ 로 표현할 수 있음

• 오차 함수(평균 제곱 오차): $J(\underline{w}) = \frac{1}{N} \sum_{n=0}^{N-1} \left\{ \underline{w}^T \underline{\phi}(\underline{x}_n) - y_n \right\}^2$

- $\underline{w} = [w_0, w_1, \dots, w_M]$
- $\underline{\phi} = [\phi_0, \phi_1, \dots, \phi_M]$
- \underline{x}_n : n번째 데이터 샘플(스칼라 또는 벡터)
- y_n : n번째 데이터 샘플에 대한 레이블(정답)

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

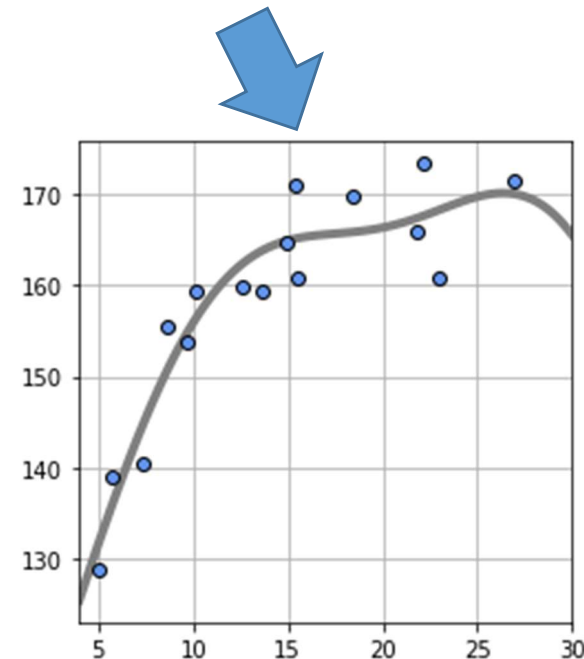
선형 기저 함수 모델

- 오차 함수(평균 제곱 오차): $J(\underline{w}) = \frac{1}{N} \sum_{n=0}^{N-1} \{ \underline{w}^T \underline{\phi}(x_n) - y_n \}^2$
- 오차 함수는 직전의 D차원 선형 회귀의 오차함수 $J(\underline{\theta}) = \frac{1}{N} \sum_{n=0}^{N-1} (\underline{\theta}^T \underline{x}_n - y_n)^2$ 와 동일함
- 따라서, 최적의 모델 파라미터를 찾는 방식도 동일함

D차원 선형 회귀 모델의 최적 파라미터	선형 기저 함수 모델의 최적 파라미터
$\underline{\theta} = (X^T X)^{-1} X^T \underline{y}$	$\underline{w} = (\Phi^T \Phi)^{-1} \Phi^T \underline{y}$

• 이때, $\Phi = \begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \dots & \phi_M(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_M(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_{N-1}) & \phi_1(x_{N-1}) & \dots & \phi_M(x_{N-1}) \end{bmatrix}$

Dummy를 포함한 M+1개의 가우스 기저 함수



Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 선형 기저 함수 모델 구현

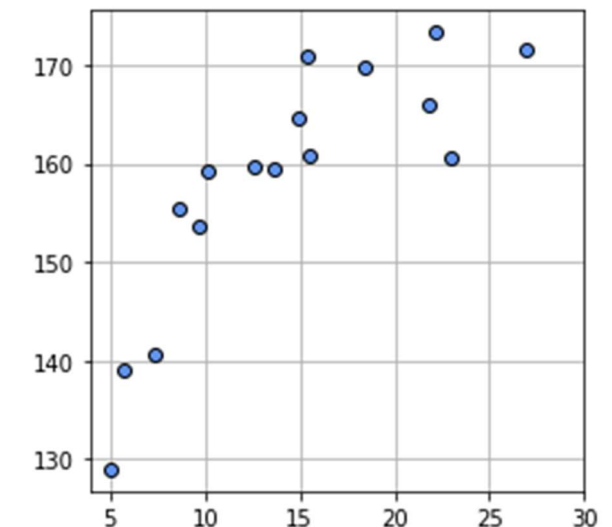
```
# 라이브러리 import
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# 임의 데이터 생성
np.random.seed(seed = 1)
X_min = 4 # X의 하한(표시 용)
X_max = 30 # X의 상한(표시 용)
X_n = 16 # X의 상한(표시용)
X = 5 + 25 * np.random.rand(X_n)

Prm_c = [170, 108, 0.2] # 생성 매개 변수

$$Y = Prm_c[0] - Prm_c[1] * \exp(-Prm_c[2] * X) + 4 * \text{np.random.randn}(X_n)$$


# 생성한 데이터를 plot하기
plt.figure(figsize = (4, 4))
plt.plot(X, Y, marker = 'o', linestyle = 'None', markeredgecolor = 'black',
         color = 'cornflowerblue')
plt.xlim(X_min, X_max)
plt.grid(True)
plt.show()
```



Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 선형 기저 함수 모델 구현

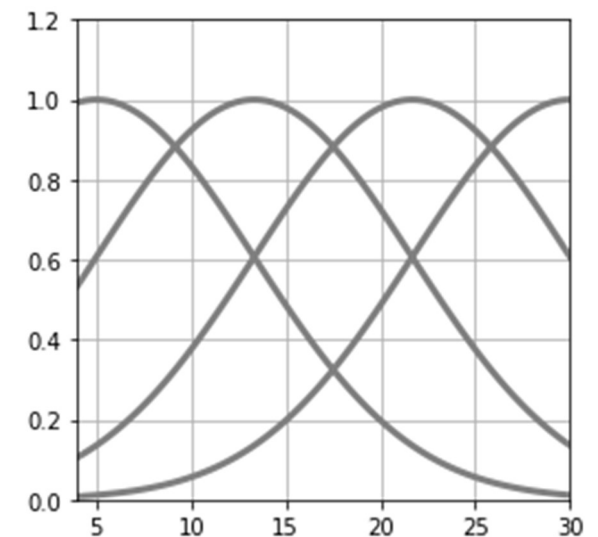
```
# 가우스 함수 정의하기
def gauss(x, mu, s) :
    return np.exp(-(x - mu) ** 2 / (2 * s ** 2))

M = 4 # M = 4인 경우를 가정

# 4개의 gauss 함수를 plot 하기
plt.figure(figsize = (4, 4))
mu = np.linspace(5, 30, M) # 데이터 샘플의 분포를 고려하여 4개의 평균값 생성
s = mu[1] - mu[0] # (A) # 표준편차 정의(인접한 가우스 함수 중심간 거리)
xb = np.linspace(X_min, X_max, 100)

for j in range(M) :
    y = gauss(xb, mu[j], s)
    plt.plot(xb, y, color = 'gray', linewidth = 3)

plt.grid(True)
plt.xlim(X_min, X_max)
plt.ylim(0, 1.2)
plt.show()
```



Sensing Layer: 데이터 정제

- 회귀를 이용한 센싱 값 분석/예측
 - 선형 기저 함수 모델 구현

```
# 선형 기저 함수 모델의 정확한 솔루션 계산하기
# = 최적 파라미터 w 계산하기

def fit_gauss_func(x, y, m) :
    # m개의 가우스 모델 만들기
    mu = np.linspace(5, 30, m) # m개의 평균값 생성
    s = mu[1] - mu[0] # 표준편차 정의
    n = x.shape[0] # 학습용 샘플의 수
    psi = np.ones((n, m + 1)) # Dummy 포함한 M+1개 가우스 기저함수

    # psi 매트릭스 생성
    for j in range(m) :
        psi[:, j] = gauss(x, mu[j], s)
    psi_T = np.transpose(psi)

    b = np.linalg.inv(psi_T.dot(psi)) # (psi^T * psi)^-1
    c = b.dot(psi_T) # (psi^T * psi)^-1 * psi^T
    w = c.dot(y) # (psi^T * psi)^-1 * psi^T * y
    return w
```


Sensing Layer: 데이터 정제

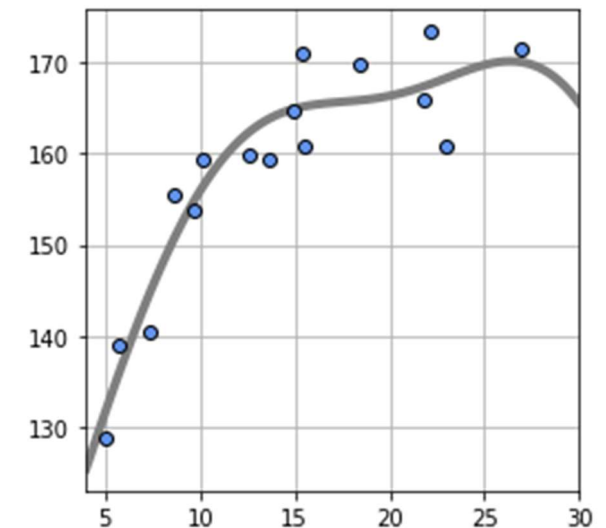
- 회귀를 이용한 센싱 값 분석/예측
 - 선형 기저 함수 모델 구현 : 최종 결과

```
M = 4 # M = 4 라고 가정
W = fit_gauss_func(X, Y, M) # 최적 파라미터 W 계산

plt.figure(figsize = (4, 4))
show_gauss_func(W)
plt.plot(X, Y, marker = 'o', linestyle = 'None',
         color = 'cornflowerblue', markeredgecolor = 'black')
plt.xlim(X_min, X_max)
plt.grid(True)

print('W = ' + str(np.round(W, 1)))
plt.show()

W = [29.4 75.7  2.9 98.3 54.9]
```

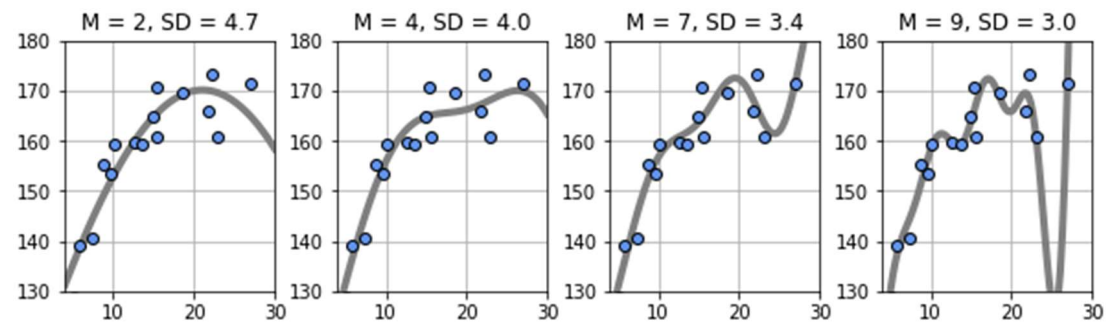
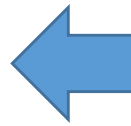
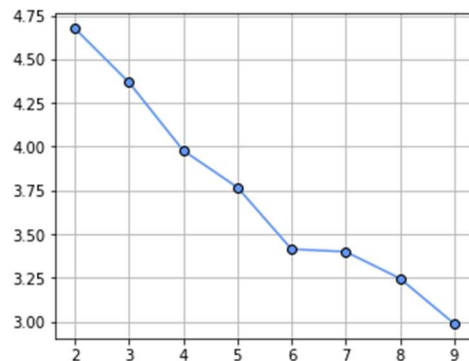


Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

선형 기저 함수 모델

- $\phi = [\phi_0, \phi_1, \dots, \phi_M]$ 에서 M 은 기저함수의 수를 나타내는데, M 은 어떻게 설정?
- M 값의 변화에 따른 오차($SD = \sqrt{MSE}$)



- M 이 증가할 수록 오차가 줄어들기는 하나, overfitting 문제가 발생함
 - Overfitting 일 발생하면 일반화(generalization) 성능이 감소하여, 전체적인 추세를 나타내는 데에는 무리가 있음
 - 즉, 새로운 데이터에 대한 예측 성능이 떨어짐
-
- 최적의 M 을 찾기 위한 방법은?

Sensing Layer: 데이터 정제

회귀를 이용한 센싱 값 분석/예측

선형 기저 함수 모델

최적의 M 찾기

- 데이터 셋을 Training set 과 Test set으로 분류하고, 다양한 M값에 대한 모델을 Test set을 대상으로 성능을 테스트 하여 최적의 M값을 찾기
- 데이터 셋을 Training set과 Test set으로 분류하기 위해 7:3, 8:2 등으로 나누는 것이 일반적이나, 전체 데이터 셋의 크기가 작은 경우에는 K-Fold Cross Validation 기법을 사용
- 예) K=5인 경우, 각 iteration 별로 MeanSquareError를 측정하고, 5번 iteration의 평균을 구한 후, 최소 MSE 성능을 보이는 M 값을 선택

