머신러닝/딥러닝
구현 심화

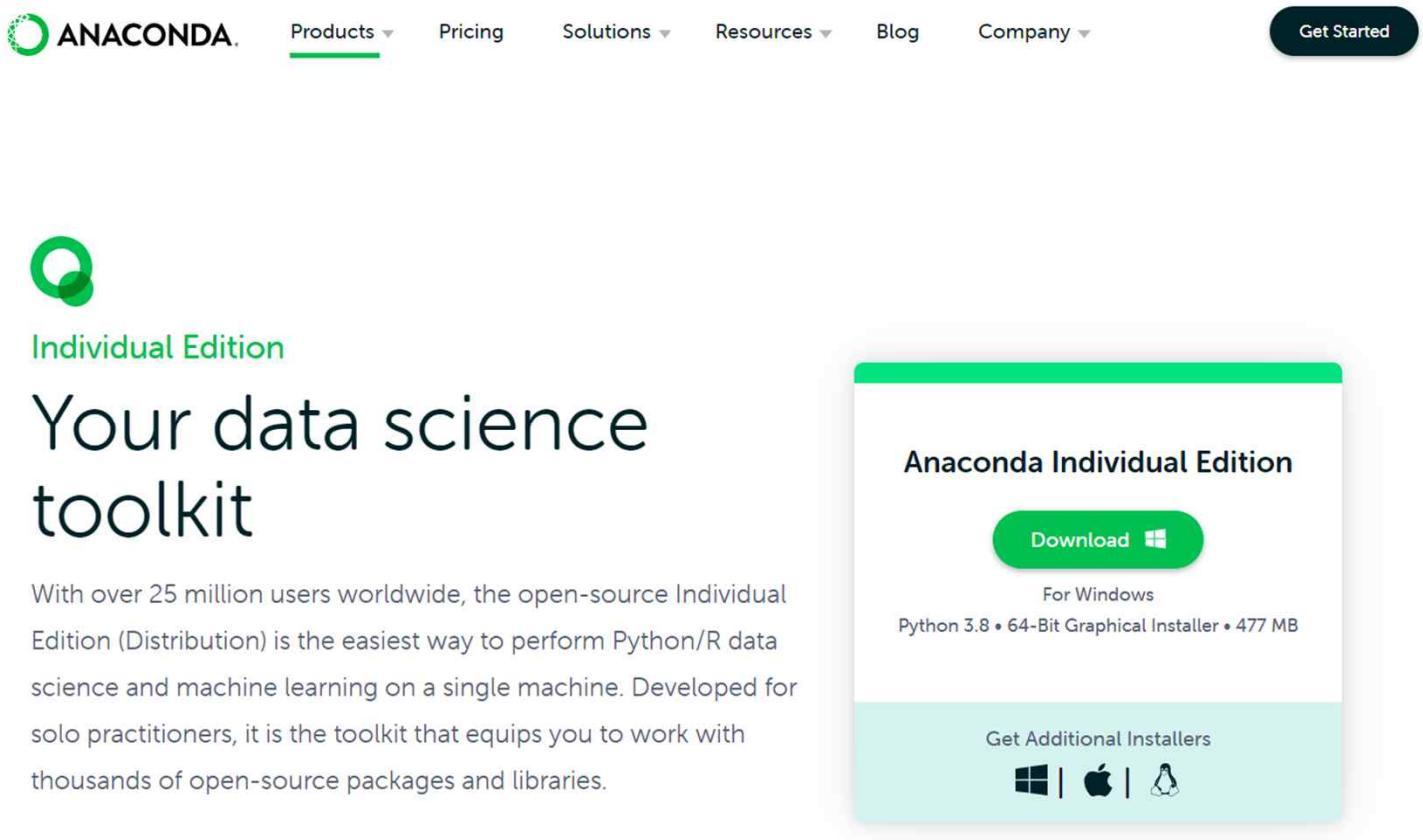
강사: 오영제

교육 환경 준비

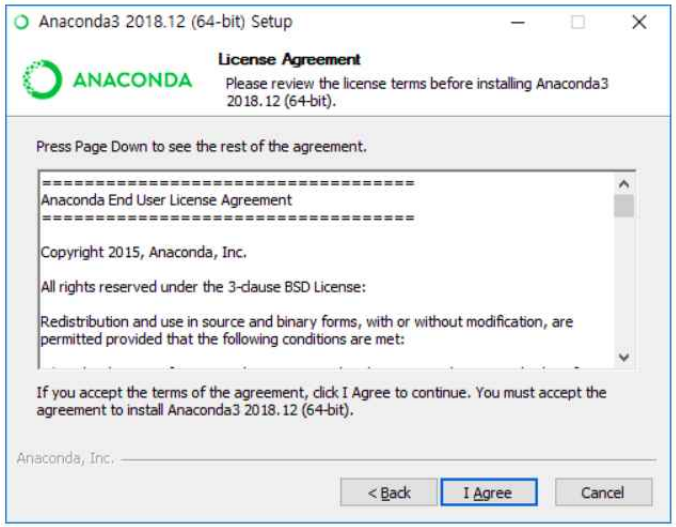
- 인터넷 연결
- Chrome Browser
- Anaconda 설치
- Gmail ID – Google Colab 이용

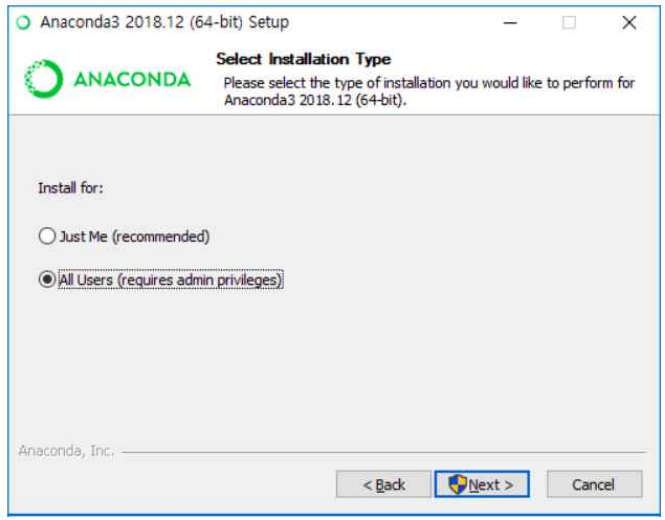
Anaconda 설치

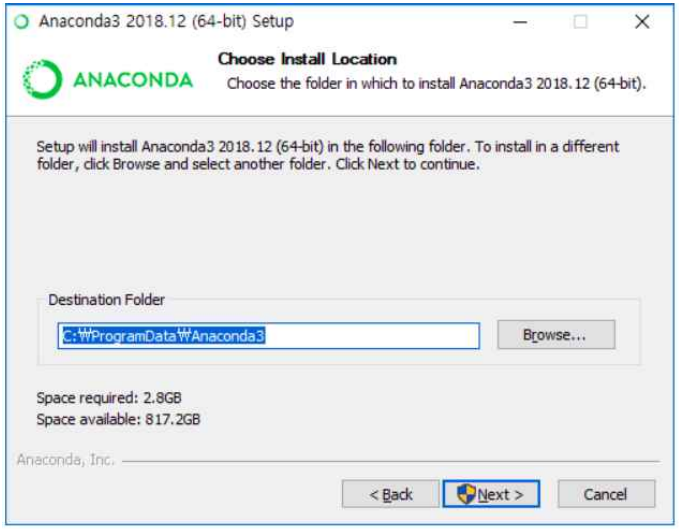
Download from https://www.anaconda.com/products/individual#download-section

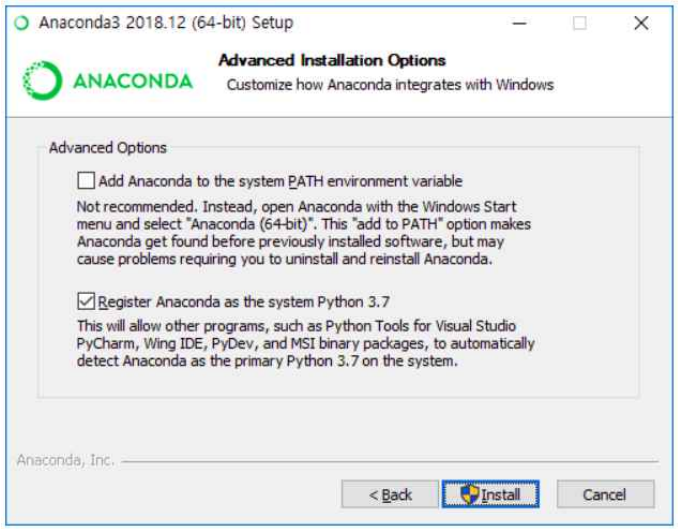


Next 를 눌러 설치









완료

Tensorflow 2.0

Tensorflow Installation

- pip install --upgrade tensorflow

- import tensorflow as tf
- tf.__version__

Sequential API

- Sequential API는 Keras의 모델 생성 방법 중 하나로, 레이어를 순서대로 쌓아서 모델을 구축하는 방법.
- 각 레이어에는 정확히 하나의 입력 텐서와 하나의 출력 텐서로 구성.
- 레이어를 쌓아가는 형태이기 때문에 직관적으로 이해하기 쉽다.
- 복잡한 모델 구조를 설계하기 어렵다. 즉, 복수의 입력 또는 출력, 공유 레이어, 비선형 토폴로지 등을 다루기 어렵다.

Sequential API

하나의 입력 텐서

```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPool2D((2,2)))

model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same',))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same',))
model.add(layers.MaxPool2D((2,2)))

model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same',))
model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same',))
model.add(layers.MaxPool2D((2,2)))

model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

하나의 출력 텐서

Functional API (함수형 API)

- Sequential API 보다 더 유연한 모델을 생성하는 방법

- 함수형 API는 Sequential API가 처리할 수 없는 비선형 토폴로지, 공유 레이어, 여러 입력 또는 출력이 있는 모델을 처리 가능

- 훈련, 평가 및 추론은 `Sequential` 모델과 같은 방식으로 작동

Functional API

```
inp = Input(shape=(64, 64, 1)) # (64, 64, 1)의 형태를 가진 데이터를 입력
x = Conv2D(32, kernel_size=4, activation='relu')(inp)  # 32개의 4x4 필터를 가지는 Conv층
x = MaxPooling2D(pool_size=(3, 3))(x)  # 3x3 크기의 윈도우를 사용하는 max pooling layer
x = Flatten()(x)
x = Dense(10, activation='relu')(x)  #FC 층
output = Dense(1, activation='sigmoid')(x)   #출력층

# Model 객체를 생성합니다. 입력 레이어와 출력 레이어를 연결.
model = Model(inputs=inp, outputs=output)
model.summary()
```

전문가용 API (Subclassing API)

- Keras의 subclass로 모델을 생성하는 방법
- tf.GradientTape를 사용하여 모델을 훈련
- tape.gradient(loss, x) 로 후진 모드 자동 미분

전문가용 – Subclassing API

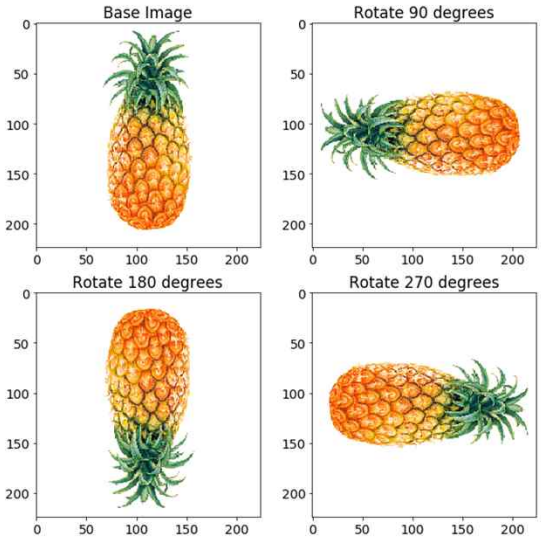
```python
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

# model instance 생성
model = MyModel()
```

```python
with tf.GradientTape() as tape:
    predictions = model(images, training=True)

    loss = loss_object(labels, predictions)

# 그래디언트를 계산합니다.
gradients = tape.gradient(loss, model.trainable_variables)

# 계산된 그래디언트를 사용해 옵티마이저를 통해 학습 가능한 변수를 업데이트
optimizer.apply_gradients(zip(gradients, model.trainable_variables))
```

실습 : 010. Sequential API + data 증강

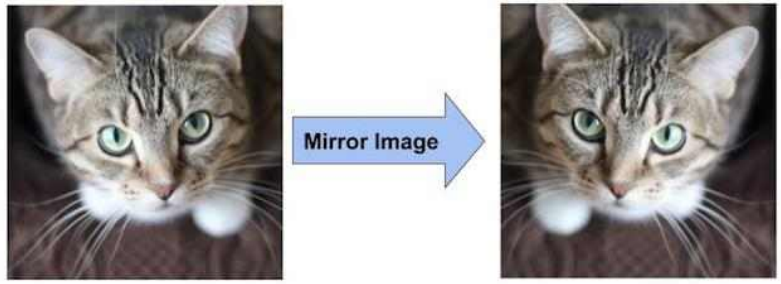
1. Sequential API를 이용하여 CNN 모델 정의
2. CIFAR-10 dataset 은 32x32 color image 를 가진 10개의 class (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)
3. 각 class 별 6,000 개씩 total 60,000 개 image
4. Data Augmentation으로 CIFAR-10 분류 CNN 모델 성능 향상

Data Augmentation

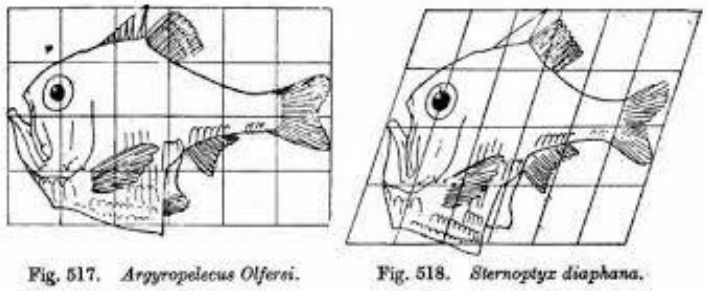


Rotation

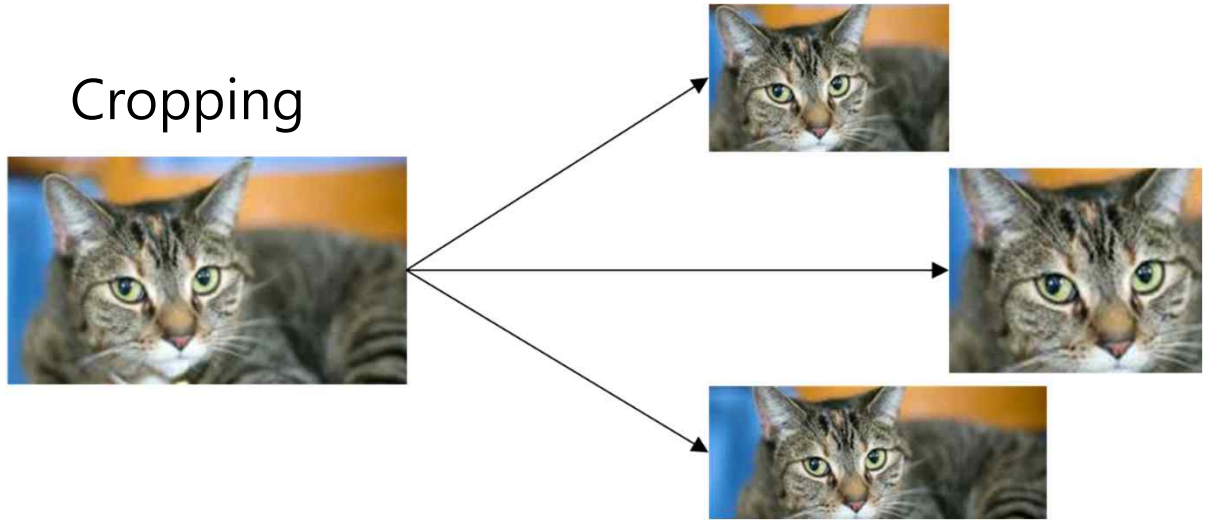
Mirroring



Shearing



Cropping



Data Augmentation

Color shifting

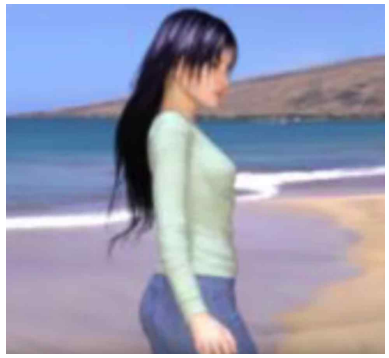
R, G, B

+50,-50,+50

-100,+55,+55

+5,0,+70

zoom



Horizontal Flip

ImageDataGenerator parameters

- train_dataget = ImageDataGenerator(
  rescale=1./255,
  rotation_range=40, ➔ 0~40 도 사이에서 random 회전
  width_shift_range=0.2, ➔ 사진 중심 20% 이동
  height_shift_range=0.2,
  shear_range=0.2 ➔ 20% shear
  zoom_range=0.2 ➔ 20% 까지 randomly zoom
  horizontal_flip=True,
  fill_mode='nearest') ➔ 생성된 공간은 인접 pixel 로 채움

ImageDataGenerator methods

```
datagen = ImageDataGenerator(
    horizontal_flip=True,       # 이미지를 무작위로 수평으로 뒤집습니다
    width_shift_range=0.1,    # 이미지를 수평으로 최대 10% 이동
    height_shift_range=0.1,   #이미지를 수직으로 최대 10% 이동
)

# 데이터 증강 인스턴스를 학습 데이터에 맞춥니다.
datagen.fit(X_train)

# 증강된 이미지와 레이블을 배치 단위로 생성
it = datagen.flow(X_train, y_train, shuffle=False)

batch_images, batch_labels = next(it)
```

실습 : 020. Functional API

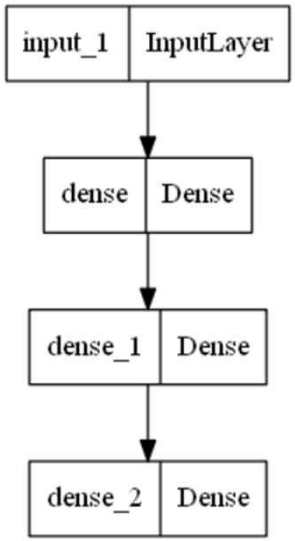
A. Standard Neural Network Models
   1. Simple Multi-layer Perceptron
   2. Convolutional Neural Network
   3. Recurrent Neural Network

B. Shared Layers Models
   1. Shared Input Layer
   2. Shared Feature Extraction Layer

C. Multiple Input and Output Models
   1. Multiple Input Model
   2. Multiple Output Model

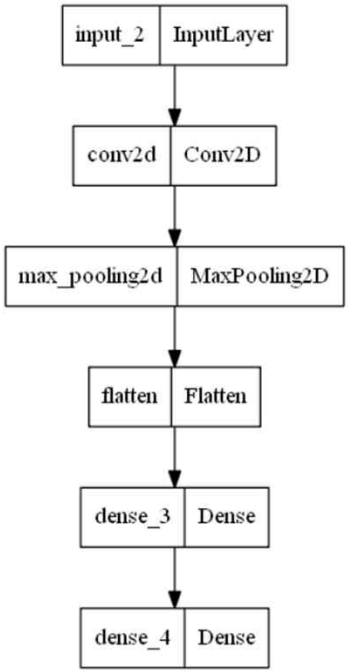
Simple Multi-layer Perceptron

A-1



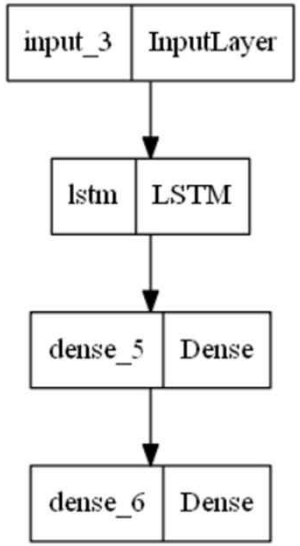
Recurrent Neural Network

A-2



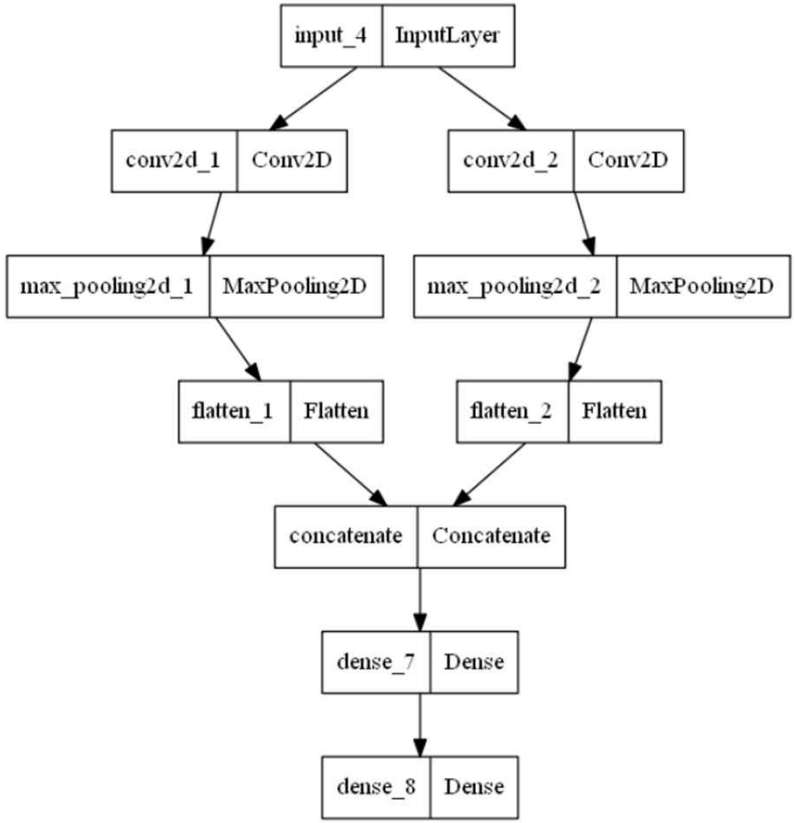
Convolutional Neural Network

A-3



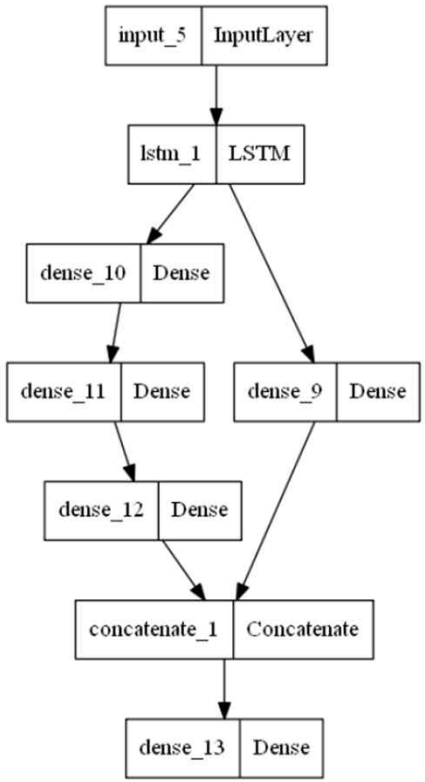
Shared Input Layer

B-1

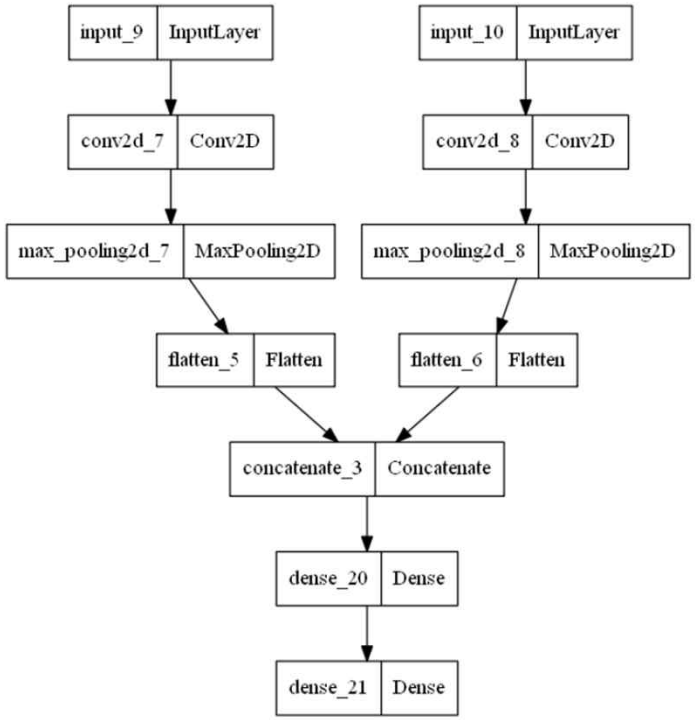


Shared Feature Extraction Layer

B-2



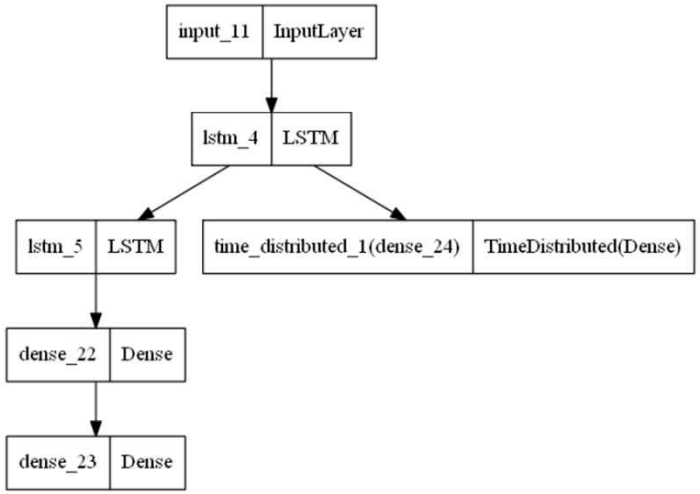
C-1



Multiple Input Model

Multiple Output Model

C-2



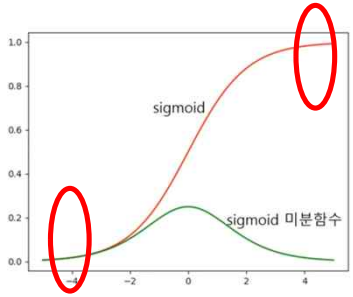
실습 : 030. Subclassing API

```python
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

# model instance 생성
model = MyModel()
```

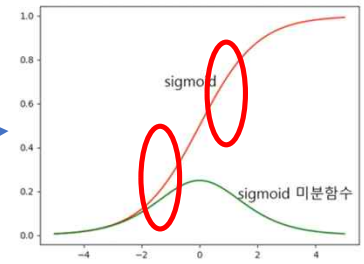
```python
@tf.function
def train_step(images, labels):

    with tf.GradientTape() as tape:
        # training=True는 학습과 추론시 다른 동작을 하는
        # 레이어(예: Dropout)가 있는 경우에만 필요합니다.
        predictions = model(images, training=True)
        loss = loss_object(labels, predictions)

    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    train_loss(loss)
    train_accuracy(labels, predictions)
```

Vanishing Gradient

기울기 소실 (Vanishing Gradient)

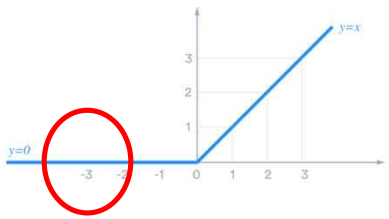


Backpropagation 진행 중 error term 을
점점 잃어버림

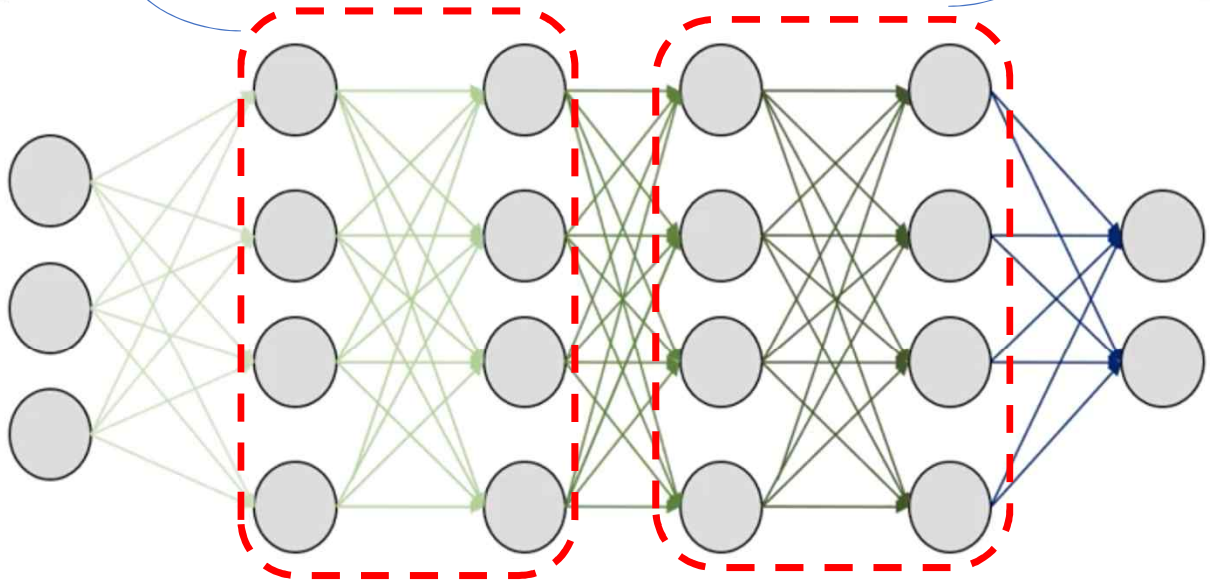


σ 의 미분 ≤ 0.25

$$\frac{d\sigma}{dz} = \sigma(1-\sigma)$$



dying ReLU

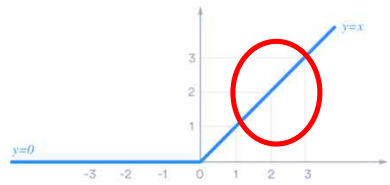


학습이 안됨

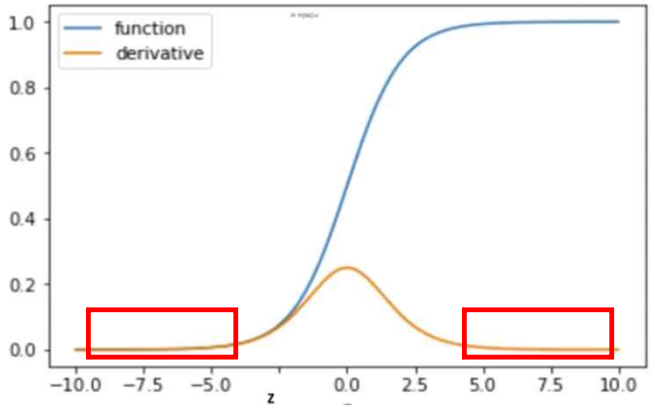
Weight 의 변화가 작음

학습이 잘됨

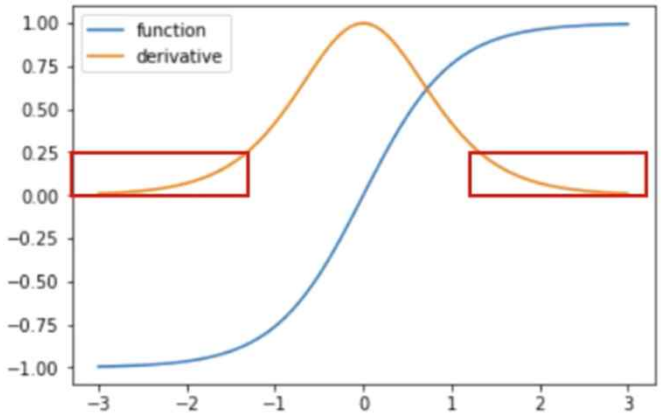
Weight 의 변화가 큼



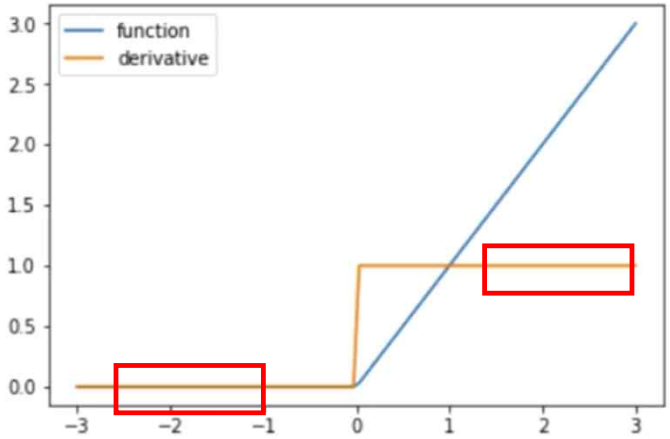
Vanishing Gradient – activation functions



sigmoid

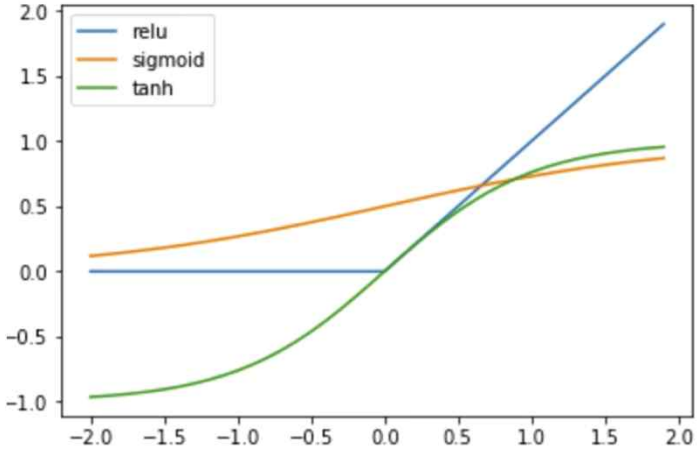


tanh



relu

Gradient 비교

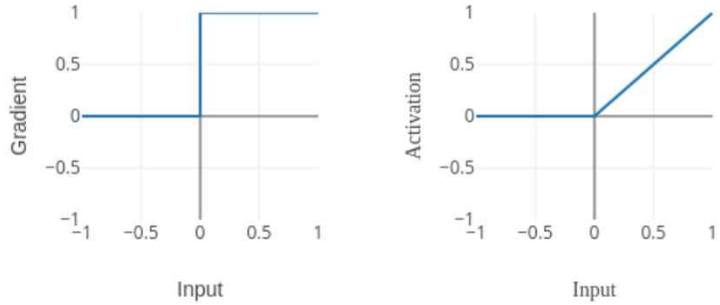


Sigmoid 와 tanh 는 vanishing gradient 문제 발생

Solutions of Vanishing Gradient

- Relu 사용

$$\max(0, z) \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$$



- Weight 의 신중한 초기화

  Ex) Xavier (Glorot) Initializer with Tanh, He Initializer with Relu

- Batch Normalization

- Residual Network

Batch Normalization

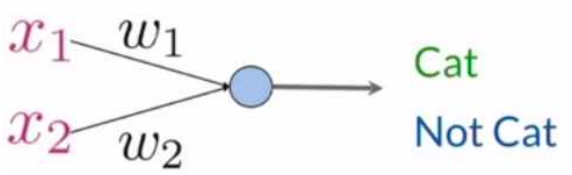
- Internal Covariance Shift (내부 공변량 변화)
  - Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상
  - 최초 Input Layer 의 normalization 효과가 Hidden Layer 를 거치면서 희석됨

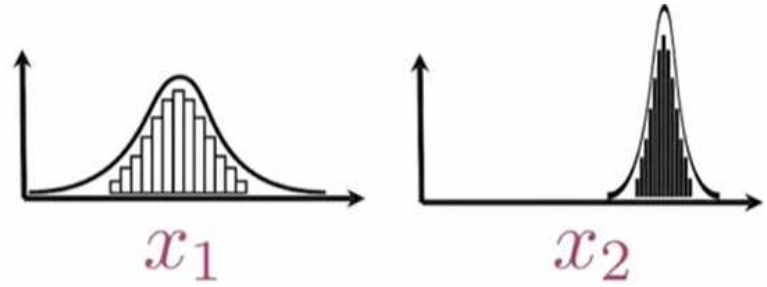
- 각 층의 input의 distribution을 평균 0, 표준편차 1인 input으로 normalize
- Faster Train

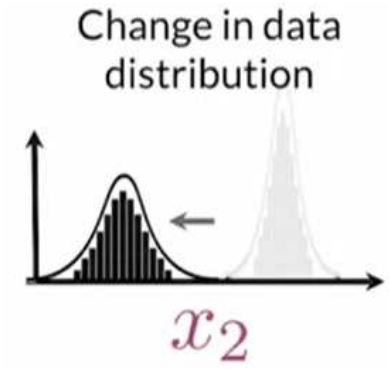
- 공분산(covariance)은 통계학에서 두 변수 간의 선형 관계의 강도를 측정하는 방법입니다. 공분산은 두 변수가 함께 변하는 경향을 나타냅니다. 구체적으로, 공분산은 두 변수의 각 값에서 그 변수의 평균을 뺀 후, 이를 모두 곱한 값들의 평균입니다.

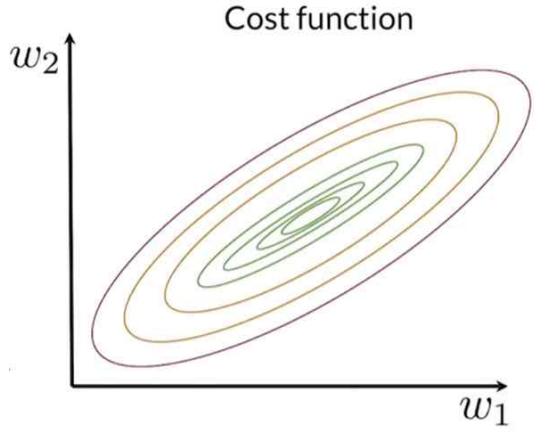
$$Cov\,(X,Y) = \frac{\sum_{i-1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n}$$

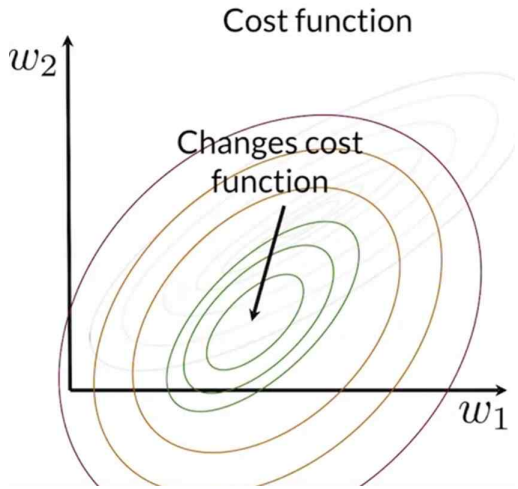
Internal Covariance 와 Batch Normalization





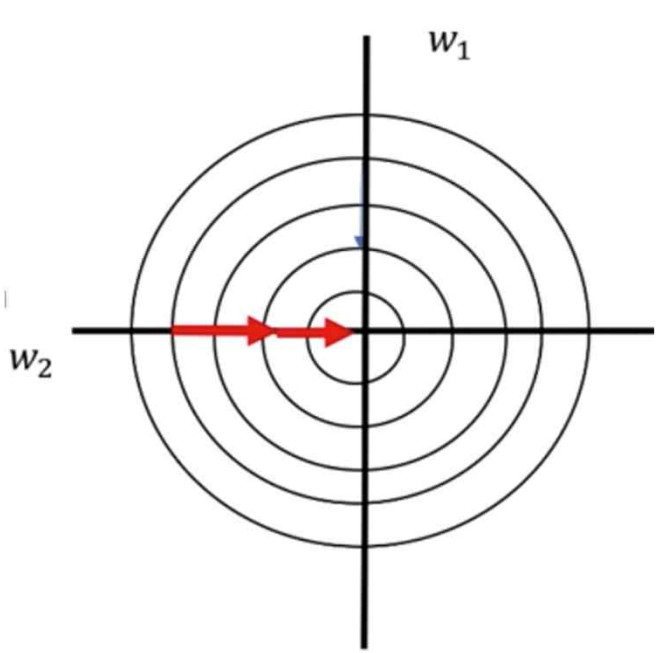






With Batch Normalization

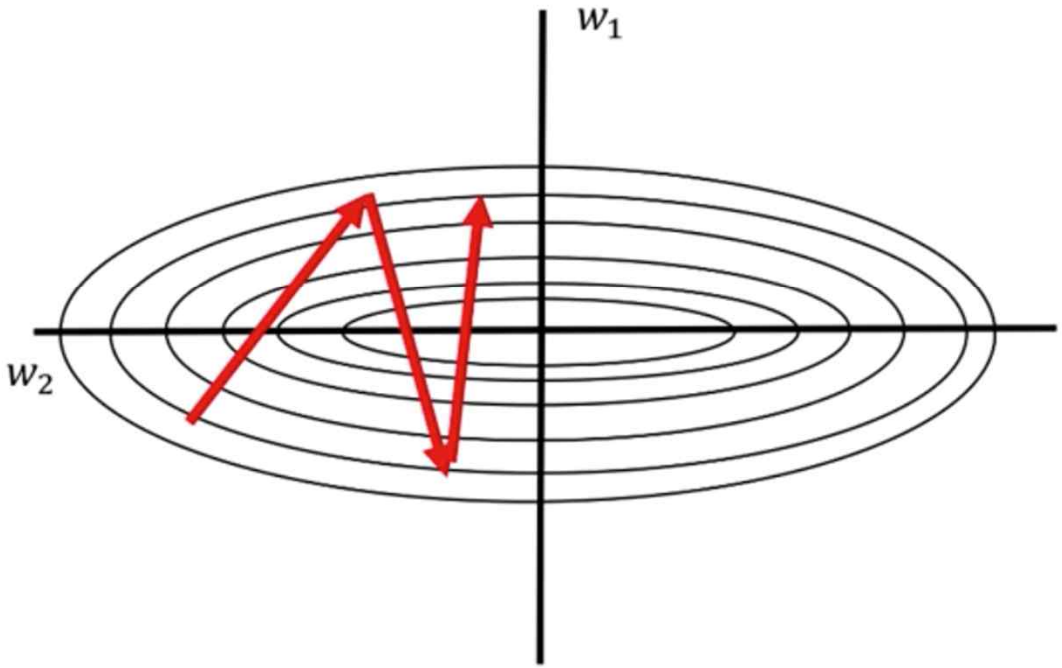
$$w_1\tilde{z}_1^1 + w_2\tilde{z}_2^1$$



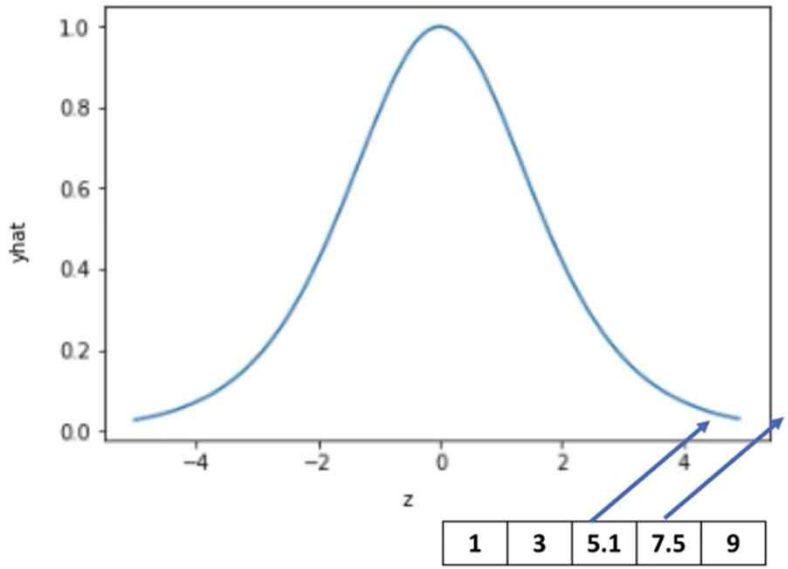
수렴 속도가 빠르다

Without Batch Normalization

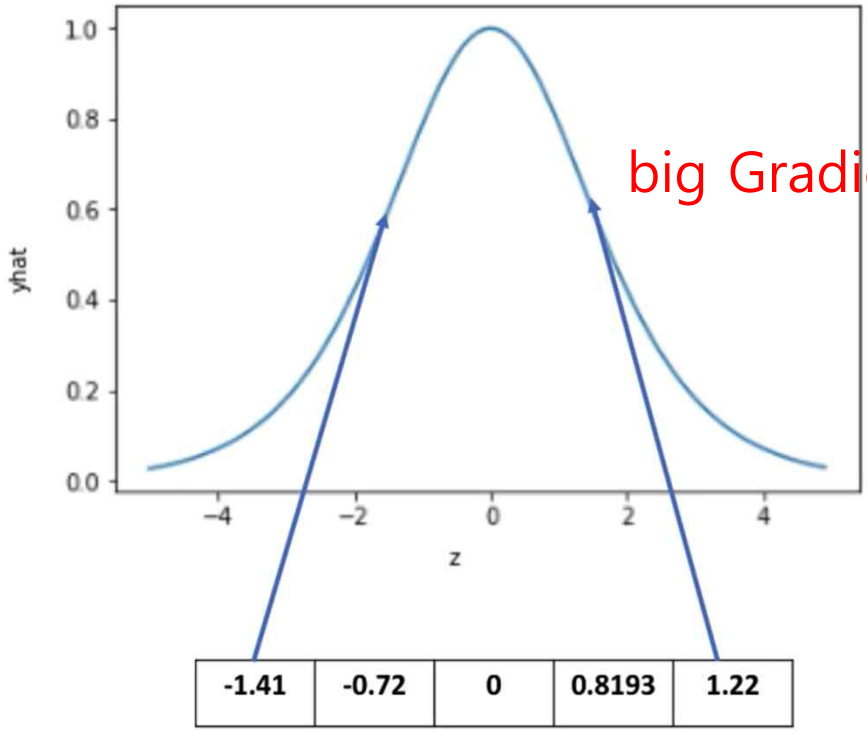
$$w_1 z_1^1 + w_2 z_2^1$$



Batch Normalize 전, 후 Gradient 비교

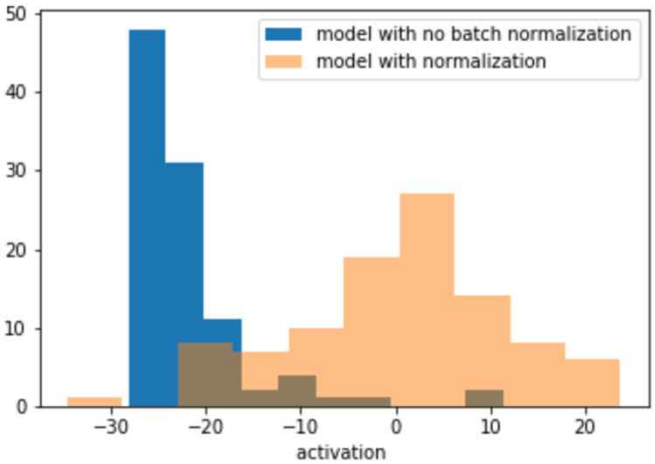


Gradient 소실



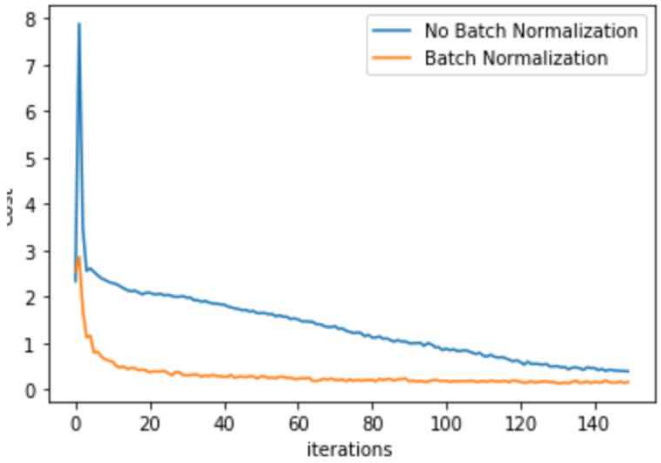
big Gradient

Batch Normalization 의 효과

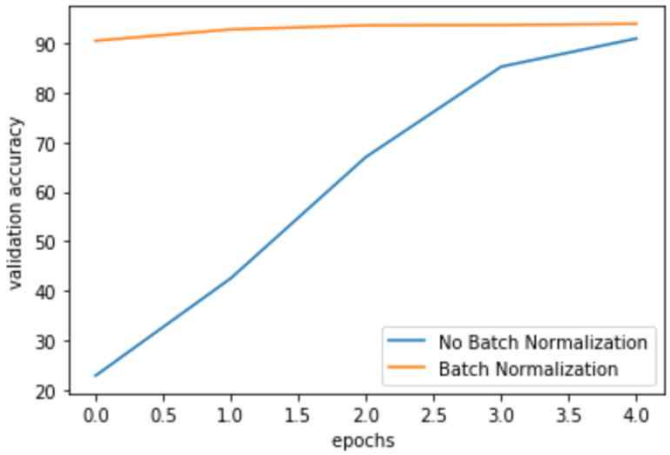


Activation 출력 비교

Loss 비교



Accuracy 비교

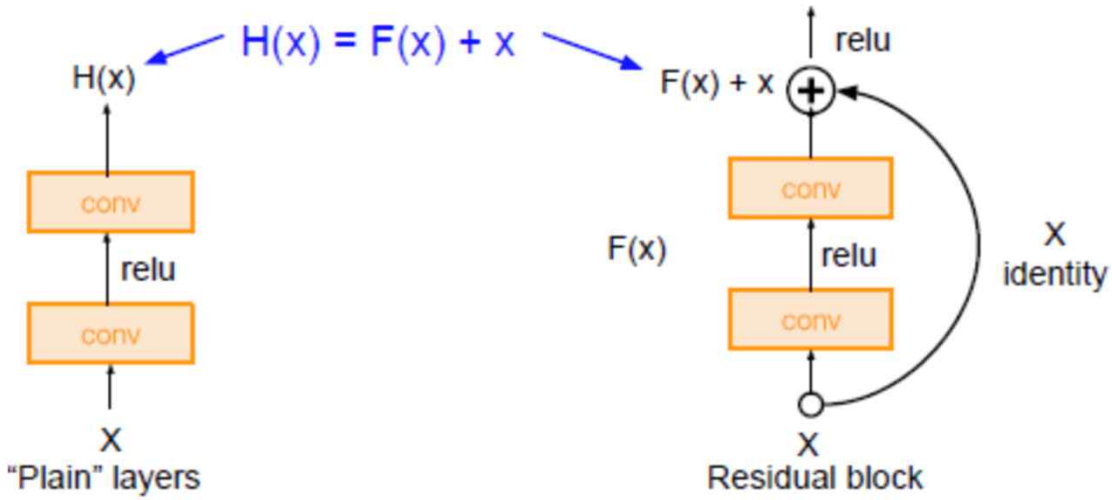


- Dropout 불필요
- 큰 Learning Rate 적용 가능
- Bias 불필요

ResNet – Identity Shortcut 적용

- 2015 년 ILSVRC 우승 모델
- 152 개 Layer 로 구성

- 이전에 학습한 정보를 보존하고, 추가적으로 학습

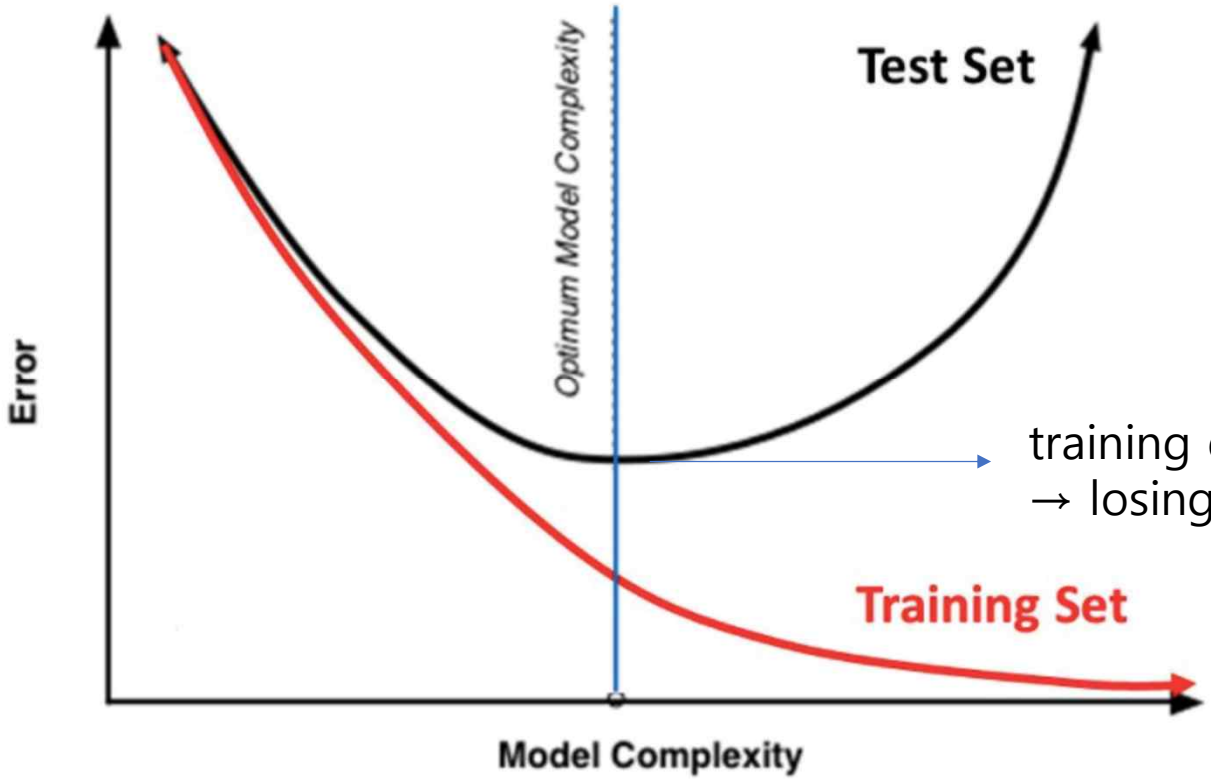


https://miro.medium.com/max/1400/1*6hF97Upuqg_LdsqWY6n_wg.png

Overfitting 방지 기법

Regularization

What is Overfitting ?



training dataset 을 기억하기 시작
→ losing generalization

과적합 방지 방법

- More Data – Best solution but 항상 가능하지 않음
- Network 의 size 축소
- Weight Regularization (주로 L2 사용) → 모델의 복잡성을 제한하는 방법

$$J(w,b) = \frac{1}{m}\Sigma L(prediction - true) + \frac{\lambda}{2m}\Sigma ||w||^2$$

손실함수

W의 norm

- Early Stopping
- Dropout : Neural Network 에서 가장 일반적으로 많이 사용

Overfitting 방지 기법 (Regularization)

- 모델의 가중치(parameters)의 크기에 따라 손실을 증가시켜, 모델이 너무 복잡해지는 것을 방지

- Linear Regression (선형회귀)

손실함수 Train loss

$$J(W) = MSE_{train}(W) + \lambda\Omega(W)$$

λ : regularization 강도 (hyperparameter)
Ω : regularizer

- 대표적 regularizer
  1. L1 regularization : $\Omega(W) = ||W||_1$ (lasso regularization)
  2. L2 regularization : $\Omega(W) = ||W||_2$ (ridge regularization)
  3. Elastic net regularization : L1 + L2

Overfitting 방지 기법 (Regularization)

- Logitstic Regression (이진 분류)

λ : regularization 강도

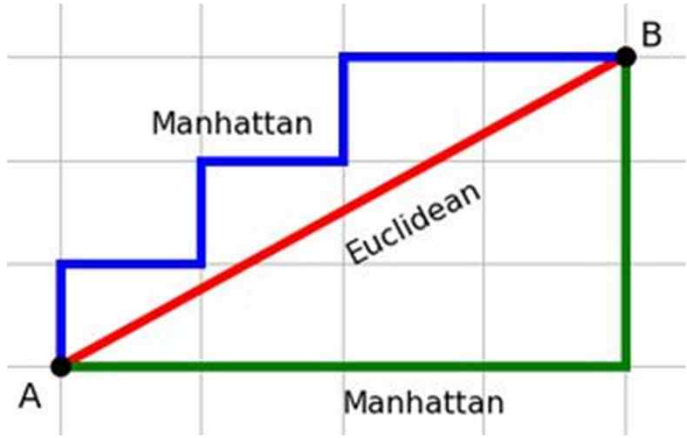
$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) + \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \lambda\frac{1}{2m}\sum_{i=0}^{m}\theta_j^2$$

Binary cross-entropy 함수

regularizer

요점 – weight(θ) 를 의도적으로 더해주면 optimization 은 더해준 만큼의 weight(θ) 영향을 줄여주는 방향으로 진행

What is L2, L1 Norm ?



- Norm - 벡터의 길이(크기)
- 원점에서 벡터 좌표까지의 거리

$$L_p = (\sum_{i}^{n} |x_i|^p)^{\frac{1}{2}}$$

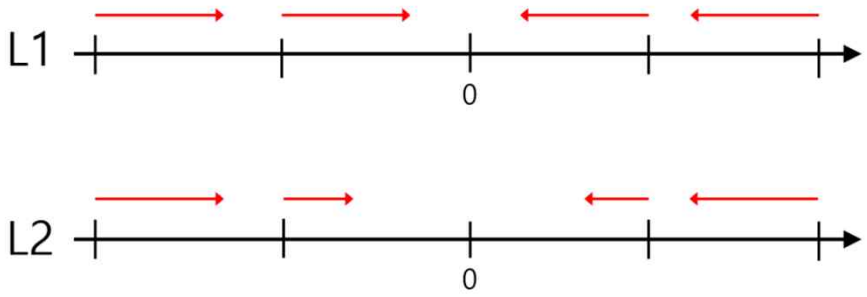
p : norm 의 차수
n : vector 의 element 수 (dimension)

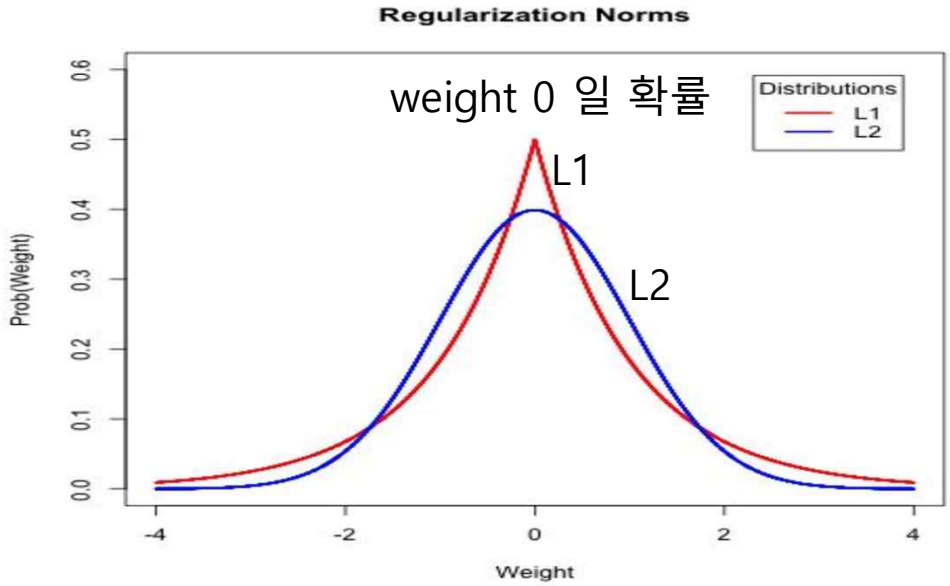
- L1 = $||W||_1 = (|W_0| + |W_1| + \cdots + |W_n|)$ (Manhattan Norm, Taxicab norm)
- L2 = $||W||_2 = \sqrt{{W_0}^2 + {W_1}^2 + \cdots + {W_n}^2}$ (Euclidean Norm, 두 점 간의 거리)

L1 vs L2 Regularization 비교

- L1과 L2 모두 regularization 효과로 theta를 0 이 되는 방향으로 이동시킴.
  - L1 - theta의 크기에 상관없이 일정한 힘으로 0 방향으로 theta 를 이동
  - L2 - theta의 크기가 작아질수록 0 으로 theta를 이동시키는 힘이 약해짐.

- 이러한 차이로 L1은 많은 feature들의 가중치를 0 으로 만들고 L2는 0 으로는 만들지 않고 0 에 가까운 값으로 만든다.





L1 vs L2 Regularization 비교

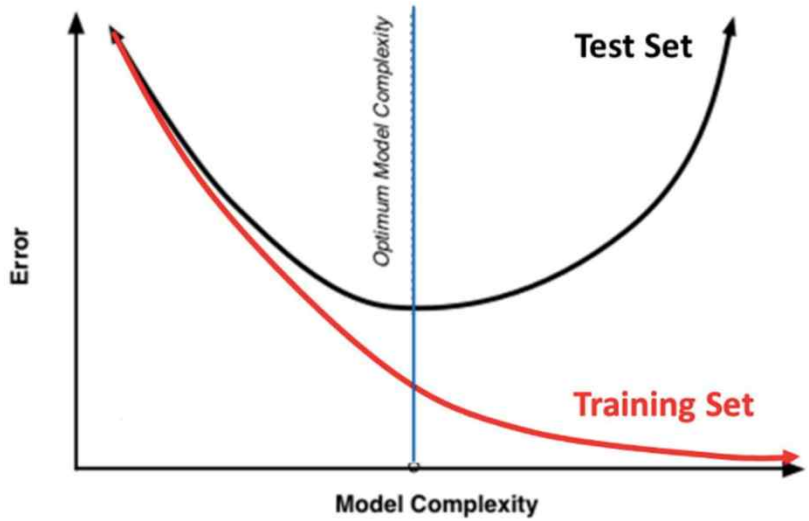
- L1 regularization 은 feature selection mechanism 으로 사용 가능
  → 모델이 단순해지는 효과

- L2 regularization 은 stable 하고 convex 하므로 수학적 특성 좋으나, 불필요한 weight 를 남겨두므로 model 이 크고 복잡해 짐.
  단, feature 에 작은 weight 라도 남기므로 일반화에는 유리

Early Stopping

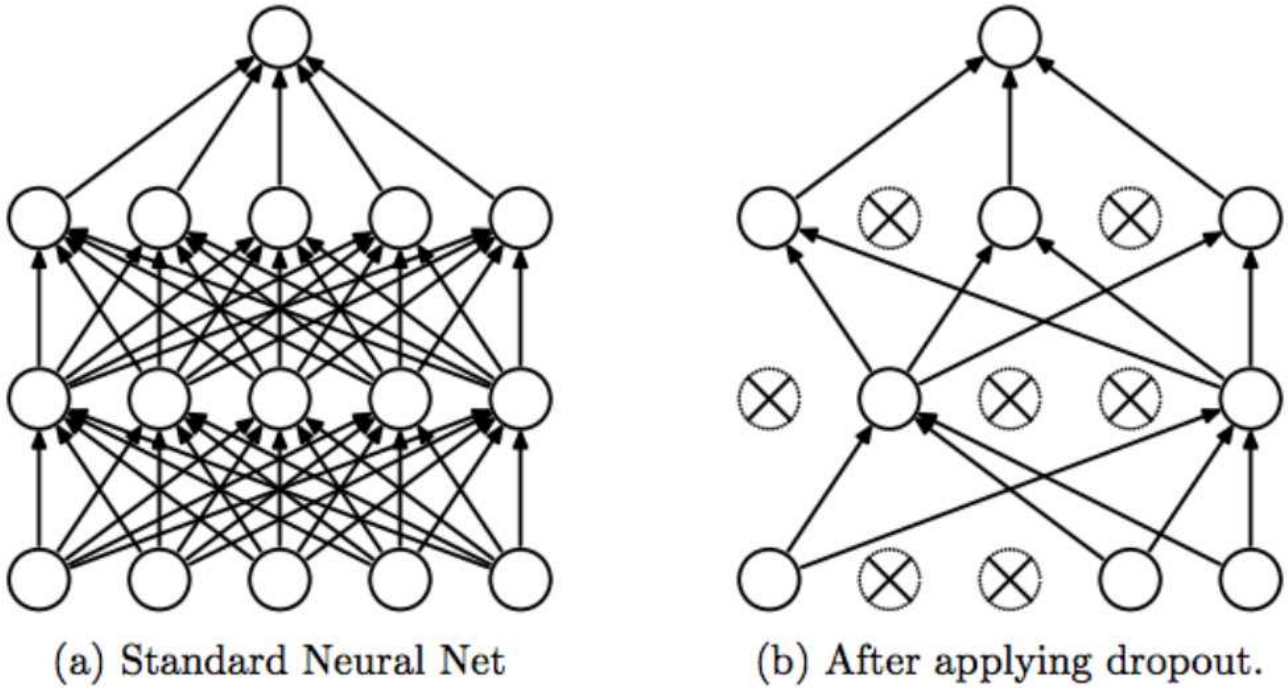
- L2 regularization 과 비슷한 효과이면서 computationally cheaper
- 두가지를 함께 사용하면 optimal hyper-parameter를 정하기 어려운 경우 도움이 된다.

Early Stopping



Dropout regularization

Random 한 drop out 을 통한 과적합 방지 (특정 feature 의존 방지)



(a) Standard Neural Net

(b) After applying dropout.

실습 : 040. Overfitting and Regularization

- IMDB 이용
- Weight Regularization 과 Dropout 이용
- Small, medium 및 Big model 에 대하여 서로 비교

비지도학습 모델 구현

Clustering

Clustering 이란 ?

- 비슷한 object 들끼리 모으는 것

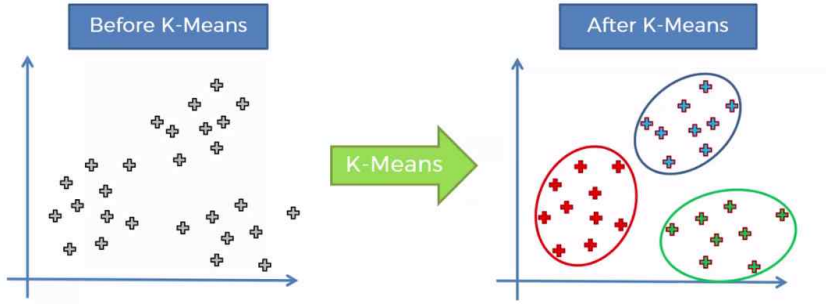
- label data 가 없는 것이 classification 과 차이

  ➔ unsupervised machine learning

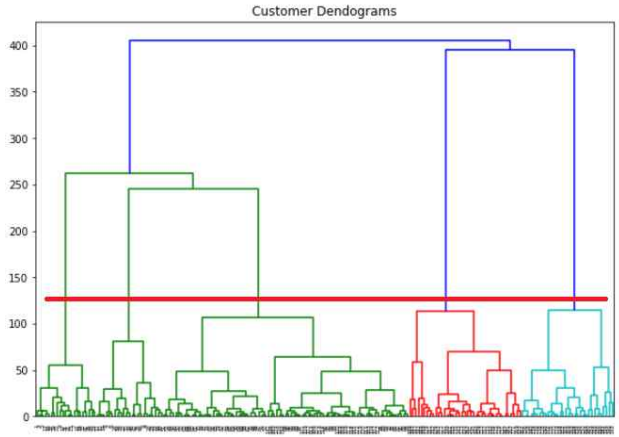
- 적용 사례
  - 고객의 구매 형태별 분류
  - 신용카드 사용의 fraud detection
  - 뉴스 자동 분류 및 추천
  - 유전자 분석
  - 이미지 양자화 등

Clustering 알고리즘의 종류

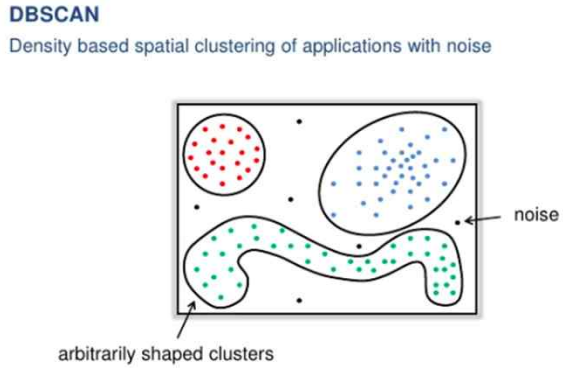
- K-Means Clustering



- Hierarchical Clustering (dendrogram)



- Density-based Clustering (DBSCAN)



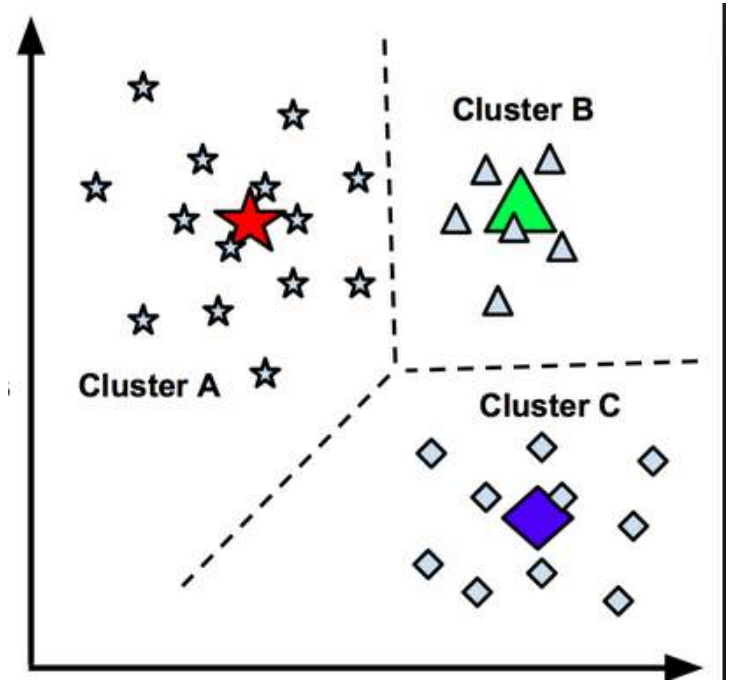
K-Means Clustering 알고리즘 – Distance 계산

Distance = Euclidean Distance

$$= \sqrt{\sum_{i=0}^{n}(x_{1i} - x_{2i})^2}$$

Ex)

고객	나이	수입	교육	
1	54	190	3	→ x1
2	50	200	8	→ x2



Distance(x1, x2) $= \sqrt{(54-50)^2 + (190-200)^2 + (3-8)^2} = 11.87$

K-Means Clustering 알고리즘

1. Random 하게 k 개의 centroid (중심점) 를 정한다.

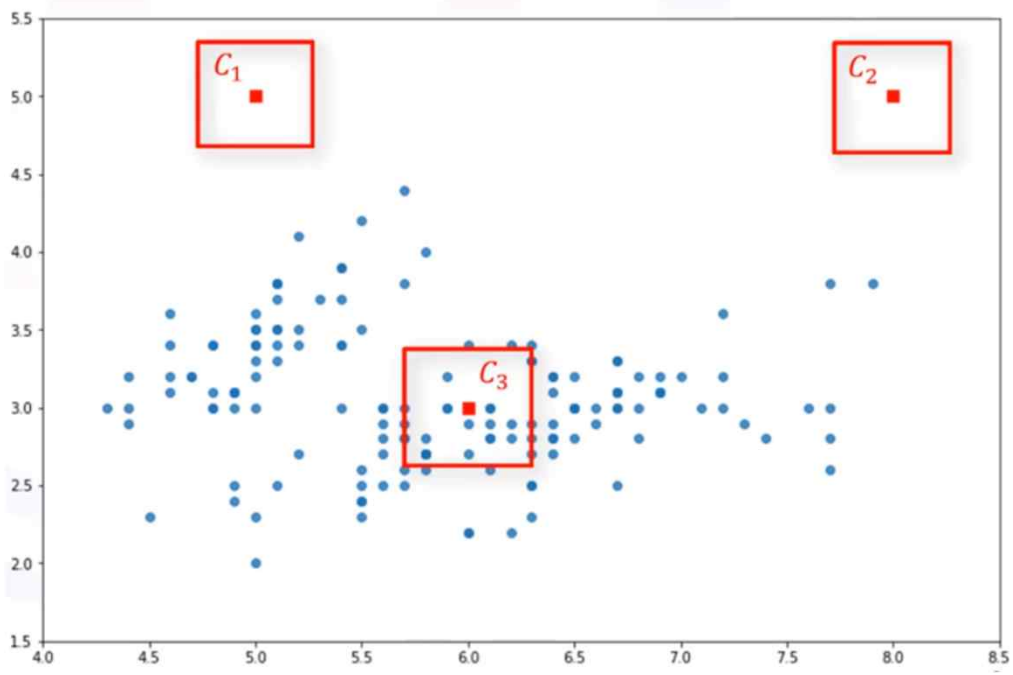
2. 각 centroid 로 부터 각 data point 까지의 거리를 계산.

3. 각 data point 를 가장 가까운 centroid 에 할당하여 cluster 를 생성.

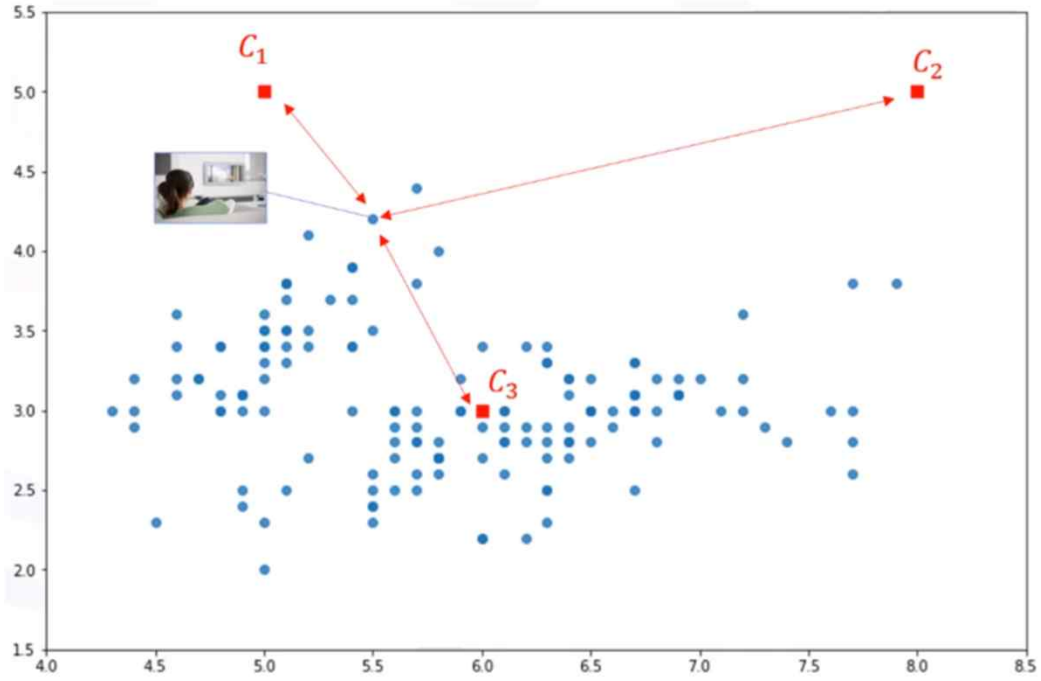
4. K centroid 의 위치를 다시 계산

5. centroid 가 더 이상 움직이지 않을 때까지 2-4 단계를 반복

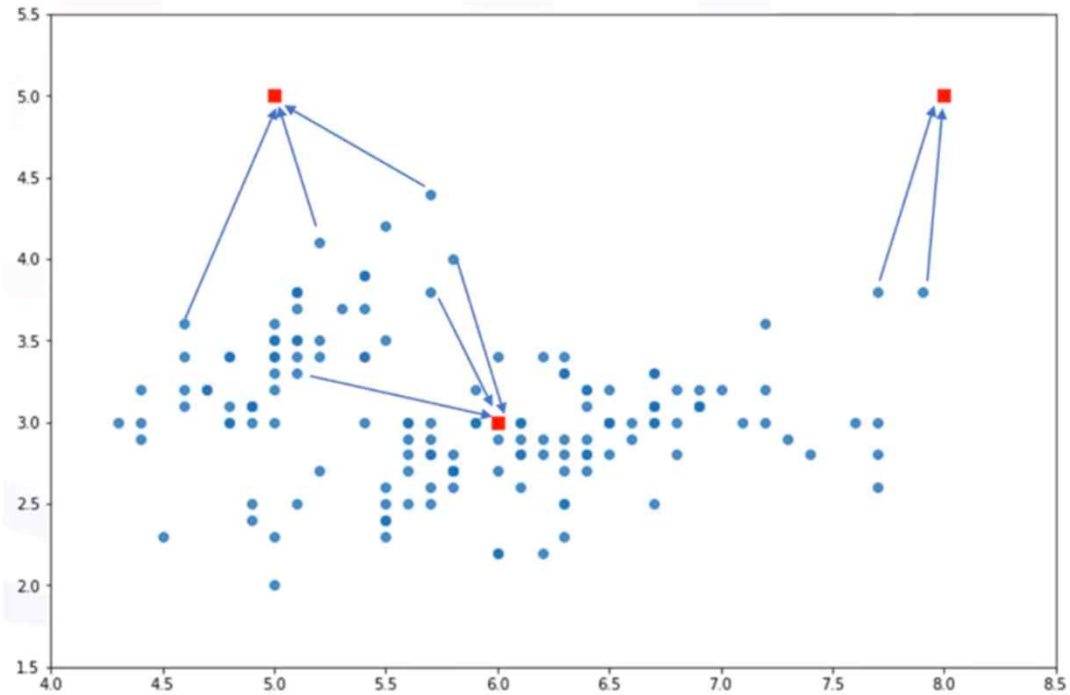
1. 임의의 centroid 선정 : k=3

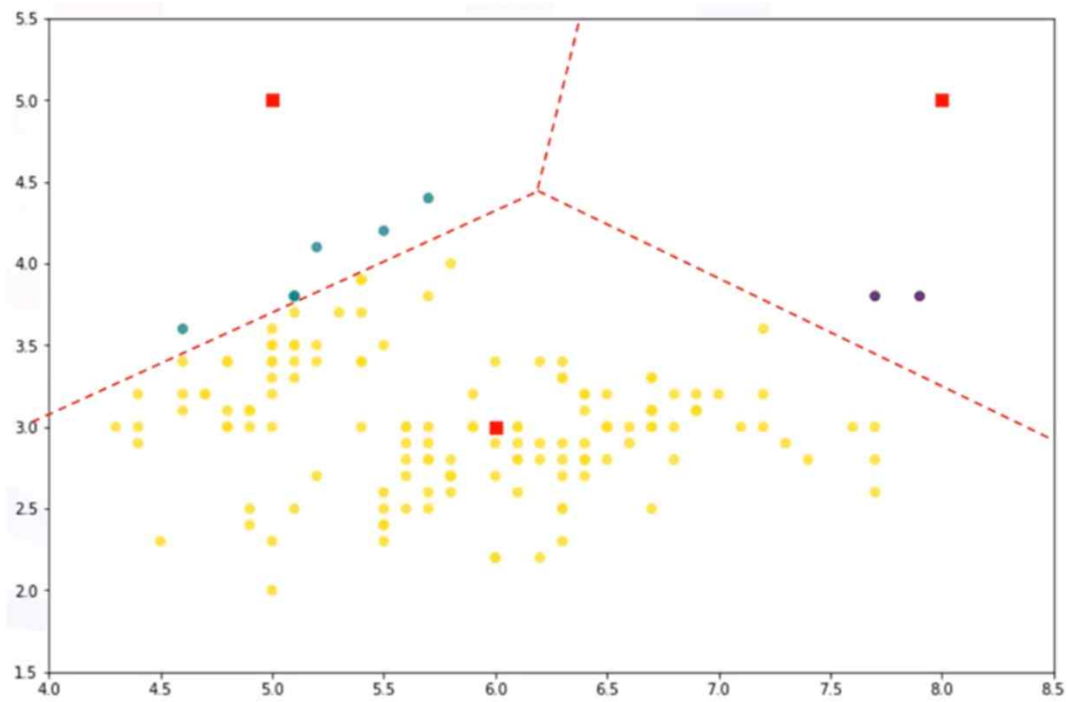


2. 거리 계산 for each data point

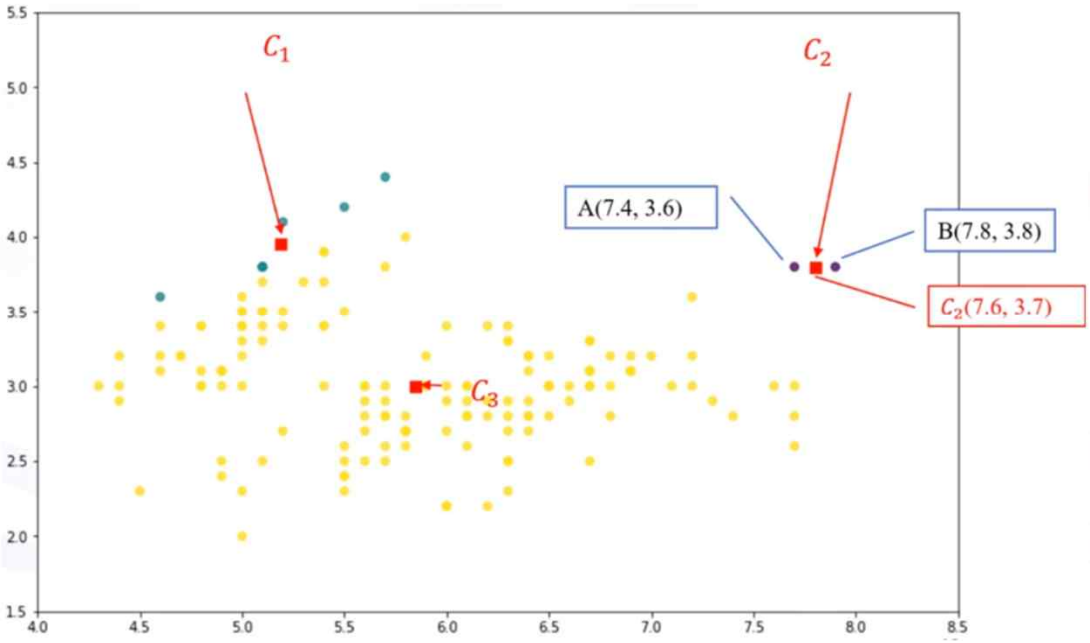


3. 가장 가까운 centroid 로 각 data point 할당

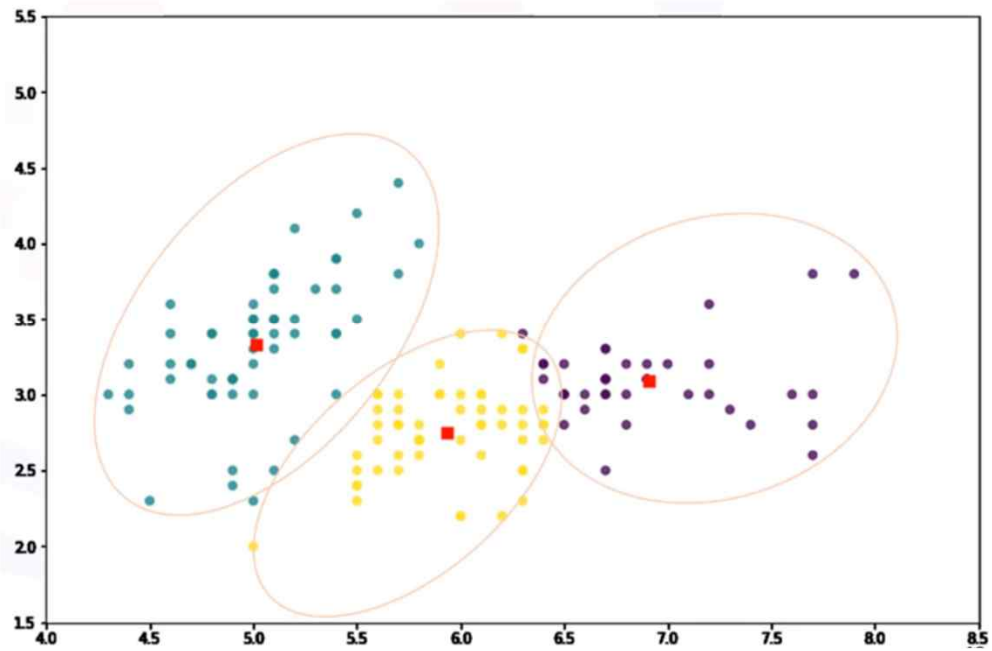




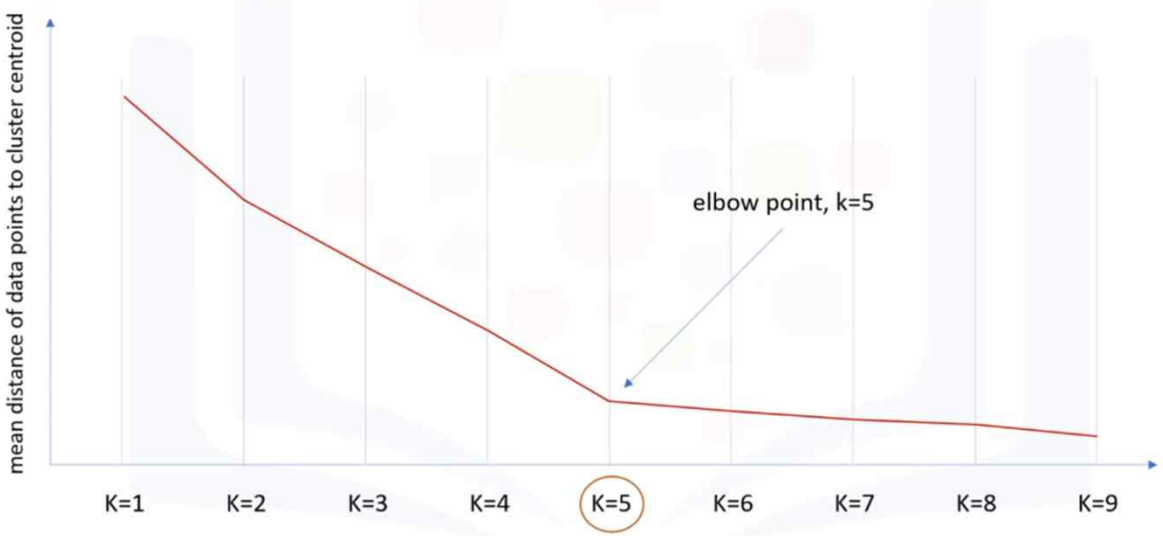
4. 각 cluster 의 new centroid 계산



5. Centroid 변화 없을 때까지 반복



Choosing k

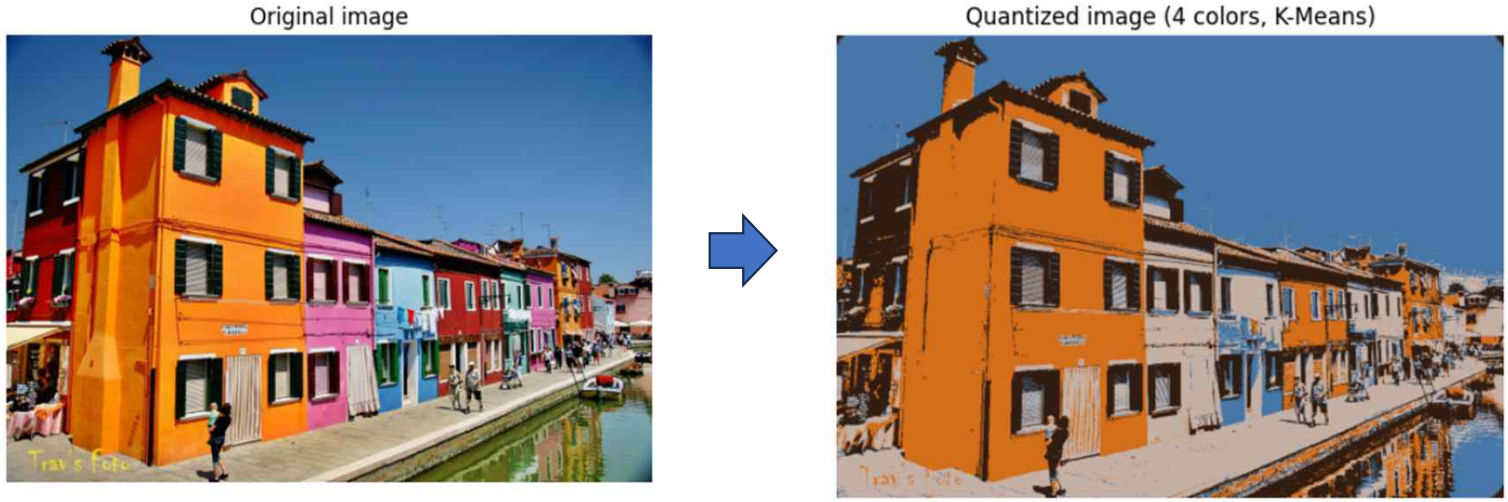


- K 를 잘 정하는 것이 중요
  - How ? → 경험적으로 $\mathrm{k} = \sqrt{n}$ (n: data sample 개수)

- feature 가 많을 경우 계산량 증가

실습: 050. 군집을 이용한 이미지 픽셀 분할

- 이미지의 픽셀을 비슷한 특성을 가진 여러 그룹으로 분류
- 이를 위해 군집 알고리즘인 k-평균(K-means) 사용
- k-평균 알고리즘은 각 픽셀의 색상 값을 벡터로 간주하고, 이를 기반으로 비슷한 색상의 픽셀을 같은 그룹으로 묶는다.



실습: 055. 군집을 사용한 준지도 학습 구현

- 준지도 학습(semi-supervised learning)은 일부 데이터만 레이블이 지정된 상황에서 사용되는 기계 학습 방법
- 레이블이 지정된 데이터를 이용해 학습한 후, 이를 기반으로 레이블이 없는 데이터에 대한 예측을 수행
- 학습 순서
  1. 군집 알고리즘을 사용해 전체 데이터를 여러 클러스터로 나눈다.
  2. 각 클러스터 내에서 레이블이 알려진 샘플의 레이블을 해당 클러스터에 속한 모든 샘플에 할당.
  3. 이렇게 하면, 레이블이 없는 샘플들도 레이블을 가지게 되며, 이 정보를 이용해 분류 모델을 학습할 수 있다.

이상 거래 검출
접근 방법

규칙(Rule) 기반 접근 방식

- 이상 거래 분석가가 알고리즘 작성
- 단점
  - Rule에 지정되지 않는 영역에 대응 못함
  - customer와 data 증가하면 human effort 함께 증가
    → 다양한 예외 case 발생
  - Hidden pattern(잠재 pattern)은 찾아낼 수 없음
  - rule에 해당 안되는 것은 모두 정상 거래로 인식
- 장점
  - 빠르게 적용 가능
  - 쉽게 설명 가능
  - simple

규칙(Rule) Examples

- 고객이 일상적으로 구매하던 지역과 다른 지역에서 사용 되었는가 ?
- 해당 고객에게 이례적으로 이전 보다 빈번히 사용 되었는가 ?
- 모르는 계좌에서 이례적으로 큰 금액을 송금 받았는가 ?
- 새로 생성된 계좌인가 ?
- 동일 IP에서 단기간에 여러 계좌가 생성 되어 송금에 사용되었는가 ?

인공 지능 기반 이상 거래 검출

- Online transaction 에서 수집된 Big Data를 활용하여 이상 거래 예측
- 인간의 개입을 최소화 하여 효율 향상, 비용 절감
- Machine Learning 기법 사용
    - Classification
    - Clustering / anomaly detection
    - Autoencoder, Variational Autoencoder

이상 거래 검출 문제의 어려운 점

- 불균형 데이터
  - 0.5% 미만의 small transaction만 이상 거래 case
- 운영 시스템의 효율성
  - 수초 이내에 flagging 필요
- 부정확한 flagging → 정상 거래 고객에게 불편을 끼침

Abnormal data (novel data, outlier) 란 ?

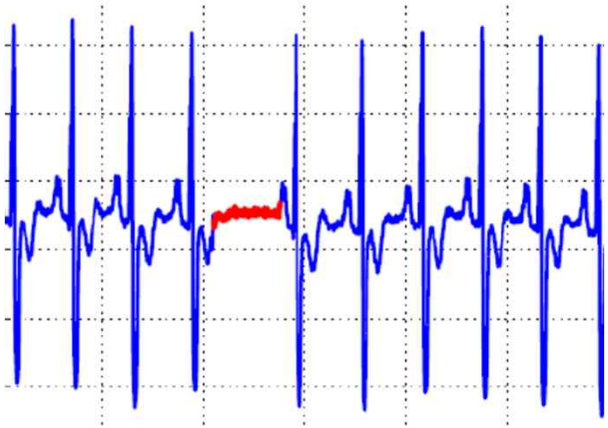
- 정상 data 와 다른 mechanism 으로 생성된 data

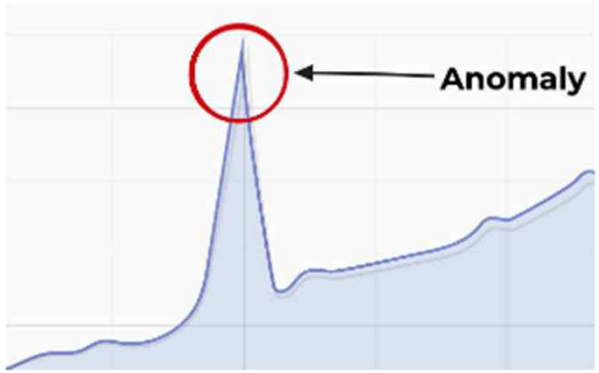
- 발생할 가능성이 매우 낮은 data
  → 다른 관측치와 비교하여 많이 벗어나 있는 관측치

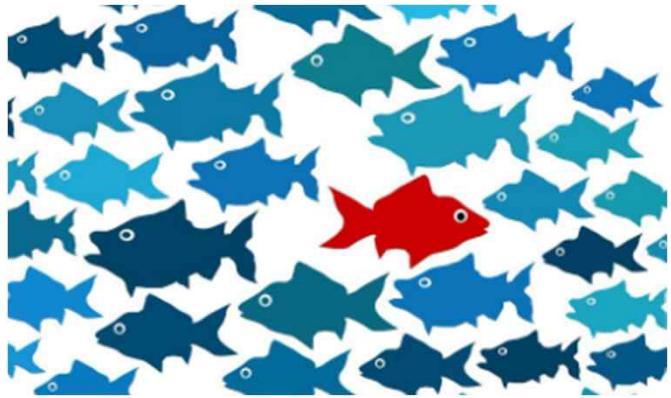
- Noise - 정상 data에 자연 발생적으로 섞여들어간 error

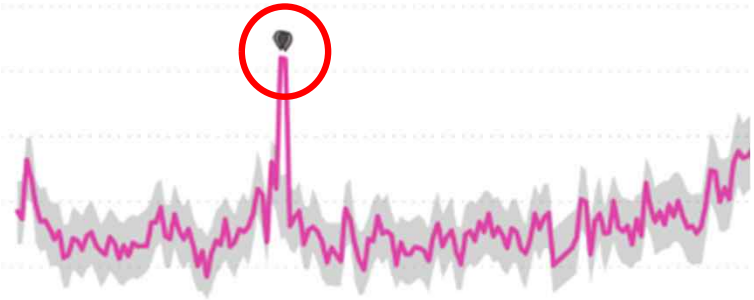
- 생산 공정, System Security, 심장박동, 신용카드 사용의 이상 pattern 등

Anomaly/outlier examples







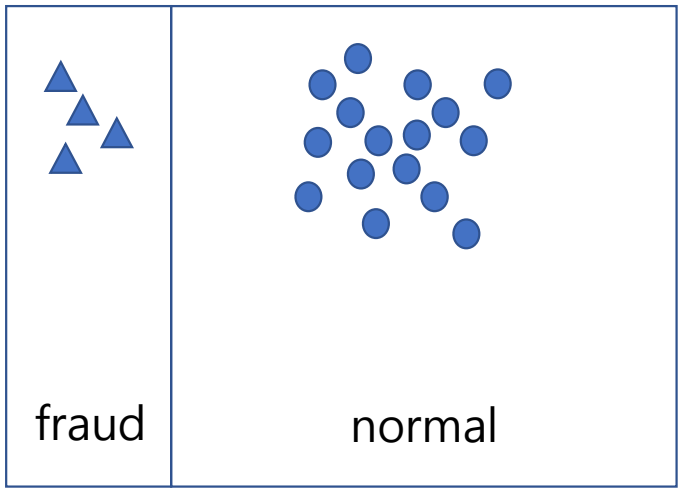


이상감지 (Anomaly Detection)

Outlier detection (이상치 감지)	• **훈련데이터에** 다른 관측치와 멀리 떨어진 관측치로 정의되는 **이상치가 포함**. • 따라서 이상치 탐지기는 비정상 관찰을 무시하고 훈련 데이터가 가장 집중된 영역을 맞추려고 한다. • 저밀도 영역에 위치한다고 가정. 따라서, **조밀한 클러스터 형성 않음**.
Novelty detection (특이치 감지)	• **훈련데이터는 정상 데이터로만 구성**. 따라서 새로운 관측치가 훈련 데이터와 비교해서 많이 떨어져 있는지 여부를 감지. → 특이치 라고도 함. • 정상으로 간주되는 학습데이터의 저밀도 영역에 있는 한 **조밀한 클러스터를 형성할 수 있다**.

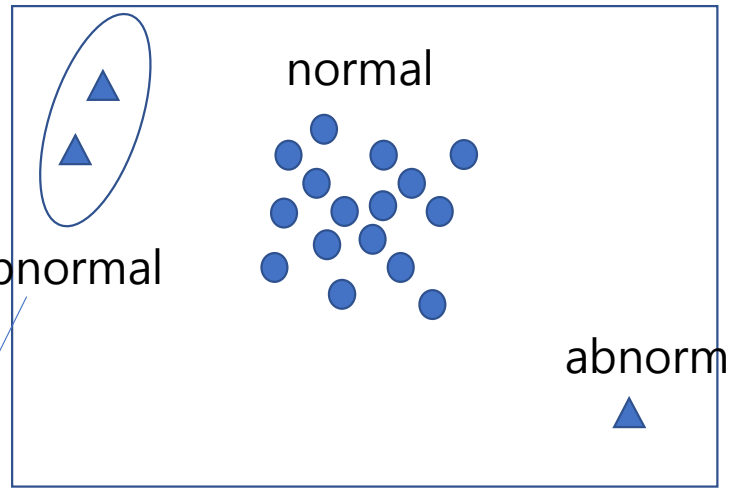
지도 학습 vs 비지도 학습 이상치 탐지

label 기반 **지도 학습**



이진 분류

밀도 기반 **비지도 학습**

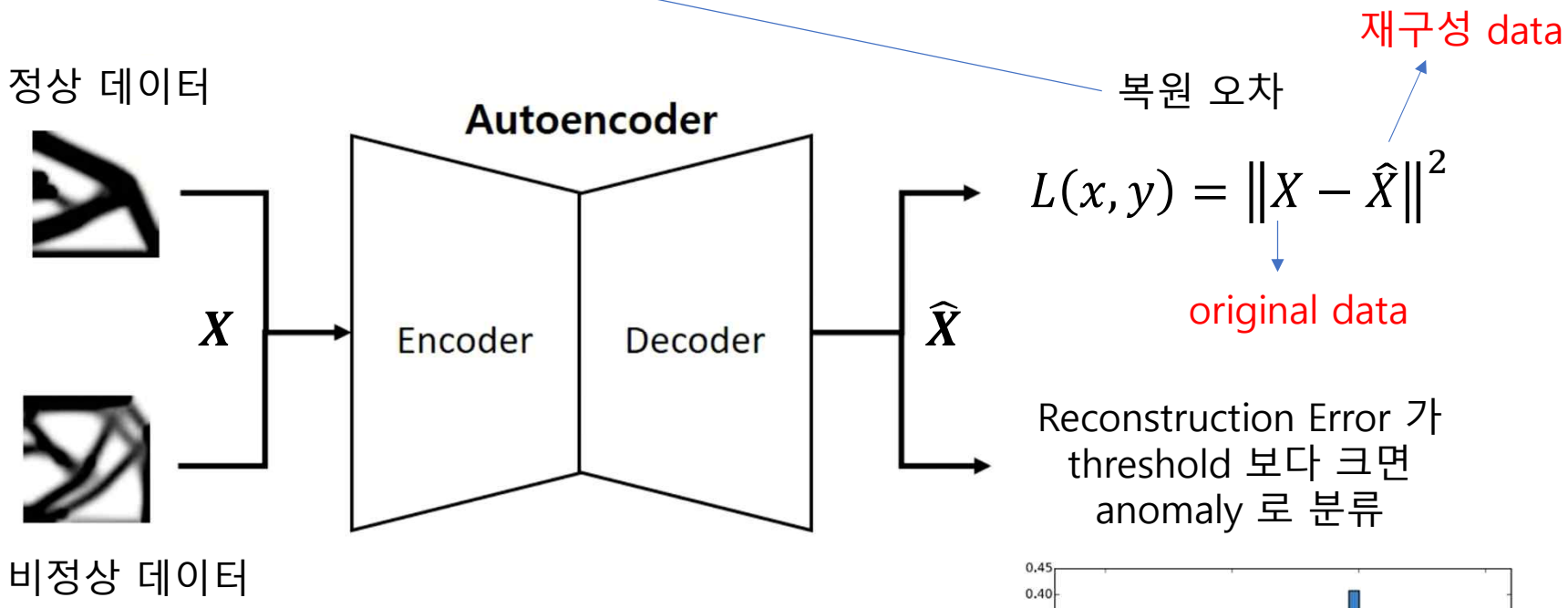


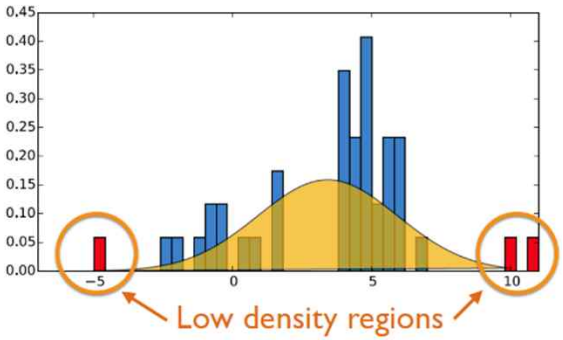
저밀도
영역에 위치

멀리 떨어진
관측치

Anomaly detection

재구성 오류 측정 이상 감지 (Autoencoder, VAE 사용)





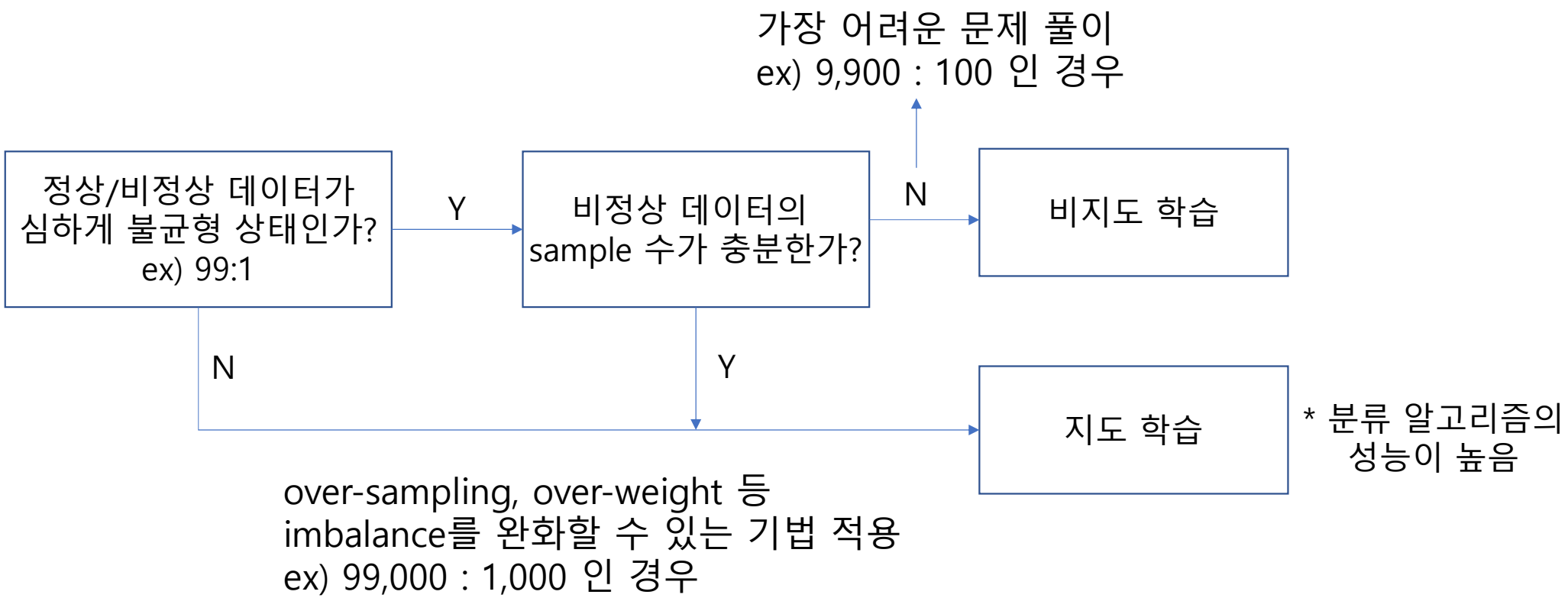
- 정상 데이터 – 복원 잘 됨 (복원 오차 small)
- 비정상 데이터 – 복원 잘 안됨 (복원 오차 big)

Anomaly Detection 방법 분류

구분	접근 방법	측정 기준	알고리즘
전통적 Machine Learning	지도 학습 방법	Label 데이터	이진 분류
	Random split 방법	Number of random splits	Isolated Forest
	Proximity(근접도) 기반 방법	clustering, density, distance	DBSCAN, LOF(Local Outlier Factor), K-nearest Kernel method
Deep Learning	Deviation(편차) 기반 방법	재구성 오류	Autoencoder
	통계적 방법	확률 분포의 차이	Variational Autoencoder

ML 알고리즘 선택 기준

- 9,900:100 과 99,000:1,000 은 같은 1% 이지만 minority class의 수가 많을 수록 유리

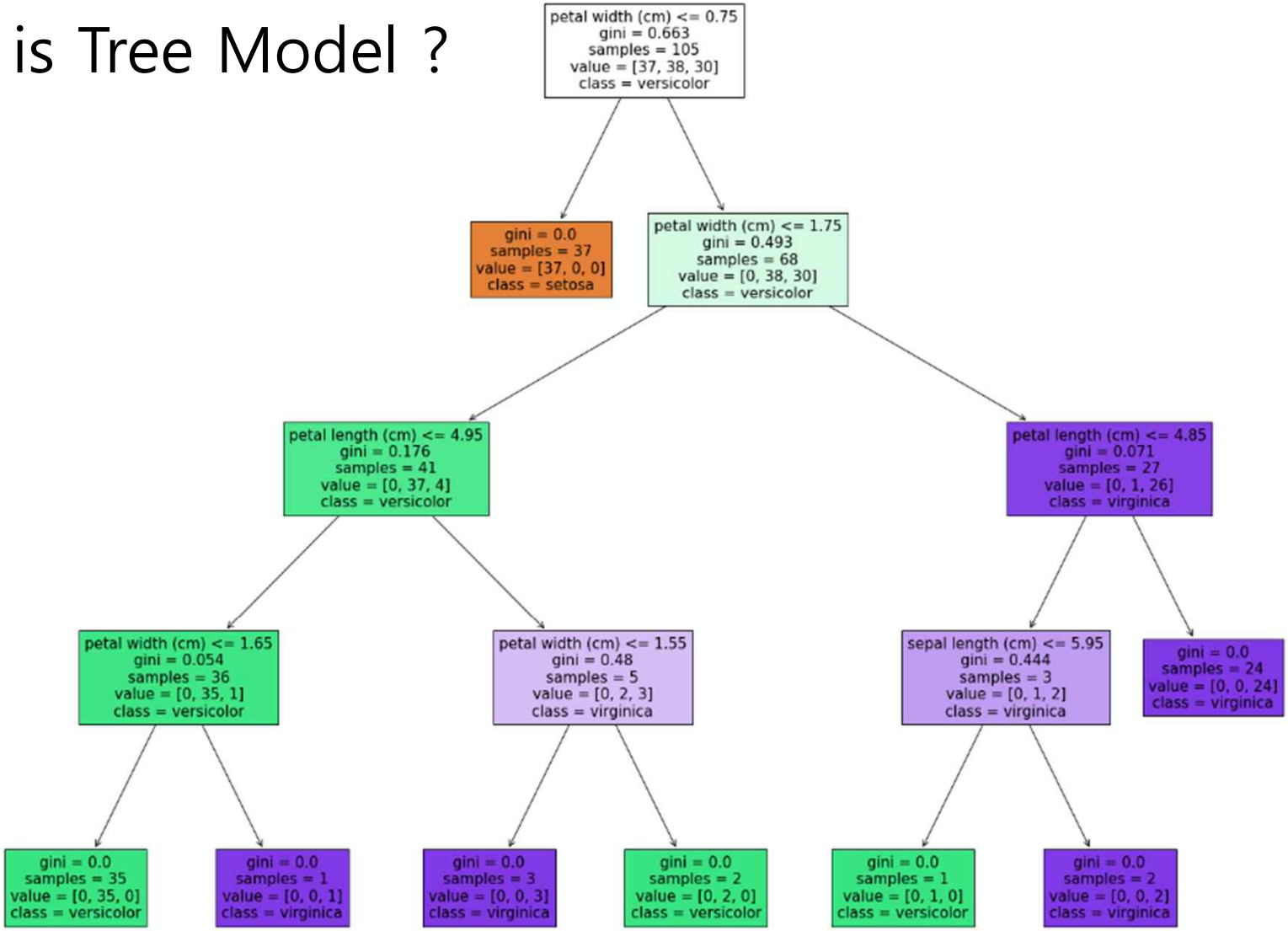


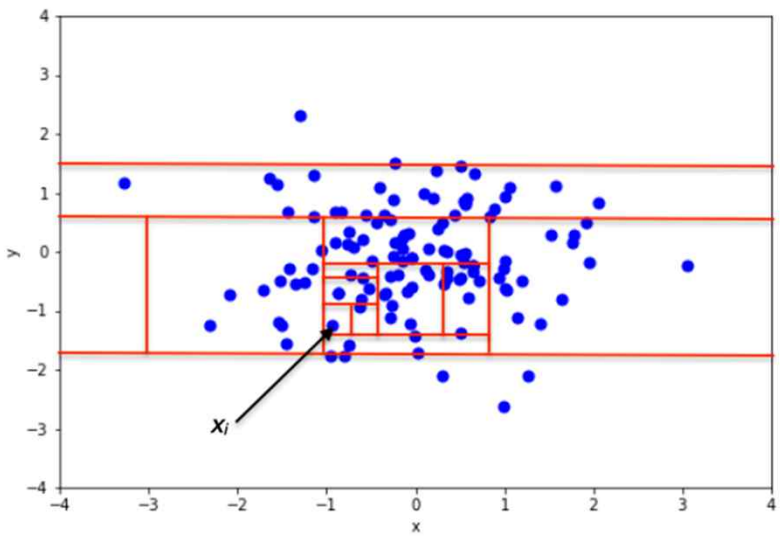
Isolation Forest

What is Isolation Forest ?

- Tree model을 이용한 이상 탐지 비지도학습 알고리즘
- Regression Decision Tree를 기반으로 실행
- Regression Tree가 재귀 이진 분할을 이용하여 영역을 나누는 개념을 이용
  - Random forest와 같이 feature를 random하게 선택
  - 선택된 feature의 maximum, minimum 값 사이의 split value를 이용해 tree 구현

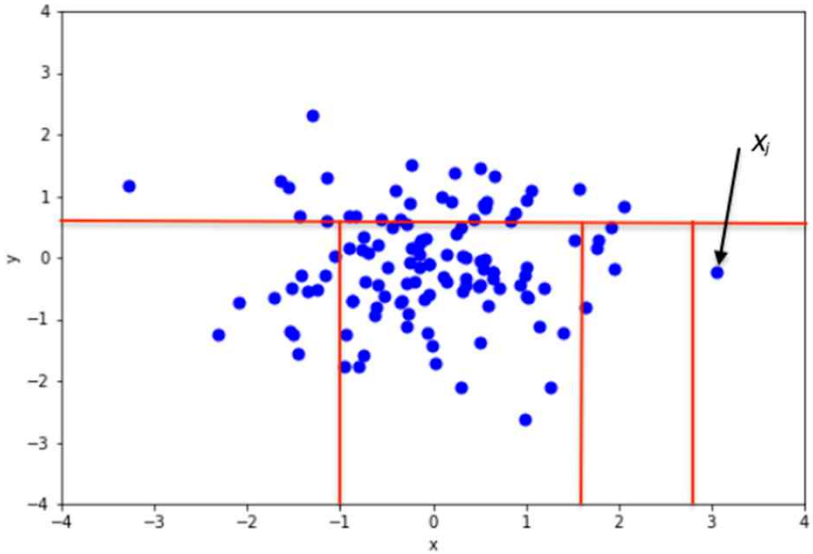
What is Tree Model ?





정상 Data

많은 분할 수



비정상 Data

적은 분할 수

Isolation Forest Algorithm

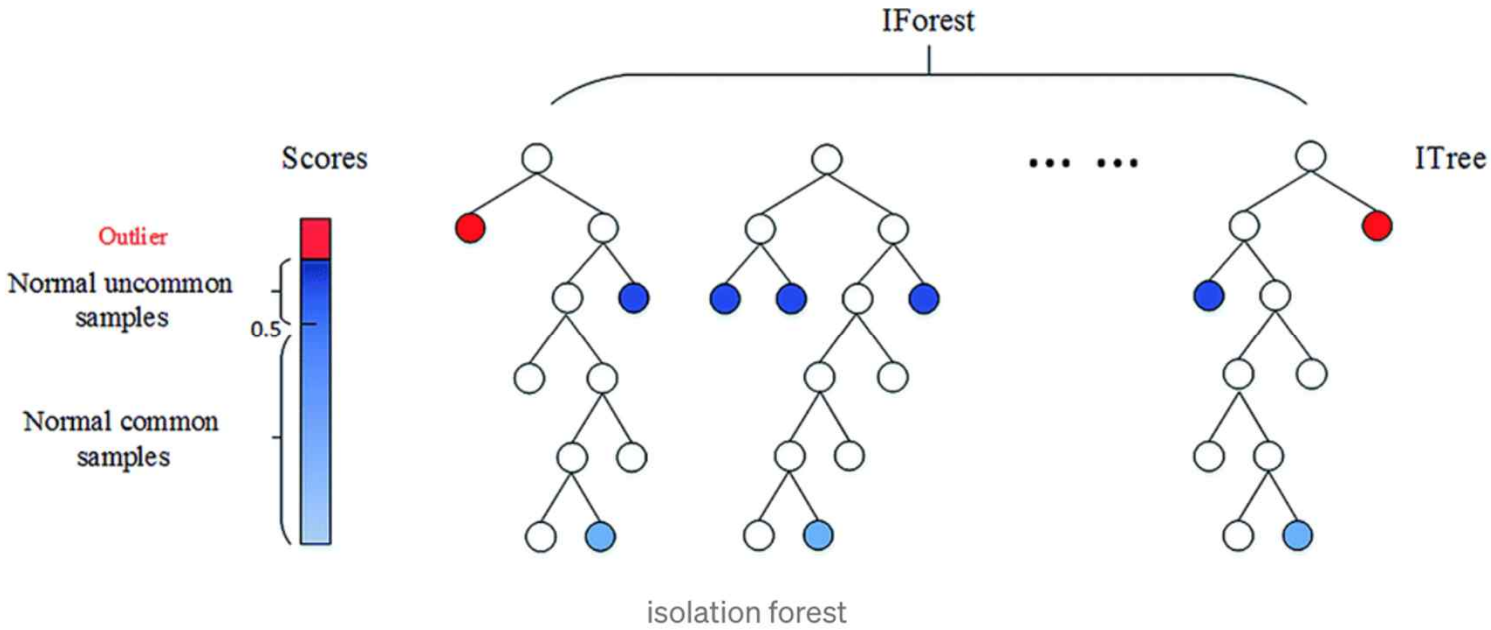
- 일반적으로 정상 데이터의 경우, 더 많은 재귀 이진 분할이 필요

- 반면에 비정상 데이터는 정상데이터에 비해 이진 분할이 덜 필요
  → 예외는 정상에 비해 분리하기가 더 쉽다
  (재귀 이진분할이기 때문에 tree의 깊이가 짧을 수록 비정상 데이터일 가능성이 높음)

- 정상 데이터 보다 비정상 데이터에서 더 적은 분할 수로 나눠진 영역에 anomaly data가 있음

  - **비정상 데이터**가 고립되려면, **root node**와 가까운 depth를 가짐
  - **정상 데이터**가 고립되려면, **tree의 말단 노드**에 가까운 depth를 가짐

- 정상 데이터 보다 비정상 데이터에서 더 적은 분할 수로 나눠진 영역에 anomaly data가 있음



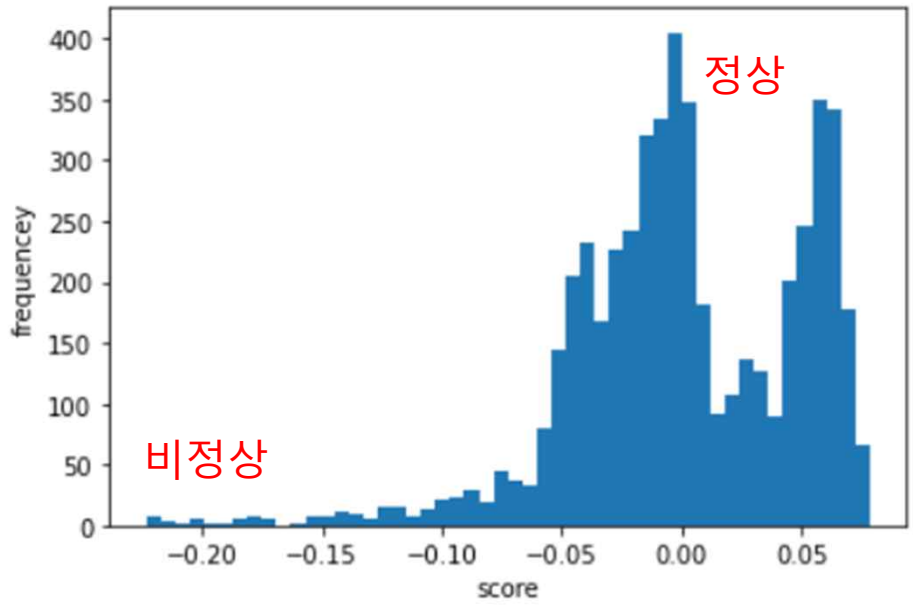
- **비정상 데이터**가 고립되려면, **root node**와 가까운 depth를 가짐
- **정상 데이터**가 고립되려면, **tree의 말단 노드**에 가까운 depth를 가짐

Isolation Forest Score

- Isolation Tree를 여러 개의 앙상블 모델을 만들어 이상 지수 score 계산

- score가 **음수**일 수록 **비정상** anomaly

- score가 **양수**이면 **정상** 데이터로 판단



실습 : 060. Isolation Forest 를 이용한 anomaly 검출

- NYC Taxi 탑승 정보

- Time Series data 를 이용한 anomaly data 검출

- New York 마라톤, 추수감사절, 크리스마스, 눈보라 친 날씨 등 평일 보다 특별히 많은 Taxi 탑승이 발생한 날자 검출

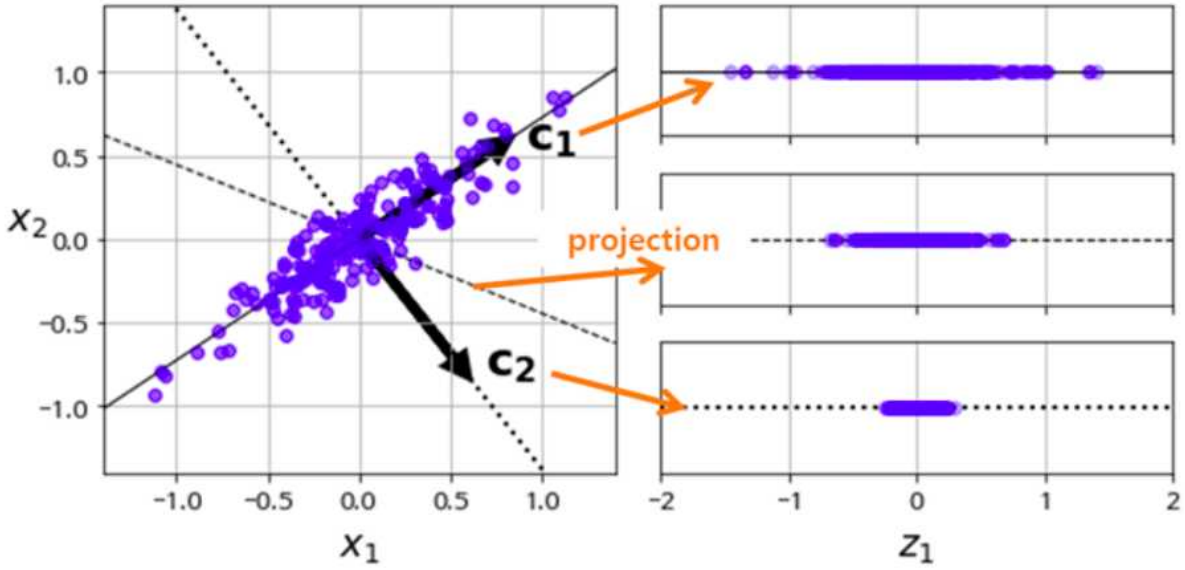
Deep Neural Network

비지도 학습 접근 방법

Autoencoder

PCA (Principal Component Analysis) - 주성분 분석

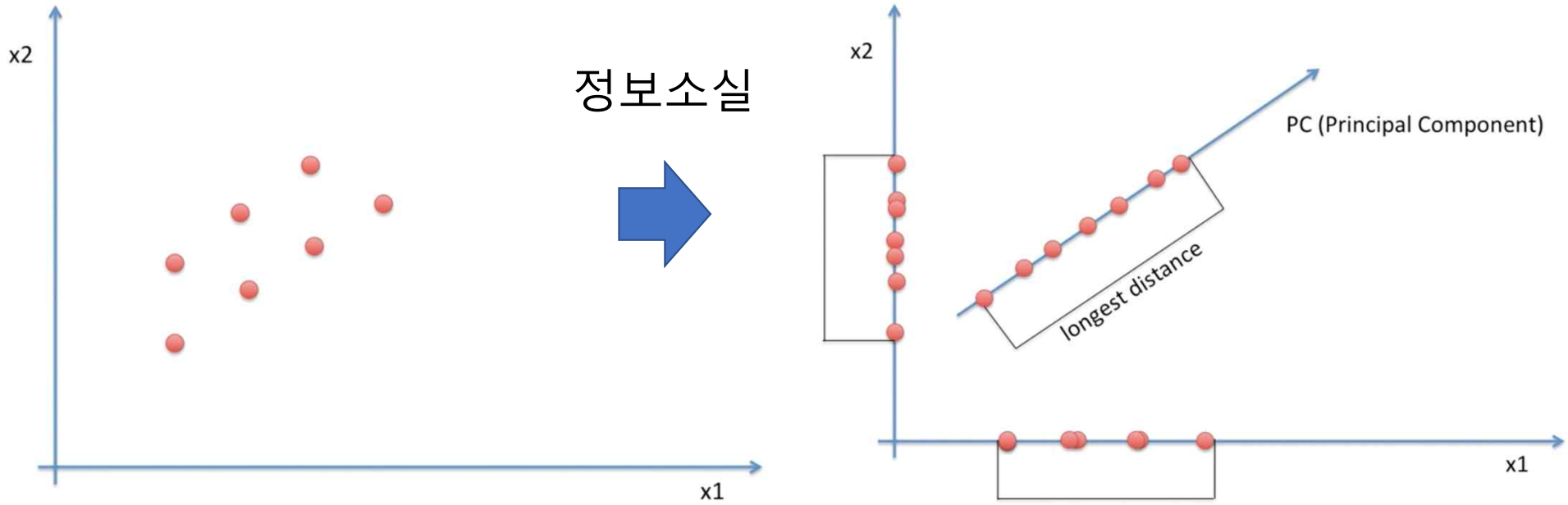
- **선형대수학의 SVD (특이값 분해)** 를 이용하여 **분산이 최대인 축**을 찾음
- 데이터의 **분산(variance)**을 최대한 보존하면서 서로 직교하는 새 축을 찾아, 고차원 공간의 표본들을 선형 연관성이 없는 저차원 공간으로 변환



분산을 최대한 보존

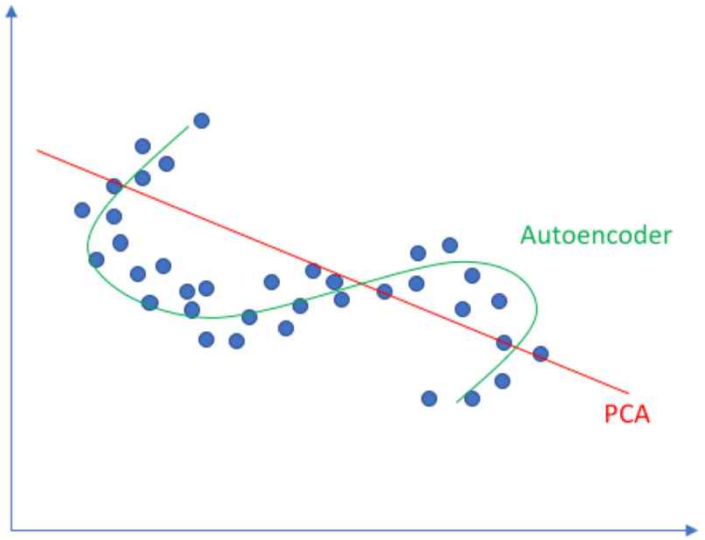
https://i.imgur.com/Uv2dlsH.gif

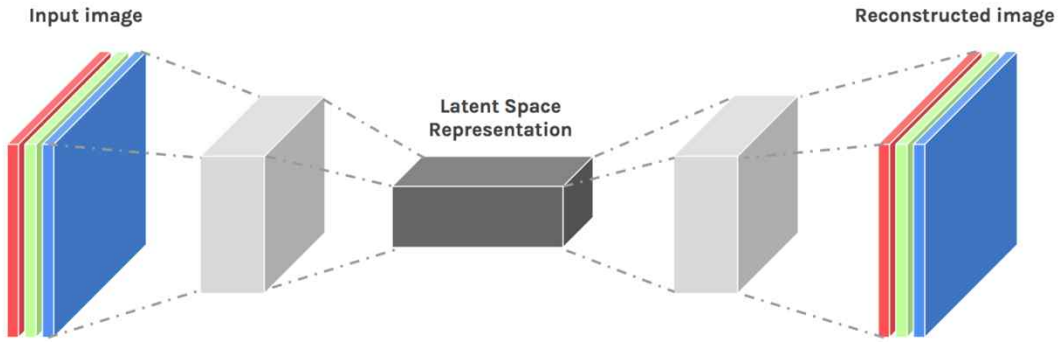
PCA - how to reduce dimension ?



Autoencoder

- 차원의 감소 : most relevant feature 추출
  - PCA : linear transformation
  - Autoencoder : non-linear transformation
- 적용 분야
  - 정보의 압축
  - Noise 제거
  - 유사한 image 검색
  - Image 변형에 의한 새로운 image 생성
  - Pre-Training





latent representation (잠재 표현) - intuition

83 12 21 42 99 18 51

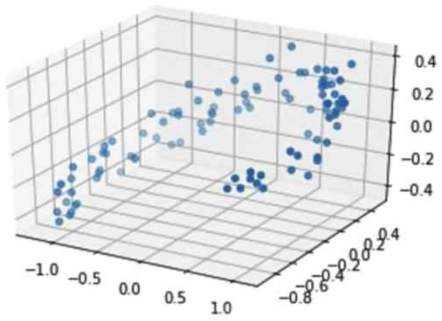
4 9 16 25 36 49 64 81 100 121 144 169

기억하기 쉬운 수열은 ?

- 첫번째 sequence 는 no pattern
- 두번째 sequence 는 제곱 pattern → latent pattern 존재

Autoencoder 의 기본 구조

input : 3 차원



encoder

latent representation 학습

decoder

X1

X2

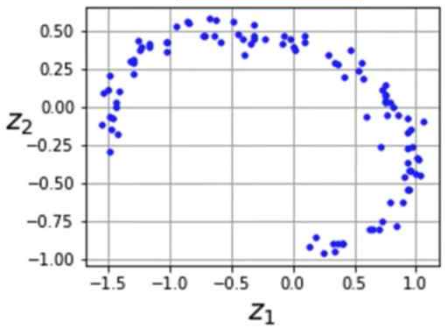
X3

2 차원

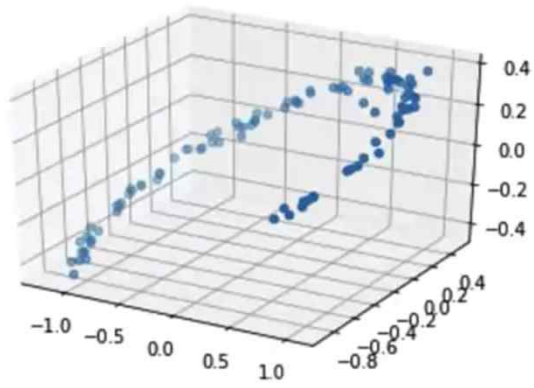
Y1

Y2

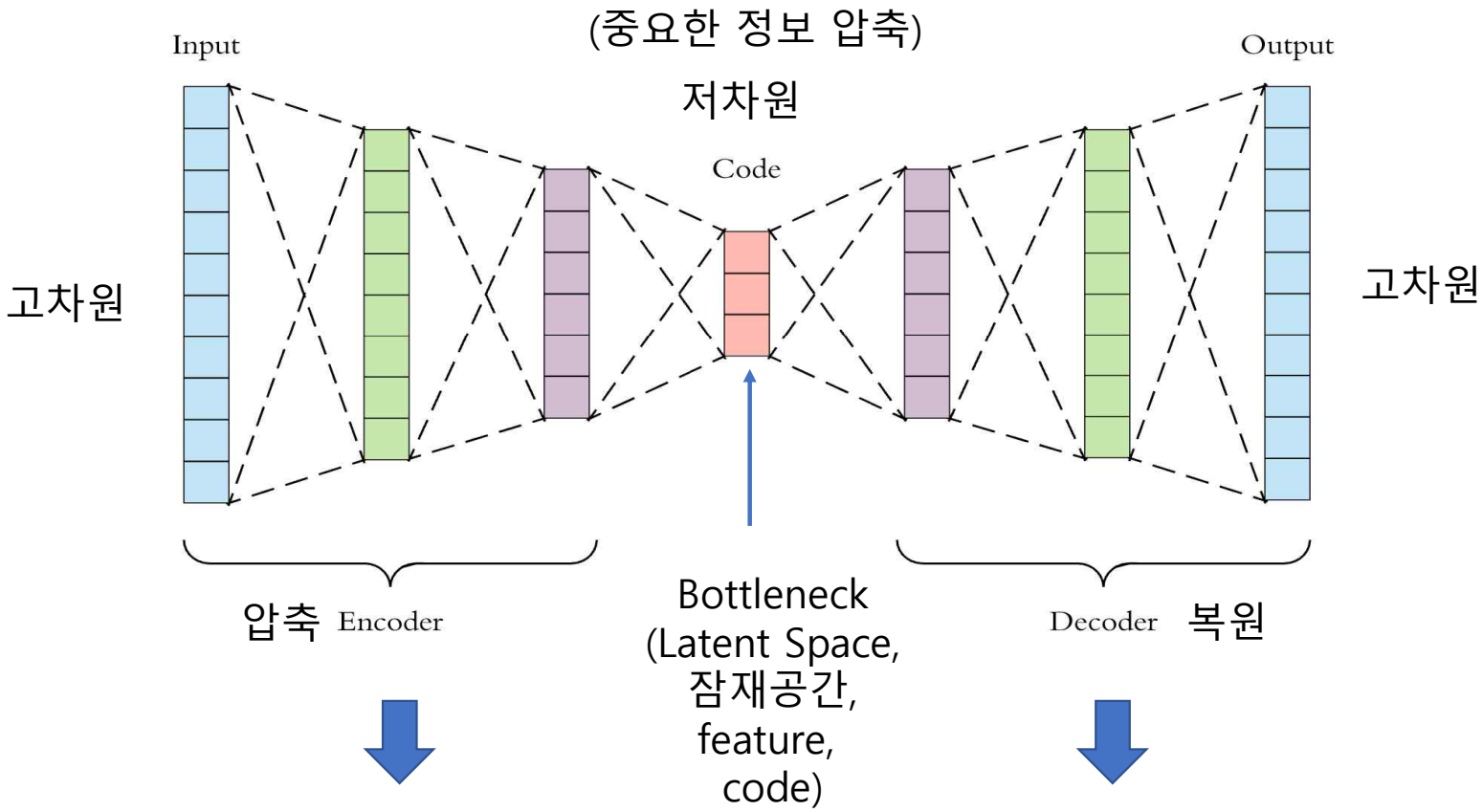
Y3



Output : 3 차원



noise 제거된 input

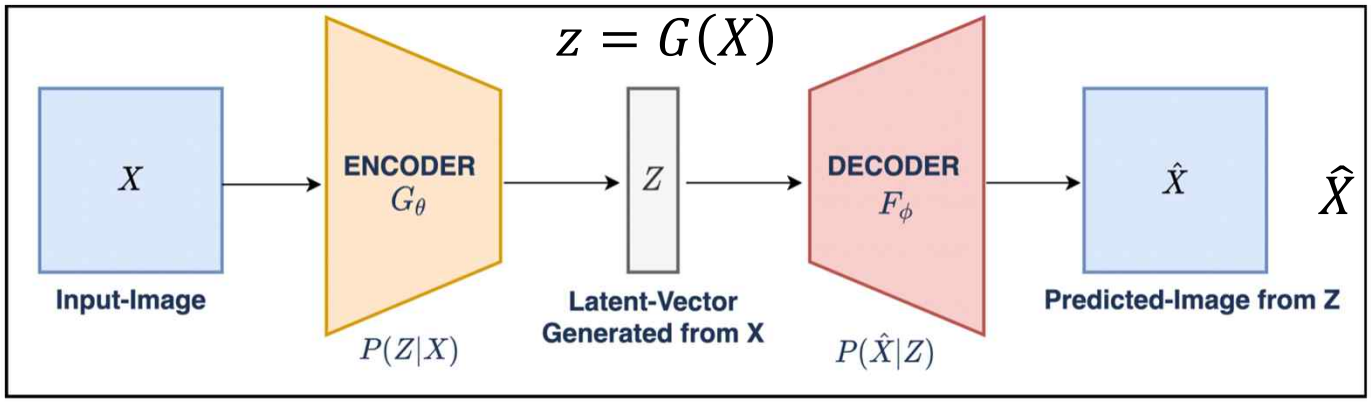


이미지와 같은 고차원 데이터를 Encoding 을 통해 저차원 Hidden 공간으로 표현

→ Latent Space (= Latent Variable)

Latent Variable 의 정보를 기반으로 **원래의 이미지를 복원**(Reconstruction) 하는 과정 수행

Autoencoder – 작동 방식



$$\hat{X} = F(z) = F\big(G(X)\big)$$

$$L(x, y) = \|X - \hat{X}\|^2$$

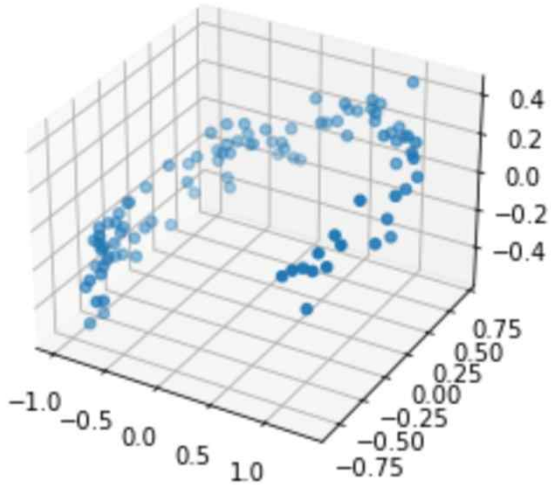
$$z = G(X)$$
$$\hat{X} = F(z) = F\big(G(X)\big)$$

비지도학습 문제를 지도학습 문제로 바꾸어 해결
(자신을 Label 로 사용)

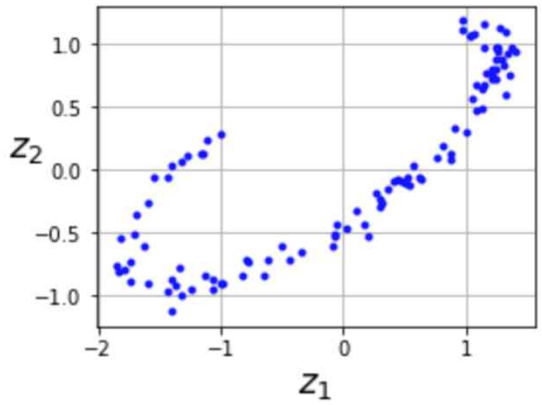
$L(x, y)$ - mse(입력이 정규분포) or cross-entropy(입력이 베르누이 분포)

실습 : 050. autoencoder 시각화

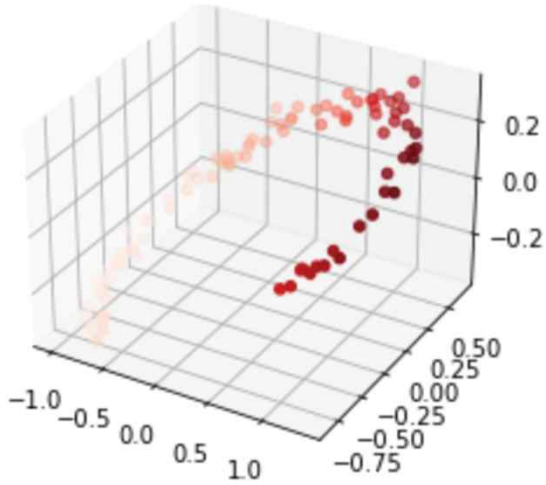
original



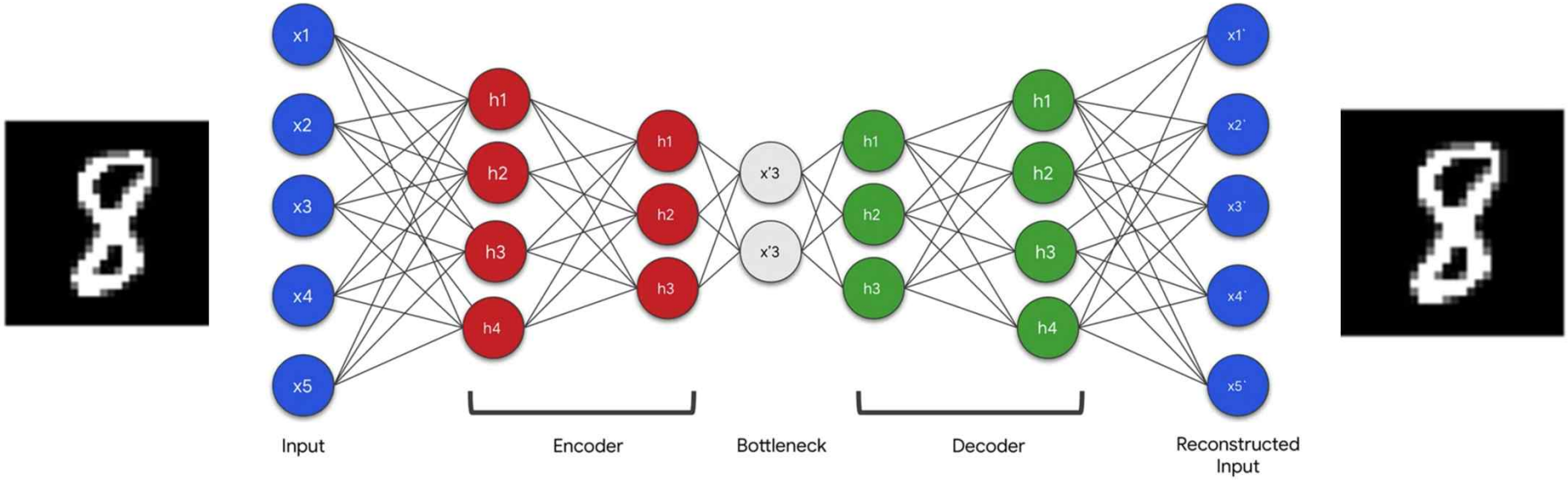
latent



restructured

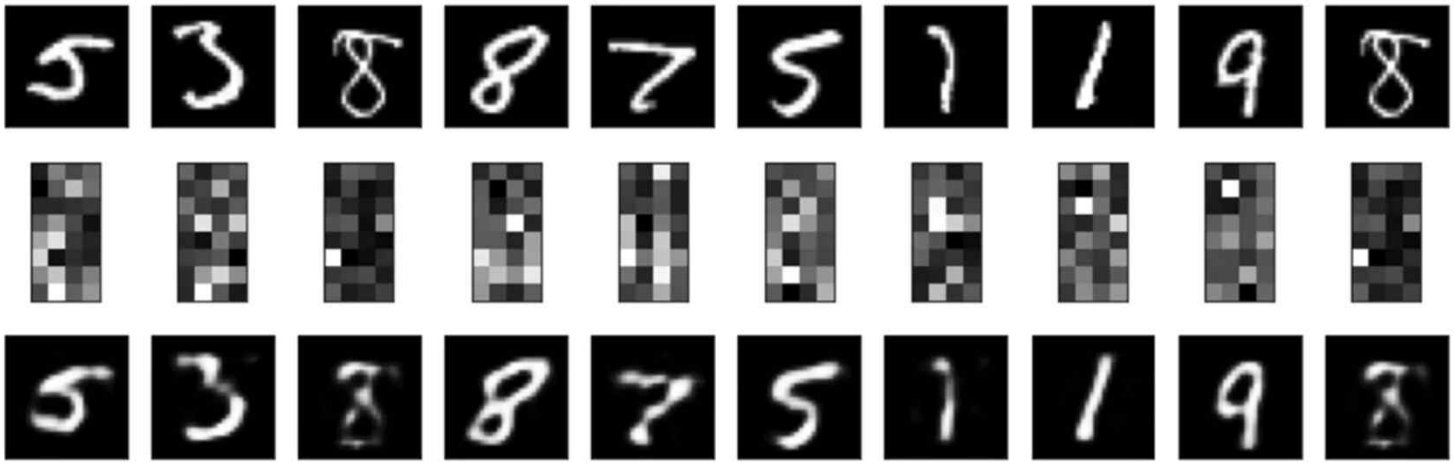


Deep Auto-Encoders



실습 : 460. Simple stacked autoencoder -MNIST

- (Fashion) mnist dataset 을 encoding 후 decoding 하여 복원
- simple / stacked autoencoder 작성

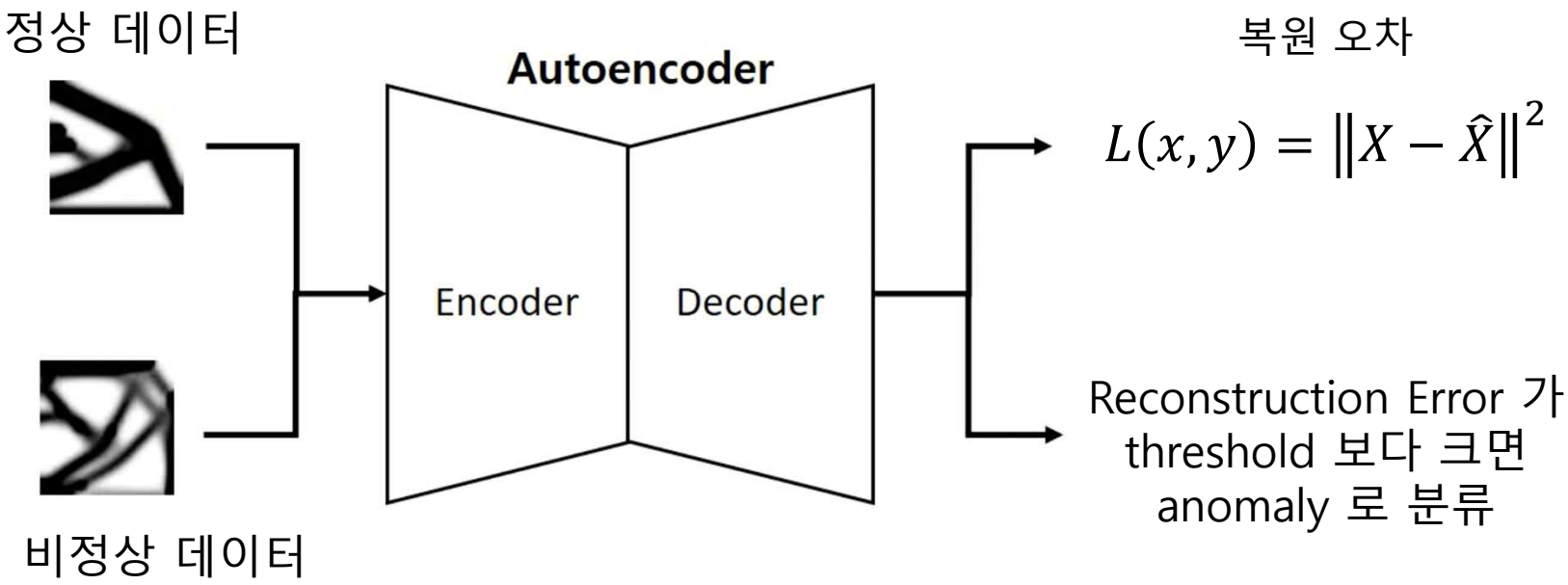


Autoencoder 의 Anomaly Detection 적용 방법

- 방법 1 – Autoencoder 의 reconstruction error 생성에 사용
  - **비지도 학습**
  - normal data 로만 autoencoder training후 재구성 오류 측정

- 방법 2 – Autoencoder 의 encoder 출력 → feature 추출
  - **지도 학습**
  - 모든 data (normal + fraud) 로 autoencoder training 후 지도학습 모델 (SVM, KNN, DNN 등)을 분류기로 적용

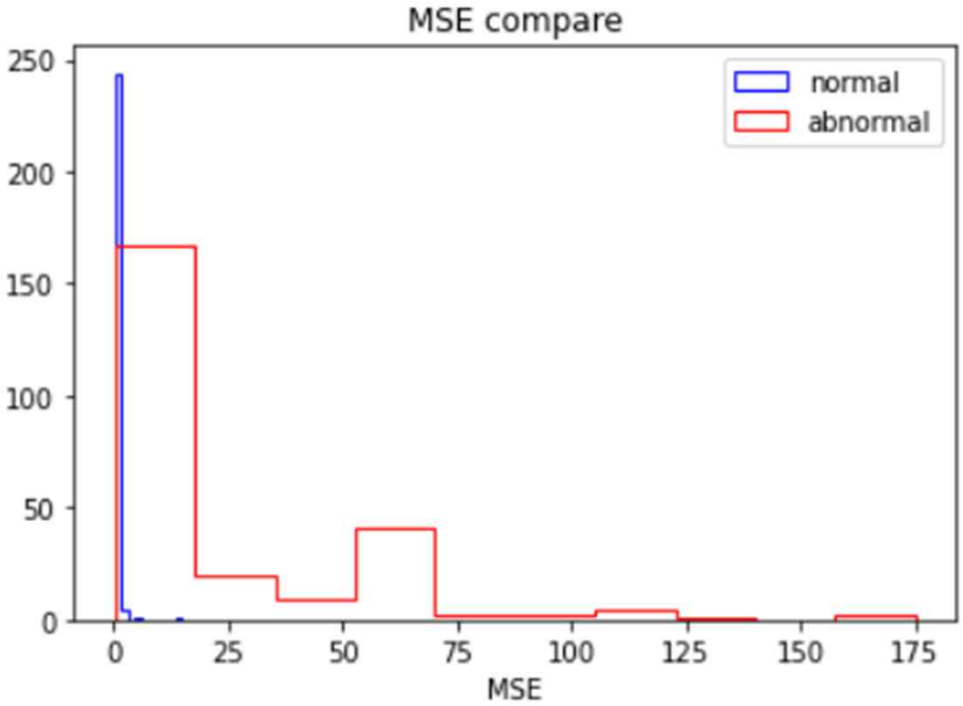
방법 1 – reconstruction error 측정

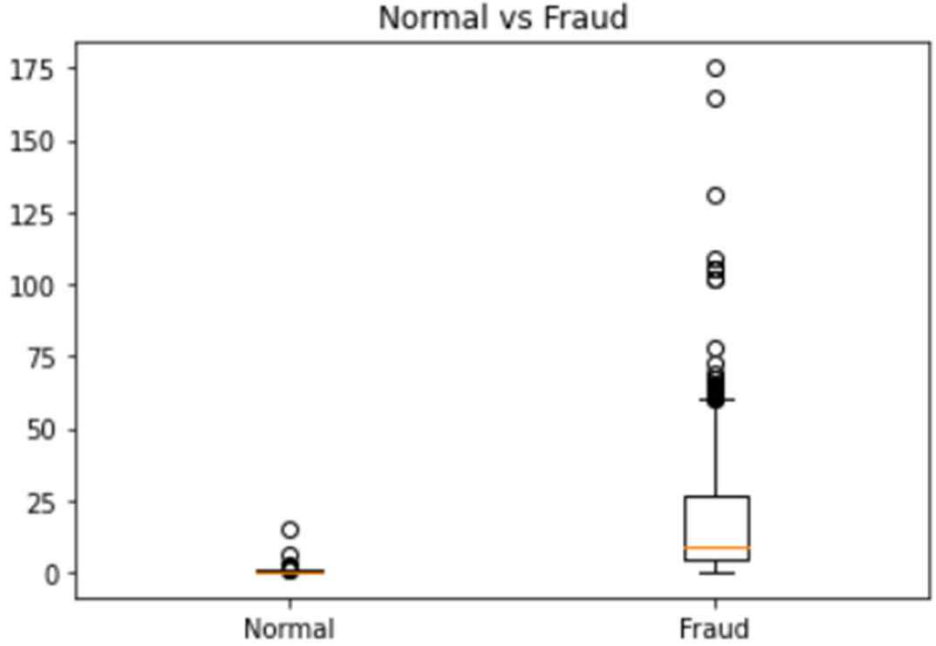


정상 데이터 – 복원 잘 됨 (복원 오차 small)

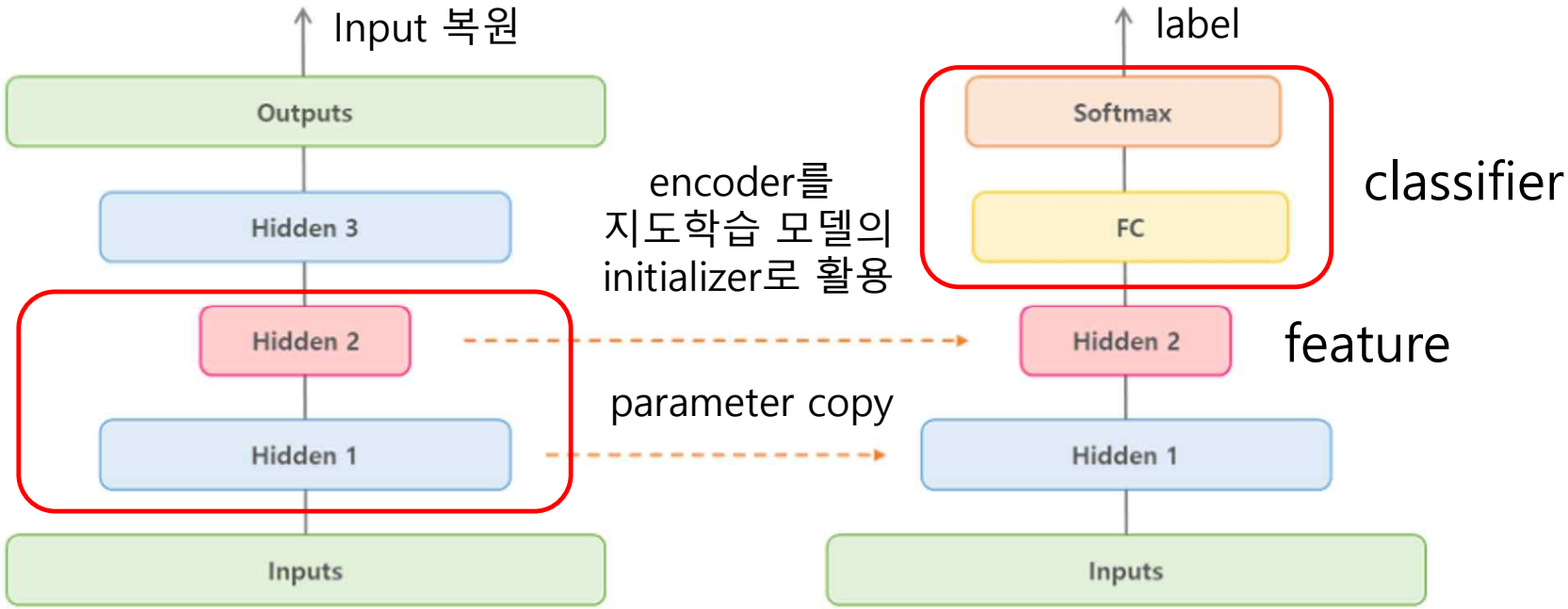
비정상 데이터 – 복원 잘 안됨 (복원 오차 big)

재구성 오류 (Reconstruction Error) 측정





방법 2 – feature extractor 로 사용



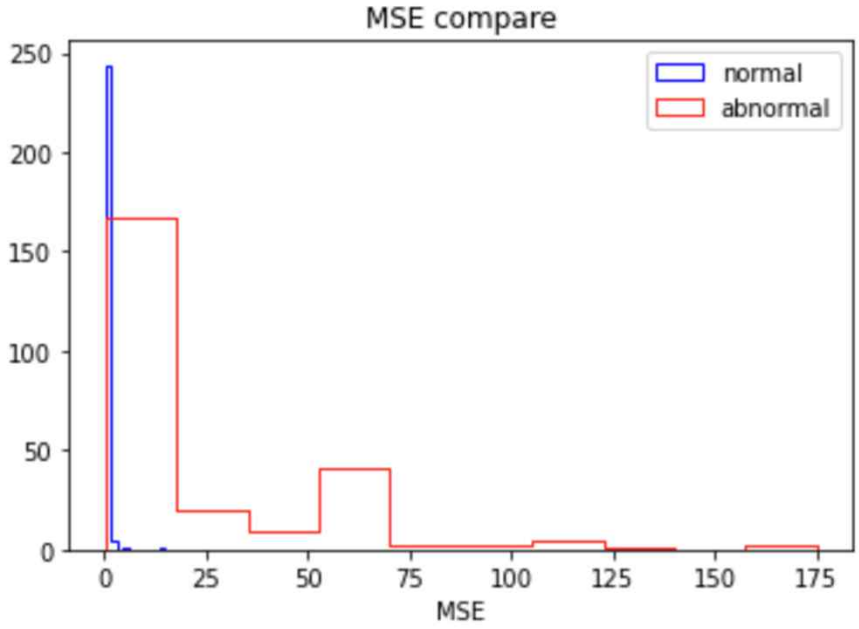
1 단계 – 모든 데이터를 사용해 autoencoder 학습

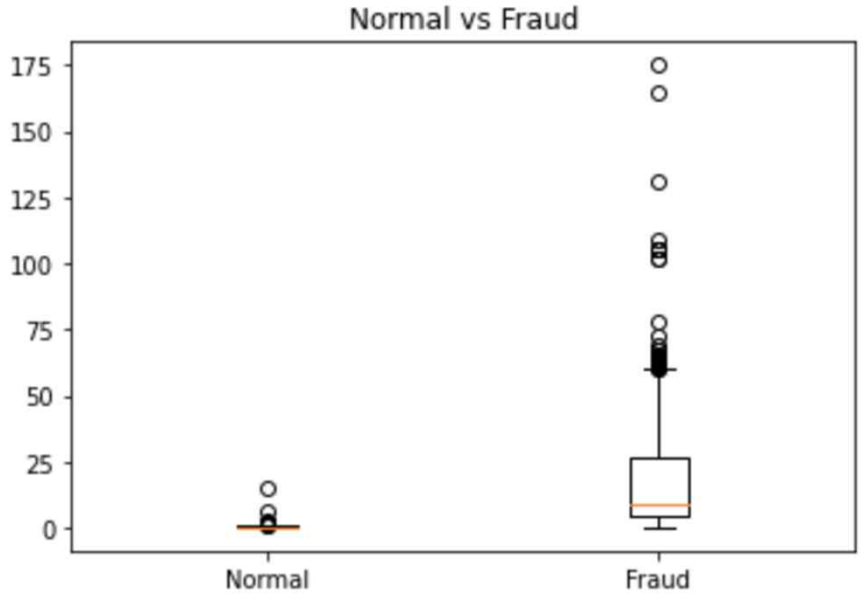
2 단계 – 소량의 데이터를 사용해 지도 학습

실습: 160. Fraud Detection – Reconstruction Error

- Highly Imbalanced Dataset - 2013 년 9 월 유럽 카드 소지자 신용 카드 거래
- 정상 거래로만 오토 인코더를 교육
- 사기성(비정상) 거래의 재구성 error 가 비사기성 거래보다 높을 것으로 기대
- Reconstruction Error 시각화
  - 정상 data 와 anomaly data 간의 MSE 차이 시각화
  - 1 std - 68%, 2 std - 95%, 3std - 99.7%

- 정상 data 와 anomaly data 의 autoencoder output (복원값)과 input(원본) 사이의 mse 를 계산





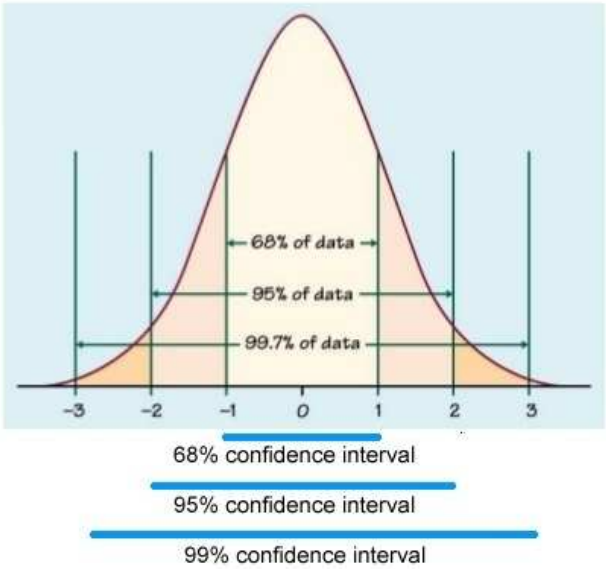
신뢰구간 (confidence interval)

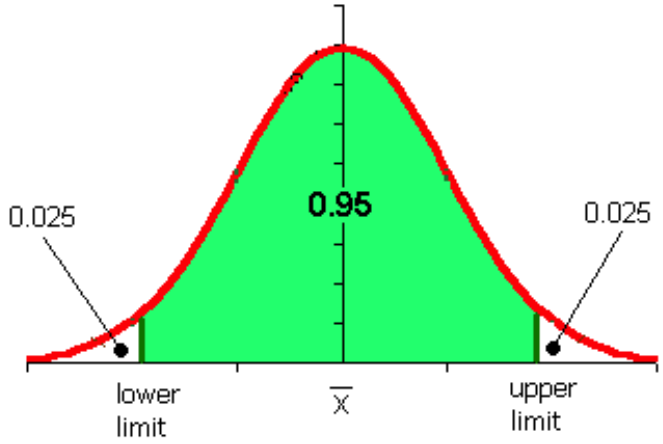
- 모평균을 포함할 확률이 xx % 가 되는 구간

  ("interval_left" 와 "interval_right" 사이에 위치할 확률이 xx %)

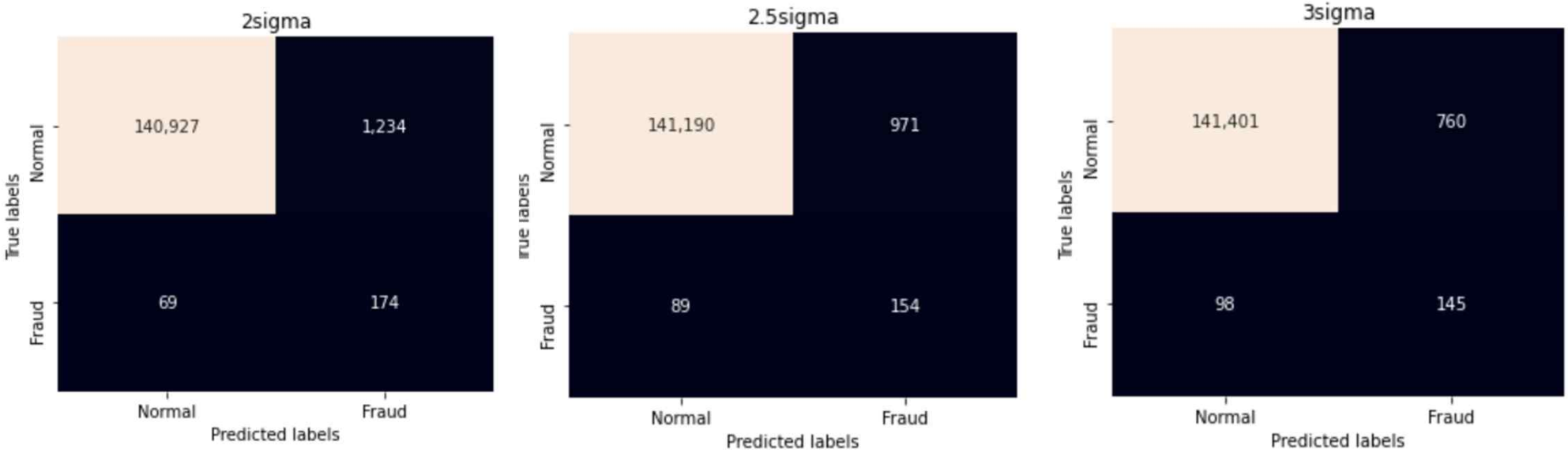
- Empirical Rule

  1 σ - 68%
  2 σ - 95%
  3 σ - 99.7%

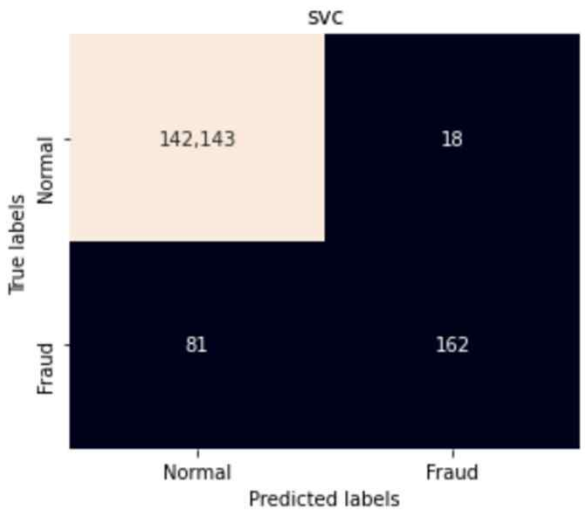




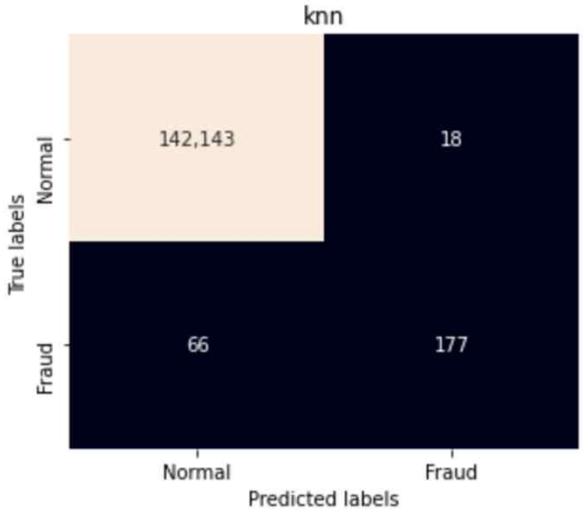
threshold 별 confusion matrix



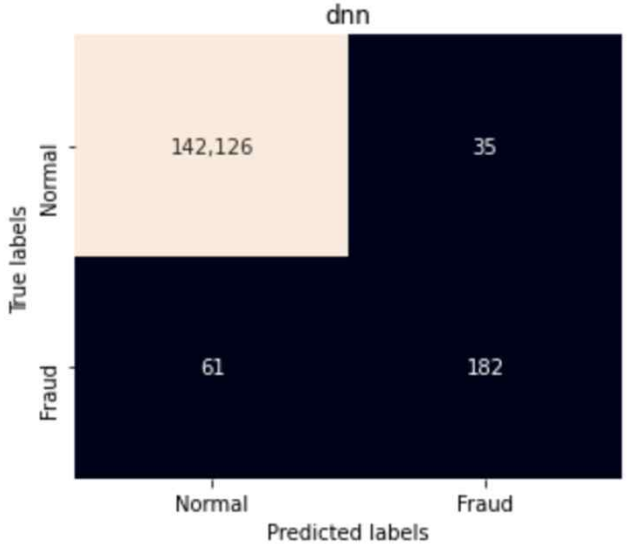
Precision 0.90
Recall 0.67



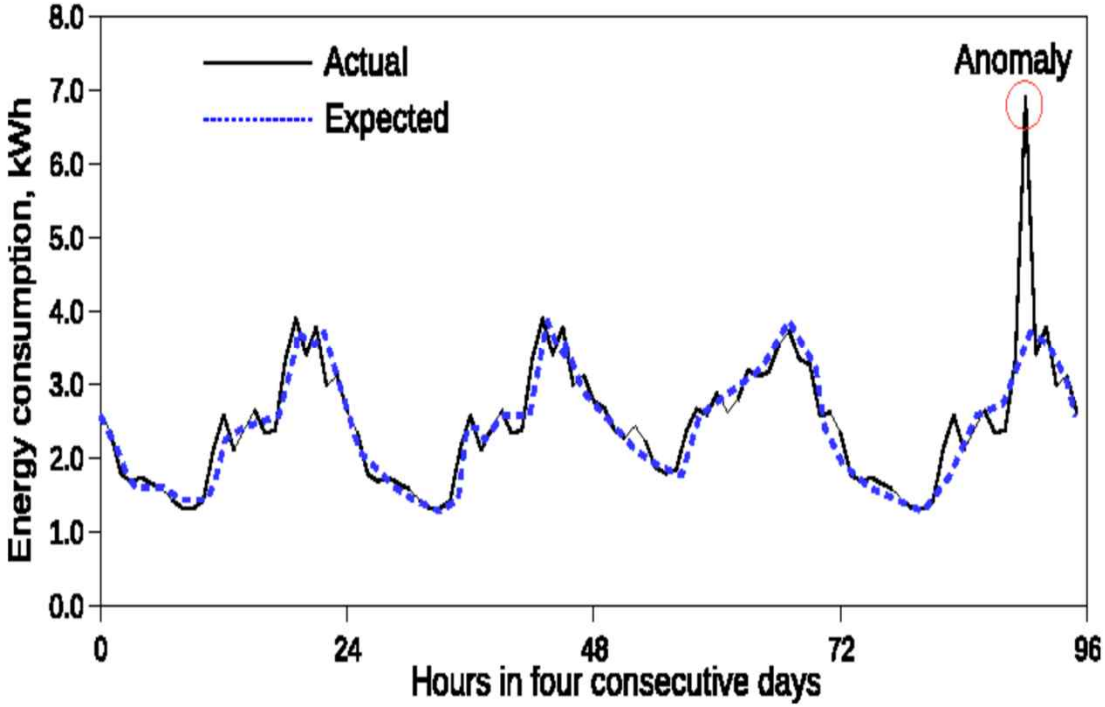
Precision 0.91
Recall 0.73



Precision 0.84
Recall 0.75



시계열 데이터의 이상치 검출



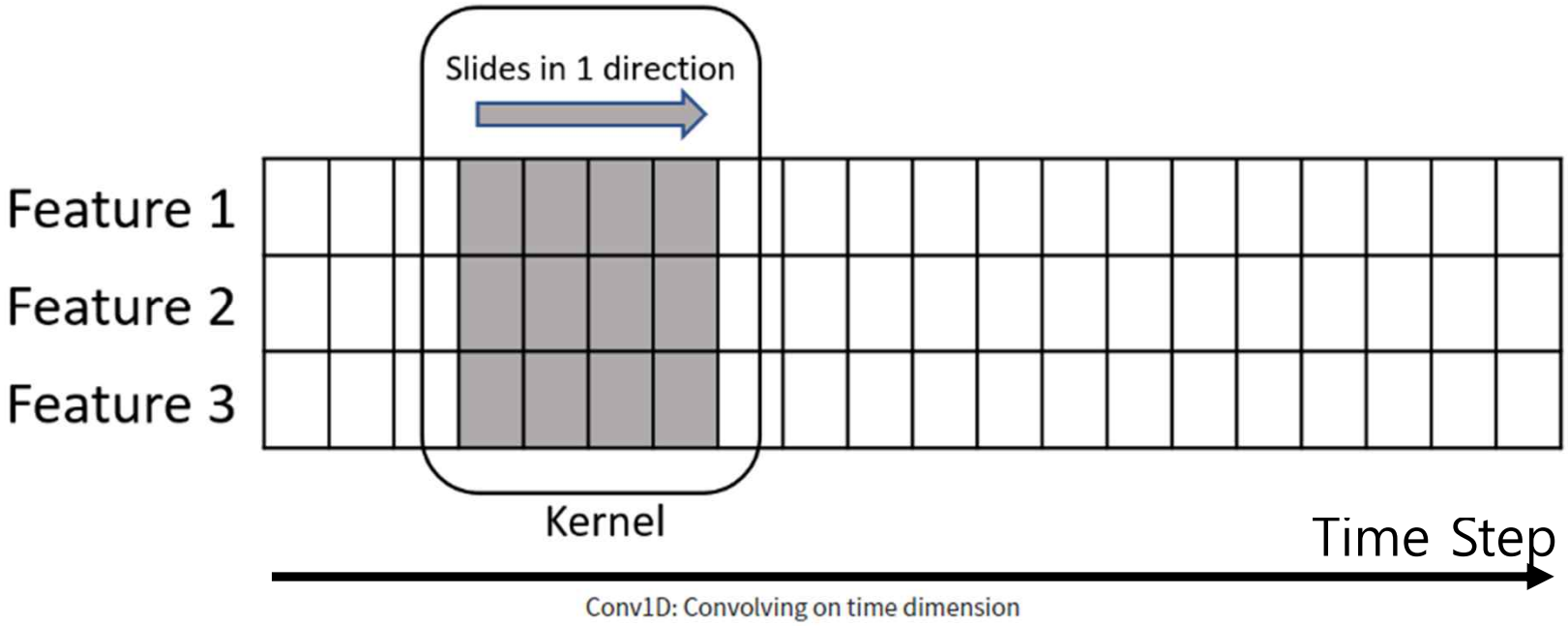
예) website의 normal한 transaction수가 예측치에 비해 크게 증가하면 service attack으로 감지

1D CNN

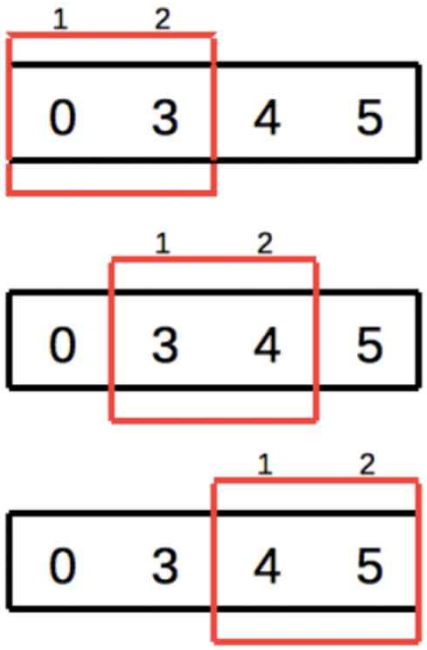
- One-direction (Time) 으로만 sliding 하므로 1D Convolution
- CNN 은 고정 길이의 data 로부터 feature 를 찾아내는 능력 탁월
- Sensor data 의 시계열 분석, audio recording 의 signal data 분석, NLP 등 **고정된 길이의 주기**(fixed length period)를 가진 데이터에 사용
- Simple pattern capture → higher layers에서 해당 feature 활용

Conv1D Convolution

Timestep 에 따라 변화 – 1 dimensional



```
tf.keras.layers.Conv1D(filters=n_filters,
                       kernel_size=2,
                       strides=1,
```



Convolution

* [1 2]

Kernel

```
[0 3] * [1 2] =>  6
[3 4] * [1 2] =>  11
[4 5] * [1 2] =>  14

array([ 6, 11, 14])
```

Extracted Feature

실습: 170. Conv1D Autoencoder를 사용한 시계열 data 이상 감지

- 비정상적인 동작 기간이 labeling 된 인공 시계열 데이터 Numenta Anomaly Benchmark(NAB) 를 사용