LG PIC
실습

# LSTM

**박석희 교수님**

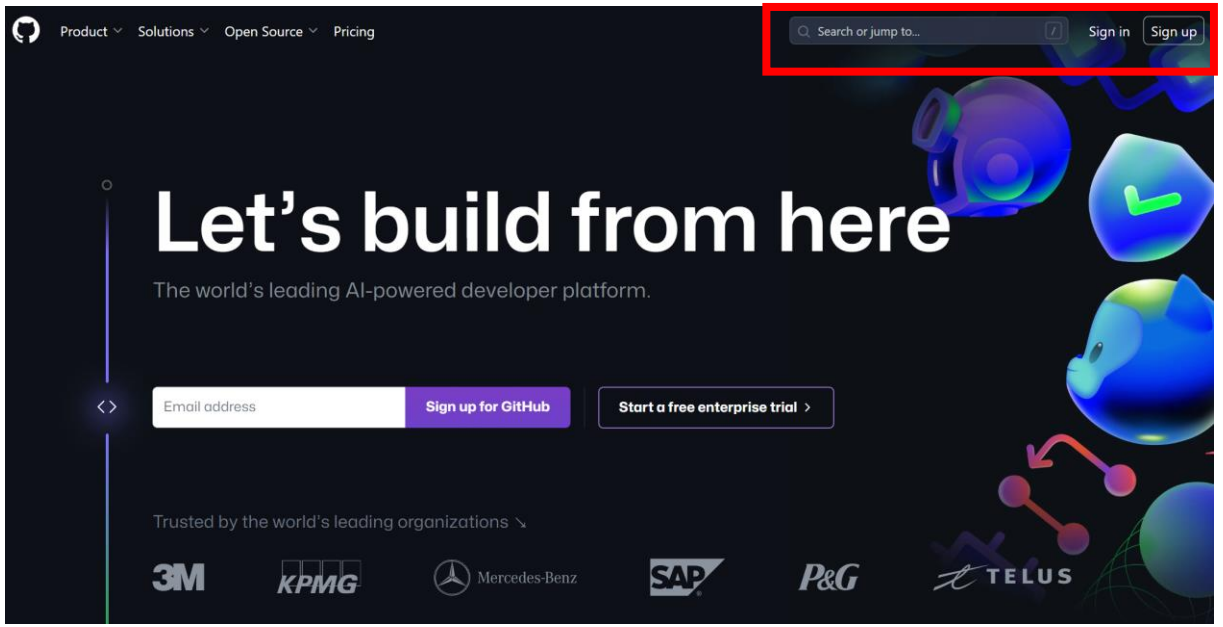
실습조교 이승문
ch273404@naver.com

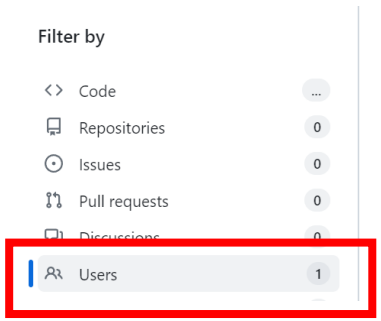
08/12(월)

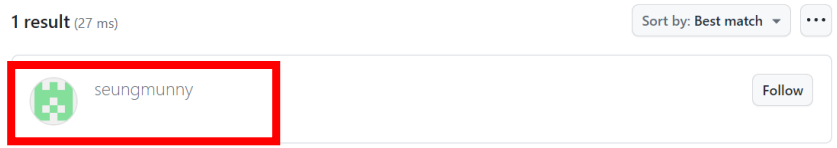
# 코드 및 데이터 다운로드

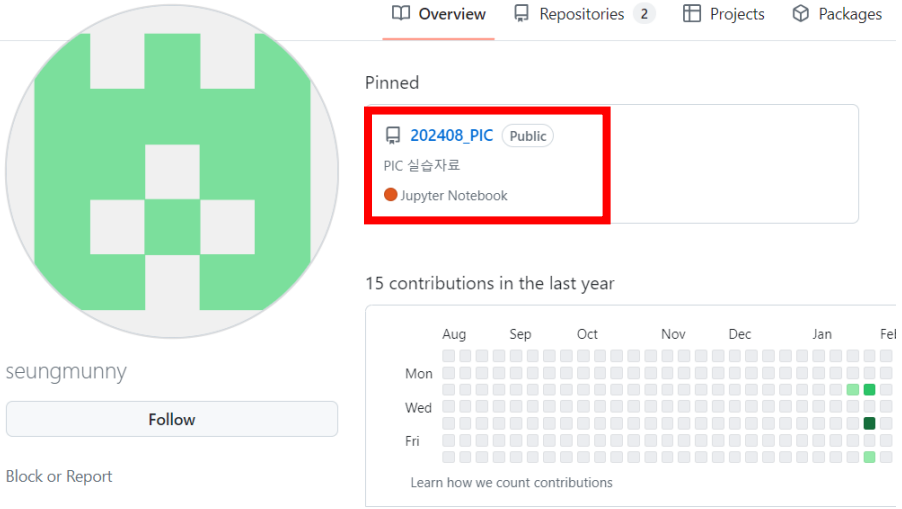
깃허브 접속

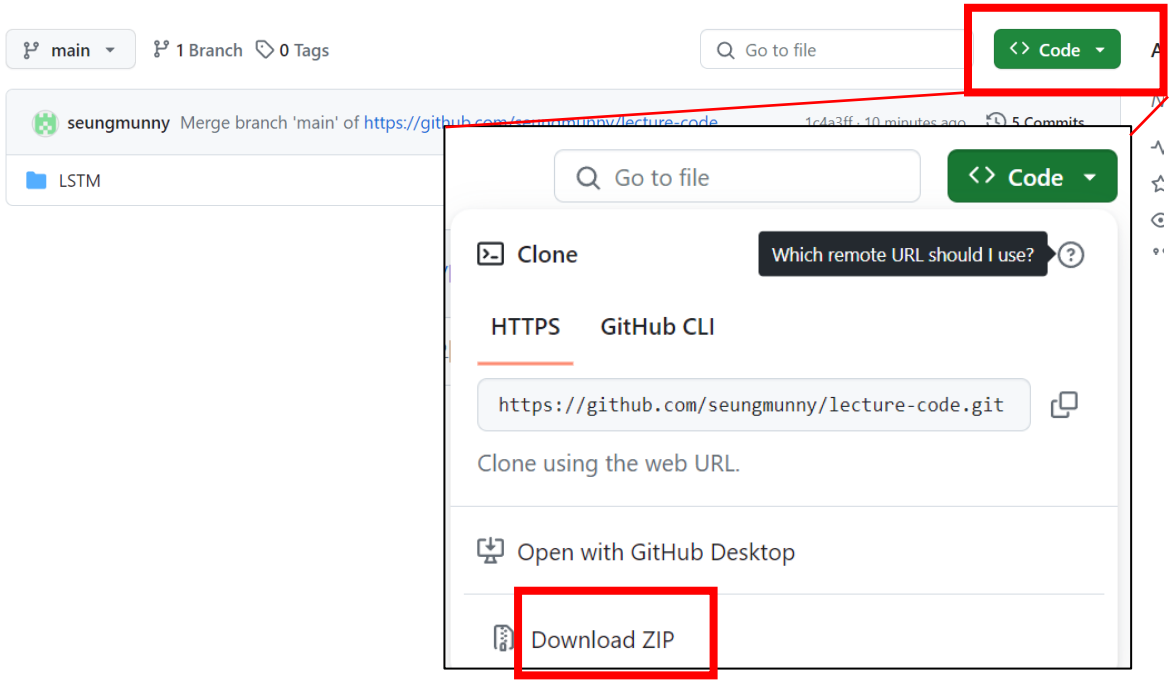
Seungmunny 입력



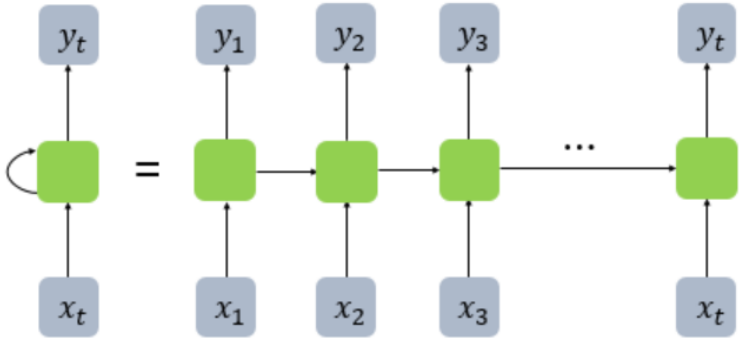



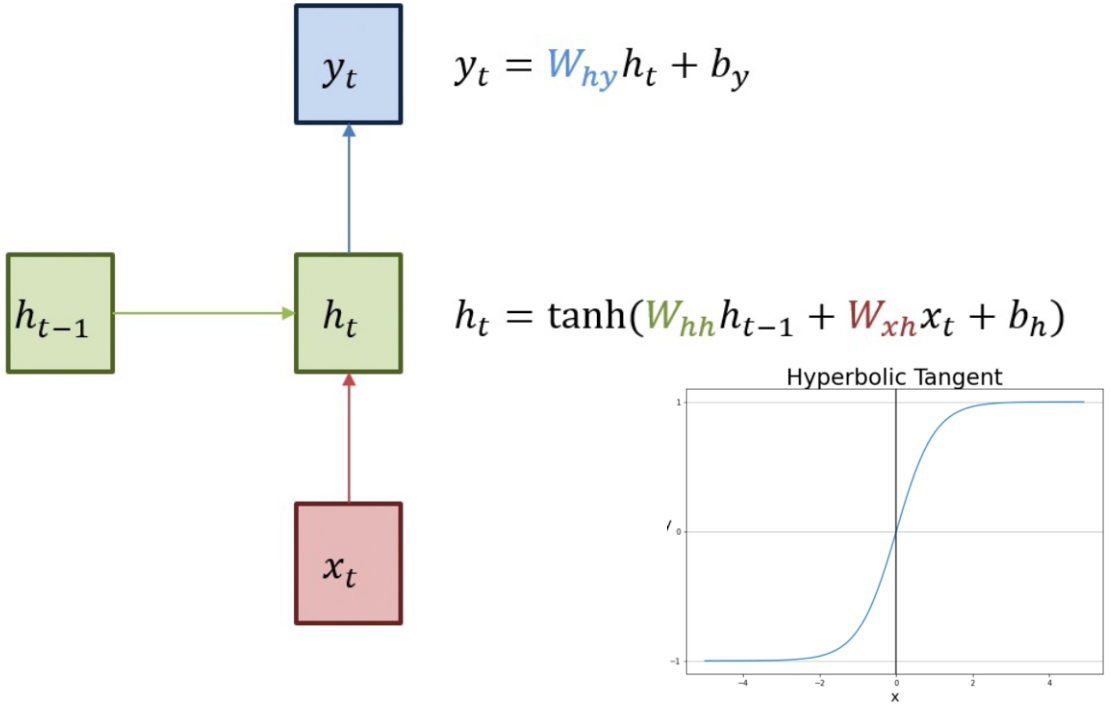



# 전통적 RNN(Recurrent Neural Network)




https://gooopy.tistory.com/54

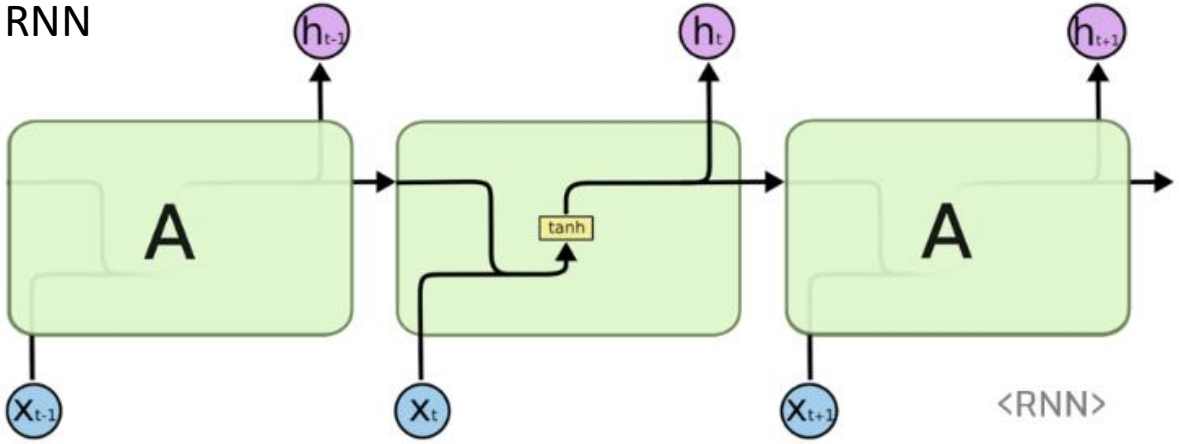
- 은닉층의 노드 결과값을 다음 은닉층의 노드에도 전달하여 이전 데이터가 다음 출력에 영향을 주는 형태
- 시계열 데이터를 다루는데 최적인 neural network
- Time step이 늘어날수록 학습 과정에서의 역전파 시 gradient가 소멸되어 파라미터 업데이트가 불가능한 vanishing gradient problem 발생-**장기 의존성 문제**
- 즉, 입력 데이터가 커질 때 데이터 뒤쪽으로 갈수록 앞쪽 데이터를 잊는다는 의미

https://velog.io/@soup1997/LSTM-Long-Short-Term-Memory

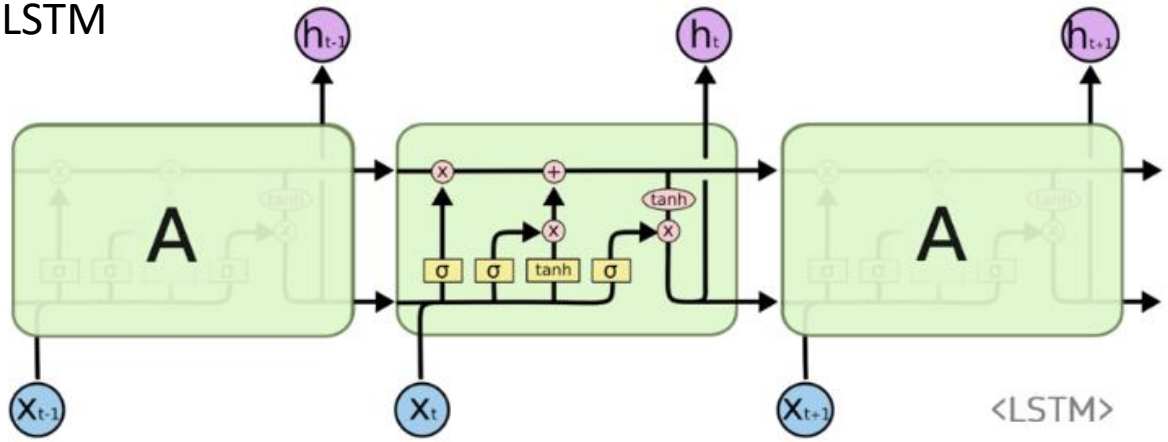
# LSTM(Long Short-Term Memory models)

- RNN의 장기의존성 문제를 해결하기 위해 hidden state에 cell state를 추가

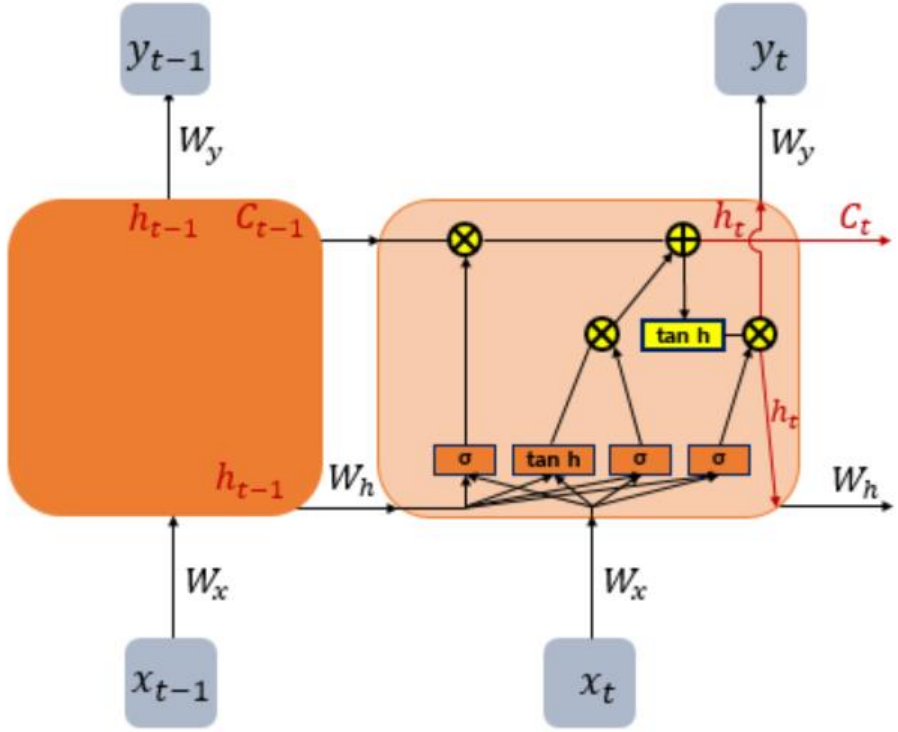
RNN




LSTM




https://ratsgo.github.io/natural%20language%20processing/2017/03/09/rnnlstm/

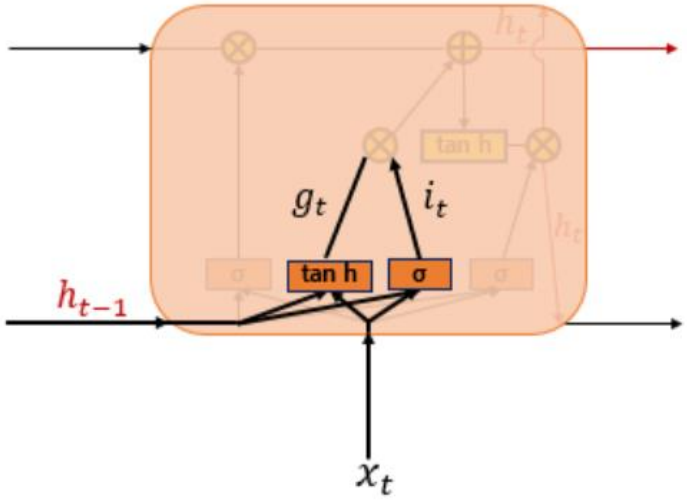



https://velog.io/@soup1997/LSTM-Long-Short-Term-Memory

# LSTM의 구조

$i_t, f_t, g_t, o_t$ are the input, forget, cell, and output gates,

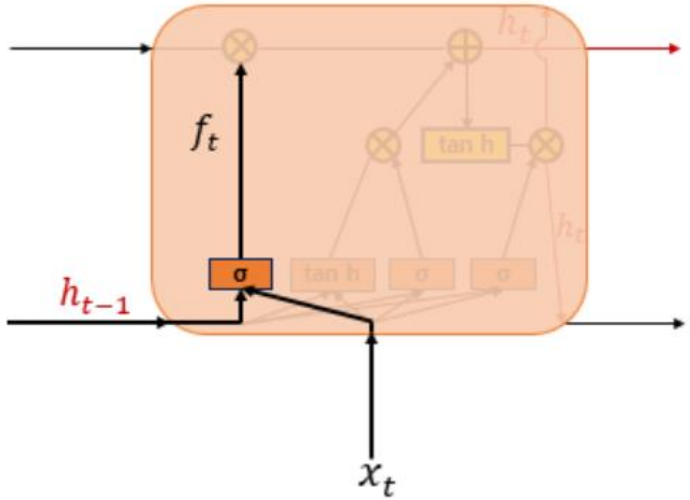
## ➢ Input gate




$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g)$$

- 현재 정보를 기억하기 위한 게이트
- σ를 통해 나온 결과인 $i_t$는 0~1 범위, tanh를 통해 나온 결과인 $g_t$ 는 -1~1 범위의 값을 가짐
- 두 값을 이용해 이번 스텝에 기억할 정보의 양을 결정

## ➢ Forget gate




$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$

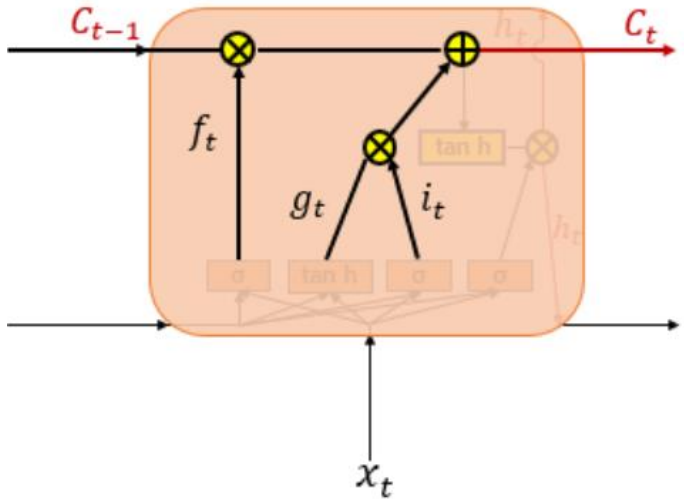
- 과거 정보를 잊기 위한 게이트
- σ를 통해 나온 결과인 $f_t$는 0~1 범위를 가지며 삭제과정을 거친 정보의 양을 뜻함
- 0은 이전 정보를 모두 잊고, 1은 이전 정보를 온전히 기억한다는 뜻

# LSTM의 구조

$i_t, f_t, g_t, o_t$ are the input, forget, cell, and output gates,

➢ **Cell state layer(장기 상태)**

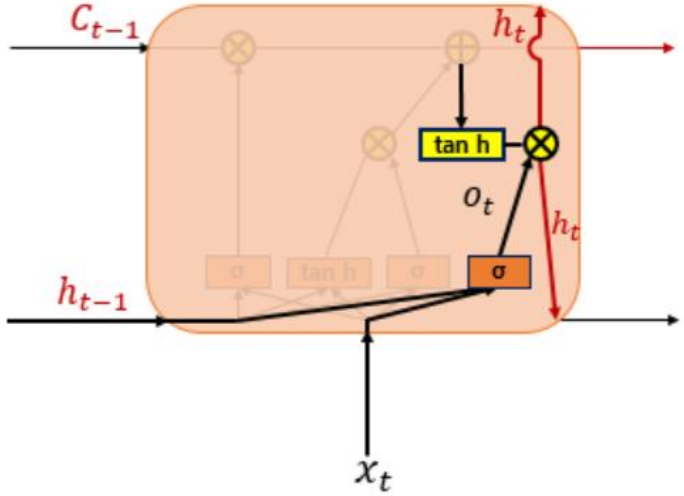
⊙는 요소별 곱셈




$$C_t = f_t \odot C_{t-1} + i_t \odot g_t$$

- 이전 cell state와 input gate와 forget gate의 값을 이용해 셀 상태 $C_t$를 구함
- 이전 시점의 셀 상태값과 입력 게이트의 결과를 얼마나 반영하는가를 결정하게 됨
- $C_t$는 다음 셀로 전달됨

➢ **Output gate layer(단기 상태)**




$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

https://velog.io/@soup1997/LSTM-Long-Short-Term-Memory

- Output gate를 통해 계산된 $h_t$는 출력층의 $y_t$를 구하는데 사용됨
- $h_t$는 다음 셀로 전달됨

# 실습

## 날씨 데이터 예측

# Miniforge jupyter notebook 실행

**가상환경 만들기**
envname에 원하는 이름 입력

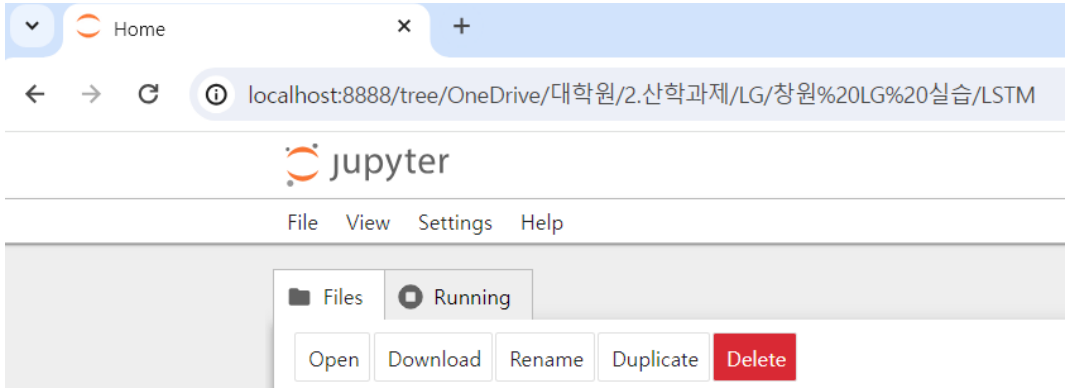
```
(base) C:\Users\USER>mamba create -n envname python=3.9.12
```

**Jupyter notebook 실행**

```
(LGPIC2408) C:\Users\USER>pip install jupyter
```

**Miniforge prompt에서 실행**
mamba install pytorch==2.1.1 torchvision==0.16.1
torchaudio==2.1.1 -c pytorch

```
(base) C:\Users\USER>mamba activate LGPIC2408
```




**라이브러리 install**

```
!pip install numpy
!pip install -U matplotlib
!pip install pandas
!pip install pyarrow
!pip install seaborn
!pip install tensorflow
!pip install scikit-learn
```

# 초기 설정 및 데이터 다운로드

## 라이브러리 import

```python
import os
import datetime

import IPython
import IPython.display
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler
import torch
from torch.utils.data import TensorDataset # 텐서데이터셋
from torch.utils.data import DataLoader # 데이터로더

mpl.rcParams['figure.figsize'] = (8, 6)
mpl.rcParams['axes.grid'] = False
```

## 데이터 불러오기

```python
df = pd.read_csv("dataset/jena_climate_2009_2016.csv")
# Slice [start:stop:step], starting from index 5 take every 6th record.
df = df[5::6]
```

### 날씨 데이터세트


이 튜토리얼은 막스 플랑크 생물 지구화학 연구소에서 기록한 날씨 시계열 데이터세트를 사용합니다.

이 데이터세트에는 온도, 대기압 및 습도와 같은 14가지 특성이 있습니다. 이러한 데이터는 2003년부터 시작해 10분 간격으로 수집되었습니다. 효율성을 위해 2009년과 2016년 사이에 수집된 데이터만 사용하겠습니다. 이 데이터세트 부분은 François Chollet이 자신이 저술한 책 Deep Learning with Python을 위해 준비했습니다.

| | p (mbar) | T (degC) | Tpot (K) | Tdew (degC) | rh (%) | VPmax (mbar) | VPact (mbar) | VPdef (mbar) | sh (g/kg) | H2OC (mmol/mol) | rho (g/m**3) | wv (m/s) | max. wv (m/s) | wd (deg) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 996.50 | -8.05 | 265.38 | -8.78 | 94.4 | 3.33 | 3.14 | 0.19 | 1.96 | 3.15 | 1307.86 | 0.21 | 0.63 | 192.7 |
| 11 | 996.62 | -8.88 | 264.54 | -9.77 | 93.2 | 3.12 | 2.90 | 0.21 | 1.81 | 2.91 | 1312.25 | 0.25 | 0.63 | 190.3 |
| 17 | 996.84 | -8.81 | 264.59 | -9.66 | 93.5 | 3.13 | 2.93 | 0.20 | 1.83 | 2.94 | 1312.18 | 0.18 | 0.63 | 167.2 |
| 23 | 996.99 | -9.05 | 264.34 | -10.02 | 92.6 | 3.07 | 2.85 | 0.23 | 1.78 | 2.85 | 1313.61 | 0.10 | 0.38 | 240.0 |
| 29 | 997.46 | -9.63 | 263.72 | -10.65 | 92.2 | 2.94 | 2.71 | 0.23 | 1.69 | 2.71 | 1317.19 | 0.40 | 0.88 | 157.0 |

https://www.tensorflow.org/tutorials/structured_data/time_series?hl=ko

# 데이터 정리

```
df.head()
```

| | Date Time | p (mbar) | T (degC) | Tpot (K) | Tdew (degC) | rh (%) | VPmax (mbar) | VPact (mbar) | VPdef (mbar) | sh (g/kg) | H2OC (mmol/mol) | rho (g/m**3) | wv (m/s) | max. wv (m/s) | wd (deg) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **5** | 01.01.2009 01:00:00 | 996.50 | -8.05 | 265.38 | -8.78 | 94.4 | 3.33 | 3.14 | 0.19 | 1.96 | 3.15 | 1307.86 | 0.21 | 0.63 | 192.7 |
| **11** | 01.01.2009 02:00:00 | 996.62 | -8.88 | 264.54 | -9.77 | 93.2 | 3.12 | 2.90 | 0.21 | 1.81 | 2.91 | 1312.25 | 0.25 | 0.63 | 190.3 |
| **17** | 01.01.2009 03:00:00 | 996.84 | -8.81 | 264.59 | -9.66 | 93.5 | 3.13 | 2.93 | 0.20 | 1.83 | 2.94 | 1312.18 | 0.18 | 0.63 | 167.2 |
| **23** | 01.01.2009 04:00:00 | 996.99 | -9.05 | 264.34 | -10.02 | 92.6 | 3.07 | 2.85 | 0.23 | 1.78 | 2.85 | 1313.61 | 0.10 | 0.38 | 240.0 |
| **29** | 01.01.2009 05:00:00 | 997.46 | -9.63 | 263.72 | -10.65 | 92.2 | 2.94 | 2.71 | 0.23 | 1.69 | 2.71 | 1317.19 | 0.40 | 0.88 | 157.0 |

```
date_time = pd.to_datetime(df.pop('Date Time'), format='%d.%m.%Y %H:%M:%S')
df=df.drop(['Tpot (K)','Tdew (degC)','VPmax (mbar)','VPact (mbar)','VPdef (mbar)','sh (g/kg)','H2OC (mmol/mol)','rho (g/m**3)','max. wv (m/s)','wd (deg)'], axis=1)
df = df[['p (mbar)', 'rh (%)', 'wv (m/s)','T (degC)' ]]
```

```
df.head()
```

| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| **5** | 996.50 | 94.4 | 0.21 | -8.05 |
| **11** | 996.62 | 93.2 | 0.25 | -8.88 |
| **17** | 996.84 | 93.5 | 0.18 | -8.81 |
| **23** | 996.99 | 92.6 | 0.10 | -9.05 |
| **29** | 997.46 | 92.2 | 0.40 | -9.63 |

시간 정보는 따로 저장
**압력과 상대습도, 바람 속도, 온도** 데이터만 사용
온도 데이터를 예측하기 위해 온도 데이터를 맨 오른쪽으로 이동

# 데이터 정리

```
1 df.describe()
```

| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| **count** | 70091.000000 | 70091.000000 | 70091.000000 | 70091.000000 |
| **mean** | 989.212842 | 76.009788 | 1.702567 | 9.450482 |
| **std** | 8.358886 | 16.474920 | 65.447512 | 8.423384 |
| **min** | 913.600000 | 13.880000 | -9999.000000 | -22.760000 |
| **25%** | 984.200000 | 65.210000 | 0.990000 | 3.350000 |
| **50%** | 989.570000 | 79.300000 | 1.760000 | 9.410000 |
| **75%** | 994.720000 | 89.400000 | 2.860000 | 15.480000 |
| **max** | 1015.290000 | 100.000000 | 14.010000 | 37.280000 |

- 바람 속도에 오류값 존재

```
1 wv = df['wv (m/s)']
2 bad_wv = wv == -9999.0
3 wv[bad_wv] = 0.0
```

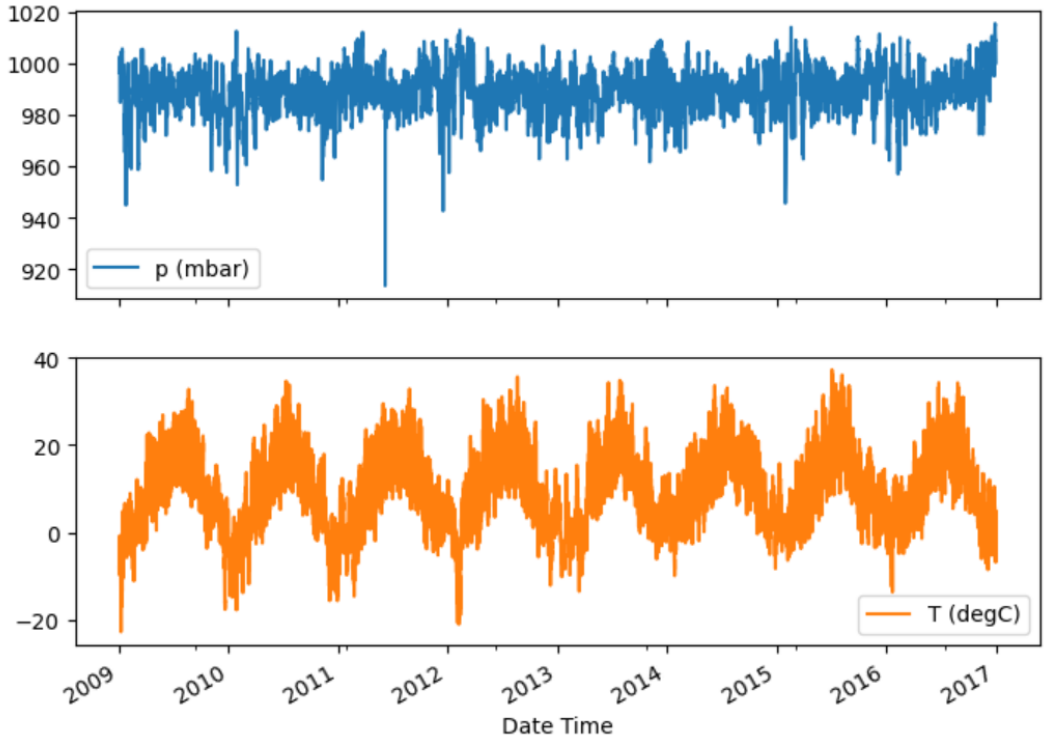
```
1 df.describe()
```

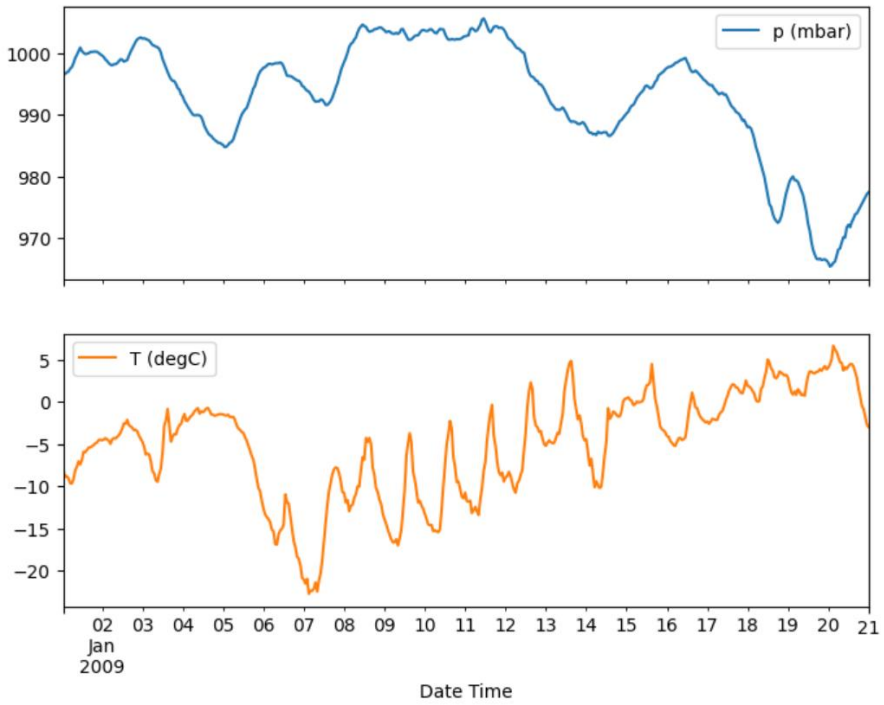
| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| **count** | 70091.000000 | 70091.000000 | 70091.000000 | 70091.000000 |
| **mean** | 989.212842 | 76.009788 | 2.130539 | 9.450482 |
| **std** | 8.358886 | 16.474920 | 1.543098 | 8.423384 |
| **min** | 913.600000 | 13.880000 | 0.000000 | -22.760000 |
| **25%** | 984.200000 | 65.210000 | 0.990000 | 3.350000 |
| **50%** | 989.570000 | 79.300000 | 1.760000 | 9.410000 |
| **75%** | 994.720000 | 89.400000 | 2.860000 | 15.480000 |
| **max** | 1015.290000 | 100.000000 | 14.010000 | 37.280000 |

- -9999.0 오류값을 0으로 대체

# 데이터 확인

```
plot_cols = ['p (mbar)','T (degC)']
plot_features = df[plot_cols]
plot_features.index = date_time
_ = plot_features.plot(subplots=True)

plot_features = df[plot_cols][:480]
plot_features.index = date_time[:480]
_ = plot_features.plot(subplots=True)
```



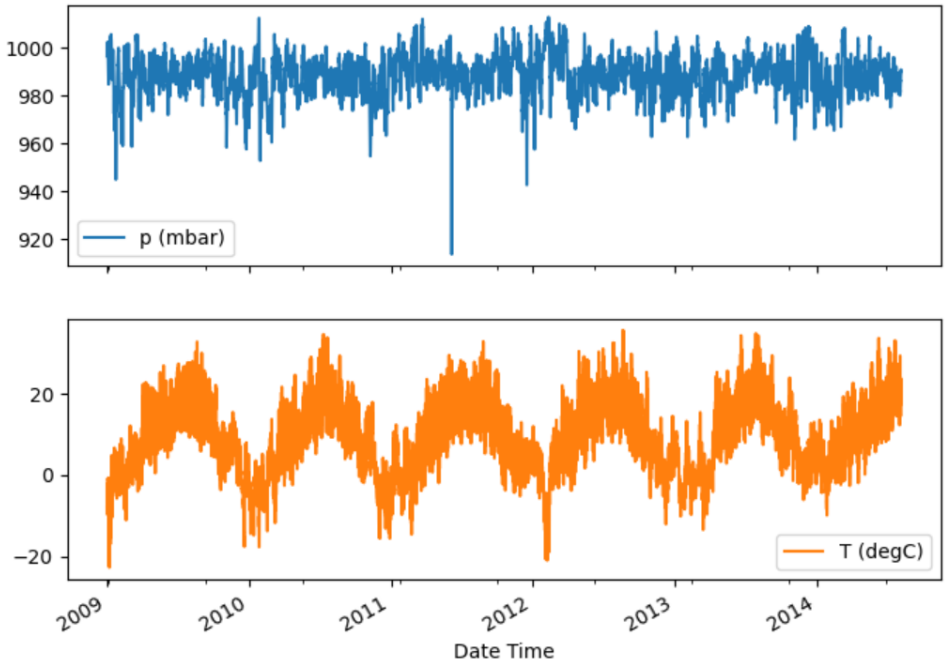



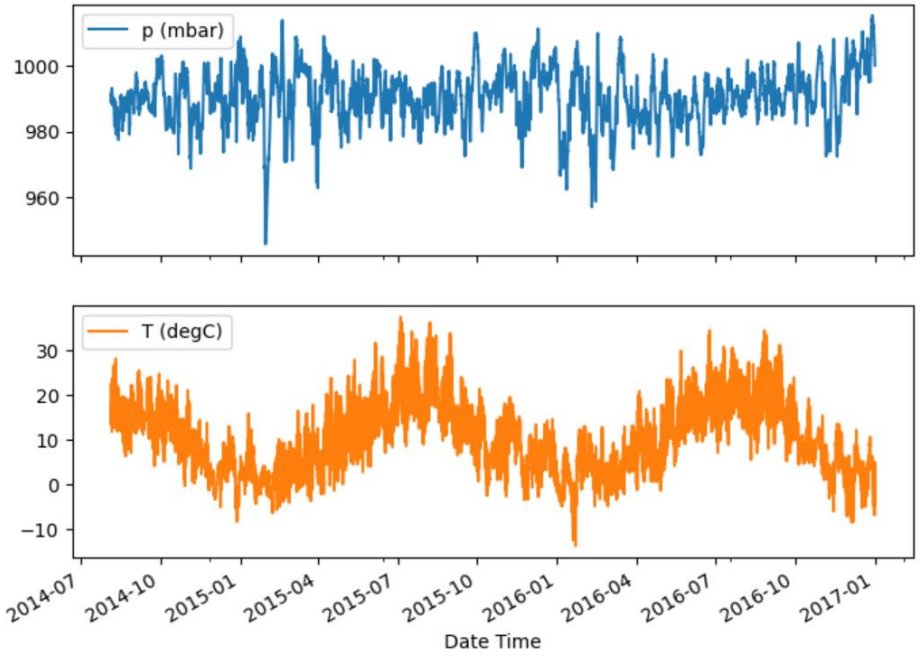
# Train, test 데이터 분할

실습

```python
# 7일간의 데이터가 입력으로 들어가고 batch size는 임의로 지정
seq_length = 7
batch = 100
train_size = int(len(df)*0.7)
train_set = df[0:train_size]
train_date_time=date_time[0:train_size]
test_set = df[train_size-seq_length:]
test_date_time=date_time[train_size-seq_length:]
```

```python
plot_cols = ['p (mbar)','T (degC)']
plot_features = train_set[plot_cols]
plot_features.index = train_date_time
_ = plot_features.plot(subplots=True)

plot_features = test_set[plot_cols]
plot_features.index = test_date_time
_ = plot_features.plot(subplots=True)
```



# 데이터 스케일링

실습

```
# Input scale
scaler_x = MinMaxScaler()
scaler_x.fit(train_set.iloc[:, :-1])

train_set.iloc[:, :-1] = scaler_x.transform(train_set.iloc[:, :-1])
test_set.iloc[:, :-1] = scaler_x.transform(test_set.iloc[:, :-1])

# Output scale
scaler_y = MinMaxScaler()
scaler_y.fit(train_set.iloc[:, [-1]])

train_set.iloc[:, -1] = scaler_y.transform(train_set.iloc[:, [-1]])
test_set.iloc[:, -1] = scaler_y.transform(test_set.iloc[:, [-1]])
```

| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| **5** | 996.50 | 94.4 | 0.21 | -8.05 |
| **11** | 996.62 | 93.2 | 0.25 | -8.88 |
| **17** | 996.84 | 93.5 | 0.18 | -8.81 |
| **23** | 996.99 | 92.6 | 0.10 | -9.05 |
| **29** | 997.46 | 92.2 | 0.40 | -9.63 |

변수값들의 크기가 제각각이므로
0-1사이 값으로 스케일링

```
train_set.head()
```

| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| **5** | 0.835433 | 0.934974 | 0.014989 | 0.251840 |
| **11** | 0.836642 | 0.921040 | 0.017844 | 0.237631 |
| **17** | 0.838859 | 0.924524 | 0.012848 | 0.238829 |
| **23** | 0.840371 | 0.914073 | 0.007138 | 0.234720 |
| **29** | 0.845107 | 0.909429 | 0.028551 | 0.224790 |

# 데이터셋 생성

```
device = torch.device('cpu')
# 데이터셋 생성 함수
def build_dataset(time_series, seq_length):
    dataX = []
    dataY = []
    for i in range(0, len(time_series)-seq_length):
        _x = time_series[i:i+seq_length, :]
        _y = time_series[i+seq_length, [-1]]
        # print(_x, "-->",_y)
        dataX.append(_x)
        dataY.append(_y)

    return np.array(dataX), np.array(dataY)

trainX, trainY = build_dataset(np.array(train_set), seq_length)
testX, testY = build_dataset(np.array(test_set), seq_length)

# 텐서로 변환
trainX_tensor = torch.FloatTensor(trainX)
trainY_tensor = torch.FloatTensor(trainY)

testX_tensor = torch.FloatTensor(testX)
testY_tensor = torch.FloatTensor(testY)
testX_tensor = testX_tensor.to(device)
testY_tensor = testY_tensor.to(device)
# 텐서 형태로 데이터 정의
dataset = TensorDataset(trainX_tensor, trainY_tensor)
# 데이터로더는 기본적으로 2개의 인자를 입력받으며 배치크기는 통상적으로 2의 배수를 사용
dataloader = DataLoader(dataset,
                        batch_size=batch,
                        shuffle=True,
                        drop_last=True)
```

```
a=[1,2,3,4,5,6]
```

```
a[0:3]
```

```
[1, 2, 3]
```

```
a[3]
```

```
4
```

```
# 7개의 데이터가 입력으로 들어가고 batch size는 임의로 지정
seq_length = 7
batch = 100
```

실습

| | p (mbar) | rh (%) | wv (m/s) | T (degC) | |
|---|---|---|---|---|---|
| 1 | 0.835433 | 0.934974 | 0.014989 | 0.251840 | Input, x1 |
| 2 | 0.836642 | 0.921040 | 0.017844 | 0.237631 | |
| 3 | 0.838859 | 0.924524 | 0.012848 | 0.238829 | |
| 4 | 0.840371 | 0.914073 | 0.007138 | 0.234720 | |
| 5 | 0.845107 | 0.909429 | 0.028551 | 0.224790 | |
| 6 | 0.836642 | 0.921040 | 0.017844 | 0.237631 | |
| 7 | 0.838859 | 0.924524 | 0.012848 | 0.238829 | |
| 8 | 0.840371 | 0.914073 | 0.007138 | 0.234720 | Output, y1 |
| 9 | 0.845107 | 0.909429 | 0.028551 | 0.224790 | |

# 데이터셋 생성

```
1
2 device = torch.device('cpu')
3 # 데이터셋 생성 함수
4 def build_dataset(time_series, seq_length):
5     dataX = []
6     dataY = []
7     for i in range(0, len(time_series)-seq_length):
8         _x = time_series[i:i+seq_length, :]
9         _y = time_series[i+seq_length, [-1]]
10        # print(_x, "-->",_y)
11        dataX.append(_x)
12        dataY.append(_y)
13
14    return np.array(dataX), np.array(dataY)
15
16 trainX, trainY = build_dataset(np.array(train_set), seq_length)
17 testX, testY = build_dataset(np.array(test_set), seq_length)
18
19 # 텐서로 변환
20 trainX_tensor = torch.FloatTensor(trainX)
21 trainY_tensor = torch.FloatTensor(trainY)
22
23 testX_tensor = torch.FloatTensor(testX)
24 testY_tensor = torch.FloatTensor(testY)
25 testX_tensor = testX_tensor.to(device)
26 testY_tensor = testY_tensor.to(device)
27 # 텐서 형태로 데이터 정의
28 dataset = TensorDataset(trainX_tensor, trainY_tensor)
29 # 데이터로더는 기본적으로 2개의 인자를 입력받으며 배치크기는 통상적으로 2의 배수를 사용
30 dataloader = DataLoader(dataset,
31                         batch_size=batch,
32                         shuffle=True,
33                         drop_last=True)
```

```
1 a=[1,2,3,4,5,6]
```

```
1 a[0:3]
```

```
[1, 2, 3]
```

```
1 a[3]
```

```
4
```

```
2 seq_length = 7
```

| | p (mbar) | rh (%) | wv (m/s) | T (degC) |
|---|---|---|---|---|
| 1 | 0.835433 | 0.934974 | 0.014989 | 0.251840 |
| 2 | 0.836642 | 0.921040 | 0.017844 | 0.237631 |
| 3 | 0.838859 | 0.924524 | 0.012848 | 0.238829 |
| 4 | 0.840371 | 0.914073 | 0.007138 | 0.234720 |
| 5 | 0.845107 | 0.909429 | 0.028551 | 0.224790 |
| 6 | 0.836642 | 0.921040 | 0.017844 | 0.237631 |
| 7 | 0.838859 | 0.924524 | 0.012848 | 0.238829 |
| 8 | 0.840371 | 0.914073 | 0.007138 | 0.234720 |
| 9 | 0.845107 | 0.909429 | 0.028551 | 0.224790 |

Input, x2

Output, y2

# LSTM 모델 작성

```python
# 7일간의 데이터가 입력으로 들어가고 batch size는 임의로 지정
seq_length = 7
batch = 100
```

```python
import torch.nn as nn

# 설정값
data_dim = 5
hidden_dim = 10
output_dim = 1
learning_rate = 0.01
nb_epochs = 100

class Net(nn.Module):
    # # 기본변수, layer를 초기화해주는 생성자
    def __init__(self, input_dim, hidden_dim, seq_len, output_dim, layers):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.output_dim = output_dim
        self.layers = layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=layers,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim, bias = True)

    # 학습 초기화를 위한 함수
    def reset_hidden_state(self):
        self.hidden = (
                torch.zeros(self.layers, self.seq_len, self.hidden_dim),
                torch.zeros(self.layers, self.seq_len, self.hidden_dim))

    # 예측을 위한 함수
    def forward(self, x):
        x, _status = self.lstm(x)
        x = self.fc(x[:, -1])
        return x
```

모델 변수

LSTM 구조 정의

# 모델 training 함수 작성

```
def train_model(model, train_df, num_epochs = None, lr = None, verbose = 10, patience = 10):

    criterion = nn.MSELoss().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
    nb_epochs = num_epochs

    # epoch마다 loss 저장
    train_hist = np.zeros(nb_epochs)

    for epoch in range(nb_epochs):
        avg_cost = 0
        total_batch = len(train_df)

        for batch_idx, samples in enumerate(train_df):

            x_train, y_train = samples

            # seq별 hidden state reset
            model.reset_hidden_state()

            # H(x) 계산
            outputs = model(x_train)

            # cost 계산
            loss = criterion(outputs, y_train)

            # cost로 H(x) 개선
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            avg_cost += loss/total_batch

        train_hist[epoch] = avg_cost

        if epoch % verbose == 0:
            print('Epoch:', '%04d' % (epoch), 'train loss :', '{:.4f}'.format(avg_cost))

        # patience번째 마다 early stopping 여부 확인
        if (epoch % patience == 0) & (epoch != 0):

            # loss가 커졌다면 early stop
            if train_hist[epoch-patience] < train_hist[epoch]:
                print('₩n Early Stopping')

                break

    return model.eval(), train_hist
```

```
for batch_idx, samples in enumerate(train_df):
```

```
>>> for i, letter in enumerate(['A', 'B', 'C']):
...     print(i, letter)
...
0 A
1 B
2 C
```

```
[tensor([[[0.8354, 0.9350, 0.0150, 0.2518],
          [0.8366, 0.9210, 0.0178, 0.2376],
          [0.8389, 0.9245, 0.0128, 0.2388],
          [0.8404, 0.9141, 0.0071, 0.2347],
          [0.8451, 0.9094, 0.0286, 0.2248],
          [0.8476, 0.9152, 0.0036, 0.2241],
          [0.8539, 0.9176, 0.1485, 0.2327]],

         [[0.8366, 0.9210, 0.0178, 0.2376],
          [0.8389, 0.9245, 0.0128, 0.2388],
          [0.8404, 0.9141, 0.0071, 0.2347],
          [0.8451, 0.9094, 0.0286, 0.2248],
          [0.8476, 0.9152, 0.0036, 0.2241],
          [0.8539, 0.9176, 0.1485, 0.2327],
          [0.8623, 0.9164, 0.0514, 0.2510]]]),
 tensor([[0.2510],
         [0.2585]])]
```

# 모델 학습

```
# 모델 학습
# 설정값
data_dim = 5
output_dim = 1
hidden_dim = 10
learning_rate = 0.01
nb_epochs = 100
net = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 20, patience = 10)
```

```
class Net(nn.Module):
    # # 기본변수, layer를 초기화해주는 생성자
    def __init__(self, input_dim, hidden_dim, seq_len, output_dim, layers):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.output_dim = output_dim
        self.layers = layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=layers,
                            # dropout = 0.1,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim, bias = True)

    # 학습 초기화를 위한 함수
    def reset_hidden_state(self):
        self.hidden = (
                torch.zeros(self.layers, self.seq_len, self.hidden_dim),
                torch.zeros(self.layers, self.seq_len, self.hidden_dim))

    # 예측을 위한 함수
    def forward(self, x):
        x, _status = self.lstm(x)
        x = self.fc(x[:, -1])
        return x
```

```
def train_model(model, train_df, num_epochs = None, lr = None, verbose = 10, patience = 10):

    criterion = nn.MSELoss().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
    nb_epochs = num_epochs

    # epoch마다 loss 저장
    train_hist = np.zeros(nb_epochs)

    for epoch in range(nb_epochs):
        avg_cost = 0
        total_batch = len(train_df)

        for batch_idx, samples in enumerate(train_df):

            x_train, y_train = samples

            # seq별 hidden state reset
            model.reset_hidden_state()

            # H(x) 계산
            outputs = model(x_train)

            # cost 계산
            loss = criterion(outputs, y_train)

            # cost로 H(x) 개선
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            avg_cost += loss/total_batch

        train_hist[epoch] = avg_cost
```

# 모델 학습

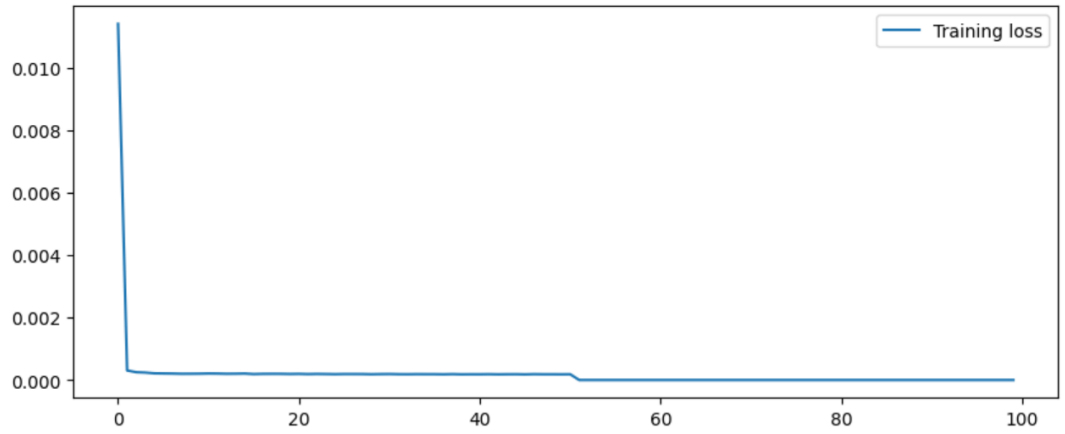
실습

```
# 모델 학습
# 설정값
data_dim = 5
output_dim = 1
hidden_dim = 10
learning_rate = 0.01
nb_epochs = 100
net = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 20, patience = 10)
```

자유롭게 설정 가능

```
Epoch: 0000 train loss : 0.0159
Epoch: 0020 train loss : 0.0002
Epoch: 0040 train loss : 0.0002

Early Stopping : 0040 epoch
```

```
1 # epoch별 손실값
2 fig = plt.figure(figsize=(10, 4))
3 plt.plot(train_hist, label="Training loss")
4 plt.legend()
5 plt.show()
```



# 모델 save & load

실습

```
# 모델 저장
PATH ="model1/model1.pth"
torch.save(model.state_dict(), PATH)

# 불러오기
model = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model.load_state_dict(torch.load(PATH), strict=False)
model.eval()
```

```
Net(
  (lstm): LSTM(5, 10, batch_first=True)
  (fc): Linear(in_features=10, out_features=1, bias=True)
)
```

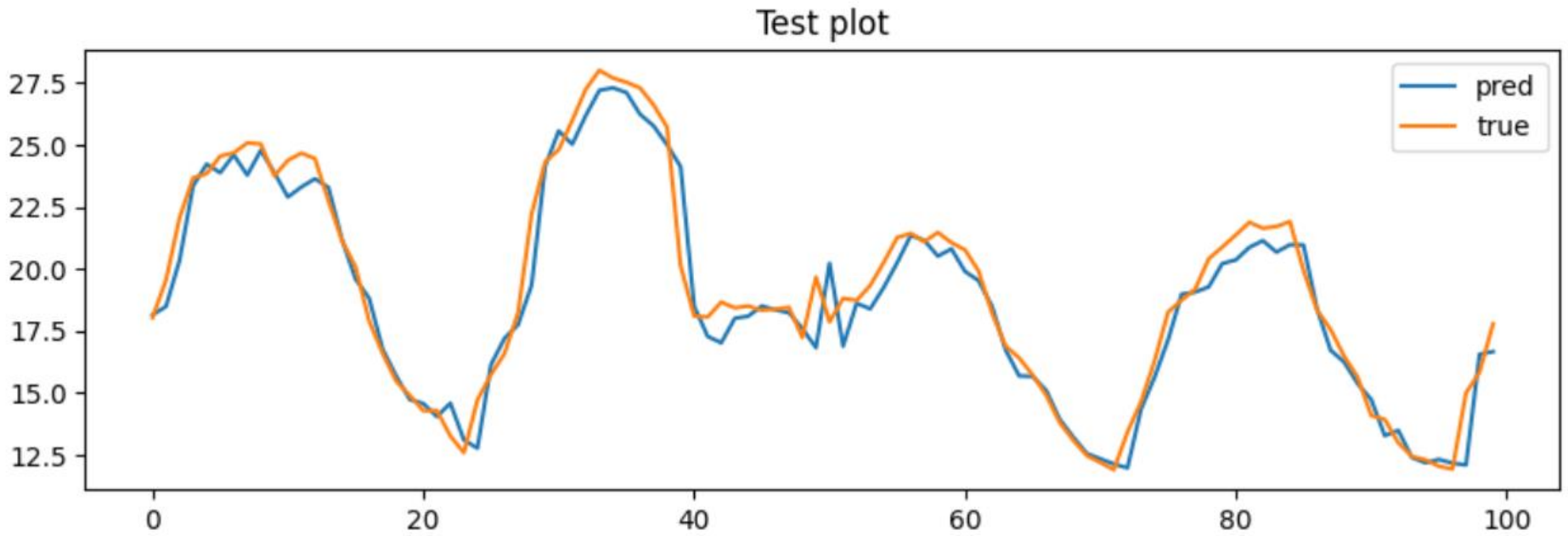
# 모델 테스트

실습

```
# 예측 테스트
a=0
testX_tensor_100=testX_tensor[a:a+100]
testY_tensor_100=testY_tensor[a:a+100]
with torch.no_grad():
    pred = []
    for pr in range(len(testX_tensor_100)):

        model.reset_hidden_state()

        predicted = model(torch.unsqueeze(testX_tensor_100[pr], 0))
        predicted = torch.flatten(predicted).item()
        pred.append(predicted)

    # INVERSE
    pred_inverse = scaler_y.inverse_transform(np.array(pred).reshape(-1, 1))
    testY_inverse = scaler_y.inverse_transform(testY_tensor_100)
```

# 모델 테스트

```
1 fig = plt.figure(figsize=(10,3))
2 plt.plot(np.arange(len(pred_inverse)), pred_inverse, label = 'pred')
3 plt.plot(np.arange(len(testY_inverse)), testY_inverse, label = 'true')
4 plt.title("Test plot")
5 plt.legend()
6 plt.show()
```



| Index | Features | Format | Description |
|---|---|---|---|
| 1 | Date Time | 01.01.2009 00:10:00 | Date-time reference |
| 2 | p (mbar) | 996.52 | 내부 압력을 정량화하는 데 사용되는 파스칼 SI 유도 압력 단위. 기상 보고서는 일반적으로 대기압을 밀리바 단위로 나타낸다.<br>The pascal SI derived unit of pressure used to quantify internal pressure. Meteorological reports typically state atmospheric pressure in millibars. |
| 3 | T (degC) | -8.02 | Temperature in Celsius |
| 4 | Tpot (K) | 265.4 | Temperature in Kelvin |
| 5 | Tdew (degC) | -8.9 | 습도에 대한 온도(섭씨) 이슬점은 공기 중 물의 절대량을 측정하는 것입니다. DP는 공기가 그 안에 있는 모든 수분을 담을 수 없고 물이 응축되는 온도입니다.<br>Temperature in Celsius relative to humidity. Dew Point is a measure of the absolute amount of water in the air, the DP is the temperature at which the air cannot hold all the moisture in it and water condenses. |
| 6 | rh (%) | 93.3 | 상대 습도는 공기가 수증기로 얼마나 포화 상태인지를 측정하는 측도이며, %RH는 수집 물체 내에 포함된 물의 양을 결정합니다.<br>Relative Humidity is a measure of how saturated the air is with water vapor, the %RH determines the amount of water contained within collection objects. |
| 7 | VPmax (mbar) | 3.33 | 포화증기압<br>Saturation vapor pressure |
| 8 | VPact (mbar) | 3.11 | 증기압<br>Vapor pressure |
| 9 | VPdef (mbar) | 0.22 | Vapor pressure deficit |
| 10 | sh (g/kg) | 1.94 | Specific humidity |
| 11 | H2OC (mmol/mol) | 3.12 | Water vapor concentration |
| 12 | rho (g/m ** 3) | 1307.75 | Airtight |
| 13 | wv (m/s) | 1.03 | Wind speed |
| 14 | max. wv (m/s) | 1.75 | Maximum wind speed |
| 15 | wd (deg) | 152.3 | Wind direction in degrees |

https://deep-deep-deep.tistory.com/60

# 실습

## LSTM 기반
## 워터펌프 고장 예측

# 데이터 읽어오기

## 라이브러리 import

```
1 import numpy as np # linear algebra
2 import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
3 from sklearn.preprocessing import MinMaxScaler
4 import os
5 import torch
6 import torch.nn as nn
7 from torch.utils.data import TensorDataset # 텐서데이터셋
8 from torch.utils.data import DataLoader # 데이터로더
9 import matplotlib.pyplot as plt
```

## 데이터 읽어오기

```
df=pd.read_csv("dataset/sensor_LSTM.csv")
df
```

UNKNOWNCLASS · UPDATED 5 YEARS AGO

**pump_sensor_data**

Pump sensor data for predictive maintenance

**Content**

The data are from all available sensor, all of them are raw value. Total sensor are 52 unit.

HUYNH DONG NGUYEN · 3Y AGO · 26,616 VIEWS

**Water pump maintenance: shutdown prediction**

Python · pump_sensor_data

https://www.kaggle.com/datasets/nphantawee/pump-sensor-data/data

https://www.kaggle.com/code/winternguyen/water-pump-maintenance-shutdown-prediction

# 데이터 확인

```
1 data.head()
```

| | Unnamed: 0 | timestamp | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 | machine_status |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 2018-04-01 0:00 | 634.3750 | 13.41146 | 16.13136 | 15.56713 | 15.05353 | NORMAL |
| 1 | 1 | 2018-04-01 0:01 | 634.3750 | 13.41146 | 16.13136 | 15.56713 | 15.05353 | NORMAL |
| 2 | 2 | 2018-04-01 0:02 | 638.8889 | 13.32465 | 16.03733 | 15.61777 | 15.01013 | NORMAL |
| 3 | 3 | 2018-04-01 0:03 | 628.1250 | 13.31742 | 16.24711 | 15.69734 | 15.08247 | NORMAL |
| 4 | 4 | 2018-04-01 0:04 | 636.4583 | 13.35359 | 16.21094 | 15.69734 | 15.08247 | NORMAL |

맨 앞 5개 데이터 확인

센서 5개 데이터와
기계상태 데이터

```
1 data.describe()
```

| | Unnamed: 0 | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 |
|---|---|---|---|---|---|---|
| count | 220320.000000 | 220301.000000 | 215522.000000 | 214869.000000 | 215213.000000 | 215725.000000 |
| mean | 110159.500000 | 590.673936 | 13.501537 | 15.843152 | 15.200721 | 14.799210 |
| std | 63601.049991 | 144.023912 | 2.163736 | 2.201155 | 2.037390 | 2.091963 |
| min | 0.000000 | 2.798032 | 0.014468 | 0.000000 | 0.028935 | 0.000000 |
| 25% | 55079.750000 | 626.620400 | 13.346350 | 15.907120 | 15.183740 | 15.053530 |
| 50% | 110159.500000 | 632.638916 | 13.642940 | 16.167530 | 15.494790 | 15.082470 |
| 75% | 165239.250000 | 637.615723 | 14.539930 | 16.427950 | 15.697340 | 15.118630 |
| max | 220319.000000 | 800.000000 | 22.251160 | 23.596640 | 24.348960 | 25.000000 |

데이터 특징 확인

Machine_status는 문자열 데이터라
확인 안됨

# 데이터 정리

실습

```
data= data.dropna()
```

Nan 데이터 제거

```
data=data[:80000]
```

데이터 개수 조절

```
data.drop(['Unnamed: 0', 'timestamp'],axis=1, inplace=True)
```

번호, 시간 데이터 제거

```
data['machine_status'].value_counts()
```

기계 상태 데이터 확인

```
machine_status
NORMAL         75341
RECOVERING      4655
BROKEN             4
Name: count, dtype: int64
```

```
conditions = [(data['machine_status'] =='NORMAL'), (data['machine_status'] =='BROKEN'), (data['machine_status'] =='RECOVERING')]
choices = [1, 0.1, 0.5]
data['Operation'] = np.select(conditions, choices, default=0)
```

데이터 라벨 encoding-문자열 피쳐를 숫자 값으로 변환

# 데이터 시각화

```
1 data.columns
```

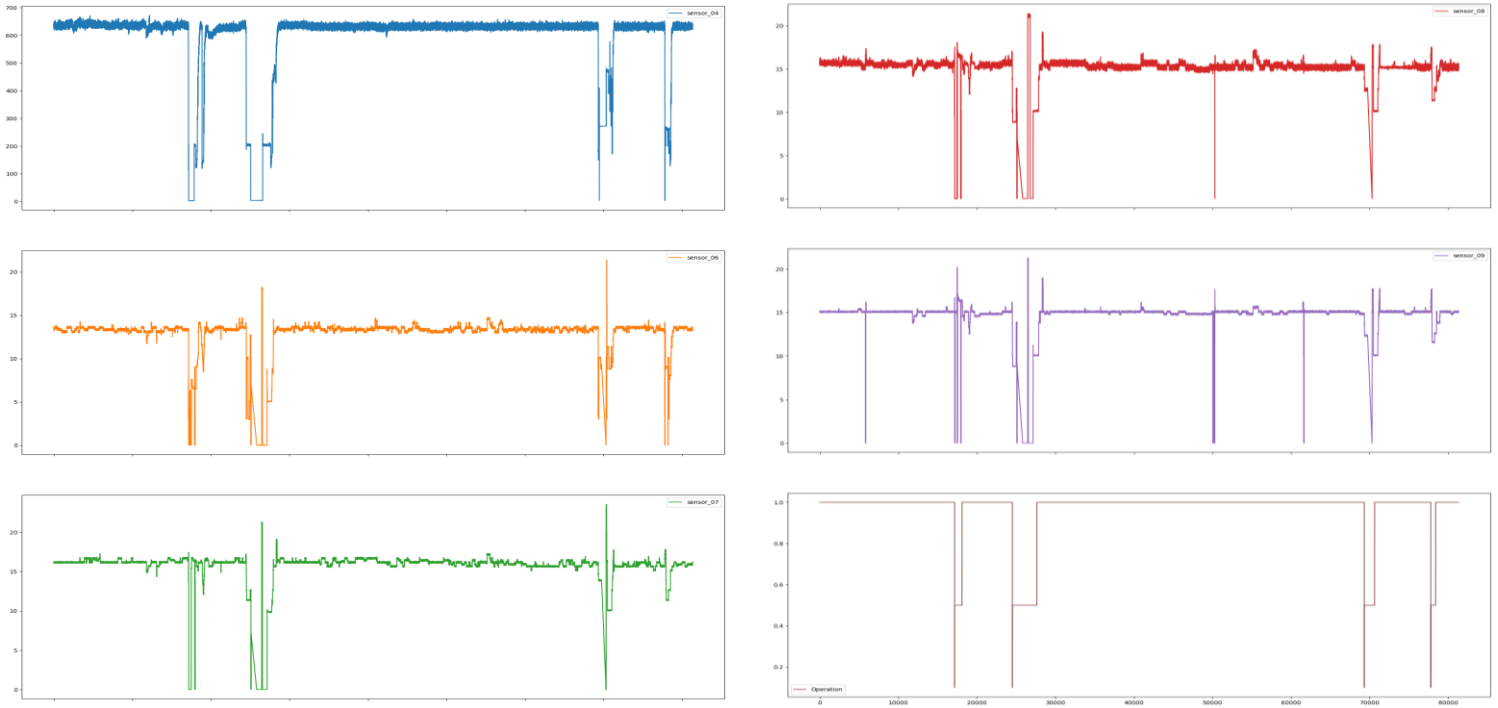
```
Index(['sensor_04', 'sensor_06', 'sensor_07', 'sensor_08', 'sensor_09',
       'machine_status', 'Operation'],
      dtype='object')
```

```
1 df = pd.DataFrame(data, columns=['sensor_04', 'sensor_06', 'sensor_07', 'sensor_08', 'sensor_09','Operation'])
```

데이터 프레임 생성

```
1 df.plot(subplots =True, sharex = True, figsize = (20,20))
```

데이터 시각화



```
1 df.shape
```

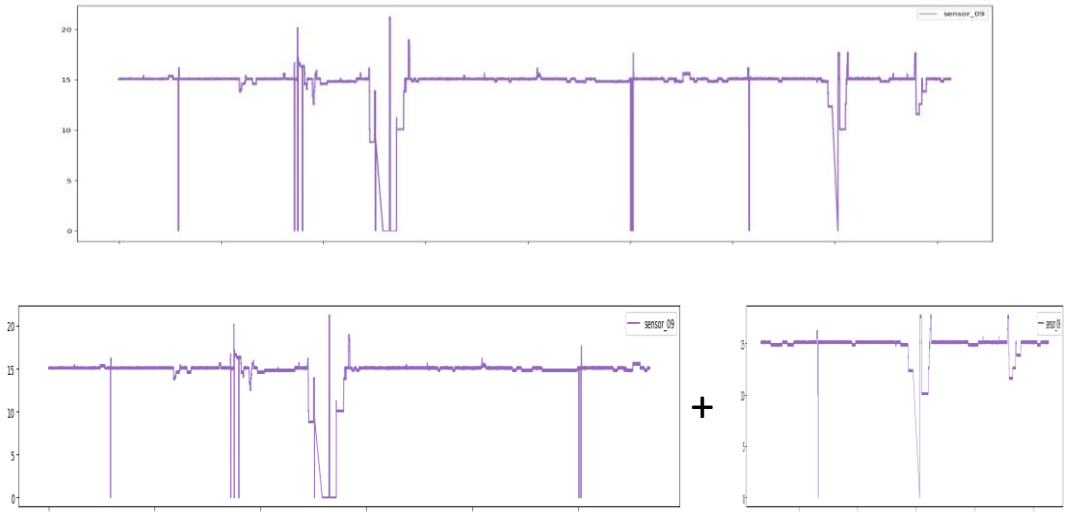
```
(80000, 6)
```

Operation
Normal : 1
Recovering : 0.5
Broken : 0.1

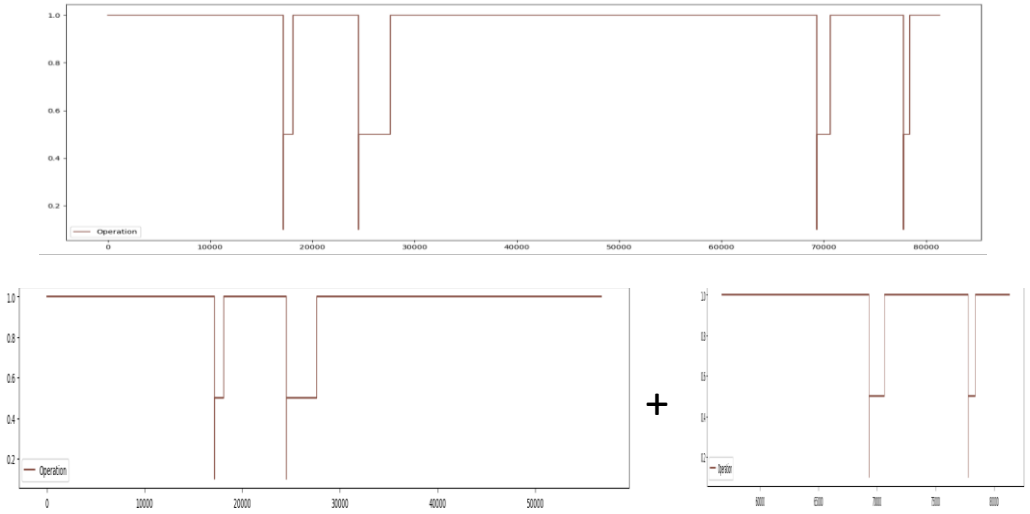
# Train, test 데이터 분할

```
1 seq_length = 20
2 batch = 100
3 train_size = int(len(df)*0.7)
4 train_set = df[0:train_size]
5 test_set = df[train_size-seq_length:]
```

Train : test = 7:3



**Sensor 9**



**Operation**

# 데이터 스케일링

```
# Input scale
scaler_x = MinMaxScaler()
scaler_x.fit(train_set.iloc[:, :-1])

train_set.iloc[:, :-1] = scaler_x.transform(train_set.iloc[:, :-1])
test_set.iloc[:, :-1] = scaler_x.transform(test_set.iloc[:, :-1])

# Output scale
scaler_y = MinMaxScaler()
scaler_y.fit(train_set.iloc[:, [-1]])

train_set.iloc[:, -1] = scaler_y.transform(train_set.iloc[:, [-1]])
test_set.iloc[:, -1] = scaler_y.transform(test_set.iloc[:, [-1]])
```

| | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 | Operation |
|---|---|---|---|---|---|---|
| 0 | 634.3750 | 13.41146 | 16.13136 | 15.56713 | 15.05353 | 1.0 |
| 1 | 634.3750 | 13.41146 | 16.13136 | 15.56713 | 15.05353 | 1.0 |
| 2 | 638.8889 | 13.32465 | 16.03733 | 15.61777 | 15.01013 | 1.0 |
| 3 | 628.1250 | 13.31742 | 16.24711 | 15.69734 | 15.08247 | 1.0 |
| 4 | 636.4583 | 13.35359 | 16.21094 | 15.69734 | 15.08247 | 1.0 |

변수값들의 크기가 제각각이므로
0-1사이 값으로 스케일링

| | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 | Operation |
|---|---|---|---|---|---|---|
| 0 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 1 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 2 | 0.950529 | 0.731029 | 0.753825 | 0.728779 | 0.705442 | 1.0 |
| 3 | 0.934442 | 0.730632 | 0.763686 | 0.732499 | 0.708844 | 1.0 |
| 4 | 0.946897 | 0.732618 | 0.761986 | 0.732499 | 0.708844 | 1.0 |

# 데이터셋 생성

```
1
2 device = torch.device('cpu')
3 # 데이터셋 생성 함수
4 def build_dataset(time_series, seq_length):
5     dataX = []
6     dataY = []
7     for i in range(0, len(time_series)-seq_length):
8         _x = time_series[i:i+seq_length, :]
9         _y = time_series[i+seq_length, [-1]]
10        # print(_x, "-->",_y)
11        dataX.append(_x)
12        dataY.append(_y)
13
14    return np.array(dataX), np.array(dataY)
15
16 trainX, trainY = build_dataset(np.array(train_set), seq_length)
17 testX, testY = build_dataset(np.array(test_set), seq_length)
18
19 # 텐서로 변환
20 trainX_tensor = torch.FloatTensor(trainX)
21 trainY_tensor = torch.FloatTensor(trainY)
22
23 testX_tensor = torch.FloatTensor(testX)
24 testY_tensor = torch.FloatTensor(testY)
25 testX_tensor = testX_tensor.to(device)
26 testY_tensor = testY_tensor.to(device)
27 # 텐서 형태로 데이터 정의
28 dataset = TensorDataset(trainX_tensor, trainY_tensor)
29 # 데이터로더는 기본적으로 2개의 인자를 입력받으며 배치크기는 통상적으로 2의 배수를 사용
30 dataloader = DataLoader(dataset,
31                         batch_size=batch,
32                         shuffle=True,
33                         drop_last=True)
```

```
1 a=[1,2,3,4,5,6]
```

```
1 a[0:3]
```

```
[1, 2, 3]
```

```
1 a[3]
```

```
4
```

```
1 seq_length = 20
```

| | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 | Operation |
|---|---|---|---|---|---|---|
| 1 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 2 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 3 | 0.950529 | 0.731029 | 0.753825 | 0.728779 | 0.705442 | 1.0 |
| ⋮ | | | ⋮ | | | |
| 18 | 0.934442 | 0.730632 | 0.763686 | 0.732499 | 0.708844 | 1.0 |
| 19 | 0.946897 | 0.732618 | 0.761986 | 0.732499 | 0.708844 | 1.0 |
| 20 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 21 | 0.950529 | 0.731029 | 0.753825 | 0.728779 | 0.705442 | 1.0 |
| 22 | 0.934442 | 0.730632 | 0.763686 | 0.732499 | 0.708844 | 1.0 |
| 23 | 0.946897 | 0.732618 | 0.761986 | 0.732499 | 0.708844 | 1.0 |

Input x1

Output y1

# 데이터셋 생성

```python

device = torch.device('cpu')
# 데이터셋 생성 함수
def build_dataset(time_series, seq_length):
    dataX = []
    dataY = []
    for i in range(0, len(time_series)-seq_length):
        _x = time_series[i:i+seq_length, :]
        _y = time_series[i+seq_length, [-1]]
        # print(_x, "-->",_y)
        dataX.append(_x)
        dataY.append(_y)

    return np.array(dataX), np.array(dataY)

trainX, trainY = build_dataset(np.array(train_set), seq_length)
testX, testY = build_dataset(np.array(test_set), seq_length)

# 텐서로 변환
trainX_tensor = torch.FloatTensor(trainX)
trainY_tensor = torch.FloatTensor(trainY)

testX_tensor = torch.FloatTensor(testX)
testY_tensor = torch.FloatTensor(testY)
testX_tensor = testX_tensor.to(device)
testY_tensor = testY_tensor.to(device)
# 텐서 형태로 데이터 정의
dataset = TensorDataset(trainX_tensor, trainY_tensor)
# 데이터로더는 기본적으로 2개의 인자를 입력받으며 배치크기는 통상적으로 2의 배수를 사용
dataloader = DataLoader(dataset,
                        batch_size=batch,
                        shuffle=True,
                        drop_last=True)
```

```
1 a=[1,2,3,4,5,6]
```

```
1 a[0:3]
```

```
[1, 2, 3]
```

```
1 a[3]
```

```
4
```

```
1 seq_length = 20
```

| | sensor_04 | sensor_06 | sensor_07 | sensor_08 | sensor_09 | Operation |
|---|---|---|---|---|---|---|
| 1 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 2 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 3 | 0.950529 | 0.731029 | 0.753825 | 0.728779 | 0.705442 | 1.0 |
| ⋮ | | | ⋮ | | | |
| 18 | 0.934442 | 0.730632 | 0.763686 | 0.732499 | 0.708844 | 1.0 |
| 19 | 0.946897 | 0.732618 | 0.761986 | 0.732499 | 0.708844 | 1.0 |
| 20 | 0.943783 | 0.735797 | 0.758245 | 0.726412 | 0.707483 | 1.0 |
| 21 | 0.950529 | 0.731029 | 0.753825 | 0.728779 | 0.705442 | 1.0 |
| 22 | 0.934442 | 0.730632 | 0.763686 | 0.732499 | 0.708844 | 1.0 |
| 23 | 0.946897 | 0.732618 | 0.761986 | 0.732499 | 0.708844 | 1.0 |

Input x2

Output y2

# LSTM 모델 작성

```
seq_length = 20
batch = 100
```

모델 변수 설정

모델 변수

LSTM 구조 정의

```
import torch.nn as nn

# 설정값
data_dim = 6
hidden_dim = 10
output_dim = 1
learning_rate = 0.01
nb_epochs = 100

class Net(nn.Module):
    # # 기본변수, layer를 초기화해주는 생성자
    def __init__(self, input_dim, hidden_dim, seq_len, output_dim, layers):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.output_dim = output_dim
        self.layers = layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=layers,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim, bias = True)

    # 학습 초기화를 위한 함수
    def reset_hidden_state(self):
        self.hidden = (
                torch.zeros(self.layers, self.seq_len, self.hidden_dim),
                torch.zeros(self.layers, self.seq_len, self.hidden_dim))

    # 예측을 위한 함수
    def forward(self, x):
        x, _status = self.lstm(x)
        x = self.fc(x[:, -1])
        return x
```

# 모델 training 함수 작성

```
for batch_idx, samples in enumerate(train_df):
```

```
>>> for i, letter in enumerate(['A', 'B', 'C']):
...     print(i, letter)
...
0 A
1 B
2 C
```

```
def train_model(model, train_df, num_epochs = None, lr = None, verbose = 10, patience = 10):

    criterion = nn.MSELoss().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
    nb_epochs = num_epochs

    # epoch마다 loss 저장
    train_hist = np.zeros(nb_epochs)

    for epoch in range(nb_epochs):
        avg_cost = 0
        total_batch = len(train_df)

        for batch_idx, samples in enumerate(train_df):

            x_train, y_train = samples

            # seq별 hidden state reset
            model.reset_hidden_state()

            # H(x) 계산
            outputs = model(x_train)

            # cost 계산
            loss = criterion(outputs, y_train)

            # cost로 H(x) 개선
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            avg_cost += loss/total_batch

        train_hist[epoch] = avg_cost

        if epoch % verbose == 0:
            print('Epoch:', '%04d' % (epoch), 'train loss :', '{:.4f}'.format(avg_cost))

        # patience번째 마다 early stopping 여부 확인
        if (epoch % patience == 0) & (epoch != 0):

            # loss가 커졌다면 early stop
            if train_hist[epoch-patience] < train_hist[epoch]:
                print('₩n Early Stopping')

                break

    return model.eval(), train_hist
```

```
tensor([[[0.9438, 0.7358, 0.7582, 0.7264, 0.7075, 1.0000],
         [0.9438, 0.7358, 0.7582, 0.7264, 0.7075, 1.0000],
         [0.9505, 0.7310, 0.7538, 0.7288, 0.7054, 1.0000],
         [0.9344, 0.7306, 0.7637, 0.7325, 0.7088, 1.0000],
         [0.9469, 0.7326, 0.7620, 0.7325, 0.7088, 1.0000],
         [0.9486, 0.7358, 0.7599, 0.7416, 0.7126, 1.0000],
         [0.9422, 0.7370, 0.7582, 0.7305, 0.7088, 1.0000],
         [0.9382, 0.7271, 0.7579, 0.7558, 0.7088, 1.0000],
         [0.9402, 0.7290, 0.7582, 0.7220, 0.7105, 1.0000],
         [0.9549, 0.7342, 0.7637, 0.7288, 0.7105, 1.0000],
         [0.9488, 0.7358, 0.7599, 0.7305, 0.7105, 1.0000],
         [0.9457, 0.7358, 0.7599, 0.7396, 0.7105, 1.0000],
         [0.9374, 0.7322, 0.7599, 0.7342, 0.7054, 1.0000],
         [0.9502, 0.7306, 0.7599, 0.7396, 0.7129, 1.0000],
         [0.9408, 0.7346, 0.7582, 0.7247, 0.7092, 1.0000],
         [0.9557, 0.7326, 0.7620, 0.7210, 0.7109, 1.0000],
         [0.9376, 0.7298, 0.7599, 0.7416, 0.7088, 1.0000],
         [0.9576, 0.7322, 0.7599, 0.7288, 0.7054, 1.0000],
         [0.9417, 0.7322, 0.7620, 0.7379, 0.7075, 1.0000],
         [0.9587, 0.7322, 0.7582, 0.7328, 0.7088, 1.0000]],

tensor([[1.],
        [1.]])
```

# 모델 학습

```python
# 모델 학습
# 설정값
data_dim = 6
hidden_dim = 10
output_dim = 1
learning_rate = 0.001
nb_epochs = 100
net = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 20, patience = 10)
```

```python
class Net(nn.Module):
    # # 기본변수, layer를 초기화해주는 생성자
    def __init__(self, input_dim, hidden_dim, seq_len, output_dim, layers):
        super(Net, self).__init__()
        self.hidden_dim = hidden_dim
        self.seq_len = seq_len
        self.output_dim = output_dim
        self.layers = layers

        self.lstm = nn.LSTM(input_dim, hidden_dim, num_layers=layers,
                            # dropout = 0.1,
                            batch_first=True)
        self.fc = nn.Linear(hidden_dim, output_dim, bias = True)

    # 학습 초기화를 위한 함수
    def reset_hidden_state(self):
        self.hidden = (
                torch.zeros(self.layers, self.seq_len, self.hidden_dim),
                torch.zeros(self.layers, self.seq_len, self.hidden_dim))

    # 예측을 위한 함수
    def forward(self, x):
        x, _status = self.lstm(x)
        x = self.fc(x[:, -1])
        return x
```

```python
def train_model(model, train_df, num_epochs = None, lr = None, verbose = 10, patience = 10):

    criterion = nn.MSELoss().to(device)
    optimizer = torch.optim.Adam(model.parameters(), lr = learning_rate)
    nb_epochs = num_epochs

    # epoch마다 loss 저장
    train_hist = np.zeros(nb_epochs)

    for epoch in range(nb_epochs):
        avg_cost = 0
        total_batch = len(train_df)

        for batch_idx, samples in enumerate(train_df):

            x_train, y_train = samples

            # seq별 hidden state reset
            model.reset_hidden_state()

            # H(x) 계산
            outputs = model(x_train)

            # cost 계산
            loss = criterion(outputs, y_train)

            # cost로 H(x) 개선
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            avg_cost += loss/total_batch

        train_hist[epoch] = avg_cost
```

# 모델 학습

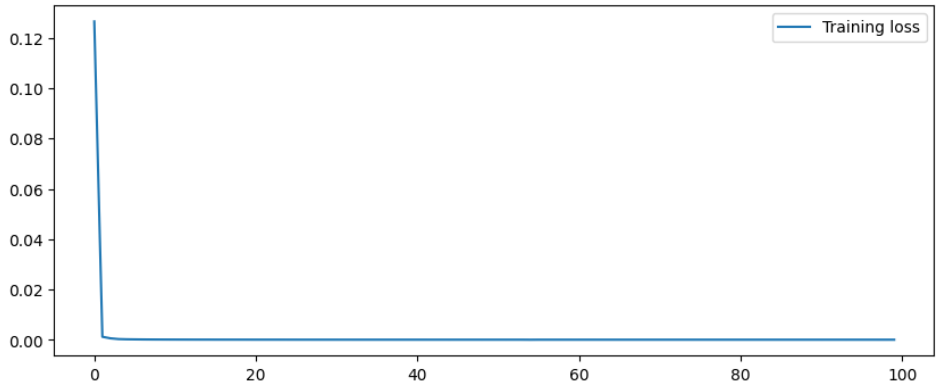
실습

```
# 모델 학습
# 설정값
data_dim = 6
output_dim = 1
hidden_dim = 10
learning_rate = 0.001
nb_epochs = 0
net = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 5, patience = 10)
```

자유롭게 설정 가능

```
Epoch: 0000 train loss : 0.1265
Epoch: 0020 train loss : 0.0001
Epoch: 0040 train loss : 0.0001
Epoch: 0060 train loss : 0.0001
Epoch: 0080 train loss : 0.0000
```

```
1 # epoch별 손실값
2 fig = plt.figure(figsize=(10, 4))
3 plt.plot(train_hist, label="Training loss")
4 plt.legend()
5 plt.show()
```



# 모델 save & load

실습

```
# 모델 저장
PATH ='model2/model01.pth'
#torch.save(model.state_dict(), PATH)

# 불러오기
model = Net(data_dim, hidden_dim, seq_length, output_dim, 1)
model.load_state_dict(torch.load(PATH), strict=False)
model.eval()
```

```
Net(
  (lstm): LSTM(6, 10, batch_first=True)
  (fc): Linear(in_features=10, out_features=1, bias=True)
)
```

# 모델 테스트

실습

```
# 예측 테스트
a=12000
testX_tensor_2000=testX_tensor[a:a+2000]
testY_tensor_2000=testY_tensor[a:a+2000]
with torch.no_grad():
    pred = []
    for pr in range(len(testX_tensor_2000)):

        model.reset_hidden_state()

        predicted = model(torch.unsqueeze(testX_tensor_2000[pr], 0))
        predicted = torch.flatten(predicted).item()
        pred.append(predicted)

    # INVERSE
    pred_inverse = scaler_y.inverse_transform(np.array(pred).reshape(-1, 1))
    testY_inverse = scaler_y.inverse_transform(testY_tensor_2000)
```

# 모델 테스트

```
1 fig = plt.figure(figsize=(10,3))
2 plt.plot(np.arange(len(pred_inverse)), pred_inverse, label = 'pred')
3 plt.plot(np.arange(len(testY_inverse)), testY_inverse, label = 'true')
4 plt.title("Test plot")
5 plt.legend()
6 plt.show()
```