# LSTM

실습

**박석희 교수님**

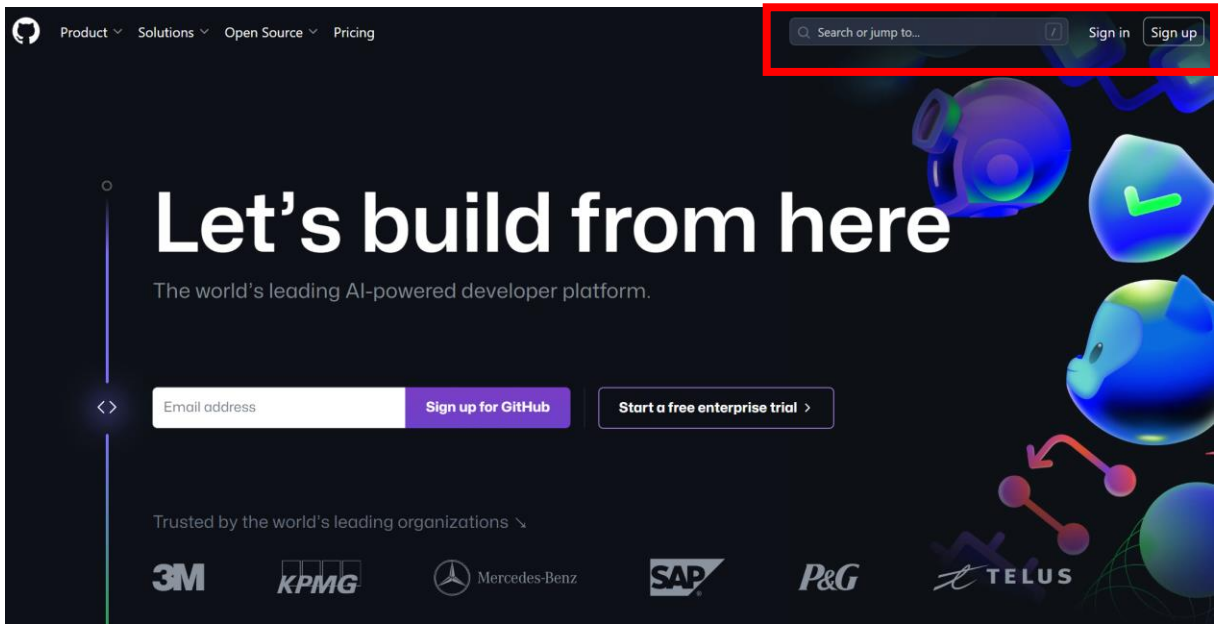
실습조교 이승문
ch273404@naver.com

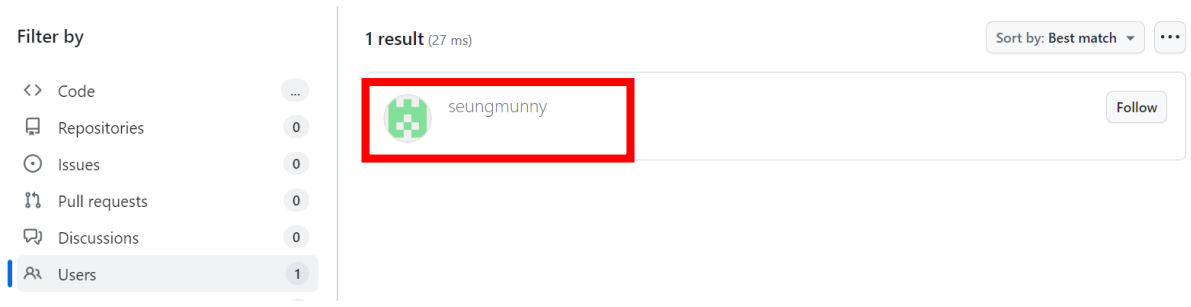
2/1(목)

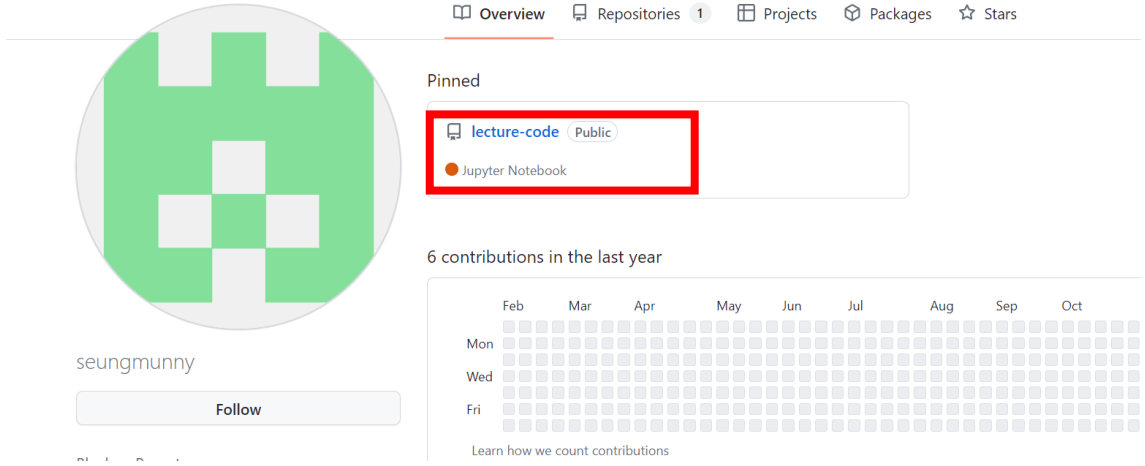
# 코드 및 데이터 다운로드

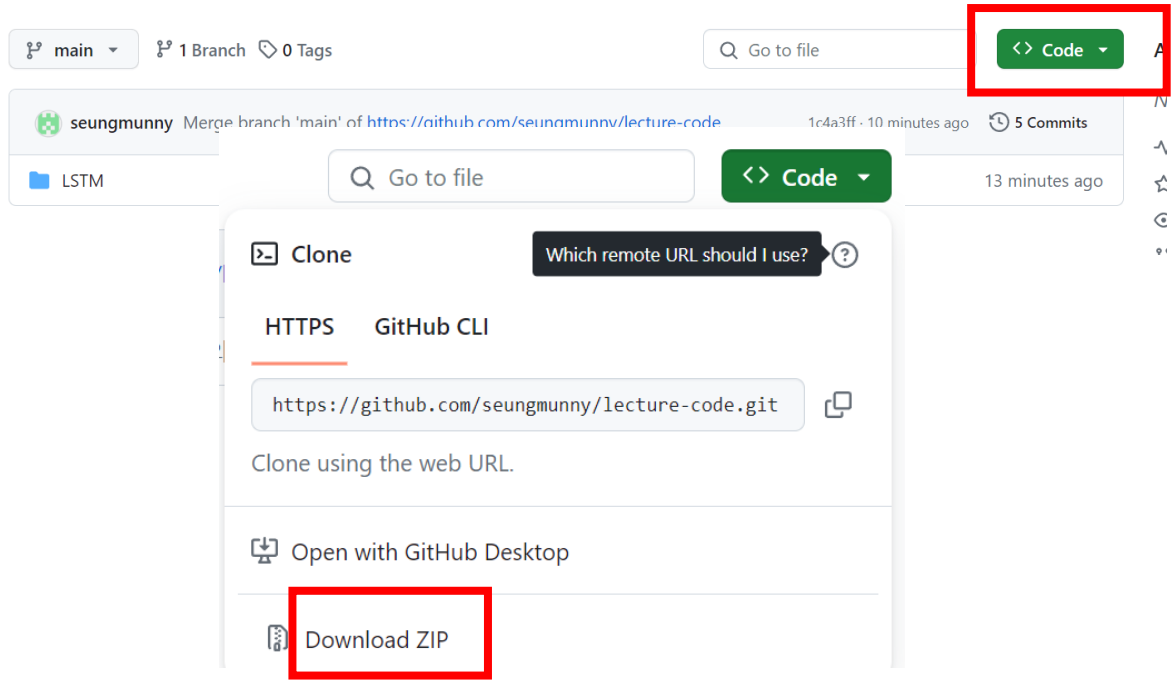
깃허브 접속

Seungmunny 입력

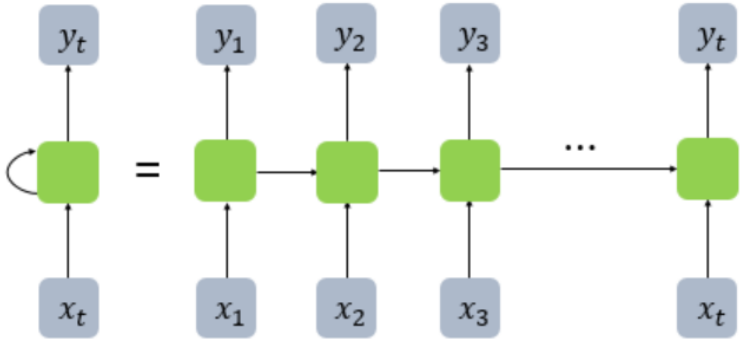


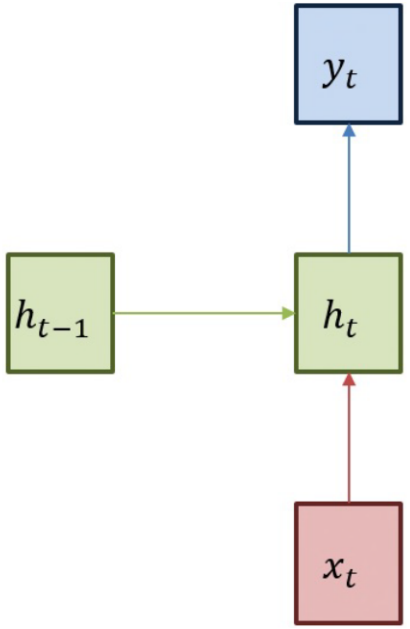



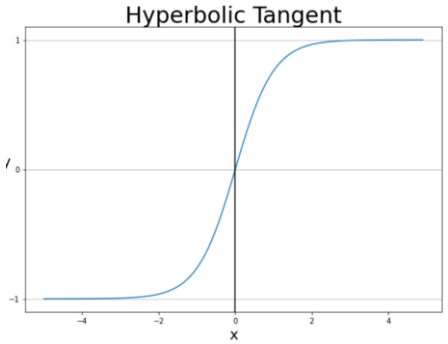
# 전통적 RNN(Recurrent Neural Network)




$$y_t = W_{hy}h_t + b_y$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$




https://gooopy.tistory.com/54

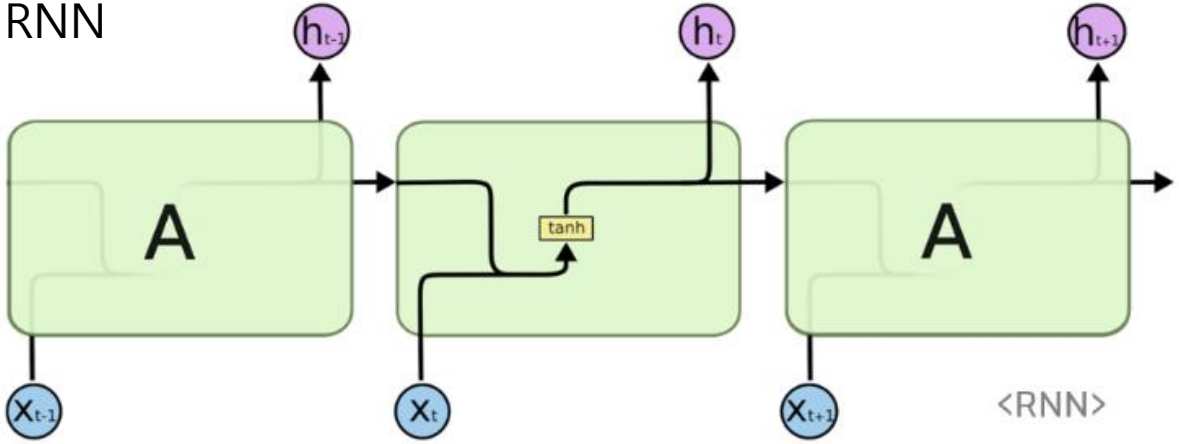
- 은닉층의 노드 결과값을 다음 은닉층의 노드에도 전달하여 이전 데이터가 다음 출력에 영향을 주는 형태
- 시계열 데이터를 다루는데 최적인 neural network
- Time step이 늘어날수록 학습 과정에서의 역전파 시 gradient가 소멸되어 파라미터 업데이트가 불가능한 vanishing gradient problem 발생-**장기 의존성 문제**
- 즉, 입력 데이터가 커질 때 데이터 뒤쪽으로 갈수록 앞쪽 데이터를 잊는다는 의미

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

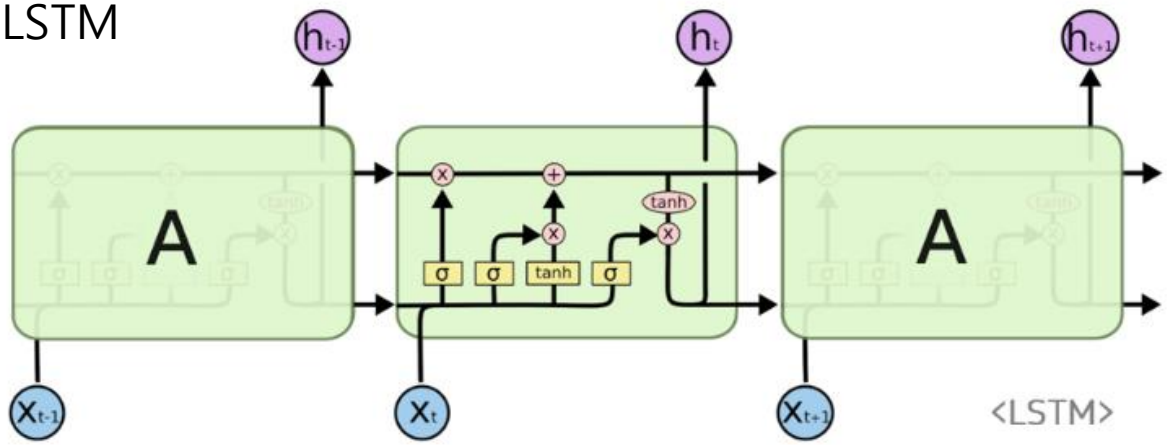
# LSTM(Long Short-Term Memory models)

- RNN의 장기의존성 문제를 해결하기 위해 hidden state에 cell state를 추가

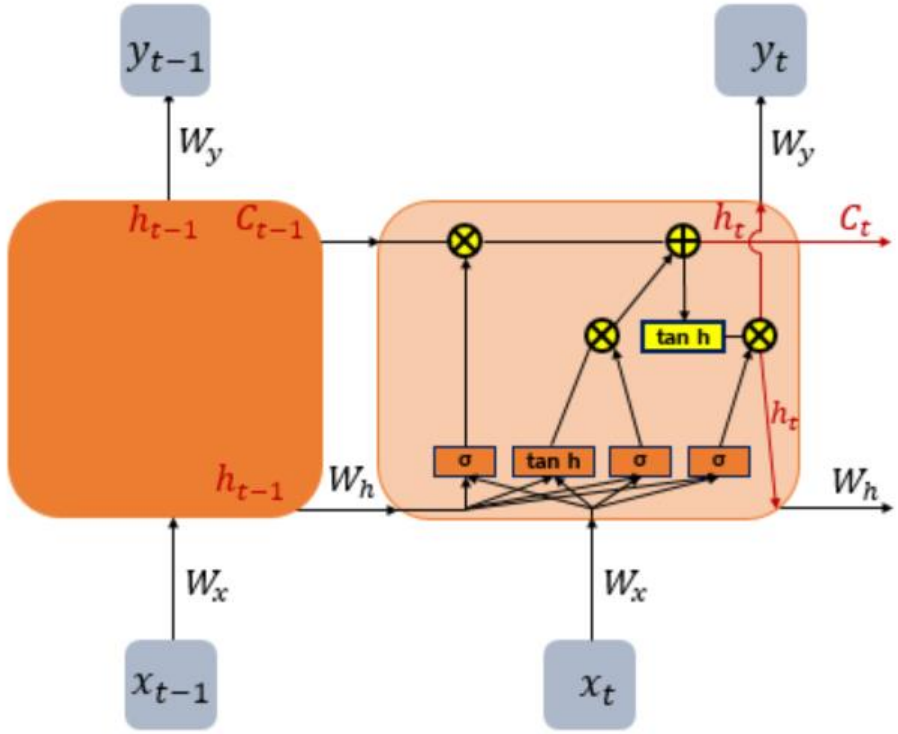
RNN




LSTM




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

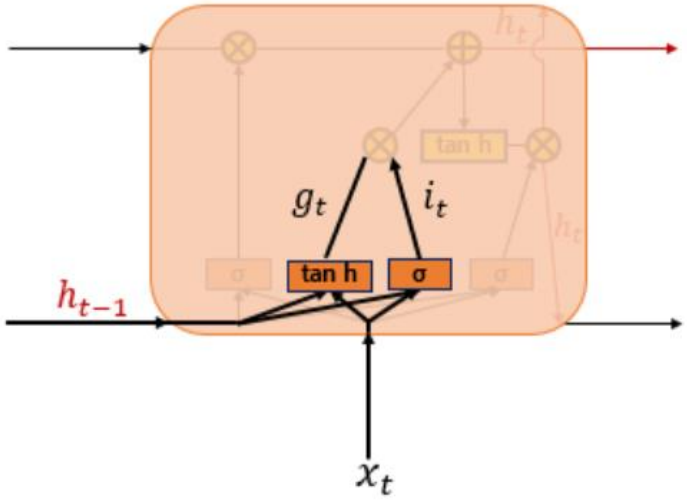



https://velog.io/@soup1997/LSTM-Long-Short-Term-Memory

# LSTM의 구조

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

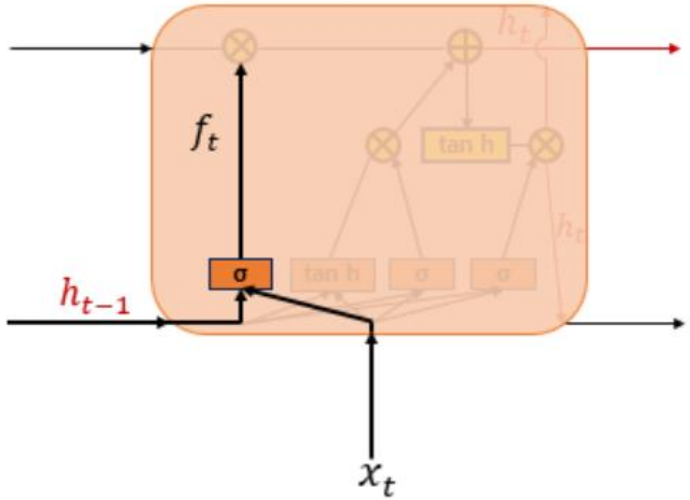
## ➢ Input gate




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

- 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




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

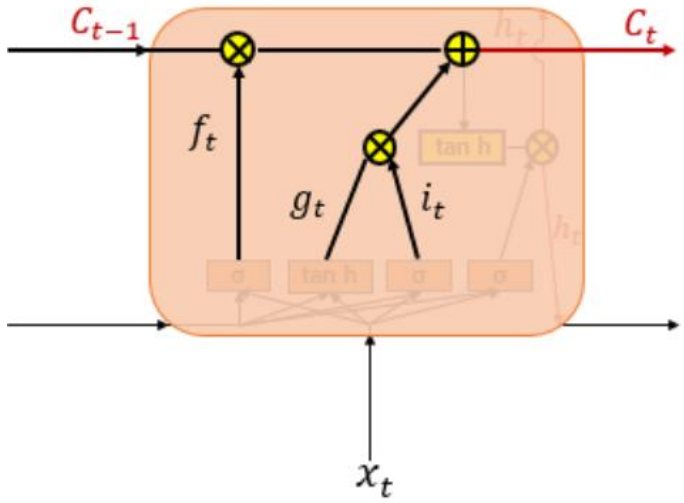
- 과거 정보를 잊기 위한 게이트
- σ를 통해 나온 결과인 $f_t$는 0~1 범위를 가지며 삭제 과정을 거친 정보의 양을 뜻함
- 0은 이전 정보를 모두 잊고, 1은 이전 정보를 온전히 기억한다는 뜻

# LSTM의 구조

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

➢ **Cell state layer(장기 상태)**

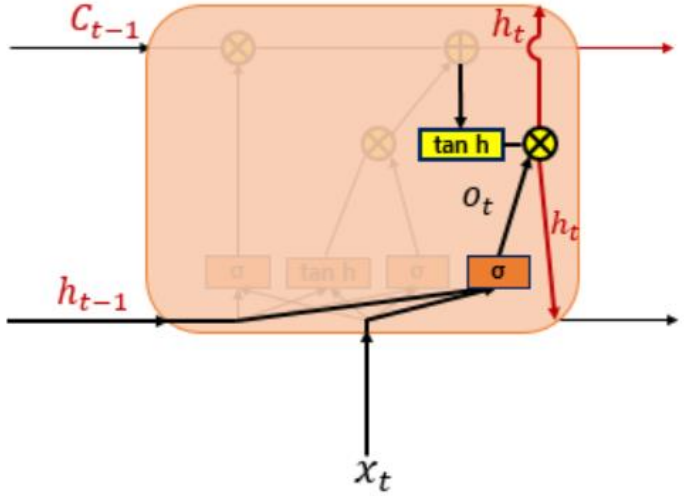
⊙는 요소별 곱셈




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

- 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)**



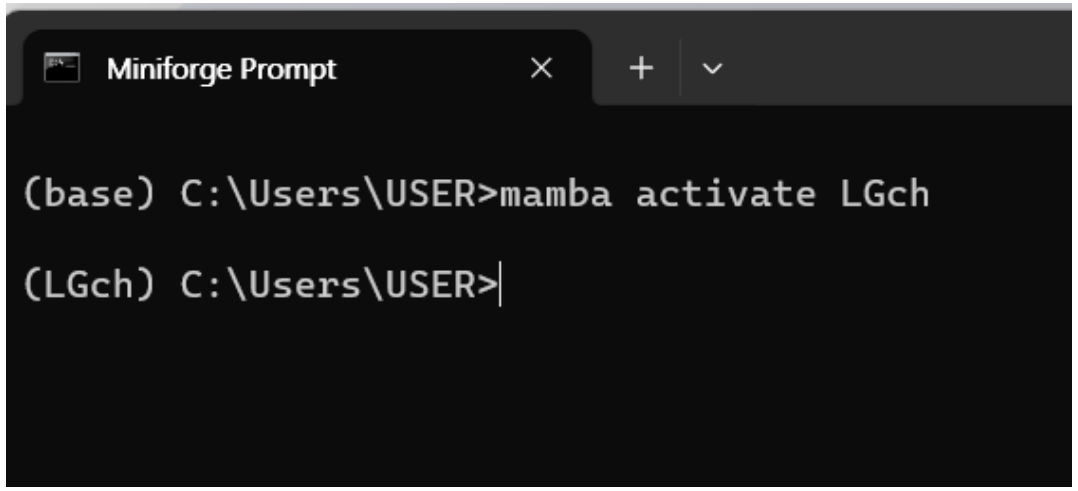

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

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

# Miniforge jupyter notebook 실행



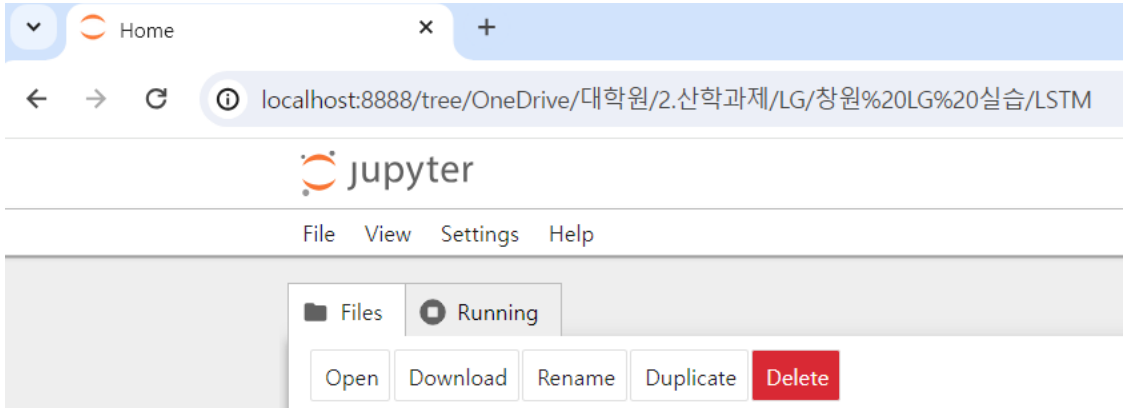

가상환경이 있다면 실행

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

가상환경 만들기 코드
envname에 원하는 이름 입력

Jupyter notebook 실행

```
(LGch) C:\Users\USER>jupyter notebook
```



# 초기 설정 및 데이터 다운로드

## 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
```

**Miniforge prompt에서 실행**

```
mamba install pytorch==2.1.1 torchvision==0.16.1
torchaudio==2.1.1 -c pytorch
```

## 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
```

# 초기 설정 및 데이터 다운로드

실습

## 데이터 다운로드

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

| | sensor1 | sensor2 | time_x | speedSet | load_value | gear_fault_desc |
|---|---|---|---|---|---|---|
| **0** | 2.523465 | 2.430168 | 2023-05-03 21:47:31.000000 | 8.332031 | 0 | No fault |
| **1** | 2.521494 | 2.430003 | 2023-05-03 21:47:31.000200 | 8.332031 | 0 | No fault |
| **2** | 2.522479 | 2.429675 | 2023-05-03 21:47:31.000400 | 8.332031 | 0 | No fault |
| **3** | 2.521330 | 2.431810 | 2023-05-03 21:47:31.000600 | 8.332031 | 0 | No fault |
| **4** | 2.522479 | 2.431317 | 2023-05-03 21:47:31.000800 | 8.332031 | 0 | No fault |
| **...** | ... | ... | ... | ... | ... | ... |
| **149995** | 2.549417 | 2.441830 | 2023-05-03 22:06:06.999000 | 40.000000 | 80 | No fault |
| **149996** | 2.496363 | 2.453820 | 2023-05-03 22:06:06.999200 | 40.000000 | 80 | No fault |
| **149997** | 2.520837 | 2.418505 | 2023-05-03 22:06:06.999400 | 40.000000 | 80 | No fault |
| **149998** | 2.499319 | 2.417027 | 2023-05-03 22:06:06.999600 | 40.000000 | 80 | No fault |
| **149999** | 2.515088 | 2.419984 | 2023-05-03 22:06:06.999800 | 40.000000 | 80 | No fault |

150000 rows × 6 columns

서로 다른 속도, load에 따른
x,y축 진동센서 데이터

## Mechanical Gear Vibration Dataset

Vibration of six gear types are measured under various working conditions

Data Card | Code (1) | Discussion (1)

### About Dataset

Reliable mechatronics systems are critical for modern fabrication as well as our daily life. They can help forecast machine downtime, find error causes, or warn about dangerous settings. Yet, their principal components (e.g., gears) are most likely to be damaged with little to no chance of early prediction, costing the industry millions of dollars per year.

To study the potential of a machine-learning-based resolution, we measure the vibration of five defective gears and one normal gear under various loads and speeds. The dataset contains six CSV files representing those six different types. With this, you can try to:

1. Classify gear defection under the same speed and load
2. Classify gear defection under any speed or load
3. Regress the speed and/or load from a given measurement

Notes:

- Sensor 1 measures the vibration (mm) along x-axis.
- Sensor 2 measures the vibration (mm) along y-axis.
- The gears are operated at 3 speeds: 8.33, 25, and 40 (rev/sec) under 2 loads: 0 and 80 (Nm).
- Sampling time is 0.0002 seconds.

Read more in the data "Provenance" section.

Hint:

- Data is of time series type
- To make a data sample, you first need to choose a sampling duration *ts*. For example, *ts = 1* (second) means that each sample is a time series of 1/0.0002 = 5,000 values. Choose *ts* wisely.

**Usability** 10.00

**License** Community Data License Agree...

**Expected update frequency** Annually

**Tags**

Business | Earth and Nature | Beginner | Classification | Time Series Analysis | Manufacturing | Regression

https://www.kaggle.com/datasets/hieudaotrung/gear-vibration?select=no_fault.csv

# 데이터 정리

실습

```
date_time = pd.to_datetime(df.pop('time_x'), format='%Y-%m-%d %H:%M:%S.%f')
df=df.drop(['gear_fault_desc'], axis=1)
df = df[['speedSet', 'load_value','sensor1', 'sensor2']]
```

대소문자 및
대괄호 확인

시간 정보는 따로 저장
Gear_fault_desc 데이터는 삭제
sensor2 데이터를 예측하기 위해 데이터 순서 변경

```
df.head()
```

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| 0 | 8.332031 | 0 | 2.523465 | 2.430168 |
| 1 | 8.332031 | 0 | 2.521494 | 2.430003 |
| 2 | 8.332031 | 0 | 2.522479 | 2.429675 |
| 3 | 8.332031 | 0 | 2.521330 | 2.431810 |
| 4 | 8.332031 | 0 | 2.522479 | 2.431317 |

```
df.describe()
```

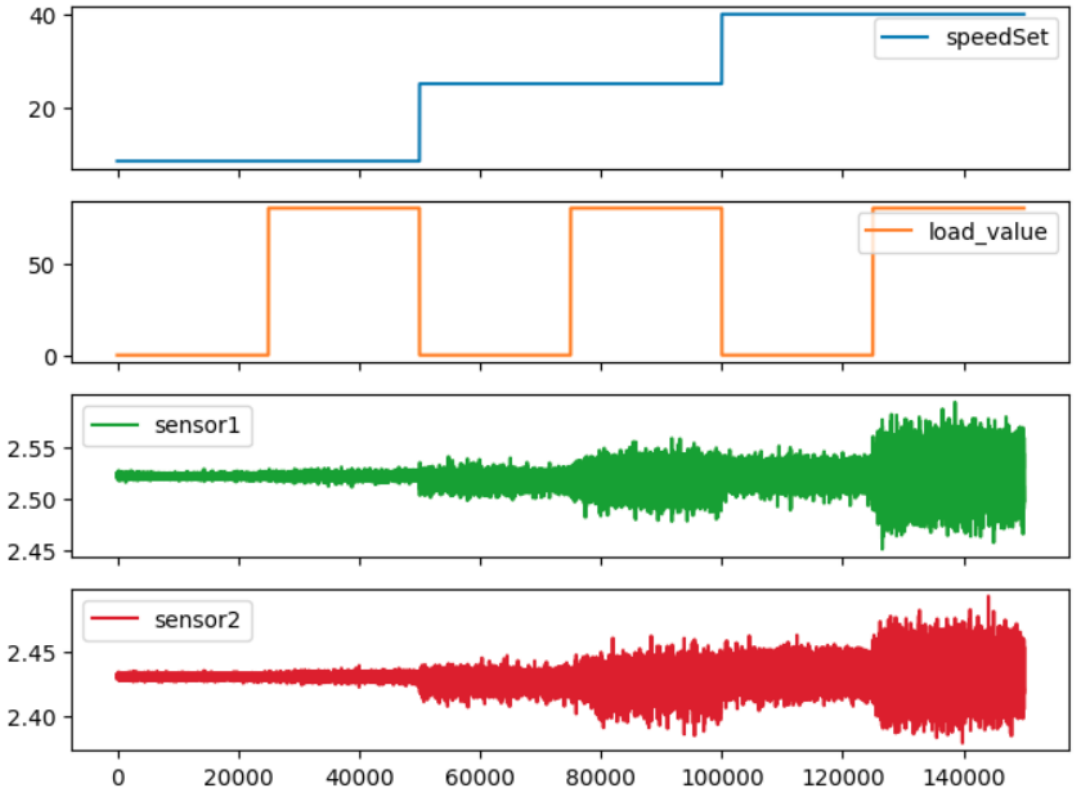
| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| count | 150000.000000 | 150000.000000 | 150000.000000 | 150000.000000 |
| mean | 24.444010 | 40.000000 | 2.520499 | 2.429825 |
| std | 12.934413 | 40.000133 | 0.007967 | 0.006862 |
| min | 8.332031 | 0.000000 | 2.451357 | 2.378756 |
| 25% | 8.332031 | 0.000000 | 2.517223 | 2.426882 |
| 50% | 25.000000 | 40.000000 | 2.520837 | 2.430003 |
| 75% | 40.000000 | 80.000000 | 2.523136 | 2.432139 |
| max | 40.000000 | 80.000000 | 2.592781 | 2.493242 |

# 데이터 확인 및 분할

실습

```
#데이터 확인
plot_cols = ['speedSet', 'load_value','sensor1', 'sensor2']
plot_features = df[plot_cols]
_ = plot_features.plot(subplots=True)
```

Data 0 1 2 3 4 5




```
#train/test data 분할
# 7개 데이터가 입력으로 들어가고 batch size는 임의로 지정
seq_length = 7
batch = 100

data=0
train_set = df[data*25000:data*25000+14000]
test_set = df[data*25000+14000-seq_length:data*25000+20000]
```

속도 및 load별 데이터 추출
20000개 데이터 사용

Train data 14000개
Test data 6000개

# 데이터 확인

```
train_set
```

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| **0** | 8.332031 | 0 | 2.523465 | 2.430168 |
| **1** | 8.332031 | 0 | 2.521494 | 2.430003 |
| **2** | 8.332031 | 0 | 2.522479 | 2.429675 |
| **3** | 8.332031 | 0 | 2.521330 | 2.431810 |
| **4** | 8.332031 | 0 | 2.522479 | 2.431317 |
| **...** | ... | ... | ... | ... |
| **13995** | 8.332031 | 0 | 2.521165 | 2.431646 |
| **13996** | 8.332031 | 0 | 2.521987 | 2.430168 |
| **13997** | 8.332031 | 0 | 2.521494 | 2.430825 |
| **13998** | 8.332031 | 0 | 2.521165 | 2.430825 |
| **13999** | 8.332031 | 0 | 2.521330 | 2.431482 |

```
test_set
```

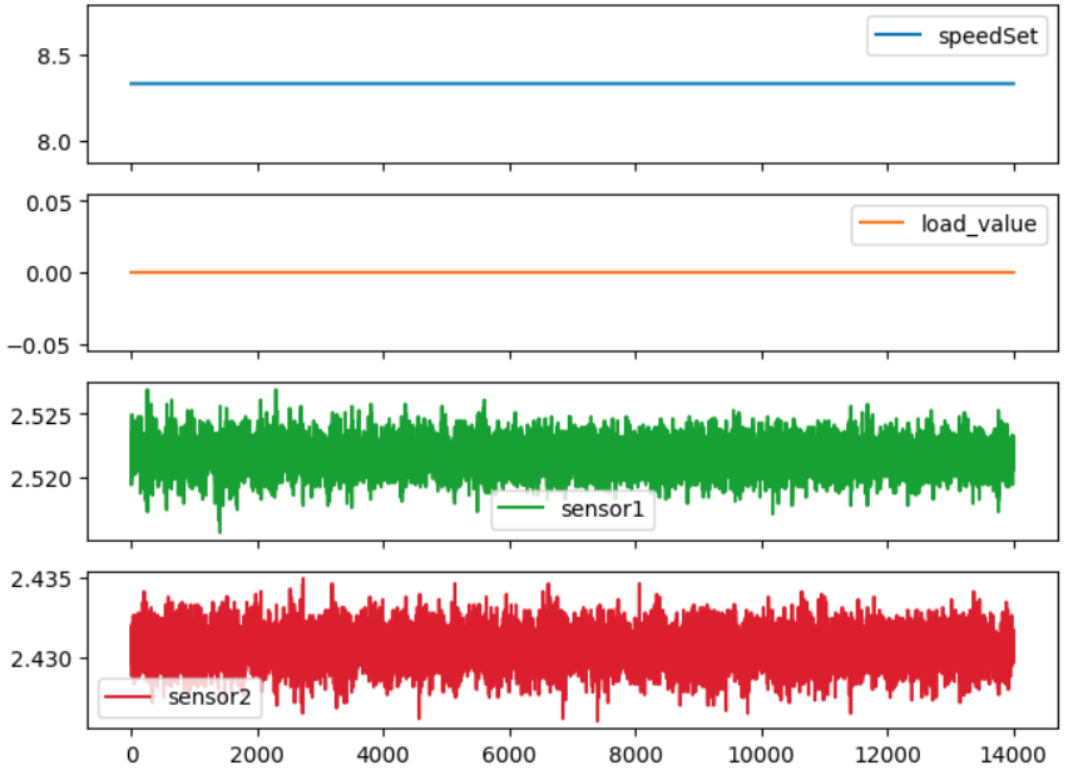
| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| **13993** | 8.332031 | 0 | 2.521165 | 2.430496 |
| **13994** | 8.332031 | 0 | 2.520508 | 2.430989 |
| **13995** | 8.332031 | 0 | 2.521165 | 2.431646 |
| **13996** | 8.332031 | 0 | 2.521987 | 2.430168 |
| **13997** | 8.332031 | 0 | 2.521494 | 2.430825 |
| **...** | ... | ... | ... | ... |
| **19995** | 8.332031 | 0 | 2.519194 | 2.430496 |
| **19996** | 8.332031 | 0 | 2.519687 | 2.429346 |
| **19997** | 8.332031 | 0 | 2.520016 | 2.429675 |
| **19998** | 8.332031 | 0 | 2.520344 | 2.430003 |
| **19999** | 8.332031 | 0 | 2.519851 | 2.431317 |

# 데이터 확인

```
plot_cols = ['speedSet', 'load_value','sensor1','sensor2']
plot_features = train_set[plot_cols]
_ = plot_features.plot(subplots=True)

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

Train data




Test data

# 데이터 스케일링

실습

```
# Input scale
train_set = train_set.copy()
test_set = test_set.copy()

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]])
```

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| 0 | 8.332031 | 0 | 2.523465 | 2.430168 |
| 1 | 8.332031 | 0 | 2.521494 | 2.430003 |
| 2 | 8.332031 | 0 | 2.522479 | 2.429675 |
| 3 | 8.332031 | 0 | 2.521330 | 2.431810 |
| 4 | 8.332031 | 0 | 2.522479 | 2.431317 |

변수값들의 크기가 제각각이므로
0-1사이 값으로 스케일링

```
train_set.head()
```

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| 0 | 0.0 | 0 | 0.695652 | 0.462963 |
| 1 | 0.0 | 0 | 0.521739 | 0.444444 |
| 2 | 0.0 | 0 | 0.608696 | 0.407407 |
| 3 | 0.0 | 0 | 0.507246 | 0.648148 |
| 4 | 0.0 | 0 | 0.608696 | 0.592593 |

# 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)
```

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

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

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

```
1 a[3]
```

```
4
```

```
2 # 7개의 데이터가 입력으로 들어가고 batch size는 임의로 지정
3 seq_length = 7
4 batch = 100
```

실습

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| 1 | 0.0 | 0 | 0.695652 | 0.462963 |
| 2 | 0.0 | 0 | 0.521739 | 0.444444 |
| 3 | 0.0 | 0 | 0.608696 | 0.407407 |
| 4 | 0.0 | 0 | 0.507246 | 0.648148 |
| 5 | 0.0 | 0 | 0.608696 | 0.592593 |
| 6 | 0.0 | 0 | 0.333333 | 0.500000 |
| 7 | 0.0 | 0 | 0.565217 | 0.518518 |
| 8 | 0.0 | 0 | 0.478261 | 0.574074 |
| 9 | 0.0 | 0 | 0.478261 | 0.574074 |
| 10 | 0.0 | 0 | 0.739130 | 0.666667 |

Input, x1

Output, y1

# 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)
```

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

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

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

```
1 a[3]
```

```
4
```

```
2 seq_length = 7
```

| | speedSet | load_value | sensor1 | sensor2 |
|---|---|---|---|---|
| 1 | 0.0 | 0 | 0.695652 | 0.462963 |
| 2 | 0.0 | 0 | 0.521739 | 0.444444 |
| 3 | 0.0 | 0 | 0.608696 | 0.407407 |
| 4 | 0.0 | 0 | 0.507246 | 0.648148 |
| 5 | 0.0 | 0 | 0.608696 | 0.592593 |
| 6 | 0.0 | 0 | 0.333333 | 0.500000 |
| 7 | 0.0 | 0 | 0.565217 | 0.518518 |
| 8 | 0.0 | 0 | 0.478261 | 0.574074 |
| 9 | 0.0 | 0 | 0.478261 | 0.574074 |
| 10 | 0.0 | 0 | 0.739130 | 0.666667 |

Input, x2

Output, y2

# LSTM 모델 작성

```
1 # 7일간의 데이터가 입력으로 들어가고 batch size는 임의로 지정
2 seq_length = 7
3 batch = 100
```

```
import torch.nn as nn

# 설정값
#data_dim = 4
#hidden_dim = 15
#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,
                            #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
```

모델 변수

LSTM 구조 정의

# 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 : %04d epoch' %(epoch))

            break

return model.eval(), train_hist
```

```
tensor([[[0.0000, 0.0000, 0.6957, 0.4630],
         [0.0000, 0.0000, 0.5217, 0.4444],
         [0.0000, 0.0000, 0.6087, 0.4074],
         [0.0000, 0.0000, 0.5072, 0.6481],
         [0.0000, 0.0000, 0.6087, 0.5926],
         [0.0000, 0.0000, 0.3333, 0.5000],
         [0.0000, 0.0000, 0.5652, 0.5185]],

        [[0.0000, 0.0000, 0.5217, 0.4444],
         [0.0000, 0.0000, 0.6087, 0.4074],
         [0.0000, 0.0000, 0.5072, 0.6481],
         [0.0000, 0.0000, 0.6087, 0.5926],
         [0.0000, 0.0000, 0.3333, 0.5000],
         [0.0000, 0.0000, 0.5652, 0.5185],
         [0.0000, 0.0000, 0.4783, 0.5741]]])
tensor([[0.5741],
        [0.5741]])
```

# 모델 학습

실습

```
# 모델 학습
# 설정값
data_dim = 4
hidden_dim = 256
output_dim = 1
learning_rate = 0.001
n_layer=2
nb_epochs = 200
net = Net(data_dim, hidden_dim, seq_length, output_dim, n_layer)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 10, 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
```

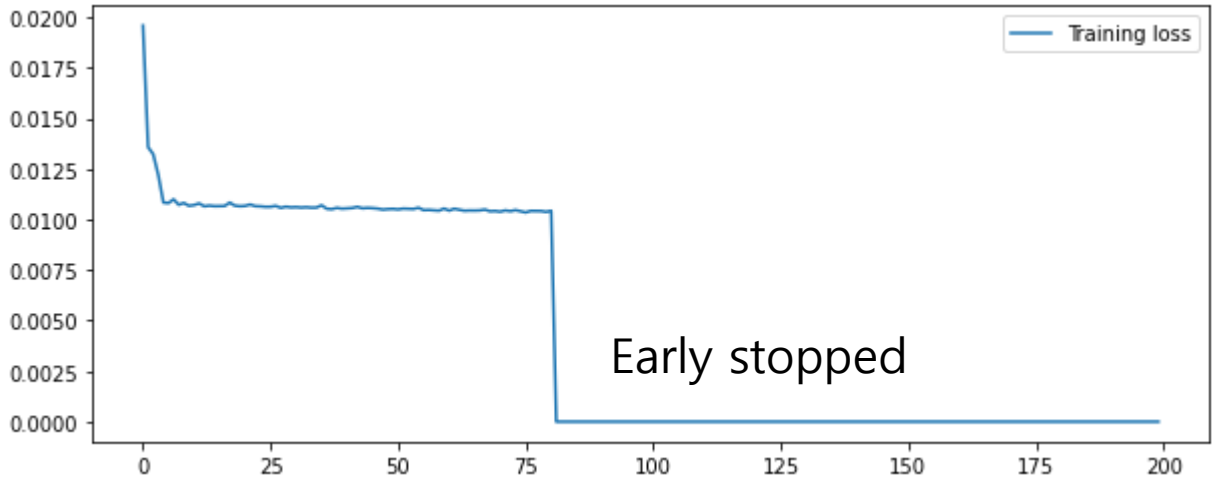
# 모델 학습

```
# 모델 학습
# 설정값
data_dim = 4
hidden_dim = 256
output_dim = 1
learning_rate = 0.001
n_layer=2
nb_epochs = 200
net = Net(data_dim, hidden_dim, seq_length, output_dim, n_layer)
model, train_hist = train_model(net, dataloader, num_epochs = nb_epochs, lr = learning_rate, verbose = 10, patience = 10)
```

```
Epoch: 0053 train loss : 0.0105
Epoch: 0054 train loss : 0.0106
Epoch: 0055 train loss : 0.0105
Epoch: 0056 train loss : 0.0105
Epoch: 0057 train loss : 0.0105
Epoch: 0058 train loss : 0.0104
Epoch: 0059 train loss : 0.0105
Epoch: 0060 train loss : 0.0104
Epoch: 0061 train loss : 0.0105
Epoch: 0062 train loss : 0.0105
Epoch: 0063 train loss : 0.0104
Epoch: 0064 train loss : 0.0104
Epoch: 0065 train loss : 0.0104
Epoch: 0066 train loss : 0.0104
Epoch: 0067 train loss : 0.0105
Epoch: 0068 train loss : 0.0104
Epoch: 0069 train loss : 0.0104
Epoch: 0070 train loss : 0.0104
Epoch: 0071 train loss : 0.0104
Epoch: 0072 train loss : 0.0104
Epoch: 0073 train loss : 0.0105
Epoch: 0074 train loss : 0.0104
Epoch: 0075 train loss : 0.0103
Epoch: 0076 train loss : 0.0104
Epoch: 0077 train loss : 0.0104
Epoch: 0078 train loss : 0.0104
Epoch: 0079 train loss : 0.0104
Epoch: 0080 train loss : 0.0104

 Early Stopping : 0080 epoch
```

```
#
fig = plt.figure(figsize=(10, 4))
plt.plot(train_hist, label="Training loss")
plt.legend()
plt.show()
```



# 모델 save & load

실습

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

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

# 모델 load

실습

```
# 모델 저장
PATH ="model/model0.pth"
#torch.save(model.state_dict(), PATH)

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

제공된 model로 성능 확인

```
Net(
  (lstm): LSTM(4, 256, num_layers=2, batch_first=True)
  (fc): Linear(in_features=256, out_features=1, bias=True)
)
```

# 모델 테스트

실습

```
# 예측테스트
a=0
testX_tensor_100=testX_tensor[a*100:a*100+100]
testY_tensor_100=testY_tensor[a*100:a*100+100]
testX_tensor_100 = testX_tensor_100.to(device)
testY_tensor_100 = testY_tensor_100.to(device)

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_np = np.array(pred).reshape(-1, 1)
    pred_inverse = scaler_y.inverse_transform(pred_np)


    testY_np = testY_tensor_100.cpu().numpy()
    testY_inverse = scaler_y.inverse_transform(testY_np)
```

# 모델 테스트

```
fig = plt.figure(figsize=(15,3))
plt.plot(np.arange(len(pred_inverse)), pred_inverse, label = 'pred')
plt.plot(np.arange(len(testY_inverse)), testY_inverse, label = 'true')
plt.title("Test plot")
plt.legend()
plt.show()
```