In [ ]:
```
import numpy as np
import pandas as pd
import pandas_datareader.data as pdr
import matplotlib.pyplot as plt

import datetime

import torch
import torch.nn as nn
from torch.autograd import Variable
from tqdm.notebook import tqdm

import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder

import random
from sklearn.preprocessing import MinMaxScaler

plt.rcParams['font.family'] = 'AppleGothic'
plt.rcParams['axes.unicode_minus'] = False
```

기상 데이터 확인 및 전처리

인터넷을 돌아다니며 적당한 크기와 정보량 많고 있는 데이터 셋을 찾던 중에 따로 서베를 기상 데이터를 찾게 되어 사용하게 되었다.

In [ ]:
```
df = pd.read_csv('/Users/sona200128/Desktop/항공고/4학년/52/data/seattle-weather.csv')
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(1461, 6)
```

In [ ]:
```
df.head()
```

Out[ ]:

	date	precipitation	temp_max	temp_min	wind	weather
0	2012-01-01	0.0	12.8	5.0	4.7	drizzle
1	2012-01-02	10.9	10.6	2.8	4.5	rain
2	2012-01-03	0.8	11.7	7.2	2.3	rain
3	2012-01-04	20.3	12.2	5.6	4.7	rain
4	2012-01-05	1.3	8.9	2.8	6.1	rain

In [ ]:
```
df['date'] = pd.to_datetime(df['date'])
df['date'].replace(np.NaN, pd.NaT)
```

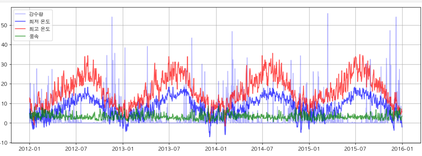
Out[ ]:
```
0       2012-01-01
1       2012-01-02
2       2012-01-03
3       2012-01-04
4       2012-01-05
           ...
1456    2015-12-27
1457    2015-12-28
1458    2015-12-29
1459    2015-12-30
1460    2015-12-31
Name: date, Length: 1461, dtype: datetime64[ns]
```

In [ ]:
```
# The column weather contains the data value in the string form and we need to predict the weather data
# so we convert it to an int as label.
df['weather']=LabelEncoder().fit_transform(df['weather'])
```

데이터 시각화

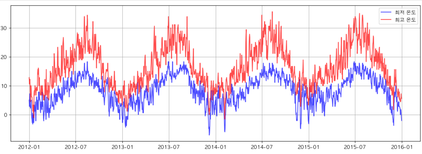
전체 데이터 시각화

In [ ]:
```
# Visualizing the temperature, observing if there's abnormal data
plt.figure(figsize = (15, 5))
plt.plot(df['date'], df['precipitation'], label='강수량', alpha=0.3, c='b')
plt.plot(df['date'], df['temp_min'], label='최저 온도', c='b', alpha=0.7)
plt.plot(df['date'], df['temp_max'], label='최고 온도', c='r', alpha=0.7)
plt.plot(df['date'], df['wind'], label= '풍속 ', c='g', alpha=0.7)
plt.legend()
plt.grid()
plt.show()
```



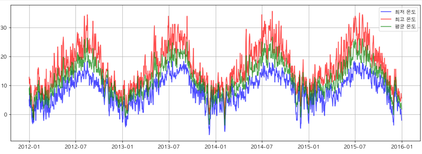
최고 온도 최저 온도 그래프

In [ ]:
```
# Visualizing the temperature, observing if there's abnormal data
plt.figure(figsize = (15, 5))
plt.plot(df['date'], df['temp_min']), label='최저 온도', c='b', alpha=0.7)
plt.plot(df['date'], df['temp_max']), label='최고 온도', c='r', alpha=0.7)
plt.legend()
plt.grid()
plt.show()
```



평균 온도 계산

In [ ]:
```
# Visualizing the temperature, observing if there's abnormal data
temp_avg = (df['temp_min'].to_numpy() + df['temp_max'].to_numpy())/2
plt.figure(figsize = (15, 5))
plt.plot(df['date'], df['temp_min']), label='최저 온도', c='b', alpha=0.7)
plt.plot(df['date'], df['temp_max']), label='최고 온도', c='r', alpha=0.7)
plt.plot(df['date'], temp_avg, label= '평균 온도 ', c='g', alpha=0.7)
plt.legend()
plt.grid()
plt.show()
```



훈련 데이터 마련

In [ ]:
```
training_set = temp_avg
```

기상 데이터 변사화

In [ ]:
```
def sliding_windows(data, seq_length):
    x = []
    y = []

    for i in range(len(data)-seq_length-1):
        _x = data[i:(i+seq_length)]
        _y = data[i+seq_length]
        x.append(_x)
        y.append(_y)

    return np.array(x),np.array(y)

sc = MinMaxScaler()
training_data = sc.fit_transform(training_set)

seq_length = 4
x, y = sliding_windows(training_data, seq_length)

train_size = int(len(y) * 0.5)
test_size = len(y) - train_size

dataX = Variable(torch.Tensor(np.array(x)))
dataY = Variable(torch.Tensor(np.array(y)))

trainX = Variable(torch.Tensor(np.array(x[0:train_size])))
trainY = Variable(torch.Tensor(np.array(y[0:train_size])))

testX = Variable(torch.Tensor(np.array(x[train_size:len(x)])))
testY = Variable(torch.Tensor(np.array(y[train_size:len(y)])))
```

LSTM 구조체

In [ ]:
```
class LSTM(nn.Module):

    def __init__(self, num_classes, input_size, hidden_size, num_layers):
        super(LSTM, self).__init__()

        self.num_classes = num_classes
        self.num_layers = num_layers
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.seq_length = seq_length

        self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size,
                            num_layers=num_layers, batch_first=True)

        self.fc = nn.Linear(hidden_size, num_classes)

    def forward(self, x):
        h_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        c_0 = Variable(torch.zeros(
            self.num_layers, x.size(0), self.hidden_size))

        # Propagate input through LSTM
        ula, (h_out, _) = self.lstm(x, (h_0, c_0))

        h_out = h_out.view(-1, self.hidden_size)

        out = self.fc(h_out)

        return out
```

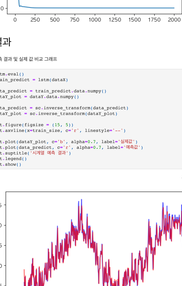
신경망 훈련

In [ ]:
```
num_epochs = 2000
learning_rate = 0.01

input_size = 1
hidden_size = 2
num_layers = 1

num_classes = 1

lstm = LSTM(num_classes, input_size, hidden_size, num_layers)

criterion = torch.nn.MSELoss()    # mean-squared error for regression
optimizer = torch.optim.Adam(lstm.parameters(), lr=learning_rate)
#optimizer = torch.optim.SGD(lstm.parameters(), lr=learning_rate)
loss_list = []
lstm.train()
for epoch in tqdm(range(num_epochs)):
    outputs = lstm(trainX)
    optimizer.zero_grad()

    # obtain the loss function
    loss = criterion(outputs, trainY)

    loss.backward()

    optimizer.step()
    loss_list.append(loss.item())
    if epoch % 100 == 0:
        print("Epoch: %d, loss: %1.5f" % (epoch, loss.item()))
```

```
  0%|          | 0/2000 [00:00<?, ?it/s]
Epoch: 0, loss: 1.54791
Epoch: 100, loss: 0.02162
Epoch: 200, loss: 0.00944
Epoch: 300, loss: 0.00493
Epoch: 400, loss: 0.00457
Epoch: 500, loss: 0.00451
Epoch: 600, loss: 0.00437
Epoch: 700, loss: 0.00425
Epoch: 800, loss: 0.00415
Epoch: 900, loss: 0.00405
Epoch: 1000, loss: 0.00399
Epoch: 1100, loss: 0.00393
Epoch: 1200, loss: 0.00388
Epoch: 1300, loss: 0.00385
Epoch: 1400, loss: 0.00381
Epoch: 1500, loss: 0.00379
Epoch: 1600, loss: 0.00377
Epoch: 1700, loss: 0.00376
Epoch: 1800, loss: 0.00375
Epoch: 1900, loss: 0.00374
```

데이터 훈련 손실 그래프프

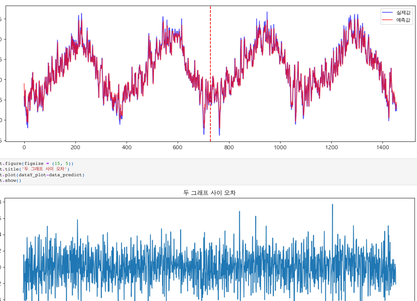
In [ ]:
```
plt.title('Loss')
plt.plot(loss_list)
plt.show()
```



결과

예측 결과 및 실제 값 비교 그래프프

In [ ]:
```
lstm.eval()
train_predict = lstm(dataX)

data_predict = train_predict.data.numpy()
dataY_plot = dataY.data.numpy()

data_predict = sc.inverse_transform(data_predict)
dataY_plot = sc.inverse_transform(dataY_plot)

plt.figure(figsize = (15, 5))
plt.axvline(x=train_size, c='r', linestyle='--')

plt.plot(dataY_plot, c='b', alpha=0.7, label='실제값')
plt.plot(data_predict, c='r', alpha=0.7, label='예측값')
plt.suptitle('시계열 예측 결과')
plt.legend()
plt.show()
```



In [ ]:
```
plt.figure(figsize = (15, 5))
plt.title('두 그래프 사이 오차')
plt.plot(dataY_plot-data_predict)
plt.show()
```