Import libraries

```
# 필요한 library가 있다면 추가하셔도 됩니다.

# load all necessary libraries
import torch
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import torch.nn.functional as F
from torch import nn, optim
%matplotlib inline

# libraries for nlp task
import regex as re
import nltk, re, string
from nltk import FreqDist
from string import punctuation
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize

#machine learning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

import torch
import torch.nn as nn
torch.manual_seed(42)

# filtering warnings
import warnings
warnings.filterwarnings('ignore')

nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
pd.set_option('display.max_columns', None)

from nltk.corpus import wordnet, stopwords
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\hyundong\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\hyundong\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\hyundong\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Preprocessing

data의 preprocessing을 진행합니다. 아래 조건에 맞게 전처리를 진행합니다. 필요한만큼 셀을 사용하시면 됩니다.

조건에 맞는 전처리를 진행하고 각각의 실행 결과(output) 창을 보여주어야합니다.

- Sentence의 문자를 모두 소문자로 변경
- stopwords의 english stopwords 제거
- WordNetLemmatizer를 이용하여 lemmatize 진행
- 정규 표현식을 사용하여 url 제거
- 정규 표현식을 사용하여 알파벳을 제외한 punctuation 포함 문자 제거
- sklearn의 LabelEncoder를 이용하여 Sentiment에 대해 label encoding 진행
- FreqDist를 사용하여 word encoding
- 제일 길이가 긴 문장을 기준으로 zero-padding 진행

```
train = pd.read_csv('./train_data.csv')
train
```

	Sentence	Sentiment
0	The GeoSolutions technology will leverage Bene...	positive
1	$ESI on lows, down $1.50 to $2.50 BK a real po...	negative
2	For the last quarter of 2010 , Componenta 's n...	positive
3	According to the Finnish-Russian Chamber of Co...	neutral
4	The Swedish buyout firm has sold its remaining...	neutral
...	...	...
5795	In 2009 , it reported net sales of approximate...	neutral
5796	H1 '08 H1 '07 Q2 '08 Q2 '07 in mln euro , unle...	neutral
5797	`` Low energy consumption and flexible loading...	neutral
5798	$SPY $MITK fast 56pc dive http://stks.co/3ffN $$	negative
5799	Investment management and investment advisory	neutral

```
train['Sentiment'].value_counts()
```

```
Sentiment
neutral     3115
positive    1838
negative     847
Name: count, dtype: int64
```

```
def cleaning_sentences(data):
    data = data.lower().split()

    stops = set(stopwords.words('english'))
    data = [word for word in data if not word in stops]

    lemmatizer = WordNetLemmatizer()
    data = [lemmatizer.lemmatize(word) for word in data]

    data = ' '.join(data)
    data = re.sub(r"http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*₩(₩),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+",' ',data)
    data = re.sub('[^a-zA-z]',' ',data)

    return data
```

```
train['Sentence'] = train['Sentence'].apply(lambda x : cleaning_sentences(x))

label_encoder = LabelEncoder()
train['Sentiment'] = label_encoder.fit_transform(train['Sentiment'])

all_words = ' '.join(train['Sentence']).split()
freq_dist = FreqDist(all_words)

sorted_word_freq = sorted(freq_dist.items(), key=lambda x: x[1], reverse=True)

word_index = {word: index+1 for index, (word,_) in enumerate(sorted_word_freq)}

train['Sentence'] = train['Sentence'].apply(lambda x : [word_index[word] for word in x.split()])

max_len = max(train['Sentence'].apply(len))
train['Sentence'] = pad_sequences(train['Sentence'], maxlen=max_len, padding='post').tolist()

train.head()
```

	Sentence	Sentiment
0	[3289, 56, 2596, 1149, 2, 1668, 39, 670, 618, ...	2
1	[4747, 2598, 3291, 137, 2599, 0, 0, 0, 0, 0, 0...	0
2	[77, 14, 529, 2, 11, 5, 1670, 1, 24, 1, 24, 17...	2
3	[62, 7, 295, 3292, 1895, 262, 88, 3, 16, 13, 1...	1
4	[217, 1150, 358, 359, 833, 41, 141, 587, 4749,...	1

Model

전처리를 진행한 데이터를 이용하여 학습을 진행하는 부분입니다. 아래의 조건들에 맞는 코드를 작성하고, 결과를 확인하면 됩니다. 필요한만큼 셀을 사용하시면 됩니다.

조건에 맞는 코드를 작성하고, 결과창(output)을 보여주어야합니다.

- train data와 validation data 분리 → train : 8 / valid : 2 비율로 분리
- target의 경우 classification이기 때문에 categorical하게 바꿔야 합니다.
- MLP 모델 구현 → 하이퍼파라미터 설정은 자유
- train / valid 과정 구현
- train set loss/ validation set loss에 대한 learning curve 출력하기
- 학습된 model은 학번_model.pth로 파일 save → ex) 20XXXXXXX_model.pth

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)

X_train, X_valid, y_train, y_valid = train_test_split(train['Sentence'],train['Sentiment'],test_size=0.2,random_state=42)

y_train = to_categorical(y_train,num_classes=3)
y_valid = to_categorical(y_valid,num_classes=3)

X_train = torch.FloatTensor(X_train.tolist()).to(device)
X_valid = torch.FloatTensor(X_valid.tolist()).to(device)
y_train = torch.FloatTensor(y_train.tolist()).to(device)
y_valid = torch.FloatTensor(y_valid.tolist()).to(device)

print(X_train.shape, y_train.shape, X_valid.shape, y_valid.shape)
```

```
     torch.Size([4640, 42]) torch.Size([4640, 3]) torch.Size([1160, 42]) torch.Size([1160, 3])
```

```
model = nn.Sequential(
          nn.Linear(42, 10),
          nn.ReLU(),
          nn.Linear(10, 10),
          nn.ReLU(),
          nn.Linear(10, 10),
          nn.ReLU(),
          nn.Linear(10, 10),
          nn.ReLU(),
          nn.Linear(10, 3),
          nn.Softmax(dim=1)
          ).to(device)

criterion = torch.nn.CrossEntropyLoss(weight=None, reduction='mean').to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=0.0001)

train_losses = []
valid_losses = []

epochs = 13920 # 나누기 1 epoch = 4640, 13920 = 3 epoch

for epoch in range(epochs):
    optimizer.zero_grad()
    hypothesis = model(X_train)
    cost = criterion(hypothesis, y_train)
    cost.backward()
    optimizer.step()

    if epoch % 100 == 0:
        with torch.no_grad():
            train_hypothesis = model(X_train)
            train_loss = criterion(train_hypothesis, y_train)
            train_losses.append(train_loss.item())

            valid_hypothesis = model(X_valid)
            valid_loss = criterion(valid_hypothesis, y_valid)
            valid_losses.append(valid_loss.item())

            print(f'Epoch {epoch}, Train Loss: {train_loss.item()}, Valid Loss: {valid_loss.item()}')

# Learning curve
plt.plot(range(0, epochs, 100), train_losses, label='Train')
plt.plot(range(0, epochs, 100), valid_losses, label='Validation')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
Epoch 0, Train Loss: 1.165626883506775, Valid Loss: 1.1629374027252197
Epoch 100, Train Loss: 1.0905544757843018, Valid Loss: 1.0900086164474487
Epoch 200, Train Loss: 1.061585783958435, Valid Loss: 1.0564857721328735
Epoch 300, Train Loss: 1.0419607162475586, Valid Loss: 1.0240249633789062
Epoch 400, Train Loss: 1.0339853763580322, Valid Loss: 1.0116071701049805
Epoch 500, Train Loss: 1.0308290719985962, Valid Loss: 1.008539080619812
Epoch 600, Train Loss: 1.0285065174102783, Valid Loss: 1.0062123537063599
Epoch 700, Train Loss: 1.0260850191116333, Valid Loss: 1.0035064220428467
Epoch 800, Train Loss: 1.0235334634780884, Valid Loss: 1.0011628866195679
Epoch 900, Train Loss: 1.0214219093322754, Valid Loss: 1.0002410411834717
Epoch 1000, Train Loss: 1.019686222076416, Valid Loss: 0.9997731447219849
Epoch 1100, Train Loss: 1.0186487436294556, Valid Loss: 0.9992079138755798
Epoch 1200, Train Loss: 1.0178979635238647, Valid Loss: 0.9987949728965759
Epoch 1300, Train Loss: 1.0173600912094116, Valid Loss: 0.9986332058906555
Epoch 1400, Train Loss: 1.0169336795806885, Valid Loss: 0.998660683631897
Epoch 1500, Train Loss: 1.016578197479248, Valid Loss: 0.9987576007843018
Epoch 1600, Train Loss: 1.0162729024887085, Valid Loss: 0.998866856098175
Epoch 1700, Train Loss: 1.0160044431686401, Valid Loss: 0.9989563226699829
Epoch 1800, Train Loss: 1.0157641172409058, Valid Loss: 0.9990341663360596
Epoch 1900, Train Loss: 1.0155433416366577, Valid Loss: 0.9991104602813721
Epoch 2000, Train Loss: 1.0153353214263916, Valid Loss: 0.9991630911827087
Epoch 2100, Train Loss: 1.0151373147964478, Valid Loss: 0.999198853969574
Epoch 2200, Train Loss: 1.01494300365448, Valid Loss: 0.999240517616272
Epoch 2300, Train Loss: 1.0147515535354614, Valid Loss: 0.9992555975914001
Epoch 2400, Train Loss: 1.0145522356033325, Valid Loss: 0.999247133731842
Epoch 2500, Train Loss: 1.0143712759017944, Valid Loss: 0.9991846680641174
Epoch 2600, Train Loss: 1.0141968727111816, Valid Loss: 0.9990923404693604
Epoch 2700, Train Loss: 1.014020562171936, Valid Loss: 0.9989758729934692
Epoch 2800, Train Loss: 1.013834834098816, Valid Loss: 0.998843252658844
Epoch 2900, Train Loss: 1.0136626958847046, Valid Loss: 0.9987555146217346
Epoch 3000, Train Loss: 1.0135035514831543, Valid Loss: 0.9986955523490906
Epoch 3100, Train Loss: 1.013357281684875, Valid Loss: 0.998654305934906
Epoch 3200, Train Loss: 1.0132145881652832, Valid Loss: 0.9986012578010559
Epoch 3300, Train Loss: 1.0130751132965088, Valid Loss: 0.9985482096672058
Epoch 3400, Train Loss: 1.0129438638687134, Valid Loss: 0.9985206127166748
Epoch 3500, Train Loss: 1.0128214359283447, Valid Loss: 0.9985085129737854
Epoch 3600, Train Loss: 1.0127058029174805, Valid Loss: 0.9984920024871826
Epoch 3700, Train Loss: 1.0125927925109863, Valid Loss: 0.9984768629074097
Epoch 3800, Train Loss: 1.0124828815460205, Valid Loss: 0.9984592795372009
Epoch 3900, Train Loss: 1.0123761892318726, Valid Loss: 0.9984366297721863
Epoch 4000, Train Loss: 1.0122723579406738, Valid Loss: 0.9984144568443298
Epoch 4100, Train Loss: 1.0121705532073975, Valid Loss: 0.9983964562416077
Epoch 4200, Train Loss: 1.0120692253112793, Valid Loss: 0.9983773827552795
Epoch 4300, Train Loss: 1.0119694471359253, Valid Loss: 0.998363733291626
Epoch 4400, Train Loss: 1.0118707418441772, Valid Loss: 0.9983516335487366
Epoch 4500, Train Loss: 1.0117701292037964, Valid Loss: 0.9983442425727844
Epoch 4600, Train Loss: 1.011673092842102, Valid Loss: 0.9983317255973816
Epoch 4700, Train Loss: 1.0115786790847778, Valid Loss: 0.9983179569244385
Epoch 4800, Train Loss: 1.0114850997924805, Valid Loss: 0.9983043670654297
Epoch 4900, Train Loss: 1.0113929510116577, Valid Loss: 0.9982871413230896
Epoch 5000, Train Loss: 1.0113028287887573, Valid Loss: 0.9982714653015137
Epoch 5100, Train Loss: 1.0112147331237793, Valid Loss: 0.9982578754425049
Epoch 5200, Train Loss: 1.0111263990402222, Valid Loss: 0.998248815536499
Epoch 5300, Train Loss: 1.011038064956665, Valid Loss: 0.9982377886772156
Epoch 5400, Train Loss: 1.010948657989502, Valid Loss: 0.9982271194458008
Epoch 5500, Train Loss: 1.010859489440918, Valid Loss: 0.99821537733078
Epoch 5600, Train Loss: 1.0107717514038086, Valid Loss: 0.9982047080993652
Epoch 5700, Train Loss: 1.0106842517852783, Valid Loss: 0.9982004165649414
Epoch 5800, Train Loss: 1.0105966329574585, Valid Loss: 0.9981914758682251
Epoch 5900, Train Loss: 1.0105100870132446, Valid Loss: 0.9981827139854431
Epoch 6000, Train Loss: 1.0104254484176636, Valid Loss: 0.9981715679168701
Epoch 6100, Train Loss: 1.01034414768219, Valid Loss: 0.998157262802124
Epoch 6200, Train Loss: 1.0102601051330566, Valid Loss: 0.9981551766395569
Epoch 6300, Train Loss: 1.0101780891418457, Valid Loss: 0.998139500617981
Epoch 6400, Train Loss: 1.0100975036621094, Valid Loss: 0.998117983341217
Epoch 6500, Train Loss: 1.0100177526474, Valid Loss: 0.9980934858322144
Epoch 6600, Train Loss: 1.0099374055862427, Valid Loss: 0.9980770945549011
Epoch 6700, Train Loss: 1.009857177734375, Valid Loss: 0.99806147813797
Epoch 6800, Train Loss: 1.0097771883010864, Valid Loss: 0.9980452060699463
Epoch 6900, Train Loss: 1.0096999406814575, Valid Loss: 0.9980309009552002
Epoch 7000, Train Loss: 1.0096237659454346, Valid Loss: 0.9980181455612183
Epoch 7100, Train Loss: 1.0095479488372803, Valid Loss: 0.9980058073997498
Epoch 7200, Train Loss: 1.0094730854034424, Valid Loss: 0.9979947805404663
Epoch 7300, Train Loss: 1.0094000101089478, Valid Loss: 0.9979798197746277
Epoch 7400, Train Loss: 1.0093282461166382, Valid Loss: 0.997959554195404
Epoch 7500, Train Loss: 1.0092580318450928, Valid Loss: 0.9979358315467834
Epoch 7600, Train Loss: 1.0091884136199951, Valid Loss: 0.9979133605957031
Epoch 7700, Train Loss: 1.0091195106506348, Valid Loss: 0.9978885054588318
Epoch 7800, Train Loss: 1.0090522766113281, Valid Loss: 0.997859537601471
Epoch 7900, Train Loss: 1.0089855194091797, Valid Loss: 0.9978302121162415
Epoch 8000, Train Loss: 1.00892174243927, Valid Loss: 0.9978020191192627
Epoch 8100, Train Loss: 1.008860468864441, Valid Loss: 0.997776448726654
```