D:\myprojects\nlp\word2vec\cbow.py

```python
# CBOW(Continumous Bag of Words) WOrd to Vector Embedding

### conda update -n base -c defaults conda
# conda create -n word2vec nltk pandas
# conda activate word2vec
# conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia
#       또는 conda install pytorch torchvision torchaudio cpuonly -c pytorch
# (word2vec) python cbow.py

# https://didu-story.tistory.com/101

import nltk
from os import path
from tqdm import tqdm
import re, sys
import pandas as pd

# Clean sentences
def clean_text(text) :
            #change capital letters to lower
    text = ' '.join(word.lower() for word in text.split(" "))
            # re.sub('패턴', '바꿀문자열', '문자열', 바꿀횟수)
    text = re.sub(r"([.,!?])", r" \1 ", text)
    text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)
    return text

#============================================================================
# Tokenizer downloading and setting
#============================================================================
        # install nltk's data
if not path.exists('./tokenizers/punkt/english.pickle'):
    #nltk.download('popular')
    nltk.download()

        # set tokenizer
tokenizer = nltk.data.load('./tokenizers/punkt/english.pickle')

#============================================================================
# data sentenses reading and cleaning
#============================================================================
            # read data text for training
            # https://www.gutenberg.org/files/84/84-h/84-h.htm (frankenstein book)
with open("./book.txt") as fp:
    book = fp.read()

            # Split the raw text book into sentences
            # tokenize the data
sentences = tokenizer.tokenize(book)

            # check the read sentences, and print some sentences
print (f'totally {len(sentences)} sentences')
for i in range(5):
    sentence = sentences[i]
    print (f"*** sentence[{i}]: {sentence}\n")

        # check the cleaned sentences
cleaned_sentences = [clean_text(sentence) for sentence in sentences]
```

```
print('='*80)
for i in range(5):
    sentence = cleaned_sentences[i]
    print (f"*** cleaned sentence[{i}]: {sentence}\n")

#=====================================================================
# CBOW data creation
#=====================================================================
MASK_TOKEN = "<MASK>"
windowSize = 5          # target token 앞뒤로 각각 5개씩, 총 앞뒤로 주변에 10개의 token을 사용한다

# 각 문장의 토큰 리스트를 순회하면서 지정된 크기(길이 11개)의 window로 묶어주기
        # lambda 매개변수 : 표현식의 예:
        # matrix = [[1, 2], [3, 4], [5, 6]]
        # squared = [num ** 2 for row in matrix for num in row]
        # print(squared)  # Output: [1, 4, 9, 16, 25, 36]
        # 2차원-->1차원으로 펴기
flatten = lambda outer_list: [item for inner_list in outer_list for item in inner_list]
        # nltk.ngrams([1,2,3,4,5], 3)) ===> [(1, 2, 3), (2, 3, 4), (3, 4, 5)]
        # 전체 cleaned_sentenses로부터,
        # 각 문장에서 11개의 단어(토큰)로 이루어진 리스트를 가져와라
windows = flatten([list(nltk.ngrams([MASK_TOKEN] * windowSize + sentence.split(' ') + [MASK_TOKEN] * windowSize,
                                    windowSize * 2 + 1)) \
                                    for sentence in tqdm(cleaned_sentences)])

print('='*80)
print(' First 10 windows data for trainig')
print('='*80)
for i in range(10):
    print(windows[i])

# CBOW 데이터로 만들어주기
data = []
for window in tqdm(windows):
    target_token = window[windowSize]   # windowSize번째, 즉 5번째 토큰이 target token이 됨
    context = []
                                        # 가운데 위치한 target token을 제외한 주변 10개ㅏ 토큰을 구함
    for i, token in enumerate(window):
        if token == MASK_TOKEN or i == windowSize:
            continue
        else:
            context.append(token)
    # '구분자'.join(리스트)
    data.append([' '.join(token for token in context), target_token])


# DataFrame의 형태로 변환
cbow_data = pd.DataFrame(data, columns=["context", "target"])
print('='*80)
print(' First 20 CBOW data for trainig')
print('='*80)
print(cbow_data[:20])

#=====================================================================
# CBOW Training
#=====================================================================

import torch
```

```
116 import torch.nn as nn
117 import torch.optim as optim
118
119 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
120 print(f"\n***** Device : {device}")
121
122 if device != 'cpu' :    # 'cuda'
123     GPU = True
124 else :
125     GPU = False
126
127 class CBOW(nn.Module):
128     def __init__(self, vocab_size, embedding_dim):
129         super(CBOW, self).__init__()
130         if GPU:
131             self.embeddings = nn.Embedding(vocab_size, embedding_dim).to(device)  # GPU로
    이동
132             self.linear = nn.Linear(embedding_dim, vocab_size).to(device)  # GPU로 이동
133         else :
134             self.embeddings = nn.Embedding(vocab_size, embedding_dim)
135             self.linear = nn.Linear(embedding_dim, vocab_size)
136
137     def forward(self, inputs):
138         embedded = self.embeddings(inputs).mean(dim=0)
139         out = self.linear(embedded)
140         log_probs = torch.log_softmax(out, dim=-1)
141         return log_probs
142
143 def make_context_vector(context, word_to_ix):
144     idxs = [word_to_ix[w] for w in context]
145     if GPU:
146         return torch.tensor(idxs, dtype=torch.long).to(device)
147     else :
148         return torch.tensor(idxs, dtype=torch.long)
149
150
151 # 가정: cbow_data DataFrame과 word_to_ix 사전이 이미 준비되어 있음
152 vocab = set(word for _, row in cbow_data.iterrows() for word in row['context'].split(' ')
    + list(row['target']))
153 print('='*80)
154 print(f"vocaburary set : {vocab}")
155 print('='*80)
156 vocab_size = len(vocab)
157 print(f"vocaburary size: {vocab_size}")
158
159 embedding_dim = 50  # 임베딩 차원 설정
160
161 word_to_ix = {word: i for i, word in enumerate(vocab)}
162 print(f"word_to_ix: {word_to_ix}")
163
164 model = CBOW(vocab_size, embedding_dim)
165 if GPU :
166     model = model.to(device)
167
168 loss_function = nn.NLLLoss()
169 optimizer = optim.SGD(model.parameters(), lr=0.001)
170
171 # 학습 시작
172 SHOW_FLAG = False
173 WRITE_FLAG = False
```

```
174
175 print("*"*80)
176 for epoch in tqdm(range(300),desc="Epoch 진행률", unit="iter"):  # 에포크 설정
177     total_loss = 0
178     sentence_no = 0
179     for _, row in tqdm(cbow_data.iterrows(),desc="Sentence 진행률", unit="iter"):     # 전체 데이터에 대하여
180         context_vec = make_context_vector(row['context'].split(' '), word_to_ix)
181 
182         target_vector = [0 for i in range(vocab_size)]
183         target_vector[word_to_ix[row['target']]] = 1.
184 
185         if GPU :
186             target_value = torch.tensor(target_vector, dtype=torch.long).to(device)
187         else :
188             target_value = torch.tensor(target_vector, dtype=torch.long)
189 
190         model.zero_grad()                   # 그래디언트 초기화
191 
192         log_probs = model(context_vec)      # 실제 출력값
193 
194         if SHOW_FLAG :
195             print('='*80)
196             print(f'context vector : {context_vec}')
197             print(f'targe_value tensor 값 [{len(target_value)}] 개: {target_value}')
198             print(f'log_probabilities tensor 값 [{len(log_probs)}] 개: {log_probs}')
199             print('='*80)
200 
201         #loss = loss_function(log_probs, torch.tensor([word_to_ix[row['target']]], dtype=torch.long))
202         loss = loss_function(log_probs, target_value)
203         loss.backward()
204         optimizer.step()
205 
206         total_loss += loss.item()
207         #print(f"Sentence [{sentence_no}]'s total_loss : {total_loss}")
208         sentence_no += 1
209         #sys.exit()
210     print(f'Epoch {epoch}: Total Loss: {total_loss}\n')
211 
212 
213 # 임베딩 값 추출
214 #embeddings = model.embeddings.weight.data
215 embeddings = model.embeddings.weight.detach()
216 
217 # embedding 단어를 리스트 type으로 저장
218 i=0
219 word = [];
220 for key, value in word_to_ix.items():
221     word.append(key)
222     i += 1
223 
224 print("*"*80, "done.")
225 with open("./embed_result.txt","w") as fp:
226     for ndx, embed in tqdm(enumerate(embeddings)):
227         embedded_word = word[ndx]
228         if embedded_word in ['happy', 'pleasant', 'furious', 'sad', 'glad', 'angry']:
229             print(f"embeddings[{ndx}], {embedded_word} : {embed}")
230         # embedding 결과를 파일로 저장
231         if WRITE_FLAG :
```

```
            fp.write("["+str(embedded_word)+"] : ")
            fp.write(str(embed.numpy())+"\n")

print("*"*80, "done.")


'''
[참고자료]
Euclidean Similarity 계산의 예
import torch

# 두 텐서 정의
tensor_a = torch.tensor([1.0, 2.0, 3.0])
tensor_b = torch.tensor([4.0, 5.0, 6.0])

# 유클리디언 거리 계산
euclidean_distance = torch.norm(tensor_a - tensor_b)

print(f'Euclidean Distance: {euclidean_distance.item()}')



# Cosine Similarity 계산의 예
import torch.nn.functional as F

# 두 텐서 정의
tensor_a = torch.tensor([1.0, 2.0, 3.0])
tensor_b = torch.tensor([4.0, 5.0, 6.0])

# 코사인 유사도 계산
cosine_sim = F.cosine_similarity(tensor_a.unsqueeze(0), tensor_b.unsqueeze(0))

print(f'Cosine Similarity: {cosine_sim.item()}')
'''
```