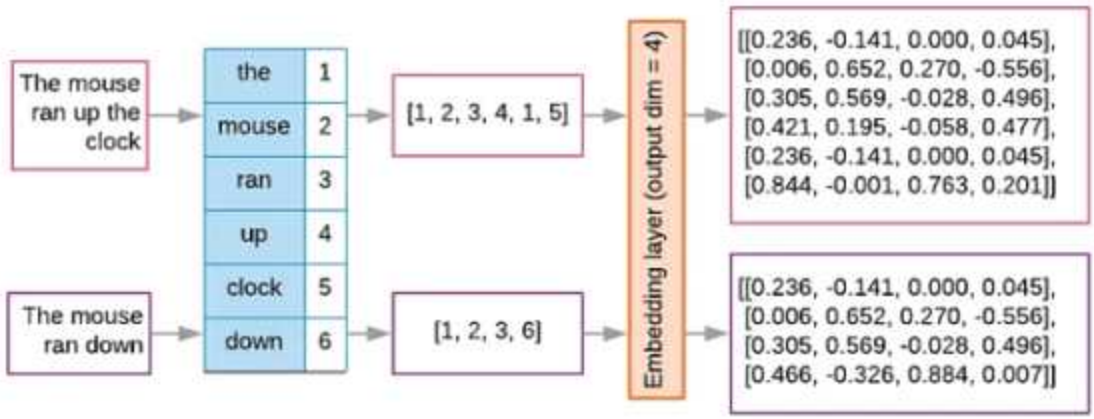
Text Embedding

◉ Embedding : 끼워넣기

- n개 원소 입력 도메인의 element 각각에 대하여 m개원소 도메인으로의 mapping 규칙
- 입력 단어(토큰)를 m 차원 숫자 공간상에서 하나의 점으로 표현
- 입력 단어(토큰)의 숫자 표현과 표현간의 거리를 이용하여 연산

◉ 컴퓨터는 문자(단어) 형태의 데이터로는 사람처럼 단어의 의미 정보를 처리하기 어려움

◉ 임베딩의 예:



(https://geoffrey-geofe.medium.com/tokenization-vs-embedding-understanding-the-differences-and-their-importance-in-nlp-b62718b5964a)

◉ 적절한 임베딩을 통하여 의미 있는 벡터 연산이 가능하다!

king − man + woman ≈ queen
Queen - Woman + Man ≈ King
Seoul - Korea + USA ≈ Washington

◉ Text Embedding

- 사람에게 편한 문자 표현(Word, Token)을 기계가 처리할 수 있는 숫자 벡터(Numerical Vector)로 변환하는 것
- 변환된 숫자 벡터가, 사람이 사용하는 단어처럼 단어간 의미를 계산할 수 있도록 변환할 필요가 있음
- 관심사항 : 임베딩 방법, 단어간 거리, 임베딩 결과의 비쥬얼 표현, 임베딩 활용

1. Word Embedding 방법 3가지

[1/3] Bag of Words 임베딩

- 입력된 문장들을 Stemming 한 후, 각 단어의 frequency를 계산하여 *출현 빈도가 높은것 부터* code를 할당하는 방법

```
from nltk.stem import SnowballStemmer
from nltk.tokenize import word_tokenize

text = 'We are lucky to live in an age in which we are still making discoveries'

# tokenization - splitting text into words
words = word_tokenize(text)
print(words)
# ['We', 'are', 'lucky', 'to', 'live', 'in', 'an', 'age', 'in', 'which',
# 'we', 'are', 'still', 'making', 'discoveries']

stemmer = SnowballStemmer(language = "english")
stemmed_words = list(map(lambda x: stemmer.stem(x), words))
print(stemmed_words)
# ['we', 'are', 'lucki', 'to', 'live', 'in', 'an', 'age', 'in', 'which',
# 'we', 'are', 'still', 'make', 'discoveri']


import collections
bag_of_words = collections.Counter(stemmed_words)   # 출현 빈도 카운팅
print(bag_of_words)
# {'we': 2, 'are': 2, 'in': 2, 'lucki': 1, 'to': 1, 'live': 1,
# 'an': 1, 'age': 1, 'which': 1, 'still': 1, 'make': 1, 'discoveri': 1}

# 이후, 출현빈도 순으로 코드 일련번호 할당
```

==> 변환된 벡터의 표현 간에, 단어나 문장의 의미(semantic)에 대한 고려가 포함되지 않는 단점이 있음.

◉ One-hot encoding : 하나의 데이터만 1의 값을 갖고 나머지는 모두 '0'의 값을 갖는 encoding

[2/3] TF-IDF (Term Frequency, Inverse Document Frequency) 임베딩

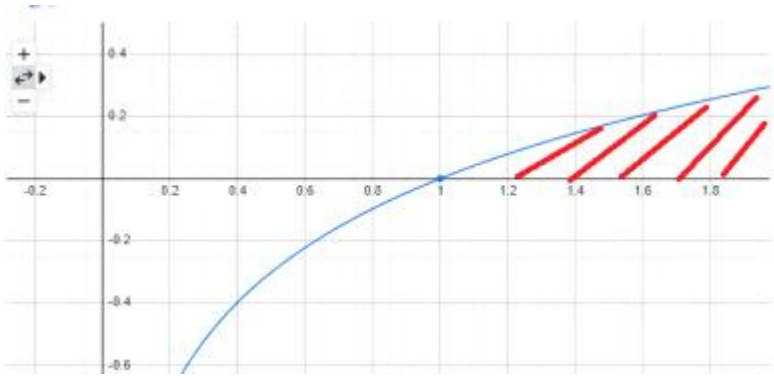
: 문서내 단어의 중요도를 개선하여 상위 코드를 할당

TF(t,d) = (t's occurrence number in document d) / (number of terms in document d)

: 한 문서 내에서 특정 term이 나타난 빈도 (빈도가 클수록 중요도가 높다고 판단)
이 하나의 지표로는, 자주 사용되지만 큰 정보가 담기지 않고 많이 반복되는 단어가 높은 값을 가지는 문제가 있음 ("I", "a", "an", ...)

IDF(t,D) = log (total # of documents in corpus D / # of documents containing term t)

: 해당 term이 사용된 term이 적을수록 큰 값을 가지므로,
"I" 나 "a"처럼 많이 사용되지만 큰 의미가 없는 단어들이 낮은 값을 갖도록 함



IDF(t,D) 의 값 범위

이 둘을 고려하면, **TF-IDF(t, d, D) = TF(t, d) x IDF(t, d)**

=> 여전히 벡터간의 의미(semantic)을 담지는 못함

=> 영어에서 470,000개 정도의 단어 크기를 고려하면 50개 단어로 된 문장을 고려해도, one-hot-vector 방식을 쓰면, 한 문장내 평균 99.99%의 값이 0으로 표현되는 낭비 발생

※ BoW와 TF-IDF 모두 단어 표현을 위한 One-hot 인코딩에 따른 Sparcce Matrix의 사용으로 *메모리 낭비가 심하고, 대량데이터에 대한 병렬처리가 어려우며, 학습을 통한 개선의 여지가 없다*

=> 보다 dense한 코딩 방식이 필요함

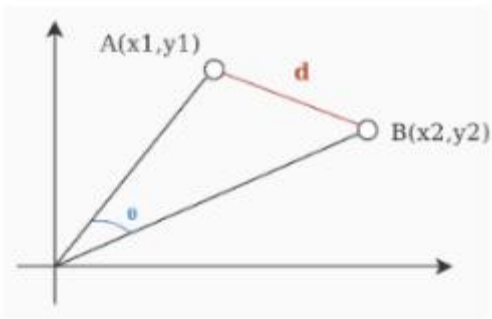
[3/3] Word2Vec 임베딩

- Dense representation embedding 방법
- “Efficient Estimation of Word Representations in Vector Space” by Google (2013)
- 두가지 구현 방법이 가능 : CBOW, Skip-gram

◉ Sliding Window (문맥을 위해 참고하는 단어의 범위를 설정)

(https://comlini8-8.tistory.com/6)

◉ 벡터간 거리(Distance)



Euclidean distance and Cosine similarity
(https://alexn.org/blog/2012/01/16/cosine-similarity-euclidean-distance/)

◉ CBOW

=> The CBOW model is a shallow neural network with three layers: an input layer, a hidden layer, and an output layer. The input layer represents the context words in a sentence. The hidden layer learns the word embeddings. The output layer predicts the target word.

=> Projection layer의 node 수 만큼의 벡터 크기를 사용

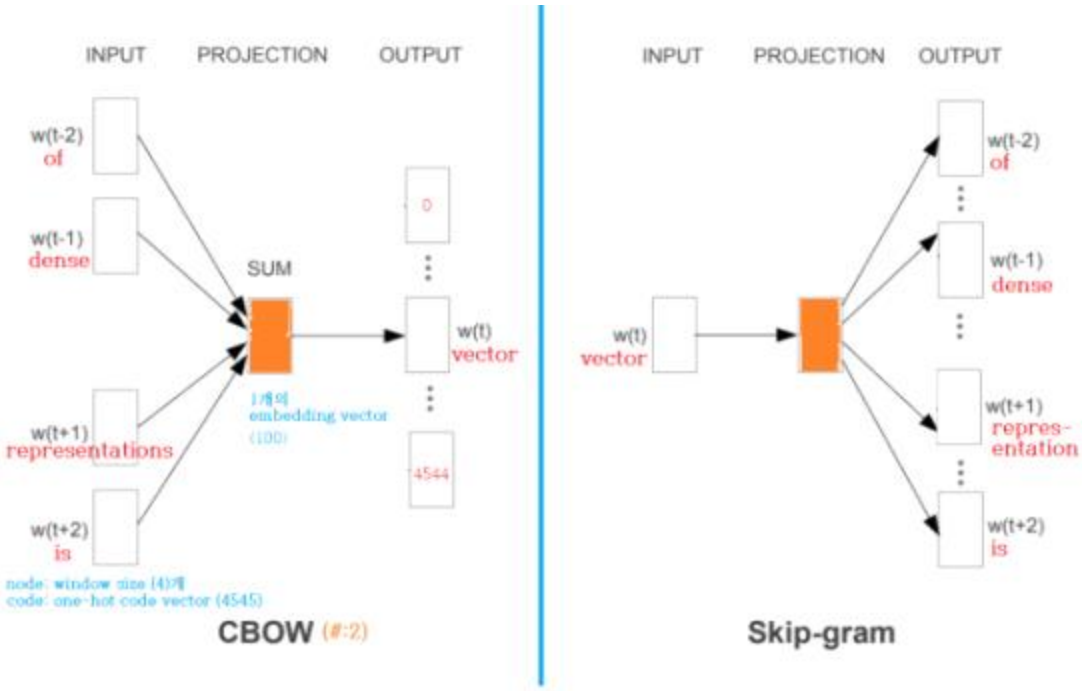
=> 학습 후, “Projection”의 출력을 numerical verctor의 코드로 사용

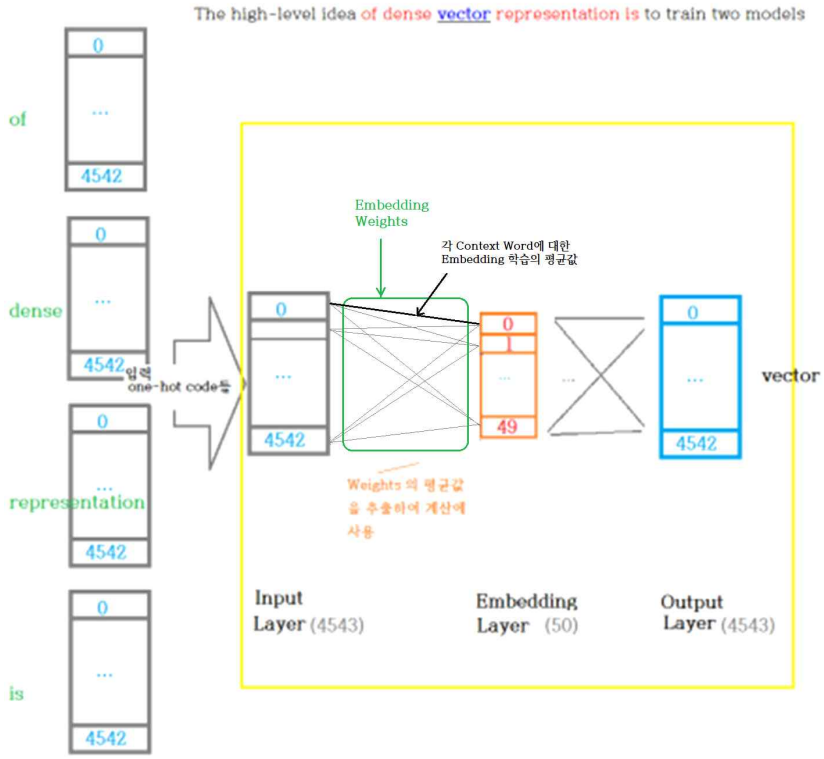
=> 문장 내 앞뒤 n개의 단어로 학습하므로 어느정도 단어의 의미정보가 코드에 반영됨.

=> 문장 전체에 대한 인코딩으로 보긴 어려움

◉ 학습데이터 세트의 동일 단어들에 대한 projection 값들의 평균을 embedding 벡터값으로 사용

ex) The high-level idea of dense vector representation is to train two models






```
# 각 Embedding 단어들에 대한 임베딩 벡터를 추출 및 평균 내어 임베딩 벡터값으로 함.
# 이 평균 벡터가 우측의 Linear Layer의 계산에 사용된다.
```

CBOW 모델 예 (window size:2, embedding_dimension:50)

◉ Embedding 평균의 예

[input 벡터]

: tensor([3618, 4355], device='cuda:0')s] 일 때,

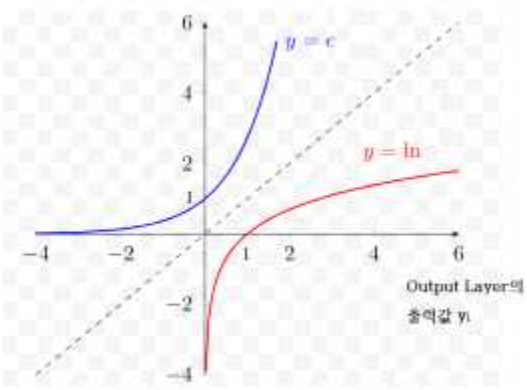
[평균 전 임베딩 값 2개]

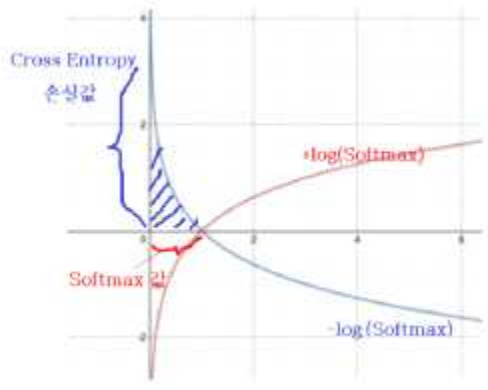
```
tensor([[ 1.0894, -0.2640,  0.1521,  0.1083, -0.0790, -0.5373, -0.4871,  1.6117,
          0.9882,  0.5591,  0.7840, -1.0096, -0.7190,  0.8490, -0.2897,  0.3536,
          0.2930,  1.1135,  0.2824, -0.5500,  0.4860, -0.3755,  2.0494, -0.8836,
          0.3544,  0.4503, -0.2181, -1.1415, -1.4641,  0.5799,  0.7792, -0.9388,
          0.0931, -1.3051, -0.4728, -0.4855, -0.4502, -1.8999,  0.2563, -0.5456,
          0.2819,  0.9200, -0.3517,  0.3692,  0.0248, -1.7956,  0.9501, -0.2243,
                                                              -0.1866, -0.2668],
        [-1.8836,  1.0856, -0.0329,  0.1300,  0.2992,  0.5211,  0.8170, -0.0690,
          0.4863, -0.8673, -0.9574,  0.9351,  0.4939,  0.4795,  0.6374,  0.7354,
         -0.1205,  0.7321,  0.8634,  0.1118,  1.6181, -0.8485,  0.6192, -1.7266,
          0.4752, -0.5349,  0.0413,  0.1409, -0.6159, -0.3798, -0.0785,  1.0609,
          0.8906,  1.3349, -2.2722, -0.3278, -0.1432, -1.0270,  0.1071, -1.0994,
          0.3042,  0.6860, -0.8365, -0.5481,  0.2190, -2.7890,  0.6672,  1.4178,
          0.9504,  0.1889]], device='cuda:0', grad_fn=<EmbeddingBackward0>)
```

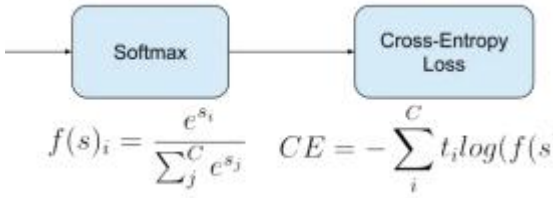
[평균 후 임베딩 값 1개]

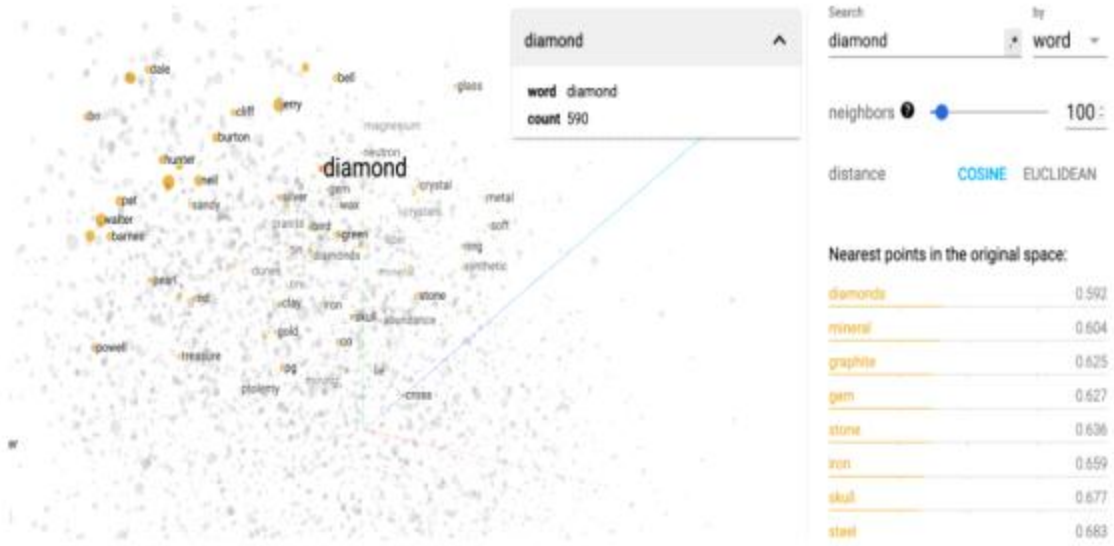
```
tensor([-0.3971,  0.4108,  0.0596,  0.1192,  0.1101, -0.0081,  0.1649,  0.7714,
         0.7373, -0.1541, -0.0867, -0.0373, -0.1126,  0.6643,  0.1738,  0.5445,
         0.0863,  0.9228,  0.5729, -0.2191,  1.0520, -0.6120,  1.3343, -1.3051,
         0.4148, -0.0423, -0.0884, -0.5003, -1.0400,  0.1001,  0.3504,  0.0610,
         0.4919,  0.0149, -1.3725, -0.4066, -0.2967, -1.4634,  0.1817, -0.8225,
         0.2931,  0.8030, -0.5941, -0.0894,  0.1219, -2.2923,  0.8087,  0.5967,
         0.3819, -0.0389], device='cuda:0', grad_fn=<MeanBackward1>)
```

◉ Cross Entropy 손실 함수









embedding projector를 이용한 visualization 의 예
(https://jiho-ml.com/weekly-nlp-5/)

◉ 개선된 방법 :
- Negative Sampling (타겟 단어의 윈도우 내에 존재하지 않는 단어세트로 학습)
  네거티브 샘플은 2~20 사이를 사용
- Sub Sampling : 빈도가 큰 단어는 학습 기회를 줄이고, 빈도가 낮은 단어는 모두 학습시킴

◉ 여전히 내재된 문제점 :
- 총 단어(말뭉치)가 커지면 학습시간이 오래 걸린다

- 다른 의미의 단어(다의어)에 대한 같은 Word Embedding 값의 사용
  (예: 인도 카레를 먹고 싶어서 인도 가까이에 자전거를 세웠다.
  The club I tried yesterday was great! (골프클럽, 클럽하우스, 클럽 샌드위치,,,)
  ==> 어떤 word embedding 값을 사용할 것인가?

- 여전히 전체 문장내에서 모든 Word의 문장 내 위치에 대한 정보가 충분히 반영되지 않는다
  (이웃한 word-pair 간 정보만 사용)

※ CBOW Coding 예

("What I can not create, I do not understand", by Richard Feynman. 1988)

[1] CBOW.PY 소스

```
# 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)   # 모든 마침표, 쉼표, 느낌표, 물음표 다음에 공백을 추가

    text =re.sub(r"[^a-zA-Z.,!?]+",r" ",text)# 알파벳(a-z, A-Z), 마침표, 쉼표, 느낌표, 물음표를
                                             #     제외한 모든 문자를 ' '(space)로 교체
    return text
#=============================================================================
# Tokenizer downloading and setting
#=============================================================================
            # install nltk's data (영어 텍스트를 문장단위로 분할 하는 사전 훈련된 모델)
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 =2          # target token 앞뒤로 각각 2개씩, 총 앞뒤로 주변에 4개의 token을 사용한다
        # 각 문장의 토큰 리스트를 순회하면서 지정된 크기(길이 5개)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,

        # cleaned_sentences로부터 가져온 각 문장에서 5개의 단어(토큰)로 이루어진 windowx 리스트를 가
져와라

        # cleaned_sentences에서 sentence를 하나씩 가져와서 nltk.ngram을 통해
        # sentence 맨 앞과 맨 뒤에 각각 2개씩의 <MASK> 토큰을 추가한 후, 5개 단어 길이의 wind리스트
로
        # 바꾼 후, flatten 하라
        # I iove you --> [<MASK>, <MASK>, 'I', 'love', 'you',<MASK>, <MASK>]
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 3번째 토큰이 target token이 됨
    context =[]
                                       # 가운데 위치한 target token을 제외한 주변 10개ㅏ 토큰을 구
함
    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])

# data를 column 제목이 있는 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
#=============================================================================
importtorch
importtorch.nn as nn
importtorch.optim as optim
device =torch.device("cuda"if torch.cuda.is_available()else "cpu")
print(f"\n***** Device : {device}")
if device !='cpu':    # 'cuda'
    GPU =True
else :
    GPU =False

class CBOW(nn.Module):
    def __init__(self,vocab_size,embedding_dim):
        super(CBOW,self).__init__()
        if GPU:
            # embedding layer 생성 함수 정의 (one-hot encoded vector->dense embedding layer)
            # vocab_size * embedding_dim (4543*100) 크기의 lookup _table 내부적으로 생성
            self.embeddings =nn.Embedding(vocab_size,embedding_dim).to(device)  # GPU로 이동
            # Linear transformation layer 생성 함수 정의 (임베딩 디멘젼->vocaburary size layer)
            # linear layer는 tensorflow의 dense layer와 유사
             self.linear =nn.Linear(embedding_dim,vocab_size).to(device)  # GPU로 이동

        else :
```

```
            self.embeddings =nn.Embedding(vocab_size,embedding_dim)
            self.linear =nn.Linear(embedding_dim,vocab_size)

    # PyTorch에서 forward 함수는 모델 객체를 통해 입력 데이터를 전달할 때 자동으로 호출됩니다.
    # 이는 PyTorch의 nn.Module 기본 클래스에 구현된 __call__ 메소드에 의해 내부적으로 처리됩니다.
    # 사용자는 직접 forward 메소드를 호출하지 않습니다. 대신, 모델 인스턴스를 함수처럼 사용하여
    # 입력 데이터를 전달하면, PyTorch가 내부적으로 forward 함수를 호출합니다.
    # forward 메소드 내에서 정의된 계산이 수행되고, 최종적으로 출력 데이터가 반환됩니다.
    def forward(self,inputs):
        #print(f"\n[input 벡터들]\n{inputs}")    # 예: tensor([3618, 4355]
        #embedded = self.embeddings(inputs)      # Embedding Layer로 부터 weights 값(= embedding
value) 추출
        #print(f"[평균전 임베딩]\n{embedded}")

        embedded =self.embeddings(inputs).mean(dim=0)  # Embedding weight를 입력 텐서별로 추출한
후 평균냄
        #print(f"[평균 후 임베딩]\n{embedded}")

        out =self.linear(embedded)     # 임베딩된 벡터를 입력으로 받아 출력 차원으로 매핑함.
                                       # 각 단어에 대한 스코어를 출력
        log_probs =torch.log_softmax(out,dim=-1)  # 각 Word가 target 단어일 확률값을 Log(SoftMax
값)으로 반환
                        # [-8.212, 1.3256, ......  ] : softmax값이 1보다 작으므로, log를 취한 결과
값 50개 모두 음수값
        return log_probs
# word_ti_ix 집합(사전)을 참고로 하여 index 값들로 이루어진 context vector들을 리턴
# 예: [32, 2324, 156, 33] 로 변환
def make_context_vector(context,word_to_ix):
    idxs =[word_to_ix[w]for w in context]
    if GPU:
        return torch.tensor(idxs,dtype=torch.long).to(device)
    else :
        return torch.tensor(idxs,dtype=torch.long)


# 가정: cbow_data DataFrame과 word_to_ix 사전이 이미 준비되어 있음
# unique한 단어 집합 생성 : set() 사용
vocab      =set      (word      for      _,row      in      cbow_data.iterrows()for      word      in
row['context'].split('')+list(row['target']))
print('='*80)
print(f"vocaburary set : {vocab}")
print('='*80)
vocab_size =len(vocab)
print(f"vocaburary size: {vocab_size}")
embedding_dim =50  # 임베딩 차원 설정
# word 기준으로 index 알수 있게 set로 표현
```

```
word_to_ix ={word:i for i,word in enumerate(vocab)}
print(f"word_to_ix: {word_to_ix}")
model =CBOW(vocab_size,embedding_dim)
if GPU :
    model =model.to(device)
# Negative Log Likelihood Loss 손실함수 설정, 아래 코드에서 손실 계산 시 사용
loss_function =nn.NLLLoss()
# 확률적 경사 하강법(Stochastic Gradient Descent, SGD) 옵티마이저 생성, 아래 코드에서 사용
optimizer =optim.SGD(model.parameters(),lr=0.001)
# 학습 시작
SHOW_FLAG =True
WRITE_FLAG =True
EPOCH_NO =2       # 300
print("*"*80)
print(f'\n총 windows(학습용 데이터) 갯수 : {len(windows)}\n')
print('='*80)
for epoch in tqdm(range(EPOCH_NO),desc="Epoch 진행률",unit="iter"):  # 에포크 설정
    total_loss =0
    windowNo =0
    for _,row in tqdm(cbow_data.iterrows(),desc="Sentence 진행률",unit="iter"):     # 전체 데이
터에 대하여

        # 학습용 window 데이터에 대한 context vector 생성
        context_vec =make_context_vector(row['context'].split(''),word_to_ix)

        # 타겟 벡터 생성
        target_vector =[0 for i in range(vocab_size)]
        target_vector[word_to_ix[row['target']]]=1.
        if GPU :
            target_value =torch.tensor(target_vector,dtype=torch.long).to(device)
        else :
            target_value =torch.tensor(target_vector,dtype=torch.long)

        model.zero_grad()                    # 그래디언트 초기화 (gradient 누적 방지)
        # 입력 벡터 생성: 주변 단어들의 임베딩 벡터를 평균내어 하나의 벡터로 합칩니다.
        # 이 평균 벡터가 CBOW 모델의 입력으로 사용됩니다.
          log_probs =model(context_vec)       # context vector를 입력 받아, 실제 출력 및 log
probability 값계산
        if SHOW_FLAG and windowNo %10000 ==0:
            print('\n','='*80)
            print(f'epoch #: {epoch}, window # : {windowNo}')
             print(f'context word, context vector 값 [{len(context_vec)}] 개: {row['context']}::
{context_vec}')
                              print(f'targe_value    tensor    값    [{len(target_value)}]    개:
{row['target']}::{target_value}\n')
            print(f'학습 중 log_probabilities tensor 값 [{len(log_probs)}] 개: {log_probs}')
```

```
            print('='*80)

                #Loss   =   loss_function(log_probs,   torch.tensor([word_to_ix[row['target']]],
dtype=torch.long).to(device))
            loss =loss_function(log_probs,target_value)   # loss 계산 (Cross Entropy 값을 계산하여
손실값으로 사용)
                                                          # -∑y(o) * log(loss),   here, y(o)sms 0또
는 1인 target 값
            loss.backward()                               # gradient 계산
            optimizer.step()                              # weights update

            total_loss +=loss.item()
            #print(f"Sentence [{windowNo}]'s total_loss : {total_loss}")
            windowNo +=1
        print(f'\n\n*** Epoch {epoch}: Total Loss: {total_loss}\n')


# 임베딩 값 추출
# 임베딩 행렬의 차원 크기를 확인
print("*"*80)
print('\n*** 임베딩 Weights의 크기: ',model.embeddings.weight.size(),'\n')
# 임베딩 행렬의 실제 값(데이터) 출력
embeddings_data =model.embeddings.weight.data  # .data는 텐서의 데이터를 가져옵니다.
print('=== 최종 임베딩 Weights === \n')
print(embeddings_data)
print('\n=== 첫번째 단어 임베딩 Weight === \n')
print(embeddings_data[0],'\n')
# 모델의 임베딩 레이어 가중치를 복사하여 계산 그래프로부터 분리된 상태로 embeddings 변수에 저장
# detach() 실행 이후에는 gradient 계산에 참여 안함
embeddings =model.embeddings.weight.detach()
# 관심 단어만 출력해보기 위해, 일단 embedding 단어를 리스트 type의 word에 저장
i=0
word =[];
for key,value in word_to_ix.items():
    word.append(key)
    i +=1
print("*"*80,"\n")
print('\n=== 샘플 몇 단어에 대한 Word Embedding === \n')
with open("./embed_result.txt","w")as fp:
    for ndx,embed in tqdm(enumerate(embeddings)):
        embedded_word =word[ndx]
        if embedded_word in ['happy','pleasant','furious','sad','glad','angry']:
            print(f"\nembeddings[{ndx}], {embedded_word}{embed}")
        # embedding 결과를 파일로 저장
        if WRITE_FLAG :
            fp.write("["+str(embedded_word)+"] : ")
```

```
            if GPU :
                fp.write(str(embed.cpu().numpy())+"\n")
            else :
                fp.write(str(embed.numpy())+"\n")
print("*"*80,"done.\n")


'''
[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()}')
'''
```

[2] 실행 결과 화면 (epoch = 2인 경우)

(word2vec) D:\myprojects\nlp\word2vec>python cbow.py
totally 1225 sentences
*** sentence[0]: You will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings.

*** sentence[1]: I arrived here yesterday, and my first task is to assure my dear sister of my welfare and increasing confidenc

I am already far north of London, and as I walk in the streets of Petersburgh, I feel a cold northern breeze play upon my cheeks, which braces my nerves and fills me with delight.

*** sentence[2]: Do you understand this feeling?

*** sentence[3]: This breeze, which has travelled from the

These reflections have dispelled the agitation with which I began my letter, and I feel my heart glow with an enthusiasm which elevates me to heaven, for nothing contributes so much to tranquillise the mind as a steady purpose-a point on which the soul m

These visions faded when I perused, for the first time, those poets whose effusions entranced my soul and lifted it to heaven.

*** sentence[4]: I also became a poet and for one year lived in a paradise of my own creation; I imagined that I also might obtain a niche in th

Six years have passed since I resolved on my present undertaking.

==========
*** cleaned sentence[0]: you will rejoice to hear that no disaster has accompanied the commencement of an enterprise which you have regarded with such evil forebodings .

*** cleaned sentence[1]: i arrived here yesterday , and my first task is to assure my dear sister of my welfare and increasing confidenc i am already far north of london , and as i walk in the streets of petersburgh , i feel a cold northern breeze play upon my cheeks , which braces my nerves and fills me with delight .

*** cleaned sentence[2]: do you understand this feeling ?

*** cleaned sentence[3]: this breeze , which has travelled from the these reflections have dispelled the agitation with which i began my letter , and i feel my heart glow with an enthusiasm which elevates me to heaven , for nothing contributes so much to tranquillise the mind as a steady purpose a point on which the soul m these visions faded when i perused , for the first time , those poets whose effusions entranced my soul and lifted it to heaven .

*** cleaned sentence[4]: i also became a poet and for one year lived in a paradise of my own creation i imagined that i also might obtain a niche in th six years have passed since i resolved on my present undertaking .

100%|██████████| 1225/1225 [00:00<00:00, 111659.73it/s]

==========

First 10 windows data for trainig

==========

('<MASK>', '<MASK>', 'you', 'will', 'rejoice')
('<MASK>', 'you', 'will', 'rejoice', 'to')
('you', 'will', 'rejoice', 'to', 'hear')
('will', 'rejoice', 'to', 'hear', 'that')
('rejoice', 'to', 'hear', 'that', 'no')
('to', 'hear', 'that', 'no', 'disaster')
('hear', 'that', 'no', 'disaster', 'has')
('that', 'no', 'disaster', 'has', 'accompanied')
('no', 'disaster', 'has', 'accompanied', 'the')
('disaster', 'has', 'accompanied', 'the', 'commencement')

100%|██████████| 35787/35787 [00:00<00:00, 394315.08it/s]

==========

First 20 CBOW data for trainig

==========

	context	target
0	will rejoice	you
1	you rejoice to	will
2	you will to hear	rejoice
3	will rejoice hear that	to
4	rejoice to that no	hear
5	to hear no disaster	that
6	hear that disaster has	no
7	that no has accompanied	disaster
8	no disaster accompanied the	has
9	disaster has the commencement	accompanied
10	has accompanied commencement of	the
11	accompanied the of an	commencement
12	the commencement an enterprise	of
13	commencement of enterprise which	an
14	of an which you	enterprise
15	an enterprise you have	which
16	enterprise which have regarded	you
17	which you regarded with	have
18	you have with such	regarded
19	have regarded such evil	with

***** Device : cuda

==========

vocaburary set : {'', 'train', 'procure', 'wishing', 'arrowy', 'disposition', 'barred', 'chasms', 'element', 'preceding', 'fills', 'still', 'comforter', 'bent', 'englishman', 'rays', 'repentance', 'having', 'groans', 'highlands', 'revenge', 'link', 'convinced', 'reflection', 'posse', 'there', 'attained', 'remembered', 'pleasure', 'prospect', 'dearest', 'bed', 'forbid', 'trial', 'convenience', 'philosopher', 'very', 'moonlight', 'darkened', 'mannheim', 'celebrated', 'momentarily', 'tremble', 'speak', 'objec', 'placed', 'free', 'interval', 'nought', 'affections', 'accompany', 'book', 'grateful', 'impassable', 'employ', 'rejoined', 'torrent', 'motives',

..... (중간 생략)

'ah', 'defend', 'stretched', 'fearful', 'bearing', 'innumerable', 'retreated', 'peace', 'gloried', 'myriads', 'always', 'choked', 'greatest', 'did', 'offer', 'help', 'excessively', 'penetrat', 'forget', 'ba', 'enjoying', 'source', 'countries', 'which', 'thoughts', 'coach', 'protection', 'government', 'february', 'preyed', 'seize', 'struck', 'air', 'unworthy', 'language', 'introduce', 'incessantly', 't', 'plainly', 'riches', 'recess', 'determin', 'painful', 'adduced', 'arising', 'geneva', 'hair', 'vast', 'conclusions', 'secret', 'st', 'consumed', 'averred'}

==========

vocaburary size: 4543

word_to_ix: {'': 0, 'train': 1, 'procure': 2, 'wishing': 3, 'arrowy': 4, 'disposition': 5, 'barred': 6, 'chasms': 7, 'element': 8, 'preceding': 9, 'fills': 10, 'still': 11, 'comforter': 12, 'bent': 13, 'englishman': 14, 'rays': 15, 'repentance': 16, 'having': 17, 'groans': 18, 'highlands': 19, 'revenge': 20, 'link': 21, 'convinced': 22, 'reflection': 23, 'posse': 24, 'there': 25, 'attained': 26, 'remembered': 27, 'pleasure': 28, 'prospect': 29, 'dearest': 30, 'bed': 31, 'forbid': 32, 'trial': 33, 'convenience': 34, 'philosopher': 35, 'very': 36, 'moonlight': 37, 'darkened': 38, 'mannheim': 39, 'celebrated': 40, 'momentarily': 41,

..... (중간 생략)

'drunken': 4478, 'dissoluble': 4479, 'lieutenant': 4480, 'loved': 4481, 'immortal': 4482, 'sir': 4483, 'curse': 4484, 'unfair': 4485, 'benevolently': 4486, 'despise': 4487, 'sufficiently': 4488, 'yes': 4489, 'ah': 4490, 'defend': 4491, 'stretched': 4492, 'fearful': 4493, 'bearing': 4494, 'innumerable': 4495, 'retreated': 4496, 'peace': 4497, 'gloried': 4498, 'myriads': 4499, 'always': 4500, 'choked': 4501, 'greatest': 4502, 'did': 4503, 'offer': 4504, 'help': 4505, 'excessively': 4506, 'penetrat': 4507, 'forget': 4508, 'ba': 4509, 'enjoying': 4510, 'source': 4511, 'countries': 4512, 'which': 4513, 'thoughts': 4514, 'coach': 4515, 'protection': 4516, 'government': 4517, 'february': 4518, 'preyed': 4519, 'seize': 4520, 'struck': 4521, 'air': 4522, 'unworthy': 4523, 'language': 4524, 'introduce': 4525, 'incessantly': 4526, 't': 4527, 'plainly': 4528, 'riches': 4529, 'recess': 4530, 'determin': 4531, 'painful': 4532, 'adduced': 4533, 'arising': 4534, 'geneva': 4535, 'hair': 4536, 'vast': 4537, 'conclusions': 4538, 'secret': 4539, 'st': 4540, 'consumed': 4541, 'averred': 4542}

****

총 windows(학습용 데이터) 갯수 : 35787

==========

Epoch 진행률: 0%| | 0/2 [00:00<?,

```
?iter/s]
 ================================================================================
epoch #: 0, window # : 0
context word, context vector 값 [2] 개: will rejoice :: tensor([4402,  929], device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([-8.9173, -7.8941, -7.7683,  ..., -8.6502, -8.9869,
-8.7325],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

 ================================================================================
epoch #: 0, window # : 10000
context word, context vector 값 [4] 개: should suffer guilty . :: tensor([1651, 2482, 1866, 3113],
device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([ -0.0500,  -8.7717, -11.3032,  ..., -10.8331,
-11.6671, -11.4743],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

 ================================================================================
epoch #: 0, window # : 20000
context word, context vector 값 [4] 개: you crossed mind . :: tensor([1497,  190, 2766, 3113],
device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([-0.7790, -4.5258, -9.0831,  ..., -8.9938, -9.1995,
-9.3674],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

 ================================================================================
epoch #: 0, window # : 30000
context word, context vector 값 [4] 개: tranquil demeanour greatly to :: tensor([1528, 2546, 4152,
1776], device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([ -0.0523, -12.8129, -10.9633,  ..., -11.1303,
-11.6893, -11.2130],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

Sentence 진행률: 35787iter [00:50, 703.55iter/s]
Sentence 진행률: 35723iter [00:50, 697.90iter/s]

*** Epoch 0: Total Loss: 39358.836245960105

Epoch           진행률:                50%|
```

```
| 1/2 [00:50<00:50, 50.87s/iter]
 ================================================================================
epoch #: 1, window # : 0
context word, context vector 값 [2] 개: will rejoice :: tensor([4402,  929], device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([ -0.0813,  -6.8351, -10.3280,  ..., -11.2089,
-11.5519, -11.2906],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

Sentence 진행률: 497iter [00:00, 714.77iter/s]
 ================================================================================
epoch #: 1, window # : 10000
context word, context vector 값 [4] 개: should suffer guilty . :: tensor([1651, 2482, 1866, 3113],
device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([-1.2329e-02, -6.0656e+00, -1.2892e+01,  ...,
-1.2409e+01,
        -1.3257e+01, -1.3077e+01], device='cuda:0',
       grad_fn=<LogSoftmaxBackward0>)
================================================================================

 ================================================================================
epoch #: 1, window # : 20000
context word, context vector 값 [4] 개: you crossed mind . :: tensor([1497,  190, 2766, 3113],
device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([ -0.2854,  -3.0567, -10.0467,  ...,  -9.9475,
-10.1725, -10.3371],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================

 ================================================================================
epoch #: 1, window # : 30000
context word, context vector 값 [4] 개: tranquil demeanour greatly to :: tensor([1528, 2546, 4152,
1776], device='cuda:0')
targe_value tensor 값 [4543] 개: tensor([0, 0, 0,  ..., 0, 0, 0], device='cuda:0')
학습 중 log_probabilities tensor 값 [4543] 개: tensor([ -0.0189, -13.6878, -11.9640,  ..., -12.1304,
-12.6914, -12.2127],
       device='cuda:0', grad_fn=<LogSoftmaxBackward0>)
================================================================================
Sentence 진행률: 35787iter [00:49, 717.51iter/s]
Sentence 진행률: 35775iter [00:49, 716.28iter/s]

*** Epoch 1: Total Loss: 4602.6263895849
```

```
Epoch 진행률: 100%|██████████████████████████████████████████████████████████████████████████| 2/2 [01:40<00:00, 50.38s/iter]
********************************************************************************

*** 임베딩 Weights의 크기: torch.Size([4543, 50])

=== 최종 임베딩 Weights ===

tensor([[ 2.1241,  1.5627,  0.4285,  ...,  0.8522,  0.2731, -1.1010],
        [-0.1305,  0.5768,  0.3895,  ...,  0.0580,  2.7898, -0.3826],
        [-0.6365,  0.7781, -0.0162,  ...,  0.1731, -0.8228, -0.9089],
        ...,
        [-1.8494,  0.0384,  0.5189,  ...,  0.0460, -1.0703,  0.6078],
        [ 0.7810,  0.0915,  0.0193,  ...,  0.7169, -1.0041, -0.3739],
        [-1.8179, -0.1967,  1.8613,  ...,  0.5120, -0.7805, -1.0677]],
       device='cuda:0')

=== 첫번째 단어 임베딩 Weight ===

tensor([ 2.1241,  1.5627,  0.4285,  0.4039,  0.2900,  1.7413,  0.0030,  0.2270,
        -0.2653,  0.9494,  0.5762, -0.1812, -0.7862,  0.3575, -0.3456, -0.6065,
         0.6546,  0.5725,  1.8827, -0.8461,  0.3905,  0.0602,  1.3168, -0.7777,
         0.7188, -1.9644,  0.0094,  0.1487,  0.6327,  2.1018, -0.1820,  0.1158,
         0.1603, -0.1425,  0.0556,  0.8788, -0.1062, -0.5256,  0.5757, -1.0852,
         0.7902, -1.7574, -0.3606, -0.2904,  0.9846,  0.2380,  0.8346,  0.8522,
         0.2731, -1.1010], device='cuda:0')

********************************************************************************* done.
642it [00:00, 3184.95it/s]

=== 샘플 몇 단어에 대한 Word Embedding ===

embeddings[890], glad : tensor([-0.9927,  0.6405, -0.5539,  0.3147, -0.0874, -0.1151, -0.3769, -0.7577,
        -1.3260,  0.2886,  0.8342,  1.3835,  1.6598, -1.2476, -1.3026,  0.6131,
         0.0468, -0.1661,  1.1373,  1.1395, -0.8783, -0.2720,  1.2946,  0.0784,
         0.8793, -0.3270,  0.6442,  0.3476,  0.1714, -0.1498,  0.7663, -0.4391,
         0.5585, -0.1294,  0.5613,  0.0758,  1.1420, -0.4019, -1.0427,  0.5848,
         0.8452, -0.0746, -1.2735,  0.5827, -0.4541,  0.1586,  1.0603, -1.0425,
        -0.2397,  0.5897], device='cuda:0')
1516it [00:01, 1445.54it/s]
embeddings[1657], sad : tensor([ 1.3032, -1.4314,  0.3534, -0.0941,  1.9225,  0.7828, -1.9444,  2.1177,
        -0.3611, -2.6296,  1.0626, -3.5438,  0.7824, -0.9444,  0.2083,  0.4697,
         0.4441, -0.5370, -0.2495, -0.3210, -0.1175, -0.3748,  0.1912,  0.1923,
        -0.8905, -0.1323, -1.4642,  0.5950,  0.3995, -0.0484, -0.3476, -0.5873,
         0.4650, -0.2543, -1.0821,  1.1023, -0.4606, -0.4853, -0.4318,  0.6348,
         0.0350, -0.0765,  0.0737,  1.6285, -1.4593,  0.1005,  0.1976,  0.5687,
```

```
        -0.9203, -2.1614], device='cuda:0')
2074it [00:01, 1961.36it/s]
embeddings[2269], angry : tensor([-1.8079,  1.2002,  0.2330, -1.7551,  1.7498, -0.4398,  0.7738,
-0.2494,
        -1.3302,  1.6438,  0.7144,  1.7497,  0.2306,  0.2656,  0.3487, -1.1309,
         0.2688, -0.8254,  0.1273, -2.6429,  0.3835, -1.0630, -0.4772,  0.4171,
        -0.7085,  0.4166, -0.1855, -1.2487, -0.3522,  0.7780, -0.1499, -1.0864,
        -0.4568, -1.0485, -0.8726,  1.2220,  0.7663, -0.9898, -0.4258,  0.8982,
         2.4930,  0.1639,  0.6914,  1.4079, -0.6537, -0.8187,  0.2563,  0.3245,
        -0.3035,  0.9934], device='cuda:0')
2991it [00:01, 2615.90it/s]
embeddings[3056], happy : tensor([ 0.8433,  1.5583,  0.1610,  1.4857,  1.8537,  0.8661,  1.8705,  1.6900,
        -0.5615, -0.5597, -1.4675,  0.5745, -0.4232, -0.1536,  1.6293, -0.4285,
         0.1723, -0.3415,  0.1364, -1.0259,  0.5326,  0.5280, -1.0947, -0.1708,
        -0.8487, -0.3820, -1.6517, -0.3660,  2.0433,  0.9543,  0.9158,  3.1012,
        -0.2179,  1.7501, -0.0959,  2.1476, -1.7930, -1.0893,  0.4925,  0.7157,
        -0.0118,  1.2912,  0.0041,  0.6235, -1.5642, -0.9671,  1.3364,  1.0999,
         2.0922,  1.3200], device='cuda:0')

embeddings[3254], pleasant : tensor([-0.2186, -0.3838, -0.6181,  1.7232, -0.8669, -0.6622, -0.3983,
0.0705,
         2.2437,  0.4012, -0.0983, -1.9736,  0.4551, -0.0312, -0.9487,  0.6270,
         0.0967,  0.1126,  1.0864, -0.5974, -2.4115,  0.6459,  0.8265, -0.6092,
         0.1861, -0.5827, -0.7598, -1.0677, -1.9891,  1.5273,  0.6923,  0.0295,
        -1.0126,  0.0939, -0.0273, -0.9161, -0.8324,  0.5876,  0.9422, -1.0538,
         0.7686, -1.0638, -0.6282, -0.7096, -0.6117,  0.4796,  0.4225,  1.6483,
         0.6275,  0.2429], device='cuda:0')
4218it [00:01, 2902.35it/s]
embeddings[4319], furious : tensor([ 0.7425, -1.6272,  0.1211, -0.8501,  1.3598,  0.9128, -0.8068,
-0.8556,
         1.3498,  0.0227,  0.4840,  0.3526, -1.1560,  1.1360,  0.6370,  0.3893,
        -0.5235, -0.4732, -1.0230,  2.7591, -0.6900,  0.0729, -0.3265,  0.0129,
        -0.0351,  0.2111, -2.2044, -1.6991, -1.5507,  1.0611, -0.0291, -0.2887,
        -2.7748,  1.0880, -0.1550, -0.8555,  0.1802, -0.6297,  1.6572,  0.4747,
        -0.5192, -1.4266, -0.2030,  0.1118,  1.5662, -1.3859,  0.1833, -0.1840,
         1.9174, -0.6119], device='cuda:0')
4543it [00:02, 2158.81it/s]
********************************************************************************** done.

(word2vec) D:\myprojects\nlp\word2vec>
```

[3] 최종 embedding 결과 파일 : embed_result.txt (epoch = 2인 경우)

```
[] : [ 2.1241095    1.562675     0.4285141    0.4038742    0.289989     1.741275
  0.00297915  0.22696362 -0.2653341    0.9494176    0.5761501   -0.1811887
 -0.7861988   0.35748595 -0.34564835 -0.60653627  0.65463525   0.57247406
  1.8827409  -0.8460746   0.39045912   0.06019455  1.3167651   -0.77768165
  0.71881473 -1.9644005   0.00935276   0.14865041  0.6327095    2.1017683
 -0.18196422  0.11575571  0.16032776 -0.14252502  0.055554     0.87879705
 -0.10620566 -0.52563506  0.5757469   -1.085218    0.7902379   -1.7573795
 -0.36061472 -0.29042172  0.98460513   0.23796982  0.83459675   0.8522259
  0.27305022 -1.1009706 ]
[train]  :  [-0.13047624     0.57678646     0.38946137   -1.034321          1.6491114
0.86571395
 -1.0076109   0.00800916   0.00797427   1.6790652   -1.197215     0.38665605
  0.40243605 -0.40457353 -0.1814661    0.5477029    1.1229335   -0.26904392
 -0.5224788   -1.50991     -0.11876122 -0.5803628   -0.6393104   -0.04842669
 -1.4747962   0.9106979   -0.5530583   -0.94254     -1.0917923    1.0314575
  0.8534797   0.81200343 -1.8405504   -0.73378587   0.384942    -0.6864165
 -1.2263125   -1.4205174   -0.14344975   0.99921554   1.0848597    0.15187785
  0.08960258 -0.30385053   0.57699597   0.210002     1.2246515    0.05799538
  2.7898362   -0.38264647]
[procure]  :  [-0.63648564     0.77808535   -0.01617103   -1.0237445     -0.40015563
-0.2846248

                                ..... (중간 생략)

[averred] : [-1.8178580e+00 -1.9668880e-01  1.8613236e+00  6.0798645e-01
  1.7559439e-01 -4.4356599e-01 -3.5936022e-01  1.5302268e+00
  2.5343575e+00  1.5745474e+00  4.2340818e-01 -1.3687502e+00
 -1.4267161e+00  4.6868908e-01 -4.1338634e-01  8.0820698e-01
 -6.3234687e-01 -1.5189347e-01 -2.6961598e-01 -6.8562323e-01
  8.5603607e-01  4.3562669e-01 -8.4975165e-01 -5.2101344e-01
  2.3898008e+00  1.3615173e+00 -9.7617698e-01  1.6556450e+00
 -1.0103662e+00  1.0018423e+00  1.0175965e+00 -3.7570795e-01
 -5.2945375e-01 -1.2392714e+00 -3.3295393e-01 -9.2760060e-04
  9.7112191e-01  2.6437959e-01  3.9212075e-01 -1.1002165e+00
  9.8008662e-01 -2.8344858e-01 -1.5412999e+00  5.6289798e-01
 -1.3813885e+00  3.1917280e-01  1.4663033e+00  5.1202971e-01
 -7.8049511e-01 -1.0677309e+00]
```

※ [Questions]

1. 감정과 관련된 몇 개 단어(happy, glad, ...) 간의 Euclidean Similarity와 Cosine Similarity를 python 프로그램으로 구하시오.
2. Similarity가 의미가 있게 되는 적절한 Epoch 수(300 ?)를 실험적으로 구해 보시오
3. Embedding Code의 길이를 변경(100)하여 실행해 보고 성능을 이전과 비교해 보시오
4. Window Size를 변경(2→5)하여 실행해 보고, 이전과 성능을 비교해 보시오