숭실대학교 Soongsil University

실습 강의

word2vec (CBOW)

2023 Fall

Natural Language Processing

kunwoo.park@ssu.ac.kr

실습 목표

- 사전 훈련된 임베딩(pretrained embedding)을 사용해 보고, 그 원리를 이해해 보자.
- word2vec CBOW 방법을 이용해 직접 임베딩을 훈련해 보자.

라이브러리 설치 및 import

- Annoy

Annoy (Approximate Nearest Neighbors Oh Yeah) is a C++ library with Python bindings to search for points in space that are close to a given query point. It also creates large read-only file-based data structures that are mmapped into memory so that many processes may share the same data.

```
# annoy 패키지를 설치합니다.
!pip install annoy
```

```
import torch
import torch.nn as nn
from tqdm import tqdm
from annoy import AnnoyIndex
import numpy as np
```

사전 학습 임베딩 사용을 위한 PreTrainedEmbeddings 클래스 선언

- 생성자
  - 단어-인덱스 매핑과, 단어 벡터를 저장한다.
  - 이 때, 인덱스에 대응되는 벡터는 해당 인덱스에 대응되는 단어의 벡터를 나타낸다.

```
class PreTrainedEmbeddings(object):
    """ 사전 훈련된 단어 벡터 사용을 위한 래퍼 클래스 """
    def __init__(self, word_to_index, word_vectors):
        """
        매개변수:
            word_to_index (dict): 단어에서 정수로 매핑
            word_vectors (numpy 배열의 리스트)
        """
        self.word_to_index = word_to_index
        self.word_vectors = word_vectors
        self.index_to_word = {v: k for k, v in self.word_to_index.items()}

        self.index = AnnoyIndex(len(word_vectors[0]), metric='euclidean')
        print("인덱스 만드는 중!")
        for _, i in self.word_to_index.items():
            self.index.add_item(i, self.word_vectors[i])
        self.index.build(50)
        print("완료!")

```

사전 학습 임베딩 사용을 위한 PreTrainedEmbeddings 클래스 선언

- `from_embeddings_file()`: 파일로부터 벡터 값을 읽어 메모리에 보관

```
20      @classmethod
21      def from_embeddings_file(cls, embedding_file):
22          """사전 훈련된 벡터 파일에서 객체를 만듭니다.
23
24          벡터 파일은 다음과 같은 포맷입니다:
25              word0 x0_0 x0_1 x0_2 x0_3 ... x0_N
26              word1 x1_0 x1_1 x1_2 x1_3 ... x1_N
27
28          매개변수:
29              embedding_file (str): 파일 위치
30          반환값:
31              PretrainedEmbeddings의 인스턴스
32          """
33          word_to_index = {}
34          word_vectors = []
35
36          with open(embedding_file) as fp:
37              for line in fp.readlines():
38                  line = line.split(" ")
39                  word = line[0]
40                  vec = np.array([float(x) for x in line[1:]])
41
42                  word_to_index[word] = len(word_to_index)
43                  word_vectors.append(vec)
44
45          return cls(word_to_index, word_vectors)
```

사전 학습 임베딩 사용을 위한 PreTrainedEmbeddings 클래스 선언

```
47      def get_embedding(self, word):
48          """
49          매개변수:
50              word (str)
51          반환값
52              임베딩 (numpy.ndarray)
53          """
54          return self.word_vectors[self.word_to_index[word]]
55
56      def get_closest_to_vector(self, vector, n=1):
57          """벡터가 주어지면 n 개의 최근접 이웃을 반환합니다
58          매개변수:
59              vector (np.ndarray): Annoy 인덱스에 있는 벡터의 크기와 같아야 합니다
60              n (int): 반환될 이웃의 개수
61          반환값:
62              [str, str, ...]: 주어진 벡터와 가장 가까운 단어
63                  단어는 거리순으로 정렬되어 있지 않습니다.
64          """
65          nn_indices = self.index.get_nns_by_vector(vector, n)
66          return [self.index_to_word[neighbor] for neighbor in nn_indices]
```

사전 학습 임베딩 사용을 위한 PreTrainedEmbeddings 클래스 선언

- `compute_and_print_analogy()`: 단어 임베딩 기반 유추 관계 분석

man : he :: woman : ? → she

man : king :: woman : ? → queen

a : a* :: b : ?

$$\hat{b}^* = \underset{x}{\operatorname{argmin}} \ \text{distance}(x, a^* - a + b)$$

```
68      def compute_and_print_analogy(self, word1, word2, word3):
69          """단어 임베딩을 사용한 유추 결과를 출력합니다
70
71          word1이 word2일 때 word3은 __입니다.
72          이 메서드는 word1 : word2 :: word3 : word4를 출력합니다
73
74          매개변수:
75              word1 (str)
76              word2 (str)
77              word3 (str)
78          """
79          vec1 = self.get_embedding(word1)
80          vec2 = self.get_embedding(word2)
81          vec3 = self.get_embedding(word3)
82
83          # 네 번째 단어 임베딩을 계산합니다
84          spatial_relationship = vec2 - vec1
85          vec4 = vec3 + spatial_relationship
86
87          closest_words = self.get_closest_to_vector(vec4, n=4)
88          existing_words = set([word1, word2, word3])
89          closest_words = [word for word in closest_words
90                                   if word not in existing_words]
91
92          if len(closest_words) == 0:
93              print("계산된 벡터와 가장 가까운 이웃을 찾을 수 없습니다!")
94              return
95
96          for word4 in closest_words:
97              print("{} : {} :: {} : {}".format(word1, word2, word3, word4))
```

사전 학습 임베딩 사용 실험

- Glove 임베딩 다운로드 (6B token, 100d)

```
# GloVe 데이터를 다운로드합니다.
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
!mkdir -p data/glove
!mv glove.6B.100d.txt data/glove
```

- 앞서 정의한 PreTrainedEmbeddings 클래스를 이용해 로드!

```
embeddings = PreTrainedEmbeddings.from_embeddings_file('data/glove/glove.6B.100d.txt')
```

사전 학습 임베딩 사용 실험

- 유추 관계 실험

```
1 embeddings.compute_and_print_analogy('man', 'he', 'woman')
```

```
man : he :: woman : she
man : he :: woman : never
```

```
1 embeddings.compute_and_print_analogy('fly', 'plane', 'sail')
```

```
fly : plane :: sail : ship
fly : plane :: sail : vessel
```

```
1 embeddings.compute_and_print_analogy('cat', 'kitten', 'dog')
```

```
cat : kitten :: dog : puppy
cat : kitten :: dog : puppies
cat : kitten :: dog : hound
```

```
1 embeddings.compute_and_print_analogy('blue', 'color', 'dog')
```

```
blue : color :: dog : bites
blue : color :: dog : description
blue : color :: dog : treats
blue : color :: dog : depending
```

실습

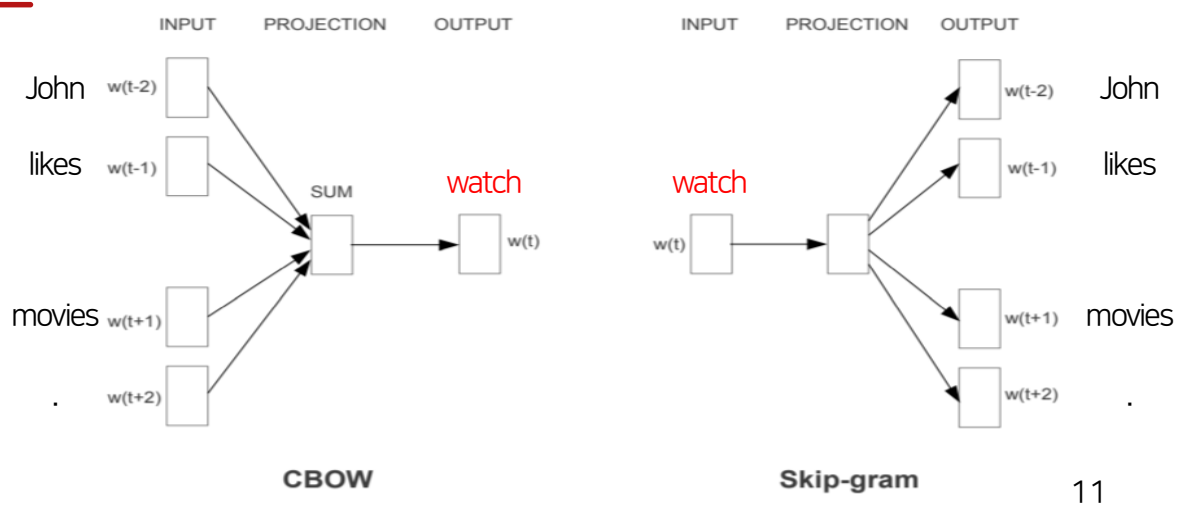
- 파일: 14강_실습_Pretrained_Embeddings.ipynb
- 아래 메소드를 이용해 특정 단어와 가장 유사한 의미를 가지는 단어를 찾아보자.
  - `get_embedding()`
  - `get_closest_to_vector()`

- king 과 유사한 단어는?
- student 와 유사한 단어는?
- student : study :: professor : x ?

word2vec

- 단어 임베딩을 주변 맥락과 단어의 관계에 기반해 학습하는 방법
  - 가설: "비슷한 문맥에서 등장하는 단어들은 비슷한 의미를 가진다"
- CBOW(Continuous Bag of Words)와 Skip-Gram 두 가지로 나뉜다.
  - CBOW: 맥락으로부터 타겟 단어 예측
  - Skip-gram: 타겟 단어로부터 맥락 예측
- 이 실습에서는 CBOW를 살펴봄

"John likes watch movies ."



실습 세팅

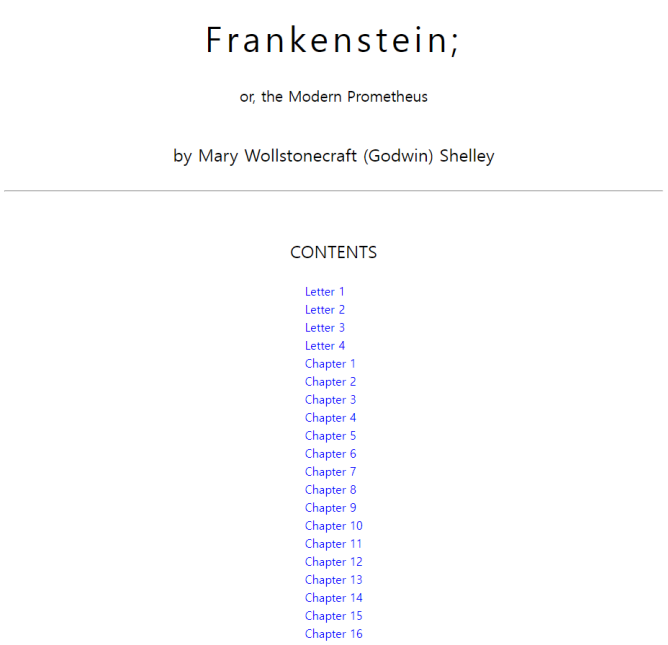
- 라이브러리 import 및 설정값 지정

```python
import os
from argparse import Namespace
import collections
import nltk.data
import numpy as np
import pandas as pd
import re
import string
import tqdm
import torch.nn as nn
import torch
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
```

```python
args = Namespace(
    raw_dataset_txt="data/frankenstein.txt",
    window_size=5,
    train_proportion=0.7,
    val_proportion=0.15,
    test_proportion=0.15,
    cbow_csv="data/frankenstein_with_splits.csv",
    # 날짜와 경로 정보
    vectorizer_file="vectorizer.json",
    model_state_file="model.pth",
    save_dir="model_storage/cbow",
    # 모델 하이퍼파라미터
    embedding_size=50,
    # 훈련 하이퍼파라미터
    seed=1337,
    num_epochs=100,
    learning_rate=0.0001,
    batch_size=32,
    early_stopping_criteria=5,
    # 실행 옵션
    cuda=True,
    expand_filepaths_to_save_dir=True
)
```

타겟 데이터

- 프로젝트 구텐베르크에서 제공하는 메리 셸리의 소설 “프랑켄슈타인”
  - 약 7만개의 단어로 구성



Frankenstein;

or, the Modern Prometheus

by Mary Wollstonecraft (Godwin) Shelley

CONTENTS

'Frankenstein,₩n₩nor the Modern Prometheus₩n₩n₩nby₩n₩nMary ₩ollstonecraft (Godwin) Shelley₩n₩n₩nLetter 1₩n₩n₩nSt. Petersburgh, Dec. 11th, 17--₩n₩nTO Mrs. Saville, England₩n₩nYou will rejoice to hear that no disaster has accompanied the₩ncommencement of an enterprise which you have regarded with such evil₩nforebodings. I arrived here yesterday, and my first task is to assure₩nmy dear sister of my welfare and increasing confidence in the success₩nof my undertaking.₩n₩nI am already far north of London, and as I walk in the streets of₩nPetersburgh, I feel a cold northern breeze play upon my cheeks, which₩nbraces my nerves and fills me with delight. Do you understand this₩nfeeling? This breeze, which has travelled from the regions towards₩n which I am advancing, gives me a foretaste of those icy climes.₩nInspirited by this wind of promise, my daydreams become more fervent₩nand vivid. I try in vain to be persuaded that the pole is the seat of₩nfrost and desolation; it ever presents itself to my imagination as the₩nregion of beauty and delight. There, Margaret, the sun is forever₩nvisible, its broad disk just skirting the horizon and diffusing a₩nperpetual splendour. There--for with your leave, my sister, I will put₩nsome trust in preceding navigators--there snow and frost are banished;₩nand, sailing over a calm sea, we may be wafted to a land surpassing in₩nwonders and in beauty every region hitherto discovered on the habitable₩nglobe. Its productions and features may be without example, as the₩nphenomena of the heavenly bodies undoubtedly are in those undiscovered₩nsolitudes. What may not be expected in a country of eternal light? I₩nmay there discover the wondrous power which attracts the needle and may₩nregulate a thousand celestial observations that require only this₩nvoyage to render their seeming eccentricities consistent forever. I₩nshall satiate my ardent curiosity with the sight of a part of the world₩nnever before visited, and may tread a land never before imprinted by₩nthe foot of man. These are my enticements, and they are sufficient to₩nconquer all fear of danger or death and to induce me to commence this₩nlaborious voyage with the joy a child feels when he embarks in a little₩nboat, with his holiday mates, on an expedition of discovery up his₩nnative river. But supposing all these conjectures to be false, you₩ncannot contest the inestimable benefit which I shall confer on all₩nmankind, to the last generation, by discovering a passage near the pole₩nto those countries, to reach which at present so many months are₩nrequisite; or by ascertaining the secret of the magnet, which, if at₩nall possible, can only be effected by an undertaking such as mine.₩n₩nThese reflections have dispe

데이터 로드 및 전처리

- NLTK의 Punkt 토큰 분할기를 사용해 텍스트를 문장으로 분할
- 각 문장을 소문자로 변환 후 구두점 제거
- 공백으로 문자열 분할해 토큰 리스트 추출

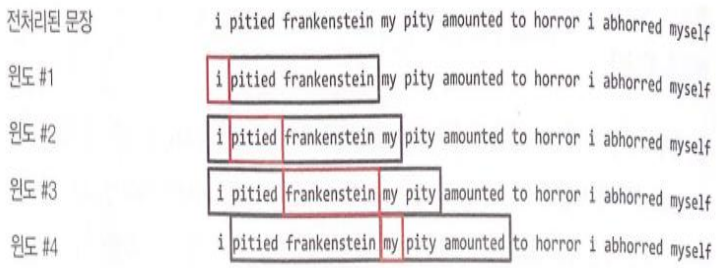
```
# 원본 데이터를 읽고, 문장으로 split
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
with open(args.raw_dataset_txt) as fp:
    book = fp.read()
sentences = tokenizer.tokenize(book)
```

```
# 데이터 정제
def preprocess_text(text):
    text = ' '.join(word.lower() for word in text.split(" "))
    text = re.sub(r"([.,!?])", r" ₩1 ", text)
    text = re.sub(r"[^a-zA-Z.,!?]+", r" ", text)
    return text

cleaned_sentences = [preprocess_text(sentence) for sentence in sentences]
```

데이터 로드 및 전처리

- CBOW 학습을 위해, 지정된 크기의 window로 맥락과 타겟 단어 쌍을 구성




```
>>> from nltk.util import ngrams
>>> list(ngrams([1,2,3,4,5], 3))
[(1, 2, 3), (2, 3, 4), (3, 4, 5)]
```

ngrams([‘[MASK]’, ‘[MASK]’, ‘i’, ‘pitted’,
‘Frankenstein’, ‘my’, ‘pity’, ‘amounted’,
‘to’, ‘horro’, ‘i’, ‘abhorred’, ‘myself’,
‘[MASK]’, ‘[MASK]’], 5)

```python
# Global vars
MASK_TOKEN = "<MASK>"

# Create windows
flatten = lambda outer_list: [item for inner_list in outer_list for item in inner_list]
windows = flatten([list(nltk.ngrams([MASK_TOKEN] * args.window_size + sentence.split(' ') + #
    [MASK_TOKEN] * args.window_size, args.window_size * 2 + 1)) #
    for sentence in tqdm.notebook.tqdm(cleaned_sentences)])

# Create cbow data
data = []
for window in tqdm.notebook.tqdm(windows):
    target_token = window[args.window_size]
    context = []
    for i, token in enumerate(window):
        if token == MASK_TOKEN or i == args.window_size:
            continue
        else:
            context.append(token)
    data.append([' '.join(token for token in context), target_token])


# Convert to dataframe
cbow_data = pd.DataFrame(data, columns=["context", "target"])
```

데이터 로드 및 전처리

- 훈련, 검증, 테스트 세트로 분할

```
# Create split data
n = len(cbow_data)
def get_split(row_num):
    if row_num <= n*args.train_proportion:
        return 'train'
    elif (row_num > n*args.train_proportion) and (row_num <= n*args.train_proportion + n*args.val_proportion):
        return 'val'
    else:
        return 'test'
cbow_data['split']= cbow_data.apply(lambda row: get_split(row.name), axis=1)

# Write split data to file
cbow_data.to_csv(args.cbow_csv, index=False)
```

CBOW Dataset 클래스 선언

```python
class CBOWDataset(Dataset):
    def __init__(self, cbow_df, vectorizer):
        """
        매개변수:
            cbow_df (pandas.DataFrame): 데이터셋
            vectorizer (CBOWVectorizer): 데이터셋에서 만든 CBOWVectorizer 객체
        """
        self.cbow_df = cbow_df
        self._vectorizer = vectorizer

        measure_len = lambda context: len(context.split(" "))
        self._max_seq_length = max(map(measure_len, cbow_df.context))

        self.train_df = self.cbow_df[self.cbow_df.split=='train']
        self.train_size = len(self.train_df)

        self.val_df = self.cbow_df[self.cbow_df.split=='val']
        self.validation_size = len(self.val_df)

        self.test_df = self.cbow_df[self.cbow_df.split=='test']
        self.test_size = len(self.test_df)

        self._lookup_dict = {'train': (self.train_df, self.train_size),
                             'val': (self.val_df, self.validation_size),
                             'test': (self.test_df, self.test_size)}

        self.set_split('train')
```

CBOW Dataset 클래스 선언

- `__getitem__()`
  - Vectorizer를 사용해 문맥(윈도우 내 타겟 단어를 제외한 단어들)을 벡터로 변환
  - Vocabulary를 사용해 타겟(윈도 가운데 단어)을 정수 인덱스로 변환

```
90      def __getitem__(self, index):
91          """파이토치 데이터셋의 주요 진입 메서드
92
93          매개변수:
94              index (int): 데이터 포인트의 인덱스
95          반환값:
96              데이터 포인트의 특성(x_data)과 레이블(y_target)로 이루어진 딕셔너리
97          """
98          row = self._target_df.iloc[index]
99
100         context_vector = ㅐ
101             self._vectorizer.vectorize(row.context, self._max_seq_length)
102         target_index = self._vectorizer.cbow_vocab.lookup_token(row.target)
103
104         return {'x_data': context_vector,
105                 'y_target': target_index}
```

Vocabulary 클래스 선언

- 감성 분석에서 사용한 클래스와 거의 동일하나, MASK 토큰이 추가됨

```python
class Vocabulary(object):
    """ 매핑을 위해 텍스트를 처리하고 어휘 사전을 만드는 클래스 """

    def __init__(self, token_to_idx=None, mask_token="<MASK>", add_unk=True,
                 unk_token="<UNK>"):
        """
        매개변수:
            token_to_idx (dict): 기존 토큰-인덱스 매핑 딕셔너리
            mask_token (str): Vocabulary에 추가할 MASK 토큰.
                모델 파라미터를 업데이트하는데 사용하지 않는 위치를 나타냅니다.
            add_unk (bool): UNK 토큰을 추가할지 지정하는 플래그
            unk_token (str): Vocabulary에 추가할 UNK 토큰
        """

        if token_to_idx is None:
            token_to_idx = {}
        self._token_to_idx = token_to_idx

        self._idx_to_token = {idx: token
                              for token, idx in self._token_to_idx.items()}

        self._add_unk = add_unk
        self._unk_token = unk_token
        self._mask_token = mask_token

        self.mask_index = self.add_token(self._mask_token)
        self.unk_index = -1
        if add_unk:
            self.unk_index = self.add_token(unk_token)
```

CBOW Vectorizer 클래스 선언

- 문맥의 토큰 수가 최대 길이보다 적으면, 나머지 항목은 MASK 토큰으로 패딩됨
- MASK 토큰으로 패딩된 경우, 이후 학습에서 사용되지 않음

```python
class CBOWVectorizer(object):
    """ 어휘 사전을 생성하고 관리합니다 """
    def __init__(self, cbow_vocab):
        """
        매개변수:
            cbow_vocab (Vocabulary): 단어를 정수에 매핑합니다
        """
        self.cbow_vocab = cbow_vocab

    def vectorize(self, context, vector_length=-1):
        """
        매개변수:
            context (str): 공백으로 나누어진 단어 문자열
            vector_length (int): 인덱스 벡터의 길이 매개변수
        """

        indices = [self.cbow_vocab.lookup_token(token) for token in context.split(' ')]
        if vector_length < 0:
            vector_length = len(indices)

        out_vector = np.zeros(vector_length, dtype=np.int64)
        out_vector[:len(indices)] = indices
        out_vector[len(indices):] = self.cbow_vocab.mask_index

        return out_vector
```

```python
    @classmethod
    def from_dataframe(cls, cbow_df):
        """데이터셋 데이터프레임에서 Vectorizer 객체를 만듭니다

        매개변수::
            cbow_df (pandas.DataFrame): 타깃 데이터셋
        반환값:
            CBOWVectorizer 객체
        """
        cbow_vocab = Vocabulary()
        for index, row in cbow_df.iterrows():
            for token in row.context.split(' '):
                cbow_vocab.add_token(token)
            cbow_vocab.add_token(row.target)

        return cls(cbow_vocab)
```

CBOW 모델

- 맥락을 이용해 타겟 단어를 예측한다. (분류 모델)

- 핵심 요소 3가지
  1. Embedding 층 사용
     - 문맥의 단어를 나타내는 인덱스를 각 단어에 대한 벡터로 만듦
  2. 전반적인 문맥을 감지하도록 벡터를 결합
  3. Linear 층에서 문맥 벡터를 사용해 예측 벡터를 계산
     - 예측 벡터는 전체 어휘 사전에 대한 확률 분포

모델 선언

self.embedding
(vocab size, embedding_size)
형태의 가중치 행렬

맥락 토큰을 입력으로 받아
임베딩 벡터로 변환

맥락 임베딩의 각 차원을
더해 하나의 벡터로 만듦

```python
class CBOWClassifier(nn.Module): # Simplified cbow Model
    def __init__(self, vocabulary_size, embedding_size, padding_idx=0):
        """
        매개변수:
            vocabulary_size (int): 어휘 사전 크기, 임베딩 개수와 예측 벡터 크기를 결정합니다
            embedding_size (int): 임베딩 크기
            padding_idx (int): 기본값 0; 임베딩은 이 인덱스를 사용하지 않습니다
        """
        super(CBOWClassifier, self).__init__()

        self.embedding =  nn.Embedding(num_embeddings=vocabulary_size,
                                       embedding_dim=embedding_size,
                                       padding_idx=padding_idx)
        self.fc1 = nn.Linear(in_features=embedding_size,
                             out_features=vocabulary_size)

    def forward(self, x_in, apply_softmax=False):
        """분류기의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
                x_in.shape는 (batch, input_dim)입니다.
            apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
                크로스-엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape은 (batch, output_dim)입니다.
        """
        x_embedded_sum = F.dropout(self.embedding(x_in).sum(dim=1), 0.3)
        y_out = self.fc1(x_embedded_sum)

        if apply_softmax:
            y_out = F.softmax(y_out, dim=1)

        return y_out
```

⇒ 타겟 앞뒤에 context word가 없을 때 padding 추가

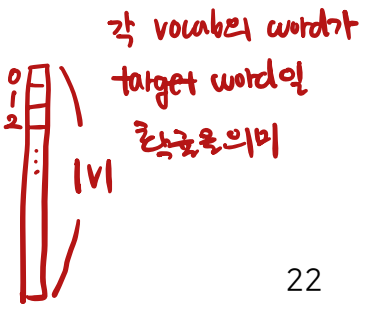
↳ embedding size → vocab size

⇒ 최종 output

self.fc1
(embedding_size,
vocab_size)
형태의 가중치 행렬

맥락 임베딩을 입력으로 받아
타겟 단어 인덱스 예측

출력 차원 수는 vocab 크기



학습 - 1

```
# CUDA 체크
if not torch.cuda.is_available():
    args.cuda = False

args.device = torch.device("cuda" if args.cuda else "cpu")

# 데이터셋과 Vectorizer
dataset = CBOWDataset.load_dataset_and_make_vectorizer(args.cbow_csv)
vectorizer = dataset.get_vectorizer()

# 모델
classifier = CBOWClassifier(vocabulary_size=len(vectorizer.cbow_vocab),
                            embedding_size=args.embedding_size)
classifier = classifier.to(args.device)

# 손실 함수와 옵티마이저
loss_func = nn.CrossEntropyLoss()
optimizer = optim.Adam(classifier.parameters(), lr=args.learning_rate)
scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer=optimizer,
                                                 mode='min', factor=0.5,
                                                 patience=1)

train_state = make_train_state(args)
```

학습 - 2

```python
for epoch_index in range(args.num_epochs):
    train_state['epoch_index'] = epoch_index

    # 훈련 세트에 대한 순회
    # 훈련 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정
    dataset.set_split('train')
    batch_generator = generate_batches(dataset,
                                       batch_size=args.batch_size,
                                       device=args.device)
    running_loss = 0.0
    running_acc = 0.0
    classifier.train()
```

훈련 코드는 로지스틱 회귀 구현과 거의 동일

```python
    for batch_index, batch_dict in enumerate(batch_generator):
        # 훈련 과정은 5단계로 이루어집니다

        # --------------------------------------
        # 단계 1. 그레이디언트를 0으로 초기화합니다
        optimizer.zero_grad()

        # 단계 2. 출력을 계산합니다
        y_pred = classifier(x_in=batch_dict['x_data'])

        # 단계 3. 손실을 계산합니다
        loss = loss_func(y_pred, batch_dict['y_target'])
        loss_t = loss.item()
        running_loss += (loss_t - running_loss) / (batch_index + 1)

        # 단계 4. 손실을 사용해 그레이디언트를 계산합니다
        loss.backward()

        # 단계 5. 옵티마이저로 가중치를 업데이트합니다
        optimizer.step()
        # --------------------------------------

        # 정확도를 계산합니다
        acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
        running_acc += (acc_t - running_acc) / (batch_index + 1)

        # 진행 바 업데이트
        train_bar.set_postfix(loss=running_loss, acc=running_acc,
                              epoch=epoch_index)
        train_bar.update()

    train_state['train_loss'].append(running_loss)
    train_state['train_acc'].append(running_acc)
```

검증

```
    # 검증 세트와 배치 제너레이터 준비, 손실과 정확도를 0으로 설정
    dataset.set_split('val')
    batch_generator = generate_batches(dataset,
                                       batch_size=args.batch_size,
                                       device=args.device)
    running_loss = 0.
    running_acc = 0.
    classifier.eval()

    for batch_index, batch_dict in enumerate(batch_generator):

        # 단계 1. 출력을 계산합니다
        y_pred =  classifier(x_in=batch_dict['x_data'])

        # 단계 2. 손실을 계산합니다
        loss = loss_func(y_pred, batch_dict['y_target'])
        loss_t = loss.item()
        running_loss += (loss_t - running_loss) / (batch_index + 1)

        # 단계 3. 정확도를 계산합니다
        acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
        running_acc += (acc_t - running_acc) / (batch_index + 1)
        val_bar.set_postfix(loss=running_loss, acc=running_acc,
                            epoch=epoch_index)
        val_bar.update()

    train_state['val_loss'].append(running_loss)
    train_state['val_acc'].append(running_acc)

    train_state = update_train_state(args=args, model=classifier,
                                     train_state=train_state)

    scheduler.step(train_state['val_loss'][-1])

    if train_state['stop_early']:
        break
```

평가

```python
# 가장 좋은 모델을 사용해 테스트 세트의 손실과 정확도를 계산합니다
classifier.load_state_dict(torch.load(train_state['model_filename']))
classifier = classifier.to(args.device)
loss_func = nn.CrossEntropyLoss()

dataset.set_split('test')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):
    # 출력을 계산합니다
    y_pred =  classifier(x_in=batch_dict['x_data'])

    # 손실을 계산합니다
    loss = loss_func(y_pred, batch_dict['y_target'])
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # 정확도를 계산합니다
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)

train_state['test_loss'] = running_loss
train_state['test_acc'] = running_acc
```

```python
print("테스트 손실: {};".format(train_state['test_loss']))
print("테스트 정확도: {}".format(train_state['test_acc']))
```

```
테스트 손실: 8.131600935318888;
테스트 정확도: 11.551470588235299
```

→ 정확도가 높지 않은 이유

1. 간단한 임베딩 방법
2. 데이터셋이 작음

훈련 임베딩 사용

```python
def pretty_print(results):
    """
    임베딩 결과를 출력합니다
    """
    for item in results:
        print ("...[%.2f] - %s"%(item[1], item[0]))

def get_closest(target_word, word_to_idx, embeddings, n=5):
    """
    n개의 최근접 단어를 찾습니다.
    """

    # 다른 모든 단어까지 거리를 계산합니다
    word_embedding = embeddings[word_to_idx[target_word.lower()]]
    distances = []
    for word, index in word_to_idx.items():
        if word == "<MASK>" or word == target_word:
            continue
        distances.append((word, torch.dist(word_embedding, embeddings[index])))

    results = sorted(distances, key=lambda x: x[1])[1:n+2]
    return results
```

```python
word = input('단어를 입력해 주세요: ')
embeddings = classifier.embedding.weight.data
word_to_idx = vectorizer.cbow_vocab._token_to_idx
pretty_print(get_closest(word, word_to_idx, embeddings, n=5))
```

```
단어를 입력해 주세요: monster
...[7.32] - cares
...[7.58] - griefs
...[7.63] - sickness
...[7.66] - trifling
...[7.69] - saw
...[7.70] - prolong
```

훈련 임베딩 사용

```python
target_words = ['frankenstein', 'monster', 'science', 'sickness', 'lonely', 'happy']

embeddings = classifier.embedding.weight.data
word_to_idx = vectorizer.cbow_vocab._token_to_idx

for target_word in target_words:
    print(f"========{target_word}==========")
    if target_word not in word_to_idx:
        print("Not in vocabulary")
        continue
    pretty_print(get_closest(target_word, word_to_idx, embeddings, n=5))
```

```
=======frankenstein=======
...[6.93] - discrimination
...[6.99] - slight
...[7.02] - oppressive
...[7.05] - spurned
...[7.11] - illustrate
...[7.11] - wandering
=======monster=======
...[7.32] - cares
...[7.58] - griefs
...[7.63] - sickness
...[7.66] - trifling
...[7.69] - saw
...[7.70] - prolong
=======science=======
...[6.85] - mutual
...[6.93] - mist
...[6.95] - swelling
...[7.01] - impression
...[7.06] - darkened
...[7.06] - nearly
```

```
========sickness========
...[6.37] - while
...[6.45] - foundations
...[6.61] - awoke
...[6.65] - consoles
...[6.70] - literally
...[6.74] - depend
========lonely========
...[6.74] - unveiled
...[6.88] - moonlight
...[7.05] - ought
...[7.08] - bed
...[7.14] - superhuman
...[7.14] - therefore
========happy========
...[6.25] - bottom
...[6.42] - injury
...[6.49] - chivalry
...[6.50] - altered
...[6.51] - penetrated
...[6.54] - danger
```

실습

- 파일명: 14강_실습_word2vec.ipynb
- #TRY IT YORSELF 주석 처리된 부분은 강의자료를 보고 코드를 완성합니다.
- 코드의 흐름을 분석해보고, 이해되지 않는 경우 편하게 질문해 주세요.