단어 임베딩 기법 Word2Vector 구현하기

In [ ]:

```
# 사용 모듈 import 할게요

# 오늘은 numpy말고 python 내장모듈 random 으로 진행 할게요.
import random

# word2Vector 구현한거 사용해기 위해서 gensim 모듈 import 할게요.
import gensim

# 데이터 읽어오기 위해서 pandas import 할게요.
import pandas as pd
import numpy as np
from pprint import pprint

# 한글 전처리 위해서 Okt import 할게요
from konlpy.tag import Okt

# 사용할 keras 모듈 import 할게요.
from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, Dense, Softmax, Activation, Input, Dot, Resh
ape
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.utils import model_to_dot
from IPython.display import SVG
```

In [ ]:

```
# pandas로 챗봇 데이터 읽어 올게요.
d_path = '../data/Chatbot_data-master/ChatbotData.csv'
df = pd.read_csv(d_path)

# 컬럼 확인할게요.
# Q는 질문이고 A는 답이고 label은 0: 중립, 1:부정, 2:긍정 이에요.
print(df.columns)
```

In [ ]:

```
# 질문 데이터만 확인할게요.
kor_df = df['Q']
print(kor_df)
```

In [ ]:

```
# DataFrame에서 필요한 values만 추출
# 질문 데이터만 가져오고 확인할게요.
kor_sentences = kor_df.values
pprint(kor_sentences[:5])
```

In [ ]:

```
# okt로 문장 전치리 할게요.
# 먼저 실행되는지 확인할게요.
okt = Okt()
print(okt.pos(kor_sentences[0]))
```

In [ ]:

```
# 각 문장단위로 토크나이즈하고 확인할게요.
# 한 10초정도 걸릴거에요.
kor_word_splited = [okt.pos(sentence) for sentence in kor_sentences]
print(kor_word_splited[:10])

# 위 2줄이 아래 6줄이랑 동일한 의미에요.
kor_word_splited = []
for idx, sentence in enumerate(kor_sentences):
    posed_words = okt.pos(sentence)
    kor_word_splited.append(posed_words)
    if idx < 10:
        print(posed_words)
```

In [ ]:

```
# 데이터 입력, 출력 페어 생성 함수 작성
def make_pair(sentences, window_size, negative_sample):
    # 학습의 입력데이터 저장할 x_train 이랑
    # 학습의 출력데이터 저장할 y_train 생성할게요.
    x_train = []
    y_train = []

    # window_size는 타겟 word 기준이니까
    # 좌, 우 에서 단어 가져오기 위해서 winsdow_size를
    # 반으로 줄여 줄게요.
    # // 는 나누기해서 몫만 가져오는 연산자에요.
    half_window_size = window_size // 2

    # 토크나이즈 진행한 문장들로 진행하는거에요.
    for sentence in sentences:

        # 각 문장의 길이(단어의 개수)만큼 진행하면서
        # slicing으로 필요한 부분한
        # 추출하는 for문이에요.
        for i in range(len(sentence)):
            # sentence(list)가 비어있으면 넘어가도록
            # 하는 if 문이에요.
            if not sentence:
                continue

            # 지금 확인할 단어가 half_window_size보다 작을때
            # 어떻게 진행할지 처리하는 if문 이에요.
            # list의 앞쪽이에요.
            if i < half_window_size:
                temp_data = sentence[0:i+half_window_size+1]

            # 지금 확인할 단어가 sentence의 길이랑 같을때
            # 어떻게 진행할지 처리하는 elif문 이에요.
            # list의 마지막쪽이에요.
            elif i == len(sentence):
                temp_data = sentence[i-half_window_size:i+1]

            # list의 앞과 뒤를 제외한 부분들을 어떻게
            # 진행할지 처리하는 else문이에요.
            else:
                temp_data = sentence[i-half_window_size:i+half_window_size+1]

            # slicing으로 추출한 데이터의 길이가
            # window_size랑 같으면, 중앙에 있는 단어가
            # 학습의 출력 데이터가 되고,
            # 나머지가 학습의 입력 데이터가 되요.
            if len(temp_data) == window_size:
                temp_y_idx = half_window_size

            # 길이가 다르면 조정해줘야 되요.
            else:
                temp_y_idx = len(temp_data) - (half_window_size+1)

            # slice한 현재 데이터 기준으로
            # 학습의 입력과 출력을 나눠주는 부분이에요
            # 입력을 위해 temp_x
            # 출력을 위해 temp_y 생성할게요.
            temp_x, temp_y = [], []
```

```
        # for문은 slice한 현재 데이터의 길이만큼 진행되요.
        for idx in range(len(temp_data)):

            # 전체를 돌기때문에 idx가
            # 위에서 찾은 출력 데이터의 idx랑 같으면
            # 넘어가도록 할게요.
            if idx == temp_y_idx:
                continue

            # (입력, 출력) 패어로 만들고, temp_x에 저장할게요.
            temp_x.append((temp_data[idx][0], temp_data[temp_y_idx][0]))

            # 지금의 (입력, 출력) 패어가 True:1 인지 False:0 인지
            # temp_y에 저장할게요.
            temp_y.append(1)

        # 이번에는 negarive_sample의 수만큼 진행하는 for문이에요.
        for j in range(negative_sample):

            # 현재 sentence copy()할게요.
            negative_sentence = sentence.copy()

            # negative_sentence에서 현재 sentence를 제거하면,
            # 나머지는 전부 window_size 밖이니까
            # 전부 negative sample이 되요.
            for item in temp_data:
                while item in negative_sentence:
                    negative_sentence.remove(item)

            # negative_sentence가 비어있으면 진행 불가니까
            # 넘어가도록 하는 if문이에요.
            if not negative_sentence:
                continue

            # 위랑 동일하게 진행할게요.
            for idx in range(len(temp_data)):

                # negative할 단어를 선택하기 위해서
                # 숫자를 random으로 선택하고, 해당 단어를 찾을게요.
                random_number = random.randrange(0, len(negative_sentence))
                negative_word = negative_sentence[random_number]

                if idx == temp_y_idx:
                    continue

                # (입력, 출력) 패어로 만들고, temp_x에 저장할게요.
                temp_x.append((temp_data[idx][0], negative_word[0]))

                # 지금의 (입력, 출력) 패어가 True:1 인지 False:0 인지
                # temp_y에 저장할게요.
                temp_y.append(0)

        # 찾은 패어들을 x_train이랑 y_train이 저장할게요.
        x_train.append(temp_x)
        y_train.append(temp_y)

    # 찾은 모든 패어들을 리턴할게요.
    return x_train, y_train
```

In [ ]:

```
# 윈도우 사이즈 설정할게요.
# 윈도우 사이즈는 타겟 단어를 포함해서
# 좌우 몇 개의 단어를 확인할거냐를 나타내는 거에요.
window_size = 3

# 네거티브 샘플 개수 설정할게요.
# 네거티브 샘플 개수는 윈도우 사이즈 밖에 있는
# 단어들 중 몇 개의 단어를 사용할거냐를 나타내는 에요.
negative_sample = 1
```

In [ ]:

```
# 작성한 함수 사용해서 학습의 입력이랑 출력으로 나눠 줄게요.
x_train, y_train = make_pair(kor_word_splited, window_size, negative_sample)
```

In [ ]:

```
# 집합으로 중복을 제거해서
# 총 몇개의 토큰들로 이루어 지는지 확인할게요.
words_set = list(sorted(set([word for sentence in kor_word_splited for word, pos in sentence
])))

# 위 1줄이랑, 아래 5줄이 동일한 내용이에요.
words_set = set()
for sentence in kor_word_splited:
    for word, pos in sentence:
        words_set.add(word)
words_set = list(sorted(words_set))


print(len(words_set))
print(words_set[:5])
print(words_set[-5:])
```

In [ ]:

```
# 입력 출력 페어 확인할게요.
# 왼쪽에 있는 하나는 (입력, 출력) 페어가
# True면 1, False면 0 으로 오른쪽에 list로 출력되요.
for tmp_x, tmp_y in zip(x_train, y_train):
    print(tmp_x, ' : ', tmp_y)
```

In [ ]:

```
# 단어 임베딩 사이즈 설정할게요.
# 임베딩 사이즈는 하시고 싶은 만큼 하시면되요.
# Embedding Layer의 output_dim으로 사용할거에요.
embedding_size = 50

# 전체 단어 개수 설정할게요.
# Embedding Layer의 input_dim으로 사용할거에요.
vocab_size = len(words_set)
```

In [ ]:

```
# (입력, 출력) 패어에서
# 입력 단어의 인풋 및 임베딩 레이어 생성하면서
# 연결할게요.
x_inputs = Input(shape=(1,), dtype='int32')
x_embedding = Embedding(vocab_size, embedding_size)(x_inputs)
```

In [ ]:

```
# (입력, 출력) 패어에서
# 출력 단어의 인풋 및 임베딩 레이어 생성하면서
# 연결할게요.
y_inputs = Input(shape=(1,), dtype='int32')
y_embedding = Embedding(vocab_size, embedding_size)(y_inputs)
```

In [ ]:

```
# 입력과 출력 두 벡터의
# Dot product 연산을 위한 레이어와
# Reshape 레이어,
# sigmoid 사용하는 출력 레이어 생성하면서
# 연결할게요.
dot_product = Dot(axes=2)([x_embedding, y_embedding])
dot_product = Reshape((1,), input_shape=(1, 1))(dot_product)
output = Activation('sigmoid')(dot_product)
```

In [ ]:

```
# 모델 생성할게요.
model = Model(inputs=[x_inputs, y_inputs], outputs=output)
model.summary()
```

In [ ]:

```
# 모델 시각화 해볼게요.
SVG(model_to_dot(model, show_shapes=True, dpi=65).create(prog='dot', format='svg'))
```

In [ ]:

```
# 모델 컴파일
model.compile(loss='binary_crossentropy', optimizer='adam')
```

In [ ]:

```
# 모델 100번 학습할게요.
for epoch in range(1, 101):
    print('Epoch: ', epoch, end=' ')
    loss = 0

    # 전부 다하면 시간이 오래걸리니까 처음부터 50개만 진행할게요.
    # 다 해보셔도 되요. 저는 안해봐서 얼마나 걸릴지 모르겠어요.
    # zip(x, y)함수는 파이썬 내장 함수로,
    # 길이가 동일한 x와 y에서 동일한 위치에 있는 값을 동시에 반환해주는 함수에요.

#     for x, y in zip(x_train[:50], y_train[:50]):

    # 값 출력을 위해서 조금만 수정할게요.
    for idx, (x, y) in enumerate(zip(x_train[:50], y_train[:50])):

        # x1은 입력 데이터가 되고
        # x는 (입력, 출력) 패어니까 한번에 입력, 출력이 반환되요.
        # x1에서는 입력만 사용할거니까, word, _ 로해서 뒤에껀 사용하지 않습니다.
        # word를 words_set에서 몇 번째 단어인지 찾아 ndarray로 타입을 변경해줄게요.
        x1 = np.asarray([words_set.index(word) for word, _ in x]).astype('int32')

        # x2는 출력 데이터가 되고
        x2 = np.asarray([words_set.index(word) for _, word in x]).astype('int32')

        # (x1, x2) 패어의 (True, False) 레이블이에요.
        y = np.asarray(y).astype('int32')

        if epoch == 1 and idx < 10:
            print('₩nx1 : ', x1)
            print('x2 : ', x2)
            print('y : ', y)

        # 모델에 하나씩 학습하도록 할게요.
        loss += model.train_on_batch([x1, x2], y)
    print('Loss: ', loss)
```

In [ ]:

```
# 학습한 모델의 weight를 불러와서 파일에 쓰도록 할게요.

# 파일 생성하고, mode를 쓰기 모드인 'w'로 설정하고
# encoding을 'utf-8'로 설정할게요.
f = open('w2v.txt', 'w', encoding='utf-8')

# 첫 번째줄 한번 쓰고 넘어갈게요.
# 총 단어의 개수랑, embeeding_size를 작성해요.
f.write('{} {}₩n'.format(vocab_size-1, embedding_size))

# 모델에서 weight만 불러 올게요.
vectors = model.get_weights()[0]

# enumerate() 사용해서 몇 번째인지와 단어를 함께 받아올게요.
for i, word in enumerate(words_set):

    # 첫 줄이랑 동일한 포맷으로 파일에 쓰도록 할게요.
    # map(function, values) 함수는 파이썬 내장함수로,
    # 모든 value를 function으로 처리한 값을 list로 반환해 줘요.
    f.write('{} {}₩n'.format(word, ' '.join(map(str, list(vectors[i, :])))))

# 마지막에 파일 닫아줄게요.
# 안닫으시면 저장이 안되요.
f.close()
```

In [ ]:

```
# weight 작성된 파일을 gensim 모듈로
# 읽고 모델로 생성할게요.
w2v = gensim.models.KeyedVectors.load_word2vec_format('./w2v.txt', encoding='utf-8', binary=Fal
se)
```

In [ ]:

```
# 모델로 확인 단어 "학교"의 벡터를 확인해볼게요.
print(w2v['학교'])
```

In [ ]:

```
# 모델로 단어 "학교"와 비슷한 단어를 확인해볼게요.
# 앞이 단어, 뒤가 유사도에요.
w2v.most_similar(positive=['학교'])
```

In [ ]:

```
# 모델로 확인 단어 "지망"의 벡터를 확인해볼게요.
print(w2v['지망'])
```

In [ ]:

```
# 모델로 단어 "지망"와 비슷한 단어를 확인해볼게요.
w2v.most_similar(positive=['지망'])
```

In [ ]:

```
# 모델로 단어 "지망"이랑 "학교"의 유사도를 확인해볼게요.
# 위랑 동일한 값이 나오는지 볼게요.
w2v.similarity('지망', '학교')
```

In [ ]:

```
# 학습 안된 단어는 사용이 안되요.
w2v.most_similar(positive=['딥러닝'])
```

In [ ]:

```

```

In [ ]:

```
# 이번에는 gensim 모듈로 Word2Vec를 만들어 볼게요.
# 쉬워요. 이때까지 했던게 한줄로 완성이 되요.
model = gensim.models.Word2Vec(kor_sentences, size=50, window=3, min_count=1, workers=1)
```

In [ ]:

```
# 모델로 확인 단어 "학교"의 벡터를 확인해볼게요.
print(model.wv['학교'])
```

In [ ]:

```
# 모델로 단어 "학교"와 비슷한 단어를 확인해볼게요.
# 앞이 단어, 뒤가 유사도에요.
model.wv.most_similar(['학교'])
```

In [ ]:

```
# 모델로 확인 단어 "지망"의 벡터를 확인해볼게요.
print(model.wv['지망'])
```

In [ ]:

```
# 모델로 단어 "지망"와 비슷한 단어를 확인해볼게요.
model.wv.most_similar(positive=['지망'])
```

In [ ]:

```
# 모델로 단어 "지망"이랑 "학교"의 유사도를 확인해볼게요.
# 위랑 동일한 값이 나오는지 볼게요.
model.wv.most_similar(positive=['학교'], negative=['지망'])
```

In [ ]:

```
# gensim으로 생성한 모델 저장해볼게요.
model.save('w2v.model')
```

In [ ]:

```
# gensim으로 저장한 모델 다시 불러와볼게요.
model = gensim.models.Word2Vec.load('w2v.model')
```

In [ ]:

```
# 동일한 값이 나오는지 한번 확인해 볼게요.
model.wv.most_similar(positive=['학교'], negative=['지망'])
```

In [ ]:

```

```