단어 임베딩 기법 FastText 구현하기

In [ ]:

```
# 사용 모듈 import 할게요
# 거의다 word2vec이랑 동일해요.
import random
import gensim
import pandas as pd
import numpy as np
from pprint import pprint
from konlpy.tag import Okt

# 한글을 자모로 나눠서 할거니까
# jamo만 하나더 import 할게요.
from jamo import h2j, j2h

from tensorflow import keras
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Embedding, Softmax, Activation, Input, Dot, Reshape
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)

# 동일하게 컬럼만 한번더 확인할게요
print(df.columns)
```

In [ ]:

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

In [ ]:

```
# 이번에는 애초에 50개만 사용할게요.
kor_sentences = list(kor_df.values)[:50]
kor_sentences[:5]
```

In [ ]:

```
# 문장 전처리는 동일해요.
okt = Okt()
kor_word_splited = [okt.pos(sentence) for sentence in kor_sentences]
```

In [ ]:

```
# words_set 생성하는 함수 작성할게요.
# 이번에는 구두점("punctuation") 제거하도록 할게요.
def make_words_set(sentences):
    words_set = set()
    for sentence in sentences:
        for word, pos in sentence:
            if pos == "Punctuation":
                continue
#             print(word)
            words_set.add(word)
    return list(sorted(words_set))
```

In [ ]:

```
# 작성한 함수로 words_set 생성할게요.
words_set = make_words_set(kor_word_splited)
```

In [ ]:

```
# words_set 확인해볼게요.
print(len(words_set))
print(words_set[:5])
print(words_set[-5:])
```

In [ ]:

```
# 단어->자음 모음으로 변경하면서
# 단어 앞뒤로 특수기호 '<'랑 '>' 넣어주면서
# 자음모음을 n-gram 변환하는 함수 작성할게요
def word2ngram(word, n):
    ngram = []
    ws = '<'+h2j(word)+'>'

    for i in range(len(ws)-(n-1)):
        ngram.append(ws[i:i+n])
    return ngram
```

In [ ]:

```
# 이번 실습에서는 tri-gram만 사용할거에요.
# tri_gram_set이랑
# 단어: tri_gram Dictionary 생성할게요.
tri_gram_set = set()
tri_gram = {}

for word in words_set:
    ngram = word2ngram(word, 3)
    tri_gram[word] = ngram

    # set.update()는 ()안에 있는 모든 데이터를
    # 한번에 set에 입력해주는 함수에요.
    tri_gram_set.update(ngram)

tri_gram_set = list(sorted(tri_gram_set))
```

In [ ]:

```
# tri-gram 딕셔너리 및 길이 확인할게요.
pprint(tri_gram)
max_len = (max([len(gram) for gram in tri_gram.values()]))
print(max_len)
```

In [ ]:

```
# tri-gram set 길이랑 데이터 확인할게요.
tri_gram_size = len(tri_gram_set)
print(tri_gram_size)
print(tri_gram_set[:5])
print(tri_gram_set[-5:])
```

In [ ]:

```
# 데이터 입력, 출력 페어 생성 함수 작성
# word2vec이랑 동일해요.
# 대신 입력되는 데이터가 단일 문장이에요.
def make_pair(sequence, window_size, negative_sample):
    x, y = [], []
    half_window_size = window_size // 2

    # 이번에는 문장에 정보가 없으면
    # Error를 발생시킬게요.
    if not sequence:
        raise ValueError('No data in sequence')

    for i in range(len(sequence)):
        if i < half_window_size:
            temp_data = sequence[0:i+half_window_size+1]
        elif i == len(sentence):
            temp_data = sequence[i-half_window_size:i+1]
        else:
            temp_data = sequence[i-half_window_size:i+half_window_size+1]

        if len(sequence) < window_size:
            temp_idx = i
        elif len(temp_data) == window_size:
            temp_idx = half_window_size
        else:
            temp_idx = len(temp_data) - (half_window_size+1)

        for idx in range(len(temp_data)):
            if idx == temp_idx:
                continue
            x.append((temp_data[idx], temp_data[temp_idx]))
            y.append(1)

        for _ in range(negative_sample):
            negative_sentence = sequence.copy()
            for item in temp_data:
                negative_sentence.remove(item)
            if not negative_sentence:
                continue
            random_number = random.randrange(0, len(negative_sentence))
            negative_word = negative_sentence[random_number]
            for idx in range(len(temp_data)):
                if idx == temp_idx:
                    continue
                x.append((temp_data[idx], negative_word))
                y.append(0)
    return x, y
```

In [ ]:

```
# 데이터 페어 생성 및 확인
NX, NY = [], []
for idx, sentence in enumerate(kor_sentences):
    words = [word for word, _ in okt.pos(sentence) if word in words_set]
    # Error 발생 시키려고 임시로 데이터 하나 추가해 볼게요.
    words = []
    # 확인만 해보시고 지우시거나 주석처리 해주세요.
    x, y = make_pair(words, window_size=3, negative_sample=1)

    for word in words:
        ngram = tri_gram[word]
        n_x, n_y = make_pair(ngram, window_size=3, negative_sample=1)
        NX += n_x
        NY += n_y

    if idx == 0:
        print(NX)
        print(NY)
```

In [ ]:

```
# 파라미터 설정할게요.
# 아까랑 거의 비슷해요.
vocab_size = len(words_set)
tri_gram_size = len(tri_gram_set)
embedding_size = 50
```

In [ ]:

```
# 모델 생성하고 summary()로 확인해 볼게요.
# 같은데, Embedding Layer에 단어 개수가 아니라
# tri_gram의 개수가 들어가요.
sub_x_inputs = Input(shape=(1,), dtype='int32')
sub_x_embedding = Embedding(tri_gram_size, embedding_size)(sub_x_inputs)

sub_y_inputs = Input(shape=(1,), dtype='int32')
sub_y_embedding = Embedding(tri_gram_size, embedding_size)(sub_y_inputs)

sub_dot_product = Dot(axes=2)([sub_x_embedding, sub_y_embedding])
sub_dot_product = Reshape((1,), input_shape=(1, 1))(sub_dot_product)
sub_output = Activation('sigmoid')(sub_dot_product)

sub_model = Model(inputs=[sub_x_inputs, sub_y_inputs], outputs=sub_output)
sub_model.summary()
```

In [ ]:

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

In [ ]:

```
# 모델 컴파일할게요.
sub_model.compile(loss='binary_crossentropy', optimizer='adam')
```

In [ ]:

```
# 모델 10번만 학습 진행할게요.
# 조금 오래 걸려요.
for epoch in range(1, 11):
    print('Epoch: ', epoch, end=' ')
    loss = 0
    for x, y in zip(NX, NY):
        # 아까는 for문으로 진행했던걸 이번에는
        # indexing으로 진행할게요.
        x1 = np.asarray(tri_gram_set.index(x[0])).astype('int32').reshape(1, -1)
        x2 = np.asarray(tri_gram_set.index(x[1])).astype('int32').reshape(1, -1)
        y = np.asarray(y).astype('int32').reshape(1, -1)
        loss += sub_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('sub_fasttext.txt', 'w', encoding='utf-8')

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

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

# enumerate() 사용해서 몇 번째인지와 단어를 함께 받아올게요.
for i, word in enumerate(tri_gram_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.
ft = gensim.models.KeyedVectors.load_word2vec_format('./sub_fasttext.txt', encoding='utf-8', bin
ary=False)
```

In [ ]:

```
# 단어는 sum(n-gram[단어]) 이니까
# 단어의 벡터로 바꿔주는 생성 함수 작성할게요.
def get_word_vector(model, word):
    res = []
    for gram in word2ngram(word, 3):
#         print(gram)
        # 학습이 안된 tri_gram은 continue로
        # 넘어가도록 작성할게요.
        try:
            res.append(model[gram])
        except KeyError:
            continue
    return np.sum(res, axis=0)
```

In [ ]:

```
# 모델로 확인 단어 "학교"의 벡터를 확인해볼게요.
# 모델 학습할때 tri-gram으로 학습 시켜서
# 단어로 입력하면, Error 나요.
print(ft['학교'])
```

In [ ]:

```
# 모델로 확인 단어 "학교"의 벡터를 확인해볼게요.
# 하까 작성한 함수로 진행할게요.
wv_1 = get_word_vector(ft, '학교')
print(wv_1)
```

In [ ]:

```
# 모델로 확인 단어 "핵교"의 벡터를 확인해볼게요.
# 아까 작성한 함수로 진행할게요.
wv_2 = get_word_vector(ft, '핵교')
print(wv_2)
```

In [ ]:

```
# 코사인 유사도 함수 작성할게요.
# -1 ~ 1 값 출력 (-1: 전혀 다름, 1: 완전 동일)
def cos_sim(A, B):
       return np.dot(A, B)/(np.linalg.norm(A)*np.linalg.norm(B))
```

In [ ]:

```
# '학교' 과 '핵교'의 유사도 계산 해볼게요.
cosim = cos_sim(wv_1, wv_2)
print(cosim)
```

In [ ]:

```
# 모델로 확인 단어 "한글"의 벡터를 확인해볼게요.
# 아까 작성한 함수로 진행할게요.
wv_3 = get_word_vector(ft, '한글')
print(wv_3)
```

In [ ]:

```
# '학교'과 '한글'의 유사도 계산 해볼게요.
cosim = cos_sim(wv_1, wv_3)
print(cosim)
```

In [ ]:

```
# 모델로 확인 단어 '딥러닝'의 벡터를 확인해볼게요.
wv_4 = get_word_vector(ft, '딥러닝')
print(wv_4)
```

In [ ]:

```
# '학교'과 '딥러닝'의 유사도
cosim = cos_sim(wv_1, wv_4)
print(cosim)
```

In [ ]:

```

```

In [ ]:

```
# 이번에는 gensim 모듈로 FastText 만들어 볼게요.
model = gensim.models.Word2Vec(kor_sentences, size=50, window=3, min_count=1, workers=1)
```

In [ ]:

```
kor_sentences = [[word for word, pos in sentence] for sentence in kor_word_splited]
print(kor_sentences[:5])
```

In [ ]:

```
model = gensim.models.FastText(kor_sentences, size=50, window=3, min_count=1, iter=10)
```

In [ ]:

```
# 본 실습에서는 sub_model만 진행했어요.
# 전체 모델은 sub_model을 기반으로,
# 다시 word2vec 처럼 단어 수준에서 skip-gram을
# 진행해요. 그래서 조금 다른값이 나올거에요.
model.wv.similarity('한국', '한쿡')
```

In [ ]:

```
# '학교'랑 비슷한 단어를 찾아볼게요.
model.wv.most_similar('학교')
```

In [ ]:

```

```