지능시스템설계 HW. 9

2021220699 전자전기공학부 이은찬

Title

밑바닥부터 시작하는 딥러닝2 실습 보고서

목차

목적

1. 강의 자료에 있는 모든 코드를 직접 코딩해본다 (실습)
2. 각 코드 블럭에 대한 설명을 재해석하여 덧붙인다. (공부)

Chap02 - 자연어와 단어의 분산 표현

1. **Input text sequence를 단어의 리스트로 만든다.**
2. 두 개의 딕셔너리 word_to_id와 id_to_word를 for문 등으로 구현하였다.
3. word_to_id는 0~len(text)까지 인덱스를 Key로 받고 인덱스에 해당하는 단어를 리턴하므로 유용하다
4. id_to_word는 반대로 단어를 key로 받아 인덱스를 리턴한다

```
text = 'You say goodbye and I say hello.'
text = text.lower().replace('.', ' .')

words = text.split(' ')

word_to_id = {}
for word in words:
    if word not in word_to_id:
        new_id = len(word_to_id)
        word_to_id[word] = new_id

id_to_word = {id_:  word for word, id_ in word_to_id.items()}

'''
id_to_word : {0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
word_to_id : {'you': 0, 'say': 1, 'goodbye': 2, 'and': 3, 'i': 4, 'hello': 5, '.': 6}
'''
```

1. 위의 모든 전처리 과정을 하나의 함수로 만들었다 → preprocess
2. 예시 문장을 입력으로 받아 preprocess 과정이 잘 이루어지는지 본다

```
def preprocess(text):
    text = text.lower()
    text = text.replace('.', ' .')
    words = text.split(' ')

    word_to_id = {}
    id_to_word = {}
    for word in words:
        if word not in word_to_id:
            new_id = len(word_to_id)
            word_to_id[word] = new_id
            id_to_word[new_id] = word

    corpus = np.array([word_to_id[w] for w in words])

    return corpus, word_to_id, id_to_word

import sys
sys.path.append('..')
#from common.util import preprocess


text = 'Hi My name is James Louis Eunchan and you say Hi Eunchan'
corpus, word_to_id, id_to_word = preprocess(text)


'''
corpus = array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 6])
word_to_id = {'hi': 0 , 'my': 1, ... , 'say': 9}
id_to_word = {0: 'hi',  1: 'my', ... ,  9: 'say'}
'''
```

1. Co-occurrence Matrix를 생성하는 함수를 생성하여 이를 통해 **빈도수 기반**의 전처리를 이루게 한다
    - 특정 단어에 대해, 그 단어의 주변에 어떤 단어가 몇 번이나 등장하는지 카운팅하여 합치는 방법
2. 이를 예시 문장에 적용하면 각 단어의 주변 단어가 몇번 나오는지를 2차원 ARRAY에 모두 받을 수 있다.

```
def create_co_matrix(corpus, vocab_size, window_size=1):
    corpus_size = len(corpus)
    co_matrix = np.zeros((vocab_size, vocab_size),dtype=np.int32)

    for idx, word_id in enumerate(corpus):
        for i in range(1, window_size + 1):
            left_idx = idx - i # left window_size
            right_idx = idx + i # right window_size

            if left_idx >= 0:
                left_word_id = corpus[left_idx]
                co_matrix[word_id, left_word_id] += 1

            if right_idx < corpus_size:
                right_word_id = corpus[right_idx]
                co_matrix[word_id, right_word_id] += 1

    return co_matrix

print(enumerate(corpus))

for word_id in corpus: print(word_id)
for idx, word_id in enumerate(corpus): print(idx, word_id)
#for idx, word_id in corpus: print(idx, word_id) #error

window_size = 1 # 주변 1개
vocab_size = len(id_to_word)
C = create_co_matrix(corpus, vocab_size, window_size)


'''

C = array([[0, 1, 0, 0, 0, 0, 1, 0, 0, 1],
       [1, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 1, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 1, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
       [1, 0, 0, 0, 0, 1, 0, 1, 0, 0],
       [0, 0, 0, 0, 0, 0, 1, 0, 1, 0],
       [0, 0, 0, 0, 0, 0, 0, 1, 0, 1],
       [1, 0, 0, 0, 0, 0, 0, 0, 1, 0]])
'''
```

1. Cosine Similarity를 통해 벡터 간 유사도를 측정하기 위해 cos_similarity()를 구현한다.
    - 두 벡터의 방향성을 짐작할 수 있으며, -1~1까지 커질 수록 방향이 같고 작을 수록 방향은 반대가 된다
2. you와 i라는 두 단어의 주변 단어 빈도수를 통해 두 단어의 유사도를 Cosine Similarity로 측정해보았다
    - 결과는 0.707정도로 **유사도가 높음***(=1에 가깝다)*을 알 수 있다.
    - 실제로 YOU와 I는 매우 용도가 유사하므로 유의미한 결과임을 알 수 있다.
3. 함수 most_similar()를 구현하여 cos similarity 기반에서 검색한 단어와 유사한 단어의 랭킹을 보여주도록 한다
    - numpy의 argsort()는 key에 대해 정렬하는 대신 key의 순서가 바뀜에 따라 value도 같이 순서가 바뀌기에 유용하다.
4. 단어 'you'에 대해 most_similar()를 통해 유사 단어 랭킹을 출력한다

```
def cos_similarity(x, y, eps=1e-8):
    # epsilon 값을 추가해,
    # 0으로 나누기 오류가 나는 것을 막아줌
    nx = x / np.sqrt(np.sum(x**2) + eps) # x의 정규화
    ny = y / np.sqrt(np.sum(y**2) + eps) # y의 정규화

    return np.dot(nx, ny)

import sys
sys.path.append('..')

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)

c0 = C[word_to_id['you']]
c1 = C[word_to_id['i']]

print('you, i 코사인 유사도:'+cos_similarity(c0,c1))

# you, i 코사인 유사도: 0.7071067758832467

def most_similar(query, word_to_id, id_to_word, word_matrix, top=5):
    if query not in word_to_id:
        print(f'{query}(을)를 찾을 수 없습니다.')
        return
    print(f'\n[query] {query}')
    query_id = word_to_id[query]
    query_vec = word_matrix[query_id]

    vocab_size = len(id_to_word)
    similarity = np.zeros(vocab_size)
    for i in range(vocab_size):
        similarity[i] = cos_similarity(word_matrix[i], query_vec)
    count = 0
    for i in (-1* similarity).argsort():
        if id_to_word[i] == query:
            continue
        print(f' {id_to_word[i]}: {similarity[i]}')

        count += 1
        if count >= top:
            return


text = 'You say goodbye and I say hello'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size= len(word_to_id)
C = create_co_matrix(corpus, vocab_size)

print(most_similar('you', word_to_id, id_to_word, C, top=5))

'''
```

```
[query] you
 hello: 0.9999999900000001
 goodbye: 0.7071067758832467
 i: 0.7071067758832467
 say: 0.0
 and: 0.0
'''
```

1. 위의 방법을 조금 더 개선하기 위해 정보이론의 **ppmi**를 도입하여 코드를 짜보자
    - 양의 상호정보량(PPMI, Positive Pointwise Mutual Information)
    - 동시 발생 케이스를 고려하는 유사도 측정량으로, 동시에 발생하는 경우에 관련성을 더 높게 친다.
2. 예시 문장을 통해 PPMI와 CO-Matrix를 비교해본다
3. 실제로 단어 you에 대해 ppmi 기반의 유사 단어 랭킹을 예시로 출력해보았다.

```
def ppmi(C, verbose=False, eps=1e-8):

    M = np.zeros_like(C, dtype=np.float32)
    N = np.sum(C)
    S = np.sum(C, axis=0)
    total = C.shape[0] * C.shape[1]
    cnt = 0

    for i in range(C.shape[0]):
        for j in range(C.shape[1]):
            pmi = np.log2(C[i,j] * N / (S[i]*S[j]) + eps)
            M[i, j] = max(0, pmi)

            if verbose:
                cnt += 1
                if cnt % (total//100) == 0:
                    print(f'{(100*cnt/total):.2f} 완료')
    return M


text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)
W = ppmi(C)
np.set_printoptions(precision=3) # 유효 자릿수를 세 자리로 표시
print('Co-occurrence Matrix')
print(C)
print('-'*50)
print('PPMI')
print(W)

print(most_similar('you', word_to_id, id_to_word, W, top=5))
'''
Co-occurrence Matrix
[[0 1 0 0 0 0 0]
 [1 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 0 1 0]]
--------------------------------------------------
PPMI
[[0.    1.807 0.    0.    0.    0.    0.   ]
 [1.807 0.    0.807 0.    0.807 0.807 0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.    1.807 0.    1.807 0.    0.   ]
 [0.    0.807 0.    1.807 0.    0.    0.   ]
 [0.    0.807 0.    0.    0.    0.    2.807]
 [0.    0.    0.    0.    0.    2.807 0.   ]]


[query] you
 goodbye: 0.40786147117614746
 i: 0.40786147117614746
 hello: 0.2763834297657013
 say: 0.0
 and: 0.0
'''
```

1. 선형대수의 기법인 SVD(Singular Value Decomposition)를 통해서 차원 축소를 해보자
    - 차원 축소는 '중요한 정보'는 최대한 유지하면서 줄이는 것이 핵심이다.
    - Numpy의 선형대수 모듈인 np.linalg.svd()를 사용한다
    - 고차원(차원=단어 개수)의 데이터를 2차원 등으로 줄일 수 있다.
    - word embedding의 개념이라고 볼 수 있다.

```
import numpy as np
import matplotlib.pyplot as plt

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)
W = ppmi(C)
# SVD
U, S, V = np.linalg.svd(W)


print(C[0]) # 동시발생 행렬
print(W[0]) # PPMI 행렬
print(U[0]) # SVD
# 2차원으로 차원 축소하기
print(U[0, :2])

# 플롯
for word, word_id in word_to_id.items():
  plt.annotate(word, (U[word_id, 0], U[word_id, 1]))
plt.scatter(U[:,0], U[:,1], alpha=0.5)
plt.show()

'''

[0 1 0 0 0 0 0]
[0.    1.807 0.    0.    0.    0.    0.   ]
[ 3.409e-01 -1.110e-16 -4.441e-16 -1.205e-01  0.000e+00 -9.323e-01
 -1.086e-16]
[ 3.409e-01 -1.110e-16]

'''
```

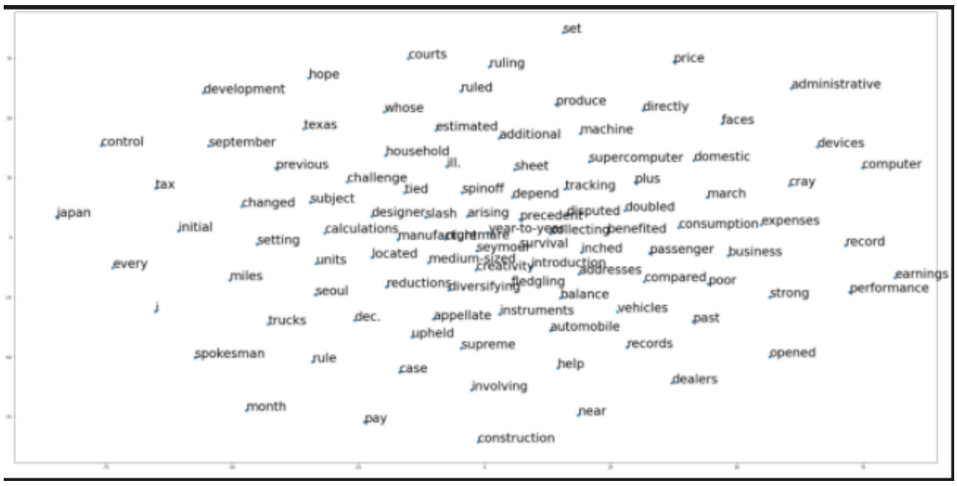
1. 대규모의 말뭉치를 가지고 있는 PTB 데이터셋을 이용하여 단어를 차원 축소해보자
    - PTB Dataset 말뭉치의 Length: 929589
2. 이 데이터셋에 SVD를 적용하고 유사도를 일부 살펴본다
3. 이를 통해서 최종적으로 모든 단어를 2차원으로 축소해서 PLOT을 출력하여 좌표 평면에 흩어진 단어들을 살펴 보자
    - 이 그림는 단어의 유사도를 거리를 통해 표현되게 된다

```
import sys
sys.path.append('..')
from dataset import ptb

corpus, word_to_id, id_to_word = ptb.load_data('train')

'''
말뭉치 크기: 929589
corpus[:30]: [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29]

id_to_word[0]: aer
```

```
id_to_word[1]: banknote
id_to_word[2]: berlitz

word_to_id['car']: 3856
word_to_id['happy']: 4428
word_to_id['lexus']: 7426
'''

window_size = 2
wordvec_size = 100

corpus, word_to_id, id_to_word = ptb.load_data('train')
vocab_size = len(word_to_id)
print('Create Co-Matrix ...')
C = create_co_matrix(corpus,vocab_size, window_size)

print('PPMI 계산...')
W = ppmi(C, verbose=True)

try:
    from sklearn.utils.extmath import randomized_svd
    U, S, V = randomized_svd(W, n_components=wordvec_size, n_iter=5,random_state=None)
except:
    U, S, V = np.linalg.svd(W)


word_vecs = U[:, :wordvec_size]
querys = ['you', 'year', 'car', 'toyota']

for query in querys:
    most_similar(query, word_to_id, id_to_word, word_vecs, top=5)


'''
[query] you
 i: 0.7067299485206604
 we: 0.6689852476119995
 do: 0.5927650332450867
 anybody: 0.5396730303764343
 'll: 0.5178669691085815

[query] year
 month: 0.7032848000526428
 quarter: 0.6717913150787354
 last: 0.629091739654541
 earlier: 0.5866899490356445
 next: 0.582098126411438

[query] car
 auto: 0.612777054309845
 corsica: 0.5656982660293579
 truck: 0.5479524731636047
 vehicle: 0.5002317428588867
 luxury: 0.47255587577819824

[query] toyota
 motor: 0.699421226978302
 nissan: 0.6756729483604431
 lexus: 0.6293434500694275
 motors: 0.5929510593414307
 honda: 0.5868485569953918
'''

%matplotlib inline
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False
tsne = TSNE(n_components=2)

# 100개의 단어에 대해서만 시각화
X_tsne = tsne.fit_transform(U[1000:1100,:])

vocab = list(id_to_word.values())

df = pd.DataFrame(X_tsne, index=vocab[1000:1100], columns=['x', 'y'])

df.head(10)

'''
x y
dec.  -25.712601  -34.736233
ruling  0.934758  71.363625
slash -11.687548  8.477380
earnings  81.321228 -16.838337
spokesman -57.374363  -50.257366
tracking  15.940672 20.198761
```

```
whose -19.829115  52.531094
addresses 18.684278 -15.217928
changed -48.074123  12.964805
past  41.325455 -35.261795
'''

fig = plt.figure()
fig.set_size_inches(40, 20)
ax = fig.add_subplot(1, 1, 1)

ax.scatter(df['x'], df['y'])

for word, pos in df.iterrows():
    ax.annotate(word, pos, fontsize=30)
plt.show()

'''
PLT의 OUTPUT은 아래와 같다
'''
```

PLT OUTPUT:



Chap03 - Word2Vec

- chap2에서 빈도수 기반의 통계 기반 기법을 썼다면 chap3에서 알아보는 Word2Vec은 추론 기반의 기법을 통한 word 전처리 기법이다.
- 추론 기법에서의 추론이란 주변 단어(맥락, context)가 주어졌을 때 "?"에 어떤 단어가 들어가는지 추측하는 것을 말한다.
- 신경망을 사용하기 위해서 모든 단어를 똑같은 차원의 벡터인 **One-hot 벡터화** 해주면 유용하다.

1. 아래 예제는 0번째 index가 활성화된 7차원의 원핫벡터를 3차원으로 축소하는 가장 간단한 예시이다.
    - 교재에서 제공하는 오픈소스의 Matmul Layer를 사용하였다. 이때 W는 훈련 전의 쓰레기값이다.

```
import sys
sys.path.append('..')
```

```
from common.layers import MatMul

c = np.array([[1, 0, 0, 0, 0, 0, 0]])
W = np.random.randn(7, 3)
layer = MatMul(W)
h = layer.forward(c)
print(h)

'''
[[-1.23123754 -0.48054396 -0.51973019]]
'''
```

1. 간단한 Word2Vec 중 **CBOW 모델**을 통해 주변 단어로 부터 타깃을 예측하는 모델을 구현해보자.
    - 아래 코드를 통해 훈련 이전의 신경망 모델링을 했고 Output은 의미가 없지만 훈련 이후에는 의미있는 정보가 될 것이다.
2. 학습을 진행하기 위해 준비한다.
    - 용이한 학습을 위해 window = 1의 주변 단어와 중심단어를 훈련용으로 전처리해서 준비한다
    - contexts_target 데이터를 만들기 위한 함수를 구현한다. → create_contexts_target()

3. 학습용 데이터를 원핫 벡터로 변환한다. 이를 통해 신경망에 비로소 훈련시킬 수 있게 된다.
    - convert_one_hot() 함수 구현
    - 모든 단어의 차원을 같게 만들기 때문이다.

```
# 샘플 맥락 데이터
c0 = np.array([[1, 0, 0, 0, 0, 0, 0]])
c1 = np.array([[0, 0, 1, 0, 0, 0, 0]])

# 가중치 초기화
W_in = np.random.randn(7, 3)
W_out = np.random.randn(3, 7)
# 계층 생성
in_layer0 = MatMul(W_in)
in_layer1 = MatMul(W_in)
out_layer = MatMul(W_out)
# 순전파
h0 = in_layer0.forward(c0)
h1 = in_layer1.forward(c1)
h = 0.5 * (h0 + h1) # average
s = out_layer.forward(h) # score
print(s)

'''
[[-2.06259707 -0.96784074  0.61153868 -0.68539261 -0.83350964 -0.75444994 0.5633758 ]]
'''

from common.util import preprocess
text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)

'''
corpus : [0 1 2 3 4 1 5 6]
id_to_word: {0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
'''

def create_contexts_target(corpus, window_size=1):
    target = corpus[window_size:-window_size]
    contexts = []

    for idx in range(window_size, len(corpus)-window_size):
        cs = []
        # wiondow_size만큼 타겟 단어 좌우 context 가져오기
        for t in range(-window_size, window_size+1):
            if t != 0:
                cs.append(corpus[idx + t])

        contexts.append(cs)

    return np.array(contexts), np.array(target)
```

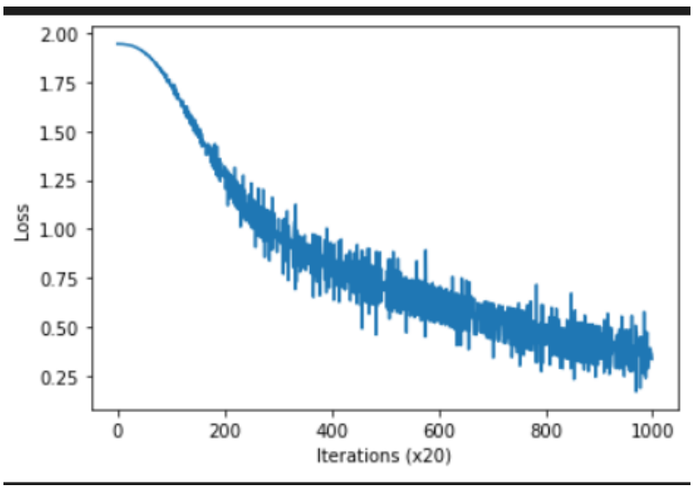
```
contexts, target = create_contexts_target(corpus, window_size=1)

print(contexts)
print(target)

'''
[[0 2]
 [1 3]
 [2 4]
 [3 1]
 [4 5]
 [1 6]]

[1 2 3 4 1 5]
'''

def convert_one_hot(corpus, vocab_size):

    N = corpus.shape[0]
    if corpus.ndim == 1:
        one_hot = np.zeros((N, vocab_size), dtype=np.int32)
        for idx, word_id in enumerate(corpus):
            one_hot[idx, word_id] = 1

    elif corpus.ndim == 2:
        C = corpus.shape[1]
        one_hot = np.zeros((N, C, vocab_size), dtype=np.int32)
        for idx_0, word_ids in enumerate(corpus):
            for idx_1, word_id in enumerate(word_ids):
                one_hot[idx_0, idx_1, word_id] = 1

    return one_hot

text = 'You say goodby and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)

contexts, target = create_contexts_target(corpus, window_size=1)

vocab_size = len(word_to_id)

target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)

print(target)
print(contexts)

'''
[[0 1 0 0 0 0 0]
 [0 0 1 0 0 0 0]
 [0 0 0 1 0 0 0]
 [0 0 0 0 1 0 0]
 [0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0]]
[[[1 0 0 0 0 0 0]
  [0 0 1 0 0 0 0]]

 [[0 1 0 0 0 0 0]
  [0 0 0 1 0 0 0]]

 [[0 0 1 0 0 0 0]
  [0 0 0 0 1 0 0]]

 [[0 0 0 1 0 0 0]
  [0 1 0 0 0 0 0]]

 [[0 0 0 0 1 0 0]
  [0 0 0 0 0 1 0]]

 [[0 1 0 0 0 0 0]
  [0 0 0 0 0 0 1]]]
'''
```

1. CBOW의 모델을 클래스로 구성해본다.
2. 데이터를 **학습**시키는 코드를 구현해본다.
    - 원핫 벡터로 변환시킨다.
    - CBOW 모델의 Layer weight들을 훈련을 통해 fitting시킨다고 생각할 수 있다.

```
import sys
sys.path.append('..')
import numpy as np
from common.layers import MatMul, SoftmaxWithLoss


class SimpleCBOW:
    def __init__(self, vocab_size, hidden_size):
        V, H = vocab_size, hidden_size

        # 가중치 초기화
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(H, V).astype('f')

        # 레이어 생성
        self.in_layer0 = MatMul(W_in)
        self.in_layer1 = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer = SoftmaxWithLoss()

        # 모든 가중치와 기울기를 리스트에 모은다.
        layers = [self.in_layer0, self.in_layer1, self.out_layer]
        self.params, self.grads = [], []

        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads

        # 인스턴스 변수에 단어의 분산 표현을 저장한다.
        self.word_vecs1 = W_in
        self.word_vecs2 = W_out.T


    def forward(self, contexts, target):
        h0 = self.in_layer0.forward(contexts[:, 0])
        h1 = self.in_layer1.forward(contexts[:, 1])
        h = (h0 + h1) * 0.5
        score = self.out_layer.forward(h)
        loss = self.loss_layer.forward(score, target)
        return loss


    def backward(self, dout=1):
        ds = self.loss_layer.backward(dout)
        da = self.out_layer.backward(ds)
        da *= 0.5
        self.in_layer1.backward(da)
        self.in_layer0.backward(da)
        return None


from common.trainer import Trainer
from common.optimizer import Adam


window_size = 1
hidden_size = 5
batch_size = 3
max_epoch = 1000

text = 'You say goodbye and I say hello.'
corpus, word_to_id, id_to_word = preprocess(text)

vocab_size = len(word_to_id)

# cbow 학습 데이터셋 생성
contexts, target = create_contexts_target(corpus, window_size)

# Input에 맞는 one-hot 표현 변환
target = convert_one_hot(target, vocab_size)
contexts = convert_one_hot(contexts, vocab_size)

# 모델 초기화
model = SimpleCBOW(vocab_size, hidden_size)
optimizer = Adam()
trainer = Trainer(model, optimizer)

# 학습
trainer.fit(contexts, target, max_epoch, batch_size)

trainer.plot()
'''
~
| 에폭 996 |  반복 1 / 2 | 시간 0[s] | 손실 0.34
| 에폭 997 |  반복 1 / 2 | 시간 0[s] | 손실 0.39
| 에폭 998 |  반복 1 / 2 | 시간 0[s] | 손실 0.36
| 에폭 999 |  반복 1 / 2 | 시간 0[s] | 손실 0.36
| 에폭 1000 |  반복 1 / 2 | 시간 0[s] | 손실 0.33
```

```
plot output은 아래와 같다
-> 훈련이 진행됨에 따라 Loss가 줄어듬을 볼 수 있다.
'''

# Word Embedding 살펴보기
word_vecs1 = model.word_vecs1

for word_id, word in id_to_word.items():
    print(word, word_vecs1[word_id])

'''
you [ 0.9810374  0.9276981 -1.0256612 -0.7274513 -1.7498773]
say [-1.1594661  -1.1794566   1.1436654   0.09712093 -1.1455401 ]
goodbye [ 1.0445026   1.0369474  -0.98637486 -1.1395257   0.3769891 ]
and [-1.0439636  -1.0600599   1.0445067  -1.9761575  -0.55348223]
i [ 1.0516133   1.0377426  -0.99020904 -1.1199282   0.3611609 ]
hello [ 0.9563576   0.9265291  -1.0225537  -0.71599466 -1.7433468 ]
. [-0.93183553 -0.9308546   0.918825    1.6551491  -1.4247652 ]
'''
```

Plot Output

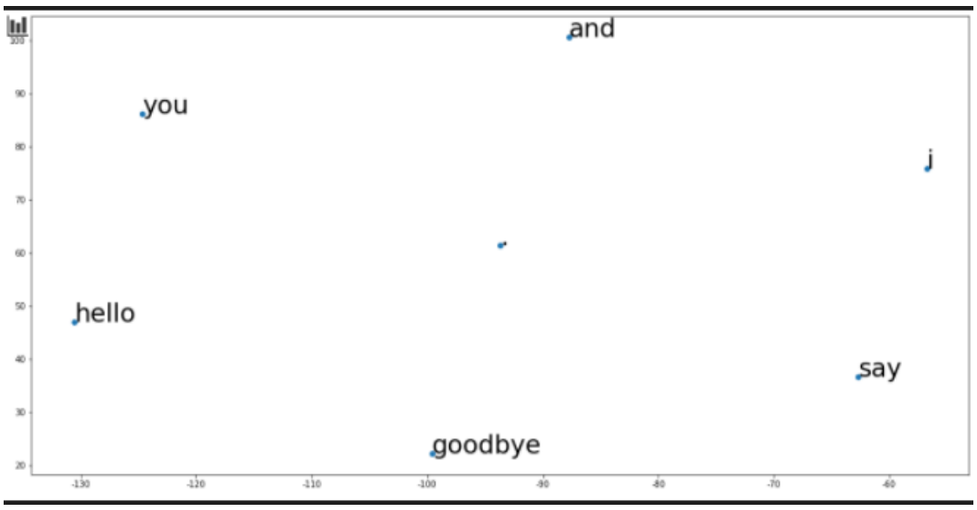


1. **t-SNE**를 이용해서 위의 CBOW 결과를 2차원 공간상으로 매핑시켜본다.
    - Plotting 과정에서 차원 축소에 대한 내용으로 추정된다. **5차원의 CBOW output**을 2차원 평면에 으깨서 출력시키게 된다.

```
%matplotlib inline
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# 그래프에서 마이너스 폰트 깨지는 문제에 대한 대처
mpl.rcParams['axes.unicode_minus'] = False
tsne = TSNE(n_components=2)

# 100개의 단어에 대해서만 시각화
X_tsne = tsne.fit_transform(word_vecs2)

vocab = list(id_to_word.values())

df = pd.DataFrame(X_tsne, index=vocab, columns=['x', 'y'])
# df.shape -> (7,2)


df.head()
```

```
'''

x y
you -124.642464 86.139900
say -62.692699  36.694683
goodbye -99.592522  22.231596
and -87.744820  100.605125
i -56.769421  75.882431

'''

fig = plt.figure()
fig.set_size_inches(20, 10)

ax = fig.add_subplot(1, 1, 1)
ax.scatter(df['x'], df['y'])

for word, pos in df.iterrows():
    ax.annotate(word, pos, fontsize=30)

plt.show()
'''
plt output은 아래와 같다
'''
```

plot output



1. 마지막으로 CBOW와 Word2Vec의 양대산맥 알고리즘인 **SkipGram**의 코드 구조를 따라서 구현해본다
    - **Skip-Gram은 CBOW의 반대로 중심 단어에서 주변 단어를 예측하도록 학습시키는 구조이다**

```
# chap03/simple_skip_gram.py
from common.layers import MatMul, SoftmaxWithLoss

class SimpleSkipGram:
    def __init__(self, vocab_size, hidden_size):
        V, H = vocab_size, hidden_size

        # 가중치 초기화
        W_in = 0.01 * np.random.randn(V, H).astype('f')
        W_out = 0.01 * np.random.randn(H, V).astype('f')

        # 레이어 생성
        self.in_layer = MatMul(W_in)
        self.out_layer = MatMul(W_out)
        self.loss_layer1 = SoftmaxWithLoss()
        self.loss_layer2 = SoftmaxWithLoss()

        # 모든 가중치와 기울기를 리스트에 모은다.
        layers = [self.in_layer, self.out_layer]
        self.params, self.grads = [], []

        for layer in layers:
            self.params += layer.params
            self.grads += layer.grads
```

```
        # 인스턴스 변수에 단어의 분산표현을 저장한다.
        self.word_vecs1 = W_in
        self.word_vecs2 = W_out.T

    def forward(self, contexts, target):
        h = self.in_layer.forward(target)
        s = self.out_layer.forward(h)
        l1 = self.loss_layer1.forward(s, contexts[:, 0])
        l2 = self.loss_layer2.forward(s, contexts[:, 1])
        loss = l1 + l2
        return loss

    def backward(self, dout=1):
        dl1 = self.loss_layer1.backward(dout)
        dl2 = self.loss_layer2.backward(dout)
        ds = dl1 + dl2
        dh = self.out_layer.backward(ds)
        self.in_layer.backward(dh)
        return None
```

결론

- 빈도수 기반의 단어 분산 표현(Chap02)는 신경망을 통한 모델링에는 적합하지가 않았다
- 추론 기반의 단어 분산 표현인 **Word2Vec**을 통해서 모든 단어를 같은 차원으로 만들면서 원핫 인코딩에 비해 차원을 줄일 수 있으며 이로 인해 신경망 훈련과 딥러닝 모델링에 매우 적합해진다.

감사합니다.