정보검색 hw_2 skip gram 구현 보고서

B515031 김인제

(a)

(i)sigmoid method구현

```
def sigmoid(x):
    """..."""

    ### YOUR CODE HERE (~1 Line)
    #numpy에 구현된 exponential 함수를 쓰면 간단하게 나타낼 수 있다.
    s = 1./(np.exp(-x)+1.)

    ### END YOUR CODE

    return s
```

(ii)naiveSoftmaxLossAndGradient method구현

```
def naiveSoftmaxLossAndGradient(
    centerWordVec,
    outsideWordIdx,
    outsideVectors,
    dataset
):
    """Naive Softmax loss & gradient function for word2vec models..."""

    ### YOUR CODE HERE (~6-8 Lines)
    #y_hat.shape = ( # of words in vocab, )
    y_hat = softmax(np.dot(outsideVectors, centerWordVec))
    #loss 값 계산
    loss = -np.log(y_hat)
    # 문제지 (b)에서 U(y_hat - y)를 구현했었다..
    # 또한 (c)에서 y_hat - y꼴이 등장하므로 계산해놓는다.
    # subtraction = y_hat - y 이다. u_o에만 1(y)값을 빼줘야 제대로 계산이 된다. (w = 0 인 case)
    # 그 이외에는  y_hat 이기 때문에 indexing으로 해당 element만 -y(-1)을 취한다.
    subtraction = y_hat
    subtraction[outsideWordIdx] -= 1
    #  aJ / av_c는 (U.T(y_hat -y))
    gradCenterVec = outsideVectors.T.dot(subtraction)
    #   aJ / aU의 계산, 실제로 (c)문제를 이용하면, matrix의 형태는 diagonal 값은 (y_hat -y)v_c.T 을 사용,
    #   그 외의 값은 y_hat v_c.T 를 사용한다.
    #   matrix의 형태는 (# of words in vocab, word vec length) 이다. np.dot을 잘 이용해 계산한다.
    gradOutsideVecs = subtraction[:, np.newaxis].dot(np.array([centerWordVec]))

    ### Please use the provided softmax function (imported earlier in this file)
    ### This numerically stable implementation helps you avoid issues pertaining
    ### to integer overflow.

    ### END YOUR CODE
```

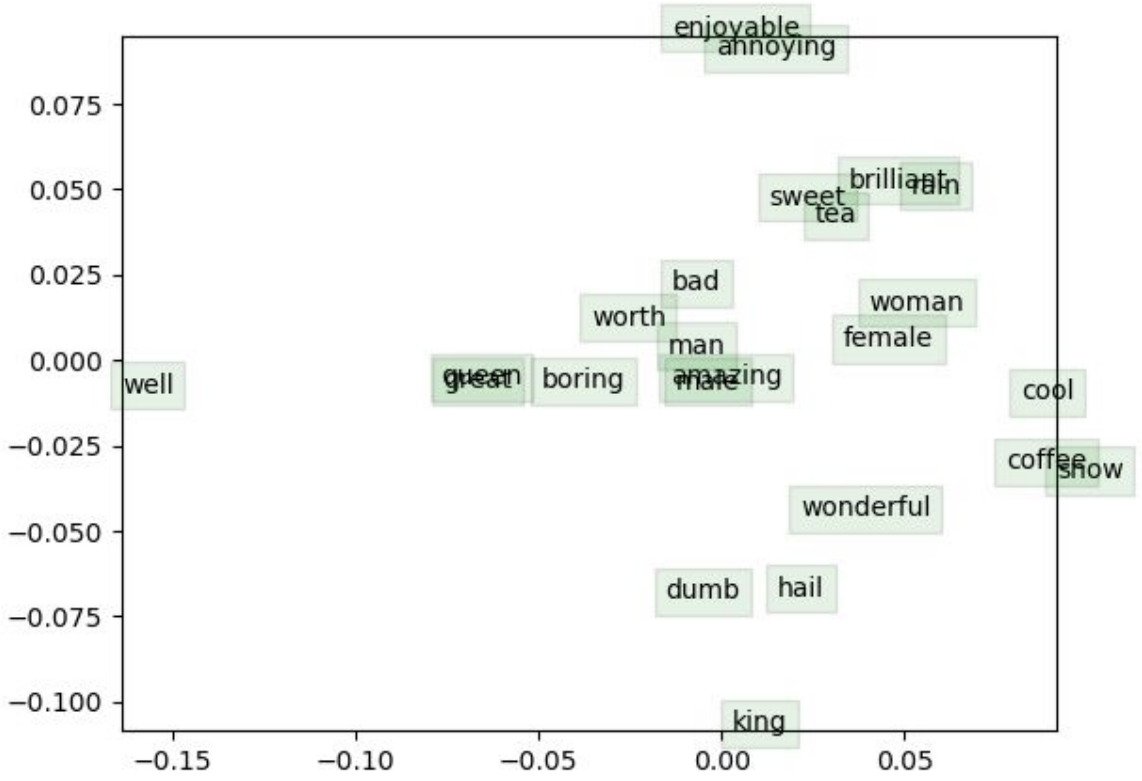
(iii)skip gram 구현

```
def skipgram(currentCenterWord, windowSize, outsideWords, word2Ind,
             centerWordVectors, outsideVectors, dataset,
             word2vecLossAndGradient=naiveSoftmaxLossAndGradient):
    """Skip-gram model in word2vec..."""

    loss = 0.0
    gradCenterVecs = np.zeros(centerWordVectors.shape)
    gradOutsideVectors = np.zeros(outsideVectors.shape)

    ### YOUR CODE HERE (~8 Lines)
    #   word2Ind dictionary로부터 현재 Vc의 index값을 얻는다
    centerWordIdx = word2Ind[currentCenterWord]
    # index 값을 이용해 word vector집합에서 현재 centerword를 얻는다.
    centerWordVec = centerWordVectors[centerWordIdx]
    # outsideWords에서 outside word(string)추출, word2Ind를 이용해 해당 word에 대한 index저장
    outsideWordIndices = [word2Ind[i] for i in outsideWords]

    # J_skipgram 의 v_c, U 에 대한 편미분을 구하고, loss를 구한다.
    # 반복문은 -m <= j <= m, j != 0 이므로 2m번 반복한다.
    for outsideWordIdx in outsideWordIndices:
        one_loss, one_gradCenter, one_gradOutside = \
            word2vecLossAndGradient(centerWordVec, outsideWordIdx, outsideVectors, dataset)

        loss += one_loss
        gradCenterVecs[centerWordIdx] += one_gradCenter
        gradOutsideVectors += one_gradOutside

    ### END YOUR CODE

    return loss, gradCenterVecs, gradOutsideVectors
```

(b) sgd method 구현

```
"""Stochastic Gradient Descent..."""                                    ⚠ 21  ✓ 16

# Anneal learning rate every several iterations
# -> parameter들이 이미 저장되어있다면, 매 iteration마다 annealing한다.
# 예를 들면 20000번 반복시에는 step size가 절반이 된다.
ANNEAL_EVERY = 20000

if useSaved:...
else:
    start_iter = 0

x = x0

if not postprocessing:
    postprocessing = lambda x: x

exploss = None

for iter in range(start_iter + 1, iterations + 1):
    # You might want to print the progress every few iterations.

    loss = None
    ### YOUR CODE HERE (~2 lines)
    # run.py 의 sgd 를 참조하여 f(x)를 찾는다.
    # f = word2vec_sgd_wrapper(skipgram, tokens, vec, dataset, C,naiveSoftmaxLossAndGradient) 이다.
    # f는 batch size = 50으로 하여 skip gram 에 대해 naiveSoftmaxLossAndGradient를 수행한다.
    # grad는 각 인자에 대한 편미분 행렬을 더한 gradient matrix.
    loss, grad = f(x)
    #step size만큼 parameter들을 update한다.
    x -= step * grad
    ...
    x = postprocessing(x)
    if iter % PRINT_EVERY == 0:...
    # 5000번 iteration 마다 parameter저장
    if iter % SAVE_PARAMS_EVERY == 0 and useSaved:
        save_params(iter, x)
    # 20000번 학습마 step size를 절반으로 줄인다.
    if iter % ANNEAL_EVERY == 0:
        step *= 0.5

return x
```

(3) 결과 plotting



man과 male, female과 woman은 의미적으로 비슷하여 묶인거라고 해석이 가능해보인다.
enjoyable과 annoying은 서로 반대되는 말이지만 syntax적으로 비슷한걸로도 볼 수 있다.
snow와 cool은 서로 연관성이 있다고 해석된 것이 흥미롭다.

- 실행문 중 일부 발췌

```
iter 39940 : [27.89096797 28.04979834 27.93828121 ... 28.05559316 28.05681879
 28.05809084]
iter 39950 : [27.77546677 27.93155118 27.82207157 ... 27.93724623 27.93844513
 27.93970205]
iter 39960 : [27.68367047 27.83895613 27.72994642 ... 27.84460498 27.84577011
 27.84702378]
iter 39970 : [27.7860415  27.93876876 27.83147309 ... 27.94429289 27.94544003
 27.94667193]
iter 39980 : [27.76403038 27.9149386  27.80904055 ... 27.92043537 27.92156633
 27.92278598]
iter 39990 : [27.81349145 27.96154234 27.85755297 ... 27.96692764 27.96803861
 27.96923584]
iter 40000 : [27.73949848 27.88719259 27.78337867 ... 27.89255059 27.89365366
 27.89484312]
sanity check: cost at convergence should be around or below 10
training took 10357 seconds
```