MennatAllah Hany Hassan

Assignment #1

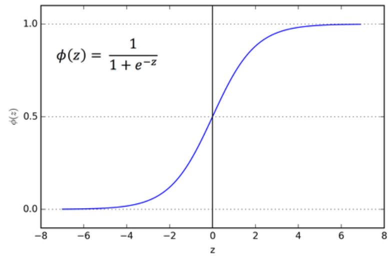
Natural Language Processing with Deep Learning

March 2019

Implementing word2vec

(a) First, implement the sigmoid function in word2vec.py to apply the sigmoid function to an input vector.

```python
def sigmoid(x):
    """
    Compute the sigmoid function for the input here.
    Arguments:
    x -- A scalar or numpy array.
    Return:
    s -- sigmoid(x)
    """

    ### YOUR CODE HERE
    s = 1/(1+np.exp(-x))
    ### END YOUR CODE

    return s
```



In the same file, fill in the implementation for the softmax and negative sampling loss and gradient functions.

<u>Softmax:</u>

$$P(O = o \mid C = c) = \frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w \in \text{Vocab}} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \quad = \quad \hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{\theta}).$$

where theta = $\boldsymbol{u}_o^\top \boldsymbol{v}_c$

$$\boldsymbol{J}_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log P(O = o | C = c). \;=\; -\log(\hat{\boldsymbol{y}})$$

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = U(\hat{\boldsymbol{y}} - \boldsymbol{y}).$$

$$\frac{\partial J}{\partial \boldsymbol{U}} = \boldsymbol{v}_c(\hat{\boldsymbol{y}} - \boldsymbol{y})^\top$$

```python
### YOUR CODE HERE

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

#Calculating loss function

theta = centerWordVec.dot(outsideVectors.T)      # 1xD * [NxD].T = 1xD * DxN => 1xN
yhat = softmax (theta)                           # 1xN
loss = - np.log(yhat[outsideWordIdx])            # 1

#calculating (y_hat - y) where y is one hot label vector with 1 only at o index
# will refer to it new_y_hat
yhat[outsideWordIdx] -=1

# (dJ / dv_c) = U * new_y_hat
gradCenterVec =  yhat.dot(outsideVectors)

#(dJ / dU) = V_c * [new_y_hat].T
gradOutsideVecs = yhat[:, np.newaxis].dot(centerWordVec[np.newaxis, :])

### END YOUR CODE

return loss, gradCenterVec, gradOutsideVecs
```

Negative Sampling:

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

$$\frac{\partial J}{\partial \boldsymbol{v}_c} = (\sigma(\boldsymbol{u}_o^\top v_c) - 1)\boldsymbol{u}_o - \sum_{k=1}^{K}(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1)\boldsymbol{u}_k$$

$$\frac{\partial J}{\partial \boldsymbol{u}_o} = (\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1)\boldsymbol{v}_c$$

$$\frac{\partial J}{\partial \boldsymbol{u}_k} = -(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1)\boldsymbol{v}_c, \quad \text{for all } k = 1, 2, \ldots, K$$

```python
### YOUR CODE HERE

theta = centerWordVec.dot(outsideVectors[indices].T)
k_0 = -np.log(sigmoid(theta[0]))
k_all = -np.sum(np.log(sigmoid(-theta[1:])))
loss = k_0 + k_all

# (dJ / dv_c)
gradCenterVec_0 = (sigmoid(theta[0]) - 1) * outsideVectors[outsideWordIdx]
gradCenterVec_k = np.sum((sigmoid(-theta[1:]) - 1)[:, np.newaxis] * outsideVectors[indices[1:]], axis=0)
gradCenterVec = gradCenterVec_0 - gradCenterVec_k


##(dJ / dU)
gradOutsideVecs = np.zeros(outsideVectors.shape)
gradOutsideVecs[outsideWordIdx] = (sigmoid(theta[0]) - 1) * centerWordVec
grad_k = -((sigmoid(-theta[1:]) - 1)[:, np.newaxis] * centerWordVec)
np.add.at(gradOutsideVecs, indices[1:], grad_k)

### END YOUR CODE

return loss, gradCenterVec, gradOutsideVecs
```

Then, fill in the implementation of the loss and gradient functions for the skip-gram model.

$$\boldsymbol{J}_{\text{skip-gram}}(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}) = \sum_{\substack{-m \le j \le m \\ j \ne 0}} \boldsymbol{J}(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U})$$

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m \ldots c+m})}{\partial \boldsymbol{U}} = \sum_{-m \le j \le m, j \ne 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial \boldsymbol{U}},$$

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m \ldots c+m})}{\partial \boldsymbol{v}_c} = \sum_{-m \le j \le m, j \ne 0} \frac{\partial F(\boldsymbol{w}_{c+j}, \boldsymbol{v}_c)}{\partial \boldsymbol{v}_c},$$

$$\frac{\partial J_{\text{skip-gram}}(\text{word}_{c-m \ldots c+m})}{\partial \boldsymbol{v}_j} = \boldsymbol{0}, \text{for all } j \ne c.$$

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

### YOUR CODE HERE

center_word_idx = word2Ind[currentCenterWord]
for outsideWord in outsideWords:
    o = word2Ind[outsideWord]
    temp_loss, grad_vc, grad_U = word2vecLossAndGradient(centerWordVectors[center_word_idx], o, outsideVectors, dataset)
    loss += temp_loss
    gradCenterVecs[center_word_idx] += grad_vc
    gradOutsideVectors += grad_U

### END YOUR CODE

return loss, gradCenterVecs, gradOutsideVectors
```

When you are done, test your implementation by running python word2vec.py.

```
(base) mennatallah@mennatallah-Inspiron-3542:~/Desktop/a2$ python word2vec.py
==== Gradient check for skip-gram with naiveSoftmaxLossAndGradient ====
Gradient check passed!
==== Gradient check for skip-gram with negSamplingLossAndGradient ====
Gradient check passed!
```

Skip-Gram with naiveSoftmaxLossAndGradient

```
=== Results ===
Skip-Gram with naiveSoftmaxLossAndGradient
Your Result:
Loss: 11.16610900153398
Gradient wrt Center Vectors (dJ/dV):
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.        ]
 [-1.26947339 -1.36873189  2.45158957]
 [ 0.          0.          0.        ]
 [ 0.          0.          0.        ]]
Gradient wrt Outside Vectors (dJ/dU):
 [[-0.41045956  0.18834851  1.43272264]
 [ 0.38202831 -0.17530219 -1.33348241]
 [ 0.07009355 -0.03216399 -0.24466386]
 [ 0.09472154 -0.04346509 -0.33062865]
 [-0.13638384  0.06258276  0.47605228]]

Expected Result: Value should approximate these:
Loss: 11.16610900153398
Gradient wrt Center Vectors (dJ/dV):
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.        ]
 [-1.26947339 -1.36873189  2.45158957]
 [ 0.          0.          0.        ]
 [ 0.          0.          0.        ]]
Gradient wrt Outside Vectors (dJ/dU):
 [[-0.41045956  0.18834851  1.43272264]
 [ 0.38202831 -0.17530219 -1.33348241]
 [ 0.07009355 -0.03216399 -0.24466386]
 [ 0.09472154 -0.04346509 -0.33062865]
 [-0.13638384  0.06258276  0.47605228]]
```

Skip-Gram with negSamplingLossAndGradient

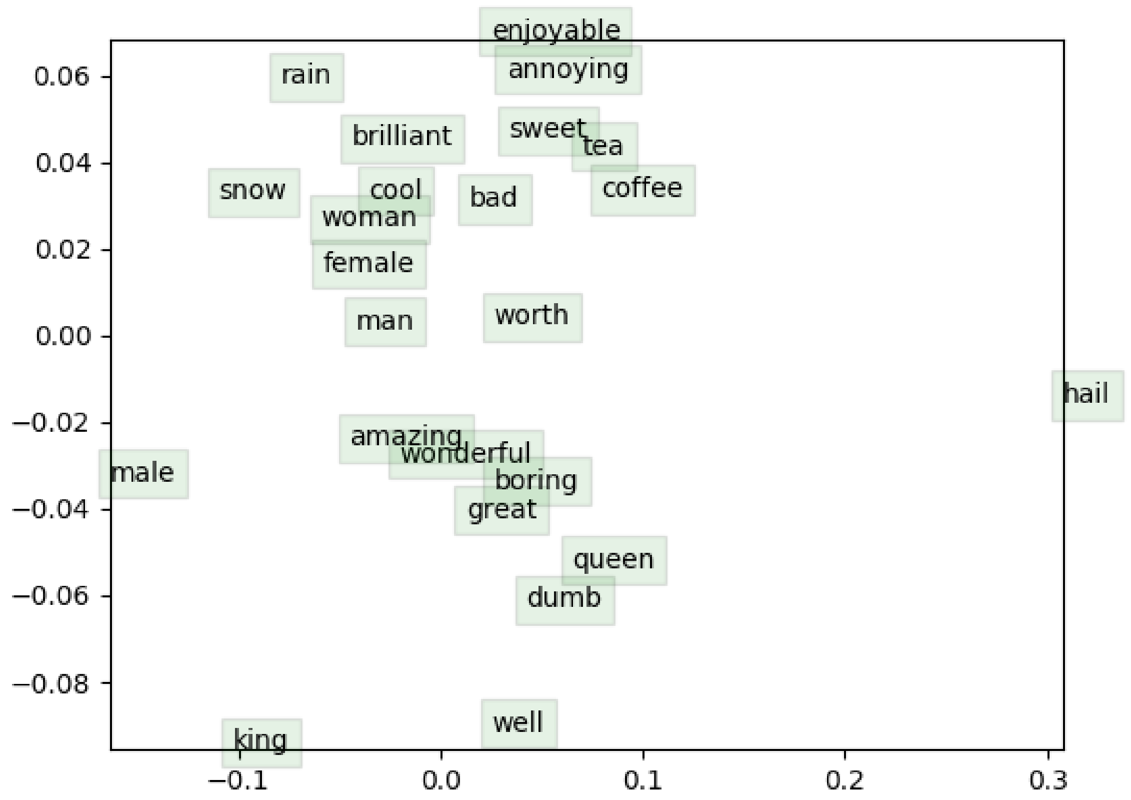
```
Skip-Gram with negSamplingLossAndGradient
Your Result:
Loss: 16.15119285363322
Gradient wrt Center Vectors (dJ/dV):
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.        ]
 [-4.54650789 -1.85942252  0.76397441]
 [ 0.          0.          0.        ]
 [ 0.          0.          0.        ]]
 Gradient wrt Outside Vectors (dJ/dU):
 [[-0.69148188  0.31730185  2.41364029]
 [-0.22716495  0.10423969  0.79292674]
 [-0.45528438  0.20891737  1.58918512]
 [-0.31602611  0.14501561  1.10309954]
 [-0.80620296  0.36994417  2.81407799]]

Expected Result: Value should approximate these:
Loss: 16.15119285363322
Gradient wrt Center Vectors (dJ/dV):
 [[ 0.          0.          0.        ]
 [ 0.          0.          0.        ]
 [-4.54650789 -1.85942252  0.76397441]
 [ 0.          0.          0.        ]
 [ 0.          0.          0.        ]]
 Gradient wrt Outside Vectors (dJ/dU):
 [[-0.69148188  0.31730185  2.41364029]
 [-0.22716495  0.10423969  0.79292674]
 [-0.45528438  0.20891737  1.58918512]
 [-0.31602611  0.14501561  1.10309954]
 [-0.80620296  0.36994417  2.81407799]]
```

(b) Complete the implementation for your SGD optimizer in sgd.py. Test your implementation by running python sgd.py.

```python
    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

        loss, gradient = f(x)
        x -= step * gradient


        ### END YOUR CODE

        x = postprocessing(x)
```

```
(base) mennatallah@mennatallah-Inspiron-3542:~/Desktop/a2$ python sgd.pyRunning sanity checks...
iter 100: 0.004578
iter 200: 0.004353
iter 300: 0.004136
iter 400: 0.003929
iter 500: 0.003733
iter 600: 0.003546
iter 700: 0.003369
iter 800: 0.003200
iter 900: 0.003040
iter 1000: 0.002888
test 1 result: 8.414836786079764e-10
iter 100: 0.000000
iter 200: 0.000000
iter 300: 0.000000
iter 400: 0.000000
iter 500: 0.000000
iter 600: 0.000000
iter 700: 0.000000
iter 800: 0.000000
iter 900: 0.000000
iter 1000: 0.000000
test 2 result: 0.0
iter 100: 0.041205
iter 200: 0.039181
iter 300: 0.037222
iter 400: 0.035361
iter 500: 0.033593
iter 600: 0.031913
iter 700: 0.030318
iter 800: 0.028802
iter 900: 0.027362
iter 1000: 0.025994
test 3 result: -2.524451035823933e-09
----------------------------------------
ALL TESTS PASSED
----------------------------------------
(base) mennatallah@mennatallah-Inspiron-3542:~/Desktop/a2$
```

(c) Show time! Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the Stanford Sentiment Treebank (SST) dataset to train word vectors, and later apply them to a simple sentiment analysis task. You will need to fetch the datasets first. To do this, run sh get datasets.sh. There is no additional code to write for this part; just run python run.py. Note: The training process may take a long time depending on the efficiency of your implementation (an efficient implementation takes approximately an hour). Plan accordingly! After 40,000 iterations, the script will finish and a visualization for your word vectors will appear. It will also be saved as word vectors.png in your project directory. Include the plot in your homework write up. Briefly explain in at most three sentences what you see in the plot.



- Similar words in context clusters with each other like [amazing, wonderful, boring, great].
- The virtual line connecting [king and queen] is approximately parallel to the line connecting [male and female].
- Words like [enjoyable, annoying, brilliant,cool and sweet] cluster with each other, and at the same time [sweet] was close to [Tea, Coffee] and [cool] was close to [rain and snow].