PROGRAMMING ASSIGNMENT 2 NLP:

Done by:

Jayetri Bardhan

UFID: 01794985

Link of github repository containing the code:

https://github.com/jayetri/nlp-coding-assignment1.git
Cloning with ssh: git@github.com:jayetri/nlp-coding-assignment1.git
This contains all the files used during coding.

Setup:

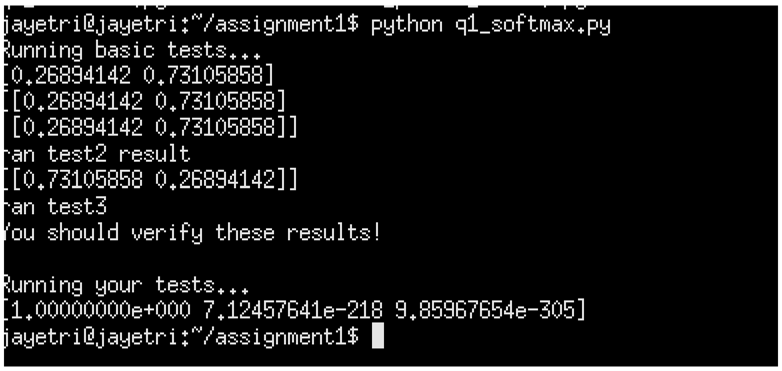
The setup was done as per the instructions. I made use of Microsoft Azure to make of GPU. I did it without using virtual environment. After installing everything from requirement.txt, I was facing some errors while running the code q3_run.py. So, I upgraded numpy along with other packages. Also, while running q3_run.py, I got an error stating that I had a pip six version less than 1.1.o which was failing to run the program and was not compatible with matlabplot.py. So, I upgraded it.

Screenshot of the output screen:

3.1.Softmax: It is an exponential function which can be represented as:

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Screenshot of the output screen while running q1_softmax.py is as follows:



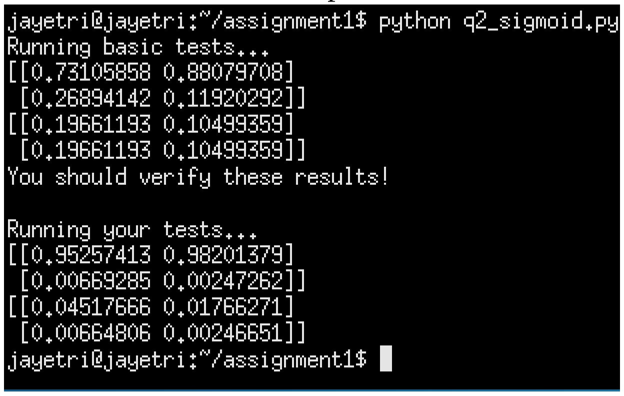
```
jayetri@jayetri:~/assignment1$ python q1_softmax.py
Running basic tests...
[0.26894142 0.73105858]
[[0.26894142 0.73105858]
 [0.26894142 0.73105858]]
ran test2 result
[[0.73105858 0.26894142]]
ran test3
You should verify these results!

Running your tests...
[1.00000000e+000 7.12457641e-218 9.85967654e-305]
jayetri@jayetri:~/assignment1$ 
```

The code was written so as to satisfy both 1D and 2D vectors.

3.2Neural basics

(1)Gradient of sigmoid function:

This is coded in q2_sigmoid.py
The screenshot of the output is as follows:



```
jayetri@jayetri:~/assignment1$ python q2_sigmoid.py
Running basic tests...
[[0.73105858 0.88079708]
 [0.26894142 0.11920292]]
[[0.19661193 0.10499359]
 [0.19661193 0.10499359]]
You should verify these results!

Running your tests...
[[0.95257413 0.98201379]
 [0.00669285 0.00247262]]
[[0.04517666 0.01766271]
 [0.00664806 0.00246651]]
jayetri@jayetri:~/assignment1$
```

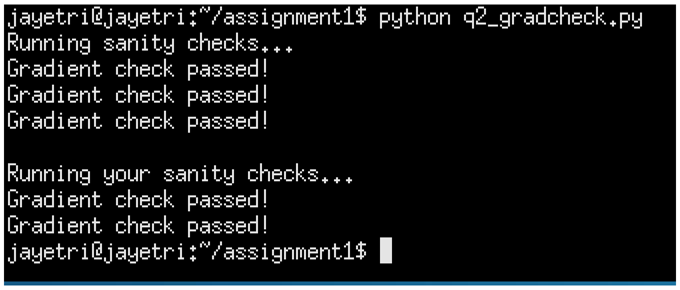
I made use of the following two equations:

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (1)$$

$$\sigma'(x) = \sigma(x)(1-\sigma(x)). \quad (2)$$

(2) Gradient check

This is code in the file: q2 gradcheck.py. It check if the gradient check is passed or not.
The screenshot of the output screen is as follows:

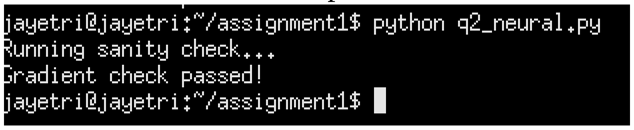


```
jayetri@jayetri:~/assignment1$ python q2_gradcheck.py
Running sanity checks...
Gradient check passed!
Gradient check passed!
Gradient check passed!

Running your sanity checks...
Gradient check passed!
Gradient check passed!
jayetri@jayetri:~/assignment1$
```

It makes use of the equation:

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$$

And sets few conditions such as:
Epsilon is set as 1e-4, and the threshold as 1e-5 and accordingly decides if the gradient checking has passed.
(c) Simple Neural Network
It has three layers with a hidden layer between the input and output layers.
The screenshot of the output is as follows:



```
jayetri@jayetri:~/assignment1$ python q2_neural.py
Running sanity check...
Gradient check passed!
jayetri@jayetri:~/assignment1$
```

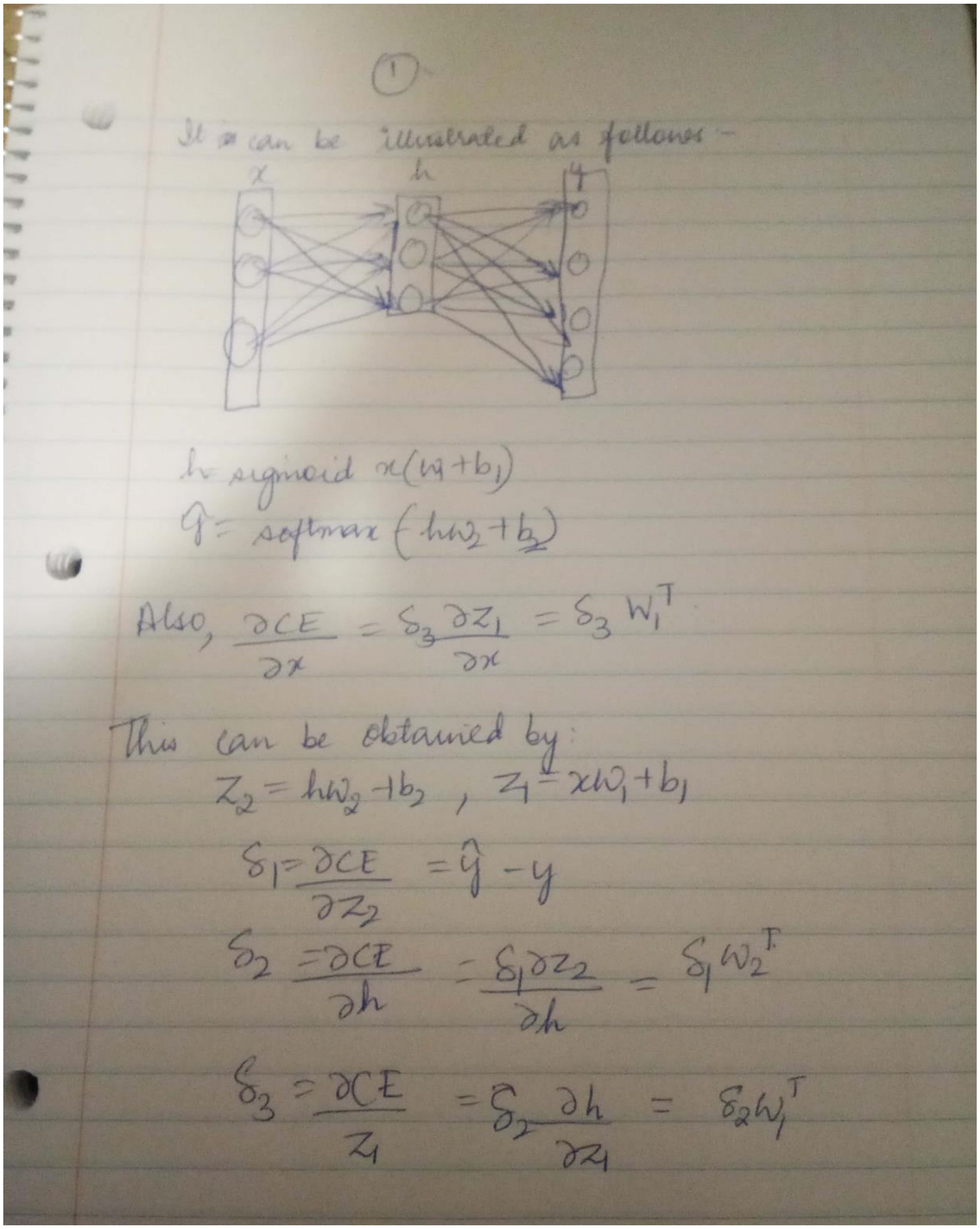
It is coded in q2_neural.py
This file includes the concept of forward propogation and back propogation.

Forward propogation involves three steps and has been coded accordingly:

1. Neuron Activation.
2. Neuron Transfer.
3. Forward Propagation

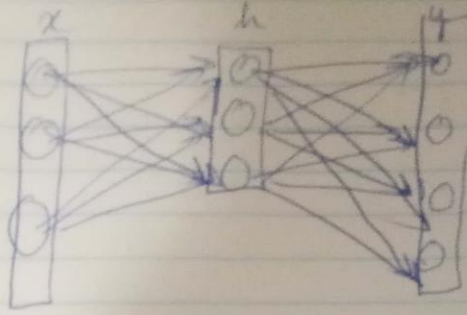
Backward propogation:
In this error is calculated which is the difference between the output obtained and expected output. The errors are backpropogated to obtain the modified weights in order to obtain an efficient network.

It includes the following equations.:



(1)

It ~~is~~ can be illustrated as follows -



$h = \text{sigmoid}\ x(w_1+b_1)$

$\hat{y} = \text{softmax}\left(hw_2+b_2\right)$

Also, $\frac{\partial CE}{\partial x} = \delta_3 \frac{\partial z_1}{\partial x} = \delta_3 W_1^T$

This can be obtained by:

$z_2 = hw_2+b_2 \quad , \quad z_1 = xw_1+b_1$

$\delta_1 = \frac{\partial CE}{\partial z_2} = \hat{y} - y$

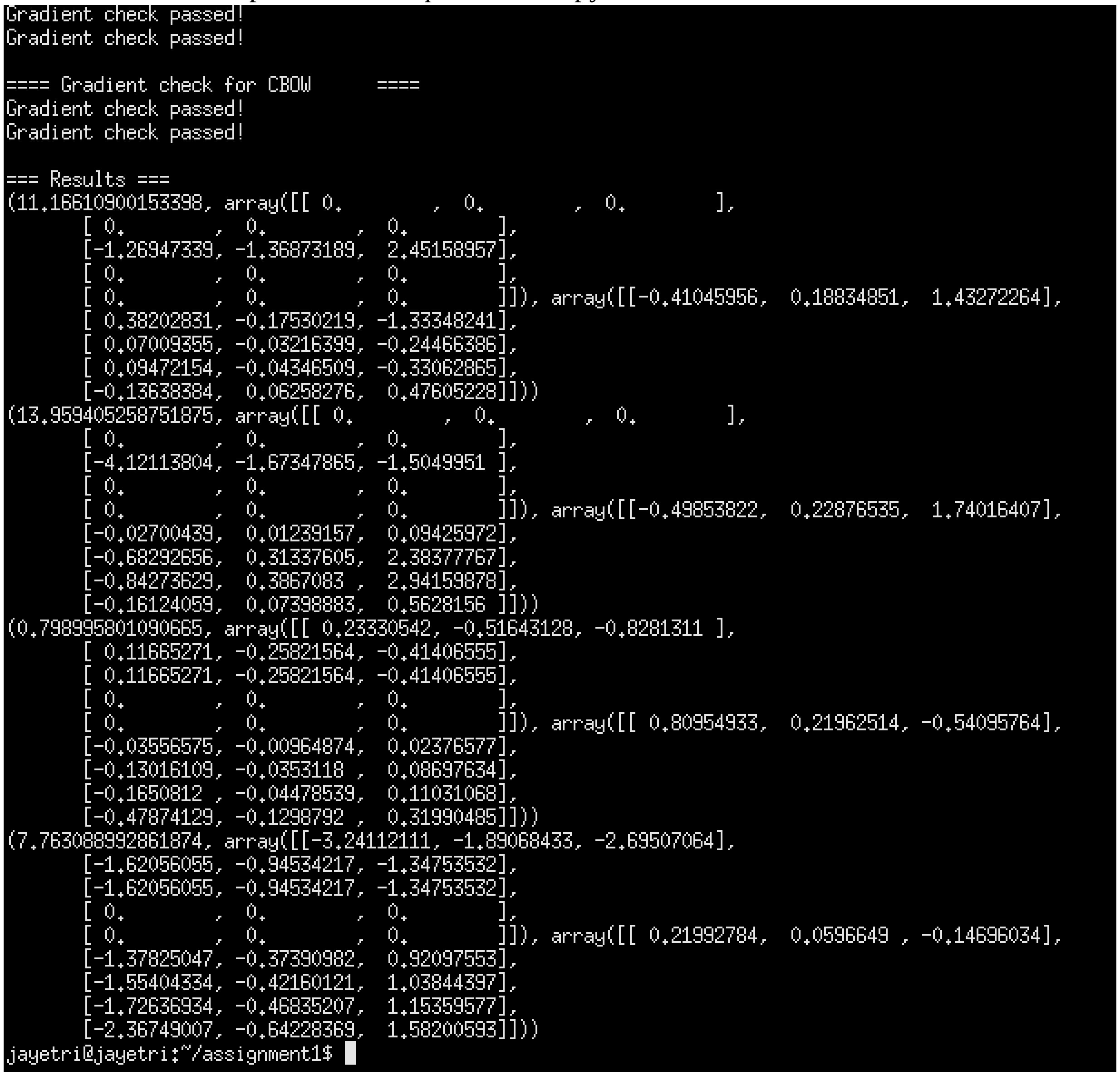
$\delta_2 = \frac{\partial CE}{\partial h} = \frac{\delta_1 \partial z_2}{\partial h} = \delta_1 w_2^T$

$\delta_3 = \frac{\partial CE}{z_1} = \delta_2 \frac{\partial h}{\partial z_1} = \delta_2 w_1^T$

3.3 Word2Vec

(1),(2) and (3)

Screenshot of the output screen for q3 word2vec.py.



```
Gradient check passed!
Gradient check passed!

==== Gradient check for CBOW      ====
Gradient check passed!
Gradient check passed!

=== Results ===
(11.166109001533398, array([[ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ],
       [-1.26947339, -1.36873189,  2.45158957],
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ]]), array([[-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]]))
(13.959405258751875, array([[ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ],
       [-4.12113804, -1.67347865, -1.5049951 ],
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ]]), array([[-0.49853822,  0.22876535,  1.74016407],
       [-0.02700439,  0.01239157,  0.09425972],
       [-0.68292656,  0.31337605,  2.38377767],
       [-0.84273629,  0.3867083 ,  2.94159878],
       [-0.16124059,  0.07398883,  0.5628156 ]]))
(0.798995801090665, array([[ 0.23330542, -0.51643128, -0.8281311 ],
       [ 0.11665271, -0.25821564, -0.41406555],
       [ 0.11665271, -0.25821564, -0.41406555],
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ]]), array([[ 0.80954933,  0.21962514, -0.54095764],
       [-0.03556575, -0.00964874,  0.02376577],
       [-0.13016109, -0.0353118 ,  0.08697634],
       [-0.1650812 , -0.04478539,  0.11031068],
       [-0.47874129, -0.1298792 ,  0.31990485]]))
(7.763088992861874, array([[-3.24112111, -1.89068433, -2.69507064],
       [-1.62056055, -0.94534217, -1.34753532],
       [-1.62056055, -0.94534217, -1.34753532],
       [ 0.        ,  0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        ]]), array([[ 0.21992784,  0.0596649 , -0.14696034],
       [-1.37825047, -0.37390982,  0.92097553],
       [-1.55404334, -0.42160121,  1.03844397],
       [-1.72636934, -0.46835207,  1.15359577],
       [-2.36749007, -0.64228369,  1.58200593]]))
jayetri@jayetri:~/assignment1$ 
```

Here I have taken care of softcost and gradient, negative sampling loss and implementing skipgram model.

The following equations are made used:

$$\hat{y}_i = Pr(w_i|\hat{r}, u_{w_{1\ldots|v|}}) = \frac{exp(u_{w_i}^T \hat{r})}{\sum_{j=1}^{|v|} exp(u_{w_j}^T \hat{r})}$$

$$J_{neg-sample}(w_i, \hat{r}, u_{w_i}, u_{w1\ldots K}) = -log(\sigma(u_{w_i}^T \hat{r})) - \sum_{k=1}^{K} log(\sigma(-u_{w_k}^T \hat{r}))$$

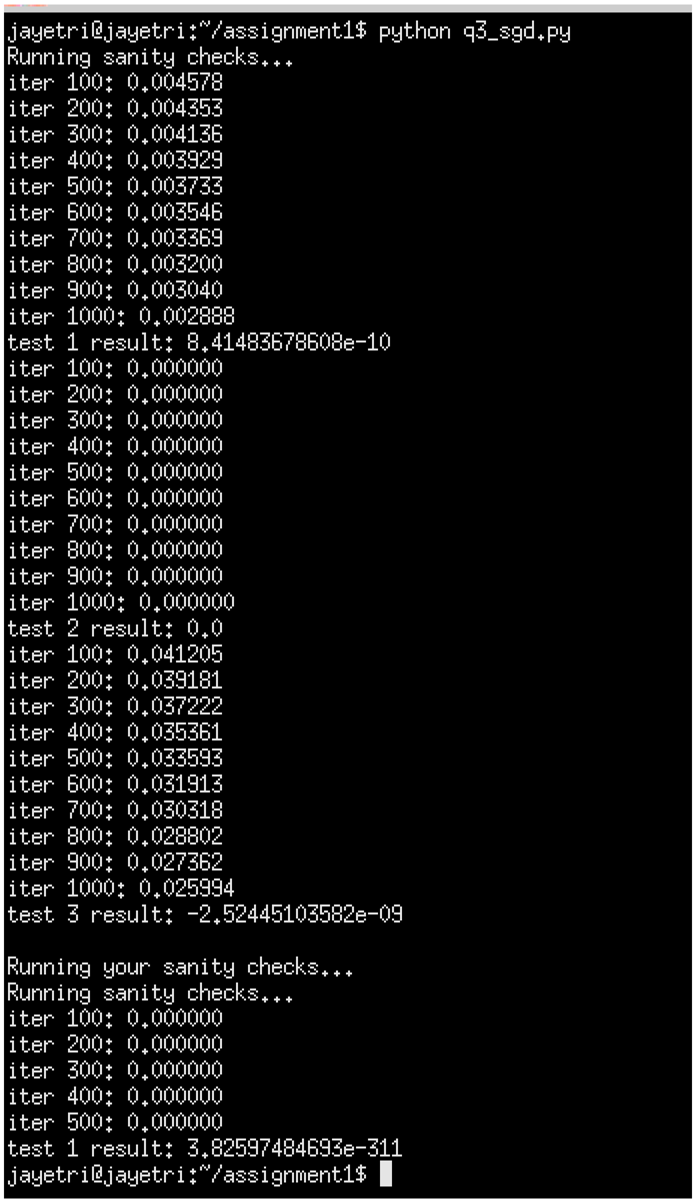
$$J_{skip-gram}(word_{i-C...i+C}) = \sum_{-C \leq j \leq C, j \neq 0} F(w_{i+j}, v_{w_i})$$

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

(4)SGD optimizer

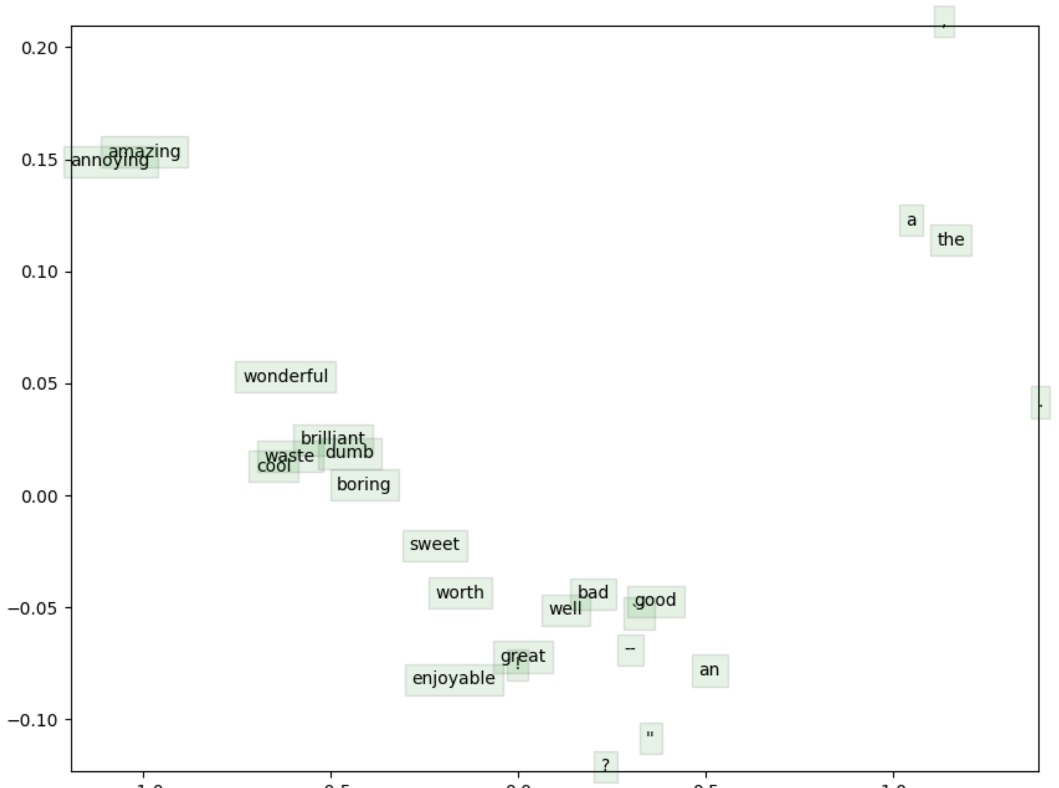
It is implemented in q3_sgd.py

The screenshot of the output is as follows:



```
jayetri@jayetri:~/assignment1$ python q3_sgd.py
Running 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: 8.41483678608e-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: -2.52445103582e-09

Running your sanity checks...
Running sanity checks...
iter 100: 0.000000
iter 200: 0.000000
iter 300: 0.000000
iter 400: 0.000000
iter 500: 0.000000
test 1 result: 3.82597484693e-311
jayetri@jayetri:~/assignment1$
```

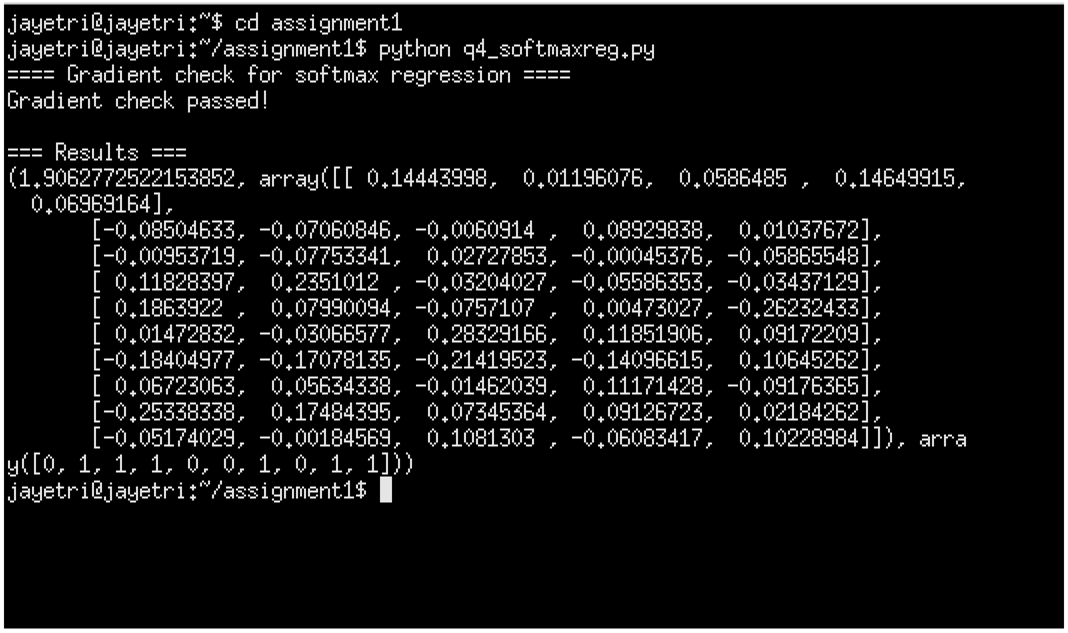
(5) When we run the code q3_run.py, we obtain the following graph:



Conclusion:
So, overall it makes use of all the functions and implements wordtovec function on the Standford dataset. In the skipgram model, it predicts the context from the source whereas in CBOW it does the reverse. The graph shows how words have been converted to vectors and how the source word is predicted from the context vector.

The code ran for 40,000 iterations and took almost 2 hrs to run on Microsoft Azure’s GPU.

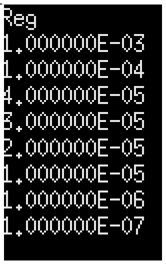
3.4 Sentiment Analysis
(1)Firstly, q4 softmaxreg.py is implemented.
The output of the screenshot is as follows:



```
jayetri@jayetri:~$ cd assignment1
jayetri@jayetri:~/assignment1$ python q4_softmaxreg.py
==== Gradient check for softmax regression ====
Gradient check passed!

=== Results ===
(1.9062772522153852, array([[ 0.14443998,  0.01196076,  0.0586485 ,  0.14649915,
  0.06969164],
       [-0.08504633, -0.07060846, -0.0060914 ,  0.08929838,  0.01037672],
       [-0.00953719, -0.07753341,  0.02727853, -0.00045376, -0.05865548],
       [ 0.11828397,  0.2351012 , -0.03204027, -0.05586353, -0.03437129],
       [ 0.1863922 ,  0.07990094, -0.0757107 ,  0.00473027, -0.26232433],
       [ 0.01472832, -0.03066577,  0.28329166,  0.11851906,  0.09172209],
       [-0.18404977, -0.17078135, -0.21419523, -0.14096615,  0.10645262],
       [ 0.06723063,  0.05634338, -0.01462039,  0.11171428, -0.09176365],
       [-0.25338338,  0.17484395,  0.07345364,  0.09126723,  0.02184262],
       [-0.05174029, -0.00184569,  0.1081303 , -0.06083417,  0.10228984]]), arra
y([0, 1, 1, 1, 0, 0, 1, 0, 1, 1]))
jayetri@jayetri:~/assignment1$
```

(2)Then q4 sentiment.py was run on the Standford dataset. This again took 3 hrs to run on Microsoft Azure's GPU.

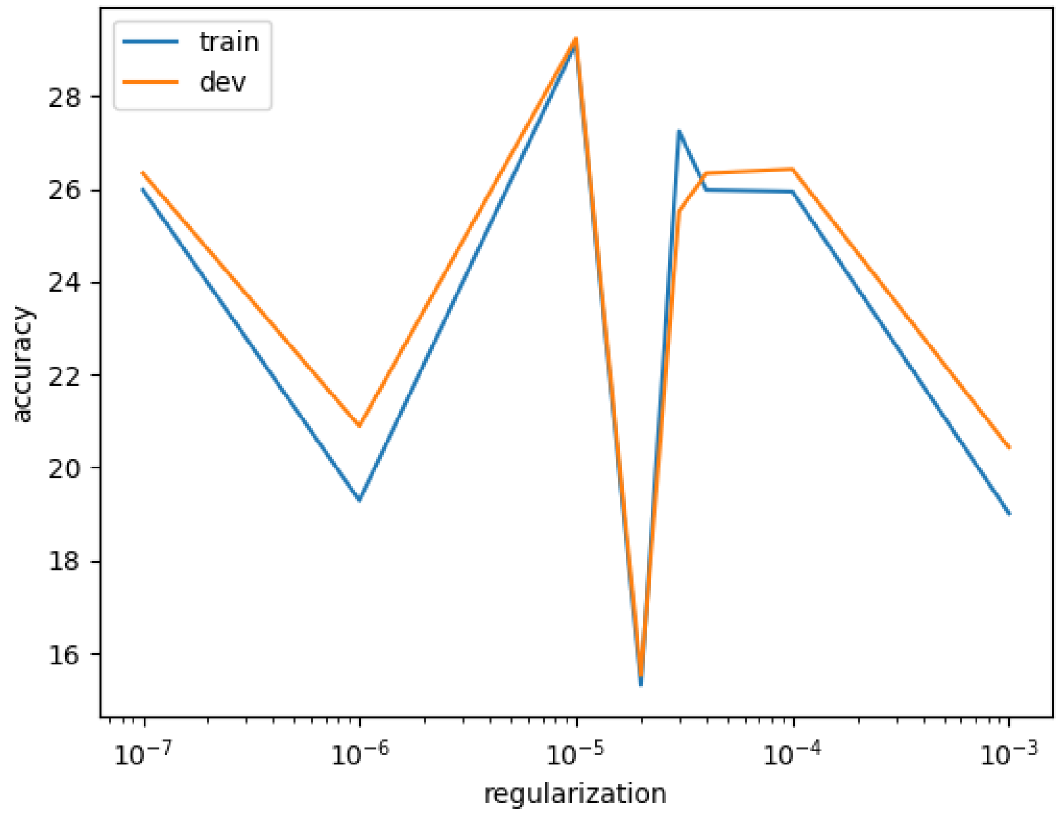
Then a plot was obtained for classification accuracy on the train and dev set with respect to the regularization value, using a logarithmic scale on the x-axis.
I have plotted for the following regularization values:



Reg
1.000000E-03
1.000000E-04
4.000000E-05
3.000000E-05
2.000000E-05
1.000000E-05
1.000000E-06
1.000000E-07

Here we are determining the sentiment of the sentence, ranging from 0 to 4, i.e. very negative to positive. This code is implemented following the concept of wordtovec and implementing the concept of sentiment in it.

We obtain the following plot:



As we observe from the graph ,maximum accuracy is obtained at 1* e-5 . It obtained the maximum training accuracy of 29.412% and 30.24% development accuracy.It is lowest for 1* e-3 and

1* e-6.

References:

1. Textbook
2. AndrewNg Coursera tutorial
3. https://stats.stackexchange.com/questions/184220/numeric-gradient-checking-how-close-is-close-enough/188724#188724
4. https://en.wikipedia.org/wiki/Softmax_function