Assignment 2: Word Vectors

Junzhe Wang - September 19, 2019

1. Basics

    a. Prove softmax is invariant to constant offset in the input.

$$((soft\max(x+c))_i = \frac{\exp(x_i+c)}{\sum_{j=1}^{\dim(x)}\exp(x_j+c)} = \frac{\exp(x_i)*\exp(c)}{\sum_{j=1}^{\dim(x)}(\exp(x_j)*\exp(c))}$$
$$= \frac{\exp(x_i)*\exp(c)}{\exp(c)*\sum_{j=1}^{\dim(x)}\exp(x_j)} = \frac{\exp(x_i)}{\sum_{j=1}^{\dim(x)}\exp(x_j)} = soft\max(x)_j$$

    b. See softmax.py

    c.

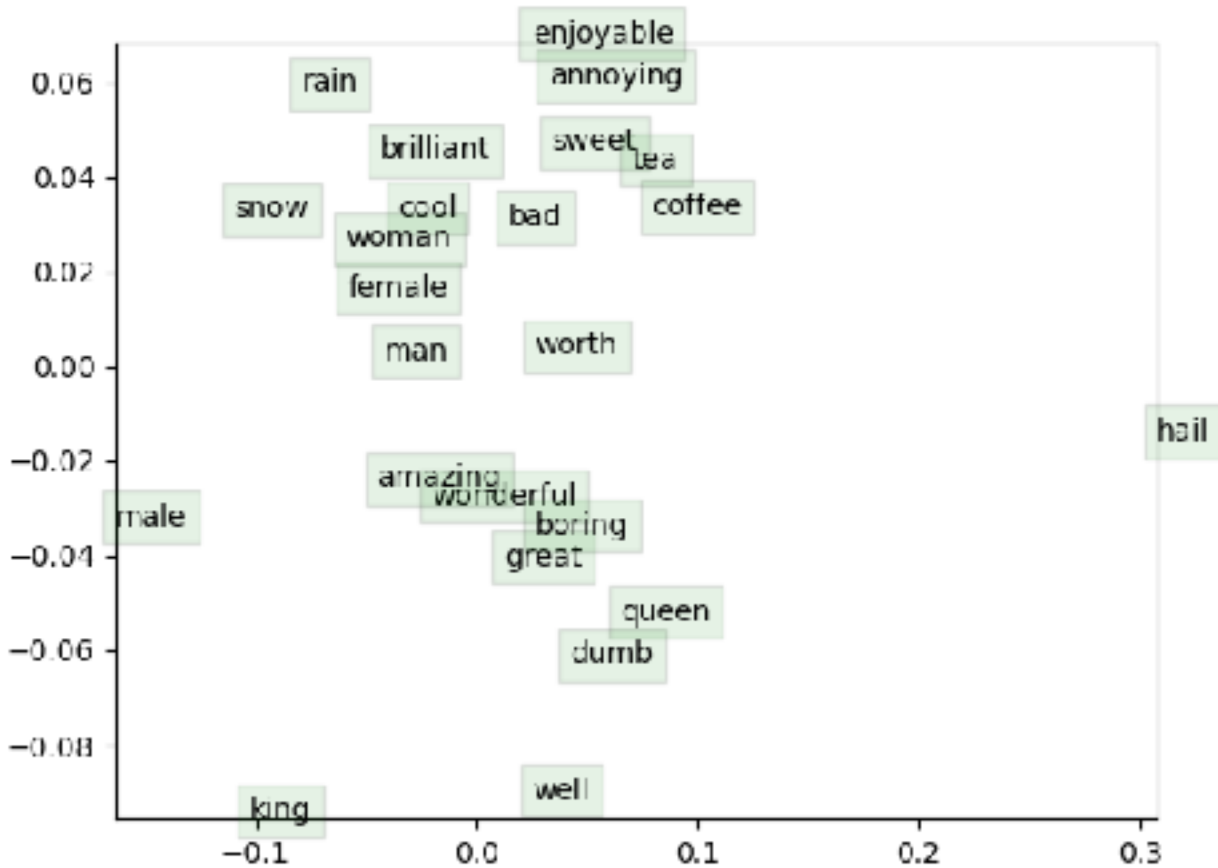
Sigmoid :	$\sigma(x) = \frac{1}{1+e^{-x}}$
Derivative of sigmoid:	$\frac{d}{dx}(1+e^{-x})^{-1} = (-1)*(1+e^{-x})^{-2}*\frac{d}{dx}(1+e^{-x}) = \frac{-1}{(1+e^{-x})^2}\frac{d}{dx}(e^{-x})$ $= \frac{-1}{(1+e^{-x})^2}*(-1)*e^{-x} = \frac{e^{-x}}{(1+e^{-x})^2}$
$\sigma(x)(1-\sigma(x))$	$\frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) = \frac{e^{-x}}{(1+e^{-x})^2}$

2. Word2vec

a	$J = \sum_{i=1}^{w} y_i \log(\frac{\exp(u_i^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)}) = \sum_{i=1}^{w} y_i [u_i^T v_c - \log(\sum_{w=1}^{W} \exp(u_w^T v_c))$
	since y is one-hot vector, there is only one non-zero item in the summation, so the loss could be written as :
	$J = -y(u_o^T v_c - \log(\sum_{w=1}^{W} \exp(u_w^T v_c))$
	$\frac{\partial J}{\partial v_c} = -[u_o - \frac{\partial(\log(\sum_{w=1}^{W} \exp(u_w^T v_c))}{\partial v_b}] = -[u_o - \frac{\sum_{w=1}^{W} \exp(u_w^T v_c) u_w}{\sum_{w=1}^{W} \exp(u_w^T v_c)}]$ $= \sum_{w=1}^{W} \frac{\exp(u_w^T v_c) u_w}{\sum_{w=1}^{W} \exp(u_w^T v_c)} - u_o = \sum_{w=1}^{W} \hat{y}_w u_w - u_o = (\hat{y} - 1) u_o = U(\hat{y} - y)$
b	$\frac{\partial J}{\partial u_w} = -[v_c - \frac{\partial(\log(\sum_{w=1}^{W} \exp(u_w^T v_c))}{\partial u_w}] = -[v_c - \frac{\sum_{w=1}^{W} \exp(u_w^T v_c) v_c}{\sum_{w=1}^{W} \exp(u_w^T v_c)}] =$ $\sum_{w=1}^{W} \frac{\exp(u_w^T v_c) v_c}{\sum_{w=1}^{W} \exp(u_w^T v_c)} - v_c = \sum_{w=1}^{W} \hat{y}_w v_c - v_c = (\hat{y} - 1) v_c = (\hat{y} - y) v_c$
c	Since $o \in/ \{1, ..., K\}$, the second part is 0 $\frac{\partial J}{\partial u_o} = -\frac{\sigma(u_o^T v_c)(1 - \sigma(u_o^T v_c)) v_c}{\sigma(u_o^T v_c)}$ $= -(1 - \sigma(u_o^T v_c)) v_c$
d	$\frac{\partial J_{skip-gram}(word_{c-m,...,c+m})}{\partial U} = \sum_{-m \le j \le m, j \ne 0} \frac{\partial F(w_i, \hat{v})}{\partial U}$
	$\frac{\partial J_{skip-gram}(word_{c-m,...,c+m})}{\partial v_c} = \sum_{-m \le j \le m, j \ne 0} \frac{\partial F(w_i, \hat{v})}{\partial v_c}$
	$\frac{\partial J_{skip-gram}(word_{c-m,...,c+m})}{\partial v_j} = 0$, for all $j \ne c$

g. In the run.py file, I tested the KNN function by computing the 5 nearest neighbor of the first word(**'great'**) among the visualizeWords, which are: [”great”, "cool", "brilliant", "wonderful", "well", "amazing", "worth", "sweet", "enjoyable", "boring", "bad", “dumb","annoying", "female", "male", "queen", "king", "man", "woman", "rain", "snow", "hail", "coffee", “tea”], and the results are ['well', 'annoying', 'amazing', 'bad', ‘queen’]

```
(py37) MacBook-Pro-3:a2 joey$ python run.py
sanity check: cost at convergence should be around or below 10
training took 0 seconds
the nearest words to great are ['well', 'annoying', 'amazing', 'bad', 'queen']
```



enjoyable
annoying
rain
brilliant
sweet
tea
snow
cool
bad
coffee
woman
female
man
worth
hail
amazing
wonderful
male
boring
great
queen
dumb
well
king
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.1
0.0
0.1
0.2
0.3