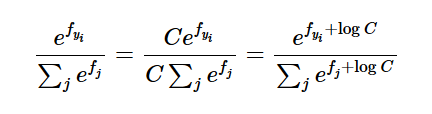
*efyi*∑*jefj*=*CefyiC*∑*jefj*=*efyi*+log*C*∑*jefj*+log*C*

**CS 224d: Assignment #1**

**1 Softmax (10 points)**

1 (a) Softmax(x)=softmax(x+c)

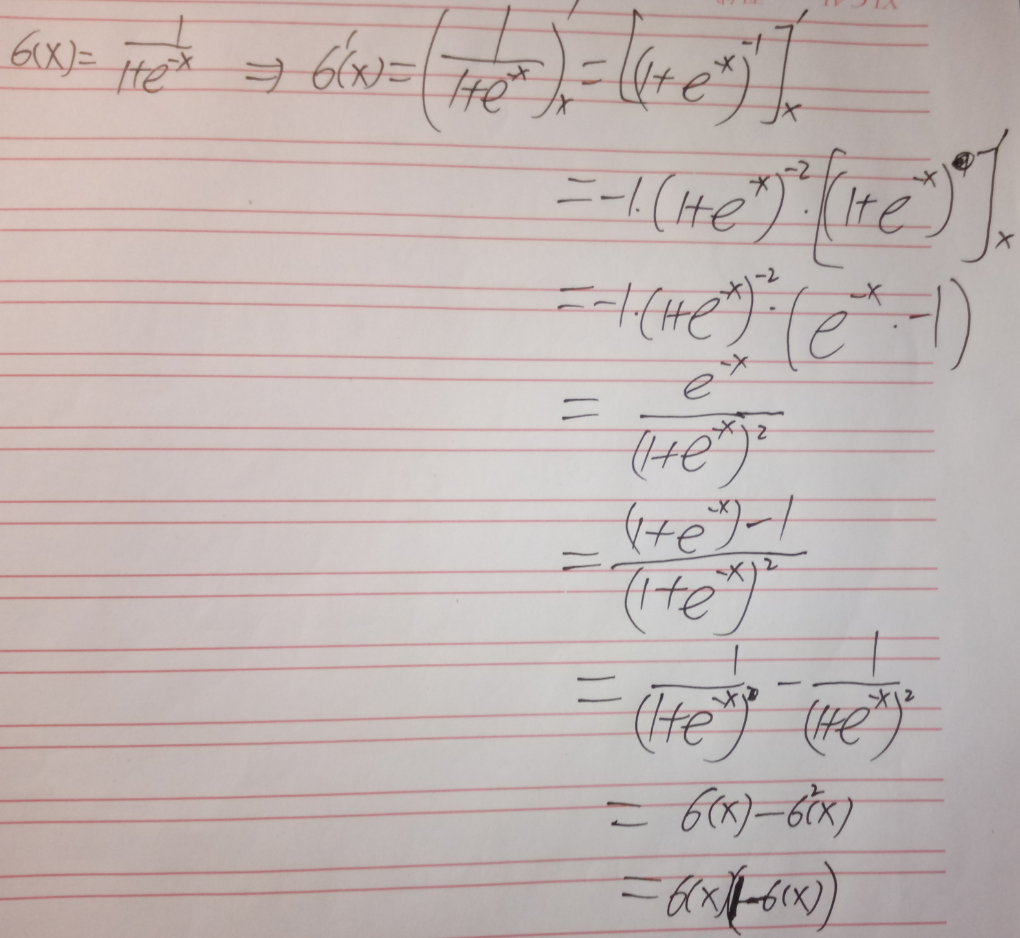


Let C=exp(c), then we can see that softmax(x)=softmax(x+c)

1 (b) check code at q1\_softmax.py

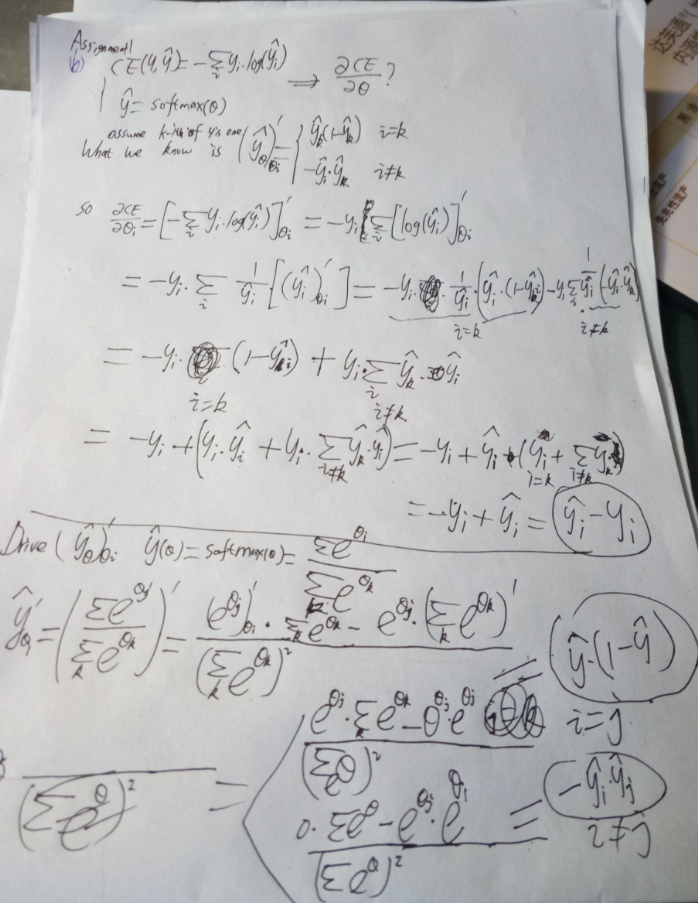
Given an input matrix of N rows and d columns, compute the softmax prediction for each row. Write your implementation in q1 softmax.py. You may test by executing python q1 softmax.py

2 Neural Network Basics (30 points)

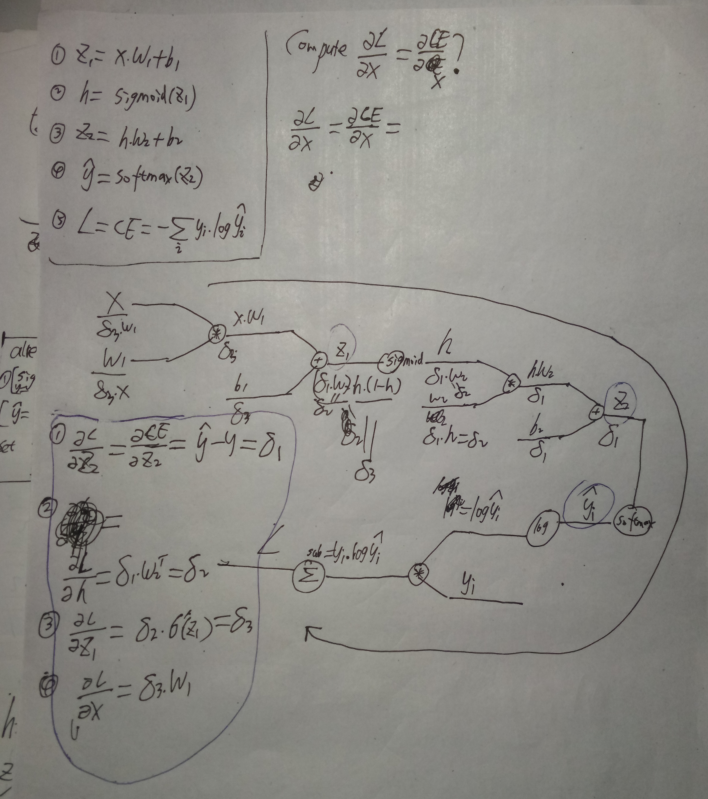
(a) (3 points) Derive the gradients of the sigmoid function and show that it can be rewritten as a function of the function value (i.e. in some expression where only σ(x), but not x, is present). Assume that the input x is a scalar for this question. 

**2 Neural Network Basics (30 points)**

（a） Derive the gradient with regard to the inputs of a softmax function when cross entropy loss is used for evaluation,

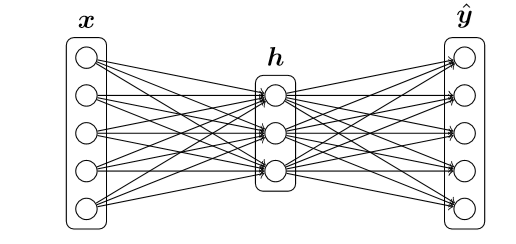


(3):



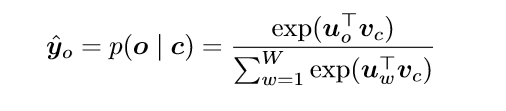
(d) (2 points) How many parameters are there in this neural network, assuming the input is Dx-dimensional, the output is Dy-dimensional, and there are H hidden units?

(Dx+1)\*H+(H+1)\*Dy



**3 word2vec (40 points + 5 bonus)**

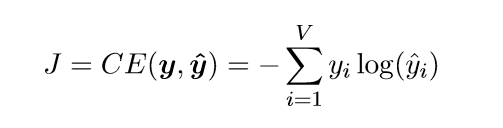
(a) Assume you are given a predicted word vector vc corresponding to the center word c for skipgram, and word prediction is made with the softmax function found in word2vec models



where w denotes the w-th word and uw (w = 1,...,W) are the “output” word vectors for all words in the vocabulary. Assume cross entropy cost is applied to this prediction and word o is the expected word (the o-th element of the one-hot label vector is one), derive the gradients with respect to vc.

Hint: It will be helpful to use notation from question 2. For instance, letting ˆ y be the vector of softmax predictions for every word, y as the expected word vector, and the loss function

where U = [u1,u2,··· ,uW] is the matrix of all the output vectors. Make sure you state the orientation of your vectors and matrices.



**Solution:**

y’=softmax(theta);

CE(y,y’)=-sum(y\*long(y’) --->dCE/dtheta=y’-y

(this is what we already know from previous homework)

**Let** Score\_i=Ui\*Vc, so Score\_o=Uo\*Vc, Score\_w=Uw\*Vc, and dScore\_o/dVc=U

So dCE/dVc=(dCE/dScore\_o)\*(dScore\_o/dVc)=(y’-y).dot(U)

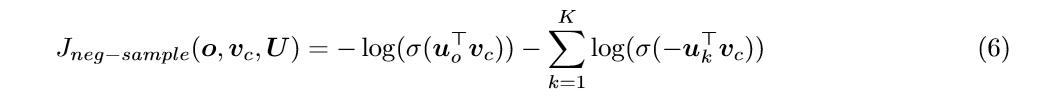
(b) (3 points) As in the previous part, derive gradients for the “output” word vectors Uw’s (including uo).

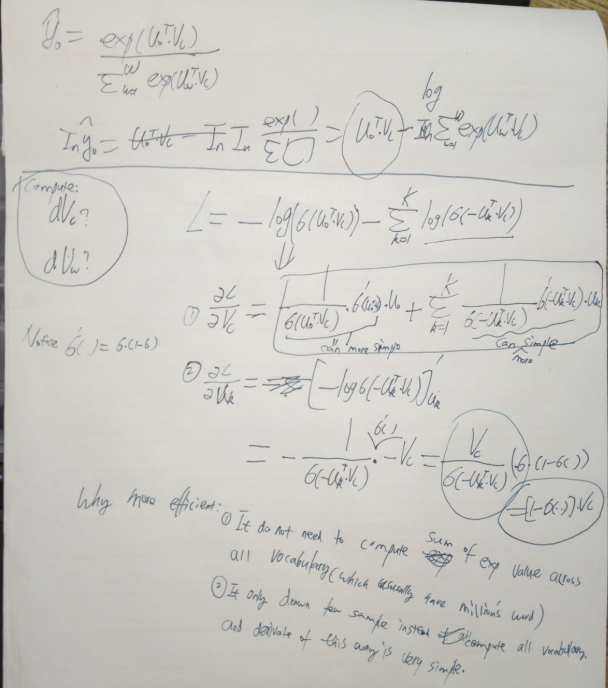
Solution:

dScore\_w/dUw=Vc

Similiarly, dCE/dUw=(dCE/dScore\_o)\*(dScore\_o/dUw)=(y’-y).dot(Vc)

(c) (6 points) Repeat part (a) and (b) assuming we are using the negative sampling loss for the predicted vector vc, and the expected output word is o. Assume that K negative samples (words) are drawn, and they are 1,··· ,K, respectively for simplicity of notation (o /∈{1,...,K}). Again, for a given word, o, denote its output vector as uo. The negative sampling loss function in this case is

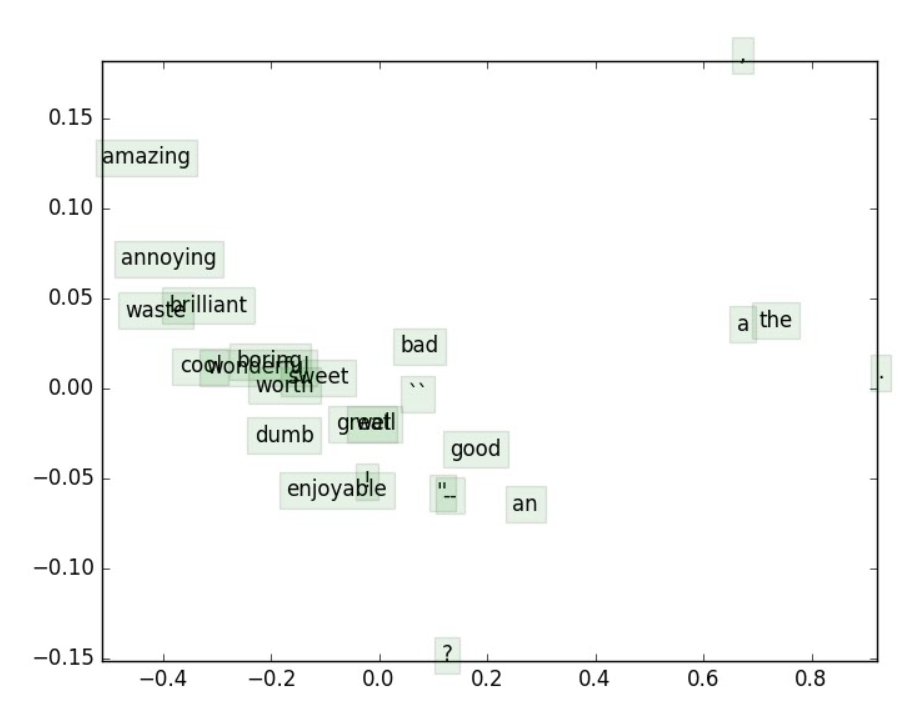




(g) Run code. And Include the plot in your homework write up. Brieﬂy explain in at most three sentences what you see in the plot.

Answer: similar word appear in nearby sapce. For example, enjoyable, great,good are nearby. And bad,boring is also near by. Words like ‘a’,’the’ are also nearby.

What did i see:



**4 Sentiment Analysis (20 points)**

(b) introduce regularization is to make our model more generalize. And it will also make weights more spread, so that encourage most weights have some kind of impact, not those few weights have big impact.

(To avoid overﬁtting to the training examples and generalizing poorly to unseen examples.)

(c): (4 points) Fill in the hyperparameter selection code in q4 sentiment.py to search for the “optimal” regularization parameter. What value did you select? Report your train, dev, and test accuracies. Justify your hyperparameter search methodology in at most one sentence. Note: you should be able to attain at least 30% accuracy on dev.

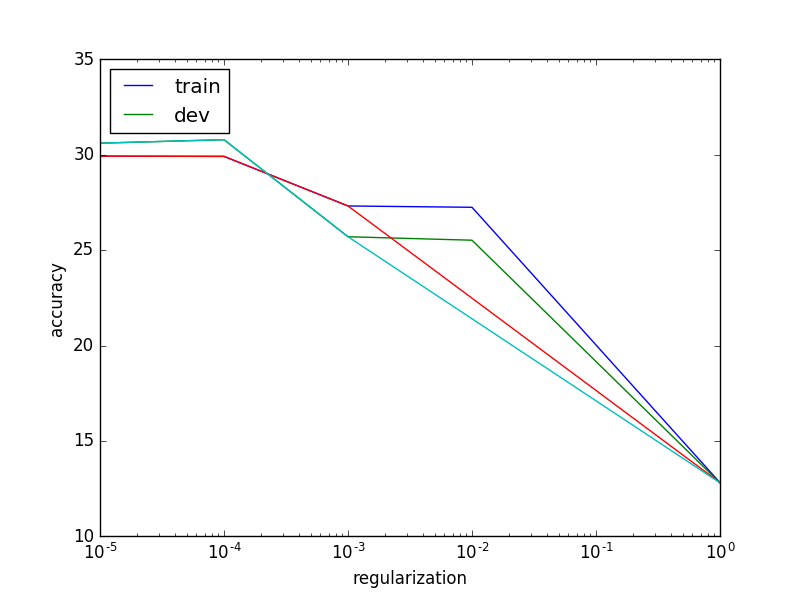
**ANSWER**:

I select reg=0.000100. Accuracy of train, dev is around 30%. Test accuracy (%): 28.099548

I search hyperparameter by it’s order of magnitudes. From very small one like 0.0001 to very big one like 100. Every time multipy 10 times. So at least i will able to find the right order of magnitude.

**What did i see:**

1. As regularization changes, both training and dev accuracy will be changed. So that the accuacy and also the behave of our model will be changed.
2. There is a point that training accuracy and dev accuracy are crossover.

****

Training for reg=0.010000

iter 10000: 1.572766

Train accuracy (%): 27.247191

Dev accuracy (%): 25.522252

Training for reg=0.100000

iter 10000: 1.584912

Train accuracy (%): 27.247191

Dev accuracy (%): 25.522252

Training for reg=1.000000

Training for reg=1

INF

Training for reg=10

INF

**Training for reg=0.000100**

**iter 10000: 1.568712**

**Train accuracy (%): 29.915730**

**Dev accuracy (%): 30.790191**

**Training for reg=0.001000**

Training for reg=0.001000

iter 10000: 1.570770

Train accuracy (%): 27.317416

Dev accuracy (%): 25.703906