CS 224N:Assignment #1

1. Softmax

a)

Proof. For all dimensions $1 \le i \le \dim(\boldsymbol{x})$

$$(\text{softmax}(\boldsymbol{x}+c))_i = \frac{\exp(x_i + c)}{\sum_{j=1}^{\dim(\boldsymbol{x})} \exp(x_j + c)} = \frac{\exp(c)\exp(x_i)}{\exp(c)\sum_{j=1}^{\dim(\boldsymbol{x})} \exp(x_j)} = \frac{\exp(x_i)}{\sum_{j=1}^{\dim(\boldsymbol{x})} \exp(x_j)} = (\text{softmax}(\boldsymbol{x}))_i \,.$$

b)

```
if len(x.shape) > 1:
    x -= np.max(x, axis=1, keepdims=True)
    x = np.exp(x) / np.sum(np.exp(x), axis=1, keepdims=True)
else:
    x -= np.max(x)
    x = np.exp(x) / np.sum(np.exp(x))
```

2. Neural Network Basics

a)

$$\sigma'(x) = \left(\frac{1}{1+e^{-x}}\right)'$$

$$= -\left(\frac{1}{1+e^{-x}}\right)^2 \cdot (e^{-x})'$$

$$= \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right)$$

$$= \sigma(x) \cdot (1-\sigma(x))$$

b)

$$\frac{\partial CE(y,\hat{y})}{\partial \theta_i} = \frac{\partial - \sum_i y_i \log(\hat{y}_i)}{\partial \theta_i}$$

$$= \frac{\partial - \log(\hat{y}_k)}{\partial \theta_i}$$

$$= -\frac{\partial \log(s(\theta_k))}{\partial \theta_i}$$

$$= \begin{cases} \hat{y}_i - 1 & i = k \\ \hat{y}_i & i \neq k \end{cases}$$

c)

Solution: Denote $\boldsymbol{z}_2 = \boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2$, and $\boldsymbol{z}_1 = \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1$, then

$$\boldsymbol{\delta}_1 = \frac{\partial CE}{\partial \boldsymbol{z}_2} = \hat{\boldsymbol{y}} - \boldsymbol{y}$$

$$\boldsymbol{\delta}_2 = \frac{\partial CE}{\partial \boldsymbol{h}} = \boldsymbol{\delta}_1 \frac{\partial \boldsymbol{z}_2}{\partial \boldsymbol{h}} = \boldsymbol{\delta}_1 \boldsymbol{W}_2^\top$$

$$\boldsymbol{\delta}_3 = \frac{\partial CE}{\boldsymbol{z}_1} = \boldsymbol{\delta}_2 \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{z}_1} = \boldsymbol{\delta}_2 \circ \sigma'(\boldsymbol{z}_1)$$

$$\frac{\partial CE}{\partial \boldsymbol{x}} = \boldsymbol{\delta}_3 \frac{\partial \boldsymbol{z}_1}{\partial \boldsymbol{x}} = \boldsymbol{\delta}_3 \boldsymbol{W}_1^\top$$

注意：公式中所用的向量均为列向量，但在程序具体实现时为了加快程序运行速度都用的是行向量。

d) $(D_x + 1) \cdot H + (H + 1) \cdot D_y.$

e)

```python
def sigmoid(x):
    s = 1 / (1 + np.exp(-x))
    return s
```

```python
def sigmoid_grad(s):
    ds = s * (1 - s)
    return ds
```

f)

```
old_val = x[ix]
x[ix] = old_val - h
random.setstate(rndstate)
fxh1, _ = f(x)

x[ix] = old_val + h
random.setstate(rndstate)
fxh2, _ = f(x)

numgrad = (fxh2 - fxh1) / (2 * h)
x[ix] = old_val
```

g)

```
### YOUR CODE HERE: forward propagation
h = sigmoid(data.dot(W1) + b1)
output = softmax(h.dot(W2) + b2)
cost = - np.sum(np.log(output[labels == 1]))
### END YOUR CODE

### YOUR CODE HERE: backward propagation
grad_output = output - labels
gradW2 = np.dot(h.T, grad_output)
gradb2 = np.sum(grad_output, axis=0, keepdims=True)
grad_h = np.dot(grad_output, W2.T) * sigmoid_grad(h)
gradW1 = np.dot(data.T, grad_h)
gradb1 = np.sum(grad_h, axis=0, keepdims=True)
### END YOUR CODE
```

3. word2vec

a) $$\frac{\partial J}{\partial v_c} = \frac{\partial J}{\partial U^T v_c} \cdot \frac{\partial U^T v_c}{\partial v_c} = (\hat{y} - y)U$$

b) $$\frac{\partial J}{\partial U} = \frac{\partial J}{\partial U^T v_c} \cdot \frac{\partial U^T v_c}{\partial U} = (\hat{y} - y)v_c$$

Solution:

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

c)

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

Similarly for CBOW, we have

$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{U}} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \boldsymbol{U}}, \quad \text{(using the definition of } \hat{\boldsymbol{v}} \text{ in the problem)}$$
$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_j} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \hat{\boldsymbol{v}}}, \quad \text{for all } j \in \{c-m, \ldots, c-1, c+1, \ldots, c+m\}$$
$$\frac{\partial J_{\text{CBOW}}(\text{word}_{c-m\ldots c+m})}{\partial \boldsymbol{v}_j} = \mathbf{0}, \quad \text{for all } j \notin \{c-m, \ldots, c-1, c+1, \ldots, c+m\}.$$

d)

```python
def normalizeRows(x):
    """ Row normalization function

    Implement a function that normalizes each row of a matrix to have
    unit length.
    """

    ### YOUR CODE HERE
    k = np.sum(x * x, axis=1, keepdims=True) ** 0.5
    x /= k + 1e-30
    ### END YOUR CODE

    return x
```

e)

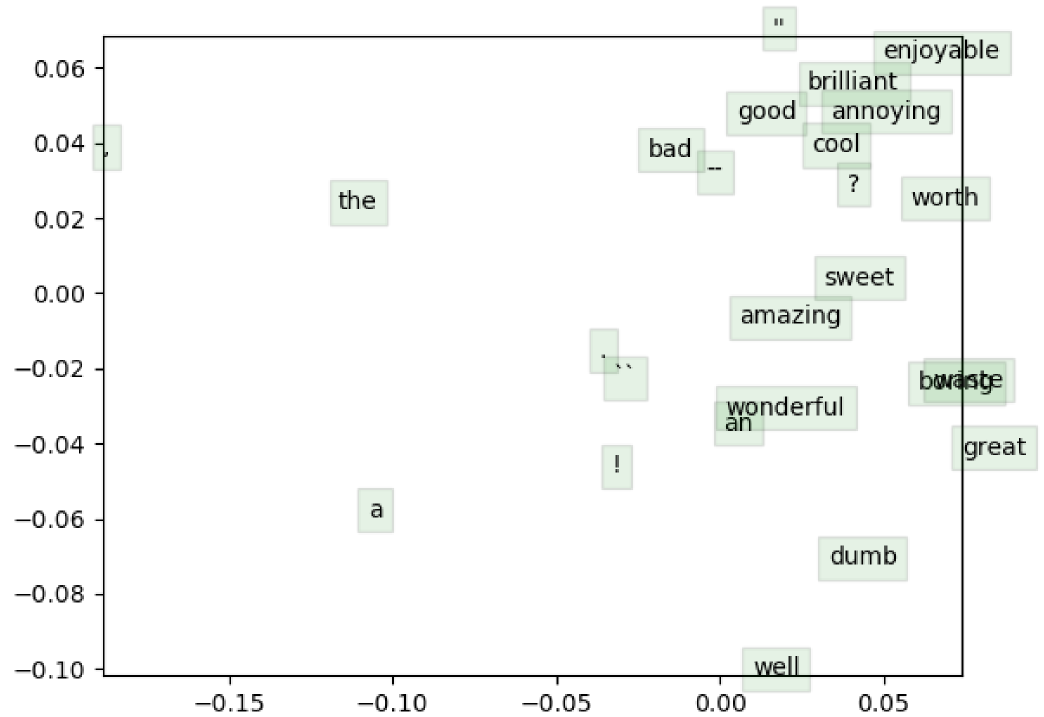
```
### YOUR CODE HERE
#vc*uoT
#1x5                    1x3           5x3
probabilities = softmax(predicted.dot(outputVectors.T))
cost = -np.log(probabilities[target])
delta = probabilities
#y^ - y
#1x5
delta[target] -= 1
#N=5
N = delta.shape[0]
#D=3
D = predicted.shape[0]
#NxD 5x3
grad = delta.reshape((N,1)) * predicted.reshape((1,D))
#1xD 1x3
gradPred = (delta.reshape((1,N)).dot(outputVectors)).flatten()
### END YOUR CODE
```

```
### YOUR CODE HERE
#5x3
grad = np.zeros(outputVectors.shape)
#1x3
gradPred = np.zeros(predicted.shape)
cost = 0
#z = uoT * vc
z = sigmoid(np.dot(outputVectors[target], predicted))
cost -= np.log(z)
grad[target] += predicted * (z - 1.0)
gradPred += outputVectors[target] * (z - 1.0)
for k in xrange(K):
    samp = dataset.sampleTokenIdx()
    z = sigmoid(np.dot(outputVectors[samp], predicted))
    cost -= np.log(1.0 - z)
    grad[samp] += predicted * z
    gradPred += outputVectors[samp] * z
### END YOUR CODE
```

```
### YOUR CODE HERE
currentI = tokens[currentWord]
predicted = inputVectors[currentI, :]
for cwd in contextWords:
    idx = tokens[cwd]
    cc, gp, gg = word2vecCostAndGradient(predicted, idx, outputVectors, dataset)
    cost += cc
    gradOut += gg
    gradIn[currentI] += gp
### END YOUR CODE
```

f)

```
### YOUR CODE HERE
cost, grad = f(x)
x = x - step * grad
x = postprocessing(x)
### END YOUR CODE
```



g)

```
### YOUR CODE HERE
D = inputVectors.shape[1]
predicted = np.zeros((D,))

indices = [tokens[cwd] for cwd in contextWords]
for idx in indices:
    predicted += inputVectors[idx, :]

cost, gp, gradOut = word2vecCostAndGradient(predicted, tokens[currentWord], outputVectors, dataset)
gradIn = np.zeros(inputVectors.shape)
for idx in indices:
    gradIn[idx] += gp
### END YOUR CODE
```

h)

4. Sentiment Analysis

```
### YOUR CODE HERE
indices = [tokens[word] for word in sentence]
sentVector = np.mean(wordVectors[indices], axis=0)
### END YOUR CODE
```

a)

b) 避免过拟合

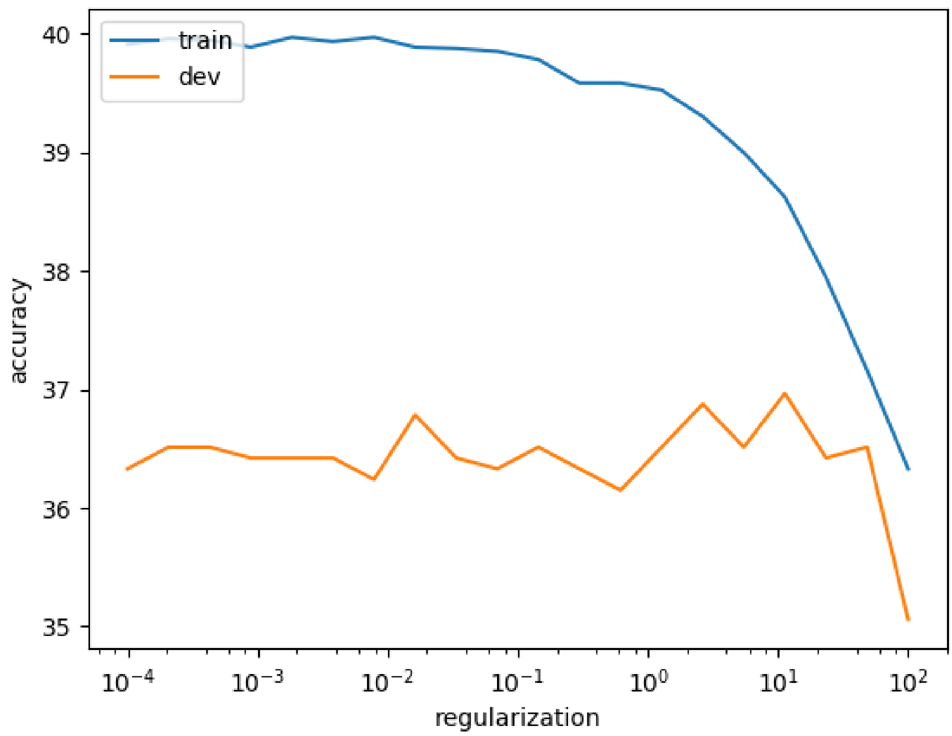
c)

```python
def getRegularizationValues():
    """Try different regularizations

    Return a sorted list of values to try.
    """
    values = None    # Assign a list of floats in the block below
    ### YOUR CODE HERE
    values = np.logspace(-4, 2, num=20, base=10)
    ### END YOUR CODE
    return sorted(values)
```

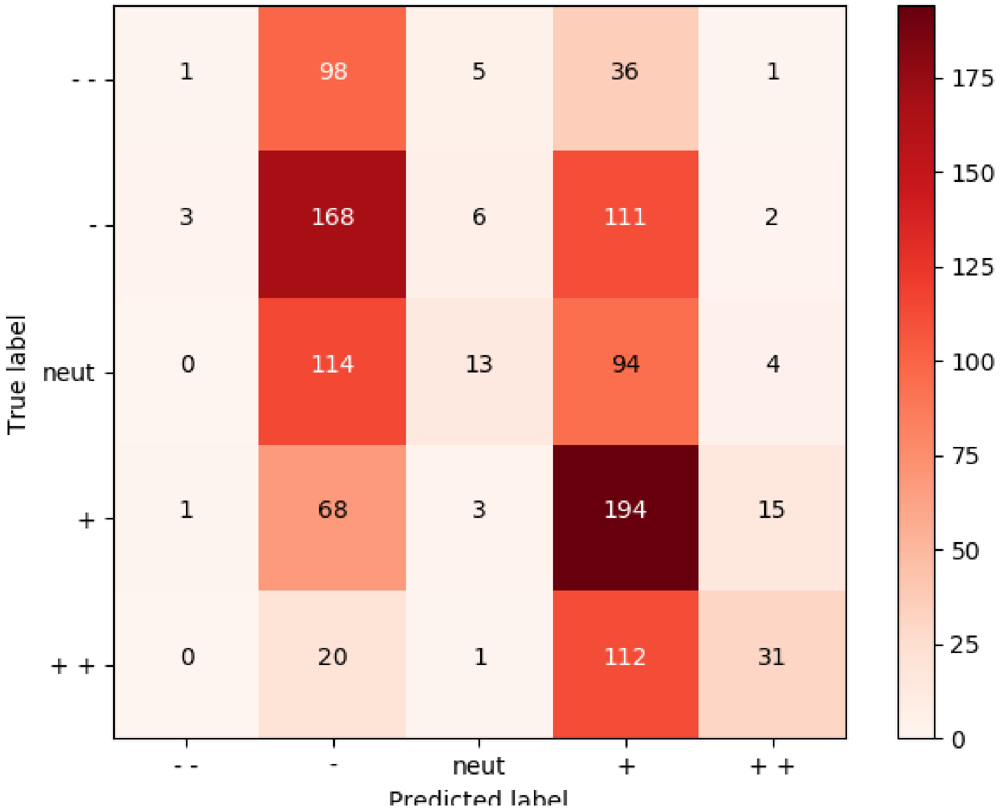
d)

```
=== Recap ===
Reg             Train   Dev     Test
1.00E-04        30.559  31.608  29.321
2.07E-04        30.583  31.608  29.276
4.28E-04        30.583  31.608  29.367
8.86E-04        30.583  31.698  29.276
1.83E-03        30.676  31.880  29.412
3.79E-03        30.618  31.244  29.593
7.85E-03        30.478  30.972  29.231
1.62E-02        30.419  31.153  29.412
3.36E-02        30.314  31.153  29.367
6.95E-02        30.255  30.972  28.688
1.44E-01        30.162  30.881  28.190
2.98E-01        29.939  30.609  27.511
6.16E-01        29.108  29.246  27.104
1.27E+00        28.464  26.794  25.339
2.64E+00        27.645  26.158  23.891
5.46E+00        27.294  25.522  23.167
1.13E+01        27.235  25.522  23.077
2.34E+01        27.235  25.522  23.032
4.83E+01        27.247  25.522  23.032
1.00E+02        27.247  25.522  23.032

Best regularization value: 1.83E-03
Test accuracy (%): 29.411765
```

```
=== Recap ===
Reg             Train   Dev     Test
1.00E-04        39.911  36.331  37.014
2.07E-04        39.958  36.512  36.968
4.28E-04        39.946  36.512  36.968
8.86E-04        39.888  36.421  37.059
1.83E-03        39.970  36.421  36.968
3.79E-03        39.934  36.421  37.149
7.85E-03        39.970  36.240  37.149
1.62E-02        39.888  36.785  37.285
3.36E-02        39.876  36.421  37.466
6.95E-02        39.853  36.331  37.195
1.44E-01        39.782  36.512  37.511
2.98E-01        39.583  36.331  37.285
6.16E-01        39.583  36.149  37.285
1.27E+00        39.525  36.512  37.330
2.64E+00        39.302  36.876  37.240
5.46E+00        38.998  36.512  37.376
1.13E+01        38.624  36.966  37.466
2.34E+01        37.945  36.421  36.923
4.83E+01        37.161  36.512  36.154
1.00E+02        36.330  35.059  35.701

Best regularization value: 1.13E+01
Test accuracy (%): 37.466063
```



e)


True label
Predicted label
- -
-
neut
+
+ +
1
98
5
36
1
3
168
6
111
2
0
114
13
94
4
1
68
3
194
15
0
20
1
112
31
0
25
50
75
100
125
150
175

f)