CS 224N: Assignment #1

writer: 纪焘

1、Softmax

(a)

$$\begin{aligned} softmax(x+c)_i &= \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} \\ &= \frac{e^{x_i}e^c}{\sum_j e^{x_j}e^c} \\ &= \frac{e^{x_i}}{\sum_j e^{x_j}} \\ &= softmax(x)_i \end{aligned}$$

(b)

```
if len(x.shape) > 1:
        # Matrix
        c = np.max(x, axis=1).reshape(x.shape[0], 1)
        x = np.exp(x - c)
        norm = np.sum(x, axis=1).reshape(x.shape[0], 1)
        x = x / norm
    else:
        # Vector
        c = np.max(x)
        x = np.exp(x - c)
        x = x / x.sum()
```

2、Neural Network Basics

(a)

$$
\begin{aligned}
\nabla_x \sigma(x) &= \nabla_x \left(\frac{1}{1+e^{-x}}\right) \\
&= \nabla_x \left[(1+e^{-x})^{-1}\right] \\
&= -(1+e^{-x})^{-2} \cdot \nabla_x (1+e^{-x}) \\
&= -(1+e^{-x})^{-2} \cdot e^{-x} \cdot \nabla_x -x \\
&= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
&= \frac{1}{(1+e^{-x})} \cdot - \frac{1}{(1+e^{-x})^2} \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}
$$

(b)

Assume that only the $\boldsymbol{k}$-th dimension of $\boldsymbol{y}$ is one.

$$
\begin{aligned}
\nabla_\theta CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \nabla_\theta \left[- \sum_i y_i \log(\hat{y_i}) \right] \\
&= \nabla_\theta \left[- y_k \log(\hat{y_k}) \right] \\
&= \nabla_\theta \left[- \log(\frac{e^{\theta_k}}{\sum_j e^{\theta_j}}) \right] \\
&= \nabla_\theta \left[- \log(e^{\theta_k}) + \log(\sum_j e^{\theta_j}) \right] \\
&= \nabla_\theta \left[\log(\sum_j e^{\theta_j}) - \theta_k \right] \\
if \quad t = k: & \\
&= \nabla_{\theta_t} \left[\log(\sum_j e^{\theta_j}) \right] - \nabla_{\theta_t} \theta_k \\
&= \frac{\nabla_{\theta_t} \left(\sum_j e^{\theta_j}\right)}{\sum_j e^{\theta_j}} - 1 \\
&= \frac{e^{\theta_t}}{\sum_j e^{\theta_j}} - 1 \\
&= \hat{\boldsymbol{y}}_t - 1 \\
if \quad t \neq k: & \\
&= \nabla_{\theta_t} \left[\log(\sum_j e^{\theta_j}) \right] - \nabla_{\theta_t} \theta_k \\
&= \frac{\nabla_{\theta_t} \left(\sum_j e^{\theta_j}\right)}{\sum_j e^{\theta_j}} \\
&= \frac{e^{\theta_t}}{\sum_j e^{\theta_j}} \\
&= \hat{\boldsymbol{y}}_t \\
\therefore \quad \nabla_\theta CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \hat{\boldsymbol{y}} - \boldsymbol{y}
\end{aligned}
$$

(c)

symbol definition:

$\boldsymbol{a} = \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1$

$\boldsymbol{h} = \sigma(\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1) = \sigma(\boldsymbol{a})$

$\boldsymbol{z} = \boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2$

$\hat{\boldsymbol{y}} = softmax(\boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2) = softmax(\boldsymbol{z})$

$$\begin{aligned}\nabla_{\boldsymbol{x}} CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \frac{\partial J}{\partial \boldsymbol{z}} CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) \cdot \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{h}} (\boldsymbol{h}\boldsymbol{W}_2 + \boldsymbol{b}_2) \cdot \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{a}} \sigma(\boldsymbol{a}) \cdot \frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}} (\boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1) \\ &= (\hat{\boldsymbol{y}} - \boldsymbol{y}) \cdot \boldsymbol{W}_2^\top \cdot \sigma(\boldsymbol{a}) \cdot (1 - \sigma(\boldsymbol{a})) \cdot \boldsymbol{W}_1^\top\end{aligned}$$

(d)

Input $\rightarrow$ **Hidden**:

$D_x * H + H$

Hidden $\rightarrow$ **Output**:

$H * D_y + D_y$

Total:

$(D_x + 1) * H + (H + 1) * D_y$

(e)

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


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

(f)

```python
def gradcheck_naive(f, x):
    ### ...
    x[ix] += h
    random.setstate(rndstate)
    fx1, _ = f(x)
    x[ix] -= 2 * h
    random.setstate(rndstate)
    fx2, _ = f(x)
    numgrad = (fx1-fx2) / (2.0*h)
    x[ix] += h
    ### ...
```

(g)

```python
def forward_backward_prop(data, labels, params, dimensions):

    ### YOUR CODE HERE: forward propagation
    M = data.shape[0]
    # (M, H)
    a = np.dot(data, W1) + b1
    hiddens = sigmoid(a)
    # (M, Dy)
    z = np.dot(hiddens, W2) + b2
    outputs = softmax(z)

    ### END YOUR CODE

    cost = -1 * labels * np.log(outputs)
    cost = cost.sum() / M

    ### YOUR CODE HERE: backward propagation

    # (M, Dy)
    gradZs = outputs - labels
    # (M, H, Dx)
    gradW2 = np.array([np.dot(hiddens[i].reshape(1, H).T,
gradZs[i].reshape(1, Dy)) for i in xrange(M)])
    # (H, Dx)
    gradW2 = gradW2.sum(axis=0) * (1.0/M)
    # (1, Dx)
    gradb2 = (gradZs.sum(axis=0) * (1.0/M)).reshape(1, Dy)
    # (M, H)
    gradAs = np.array([np.dot(gradZs[i].reshape(1, Dy),
W2.T)*sigmoid_grad(hiddens[i]) for i in xrange(M)])
    # (M, Dx, H)
    gradW1 = np.array([np.dot(data[i].reshape(1, Dx).T, gradAs[i].reshape(1,
H)) for i in xrange(M)])
    # (Dx, H)
    gradW1 = gradW1.sum(axis=0) * (1.0/M)
    # (1, H)
    gradb1 = gradAs.sum(axis=0) * (1.0/M)

    ### END YOUR CODE
```

3、word2vec

(a)

$$
\begin{aligned}
\nabla_{\boldsymbol{v}_c} CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \nabla_{\boldsymbol{v}_c} \Big[-\sum_i y_i \log(\hat{y}_i) \Big] \\
&= \nabla_{\boldsymbol{v}_c} \Big[- y_o \log(\hat{y}_o) \Big] \\
&= \nabla_{\boldsymbol{v}_c} \Big[- \log(\frac{\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)}) \Big] \\
&= \nabla_{\boldsymbol{v}_c} \Big[- \log\big(\exp(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\big) + \log\big(\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\big) \Big] \\
&= \nabla_{\boldsymbol{v}_c} \big(- \boldsymbol{u}_o^\top \boldsymbol{v}_c \big) + \nabla_{\boldsymbol{v}_c} \Big[\log\big(\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\big) \Big] \\
&= -\boldsymbol{u}_o + \frac{1}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{v}_c} \Big[\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \Big] \\
&= -\boldsymbol{u}_o + \frac{1}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \sum_{j=1}^{W} \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c) \cdot \nabla_{\boldsymbol{v}_c} (\boldsymbol{u}_j^\top \boldsymbol{v}_c)) \\
&= -\boldsymbol{u}_o + \sum_{j=1}^{W} \frac{\exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \boldsymbol{u}_j \\
&= -\boldsymbol{u}_o + \sum_{j=1}^{W} \hat{\boldsymbol{y}}_j \cdot \boldsymbol{u}_j
\end{aligned}
$$

(b)

$$
\begin{aligned}
\nabla_{\boldsymbol{u}_w} CE(\boldsymbol{y}, \hat{\boldsymbol{y}}) &= \nabla_{\boldsymbol{u}_w} \big(- \boldsymbol{u}_o^\top \boldsymbol{v}_c \big) + \nabla_{\boldsymbol{u}_w} \Big[\log\big(\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)\big) \Big] \\
if \quad w = o: & \\
&= -\boldsymbol{v}_c + \frac{1}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{u}_w} \Big[\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \Big] \\
&= -\boldsymbol{v}_c + \frac{1}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \sum_{j=1}^{W} \exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c) \cdot \nabla_{\boldsymbol{u}_j} (\boldsymbol{u}_j^\top \boldsymbol{v}_c) \\
&= -\boldsymbol{v}_c + \sum_{j=1}^{W} \frac{\exp(\boldsymbol{u}_j^\top \boldsymbol{v}_c)}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \boldsymbol{v}_c \\
&= -\boldsymbol{v}_c + \hat{y}_j \cdot \boldsymbol{v}_c \\
&= (\hat{y}_j - 1) \cdot \boldsymbol{v}_c \\
else \quad w \neq o: & \\
&= \frac{1}{\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{u}_w} \Big[\sum_{w=1}^{W} \exp(\boldsymbol{u}_w^\top \boldsymbol{v}_c) \Big] \\
&= \hat{y}_j \cdot \boldsymbol{v}_c
\end{aligned}
$$

(c)

$$
\begin{aligned}
\nabla_{\boldsymbol{v}_c} J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) &= \nabla_{\boldsymbol{v}_c} \left[-\log\left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) - \sum_{k=1}^{K} \log\left(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \right] \\
&= -\nabla_{\boldsymbol{v}_c} \left[\log\left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \right] - \nabla_{\boldsymbol{v}_c} \left[\sum_{k=1}^{K} \log\left(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \right] \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{v}_c} \left[\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right] - \sum_{k=1}^{K} \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{v}_c} \left[\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right] \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \cdot \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) \cdot \left(1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \cdot \nabla_{\boldsymbol{v}_c} (\boldsymbol{u}_o^\top \boldsymbol{v}_c) \\
&\quad - \sum_{k=1}^{K} \frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \cdot \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) \cdot \left(1 - \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \cdot \nabla_{\boldsymbol{v}_c} (-\boldsymbol{u}_k^\top \boldsymbol{v}_c) \\
&= -\left(1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \cdot \boldsymbol{u}_o - \sum_{k=1}^{K} \left(1 - \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \cdot -\boldsymbol{u}_k \\
&= \left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1\right) \cdot \boldsymbol{u}_o - \sum_{k=1}^{K} \left(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) - 1\right) \cdot \boldsymbol{u}_k
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{\boldsymbol{u}_w} J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) &= \nabla_{\boldsymbol{u}_w} \left[-\log\left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) - \sum_{k=1}^{K} \log\left(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \right] \\
&= -\nabla_{\boldsymbol{u}_w} \left[\log\left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \right] - \nabla_{\boldsymbol{u}_w} \left[\sum_{k=1}^{K} \log\left(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \right] \\
if \quad w = o: & \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{u}_w} \left[\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right] \\
&= -\frac{1}{\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)} \cdot \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) \cdot \left(1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \cdot \nabla_{\boldsymbol{u}_o} (\boldsymbol{u}_o^\top \boldsymbol{v}_c) \\
&= -\left(1 - \sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)\right) \cdot \boldsymbol{v}_c \\
&= \left(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c) - 1\right) \cdot \boldsymbol{v}_c \\
if \quad w \in \{1, \cdots, K\}: & \\
&= -\frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \cdot \nabla_{\boldsymbol{u}_w} \left[\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right] \\
&= -\frac{1}{\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)} \cdot \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c) \cdot \left(1 - \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \cdot \nabla_{\boldsymbol{u}_w} (-\boldsymbol{u}_k^\top \boldsymbol{v}_c) \\
&= -\left(1 - \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \cdot -\boldsymbol{v}_c \\
&= \left(1 - \sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c)\right) \cdot \boldsymbol{v}_c \\
else: & \\
&= None
\end{aligned}
$$

$$
\frac{\nabla_{\boldsymbol{u}_w} J_{softmax-CE}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U})}{\nabla_{\boldsymbol{u}_w} J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U})} = \frac{W}{K}
$$

(d)

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

$$if \quad i = c:$$

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

$$else \quad i \ne c:$$

$$\frac{\partial J_{skip-gram}(w_{c-m\cdots c+m})}{\partial \boldsymbol{v}_i} = \mathbf{0}$$

$$\frac{\partial J_{CBOW}(w_{c-m\cdots c+m})}{\partial \boldsymbol{U}} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \boldsymbol{U}}$$

$$if \quad i \in \{c_{window}\}:$$

$$\frac{\partial J_{CBOW}(w_{c-m\cdots c+m})}{\partial \boldsymbol{v}_i} = \frac{\partial F(\boldsymbol{w}_c, \hat{\boldsymbol{v}})}{\partial \boldsymbol{v}_i}$$

$$else \quad i \notin \{c_{window}\}:$$

$$\frac{\partial J_{CBOW}(w_{c-m\cdots c+m})}{\partial \boldsymbol{v}_i} = \mathbf{0}$$

(e)

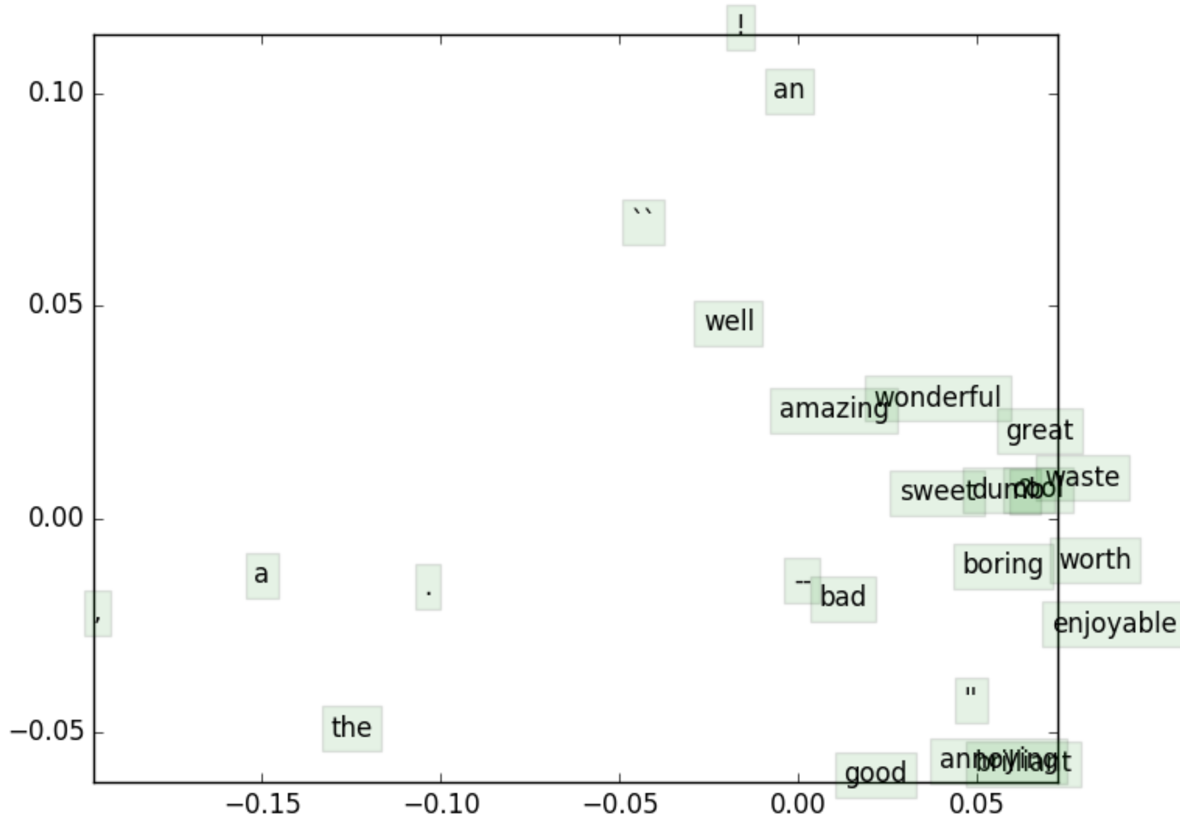
```python
def normalizeRows(x):
    ### YOUR CODE HERE
    x = x / np.sqrt(np.sum(x**2, axis=1)).reshape(x.shape[0], 1)
    ### END YOUR CODE


def softmaxCostAndGradient(predicted, target, outputVectors, dataset):
    ### YOUR CODE HERE
    v_c = predicted
    o = target
    u_o = outputVectors[target]
    y_ = softmax(v_c.dot(outputVectors.T))
    cost = -np.log(y_[o])
    y_[o] -= 1
    gradPred = (y_.reshape(1, y_.shape[0]).dot(outputVectors)).flatten()
    grad = y_.reshape(y_.shape[0], 1) * v_c.reshape(1, v_c.shape[0])
    ### END YOUR CODE


def negSamplingCostAndGradient(predicted, target, outputVectors, dataset,
                               K=10):
    ### YOUR CODE HERE
    v_c = predicted
    o = target
    u_o = outputVectors[target]
    c_o = sigmoid(v_c.dot(u_o))
```

```python
    cost = -np.log(c_o)
    gradPred = (c_o-1) * u_o

    grad = np.zeros(outputVectors.shape)
    grad[o] = (c_o-1) * v_c

    for k in indices[1:]:
        u_k = outputVectors[k]
        c_k = sigmoid(-v_c.dot(u_k))
        cost -= np.log(c_k)
        gradPred -= (c_k-1) * u_k
        grad[k] += (1-c_k) * v_c
    ### END YOUR CODE


def skipgram(currentWord, C, contextWords, tokens, inputVectors,
outputVectors,
             dataset, word2vecCostAndGradient=softmaxCostAndGradient):
    ### YOUR CODE HERE
    c = tokens[currentWord]
    v_c = inputVectors[c]
    for wordo in contextWords:
        o = tokens[wordo]
        costo, gradv_c, gradopv = word2vecCostAndGradient(v_c, o,
outputVectors, dataset)
        cost += costo
        gradIn[c] += gradv_c
        gradOut +=  gradopv
    ### END YOUR CODE
```

(f)

```python
def sgd(f, x0, step, iterations, postprocessing=None, useSaved=False,
        PRINT_EVERY=10):
    ### YOUR CODE HERE
    cost, grad = f(x)
    x -= step * grad
    x = postprocessing(x)
    ### END YOUR CODE
```

(g)



(h)

```
def cbow(currentWord, C, contextWords, tokens, inputVectors, outputVectors,
         dataset, word2vecCostAndGradient=softmaxCostAndGradient):
    ### YOUR CODE HERE
    v_c = np.zeros(inputVectors.shape[1])
    for word in contextWords:
        v_c += inputVectors[tokens[word]]
    o = tokens[currentWord]
    cost, grad, gradOut = word2vecCostAndGradient(v_c, o, outputVectors,
dataset)
    for word in contextWords:
        gradIn[tokens[word]] += grad
    ### END YOUR CODE
```

4、Sentiment Analysis

(a)

```python
def getSentenceFeatures(tokens, wordVectors, sentence):
    ### YOUR CODE HERE
    for word in sentence:
        sentVector += wordVectors[tokens[word]]
    sentVector /= len(sentence)
    ### END YOUR CODE
```

(b)

> Reduced overfitting

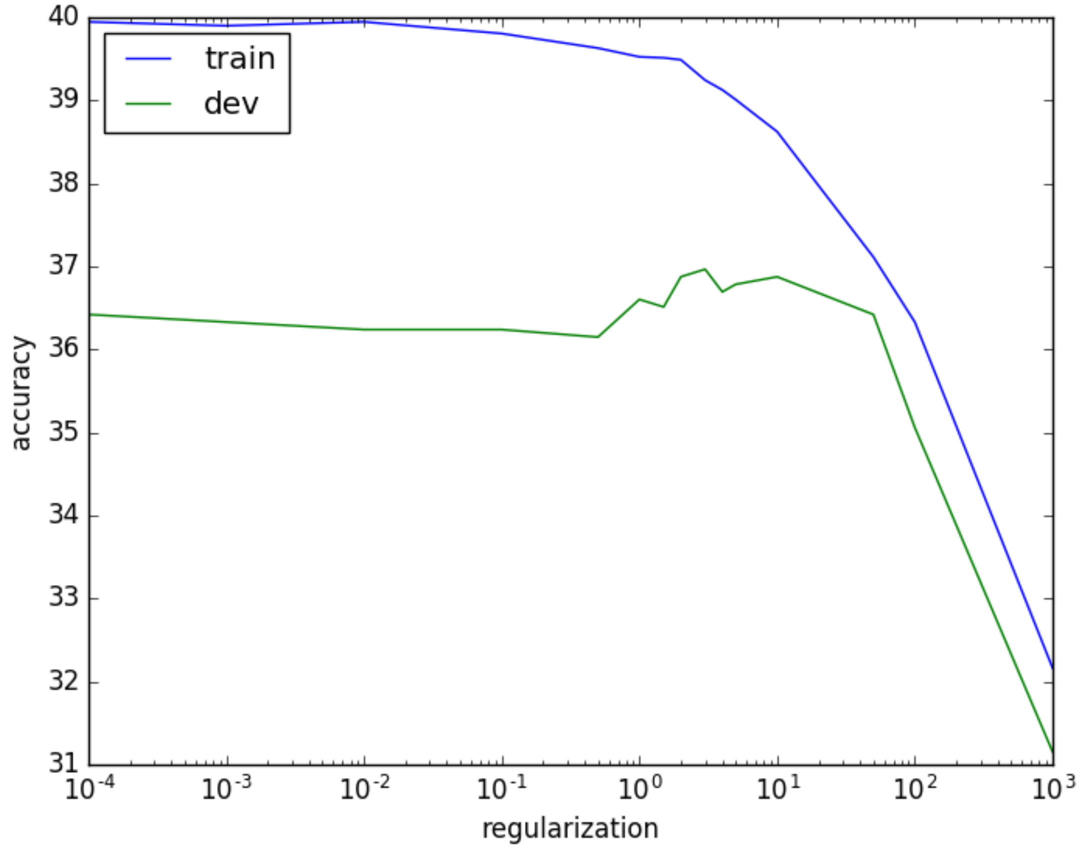
(c)

```python
def getRegularizationValues():
    values = [0.0001, 0.001, 0.01, 0.1, 0.5, 1, 1.5, 2, 3, 4, 5, 10, 50,
100, 1000]
```

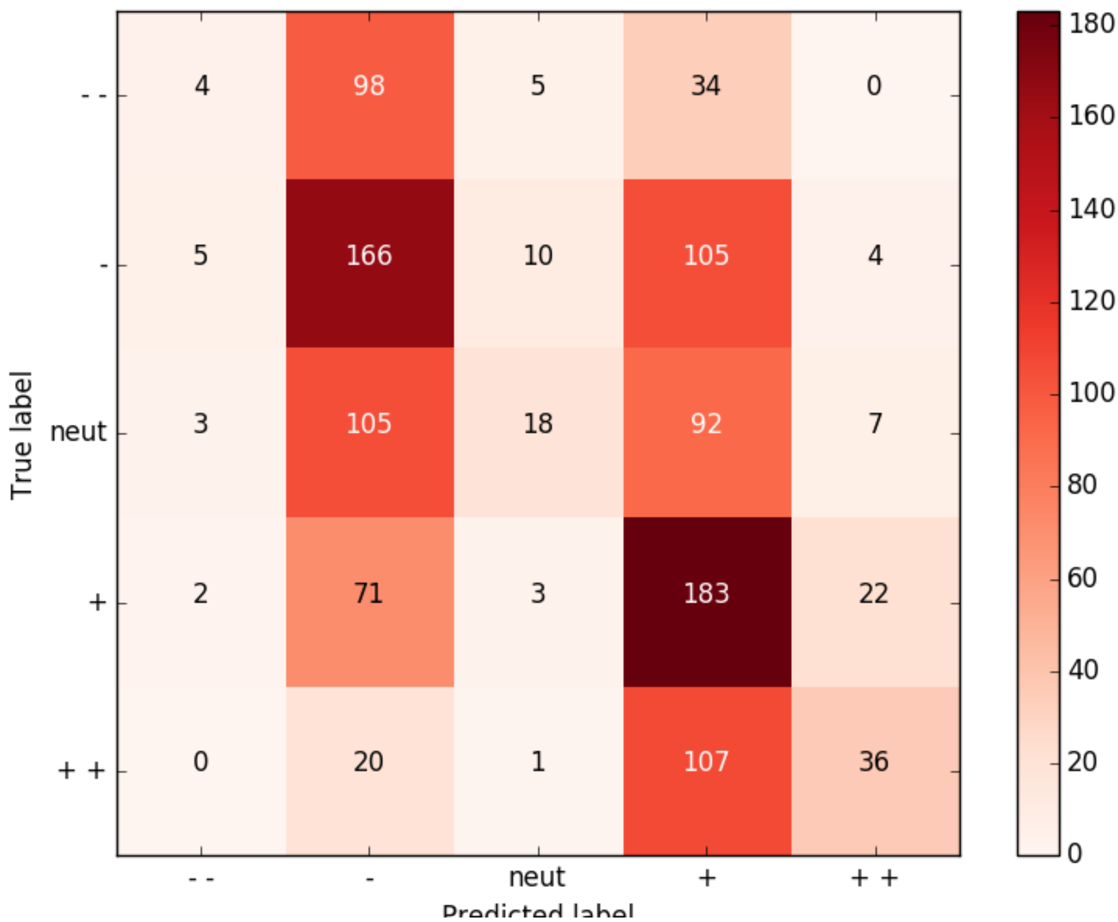
(d)

> 1. python q4 sentiment.py --yourvectors
>   Train: 31.110
>   Dev: 32.698
>   Test: 30.407
> 2. python q4 sentiment.py --pretrained
>   Train: 39.244
>   Dev: 36.966
>   Test: 37.195

(e)


train
dev
accuracy
regularization
40
39
38
37
36
35
34
33
32
31
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}

(f)


True label
Predicted label
- -
-
neut
+
+ +
4 98 5 34 0
5 166 10 105 4
3 105 18 92 7
2 71 3 183 22
0 20 1 107 36
180
160
140
120
100
80
60
40
20
0

(g)