1a.

softmax(x+c) = $\frac{e^{(Xi+C)}}{\sum_j e^{Xj+C}} = \frac{e^{C}\cdot e^{Xi}}{e^{C}\cdot\sum_j e^{Xi}} = \frac{e^{Xi}}{\sum_j e^{Xi}}$ = softmax(x)

1b.

```
orig_shape = x.shape

if len(x.shape) > 1:
    # Matrix
    ### YOUR CODE HERE
    x = x.T
    e_x = np.exp(x - np.max(x,axis=0))
    x = (e_x / e_x.sum(axis=0)).T
    ### END YOUR CODE
else:
    # Vector
    ### YOUR CODE HERE
    x = x.T
    e_x = np.exp(x - np.max(x,axis=0))
    x = (e_x / e_x.sum(axis=0)).T
    ### END YOUR CODE
assert x.shape == orig_shape
return x
```

2a.

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

2b.

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

$$= \frac{d(-y_k \log(\hat{y}_k))}{d\theta}$$

$$= \frac{d(-\log(\frac{e^{\theta_k}}{\sum_j e^{\theta_j}}))}{d\theta}$$

$$= d[-\log(\exp(e^{\theta_k}))+\log(\sum_j \log(\exp(e^{\theta_j})))]/d\theta$$

$$= d[-\theta_k+\log(\sum_j \log(\exp(e^{\theta_j})))]/d\theta$$

if t=k:

$$d[-\theta_k + \log(\sum_j \log(\exp(e^{\theta_j})))]/d\theta$$

$$= \frac{1}{\sum_j (e^{\theta_j})} \cdot \frac{d\sum_j (e^{\theta_j})}{d\theta} - 1$$

$$= \frac{e^{\theta_t}}{\sum_j (e^{\theta_j})} - 1$$

$$= \hat{y}_t - 1$$

if t $\neq$ k:

$$d[-\theta_k + \log(\sum_j \log(\exp(e^{\theta_j})))]/d\theta$$

$$= \frac{e^{\theta_t}}{\sum_j (e^{\theta_j})}$$

$$= \hat{y}_t$$

2c.

z1 = xW1 + b1
h = σ(xW1 + b1)
z2 = hW2 + b2
y = softmax(z2)
J = CE(y, $\hat{y}$)

$$\frac{dJ}{dx} = \frac{dJ}{dz_2} \quad \frac{dZ_2}{dh} \quad \frac{dh}{dz_2} \quad \frac{dz_1}{dx}$$

$$= ((\hat{y} - y) \cdot W_2^T) \circ (\sigma' (Z_1) \cdot W_1^T)$$

2d.

Dx*H + H + H*Dy + Dy

2e.

```
s =  1 / (1+np.exp(-x))
```

```
ds =  s * (1-s)
```

2f.

```
it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])
while not it.finished:
    ix = it.multi_index

    # Try modifying x[ix] with h defined above to compute
    # numerical gradients. Make sure you call random.setstate(rndstate)
    # before calling f(x) each time. This will make it possible
    # to test cost functions with built in randomness later.

    old_val = x[ix]
    x[ix] = old_val - h
    random.setstate(rndstate)
    (fxh1, _) = f(x)
    print typeof(fxh1)
    x[ix] = old_val + h
    random.setstate(rndstate)
    (fxh2, _) = f(x)

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

    # Compare gradients
    reldiff = abs(numgrad - grad[ix]) / max(1, abs(numgrad), abs(grad[ix]))
    if reldiff > 1e-5:
        print "Gradient check failed."
        print "First gradient error found at index %s" % str(ix)
        print "Your gradient: %f \t Numerical gradient: %f" % (
            grad[ix], numgrad)
        return

    it.iternext() # Step to next dimension

print "Gradient check passed!"
```

2g.

```
### YOUR CODE HERE: forward propagation

z2 = np.dot(data, W1) + b1
h2 = sigmoid(z2)
z3 = np.dot(h2, W2) + b2
y = softmax(z3)
### END YOUR CODE

M = dimensions[2]
cost = np.sum((-1*labels*np.log(y))) / M
#print cost.shape
### YOUR CODE HERE: backward propagation
delta3 = (y - labels) / M
#print delta3.shape

gradW2 = np.dot(h2.T, delta3)
#print gradW2.shape
gradb2 = np.sum(delta3,axis = 0)
#print gradb2.shape
delta2 = sigmoid_grad(h2) * np.dot(delta3, W2.T)  #hardmard product
#print delta2.shape

gradW1 = np.dot(data.T, delta2)
#print gradW1.shape
gradb1 = np.sum(delta2, axis = 0)
#print gradb1.shape
### END YOUR CODE
```

3a.

$$\frac{dCE(y,\hat{y})}{dv_c} = \frac{d[-\sum_i y_i \log(\hat{y}_i)]}{dv_c}$$

$$= \frac{d[-\log(\frac{\exp(u_o^T v_c)}{\sum_{w=1}^{W}\exp(u_w^T v_c)})]}{dV_c}$$

$$= d[-\log(\exp(u_o^T v_c)) + \log(\sum_{w=1}^{W}\exp(u_w^T v_c))]/dV_c$$

$$= -u_0 + \frac{1}{\sum_{w=1}^{W}\exp(u_w^T v_c)} \frac{d[\log(\sum_{w=1}^{W}\exp(u_w^T v_c))]}{dV_c}$$

$$= -u_0 + \sum_{j=1}^{W}(\frac{\exp(u_w^T v_c)}{\sum_{j=1}^{W}\exp(u_w^T v_c)}) \cdot u_j$$

$$= -u_0 + \sum_{j=1}^{W}\hat{y}_j \cdot u_j$$

3b.

if w = o :

$$d[-\log(\exp(u_o^T v_c)) + \log(\sum_{w=1}^{W}\exp(u_w^T v_c))]/du_w$$

$$= \quad -v_c + \sum_{j=1}^{W}(\frac{\exp(u_w^T v_c)}{\sum_{j=1}^{W}\exp(u_w^T v_c)}) \cdot v_c$$

$= \quad -v_c + \hat{y}_j \cdot v_c$

$= \quad (\hat{y}_j - 1) \cdot v_c$

else w $\neq$ o :

$= \quad \hat{y}_j \cdot v_c$

3c.

$$\frac{d\,J_{negsample}(o, v_c, U)}{d\,V_c} = -\left(\frac{1}{\sigma(u_0^T v_c)}\right) \cdot \sigma(u_0^T v_c) \cdot (1 - \sigma(u_0^T v_c)) \cdot u_0 - \sum_{k=1}^{K} \left(-\left(\frac{1}{\sigma(-u_k^T v_c)}\right) \cdot \sigma(-u_k^T v_c) \cdot (1 - \sigma(-u_k^T v_c)) \cdot -u_k\right)$$

$$= \quad (\sigma(u_o^T v_c) - 1) \cdot u_o - \sum_{k=1}^{K} (\sigma(-u_k^T v_c) - 1) \cdot u_k$$

$$\frac{dJ}{d\,u_o} = (\sigma(u_o^T v_c) - 1)\, v_c$$

$$\frac{d\,J}{d\,u_k} = -(\sigma(-u_k^T v_c) - 1)\, v_c$$

3d.

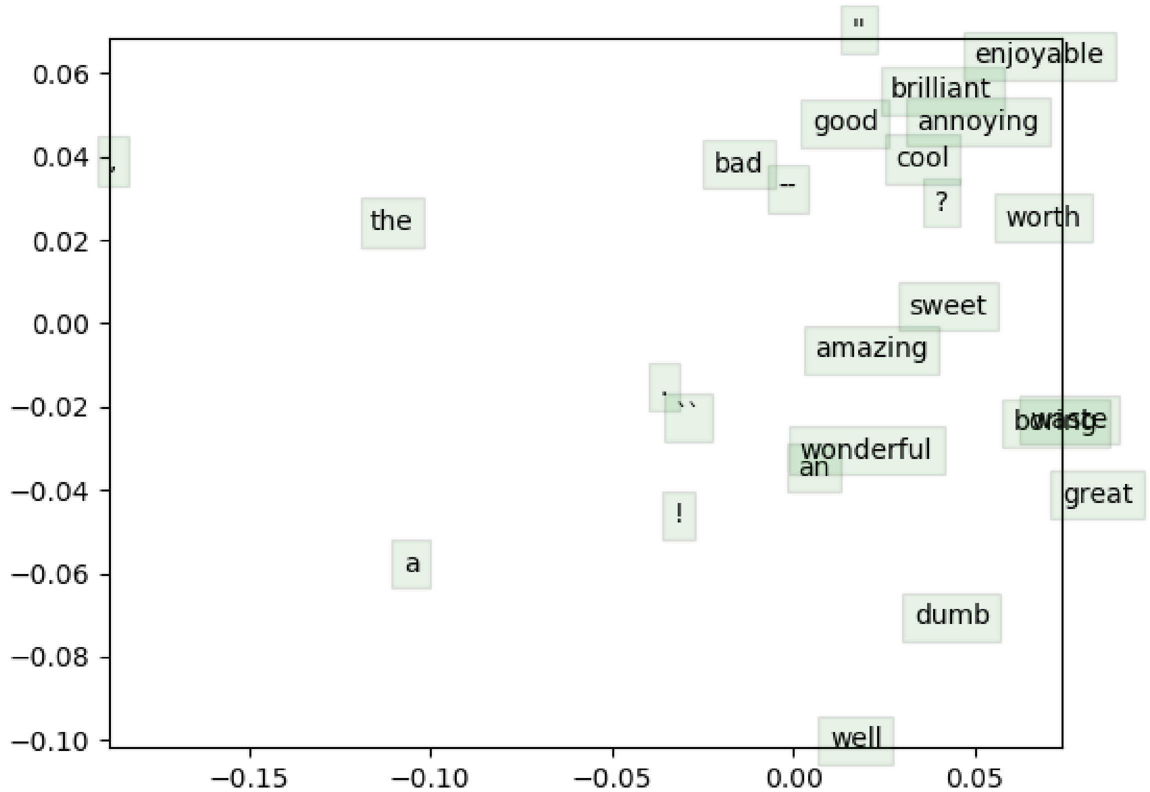
```
#  Calculate the predictions:
vhat = predicted
z = np.dot(outputVectors, vhat)
preds = softmax(z)

#  Calculate the cost:
cost = -np.log(preds[target])

#  Gradients
z = preds.copy()
z[target] -= 1.0

grad = np.outer(z, vhat)
gradPred = np.dot(outputVectors.T, z)
### END YOUR CODE
```

```python
grad = np.zeros(outputVectors.shape)
gradPred = np.zeros(predicted.shape)
cost = 0
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
```

```python
### YOUR CODE HERE
cword_idx = tokens[currentWord]
vhat = inputVectors[cword_idx]

for j in contextWords:
    u_idx = tokens[j]
    c_cost, c_grad_in, c_grad_out = \
        word2vecCostAndGradient(vhat, u_idx, outputVectors, dataset)
    cost += c_cost
    gradIn[cword_idx] += c_grad_in
    gradOut += c_grad_out
### END YOUR CODE
```

3e.

```python
cost, grad = f(x)
x -= step * grad
x = postprocessing(x)
```



4a.

```
for s in sentence:
    sentVector += wordVectors[tokens[s], :]

sentVector *= 1.0 / len(sentence)
```

4b.

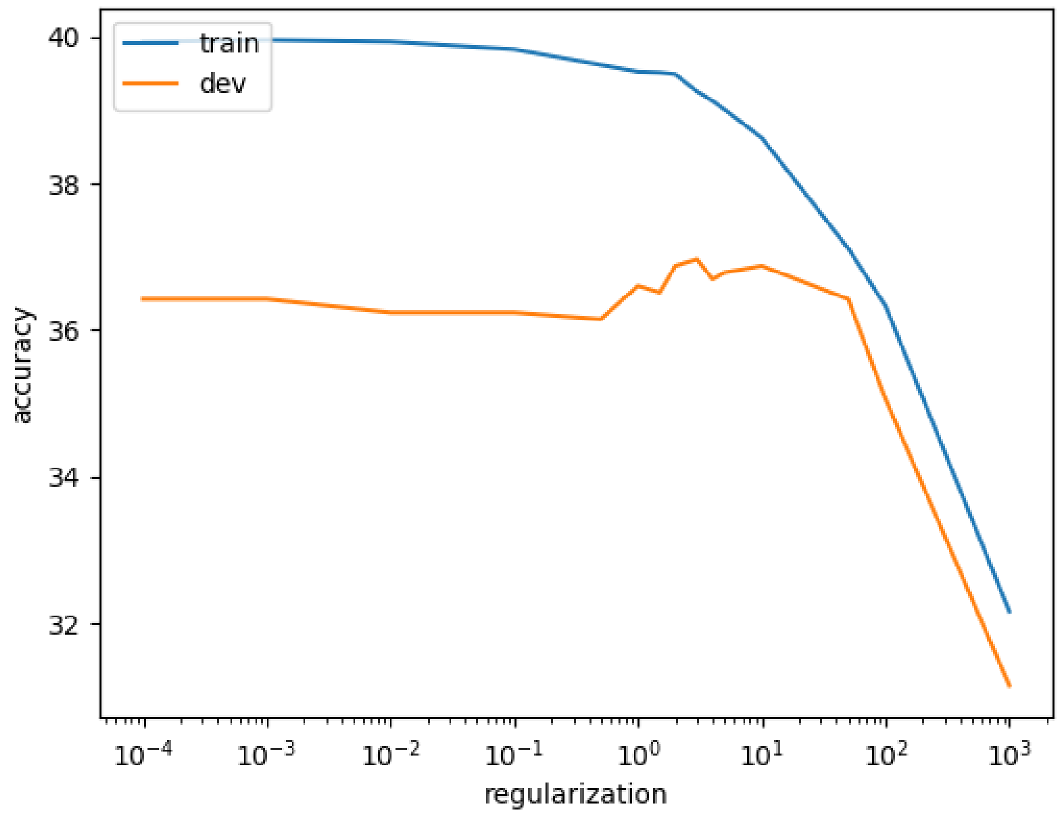
防止过拟合

4c.

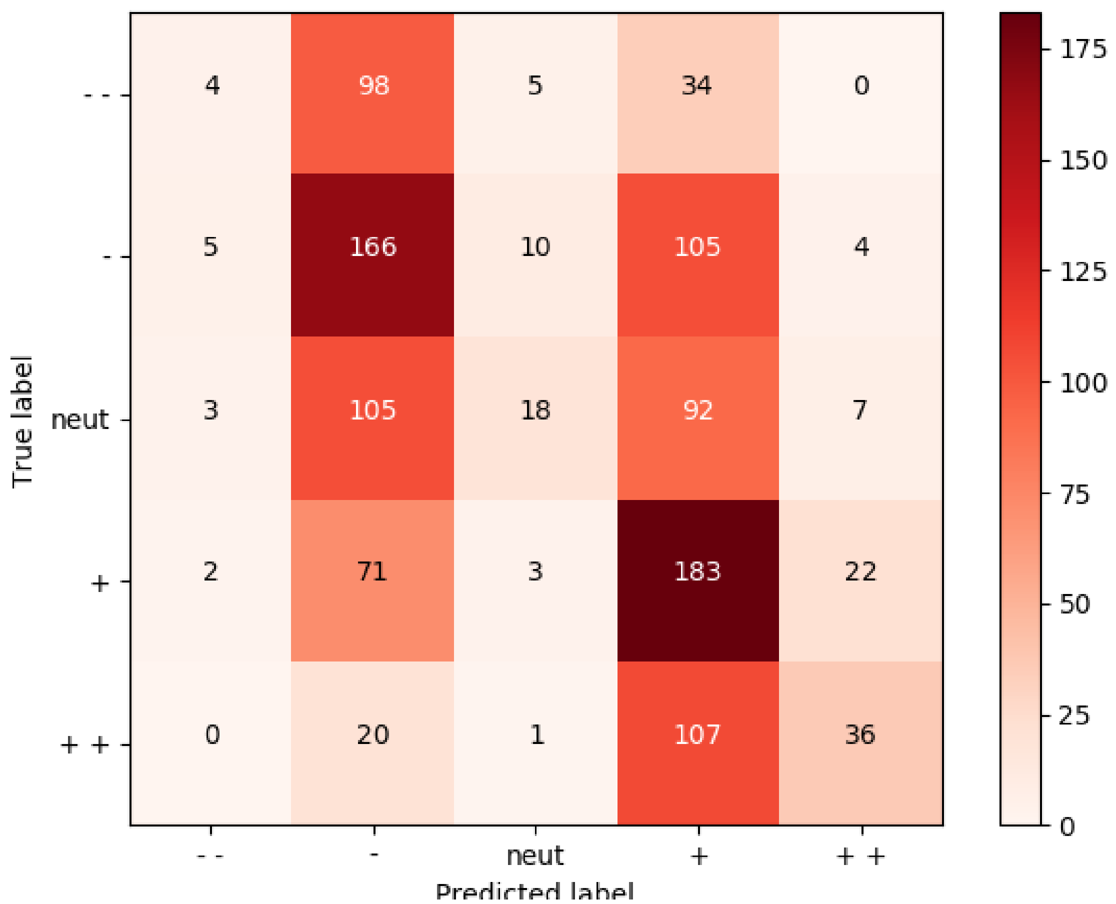
bestResult = max(results, key=lambda x: x["dev"])

4d.

4e


train
dev
accuracy
regularization
40
38
36
34
32
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}

4f.


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
175
150
125
100
75
50
25
0