Stanford University CS224N

Name: ZIAUL HASAN HASHMI

Assignment #1 Solutions

(1) SOFTMAX

(a)

For each dimension i

$$\text{softmax}(x+c)_i = \frac{e^{x_i+c}}{\sum_j e^{x_j+c}} = \frac{e^c \cdot e^{x_i}}{\sum_j e^c \cdot e^{x_j}} = \frac{e^c \cdot e^{x_i}}{e^c \cdot \sum_j e^{x_j}}$$

$$= \frac{e^{x_i}}{\sum_j e^{x_j}} = \text{softmax}(x_i)$$

Q.E.D

(b)

Please check code.

(2) NEURAL NETWORK BASICS

(a)

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

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

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

$$\hat{y} = \text{softmax}(\theta)$$

$$\Rightarrow \hat{y}_i = \frac{e^{\theta_i}}{\sum_j e^{\theta_j}}$$

$$\Rightarrow \frac{\partial \hat{y}_i}{\partial \theta_i} = \frac{e^{\theta_i}}{\sum_j e^{\theta_j}} \left(1 - \frac{e^{\theta_j}}{\sum_j e^{\theta_j}}\right) = \hat{y}_i(1 - \hat{y}_i) \quad - (1)$$
$$= -\hat{y}_i(\hat{y}_i - 1)$$

Also $$\frac{\partial \hat{y}_i}{\partial \theta_j} = \frac{-e^{\theta_i} e^{\theta_j}}{(\sum_j e^{\theta_j})^2} = -\hat{y}_i \hat{y}_j \quad - (2)$$

If k is the true class,

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

$$\therefore \frac{\partial CE}{\partial \theta_i} = \frac{-1}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial \theta_i} = \begin{cases} \hat{y}_i - 1 & \text{if } i = k \\ \hat{y}_i & \text{otherwise} \end{cases} \quad [\text{Using } (1) \text{ \& } (2)]$$

$$\therefore \boxed{\frac{\partial CE}{\partial \theta} = \hat{y} - y}$$

(c) $h = \text{sigmoid}(xW_1 + b_1)$ $\quad \hat{y} = \text{softmax}(hW_2 + b_2)$

Let say $z_1 = xW_1 + b_1$ & $z_2 = hW_2 + b_2$

$\therefore \quad h = \text{sigmoid}(z_1)$

$\hat{y} = \text{softmax}(z_2)$

$$CE(y, \hat{y}) = -\sum_i y_i \log y_i$$

$$\frac{\partial CE}{\partial z_2} = \hat{y} - y \quad \text{(proved in part (b))}$$

Using backpropagation / chain rule.

$$\frac{\partial CE}{\partial h} = \frac{\partial CE}{\partial z_2} \cdot \frac{\partial z_2}{\partial h}$$

$$= (\hat{y} - y) \cdot W_2^T$$

$$\therefore \quad \frac{\partial CE}{\partial z_1} = \frac{\partial CE}{\partial h} \odot \sigma'(z_1)$$

$$= \left[(\hat{y} - y) \cdot W_2^T\right] \odot \sigma'(z_1)$$

Following the same approach

$$\frac{\partial CE}{\partial x} = \frac{\partial CE}{\partial z_1} \cdot \frac{\partial z_1}{\partial x}$$

$$\boxed{\frac{\partial CE}{\partial x} = \left[\left[(\hat{y} - y) \cdot W_2^T\right] \odot \sigma'(z_1)\right] \cdot W_1^T}$$

(d) No. of Weights in layer 1 = $D_x \cdot H$

biases in layer 1 = H

No. of weights in layer 2 = $D_y \cdot H$

biases in layer 2 = D_y

$$\boxed{\text{Total number of parameters} = (D_x + 1)H + D_y(H + 1)}$$

Check code for parts (e), (f) & (g)

(3) word 2 vec

(a) $\hat{y}_o = p(o|c) = \dfrac{\exp(u_o^T v_c)}{\sum_{w=1}^{W} \exp(u_w^T v_c)}$

$J_{\text{softmax-CE}}(o, v_c, U) = CE(y, \hat{y})$

Assuming $\theta = [u_1^T v_c \quad u_2^T v_c \quad \dots \quad u_W^T v_c]$ - row vector of shape $1 \times W$

$\dfrac{\partial J}{\partial \theta} = \hat{y} - y$ (from part(b) Q2) assume $\hat{y} - y$ is also a row vector of shape $1 \times W$

Also $\dfrac{\partial \theta}{\partial v_c}$ is a Jacobian and it can be written in matrix form as follows

$$\frac{\partial \theta}{\partial v_c} = [u_1, u_2, u_3 \dots u_W] = U \quad \text{(dimension } W \times d\text{)}$$

And $\dfrac{\partial J}{\partial \theta} = \underset{1 \times W}{\left[\partial J/\partial \theta_1 \quad \dots \quad \partial J/\partial \theta_W\right]}$ & $\dfrac{\partial \theta}{\partial v_c} = \begin{bmatrix} \frac{\partial \theta_1}{\partial v_{c_1}} & \frac{\partial \theta_1}{\partial v_{c_2}} & \dots & \frac{\partial \theta_1}{\partial v_{c_d}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \theta_W}{\partial v_{c_1}} & \frac{\partial \theta_W}{\partial v_{c_2}} & \dots & \frac{\partial \theta_W}{\partial v_d} \end{bmatrix}_{(W \times d)}$

Since $\dfrac{\partial J}{\partial v_{c_1}} = \sum_i \dfrac{\partial J}{\partial \theta_i} \dfrac{\partial \theta_i}{\partial v_{c_1}}$

$\therefore \dfrac{\partial J}{\partial v_c}$ can be written as matrix multiplication of U & $\hat{y} - y$ after arranging dimensions.

$$\boxed{\frac{\partial J}{\partial v_c} = (\hat{y} - y)\, U}$$

$1 \times d$ row vector

Please note that multiplication order could be changed depending upon how matrix dimensions are defined

(b) Assuming $\theta = [u_1^T v_c \;\; u_2^T v_c \;\; \dots \;\; u_W^T v_c]$ - row vector of shape $1 \times W$

$$\frac{\partial J}{\partial \theta} = (\hat{y} - y) \qquad \text{(row vector of shape } 1 \times W)$$

$$\frac{\partial \theta}{\partial u_i} = \begin{bmatrix} 0 \leftarrow \text{row vector} \\ \vdots \\ v_c^T \\ \vdots \\ 0 \end{bmatrix} = v_i \quad \Rightarrow \text{is a Jacobian matrix of shape } W \times d$$

From part (a)

$$\frac{\partial J}{\partial u_i} = (\hat{y} - y)\, v_i$$

$$= [v_{c_1}(\hat{y}_i - y_i) \;\; v_{c_2}(\hat{y}_i - y_i) \dots \;\; v_{c_d}(\hat{y}_i - y_i)]$$

row vector $d \times 1$

$$\frac{\partial J}{\partial U} = \begin{bmatrix} v_{c_1}(\hat{y}_1 - y_1) & v_{c_2}(\hat{y}_1 - y_1) & \dots & v_{c_d}(\hat{y}_1 - y_1) \\ \vdots & & & \\ v_{c_1}(\hat{y}_W - y_W) & v_{c_2}(\hat{y}_W - y_W) & \dots & v_{c_d}(\hat{y}_W - y_W) \end{bmatrix} \text{ - Shape } W \times d$$

$$= (\hat{y} - y)^T v_c \qquad \text{(assuming } v_c \text{ is a row vector } 1 \times d)$$

$$\boxed{\frac{\partial J}{\partial U} = (\hat{y} - y)^T v_c}$$ where $(\hat{y} - y)$ & v_c are row vectors

The order can be different depending upon how the vector dimensions are defined.

(c)

$$J_{\text{neg-sample}}(o, v_c, U) = -\log\left(\sigma(u_o^T v_c)\right) - \sum_{k=1}^{k} \log\left(\sigma(-u_k^T v_c)\right)$$

We know that $\sigma' = \sigma(1-\sigma)$

& Also $\frac{\partial a^T x}{\partial x} = a$ & $\frac{\partial x^T a}{\partial x} = a$

$$\therefore \frac{\partial J}{\partial v_c} = -\frac{1}{\sigma(u_o^T v_c)} \times \sigma(u_o^T v_c)\left(1-\sigma(u_o^T v_c)\right) \cdot u_o - \sum_{k=1}^{k} \frac{1}{\sigma(-u_k^T v_c)} \times \sigma(-u_k^T v_c)\left(1-\sigma(-u_k^T v_c)\right)(-u_k)$$

$$\boxed{\frac{\partial J}{\partial v_c} = \left(\sigma(u_o^T v_c) - 1\right) u_o - \sum_{k=1}^{k} \left(\sigma(-u_k^T v_c) - 1\right) u_k}$$

Similarly, following the same calculations.

$$\boxed{\frac{\partial J}{\partial u_o} = \left(\sigma(u_o^T v_c) - 1\right) v_c}$$

& $$\boxed{\frac{\partial J}{\partial u_k} = -\left(\sigma(-u_k^T v_c) - 1\right) v_c \quad \text{for } k = 1,..k}$$

For the original softmax-CE loss, we have to calculate W sums of $\exp(u_w^T v_c)$ so it can be very expensive operation depending on the size of vocabulary whereas for negative sampling loss only k sums need to be calculated [Each summation item contains a dot product]. Therefore speed up ratio is k : W.

(d) $J_{\text{skip-gram}}(\text{word}_{c-m}, \dots c+m) = \sum_{-m \le j \le m,\ j \neq 0} F(w_{c+j}, v_c)$

We already know how to calculate $\frac{\partial F}{\partial v_c}$, $\frac{\partial F}{\partial U}$ $\left(\frac{\partial F}{\partial w_{(t)}}\right)$

Therefore, for skip-gram model, gradients for the context window will be given as follows

$$\frac{\partial J_{\text{skip gram}}(\text{word}_{t-m}, \dots t+m)}{\partial v_c} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(w_{t+j}, v_c)}{\partial v_c}$$

$$\frac{\partial J_{\text{skip gram}}(\text{word}_{t-m}, \dots t+m)}{\partial U} = \sum_{-m \le j \le m,\ j \neq 0} \frac{\partial F(w_{t+j}, v_c)}{\partial U}$$

for $v_j \ \forall j \neq t$ $\frac{\partial J}{\partial v_j}$ will be zero since v_j is not part of J function.

and for CBOW,

$$\frac{\partial J_{CBOW}(\text{word}_{t-m} \dots t+m)}{\partial U} = \frac{\partial F(w_t, \hat{v})}{\partial U} \quad \text{where } \hat{v} = \sum_{-m \le j \le m,\ j \neq 0} v_{t+j}$$

&

$$\frac{\partial J_{CBOW}(\text{word}_{t-m} \dots t+m)}{\partial v_j} = \frac{\partial F(w_t, \hat{v})}{\partial \hat{v}} \left[\frac{\partial \hat{v}}{\partial v_j}\right]$$

$\frac{\partial \hat{v}}{\partial v_j}$: $d \times d$ diagonal matrix with diagonal elements 1

$\frac{\partial F(w_t, \hat{v})}{\partial \hat{v}}$: $d \times 1$

for $\forall j \in \{t-m, \dots t-1, t+1, \dots c+m\}$

$= \frac{\partial F(w_t, \hat{v})}{\partial \hat{v}}$

& for $j \notin \{t-m, \dots t-1, t+1, \dots t+m\}$

$\frac{\partial J}{\partial v_j} = 0$

Proof.

$$\frac{\partial J_{CBOW}}{\partial v_j} = \underbrace{\frac{\partial F(w_t, \hat{v})}{\partial \hat{v}}}_{1 \times d} \underbrace{\frac{\partial \hat{v}}{\partial v_j}}_{d \times d} = \underbrace{\frac{\partial F(w_t, \hat{v})}{\partial v}}_{\text{diagonal matrix with diagonal entries as 1}}$$

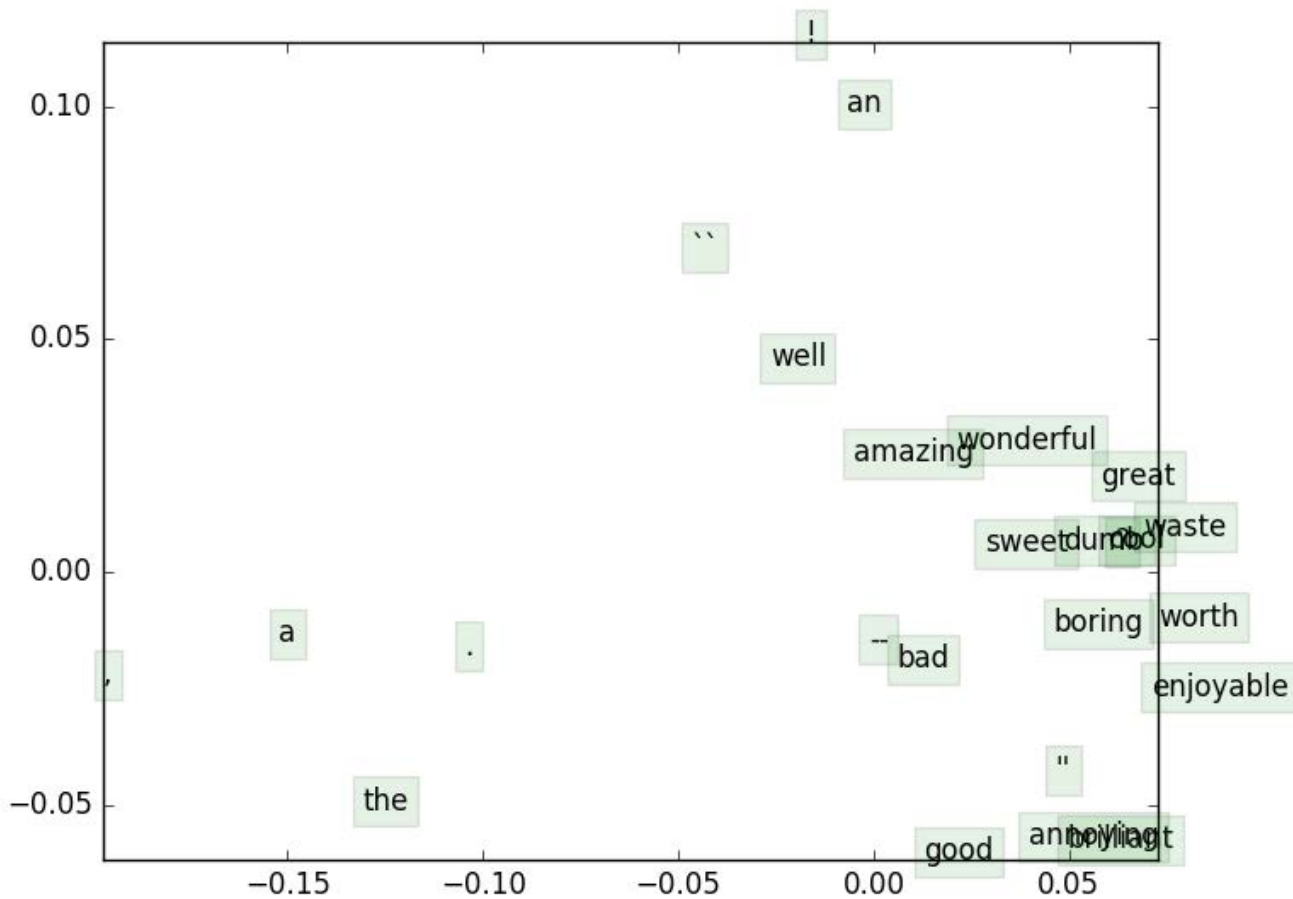
(e) Please check code

(f) Please check code

(g) Please check figure

It can be inferred from the plot that the words that occur in similar contextual meanings are located in the vicinity of each other (e.g. "amazing" & "wonderful" & "well" also "boring" & "enjoyable"

(h) please check code.


!
an
``
well
amazing
wonderful
great
waste
sweet
dumb
boring
worth
a
.
--
bad
enjoyable
"
the
good
0.10
0.05
0.00
−0.05
−0.15
−0.10
−0.05
0.00
0.05

(4) Sentiment Analysis

(a) Please check code

(b) regularization is used to solve overfitting problem in order for the model to better generalize over unseen Examples.

(c) Please check code.
(Code attached for choose best Model)

```python
def chooseBestModel(results):
    """Choose the best model based on dev set performance.

    Arguments:
    results -- A list of python dictionaries of the following format:
        {
            "reg": regularization,
            "clf": classifier,
            "train": trainAccuracy,
            "dev": devAccuracy,
            "test": testAccuracy
        }

    Each dictionary represents the performance of one model.

    Returns:
    Your chosen result dictionary.
    """
    bestResult = None

    ### YOUR CODE HERE
    best_dev_acc = 0.0
    for model in results:
        if model['dev'] > best_dev_acc:
            bestResult = model
            best_dev_acc = model['dev']
    ### END YOUR CODE
    return bestResult
```

(d)

Best train accuracy = 39.934 %
Best dev accuracy = 36.785 %
Best test accuracy = 37.1945 %

} pretrained.

Best train accuracy = 31.168 %
Best dev accuracy = 32.698 %
Best test accuracy = 30.317 %

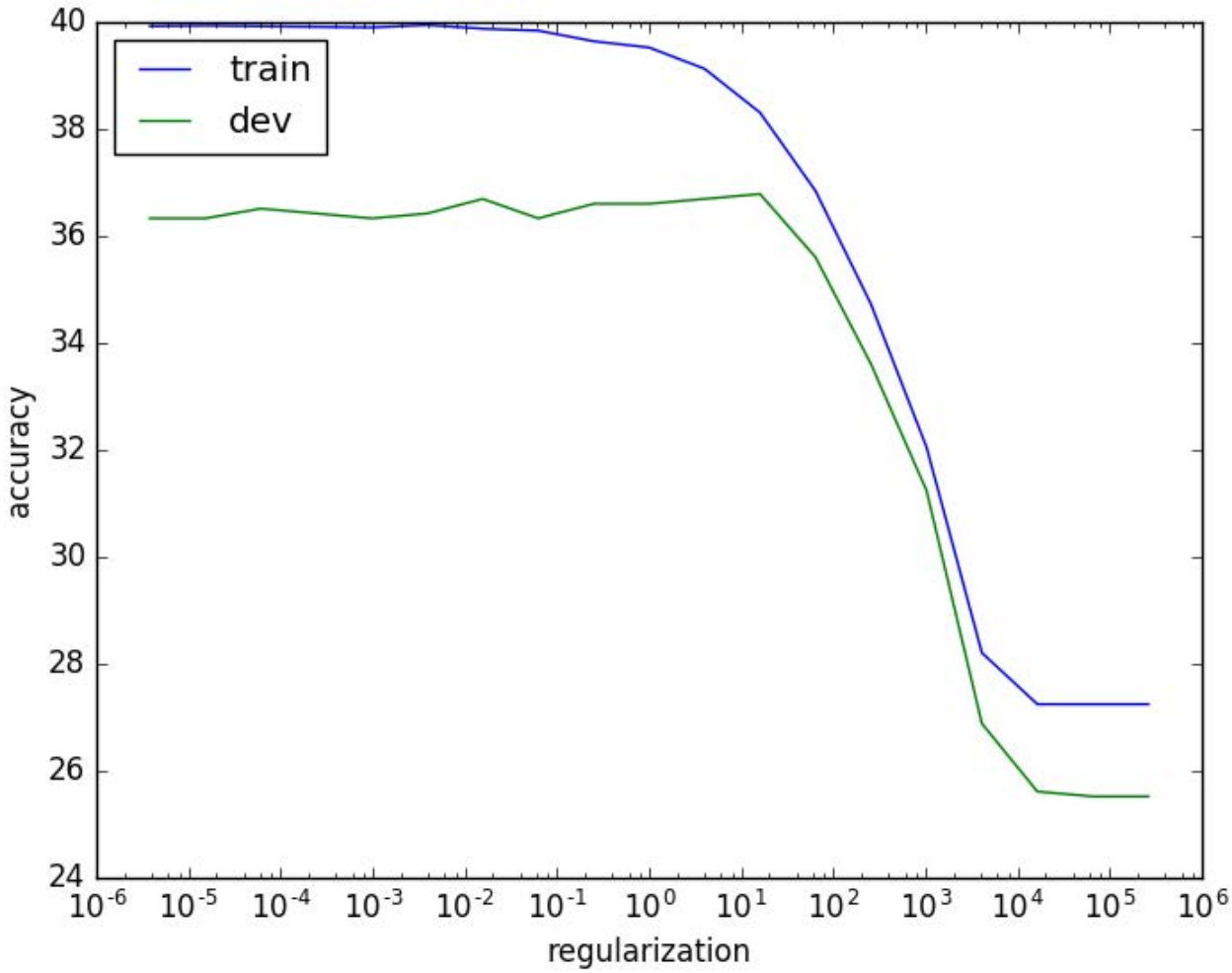
} "your vectors"

Possible reasons why pretrained gives better performance:

1) pretrained glove vectors were probably trained on larger corpus
2) Glove vectors probably has more optimized & higher dimensions compared to word2vec.
3) Glove vectors have a better objective function (loss) compared to word2vec as demonstrated by authors.

(e) Please check the plot.

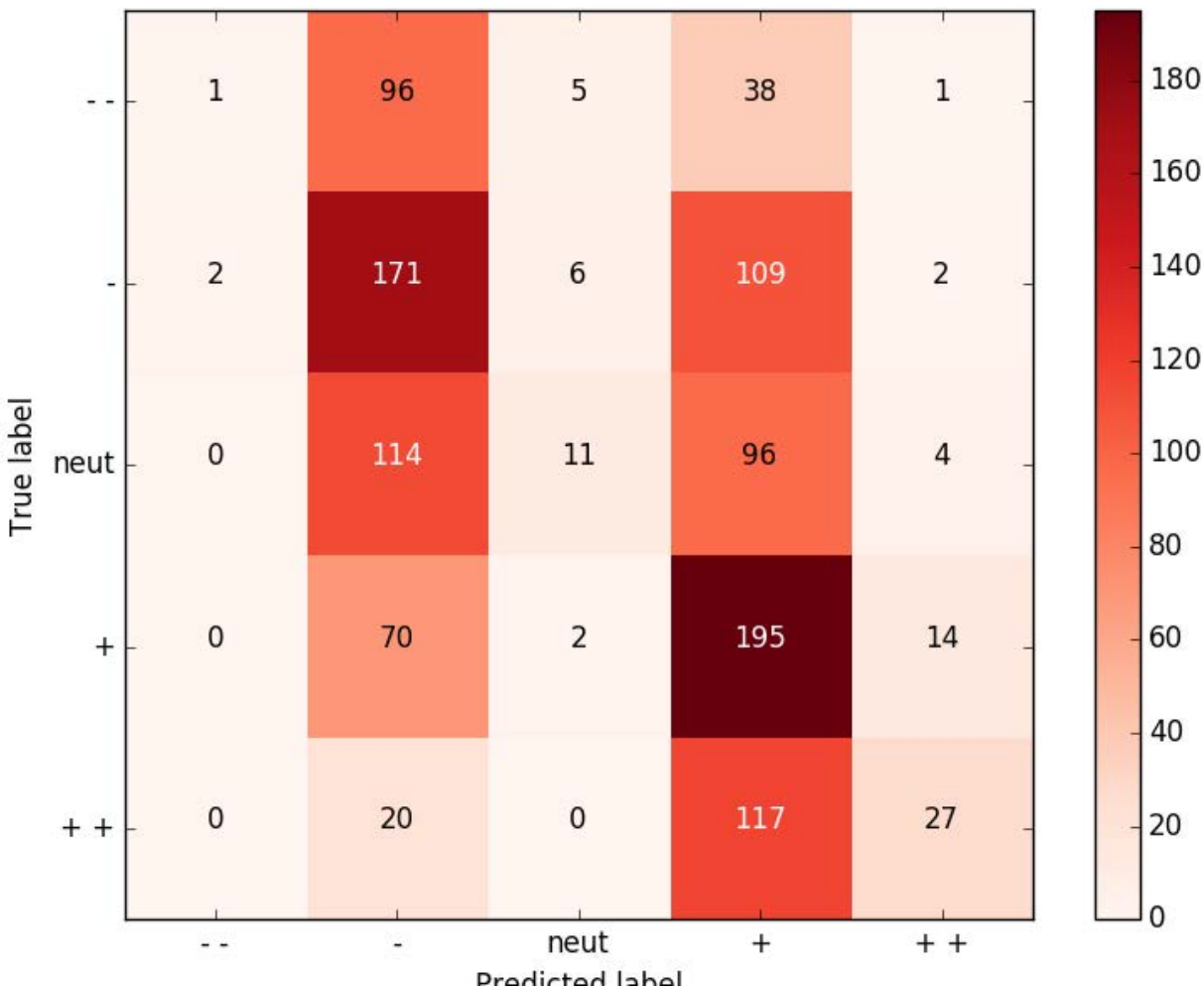
It can be seen from the plot that regularization parameter is increased training error gets closer and closer to dev set accuracy so in a way it better generalizes on unseen examples. Also if the regularization is increased too much, then both training and dev accuracy falls.


train
dev
accuracy
regularization
40
38
36
34
32
30
28
26
24
10^{-6}
10^{-5}
10^{-4}
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}

(f) Please check the plot for confusion matrix

From the confusion matrix, it can be interpreted that while in general accuracy is quite low, majority of '-' & '--' and majority of '+' & '++' are classified within their group (i.e '-' & '--' are predicted to be either '-' & '--' and via versa for '+' & '++')

Also neutral ~~set~~ sentences are mostly predicted to be '+' or '-' possibly due to mixture of words and no clear pattern.
One clear pattern is that very negative or very positive examples are rarely mixed with each other.


True label
- -
-
neut
+
+ +
1
96
5
38
1
2
171
6
109
2
0
114
11
96
4
0
70
2
195
14
0
20
0
117
27
- -
-
neut
+
+ +
Predicted label
180
160
140
120
100
80
60
40
20
0

(g) Example 1:

ultimately feels empty and unsatisfying, like swallowing a communion wafer without the wine

True : 0

Predicted : 3

In this sentence there are positive words such as "ultimately" & "communion" etc & also negative words such as "unsatisfying" and slightly negative words such as "empty" so just averaging word vectors doesn't really clarify it as a "negative" sentence.

Example 2:

falls neatly into the category of good, stupid fun.

True : 3

Predicted : 1

Probably the word "stupid" averages out the positiveness of the sentence.

Example 3:

the humor isn't as sharp, the effects not as innovative, nor the story as imaginative as in the original

True : 0

Predicted : 3

There are lots of positive words such as "sharp", "innovative" & "imginative" which probably increases the positiveness of the sentence.

Averaging doesn't seem to handle negative well & also word order is lost. Perhaps instead of averaging over words, we can average over multiple context windows.