

EEL6935 Programming HW2

Natural language processing (NLP)

Summary

The goal of this assignment is to do sentiment analysis by implementing and training a basic neural network in Python. The result show the accuracy of this basic neural network is not satisfactory (barely above 30%). This shows that sentiment analysis requires more complicated machine learning system.

Environment Setup

The project is setup in Python 2.x environment, fortunately, the package comes with a requirements.txt file showing all required python modules. After installing python with Anaconda, the setup can be easily done by simply running:

```
> pip install -r requirements.txt
```

In terminal, in my case, is Mac OS terminal running zsh.

Get Dataset

The code package doesn't comes with dataset, but a shell script to get them instead. Downloading the dataset can be done by running:

```
> cd big_data/datasets  
> bash ./get_datasets.sh
```

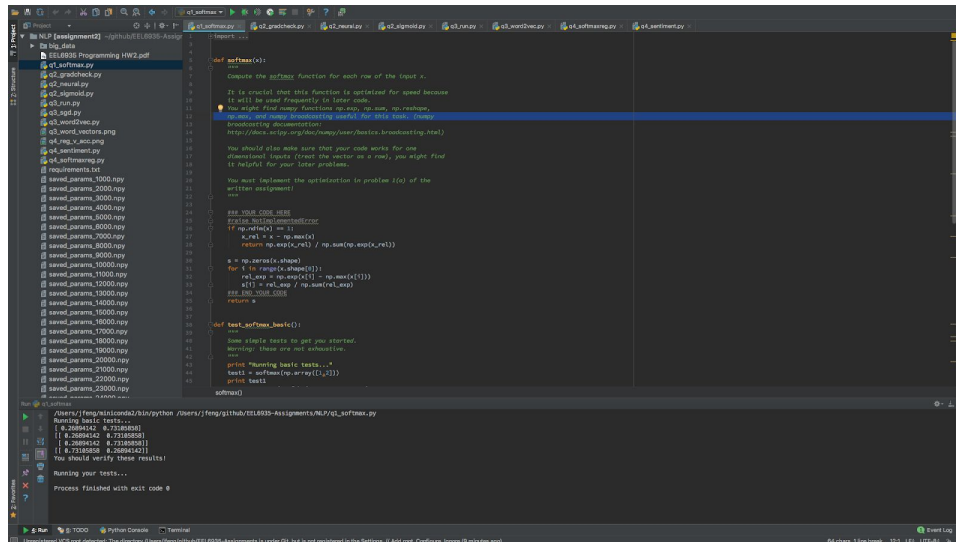
Implement and Training Neural Network

Following the homework instruction, the unfinished python code can be done. The finished codes can be found in the homework code repository hosted on GitHub:

<https://github.com/ufjfeng/EEL6935-Assignments/tree/master/NLP>

After finish each python file, the file is executed for testing purpose. The screenshots for each test result are:

q1_softmax.py



```
def softmax(x):
    """Compute the softmax function for each row of the input x.

    It is crucial that this function is optimized for speed because
    it will be used frequently in later code.
    You might find numpy functions np.sum, np.reshape,
    np.max, and numpy broadcasting useful for this task. (Copy
    the docstring for numpy broadcasting to see this task.)
    https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html)
    You should also make sure that your code works for one
    dimensional inputs (e.g. vectors) as well as 2 dimensional
    inputs (e.g. matrices). It is helpful to take the gradient of
    the softmax function as well.

    You must implement the optimization to problem 1(c) of the
    written assignment.

    """
    # YOUR CODE HERE
    #raise NotImplementedError
    if np.ndim(x) == 1:
        x = np.reshape(x, (-1,))
        return np.exp(x) / np.sum(np.exp(x))
    else:
        x = np.reshape(x, (-1,))
        for i in range(x.shape[0]):
            x[i] = x[i] - np.max(x[i])
            x[i] = np.exp(x[i]) / np.sum(np.exp(x[i]))
        return x

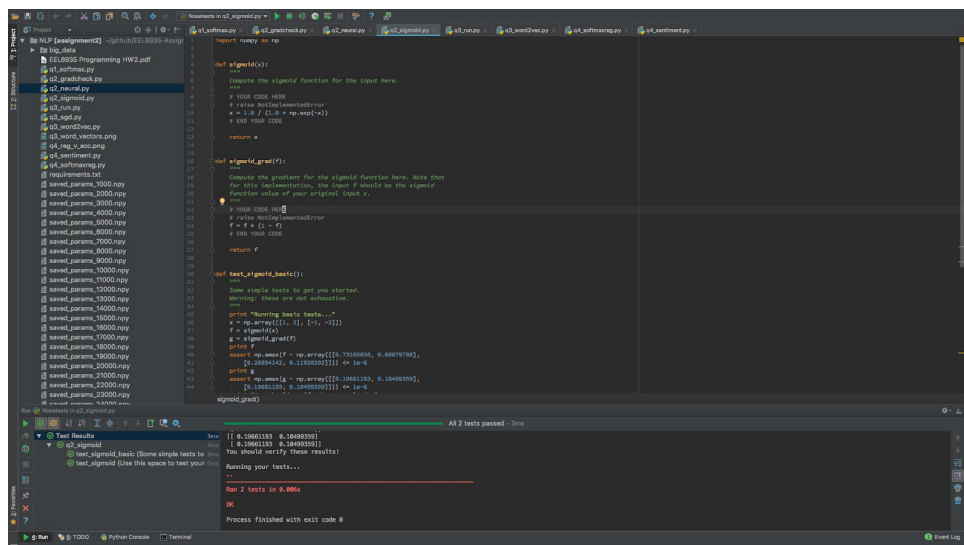
def test_softmax_basic():
    """Some simple tests to get you started.
    Warning: these are not exhaustive.
    """
    print("Running basic tests...")
    test1 = softmax(np.array([1, 2]))
    print test1
    softmax()
```

Output: [0.26894142 0.73105858]
[0.26894142 0.73105858]
[0.26894142 0.73105858]
You should verify these results!

Running your tests...
Process finished with exit code 0

The implementation of soft max is simple, the tricky part is how to avoid the floating number overflow. The method I used here is calculate the difference between each number and the maximum one to reduce the number range.

q2_sigmoid.py



```
def sigmoid(x):
    """Compute the sigmoid function for the input here.

    """
    # YOUR CODE HERE
    #raise NotImplementedError
    s = 1.0 / (1.0 + np.exp(-x))
    return s

def sigmoid_grad(f):
    """Compute the gradient for the sigmoid function here. Note that
    for this implementation, the input f should be the sigmoid
    function value of your original input x.

    """
    # YOUR CODE HERE
    #raise NotImplementedError
    f = f * (1 - f)
    return f

def test_sigmoid_basic():
    """Some simple tests to get you started.
    Warning: these are not exhaustive.
    """
    print("Running basic tests...")
    x = np.array([1, 2], [-1, -2])
    f = sigmoid(x)
    g = sigmoid_grad(f)
    print f
    assert np.allclose(f, np.array([0.73105858, 0.88077188,
                                     0.26894142, 0.11902551]) * 1e-6)
    print g
    assert np.allclose(g, np.array([0.26894142, 0.11902551,
                                     0.73105858, 0.88077188]) * 1e-6)

sigmoid_grad()
```

Test Results: All 2 tests passed - 3ms

test_sigmoid_basic (Some simple tests to...)
test_sigmoid (Use this space to test your...)
Running your tests...
Run 2 tests in 0.000s
OK
Process finished with exit code 0

The sigmoid functions and its gradient is implemented in this file. Both of them can be implemented in a line of code

$$f = 1.0 / (1.0 + \text{np.exp}(-x))$$

$$g = f * (1 - f)$$

Where x is the input of sigmoid function and f is the function value, and g is the gradient of sigmoid function with value f.

q2_neural.py

```

def forward_backward_prop(data, labels, params, dimensions):
    """ Forward and backward propagation for a two-layer sigmoid network

    Compute the forward propagation and for the cross entropy cost,
    and backward propagation for the gradients for all parameters.

    Parameters:
    data -- Numpy array of input data of shape (N, 1)
    labels -- Numpy array of target labels of shape (N, 1)
    params -- Dictionary of parameters: W1, W2, b1, b2
    dimensions -- Dictionary of dimensions: D1, D2

    Returns:
    cost -- Scalar value of the cross entropy cost
    grads -- Dictionary of gradients for all parameters
    """
    # Unpack parameters
    W1 = params['W1']
    W2 = params['W2']
    b1 = params['b1']
    b2 = params['b2']

    # YOUR CODE HERE: Forward propagation
    # Calculate the output y
    z = np.dot(data, W1) + b1
    h = sigmoid(z)
    y = np.dot(h, W2) + b2
    cost = -np.sum(labels * np.log(y))

    # YOUR CODE HERE: Backward propagation
    # Calculate the gradients
    delta = y - labels
    gradW2 = np.dot(h.T, delta)
    gradb2 = np.sum(delta, axis=0, keepdims=True)

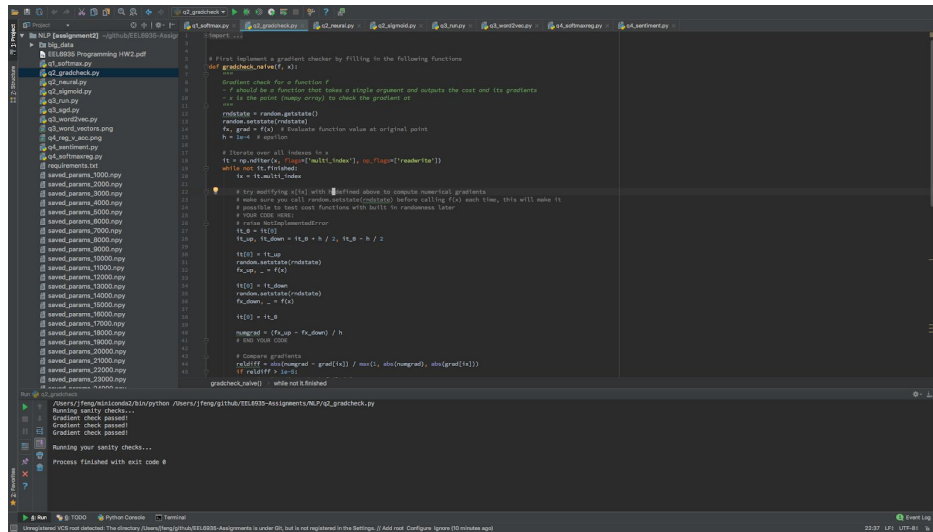
    dh = np.dot(delta, W2.T)
    dx = sigmoid_derivative(h) * dh
    gradW1 = np.dot(data.T, dx)
    gradb1 = np.sum(dx, axis=0, keepdims=True)

    # END YOUR CODE
    return cost, {'W1': gradW1, 'W2': gradW2, 'b1': gradb1, 'b2': gradb2}

```

The implementation of neural network serves as the core of this project. In this project, the neural network contains two layers with both forward and backward propagation. The function is implemented such that both cost and gradient can be calculated at the same time. Thanks to the python module numpy, the vector operation is much faster than the python built-in for loop.

q2_gradcheck.py



```
1 import numpy as np
2
3 # First implement a gradient checker by filling in the following functions
4 def gradcheck_naive(f, x):
5     """Gradient check for a function f.
6     - f should be a function that takes a single argument and outputs the cost and its gradients
7     - x is the point (numpy array) to check the gradient of
8     """
9     rndstate = random.getstate()
10    random.setstate(rndstate)
11    fx, grad = f(x) # Evaluate function value at original point
12    h = 1e-4 # direction
13
14    # Iterate over all indices in x
15    it = np.nditer(x, flags['multi_index'], op_flags=['readwrite'])
16    while not it.finished:
17        # Try modifying x[i] with h and -h to compute numerical gradients
18        # Note: we are not using np.gradient() because calling f(x) each time, this will make it
19        # possible to test each function with built-in randomness later
20        # Note: this code uses np.nditer to iterate over the multi-index
21        it_0 = it[0]
22        it_up, it_down = it_0 + h / 2, it_0 - h / 2
23        it[0] = it_up
24        random.setstate(rndstate)
25        fx_up = f(x)
26        it[0] = it_down
27        random.setstate(rndstate)
28        fx_down = f(x)
29        it[0] = it_0
30        numgrad = (fx_up - fx_down) / h
31        # Error: YOUR CODE
32
33        # Compare gradients
34        diff = abs(numgrad - grad[it_0]) / max(1, abs(numgrad), abs(grad[it_0]))
35        if diff > 1e-5:
36            gradcheck_naive = while not it.finished
```

Running: q2_gradcheck.py

```
1 Gradient check passed!
2 Gradient check passed!
3 Gradient check passed!
4 Gradient check passed!
5 Gradient check passed!
6 Gradient check passed!
7 Gradient check passed!
8 Gradient check passed!
9 Gradient check passed!
10 Gradient check passed!
11 Gradient check passed!
12 Gradient check passed!
13 Gradient check passed!
14 Gradient check passed!
15 Gradient check passed!
16 Gradient check passed!
17 Gradient check passed!
18 Gradient check passed!
19 Gradient check passed!
20 Gradient check passed!
21 Gradient check passed!
22 Gradient check passed!
23 Gradient check passed!
24 Gradient check passed!
25 Gradient check passed!
26 Gradient check passed!
27 Gradient check passed!
28 Gradient check passed!
29 Gradient check passed!
30 Gradient check passed!
31 Gradient check passed!
32 Gradient check passed!
33 Gradient check passed!
34 Gradient check passed!
35 Gradient check passed!
36 Gradient check passed!
37 Gradient check passed!
38 Gradient check passed!
39 Gradient check passed!
40 Gradient check passed!
41 Gradient check passed!
42 Gradient check passed!
43 Gradient check passed!
44 Gradient check passed!
45 Gradient check passed!
46 Gradient check passed!
47 Gradient check passed!
48 Gradient check passed!
49 Gradient check passed!
50 Gradient check passed!
51 Gradient check passed!
52 Gradient check passed!
53 Gradient check passed!
54 Gradient check passed!
55 Gradient check passed!
56 Gradient check passed!
57 Gradient check passed!
58 Gradient check passed!
59 Gradient check passed!
60 Gradient check passed!
61 Gradient check passed!
62 Gradient check passed!
63 Gradient check passed!
64 Gradient check passed!
65 Gradient check passed!
66 Gradient check passed!
67 Gradient check passed!
68 Gradient check passed!
69 Gradient check passed!
70 Gradient check passed!
71 Gradient check passed!
72 Gradient check passed!
73 Gradient check passed!
74 Gradient check passed!
75 Gradient check passed!
76 Gradient check passed!
77 Gradient check passed!
78 Gradient check passed!
79 Gradient check passed!
80 Gradient check passed!
81 Gradient check passed!
82 Gradient check passed!
83 Gradient check passed!
84 Gradient check passed!
85 Gradient check passed!
86 Gradient check passed!
87 Gradient check passed!
88 Gradient check passed!
89 Gradient check passed!
90 Gradient check passed!
91 Gradient check passed!
92 Gradient check passed!
93 Gradient check passed!
94 Gradient check passed!
95 Gradient check passed!
96 Gradient check passed!
97 Gradient check passed!
98 Gradient check passed!
99 Gradient check passed!
100 Gradient check passed!
```

Gradient check plays important role in many machine learning system. In this part of the code, the partial gradient of each dimension has been tested in different method and checked for equality. The code used for calculating test gradient is:

```
it_0 = it[0]
it_up, it_down = it_0 + h / 2, it_0 - h / 2
```

```
it[0] = it_up
random.setstate(rndstate)
fx_up, _ = f(x)
```

```
it[0] = it_down
random.setstate(rndstate)
fx_down, _ = f(x)
```

```
it[0] = it_0

numgrad = (fx_up - fx_down) / h
```

At the beginning of the q3_run.py, the PRNG is seeded as 314, so I added the same command in this code too.

```
random.seed(314)
```

q3_word2vec.py

[illegible]

In this file, we need to implement three functions:

- softmaxCostAndGradient
- negSamplingCostAndGradient
- skipgram

In order to represent the word with vector and feed them into neural networks. Hence apply stochastic gradient descent (SGD) algorithm on it.

q3_sgd.py

[illegible]

q3_run.py

For some reason yet to be discovered, q3_run.py won't run with python directly and throws an error message:

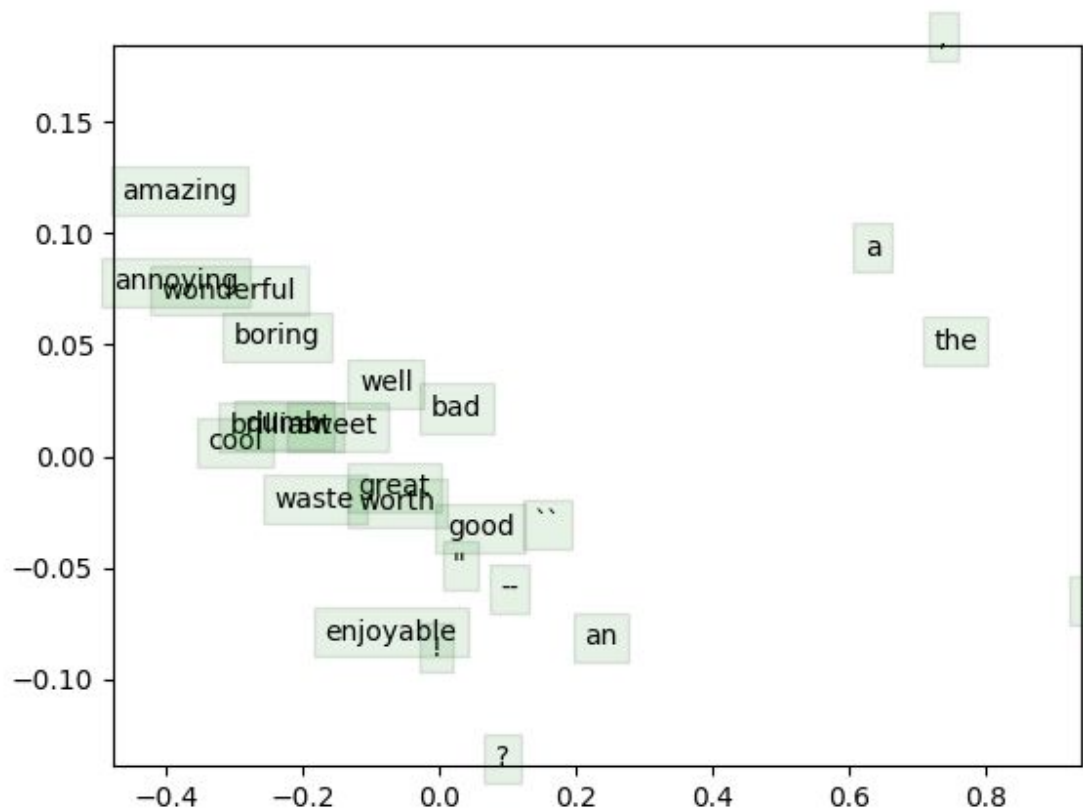
RuntimeError: Python is not installed as a framework. The Mac OS X backend will not be able to function correctly if Python is not installed as a framework. See the Python documentation for more information on installing Python as a framework on Mac OS X. Please either reinstall Python as a framework, or try one of the other backends. If you are using (Ana)Conda please install python.app and replace the use of 'python' with 'pythonw'. See 'Working with Matplotlib on OSX' in the Matplotlib FAQ for more information.

So this file has to be run in terminal.

After implementing the neural network, we can train the network with downloaded dataset by running:

```
> pythonw q3_run.py
```

This will take a while since the training has 40,000 iterations and the neural network is implemented in python. After training, a plot indicating the training result of each word vector is shown as:



q4_softmaxreg.py

```

1  # Import necessary libraries
2  import numpy as np
3  import random
4  import sys
5  import os
6  import time
7  import pickle
8  import math
9  import logging
10 import argparse
11
12 # Define the softmax regression function
13 def softmax(features, labels, weights, regularization=0.0):
14     cost, grad = softmax_regression(features, labels, weights, regularization)
15     return cost, grad
16
17 # Define the softmax regression function
18 def softmax_regression(features, labels, weights, regularization):
19     # Calculate the cost function
20     cost = 0.0
21     for i in range(len(features)):
22         # Calculate the predicted probability for each class
23         z = np.dot(features[i], weights)
24         # Calculate the softmax function
25         exp_z = np.exp(z)
26         # Calculate the predicted probability
27         p = exp_z / (np.sum(exp_z))
28         # Calculate the cost for this example
29         cost += -np.log(p[labels[i]])
30     # Calculate the average cost
31     cost /= len(features)
32     # Calculate the gradient
33     grad = np.zeros_like(weights)
34     for i in range(len(features)):
35         # Calculate the predicted probability for each class
36         z = np.dot(features[i], weights)
37         # Calculate the softmax function
38         exp_z = np.exp(z)
39         # Calculate the predicted probability
40         p = exp_z / (np.sum(exp_z))
41         # Calculate the gradient for this example
42         grad += (p - 1) * features[i]
43     # Calculate the average gradient
44     grad /= len(features)
45     # Add the regularization term to the gradient
46     grad += regularization * weights
47     return cost, grad
48
49 # Define the softmax regression function
50 def softmax_check():
51     # Load the data
52     data = load_data()
53     # Split the data into training and testing sets
54     train_data, test_data = split_data(data)
55     # Initialize the weights
56     weights = np.zeros_like(train_data[0])
57     # Train the model
58     for i in range(1000):
59         # Calculate the cost and gradient
60         cost, grad = softmax(train_data, train_data[1], weights)
61         # Update the weights
62         weights -= grad
63     # Test the model
64     test_cost, test_grad = softmax(test_data, test_data[1], weights)
65     # Print the results
66     print("Test Results: cost={}, grad={}".format(test_cost, test_grad))
67
68 # Main function
69 def main():
70     # Parse the command line arguments
71     parser = argparse.ArgumentParser()
72     parser.add_argument('--data', type=str, default='data.pkl')
73     parser.add_argument('--weights', type=str, default='weights.pkl')
74     parser.add_argument('--epochs', type=int, default=1000)
75     parser.add_argument('--regularization', type=float, default=0.0)
76     parser.add_argument('--seed', type=int, default=1)
77     parser.add_argument('--verbose', type=bool, default=False)
78     args = parser.parse_args()
79     # Set the random seed
80     random.seed(args.seed)
81     # Load the data
82     data = load_data(args.data)
83     # Split the data into training and testing sets
84     train_data, test_data = split_data(data)
85     # Initialize the weights
86     weights = np.zeros_like(train_data[0])
87     # Train the model
88     for i in range(args.epochs):
89         # Calculate the cost and gradient
90         cost, grad = softmax(train_data, train_data[1], weights, args.regularization)
91         # Update the weights
92         weights -= grad
93     # Test the model
94     test_cost, test_grad = softmax(test_data, test_data[1], weights, args.regularization)
95     # Print the results
96     print("Test Results: cost={}, grad={}".format(test_cost, test_grad))
97
98 # Run the main function
99 if __name__ == '__main__':
100    main()

```

This part of the code convert a sentence into features vector (sentence featurizer) and a softmax regression.

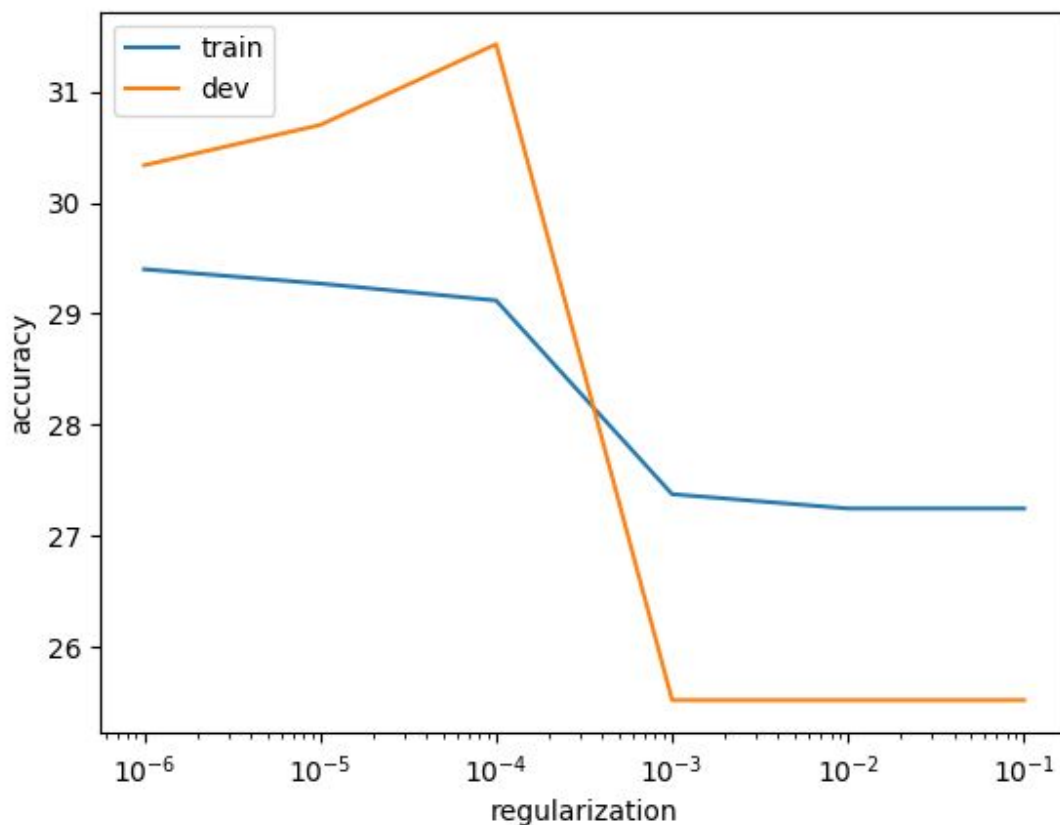
q4_sentiment.py

A proper machine learning system will not be done without the regularization term setup properly. In order to select the best regularization parameter, we ran an experiment in Q4 by testing the network performance with regularization coefficient vary from $1e-1$ to $1e-6$. This can be achieved by adding a line:

```
REGULARIZATION = [10 ** -i for i in reversed(range(1, 7))]
```

Into q4_sentiment.py.

After another simulation of, the result of different regularization result is shown.



=== Recap ===

Reg	Train	Dev
1.000000E-06	29.400749	30.336058
1.000000E-05	29.272004	30.699364

1.000000E-04	29.119850	31.425976
1.000000E-03	27.375936	25.522252
1.000000E-02	27.247191	25.522252
1.000000E-01	27.247191	25.522252

Best regularization value: 1.000000E-04

Test accuracy (%): 27.556561

The plot and recap shows that $1e-4$ seems to be a relative good choice for regularization since it produces the best result for sentiment analysis.

```

iter 6100: 1.584932
iter 6200: 1.584932
iter 6300: 1.584932
iter 6400: 1.584932
iter 6500: 1.584932
iter 6600: 1.584932
iter 6700: 1.584932
iter 6800: 1.584932
iter 6900: 1.584932
iter 7000: 1.584932
iter 7100: 1.584932
iter 7200: 1.584932
iter 7300: 1.584932
iter 7400: 1.584932
iter 7500: 1.584932
iter 7600: 1.584932
iter 7700: 1.584932
iter 7800: 1.584932
iter 7900: 1.584932
iter 8000: 1.584932
iter 8100: 1.584932
iter 8200: 1.584932
iter 8300: 1.584932
iter 8400: 1.584932
iter 8500: 1.584932
iter 8600: 1.584932
iter 8700: 1.584932
iter 8800: 1.584932
iter 8900: 1.584932
iter 9000: 1.584932
iter 9100: 1.584932
iter 9200: 1.584932
iter 9300: 1.584932
iter 9400: 1.584932
iter 9500: 1.584932
iter 9600: 1.584932
iter 9700: 1.584932
iter 9800: 1.584932
iter 9900: 1.584932
iter 10000: 1.584932
Train accuracy (%): 27.247191
Dev accuracy (%): 25.522252

=== Recap ===
Reg      Train      Dev
1.000000E-06  29.480749  30.336058
1.000000E-05  29.272804  30.699364
1.000000E-04  29.119850  31.425976
1.000000E-03  27.375936  25.522252
1.000000E-02  27.247191  25.522252
1.000000E-01  27.247191  25.522252

Best regularization value: 1.000000E-04
Test accuracy (%): 27.556561
objc[35246]: Class FIFinderSyncExtensionHost is implemented in both /System/Library/PrivateFrameworks/FinderKit.framework/Versions/A/FinderKit (0x7fff8c777b68) and /System/Library/PrivateFrameworks/FileProvider.framework/OverrideBundLes/F
inderSyncCollaborationFileProviderOverride.bundle/Contents/MacOS/FinderSyncCollaborationFileProviderOverride (0x10f9f8cd8). One of the two will be used, which one is undefined.
feng@21xin-MBP6 ~ % open .
feng@21xin-MBP6 ~ % open .

```

Analyze

With the help of numpy implementation, the vector operation used for neural networks can be done much faster than the build-in python methods, but due to the limitation of python Global Interpreter Lock (GIL), only one thread can be executed by CPU for each python process. And the vector operation can get huge performance boost from parallelization. Hence the need for GPU is obvious. A implementation in TensorFlow or Caffe should have much better performance. And the result also shows the naive neural network implementation only provides about 30% of accuracy, which is even worse than flipping a coin (50%). So an applicable sentiment analysis system should be built based on more complicated neural networks, for example CNN or RNN.