```
import numpy as np
import tensorflow as tf
import numpy as np
import review_proc as rp, preprocess, rnn, word2vec
import matplotlib.pyplot as plt
import plotutil as putil
import argparse, os, sys, random, re
from matplotlib.colors import LogNorm

parser = argparse.ArgumentParser( \
    description = 'Perform a greedy semantic attack on a recurrent neu
ral network')

parser.add_argument('-wi',
    help = 'Word to Index dictionary mapping string to integer',
    default = 'word_to_index.npy')

parser.add_argument('-iv',
    help = 'Index to Vector numpy array mapping integer to vector',
    default = 'index_to_vector.npy')

args = parser.parse_args()
word_embedding_filename = args.iv
word_to_embedding_index_filename = args.wi

try:
    word_embedding = np.load(word_embedding_filename)
    word_to_embedding_index = np.load(word_to_embedding_index_filename
).item()
except FileNotFoundError:
    print('Word embedding not found, running word2vec')
    word2vec.w2v(corpus_filename = './corpus/imdb_train_corpus.txt')

embedding_norm = np.linalg.norm(word_embedding,axis=1)
embedding_norm.shape = (10000,1)
normalized_word_embedding = word_embedding / embedding_norm
m = word_to_embedding_index
# Reverse dictionary to look up words from indices
embedding_index_to_word = dict(zip(m.values(), m.keys()))

g = tf.Graph()
rv = rp.review('./aclImdb/test/posneg/9999_10.txt')
with g.as_default():
    global_step_tensor = tf.Variable(0, trainable = False, name = 'glo
bal_step')
    # Create RNN graph
    r = rnn.classifier(
        batch_size = 1,
        learning_rate = 0.0,
        hidden_size = 16,
        max_time = 1024,
        embeddings = word_embedding,
        global_step = global_step_tensor
    )
    with tf.Session() as sess:
        tf.train.Saver().restore(sess, './ckpts/gridckpt_16_10/imdb-rn
n-e15.ckpt')
```

```
        print(rv.tokens)
        rv.translate(r.max_time,word_to_embedding_index,embedding_inde
x_to_word)
        rv.vec(word_embedding)
        decision, probability, batch_grad = r.infer_dpg(sess,rv)
        print(probability)
        rnn_sentiment = 'pos' if not decision[0] else 'neg'
        print('Neural Net Decision: ',rnn_sentiment,' Actual: ',rv.sen
timent)
        if rnn_sentiment != rv.sentiment:
            pass
        grad = batch_grad[0][0,0:rv.length,:]
        W = word_embedding; G = grad
        D = W @ (G.T)
        #print(np.amax(D),np.amin(D))
        c = np.sum(np.multiply(rv.vector_list,G),axis=1)
        d = D - c
        #np.save('predicted_diff.npy',d)
        actual_diff = np.load('actual_diff.npy')
        print(actual_diff.shape)
        print(d.shape)
        d.shape = (d.size,)
        actual_diff.shape = (d.size,)
        fig,ax = plt.subplots()
        fig.set_size_inches(5.5,10.5)
        #ax.plot(d,actual_diff,'.',ms=3)
        # Add colored dots for high norm words
        print(G.shape)
        d.shape = (10000,114)
        actual_diff.shape = (10000,114)
        n = np.linalg.norm(G,axis=1)
        i = np.argsort(n)[-10:]
        for index in reversed(i):
            print(index)
            ax.plot(d[0:1000,index],actual_diff[0:1000,index],'.',ms=2
,
                label=rv.word_list[index])
        #ax.plot([-0.2,0.2],[0.5,0.5],'k')
        plt.xlim((-0.1,0.1)); plt.ylim((-1.0,0.1))
        ax.axhline(y=0, color='k')
        ax.axvline(x=0, color='k')
        plt.title('Top 10 Gradients Norms',fontsize=30)
        plt.xlabel('Approximated Delta',fontsize=20)
        plt.ylabel('Actual Delta',fontsize=20)
        plt.tight_layout()
        plt.legend()
        fig.savefig('./writing/images/norm_color.png', dpi=300)
        plt.show()
        i = np.argmin(actual_diff)
        print(np.unravel_index(i,(10000,114)))
        print(rv.tokens[16])
        print(embedding_index_to_word[9050])
        rv.tokens[16] = embedding_index_to_word[9050]
        rv.translate(r.max_time,word_to_embedding_index,embedding_inde
x_to_word)
        decision, probability, batch_grad = r.infer_dpg(sess,rv)
        print(probability)
```