```
# Name: Cory Nezin
# Date: 03/30/2018
# Task: Perform an exponential search window attack

import tensorflow as tf
import numpy as np
import review_proc as rp
import preprocess, rnn, word2vec, wa
import plotutil as putil
import argparse, os, sys, random, re
import time
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

word_embedding = np.load('index_to_vector.npy')
word_to_embedding_index = np.load('word_to_index.npy').item()

m = word_to_embedding_index
embedding_index_to_word = dict(zip(m.values(),m.keys()))

k = 129
c = [0]*1024
t = []
f = 0
root_dir = './aclImdb/test/posneg/'
restore_name = '4786_9.txt'
flag = True
for file_name in os.listdir(root_dir):
    if file_name != restore_name and flag:
        print(file_name)
        continue
    elif flag:
        flag = False
        continue
    word_count = 0
    g = tf.Graph()
    print('Running attack on: ' + file_name)
    rvo = rp.review(root_dir + file_name)
    rvo.translate(rvo.length, word_to_embedding_index, embedding_index
_to_word)
    rvo.vec(word_embedding)
    # Actual Neural Network
    with g.as_default():
        global_step_tensor = tf.Variable(0, trainable=False, name='glo
bal_step')
        r = rnn.classifier(
            batch_size = 1,
            learning_rate = 0.0,
            hidden_size = 8,
            max_time = rvo.length,
            embeddings = word_embedding,
            global_step = global_step_tensor)
        with tf.Session() as sess:
            restore_name = './ckpts/gridckpt_8_10/imdb-rnn-e15.ckpt'
            tf.train.Saver().restore(sess,restore_name)
            decision, _ , _ = r.infer_dpg(sess,rvo)
    rnn_sentiment = 'pos' if not decision[0] else 'neg'
    if rnn_sentiment != rvo.sentiment:
        print('Neural net was wrong, continuing...')
        continue
    print('Starting clock...')
    t0 = time.clock()
    rv = rp.review(root_dir + file_name)
    rv.translate(rvo.length, word_to_embedding_index, embedding_index_
to_word)
    rv.vec(word_embedding)
    ii = 0;
    ind = np.random.permutation(rv.length)
    # infer stand in gradient
    print('Starting loop')
    print(rnn_sentiment,rvo.sentiment)
    while  rnn_sentiment == rvo.sentiment:
        print(ii,end=' ',flush=True)
        g = tf.Graph()
        with g.as_default():
            global_step_tensor = tf.Variable(0, trainable=False, name=
'global_step')
            r = rnn.classifier(
                batch_size = 1,
                learning_rate = 0.0,
                hidden_size = 16,
                max_time = rv.length,
                embeddings = word_embedding,
                global_step = global_step_tensor)
            with tf.Session() as sess:
                restore_name = './ckpts/gridckpt_16_10/imdb-rnn-e15.ck
pt'
                tf.train.Saver().restore(sess,restore_name)
                decision, probability, grad = r.infer_dpg(sess,rvo)
        grad = grad[0][0,:,:]
        # select a word i in the sequence x*
        if ii == 1024 or ii >= rvo.length:
            break
            f = f + 1
        i = ind[ii]
        ii = ii + 1
        # grad is rv.length by 128 (?)
        diff = np.sign(word_embedding[rv.index_vector[0,i],:] - word_e
mbedding)
        if rvo.sentiment == 'pos':
            jack = np.sign(grad[i,:])
        else:
            jack = -np.sign(grad[i,:])
        sgn_diff = diff-jack
        sgn_norm = np.linalg.norm(sgn_diff,axis=1,ord=0)
        j = np.argmin(sgn_norm)
        rv.index_vector[0,i] = j
        word_count = word_count + 1
        g = tf.Graph()
        # Infer decision
        with g.as_default():
            global_step_tensor = tf.Variable(0, trainable=False, name=
'global_step')
            r = rnn.classifier(
                batch_size = 1,
                learning_rate = 0.0,
                hidden_size = 8,
                max_time = rvo.length,
                embeddings = word_embedding,
                global_step = global_step_tensor)
            with tf.Session() as sess:
                restore_name = './ckpts/gridckpt_8_10/imdb-rnn-e15.ckp
t'
                tf.train.Saver().restore(sess,restore_name)
                decision, probability , _ = r.infer_dpg(sess,rv)
        rnn_sentiment = 'pos' if not decision[0] else 'neg'

    c[word_count] = c[word_count] + 1
    t1 = time.clock()
    print('')
    print('Time taken',t1-t0)
    t.append(t1-t0)
    np.save('./fgsm_black/t/'+file_name[:-4] + '.npy',np.array(t))
    np.save('./fgsm_black/c/'+file_name[:-4] + '.npy',np.array(c))
    np.save('./fgsm_black/f/'+file_name[:-4] + '.npy',np.array(f))
    print('histogram:',c)
    print('fails:',f)
    print('total:',sum(c))
```