```
# 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 = 32
c = [0]*1024
t = []
f = 0
root_dir = ’./aclImdb/test/posneg/’
for file_name in os.listdir(root_dir):
    t0 = time.clock()
    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)
    # Generating the gradient
    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 = 16,
            max_time = rvo.length,
            embeddings = word_embedding,
            global_step = global_step_tensor)
        with tf.Session() as sess:
            restore_name = ’./ckpts/gridckpt_16_10/imdb-rnn-e15.ckpt’
            tf.train.Saver().restore(sess,restore_name)
            decision, probability, grad = r.infer_dpg(sess,rvo)
    grad = np.linalg.norm(grad[0][0,:,:],axis=1)
    window_size = 200
    hidden_size = 16
    restore_name = ’./ckpts/gridckpt_16_10/imdb-rnn-e15.ckpt’
    #ii,jj,pp = wa.win_atk(rvo, window_size, word_embedding, hidden_si
ze, restore_name)
    ii,jj,pp = wa.gaws(rvo, window_size, word_embedding, \
                       hidden_size, restore_name,grad,k,decision)
    if ii is None:
        continue
    if rvo.sentiment == 'pos':
        args = np.argsort(pp)
    else:
        args = np.flip(np.argsort(pp),axis=0)
    ii = ii[args]
    jj = jj[args]
    minm = float('inf')
    R = 1
    L = 0
    val_found = False
    print('Original review:')
    print(' '.join(rvo.tokens))
    while True:
        rv = rp.review(root_dir + file_name)
        if not val_found:
            R = R * 2
        m = (L + R) // 2 # m initialized as 1
        if m >= args.size:
            print('No derivation found.')
            f = f + 1
            break
        g = tf.Graph()
        ix = ii[:m+1]
        jx = jj[:m+1]
        w = [embedding_index_to_word[i] for i in ix]
        for n,j in enumerate(args[:m+1]):
            rv.tokens[j] = w[n]
        rv.translate(rv.length, word_to_embedding_index, embedding_ind
ex_to_word)
        rv.vec(word_embedding)
        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, batch_grad = r.infer_dpg(sess,r
v)
                rnn_sentiment = 'pos' if not decision[0] else 'neg'
                if rnn_sentiment == rv.sentiment and val_found:
                    L = m + 1
                elif rnn_sentiment != rv.sentiment:
                    print('New detected sentiment:')
                    print(rnn_sentiment)
                    print('Number of changed words:')
                    print(m+1)
                    print('ix:',ix)
                    print('jx:',jx)
                    print('Replaced',[rvo.tokens[args[n]] for n in ran
ge(m+1)]\
                          ,'with',[w[n] for n in range(m+1)])
                    np.save('./gaws_testing/ix/'+file_name[-4:]+'.npy'
,ix)
                    np.save('./gaws_testing/jx/'+file_name[-4:]+'.npy'
,jx)
                    minm = min(m+1,minm)
                    val_found = True
                    R = m - 1
                if L > R:
                    print('Search complete, minimum value is: ',minm)
                    c[minm] += 1
                    break
    t1 = time.clock()
    print('Time taken',t1-t0)
    t.append(t1-t0)
    np.save('./gaws_testing/ii/' + file_name[:-4] + '.npy',ii)
    np.save('./gaws_testing/jj/' + file_name[:-4] + '.npy',jj)
    np.save('./gaws_testing/pp/' + file_name[:-4] + '.npy',pp)
    np.save('./gaws_testing/t/'+file_name[:-4] + '.npy',np.array(t))
    if sum(c) % 50 == 0:
        print('histogram:',c)
        print('fails:',f)
```