**东北大学**

语言分析与机器翻译报告

|  |  |
| --- | --- |
| 报告题目： | 基于LSTM的情感分析 |
| 学 院： | 计算机科学与工程学院 |
| 年 级： | 2020级 |
| 专 业： | 计算机科学与技术 |
| 姓 名： | 赵肖肖 |
| 学 号： | 2001881 |

2020 年 12 月 15 日

目录

[一、 主要内容 3](#_Toc59329830)

[二、循环神经网络及LSTM模型 3](#_Toc59329831)

[1.1循环神经网络ＲＮＮ的结构及原理 3](#_Toc59329832)

[1.2 LSTM 3](#_Toc59329833)

[1.3关于我的理解 4](#_Toc59329834)

[1.3.1我对循环神经网络的理解 5](#_Toc59329835)

[1.3.2我对LSTM技术的理解： 6](#_Toc59329836)

[三、代码解读 7](#_Toc59329837)

[3.1数据预处理 7](#_Toc59329838)

[3.1.1数据了解 7](#_Toc59329839)

[3.1.2代码分析 8](#_Toc59329840)

[3.2用tensorflow搭建模型 9](#_Toc59329841)

[附录一 12](#_Toc59329842)

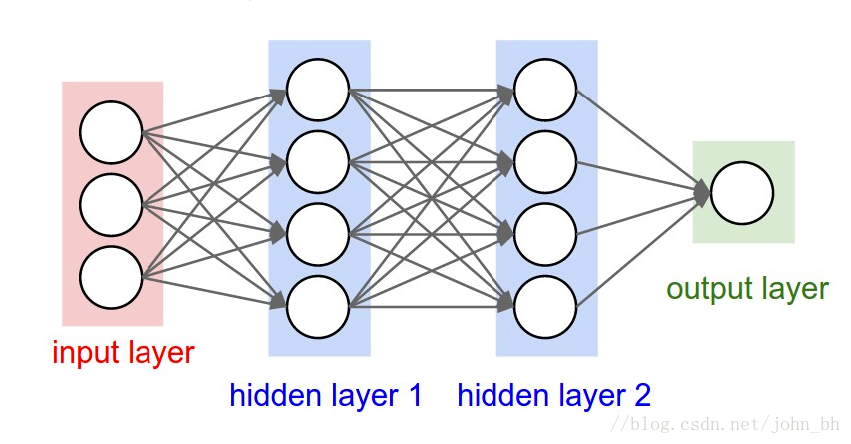
# 主要内容

本报告中首先介绍了循环神经网络RNN以及LSTM模型的结构及原理，以及在学习后自己对循环神经网络以及LSTM的理解，以及二者的对比。然后本报告中还介绍了基于LSTM模型的情感分析的实现过程。通过解读代码，对LSTM模型有了更深的理解和体会。

# 二、循环神经网络及LSTM模型

## 1.1循环神经网络ＲＮＮ的结构及原理

一般的神经网络如以下结构图所示：



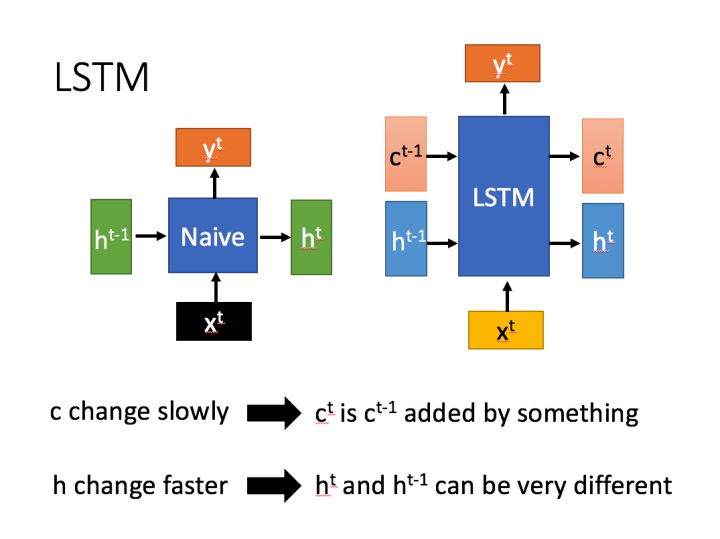
　　　　　　　　　　图一　神经网络结构图

RNN的目的是处理序列数据

RNN的具体做法为网络会对前面的信息进行记忆，并应用于当前输出的计算中，即隐层之间的节点不再无连接，并且隐层的输入不仅包括输入层的输出，还包括上一时刻隐层的输出。  
RNN的功能特点：1、隐层节点之间可以互连也可以自连2、网络共享权重U、V、W3、RNN网络中，每一步的输出不是必须的，每一步的输入也不是必须的。

## 1.2 LSTM

长短期记忆（Long short-term memory, LSTM）是一种特殊的RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的RNN，LSTM能够在更长的序列中有更好的表现。



与RNN重复模块链的简单的一层相比，LSTM拥有四层，这四层以特殊的方式进行交互。“忘记门” Sigmoid层实现-> 存储 tanh层 创建候选向量Ct -> 更新 将上一个状态值Ct−1更新为Ct-> 输出 运行一个sigmoid层，它决定了我们要输出的细胞状态的哪些部分。 然后，我们将单元格状态通过tanh（将值规范化到-1和1之间），并将其乘以Sigmoid门的输出，至此我们只输出了我们决定的那些部分。

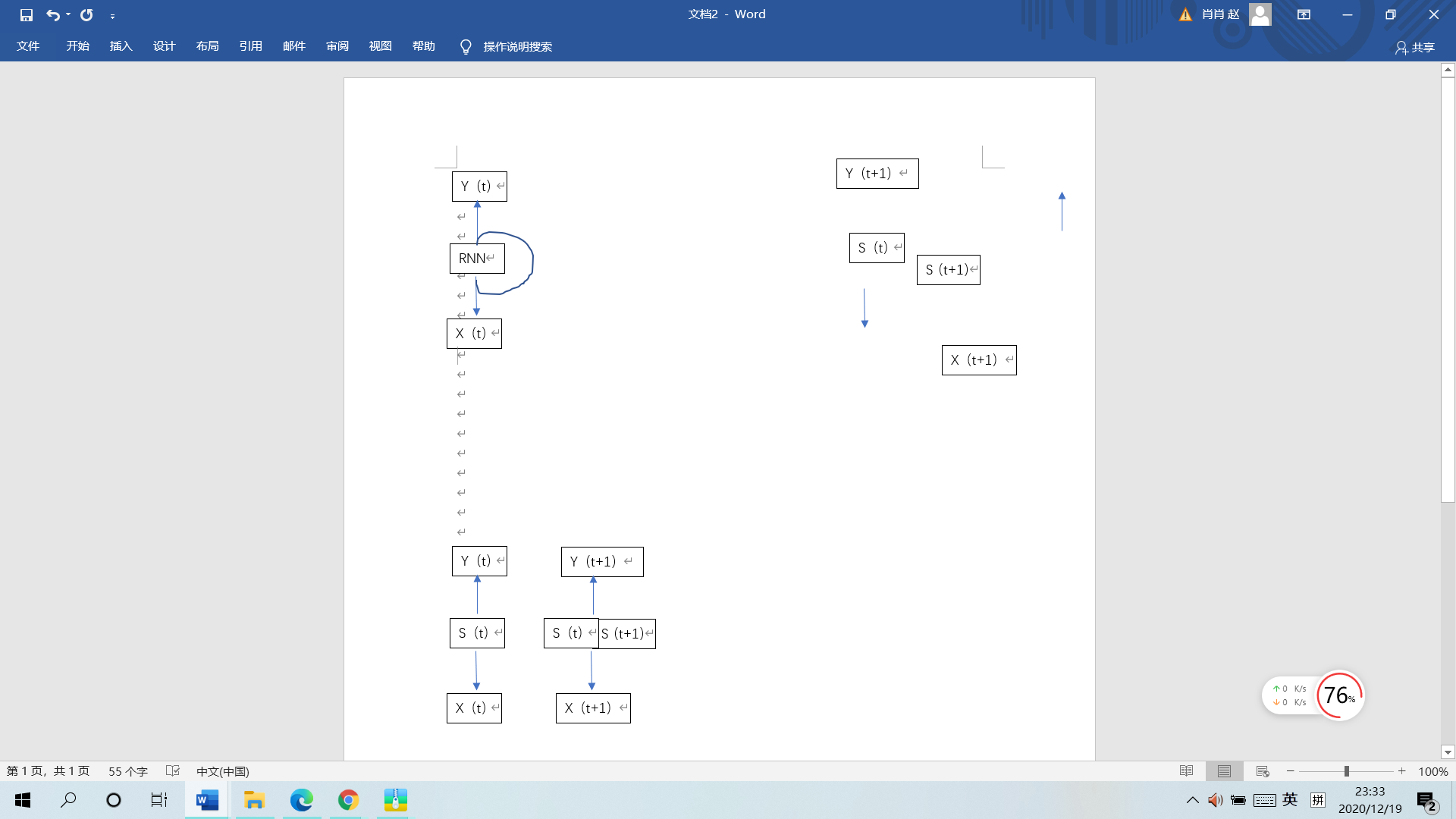
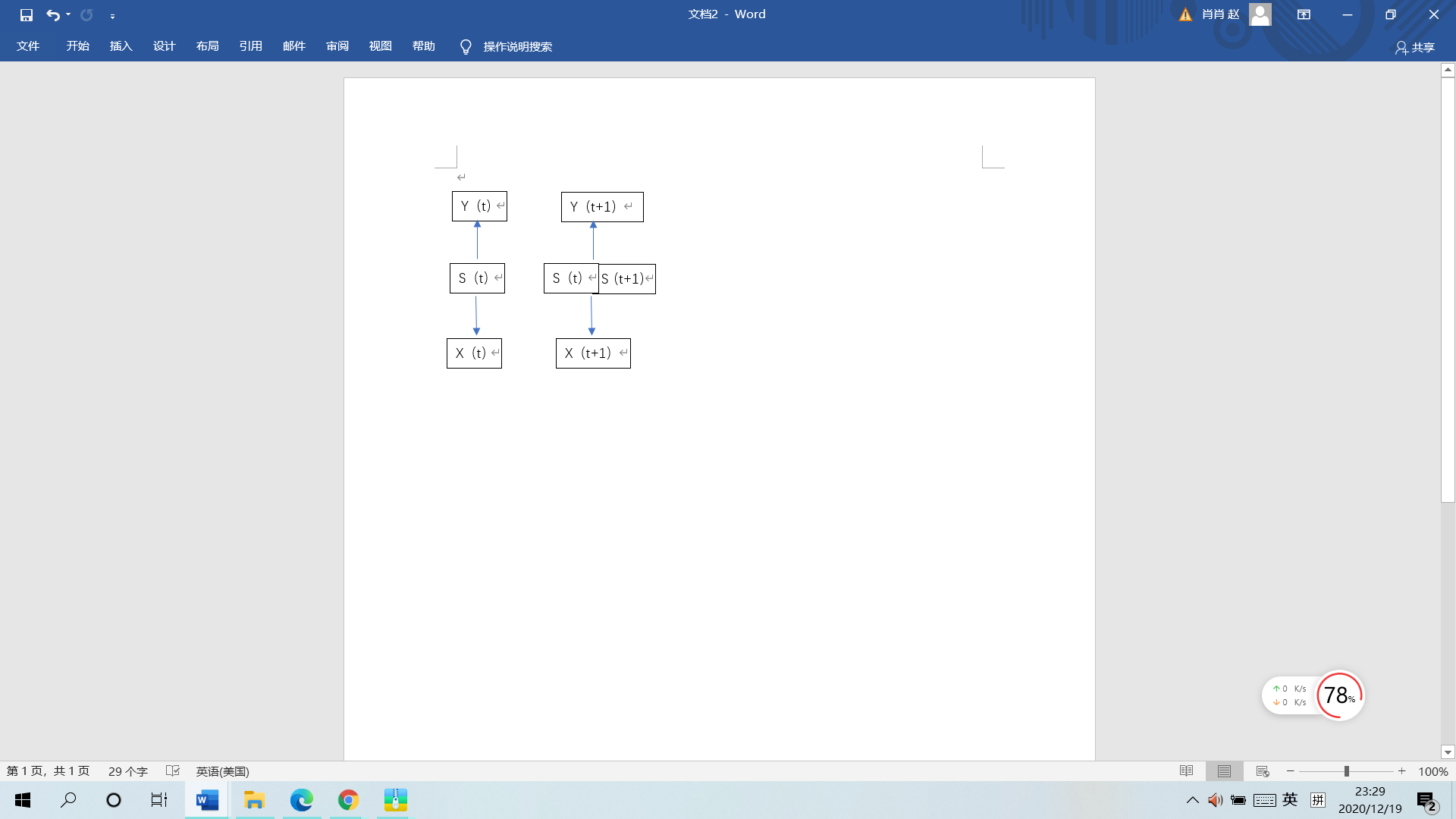
## 1.3关于我的理解

### 1.3.1我对循环神经网络的理解

对于预测顺序排列是一件非常重要的事情，我们可以预测下一个按照一定排列的字，但是打断顺序后我们就没有办法分析自己到底在说什么了。

就比如现在有一组序列数据Data０－３，我们在预测ｒｅｓｕｌｔｓ０的时候，我们会基于Data０去预测，同样在预测其他数据的时候，我们也都是基于其他单个的数据，每次使用的神经网络都是同一个NN，但如果这些数据是有关联顺序的，那么就像在厨房做菜一样，酱料A要比酱料B先放，否则味道就会不好，但普通的神经网络并不能让NN了解这些数据之间的关联。那么怎样才能让数据之间的关联也被NN加以分析呢？想想人类是怎样分析各种数的关联的，其实最基本的方法就是记住之前发生的事情。我们让神经网络也具备这种之前发生事情的能力。在分析Data０的时候，我们把分析结果存入记忆；然后在分析Data１的时候NN会产生新的记忆但是新的记忆和老的记忆没有关联，我们就把老的记忆调用过来一起分析。如果继续分析更多的数据，NN就会把之前的记忆都累积起来一起分析。

每次RNN运算完以后都会产生一个对于当前状态的描述state，这里用S（ｔ）来表示ｔ时刻的state，在X（t+１）时也会产生一个ｓ（ｔ＋１），但Y（t+１）是S（ｔ）加上ｓ（ｔ＋１）共同创造的（如图ａ），所以我们通常看到的RNN也就可以表示成为（图ｂ）的形式。



　　图ａ　　　　　　　　　　　　　　　　图ｂ

但RNN的形式不单单只有这样一种，它的结构形态很自由，如果用于分类的问题，比如一个人说了一句话，这句话带有的感情色彩是积极的还是消极的，那我们可以用只有最后一个时间点输出判断结果的RNN，又或者适用于图片描述的RNN，我们只需要一个X来代替输入的图片，然后生成对图片描述的一段话或者语言翻译的RNN：给出一段英文要求翻译成中文等。有了这些不同形式的RNN，RNN就变得越来越强大。目前存在许多有趣的RNN应用，比如说让RNN来描述照片，让RNN来写学术论文，让RNN来写程序脚本，让RNN来作曲。

### 1.3.2我对LSTM技术的理解：

LSTM是Long short-term memory的简称，中文又叫做长短期记忆，是当下最流行的RNN形式之一。RNN是在有序的数据上进行学习的，为了记住这些数据，RNN会像人一样产生对先前发生事件的记忆。但一般情况下RNN就像一位老人，还是比较健忘的。原因如下：例如有这样一个RNN，它的输入值是一句话“我今天要做红烧排骨，首先要准备排骨，然后．．．，最后美味的一道菜就完成了。”现在让RNN分析今天做了什么菜，但RNN分析之后给出的结果可能为辣子鸡这个答案，由于判断失误，RNN就要开始学习这个长序列X（输入的话）和红烧排骨的关系，而RNN所需要的关键数据红烧排骨却出现在了句子的开头。

接下来介绍RNN是怎样学习的。如下图所示

t1 t2 t3 …………………………………… tn

红烧排骨

红烧排骨这个信息元的记忆要经过长途跋涉才能到达最终的时间点，然后我们可以得到误差，然而在反向传递得到误差的时候，它会在每一步都乘以一个自己的参数W，如果这个W是个小于１的数，比如０.９，这个０.９不断乘以误差传到初始时间的时候，误差就会成为一个接近于０的数，所以对于初始时刻，误差就相当于消失了，我们把这个问题叫做梯度消失或者梯度弥散。反之，如果ｗ是个大于１的数，比如１.１，不断累成，乘到初始时间点的时候可能会变成一个无穷大的数，RNN被无穷大的数给撑死了，这种情况被称为梯度爆炸。这就是普通RNN没有办法回忆起久远记忆的原因。

LSTM就是为了解决RNN的这个问题而产生的，LSTM和普通RNN相比多出了三个控制器：输入控制、输出控制和忘记控制。

LSTM多了一个控制全局的记忆，如下图所示：

忘记

输出

输入

LSTM RNN

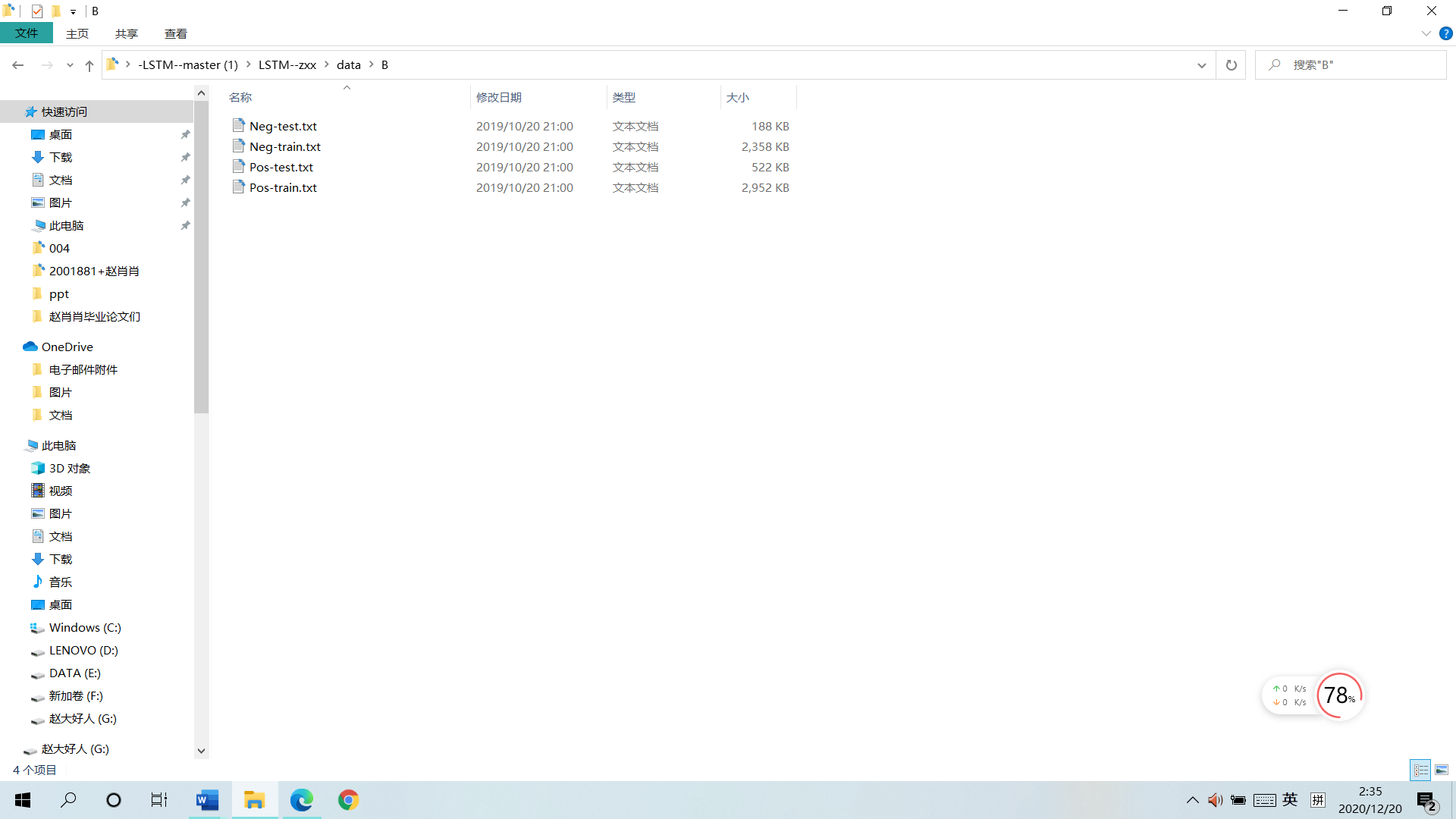
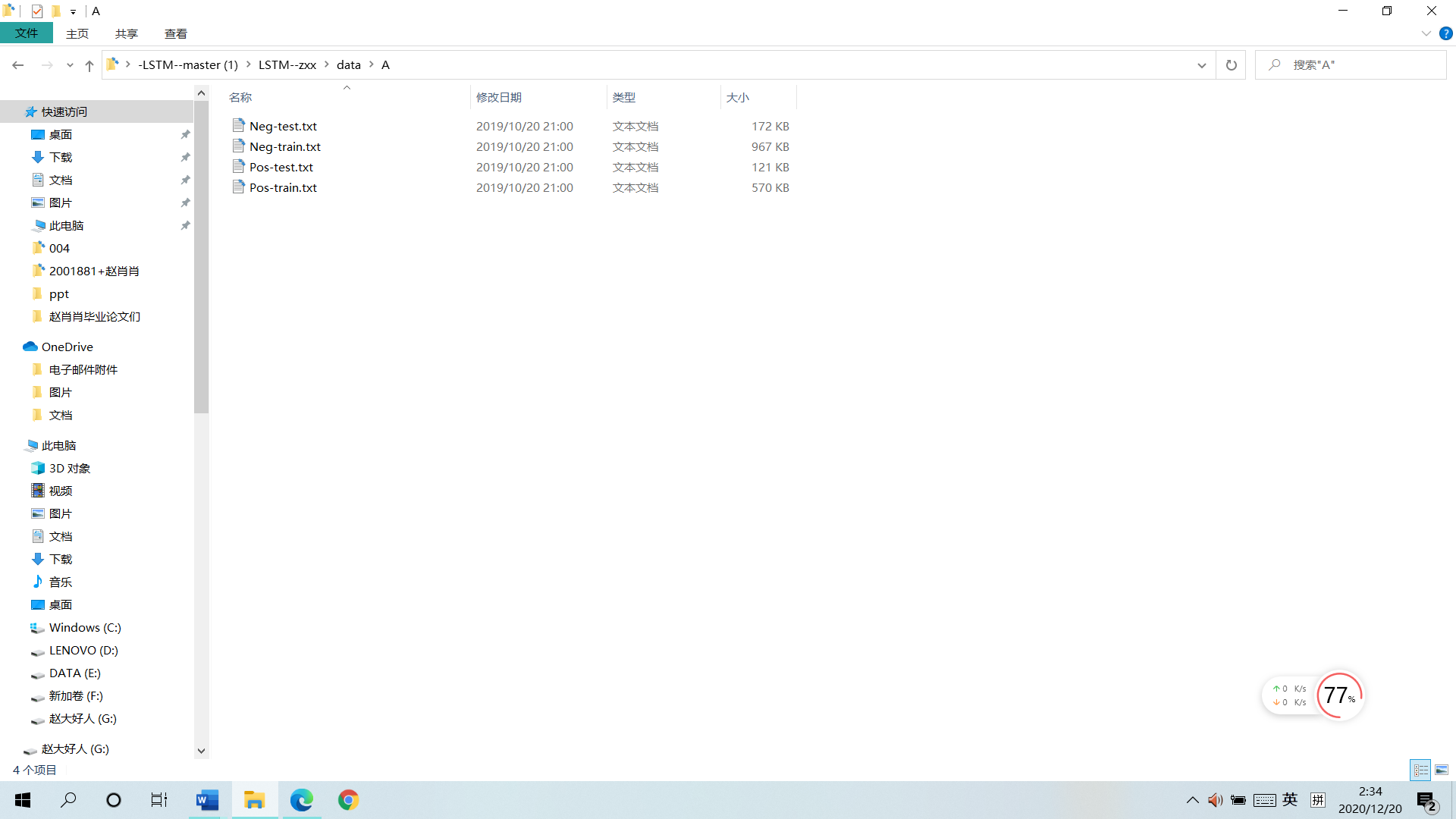
可以把粗线想象成为电影的主线剧情，而原本的RNN剧情就是分线剧情，三个控制器都是在原本的RNN体系上。首先看输入方面：如果分线剧情对于剧中的结果十分重要，输入控制就会将这个分线剧情按照重要程度写入这个剧情再进行分析。在忘记方面：如果此时的分线剧情改变了我们对之前剧情的想法，那么忘记控制就会将之前的某些主线剧情忘记，按比例替换成现在的新剧情。所以主线剧情的更新就取决于输入和忘记控制。在输出方面：输出控制会基于主线剧情和分线剧情判断要输出的到底是什么。基于这些控制机制，LSTM就像延缓记忆衰退的良药，可以带来更好的结果。

# 三、代码解读

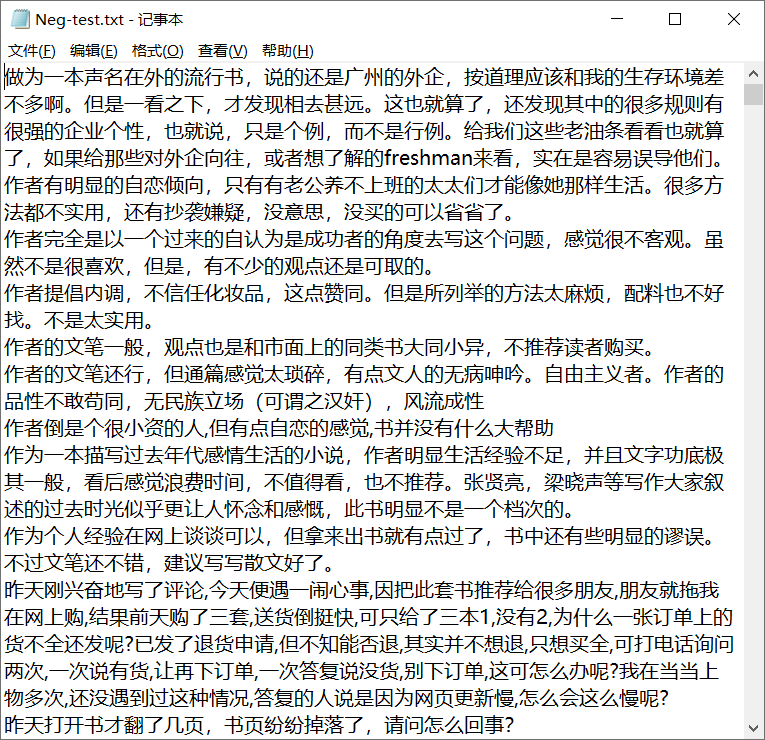
## 3.1数据预处理

### 3.1.1数据了解

共有两份数据文件A数据为酒店评价,B数据为图书评价;其中每份数据中包含四个txt文件,如下图所示:



在txt文件中,每一行就是一条评价,如下图所示



### 3.1.2代码分析

timeA=time.time()

word2vec\_path = 'word2vec/word2vec.model'

model=gensim.models.Word2Vec.load(word2vec\_path)

dimsh=model.vector\_size

MAX\_SIZE=25

stopWord = makeStopWord()

在这一块定义了一个时钟，计时A，在程序跑完后计时B，用来计算程序运行总时间。接下来是加载词向量。dimsh为词向量的维度，在这里，词向量的维度为200维。每条评价的长度是不同的。但是在向神经网络输入数据时，应该保持数据shape的一致性。所以定义了MAX\_SIZE，如果一条评价不够25个词，在将评价转成矩阵时，不够的位置用0补齐，超过25个词汇的评价，第25个词以后的词汇都抛弃掉。stopWord为停用词，有些词汇只是用来表示语气的，并不具有实意，这些词统称停用词，在训练时应该去除掉。

trainData, trainSteps, trainLabels = makeData('data/B/Pos-train.txt',

'data/B/Neg-train.txt')

print("In test data:")

testData, testSteps, testLabels = makeData('data/B/Pos-test.txt',

'data/B/Neg-test.txt')

trainLabels = np.array(trainLabels)

这块调用了makeData来制作训练集和测试集。makeData有两个参数，posPath和negPath，分别为积极评价的路径和消极评价的路径。然后它有三个返回值，为Data, Steps和Labels。

makeData函数内部是分为三步执行的：

1.获取词汇，返回的格式为[[word1,word2...],[word1,word2...],...],其中每个列表代表一个评价。

2.将每条评价都转换为对应的矩阵

3.将积极数据和消极数据混合在一起打乱，制作数据集

## 3.2用tensorflow搭建模型

tensorflow是基于计算图来运行的。我们需要先把计算图搭好，每次运行时，我们可以用run函数来向计算图内填充数据和获取一些参数。第一句是用来定义一个计算图，第二句是用来构建计算图。

graph = tf.Graph()

with graph.as\_default():

训练数据是通过占位符来传入的，每次传入一批（batch\_size大小）的数据。在这里需要传入的不止有训练数据，还有标签和每条评价对应的长度。

测试数据在计算图内是常量（constant）的形式。

需要注意的是，在这里，LSTM是用tensorflow自带的函数来实现的，比自己重复造轮子要方便多了。运算速度也要快很多。

要使用tensorflow自带的rnn模型，首先也是需要定义参数。在这里定义了lstm\_cell。需要定义它的输出维度，在这里传入参数num\_nodes，num\_nodes设的是128.

w1、b1、w2、b2为普通的矩阵，LSTM的输出值经过w1、b1、w2、b2的变换后就是最后的输出值。

tf\_train\_dataset = tf.placeholder(tf.float32,shape=(batch\_size,MAX\_SIZE,dimsh))

tf\_train\_steps = tf.placeholder(tf.int32,shape=(batch\_size))

tf\_train\_labels = tf.placeholder(tf.float32,shape=(batch\_size,output\_size))

tf\_test\_dataset = tf.constant(testData,tf.float32)

tf\_test\_steps = tf.constant(testSteps,tf.int32)

lstm\_cell = tf.nn.rnn\_cell.BasicLSTMCell(num\_units = num\_nodes,

state\_is\_tuple=True)

w1 = tf.Variable(tf.truncated\_normal([num\_nodes,num\_nodes // 2], stddev=0.1))

b1 = tf.Variable(tf.truncated\_normal([num\_nodes // 2], stddev=0.1))

w2 = tf.Variable(tf.truncated\_normal([num\_nodes // 2, 2], stddev=0.1))

b2 = tf.Variable(tf.truncated\_normal([2], stddev=0.1))

接下来是具体的计算过程。train\_logits为最后的输出值，它传入的参数有tf\_train\_dataset, tf\_trainsteps。tf.nn.dynamicrnn为tensorflow自带的rnn函数，是用来执行rnn计算过程的。它需要传入定义的lstm\_cell，sequencelength（即为每条评价对应的长度）,inputs(数据)。需要注意的是，rnn会在每一个步数产生一个输出值，在这里只取最后一个步数的输出值作为输出。last\_states的shape为(batch\_size,num\_nodes,2)。分别为对应的state和最后一个步数的输出值h。这里用last\_states[-1]来获取h。经过w1，w2的变换后得到logits作为最后的输出。

tf.nn.softmax\_cross\_entropy\_with\_logits函数直接完成了softmax和计算交叉熵的操作。然后用tf.reduce\_mean计算出了的loss。optimizer为优化器，优化目标为loss，学习率为0.01。

test\_prediction为在测试集上的预测。

# 附录一

import warnings

warnings.filterwarnings(action='ignore', category=UserWarning, module='gensim')

import gensim

from gensim.models import word2vec

import jieba

import tensorflow as tf

import numpy as np

import time

from random import randint

from random import shuffle

def makeStopWord():

with open('stopword.txt','r',encoding = 'utf-8') as f:

lines = f.readlines()

stopWord = []

for line in lines:

words = jieba.lcut(line,cut\_all = False)

for word in words:

stopWord.append(word)

return stopWord

def words2Array(lineList):

linesArray=[]

wordsArray=[]

steps = []

for line in lineList:

t = 0

p = 0

for i in range(MAX\_SIZE):

if i<len(line):

try:

wordsArray.append(model.wv.word\_vec(line[i]))

p = p + 1

except KeyError:

t=t+1

continue

else:

wordsArray.append(np.array([0.0]\*dimsh))

for i in range(t):

wordsArray.append(np.array([0.0]\*dimsh))

steps.append(p)

linesArray.append(wordsArray)

wordsArray = []

linesArray = np.array(linesArray)

steps = np.array(steps)

return linesArray, steps

def convert2Data(posArray, negArray, posStep, negStep):

randIt = []

data = []

steps = []

labels = []

for i in range(len(posArray)):

randIt.append([posArray[i], posStep[i], [1,0]])

for i in range(len(negArray)):

randIt.append([negArray[i], negStep[i], [0,1]])

shuffle(randIt)

for i in range(len(randIt)):

data.append(randIt[i][0])

steps.append(randIt[i][1])

labels.append(randIt[i][2])

data = np.array(data)

steps = np.array(steps)

return data, steps, labels

def getWords(file):

wordList = []

trans = []

lineList = []

with open(file,'r',encoding='utf-8') as f:

lines = f.readlines()

for line in lines:

trans = jieba.lcut(line.replace('\n',''), cut\_all = False)

for word in trans:

if word not in stopWord:

wordList.append(word)

lineList.append(wordList)

wordList = []

return lineList

def makeData(posPath,negPath):

#获取词汇，返回类型为[[word1,word2...],[word1,word2...],...]

pos = getWords(posPath)

print("The positive data's length is :",len(pos))

neg = getWords(negPath)

print("The negative data's length is :",len(neg))

#将评价数据转换为矩阵，返回类型为array

posArray, posSteps = words2Array(pos)

negArray, negSteps = words2Array(neg)

#将积极数据和消极数据混合在一起打乱，制作数据集

Data, Steps, Labels = convert2Data(posArray, negArray, posSteps, negSteps)

return Data, Steps, Labels

#----------------------------------------------

# Word60.model 60维

# word2vec.model 200维

timeA=time.time()

word2vec\_path = 'word2vec/word2vec.model'

model=gensim.models.Word2Vec.load(word2vec\_path)

dimsh=model.vector\_size

MAX\_SIZE=25

stopWord = makeStopWord()

print("In train data:")

trainData, trainSteps, trainLabels = makeData('data/B/Pos-train.txt',

'data/B/Neg-train.txt')

print("In test data:")

testData, testSteps, testLabels = makeData('data/B/Pos-test.txt',

'data/B/Neg-test.txt')

trainLabels = np.array(trainLabels)

del model

print("-"\*30)

print("The trainData's shape is:",trainData.shape)

print("The testData's shape is:",testData.shape)

print("The trainSteps's shape is:",trainSteps.shape)

print("The testSteps's shape is:",testSteps.shape)

print("The trainLabels's shape is:",trainLabels.shape)

print("The testLabels's shape is:",np.array(testLabels).shape)

num\_nodes = 128

batch\_size = 16

output\_size = 2

graph = tf.Graph()

with graph.as\_default():

tf\_train\_dataset = tf.placeholder(tf.float32,shape=(batch\_size,MAX\_SIZE,dimsh))

tf\_train\_steps = tf.placeholder(tf.int32,shape=(batch\_size))

tf\_train\_labels = tf.placeholder(tf.float32,shape=(batch\_size,output\_size))

tf\_test\_dataset = tf.constant(testData,tf.float32)

tf\_test\_steps = tf.constant(testSteps,tf.int32)

lstm\_cell = tf.nn.rnn\_cell.BasicLSTMCell(num\_units = num\_nodes,

state\_is\_tuple=True)

w1 = tf.Variable(tf.truncated\_normal([num\_nodes,num\_nodes // 2], stddev=0.1))

b1 = tf.Variable(tf.truncated\_normal([num\_nodes // 2], stddev=0.1))

w2 = tf.Variable(tf.truncated\_normal([num\_nodes // 2, 2], stddev=0.1))

b2 = tf.Variable(tf.truncated\_normal([2], stddev=0.1))

def model(dataset, steps):

outputs, last\_states = tf.nn.dynamic\_rnn(cell = lstm\_cell,

dtype = tf.float32,

sequence\_length = steps,

inputs = dataset)

hidden = last\_states[-1]

hidden = tf.matmul(hidden, w1) + b1

logits = tf.matmul(hidden, w2) + b2

return logits

train\_logits = model(tf\_train\_dataset, tf\_train\_steps)

loss = tf.reduce\_mean(

tf.nn.softmax\_cross\_entropy\_with\_logits(labels=tf\_train\_labels,

logits=train\_logits))

optimizer = tf.train.GradientDescentOptimizer(0.01).minimize(loss)

test\_prediction = tf.nn.softmax(model(tf\_test\_dataset, tf\_test\_steps))

num\_steps = 20001

summary\_frequency = 500

with tf.Session(graph = graph) as session:

tf.global\_variables\_initializer().run()

print('Initialized')

mean\_loss = 0

for step in range(num\_steps):

offset = (step \* batch\_size) % (len(trainLabels)-batch\_size)

feed\_dict={tf\_train\_dataset:trainData[offset:offset + batch\_size],

tf\_train\_labels:trainLabels[offset:offset + batch\_size],

tf\_train\_steps:trainSteps[offset:offset + batch\_size]}

\_, l = session.run([optimizer,loss],

feed\_dict = feed\_dict)

mean\_loss += l

if step >0 and step % summary\_frequency == 0:

mean\_loss = mean\_loss / summary\_frequency

print("The step is: %d"%(step))

print("In train data,the loss is:%.4f"%(mean\_loss))

mean\_loss = 0

acrc = 0

prediction = session.run(test\_prediction)

for i in range(len(prediction)):

if prediction[i][testLabels[i].index(1)] > 0.5:

acrc = acrc + 1

print("In test data,the accuracy is:%.2f%%"%((acrc/len(testLabels))\*100))

#####################################

timeB=time.time()

print("time cost:",int(timeB-timeA))