

# 实验报告

---

语言模型与新闻内容预测

17341046 郭梓煜

- 实验报告
  - 环境配置
  - 实验要求
  - 实验过程
    - 新闻数据爬取过程
    - 对爬取材料预处理过程
    - 建立模型及训练过程
      - n-gram 模型
      - LSTM模型
    - 预测过程及结果示例
      - 预测过程
      - 结果示例
  - 总结与思考
    - 遇到的困难及解决方法
    - 心得体会

## 环境配置

- python3.6
- vscode
- requests , beautifulsoup (用于爬虫)
- jieba, thulac 第三方库 (用于分词)
- TensorFlow, Keras 第三方库 (用于LSTM模型实现)
- coding: utf-8
- 以上第三方库的环境配置可通过`pip install xxx`轻松地配置完成

## 实验要求

1. 从国内主流新闻网站（如腾讯、新浪、网易等）的科技频道抓取新闻内容。
  - 要求新闻语言为中文，发布日期为2019年1月1日以后。
  - 数量至少为1000条。
2. 对新闻数据进行预处理（分句、分词、去噪）。
  - 完成建立词表的工作。
3. 使用预处理后的新闻数据训练两个语言模型。
  - 其一为简单的n-gram语言模型（n取2或者3）。
  - 其二为基于RNN/LSTM/GRU的语言模型。
  - 词典大小、网络和训练的超参数等可以自己指定。

## 实验过程

新闻数据爬取过程

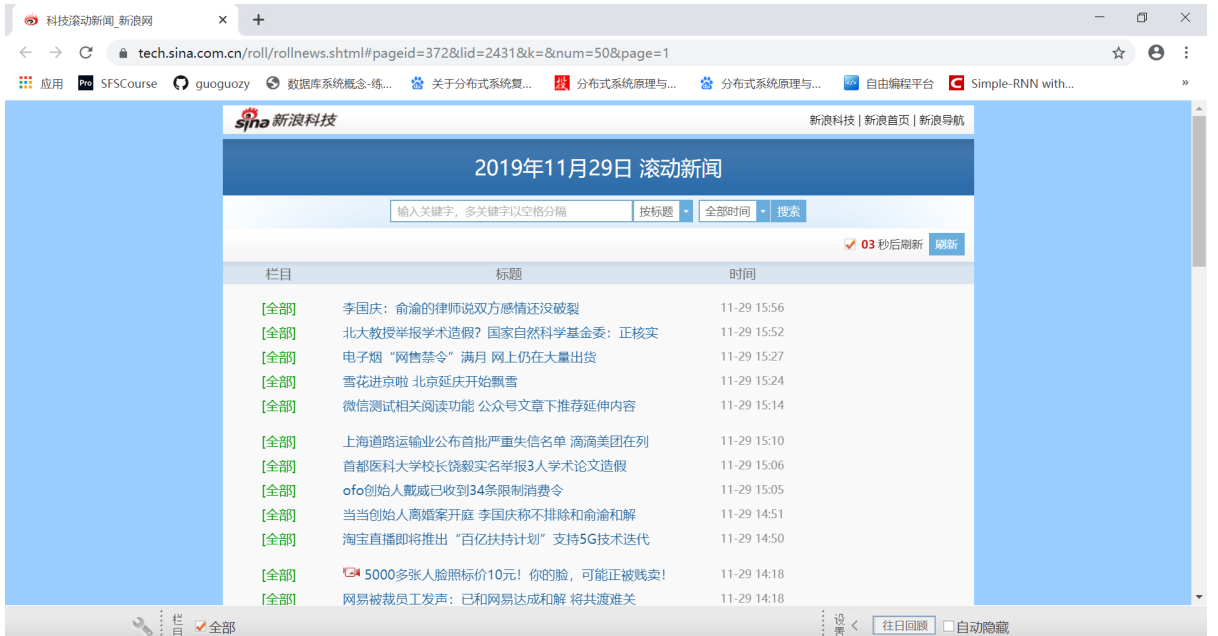
• 新闻网站的选择

- 由于腾讯新闻的网站需要通过动态爬取，模拟人拉动页面，才能刷新新闻。需要通过模拟滑轮滚动事件才能获取到足够的新闻txt，而且可能会出现拉取失败，获取不到新的新闻的现象而导致数据量不足。
- 另外，通过查看网页的具体信息(F12)可知(如下图)，要获取其中的新闻链接也比较困难，所以放弃了腾讯新闻的爬取。



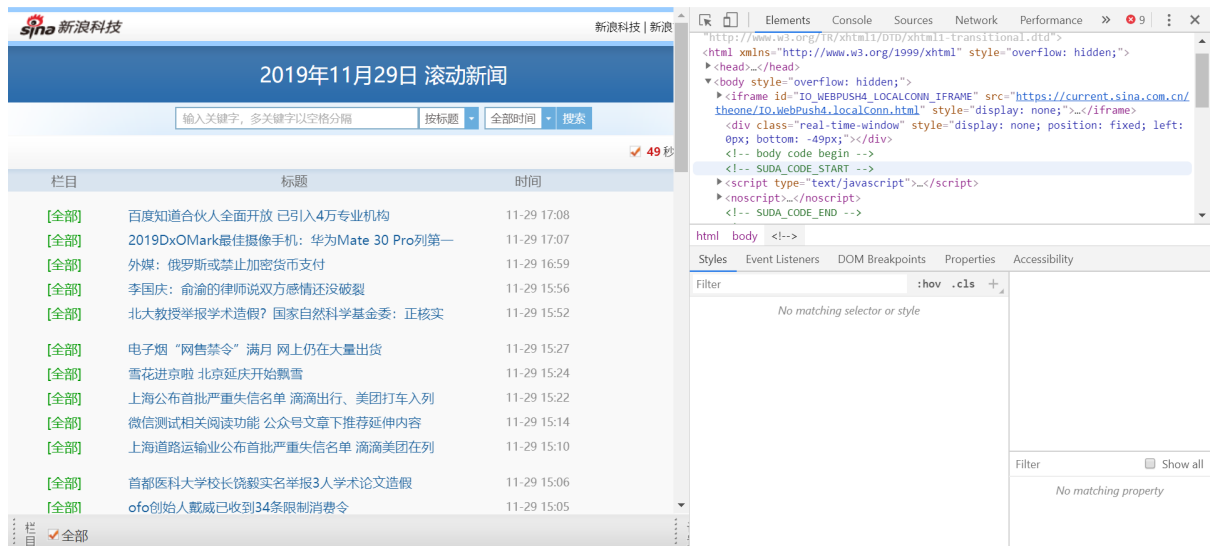
- 综合考察了几个主流新闻科技网站，最终选择了新浪新闻，爬取的难度适中，新闻也会实时更新，数据量也足够。新浪科技新闻为滚动实时更新的新闻网站，对爬取的实现难度要求较高，不像网易新闻属于固定的链接，采用定时更新，虽然爬取容易，但是新闻不够“新”。

爬取的新浪新闻网页如下：



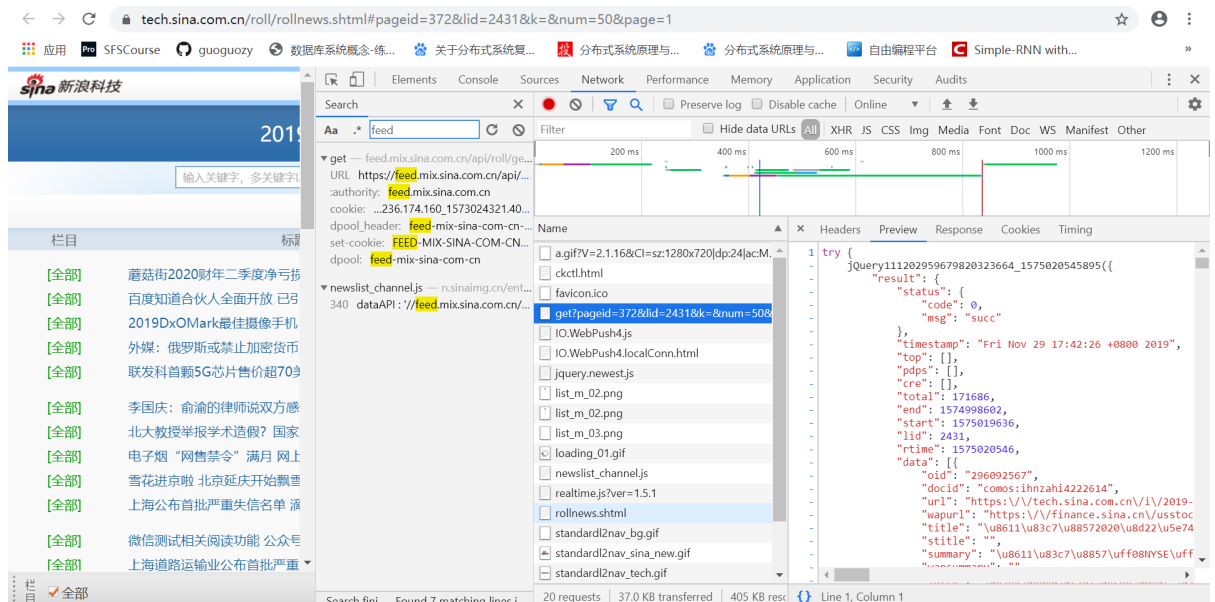
• 新浪新闻的爬取

## ○ 首先查看网页



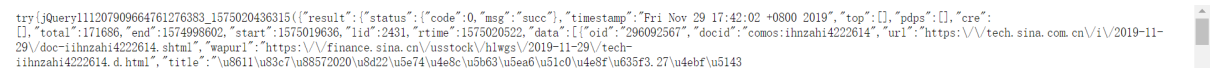
发现不能直接从框架上获取到新闻的链接，如果单纯地获取该网页的子链接，只会得到框架上的一些链接，这将是毫无意义的。

查看该网页的网络状态，发现该网页是通过动态地接受网络中发送过来的包来实现实时更新的。如下图：



查看get到的URL，双击点开后可以看到诸如下面的链接：

有许多的链接这里展示一个，我们要获取其中的URL；



图中的URL的内容即网页上滚动的新闻链接。

## ○ 编写代码获取链接(get\_news.py)

- 只需要对足够数量的page接受网络的包，解析出其中的链接即可。
- 由于对这方面之前并没有接触过，所以可以借鉴网上的一些代码，如CSDN\_blog来进行实现。
- 从新浪新闻的科技新闻网页不同page的链接可以看出只需改动最后的page=？，即可实现对不同page的访问。



这也方便了代码的实现。

- 获取链接代码如下：  
调用request等库函数来实现

```
def get_page_link():
    """
    Function: 使用request库获取链接
    Return: 新闻链接的list
    """
    page = 1
    pagelinks = []
    while page <= 180: # 设置爬取网页数量
        init_url = 'https://feed.mix.sina.com.cn/api/roll/get?
        pageid=372&lid=2431&k=&num=50&page=' + str(page)
        request = urllib.request.Request(init_url)
        response = urllib.request.urlopen(request)
        data = response.read()
        data = data.decode('gbk') # 设置解码方式
        reg_str = r'"url": "(.*)"'
        pattern = re.compile(reg_str, re.DOTALL)
        items = re.findall(pattern, data)
        for item in items:
            pagelinks.append(item)
        page += 1
    return pagelinks
```

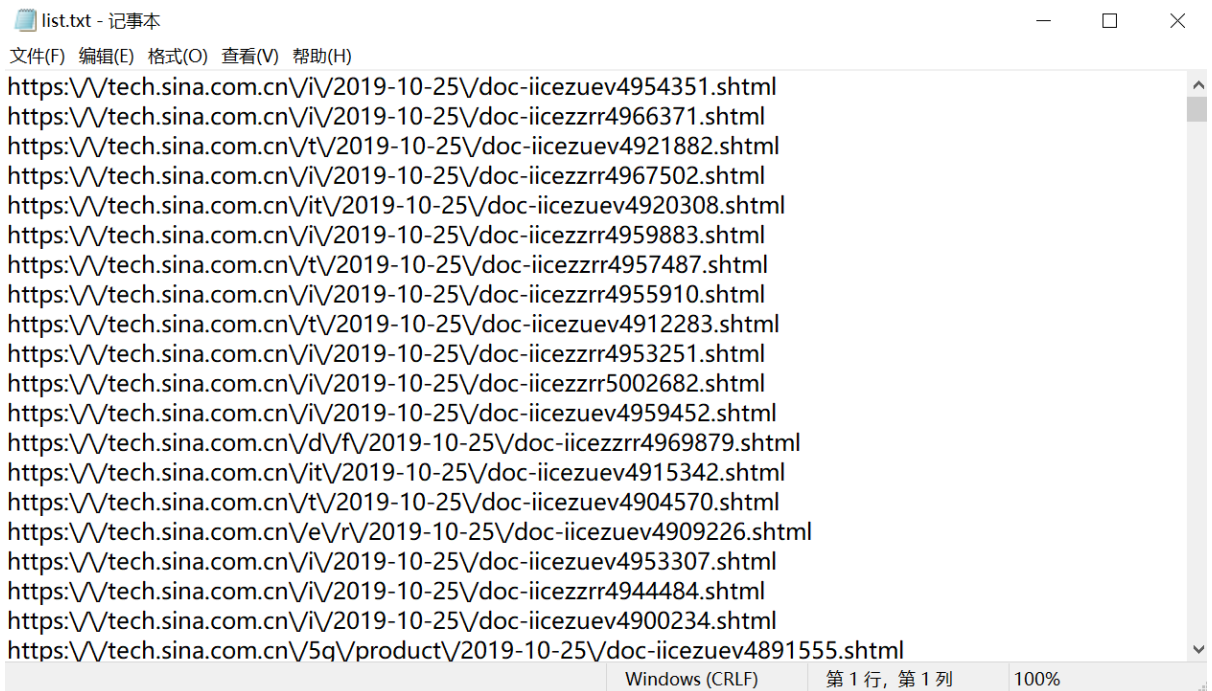
- 去除重复的链接并记录  
代码如下：

```
def del_same_link(pagelinks):
    """
    Function: 去除重复链接
    Return: 包含不重复链接的list
    """
    unrepect_url = []
```

```
for l in pagelinks:
    if l not in unrepect_url:
        unrepect_url.append(l)
return unrepect_url
```

```
def writelist(list):
    ...
    Function: 记录可用不重复链接
    Return: None
    ...
    f = open("D:\\code\\py\\NLP\\hw2\\list.txt", "w")
    for row in list:
        f.write(row+'\n')
    f.close()
```

记录的list.txt内容大致如下:



```
list.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
https://tech.sina.com.cn/i/2019-10-25/doc-iicezuev4954351.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4966371.shtml
https://tech.sina.com.cn/t/2019-10-25/doc-iicezuev4921882.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4967502.shtml
https://tech.sina.com.cn/it/2019-10-25/doc-iicezuev4920308.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4959883.shtml
https://tech.sina.com.cn/t/2019-10-25/doc-iicezzrr4957487.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4955910.shtml
https://tech.sina.com.cn/t/2019-10-25/doc-iicezuev4912283.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4953251.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr5002682.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezuev4959452.shtml
https://tech.sina.com.cn/d/f/2019-10-25/doc-iicezzrr4969879.shtml
https://tech.sina.com.cn/it/2019-10-25/doc-iicezuev4915342.shtml
https://tech.sina.com.cn/t/2019-10-25/doc-iicezuev4904570.shtml
https://tech.sina.com.cn/e/r/2019-10-25/doc-iicezuev4909226.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezuev4953307.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezzrr4944484.shtml
https://tech.sina.com.cn/i/2019-10-25/doc-iicezuev4900234.shtml
https://tech.sina.com.cn/5q/product/2019-10-25/doc-iicezuev4891555.shtml
Windows (CRLF) 第 1 行, 第 1 列 100%
```

#### o 通过链接提取新闻内容

观察链接页面的内容，查看新闻内容的标签，使用find匹配并将新闻内容提取出来，并将内容记入文本。

```
def get_news_content(list, n, num):
    ...
    Function: 获取链接的新闻内容，并写入txt文件
    Return: True or False
    ...
    print(n)
    url = str(list[n].replace('\\', ''))

    req = requests.get(url)
```

```

req.encoding = 'utf-8'
textreq = BeautifulSoup(req.text, features='lxml')

article = textreq.find('div', class_="article", id="artibody")
if not article:
    print("error\n")
    return False
else:
    body = article.get_text()
    index = body.find('(function')
    if index > 0:
        body = body[:index]
    if len(body) > 300 and '{' not in body:
        body = body.strip()
        fp = open('E:\\code\\py\\NLP\\hw2\\data\\' +
                  str(num+1) + '.txt', 'w', encoding='utf-8')
        fp.write(body)
        fp.close()
    else:
        return False
    return True

```

同时对文本的质量进行挑选，如果文本长度小于300将舍弃这一文本。  
如上的代码实现即：

```

body = article.get_text()
index = body.find('(function')
if index > 0:
    body = body[:index]
if len(body) > 300 :
    body = body.strip()
...

```

## 对爬取材料预处理过程

- 文本预处理过程包括分句、分词、去噪等过程，同时还有建立词表。(clean.py)
  - 分句
    - 分句需要根据中文句子结束的标志来判断，这就值得深思了。
      - 句子结束的标志应该有：[! ? ...。 ; ]
- 事实上，在分句的时候还考虑过一种情况，那就是“”对分句的影响。比方说，xxx说：“...”  
这样一来，”才是句子结束的标志，而”并不是所有的句子的结束标志。这个问题困扰了我很久。后来仔细一想并没有必要，依旧以。来分句即可。其他的符号在分好句去掉即可，对分句结果并没有影响。
- 至于分句的代码实现，并不复杂，如下：  
(截取自clean.py中函数的部分)

```

for file in range(start, end):
    fread = open('hw2/data/' + str(file+1)+'.txt',
                'r',encoding='utf-8')
    for line in fread.readlines():
        line = re.split(r'[! ? ... ; ]', line)
    ...

```

- 分词

- 分词库仍然采用jieba进行分词  
比较了thulac以及北大研发的pkuseg的新闻分词模型的分词效果之后，仍然觉得jieba分词的效果更加符合预期效果。
- 分词的代码实现也不难，只需对分出来的每一句调用jieba分词，如下：  
(截取自clean.py中函数的部分)

```

for file in range(start, end):
    fread = open('hw2/data/' + str(file+1)+'.txt',
                'r',encoding='utf-8')
    ...
    for line in fread.readlines():
        line = re.split(r'[! ? ... ; ]', line)
        fwrite = open('hw2/word/' + str(file+1)+'.txt',
                    'a',encoding='utf-8')
        for sentences in line:
            word_list = list(jieba.cut(sentences.replace(' ', '')))
        ...

```

- 其中仍然会出现一些新鲜词汇，jieba分不出来，可使用add\_word加入词库，再进行分词

```

jieba.add_word('云计算')
jieba.add_word('区块链')
...

```

- 去噪

- 去噪应该是预处理过程中最繁杂的，毕竟文本会出现什么字符(各种奇奇怪怪的字符你根本想象不到)
- 一开始，我将文本中的字符以string.punctuation加上zhon.punctuation进行排除，只要出现在该集合中的字符都去掉。(zhon为汉字包，可通过from zhon.hanzi import punctuation调用)
- 尽管如此，处理后的文本仍然不是很符合理想，所以我想到了使用编码范围的限制来去噪，最终去噪结果还算较好。
- 去掉标点符号代码如下：

```

def remove_punctuation(line):
    # 只保留中文、大小写字母和阿拉伯数字

```



- 去掉阿拉伯数字及英文:

- 另外地还有一些特殊的圆角，半角字符等等，独立的编写代码遍历1000个txt收集字符得到结果如下：

○ 有了上面的铺垫，再进行去噪就好写很多了，代码如下：

8 / 25



```
if word_list:
    fwrite.write(' '.join(word_list)+'\n')
```

- 建立词表

- 我认为一些停用词以及一些长度只有1的词对之后的工作没有任何意义，所以在建立词表时都将其去掉了
- 停用词表来源于PPT中GitHub下载所得，加载停用词表，代码如下：

```
def get_stwlist():
    stwlist = [x.strip()
               for x in open('hw2/stopwords/哈工大停用词表.txt',
                           encoding='utf-8').readlines()]
    # stwlist += [x.strip()
    #             for x in open('hw2/stopwords/百度停用词表.txt', encoding='utf-8').readlines()]
    stwlist += [x.strip() for x in open('hw2/stopwords/中文停用词表.txt',
                                       encoding='utf-8').readlines()]
    stwlist += [x.strip() for x in open('hw2/stopwords/四川大学机器智能实验室
    停用词库.txt',
                                       encoding='utf-8').readlines()]

    return stwlist
```

- 接下来建立词表就不难了，只需将词加以判断加入All\_words这一dict中，key代表词，value代表出现次数。

```
def create_words_table(start, end):
    All_words = {}
    stwlist = get_stwlist()
    for file in range(start, end):
        fread = open('hw2/word/' + str(file+1) + '.txt',
                    'r', encoding='utf-8')
        for line in fread.readlines():
            if line:
                line = line.rstrip('\n')
                line = re.split(r'\s', line)
                for i in line:
                    if i not in stwlist
                    and remove_punctuation(remove_number_english(i)) !=
                    ..
                    and remove_punctuation(remove_number_english(i))
                    not in
                    zhon_punctuation+en_punctuation and len(i) > 1:
                        # 此处为判断是否为停用词以及是否还有奇怪的字符以及是否长度大于1
                        if i in All_words: #是否已存在于词表中
                            All_words[i] = All_words[i]+1
                        else:
                            All_words[i] = 1
```

```
fsum = open('hw2/words_table.txt', 'w', encoding='utf-8')
number = 1
words_list = sorted(All_words.items(), key=lambda d: d[1],
reverse=False)
for item in words_list:
    fsm.write(str(number)+' '+str(item[0])+' '+str(item[1])+'\n')
    number = number+1
```

## 建立模型及训练过程

### n-gram 模型

- n-gram模型较为简单，是基于统计原理建立的模型。原理即下图：

$$p(w_k|w_{k-n+1}^{k-1}) = \frac{p(w_{k-n+1}^k)}{p(w_{k-n+1}^{k-1})} \approx \frac{\text{count}(w_{k-n+1}^k)}{\text{count}(w_{k-n+1}^{k-1})}$$

也就是说，只需要找到[MASK]前面n-1个词，然后在所有数据中匹配，将这n-1个词后一个词统计下来，哪个词出现频率高那么答案就可能是哪个。

- 这需要将所有数据的停止词去掉。我认为，不管是question.txt还是1000个新闻数据集，停止词对预测都没有帮助，所以需要将其去除。获取停用词表可直接调用clean.py实现的即可(import clean)。
  - 去除代码如下：
    - 去除question.txt中的停止词
    - (代码截自n-gram.py部分)

```
fread = open('hw2/questions.txt', 'r')
match_word = {}
question = {}
number = 1
stwlist = clean.get_stwlist()
# 读question.txt，记录前n-1个词及问题
for sentences in fread.readlines():
    word_list = list(jieba.cut(sentences))
    # 去掉标点和空格
    for i in range(0, len(word_list)):
        word_list[i] = clean.remove_punctuation(word_list[i])
    word_list = [x for x in word_list if x != '']
    # 去掉停止词
    num_word = 0
    while num_word < len(word_list):
        if word_list[num_word] in stwlist:
            del word_list[num_word]
        else:
            num_word += 1
    ...
```

去除数据集中的停止词:

(代码摘自clean.py部分)

```
def clean_word(start,end):
    # 将分好词的1000个txt去掉停止词
    stwlist=get_stwlist()
    for file in range(start,end):
        fread = open('hw2/word/' + str(file+1)+'.txt',
            'r',encoding='utf-8')
        fwrite = open('hw2/word_cleaned/' + str(file+1)+'.txt',
            'w',encoding='utf-8')
        for line in fread.readlines():
            line=line[:len(line)-1].split(' ')
            i=0
            while i <len(line):
                if line[i] in stwlist:
                    del line[i]
                else:
                    i+=1
            if line:
                fwrite.write(' '.join(line))
                fwrite.write('\n')
```

- 分别对2-gram, 3-gram记录前面的不同n个word (词)
  - 使用index方法可以快速地定位到[MASK]
  - 使用字符串的拼接, 将结果拼接为字符串, 以空格为间, 这样一来只需要使用in即可快速在数据集中找到我们要的结果了, 大大地减少了代码量。

```
if n == 2:
    match_word[word_list[word_list.index('MASK')-1]+' ' ] = {}
    question[number] = word_list[word_list.index('MASK')-1]+' '
elif n == 3:
    match_word[word_list[word_list.index(
        'MASK')-2]+' '+word_list[word_list.index('MASK')-1]+' ' ] = {}
    question[number] = word_list[word_list.index(
        'MASK')-2]+' '+word_list[word_list.index('MASK')-1]+' '
```

- 遍历数据集, 收集预测结果
  - 将预测结果记录在match\_word这一dict中, key为前n-1个单词的字符串, value为另一个dict, 记录所有预测结果, key为预测结果, value为出现次数。

```
for front_n_word in match_word.keys():
    for i in range(0, 1000):
        fread = open('hw2/word_cleaned/'+str(i+1)+'.txt', 'r')
        for lines in fread.readlines():
            lines = lines.rstrip('\n')
            if front_n_word in lines:
```

```

        answer_word = lines[lines.index(
            front_n_word)+len(front_n_word):].split(' ')[0]
        if answer_word in match_word[front_n_word]:
            match_word[front_n_word][answer_word] += 1
        else:
            match_word[front_n_word][answer_word] = 1

```

- 将预测结果排序，即对字典排序  
直接调用sorted排序即可

```

match_word[front_n_word] = sorted(
    match_word[front_n_word].items(), key=lambda d: d[1],
    reverse=True)

```

- 将预测结果写入预测文件
  - 输出前五个概率最高的预测结果

```

fwrite = open('hw2/prediction.txt', 'w')
for i in range(1, 101):
    num = 0
    for item in match_word[question[i]]:
        num += 1
        if num == 6:
            break
    fwrite.write(str(item[0])+' ')
    fwrite.write('\n')

```

- 计算预测正确率
  - 以第一个预测结果为标准与答案比较

```

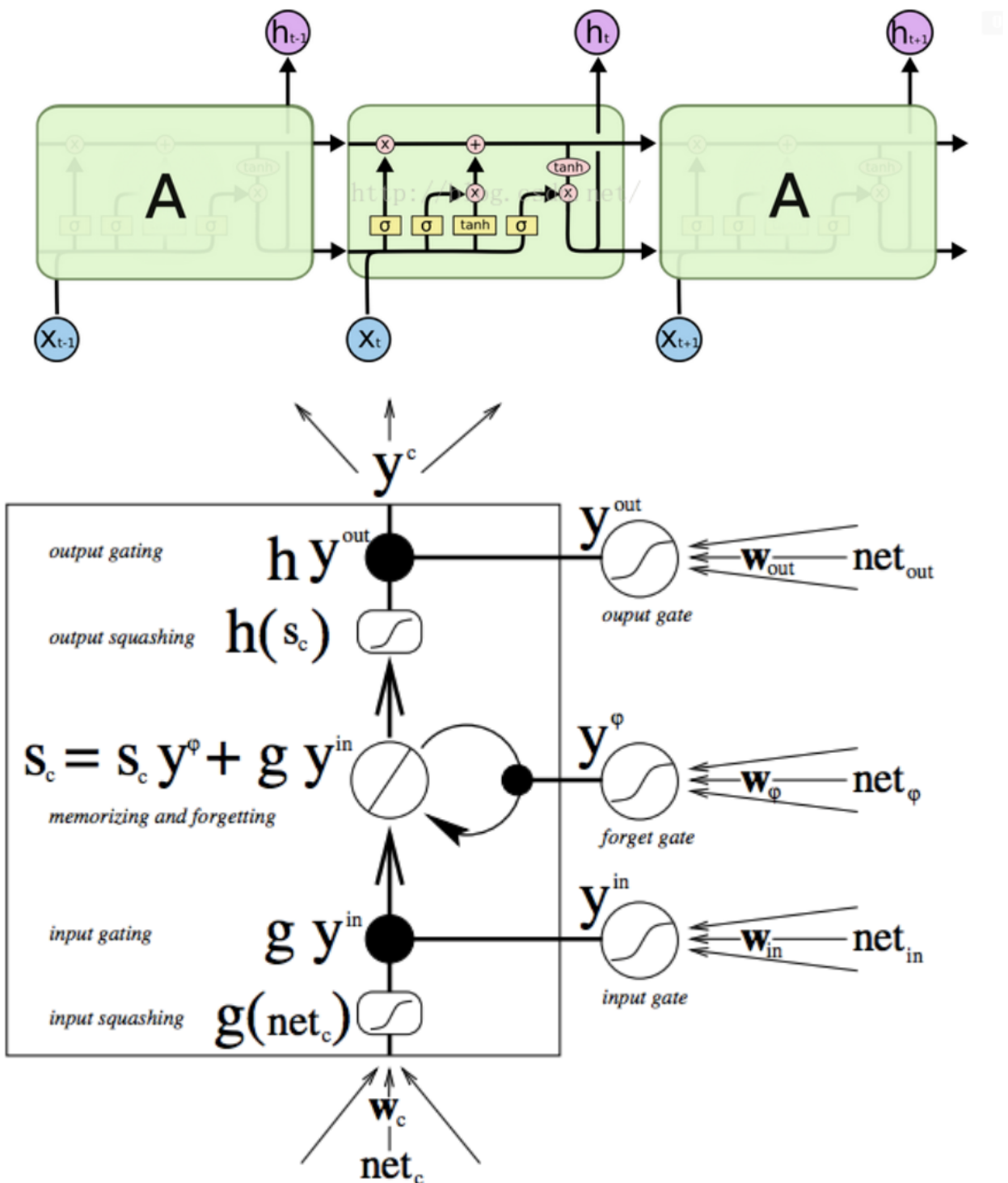
def accurate():
    bingo = 0
    f1 = open('hw2/answer.txt')
    f2 = open('hw2/prediction_2.txt')
    answer_list = []
    pre_list = []
    for i in f1.readlines():
        i = i.rstrip('\n')
        answer_list.append(i)
    for i in f2.readlines():
        i = i.rstrip('\n')
        pre_list.append(i)
    for i in range(0, 100):
        if answer_list[i] == pre_list[i].split(' ')[0]:
            # if answer_list[i] in pre_list[i].split(' '):
                bingo += 1
    print(bingo/100)

```

- 使用3-gram模型预测经常出现结果为空的情况，这时可以回退，将空的情况使用2-gram获得结果，由于n-gram.py中已经实现了n\_gram函数，可以处理2-gram和3-gram只需将2, 3结果合并即可，这里就不多加展示。

## LSTM模型

- 长短期记忆模型（long-short term memory）是一种特殊的RNN模型，是为了解决RNN模型梯度弥散的问题而提出的；在传统的RNN中，训练算法使用的是BPTT，当时间比较长时，需要回传的残差会指数下降，导致网络权重更新缓慢，无法体现出RNN的长期记忆的效果，因此需要一个存储单元来存储记忆，因此LSTM模型被提出；



- 数据的处理(lstm\_data\_process.py)  
对训练以及预测的数据进行处理便于后面模型构建的使用。
  - 创建一个新词表包含训练集和测试集的所有词汇
    - 同样使用dict记录, key为词, value为序号
    - MASK用序号-1特殊标注
    - 为避免重复执行, 将结果dict写入txt方便以后使用

```
def create_new_table():
    All_words = {}
    word_num = 1

    # 将1000个txt的词加入词表
    for file in range(0, 1000):
        fread = open('hw2/word_cleaned/'+str(file+1)+'.txt', encoding='utf-8')
        for line in fread.readlines():
            line = line[:len(line)-1].split(' ')
            for i in line:
                if i not in All_words.keys():
                    All_words[i] = word_num
                    word_num += 1

    # 将question中的词加入词表
    fread = open('hw2/questions.txt', encoding='utf-8')
    for sentences in fread.readlines():
        word_list = list(jieba.cut(sentences))
        # 去掉标点和空格
        for i in range(0, len(word_list)):
            word_list[i] = clean.remove_punctuation(word_list[i])
        word_list = [x for x in word_list if x != '']
        # 去掉停止词
        for i in range(0, len(word_list)):
            if word_list[i] in clean.get_stwlist():
                word_list[i] = ''
        word_list = [x for x in word_list if x != '']
        for i in word_list:
            if i == 'MASK':
                All_words[i] = -1
            if i not in All_words.keys():
                All_words[i] = word_num
                word_num += 1

    # 将answer中词加入词表
    fread = open('hw2/answer.txt', encoding='utf-8')
    for line in fread.readlines():
        if str(line.rstrip('\n')) not in All_words.keys():
            All_words[str(line.rstrip('\n'))] = word_num
            word_num += 1

    fwrite = open('hw2/All_words_dict.txt', 'w', encoding='utf-8')
```

```
fwrite.write(str(All_words))
```

- 词表大致如下:

All\_words\_dict.txt ×

🔍 📄 🗑️ ...

hw2 > All\_words\_dict.txt

```
1 {'新浪': 1, '科技': 2, '讯': 3, '北京': 4, '时间': 5, '10': 6, '月': 7, '25': 8, '日': 9, '晚间': 10, '
消息': 11, '网易': 12, '有道': 13, '今日': 14, '正式': 15, '登陆': 16, '纽交所': 17, '股票代码': 18,
'DAO': 19, '开盘价': 20, '13.75': 21, '美元': 22, '发行价': 23, '17': 24, '下跌': 25, '19%': 26, '凌晨':
27, '美国': 28, '证券交易': 29, '委员会': 30, 'SEC': 31, '提交': 32, 'IPO': 33, '招股书': 34, '摩根斯
坦利': 35, '瑞士': 36, '信贷': 37, '花旗': 38, '中': 39, '金': 40, '汇丰': 41, '担任': 42, '公开': 43, '
招股': 44, '联席': 45, '承销商': 46, '享有': 47, '总计': 48, '84': 49, '万股': 50, 'ADS': 51, '超额':
52, '配售': 53, '权': 54, '更新版': 55, '显示': 56, '计划': 57, '发行': 58, '560': 59, '股份': 60, '存托':
61, '证券': 62, '定价': 63, '计算': 64, '募资': 65, '规模': 66, '9520': 67, '万美元': 68, '市值': 69,
'达到': 70, '19': 71, '亿美元': 72, '同步进行': 73, '私募': 74, '最大': 75, '机构': 76, '股东': 77,
'Orbis': 78, '基金': 79, '承诺': 80, '道': 81, '购买': 82, '总额': 83, '1.25': 84, 'A': 85, '类': 86, '
普通股': 87, '融资': 88, '2.202': 89, '今年': 90, '上半年': 91, '营收': 92, '5.49': 93, '亿元': 94, '人
民币': 95, '去年同期': 96, '3.28': 97, '同比': 98, '增长': 99, '67.67%': 100, '净亏损': 101, '1.68':
102, '8275.1': 103, '万元': 104, '扩大': 105, '102.89%': 106, '2018': 107, '在线': 108, '课程': 109, '
代表': 110, '智能': 111, '学习业务': 112, '成为': 113, '第一': 114, '大营': 115, '收': 116, '来源': 117,
'2019': 118, '学习': 119, '业务收入': 120, '3.1': 121, '58.1%': 122, '占': 123, '总营收': 124,
'57.4%': 125, '广告': 126, '收入': 127, '2.3': 128, '亿': 129, '82.5%': 130, '本次': 131, '完成': 132,
'前': 133, '丁磊': 134, '实益': 135, '持有': 136, '29': 137, '751': 138, '158': 139, '股': 140, '持股':
141, '比例': 142, '30.1%': 143, '周枫': 144, '20': 145, '341': 146, '200': 147, '20.6%': 148, '吴迎
晖': 149, '840': 150, '000': 151, '1.9%': 152, '方面': 153, '网易公司': 154, '65': 155, '387': 156,
'160': 157, '66.2%': 158, 'CEO': 159, '全资': 160, '拥有': 161, '开曼群岛': 162, 'PengKeHoldingsInc':
163, '泽宇': 164, '信息': 165, '名单': 166, '涉及': 167, '包括': 168, '万达': 169, '电影': 170, '三夫':
171, '户外': 172, '蓝色': 173, '光标': 174, '公司': 175, '董事': 176, '机场': 177, '艺人': 178, '代拍':
179, '乱象': 180, '登上': 181, '微博热': 182, '搜': 183, '一时间': 184, '个人信息': 185, '买卖': 186, '推
上': 187, '风口浪尖': 188, '新': 189, '京报': 190, '记者': 191, '注意': 192, '明星': 193, '上市公司':
```

- 之后需要使用新词表，只需读取All\_words\_dict.txt即可
  - 使用eval可以很方便的读取

```
def get_all_words_dict():
    f = open('hw2/All_words_dict.txt', 'r', encoding='utf-8')
    return dict(eval(f.read()))
```

- 将所有数据（1000个txt，question.txt，answer.txt）都转换为序号表示

- 将所有数据使用All\_words\_dict转换为对应的序号

```
def transfer_word_to_num():
    All_words = get_all_words_dict()
    for file in range(0, 1000):
        fread = open('hw2/word_cleaned/'+str(file+1)+'.txt',
            encoding='utf-8')
        fwrite = open('hw2/word_train/'+str(file+1) +
            '.txt', 'w', encoding='utf-8')
        for line in fread.readlines():
            line = line[:len(line)-1].split(' ')
            temp = []
            for i in line:
                temp.append(All_words[i])
            if len(temp) >= 5:
                fwrite.write(str(temp)+'\n')

        fread = open('hw2/questions.txt', encoding='utf-8')
        fwrite = open('hw2/questions_num.txt', 'w', encoding='utf-8')
        for sentences in fread.readlines():
```



```

word_list = list(jieba.cut(sentences))
# 去掉标点和空格
for i in range(0, len(word_list)):
    word_list[i] = clean.remove_punctuation(word_list[i])
word_list = [x for x in word_list if x != '']
# 去掉停止词
for i in range(0, len(word_list)):
    if word_list[i] in clean.get_stwlist():
        word_list[i] = ''
word_list = [x for x in word_list if x != '']
temp = []
for i in word_list:
    temp.append(All_words[i])
fwrite.write(str(temp)+'\n')

fread = open('hw2/answer.txt', encoding='utf-8')
fwrite = open('hw2/answer_num.txt', 'w', encoding='utf-8')
fwrite.write('\n'.join([str(All_words[str(line.rstrip('\n'))])
                        for line in fread.readlines()])))

```

- 转换后结果大致如下：(以questions.txt转换结果为例)

questions\_num.txt × answer\_num.txt

```

hw2 > questions_num.txt
1 [1028, 26545, 7673, 316, 1019, 1204, 2589, 1964, -1, 9027, 1964, 289]
2 [522, 404, 13638, 1024, -1, 607, 113, 367, 5029]
3 [3341, 476, 22355, 314, -1, 45250, 3643, 5046, 476, 1679, 12170, 1034]
4 [11932, 2941, 4607, -1, 22867, 20586, 45251, 8642, 45252, 30714, 45253]
5 [13160, 271, 3508, 8986, 1085, 432, -1, 2626]
6 [505, 1292, 16620, 11227, 728, 505, 5078, 6013, 4597, 316, 252, -1, 4597, 253, 10888, 35571, 4597, 2712]
7 [11965, 3781, 3401, 6542, 925, 3781, 522, 9042, 470, -1, 3656, 76]
8 [1578, 617, 5593, 1210, 993, 12013, 5593, 4577, -1, 11500, 1679, 461, 1971, 1035, 1962, 12013, 5593, 12]
9 [4669, 3642, 4220, 1512, -1, 17076, 1385, 41976, 865, 5758]
10 [7, 145, 9, 1256, 1257, 1724, 3436, 25911, 27397, 530, 3410, 236, 434, 461, 1034, 1177, 211, 2287, 877,
11 [7756, 3247, 430, 745, 4016, -1, 1137, 745, 1045, 45254, 4525, 17439, 3914, 1667, 11463, 815, 11463, 74]
12 [45257, 1651, 576, 1590, 510, 495, 1022, 12279, -1, 9411, 14693, 821, 1034, 1157, 2210, 1211, 2760]
13 [45258, 404, 877, 1033, 1960, 377, 1034, 2905, -1, 773, 43284, 6407, 1966, 709, 993]
14 [461, 1032, 17283, 10792, 1373, 7924, 127, 1300, -1, 3382, 450, 905, 3394, 12522, 43489]
15 [1077, 3629, 2691, 316, 5014, 522, 195, 236, 5014, 2539, 3629, 2901, 3629, 2901, 39, 272, -1, 3629, 506]
16 [2489, 45259, 1158, 1746, 503, 1088, 811, 2042, -1, 3914, 17592, 702]
17 [658, 456, 4771, 22783, 4842, 36712, 31476, 8492, 29467, 7110, 232, 1014, 34483, 7332, 2047, 1655, -1,
18 [3416, 4043, 26449, -1, 6031, 752, 617, 4196, 1649, 1079, 14771, 16368, 26450, 165, 2838, 7501, 10150]
19 [4392, 1644, 610, 1880, 2287, 3464, 275, 5231, 877, -1, 610, 1898, 2287, 450, 948, 621, 3509]
20 [9128, 658, 4739, 24313, 45260, 45261, 57, 11572, 555, 66, 1381, 31113, -1, 15267, 2510, 271, 2466, 175]
21 [877, 22305, 13512, 495, 41245, 367, -1]
22 [2557, 6217, 1560, 44743, 10904, -1, 4632, 9536, 279, 2660, 12380, 10904]
23 [189, 6519, 434, 38070, 45262, -1, 27430, 173, 7768, 189, 38070, 189, 2883]
24 [15155, 153, 236, 1662, 15926, 4895, 9672, 492, 5988, 5484, 15155, 4021, 1732, 23734, 5484, 1439, 6674,

```

- 调用tensorflow来实现LSTM模型的构建

- 一开始我想使用TensorFlow-GPU版本，因为GPU运行速度确实比较快，后来import失败，查看原因原来是电脑并非英伟达显卡，所以使用不了，只能退而求其次使用TensorFlow-CPU版本，效果一样只是运行较慢。
- 由于之前并没有对TensorFlow方面的编程经验，所以在实现时，难免到网上阅读相关[官方文档](#)，再进而根据自己的理解加以实现。
- 代码实现
  - 加载train数据

```
def load_train():
    pre_path = 'hw2/word_train/'
    txt_list = os.listdir(pre_path)
    txt_list = txt_list[0:1000]
    train_x = []
    train_y = []
    result1 = []
    result2 = []
    for txt in txt_list:
        with open(pre_path+txt, 'r', encoding='utf-8') as f:
            lines = f.readlines()
            for line in lines:
                line = line.strip()
                line = eval(line)
                seg_x, seg_y = seq_seg(line)
                result1.extend(seg_x[0])
                result2.extend(seg_x[1])
                train_y.extend(seg_y)
    train_x = [result1, result2]
    return train_x, train_y
```

其中的seq\_seg函数即对每一行数据进行切割，将每一个词或者说是每一个序号的前五个和后五个序号切割出来，不足的补为0。

```
def seq_seg(seq):
    global time_steps
    seg_x = []
    seg_y = []
    size = time_steps//2
    result1 = []
    result2 = []
    for i in range(1, len(seq)):
        iter_x1 = []
        iter_x2 = []
        iter_y = seq[i]
        if i >= size:
            iter_x1 = seq[i-size:i]
        else:
            iter_x1 = [0 for _ in range(size-i)]+seq[0:i]
        if i+size < len(seq):
            iter_x2 = seq[i+1:i+size+1]
        else:
            iter_x2 = seq[i+1:len(seq)]+[0 for _ in range(size-len(seq)+i+1)]
        result1.append(iter_x1)
        result2.append(iter_x2)
        seg_y.append(iter_y)
    seg_x = [result1, result2]
    return seg_x, seg_y
```

## ■ 加载test数据

```
def load_test(length):
    test_x = []
    test_y = []
    dic = {}
    with open('hw2/questions_num.txt', 'r', encoding='utf-8') as f:
        lines = f.readlines()
        test_x = [eval(s.strip()) for s in lines]
    with open('hw2/answer_num.txt', 'r', encoding='utf-8') as f:
        lines = f.readlines()
        test_y = [eval(s.strip()) for s in lines]
    with open('hw2/All_words_dict.txt', 'r', encoding='utf-8') as f:
        dic = eval(f.read())
    result1 = []
    result2 = []
    for seq in test_x:
        index = seq.index(-1)
        left = []
        right = []
        if index >= length:
            left = seq[index-length:index]
        else:
            left = [0 for _ in range(length-index)]+seq[0:index]
        if index+length < len(seq):
            right = seq[index+1:index+length+1]
        else:
            right = seq[index+1:]+[0 for _ in range(length-len(seq)+index+1)]

        result1.append(left)
        result2.append(right)
    test_x = [result1, result2]
    return test_x, test_y, dic
```

## ■ 构造模型

- 参考[官方文档](#)利用tf.keras系列库函数可以快速地实现模型的建立。
- 在 tf.keras.layers 中有很多层，下面是一些通用的构造函数的参数：
  - activation: 设置层的激活函数。此参数由内置函数的名称或可调用对象指定。默认情况下，不应用任何激活。
  - kernel\_initializer 和 bias\_initializer: 设置层创建时，权重和偏差的初始化方法。指定方法：名称 或 可调用对象。默认为"Glorot uniform" initializer。
  - kernel\_regularizer 和 bias\_regularizer: 设置层的权重、偏差的正则化方法。比如：L1 或 L2 正则。默认为空。
- 而我创建的模型是双输入的前五个和后五个。代码如下：

```

def create_model(dic_len, axis=64):
    global time_steps
    Input1 = tf.keras.layers.Input(shape=(time_steps//2),
dtype='float32')
    Input2 = tf.keras.layers.Input(shape=(time_steps//2),
dtype='float32')
    encode = tf.keras.layers.Embedding(
        dic_len, axis, input_length=time_steps//2, trainable=True)
    v1 = encode(Input1)
    v2 = encode(Input2)

    l1 = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(128, return_sequences=True))(v1)
    l2 = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(128, return_sequences=True))(v2)

    l = tf.keras.layers.Concatenate(axis=1)([l1, l2])
    l = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(128, return_sequences=False))(l)
    l = tf.keras.layers.Dropout(0.5)(l)

    d = tf.keras.layers.Dense(512, activation='relu')(l)
    d = tf.keras.layers.BatchNormalization()(d)
    output = tf.keras.layers.Dense(dic_len, activation='softmax')
(d)

    model = tf.keras.Model(inputs=[Input1, Input2],
outputs=output)

    return model

```

#### ■ 配置训练过程

- 构建模型后，通过调用`compile`方法配置其训练过程：

`tf.keras.Model.compile`有三个重要参数：

- **optimizer**：训练过程的优化方法。此参数通过 `tf.train` 模块的优化方法的实例来指定，比如：`AdamOptimizer`，`RMSPropOptimizer`，`GradientDescentOptimizer`。
- **loss**：训练过程中使用的损失函数（通过最小化损失函数来训练模型）。常见的选择包括：均方误差（`mse`），`categorical_crossentropy`和`binary_crossentropy`。损失函数由名称或通过从`tf.keras.losses`模块传递可调用对象来指定。
- **metrics**：训练过程中，监测的指标（Used to monitor training）。指定方法：名称 或 可调用对象 from the `tf.keras.metrics` 模块。

- `tf.keras.Model.fit` 有三个重要的参数：

- **epochs**：训练多少轮。（小批量）
- **batch\_size**：当传递NumPy数据时，模型将数据分成较小的批次，并在训练期间迭代这些批次。此整数指定每个批次的大小。请注意，如果样本总数不能被批量大小整除，则最后一批可能会更小。

- **validation\_data**: 在对模型进行原型设计时, 您希望轻松监控其在某些验证数据上的性能。传递这个参数 - 输入和标签的元组 - 允许模型在每个epoch的末尾以传递数据的推理模式显示损失和度量。
- 回调是传递给模型的对象, 用于在训练期间自定义和扩展其行为。
  - **tf.keras.callbacks.TensorBoard**: 使用TensorBoard 监测模型的行为。代码如下:

```

network = create_model(size+1, 64)
# 设置优化器, 损失函数, 评估函数
# callback, 可视化
s = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime(time.time()))
s = '.\\logs\\LV1'

callback = [tf.keras.callbacks.TensorBoard(
    log_dir=s,
    write_graph=True,
    write_images=True
)]
network.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['acc'])
print(network.summary())

# 训练
try:
    network.fit([train_x[0], train_x[1]], train_y,
                batch_size=128, callbacks=callback,
                epochs=10, verbose=1, shuffle=True,
                validation_data=([test_x[0], test_x[1]], test_y))
except Exception as e:
    print(e)
network.save_weights('lastmodel_weights.h5')

# 测试
test(network, test_x)

```

- 结果测试
  - 将模型预测得到的结果和answer.txt输出在同一个predict中便于比较

```

def test(network, test_x):
    dic = get_dic()
    out = network([test_x[0], test_x[1]])
    out = tf.argmax(out, axis=1)
    out = out.numpy().astype('int32')

    result1 = []
    for i in range(len(out)):
        if dic.get(out[i]):

```

```
        result1.append(dic[out[i]])

print(str(result1))
with open('hw2/answer.txt', 'r', encoding='utf-8') as f:
    labels = f.readlines()
    labels = [line.strip() for line in labels]
print(labels)
with open('predict.txt', 'w', encoding='utf-8') as f:
    for i in range(len(result1)):
        f.write(result1[i]+' '+labels[i]+'\\n')
```

- 训练中指标的变化
  - 以下给出一个epoch为10时的训练指标样例:

Using TensorFlow backend.  
2019-11-30 07:30:44.356840: I tensorflow/core/platform/cpu\_feature\_guard.cc:142] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2  
Model: "model"

Layer (type) Connected to	Output Shape	Param #
=====		
input_1 (InputLayer)	[(None, 5)]	0
input_2 (InputLayer)	[(None, 5)]	0
embedding (Embedding) input_1[0][0]  input_2[0][0]	(None, 5, 64)	2901184
bidirectional (Bidirectional) embedding[0][0]	(None, 5, 256)	197632
bidirectional_1 (Bidirectional) embedding[1][0]	(None, 5, 256)	197632
concatenate (Concatenate) bidirectional[0][0]  bidirectional_1[0][0]	(None, 10, 256)	0

bidirectional_2 (Bidirectional)	(None, 256)	394240
concatenate[0][0]		

---

dropout (Dropout)	(None, 256)	0
bidirectional_2[0][0]		

---

dense (Dense)	(None, 512)	131584
dropout[0][0]		

---

batch_normalization (BatchNormaliza	(None, 512)	2048
dense[0][0]		

---

dense_1 (Dense)	(None, 45331)	23254803
batch_normalization[0][0]		

---

=====

Total params: 27,079,123

Trainable params: 27,078,099

Non-trainable params: 1,024

---

None

Train on 513267 samples, validate on 100 samples

Epoch 1/10

513267/513267 [=====] - 2056s 4ms/sample - loss: 8.0120 - acc: 0.0587 - val\_loss: 7.1672 - val\_acc: 0.0900

Epoch 2/10

513267/513267 [=====] - 1909s 4ms/sample - loss: 6.6945 - acc: 0.1219 - val\_loss: 6.9419 - val\_acc: 0.0900

Epoch 3/10

513267/513267 [=====] - 1910s 4ms/sample - loss: 5.8621 - acc: 0.1729 - val\_loss: 6.7225 - val\_acc: 0.1500

Epoch 4/10

513267/513267 [=====] - 1909s 4ms/sample - loss: 5.1958 - acc: 0.2157 - val\_loss: 7.0147 - val\_acc: 0.1300

Epoch 5/10

513267/513267 [=====] - 1907s 4ms/sample - loss: 4.6552 - acc: 0.2560 - val\_loss: 7.1522 - val\_acc: 0.1400

Epoch 6/10

513267/513267 [=====] - 1912s 4ms/sample - loss: 4.1998 - acc: 0.2943 - val\_loss: 7.4063 - val\_acc: 0.1200

Epoch 7/10

513267/513267 [=====] - 2314s 5ms/sample - loss: 3.8223 - acc: 0.3326 - val\_loss: 7.7795 - val\_acc: 0.1100 0.1100

Epoch 8/10

513267/513267 [=====] - 2062s 4ms/sample - loss: 3.5139 - acc: 0.3669 - val\_loss: 8.2443 - val\_acc: 0.0900

Epoch 9/10

513267/513267 [=====] - 1909s 4ms/sample - loss: 3.2574 - acc: 0.3982 - val\_loss: 8.7362 - val\_acc: 0.1300



```
Epoch 10/10
513267/513267 [=====] - 1973s 4ms/sample -
loss: 3.0485 - acc: 0.4249 - val_loss: 8.9637 - val_acc: 0.1300
```

可以看到随着epoch增加，loss降低，metrics增加，10轮以后loss下降率逐渐降低。

- 改变epoch的次数从1到10到30，预测结果的正确率从小不断变高，loss也不断下降，由于时间关系，（TensorFlow-CPU速度实在是太慢了）难以进一步地进行下去，但预测这样下去正确率应该会变高，并到达一个界限。

## 预测过程及结果示例

### 预测过程

- n-gram预测过程只需运行n-gram.py并修改其n参数即可进行2-gram和3-gram的预测。
- LSTM模型的预测只需运行LSTM.py即可看到预测结果

### 结果示例

- 运行n-gram以及LSTM在本地文件夹将出现prediction\_?.txt，可调节输出文件名。若有行为空，比如3-gram存在许多结果为空的情况，由于同时也跑了2-gram就不将3-gram回退了，空的情况代表预测不出来，数据集中不存在一样的前n个词。

预测结果文件内容格式大致如下：（以2-gram为例）

```
prediction_2.txt ×
hw2 > prediction_2.txt
1 肿瘤 网络 死亡率 享有 死亡
2 到来 新 带来了 技术
3 采访 媒体 21 记者 北欧
4 智能 领域 挑战 体验 想象
5 缴税 App 能力 做 跑步
6 51 转给 火星 BM3D 二十多年
7 服务 发展 管理 提供 数字化
8 广阔 看好 有何 单细胞 点子
9 投资 人数 建设 一点点 5G
10 大型 主动 传统 地区 广告
11 筛查 做 中 生态圈 全身性
12 类 机器人
13 规模 预期 中 发展 中国
14 发展 经营 推进 运营 投入
15 网 新 指出 美 发现
16 生产 裁员 成本 建 量产
17 总经理 开 导航系统 发展 控制
18 加大 总理 克赖斯特彻奇 年内 媒体
19 厂商 市场 App 销量 销售
20 选择 团队
21 方式
22 一加 售价 搭载 打广告 正式
```

- 编写正确率测试代码（截自n-gram.py代码部分）

```
def accurate():
    bingo = 0
    f1 = open('hw2/answer.txt',encoding='utf-8')
    f2 = open('predict.txt',encoding='utf-8')
    #f2 = open('hw2/prediction_2.txt',encoding='utf-8')
    #f2 = open('hw2/prediction_3.txt',encoding='utf-8')
    answer_list = []
    pre_list = []
```

```

for i in f1.readlines():
    i = i.rstrip('\n')
    answer_list.append(i)
for i in f2.readlines():
    i = i.rstrip('\n')
    pre_list.append(i)
for i in range(0, 100):
    if answer_list[i] == pre_list[i].split(' ')[0]:
        # if answer_list[i] in pre_list[i].split(' '):
            print(i, answer_list[i])
            bingo += 1
print(bingo/100)

```

- 2-gram结果及正确率展示

```

PS E:\code\py\NLP> cd 'e:\code\py\NLP'; ${env:PYTHONIOENCODING}='UTF-8'; ${env:PYTHONUNBUFFERED}='1'; & 'C:\Users\lenovo\AppData\Local\Programs\Python\Python36\python.exe' 'c:\Users\lenovo\.vscode-insiders\extensions\ms-python.python-2019.11.50794\pythonFiles\ptvsd_launcher.py' '--default' '--client' '--host' 'localhost' '--port' '59915' 'e:\code\py\NLP\hw2\n-gram.py'
7 广阔
18 厂商
32 发展
40 成本
57 汽车
72 影响
81 套餐
90 合作
0.08

```

可见，2-gram预测结果与答案一致的项，以及2-gram预测正确率为8%。

- 3-gram结果及正确率展示

```

PS E:\code\py\NLP> cd 'e:\code\py\NLP'; ${env:PYTHONIOENCODING}='UTF-8'; ${env:PYTHONUNBUFFERED}='1'; & 'C:\Users\lenovo\AppData\Local\Programs\Python\Python36\python.exe' 'c:\Users\lenovo\.vscode-insiders\extensions\ms-python.python-2019.11.50794\pythonFiles\ptvsd_launcher.py' '--default' '--client' '--host' 'localhost' '--port' '59920' 'e:\code\py\NLP\hw2\n-gram.py'
14 学习
18 厂商
36 诽谤
41 泄露
69 发布
75 消费者
79 人工智能
81 套餐
90 合作
0.09

```

可见，3-gram预测结果与答案一致的项，以及3-gram预测正确率为9%。

- LSTM结果及正确率展示

```

PS E:\code\py\NLP> cd 'e:\code\py\NLP'; ${env:PYTHONIOENCODING}='UTF-8'; ${env:PYTHONUNBUFFERED}='1'; & 'C:\Users\lenovo\AppData\Local\Programs\Python\Python36\python.exe' 'c:\Users\lenovo\.vscode-insiders\extensions\ms-python.python-2019.11.50794\pythonFiles\ptvsd_launcher.py' '--default' '--client' '--host' 'localhost' '--port' '59848' 'e:\code\py\NLP\hw2\n-gram.py'
5 故宫
13 增长
14 学习
36 诽谤
41 泄露
57 汽车
65 销量
69 发布
79 人工智能
81 套餐
90 合作
92 人工智能
99 企业
0.13

```

可见，LSTM预测结果与答案一致的项，以及LSTM预测正确率为13%。

- 总的来说，预测性能：LSTM > 3-gram > 2-gram

## 总结与思考

遇到的困难及解决方法

## 1. 新知识的学习问题

这次的实验难度挑战确实较大，需要学习许多之前没有过的新知识。比如说，python爬虫，TensorFlow建立LSTM模型，训练等等。而这些东西往往直接上网谷歌难以解决问题，这就迫使我和以往学习不一样地去阅读官方文档，尽管官方文档较为晦涩，也没有具体的解析，是块难啃的硬骨头，但是还是考验了我的耐心，通过和同学交流以及共同学习，查阅相关文献，解决了新知识的难题。

## 2. 分句，去噪时的问题

分句时，由于如何判断句子结束的问题，我困扰了很久。在文本中甚至有一些以引号，或者三个，四个，五个点来作为句子结束标志。这可能是小编的失误编辑问题。另外还有一些特殊符号造成的问题。这些都引起了很大的问题。与一些同学讨论之后，发现他们都忽视了这一问题，但也没有好的办法。后来我想起了使用中文对应的编码范围来解决这一问题，通过中文的编码使用re库来过滤文本（详情可见clean.py中的remove\_xxx函数）成功地解决了问题。

## 3. TensorFlow使用问题

一开始经有过TensorFlow编程经验的同学介绍，TensorFlow-GPU的运行速度会快一些，可惜的是在import tensorflow的时候出现了问题，could not find cuda xxx，经查阅相关错误，发现是由于显卡并非英伟达的，而使用不了TensorFlow-GPU，只能使用TensorFlow-CPU了，导致后来训练模型的时候，一次epoch就要二三十分钟，大大限制训练次数。

## 4. python用法问题

这一次实验让我对python的一些函数应用地更加自如了。一些同学需要十几行的代码我可以一行实现，对字符串的切片以及index或eval的操作更加娴熟，所以说，在觉得这样实现较为复杂的时候，可以先谷歌一下看看有没有更方便的函数可以使用。

## 5. 数据问题

这一次实验后面的操作都是建立在前一步的基础上的，如果第一步收集的新闻数据质量太低可能会导致后期的实现以及预测的结果较为不理想。好在我在第一步时添加了关于新闻的质量过滤，防止爬取到的新闻过短甚至只有一句话或者存在垃圾。

## 心得体会

经过了历时将近一个月的实验，对自然语言处理的具体实现，python爬虫实现以及模型的构建和对文本的特殊处理都有深入的了解。尽管实验中多多少少存在一些不足诸如硬件上的限制等等，也是学到了许多，在下一次的作业会继续努力，更上一层楼。