



自然语言处理

期中大项目：文本预测

数据科学与计算机学院 17大数据与人工智能
17341015 陈鸿峥

目录

1	新闻内容抓取	2
1.1	提取新闻链接地址	2
1.2	下载新闻文本	4
1.3	新闻数据处理	4
1.4	实施细节	4
2	新闻数据预处理	5
3	模型训练	5
3.1	n-gram模型	6
3.1.1	去除停止词及加句子标记	6
3.1.2	生成n-gram列表	6
3.1.3	模型预测	6
3.2	LSTM模型	7
3.2.1	数据生成	7
3.2.2	模型搭建	9
3.2.3	模型训练	10
3.2.4	模型预测	12
3.2.5	其他实施细节	13
4	实验结果	13
4.1	n-gram模型	13
4.2	LSTM模型	14
4.3	综合比较	17
5	总结与思考	18

一、新闻内容抓取

新闻内容的抓取又可以分成以下的三个阶段：生成包含科技新闻页面的url列表，从url列表中下载对应的新闻文本，对原始新闻文本进行一些简单的处理。

源代码可见spider.py。下面将分别对这三个阶段进行说明。

1. 提取新闻链接地址

新闻链接主要从网易科技的以下三个主页面进行抓取：

- IT业界_网易科技频道：http://tech.163.com/special/it_2016/
第X个页面的地址为/it_2016_X， $X \in [1, 20]$ ，每个页面有20条新闻。

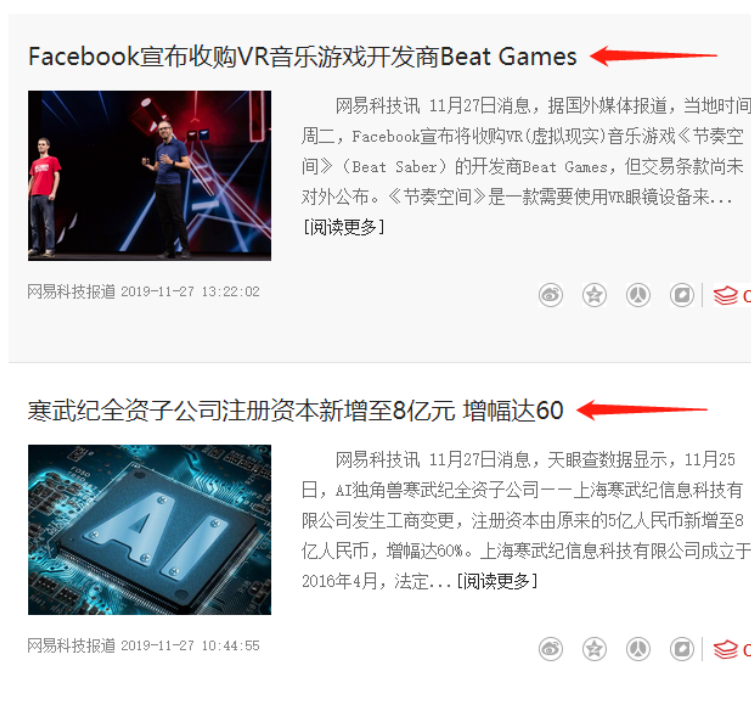


图 1: IT业界_网易科技频道

- 滚动_网易科技频道：<http://tech.163.com/special/gd2016/>
第X个页面的地址为/gd2016_X， $X \in [1, 20]$ ，每个页面有20条新闻。

张朝阳：希望搜狐2020年实现盈利，股票赶快涨起



“希望公司在2020年实现盈利，股票赶快涨起来。”在离2019年结束还有最后一个月的时候，搜狐公司董事局主席兼首席执行官张朝阳在接受媒体时透露了对明岁盈利的期望。2019年，张朝阳在多个场合提出搜狐要回归媒体。在接受媒体采访他表示，新技术的... [阅读更多]

网易科技报道 2019-11-27 12:51:55



美团点评市值突破6000亿港元，仅次于阿里、腾讯



野村发表研究报告称，分别升美团点评2019及2020年收入预测2%及3%，并将其目标价由107港元升至120港元，维持“买入”评级。11月27日，美团点评涨近3%，截至发稿，报103.8港元，创上市以来新高，市值突破6000亿港元，为市值仅... [阅读更多]

第一财经 2019-11-27 11:08:37



图 2: 滚动_网易科技频道

- 原创报道_网易科技: <http://tech.163.com/special/0009rt/ycbd.html>
第X个页面的地址为/ycbd_i.html, $X \in [1, 10]$, 每个页面有100条新闻。

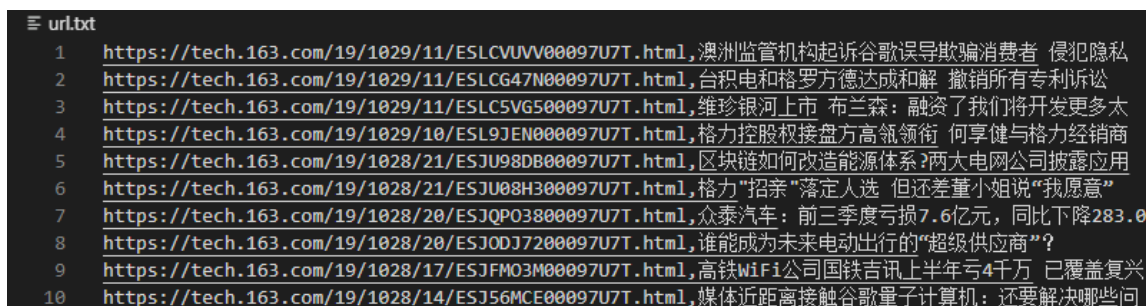
原创报道	
▪ 刘强东卸任佳康医药经理职务 11月卸任多家企业高管	(2019-11-27 13:44:11)
▪ Facebook宣布收购VR音乐游戏开发商Beat Games	(2019-11-27 13:22:02)
▪ 张朝阳：希望搜狐2020年实现盈利，股票赶快涨起来	(2019-11-27 12:51:55)
▪ 寒武纪全资子公司注册资本新增至8亿元 增幅达60%	(2019-11-27 10:44:55)
▪ 苹果员工称支持库克结交特朗普：对公司有利	(2019-11-27 09:03:02)
▪ 施乐致信惠普董事会：我们会让你们的股东同意收购	(2019-11-27 07:54:14)
▪ 尽管遭受挫折，华为手机Q3销量大涨，继续追赶三星	(2019-11-27 07:50:07)
▪ 传蚂蚁金服拟建10亿美元创投基金 专投印度东南亚	(2019-11-27 07:16:39)
▪ 美小偷用蓝牙耳机精准扫描车内电脑手机 然后砸车盗窃	(2019-11-27 07:04:36)
▪ 阿里健康发布中期公告：收入41亿 同比增119.1%	(2019-11-26 20:05:12)
▪ 快手高级副总裁马宏彬：春节前补贴66.6亿教育流量	(2019-11-26 19:44:57)
▪ 易车Q3财报：营收25.41亿元，净亏损1.62亿元	(2019-11-26 19:37:37)
▪ 报废电动车有去处？这家车企从旧电池提取镍和钴	(2019-11-26 17:46:14)
▪ 快手又在音乐方面发力了！200亿流量扶持音乐主播	(2019-11-26 17:34:30)
▪ 赵明：荣耀V30要引爆5G市场 未来新品几乎全部转向5G	(2019-11-26 17:31:16)

图 3: 原创报道_网易科技

这三个页面的HTML结构比较清晰，且不需要不包含javascript代码通过后台数据库进行数据的动态生成，因此大大降低了爬虫抓取数据的难度。

在每一个入口页面找到内层文本标签中的href标记，并将其中的链接提取出来，加上新闻

标题一起写入csv文件，即url.txt，如下图所示。



```

url.txt
1 https://tech.163.com/19/1029/11/ESLCVUV00097U7T.html,澳洲监管机构起诉谷歌误导欺骗消费者 侵犯隐私
2 https://tech.163.com/19/1029/11/ESLCG47N00097U7T.html,台积电和格罗方德达成和解 撤销所有专利诉讼
3 https://tech.163.com/19/1029/11/ESLC5VG500097U7T.html,维珍银河上市 布兰森：融资了我们将开发更多太
4 https://tech.163.com/19/1029/10/ESL9JEN000097U7T.html,格力控股权接盘方高瓴领衔 何享健与格力经销商
5 https://tech.163.com/19/1028/21/ESJU98DB00097U7T.html,区块链如何改造能源体系?两大电网公司披露应用
6 https://tech.163.com/19/1028/21/ESJU08H300097U7T.html,格力“招亲”落定人选 但还差董小姐说“我愿意”
7 https://tech.163.com/19/1028/20/ESJQP03800097U7T.html,众泰汽车：前三季度亏损7.6亿元，同比下降283.0
8 https://tech.163.com/19/1028/20/ESJODJ7200097U7T.html,谁能成为未来电动出行的“超级供应商”？
9 https://tech.163.com/19/1028/17/ESJFM03M00097U7T.html,高铁WiFi公司国铁吉讯上半年亏4千万 已覆盖复兴
10 https://tech.163.com/19/1028/14/ESJ56MCE00097U7T.html,媒体近距离接触谷歌量子计算机：还要解决哪些问
  
```

图 4: url.txt文件实例

2. 下载新闻文本

我在程序中封装了以下函数

```
def crawl(url,outfile_name)
```

其中url为从上一阶段获得的每一条新闻的url地址，outfile_name为输出文本的名字。

网易所有的新闻文本都会被封装在<post_text>属性中，因此只需下载对应页面后，对HTML网页进行解析获取对应标签内容即可。

但要注意大多数网页可以用lxml进行解析，但对于少部分网页，需要使用html.parser配合gb18030编码来处理中文。

3. 新闻数据处理

对下载下来的原始新闻文本做一些简单的处理，包括：

- 删除行中的多余空格（用lstrip和rstrip）
- 删除网易新闻的标记，如“_网易科技”、“网易科技讯”等
- 删除其他无关字符，如@@LinkCard

4. 实施细节

这一部分主要采用Python的bs4和request包进行新闻内容的抓取，其他一些实现细节如下：

- 为避免网页的反爬虫机制，需要对爬虫进行伪装，因此需要自行设置网页请求头send_headers
- 在网页抓取时设置time_out，防止抓取网页时因为网速过慢等原因卡死整个程序
- 读写文件时注意强制声明编码为utf-8
- 为了避免网页的重复抓取，采用os.path.isfile判断文件是否存在，若某一新闻文件已经存在，则不再访问对应网页。

- 使用Python的多进程库multiprocessing¹并发抓取网页，加快任务执行速度。

二、新闻数据预处理

新闻数据预处理在前一节已经完成了一部分，因此这里着重于中文的分词。采用结巴分词(cut)工具，完整代码可见cut.py。

核心代码如下所示。

```
text = jieba.lcut(intext,cut_all=False)
```

这里采用了精确模式(cut_all=False)，可以确保一些词汇不会被分割得太细，如避免“人工智能”分割为“人工|智能”；同时采用lcut可以使结果直接返回一个列表，而不是Python的迭代器。

除了前述的预处理工作，在本部分还进行了以下工作：

- 将多个空行换为单一空行，即

```
intext = intext.replace('\n\n','\n')
```

- 尽可能将不同的句子分置在不同的行中，即将在句号末换行。
- 用collections.Counter对分词后的列表进行计数，生成dict.txt（这一部分在后面被移到各自的训练模型文件中）

生成预处理后的分词文件实例如下。

```
1  华为 全面 启航 计算 战略： 打造 “ 鲲鹏 + 昇 腾 ” 双引擎 布局 。
2
3  9 月 19 日 消息 ， 2019 华为 全 联接 大会 期间 ， 华为 基于 “ 鲲鹏 + 昇 腾 ” 双引擎 正式 全面 启航 计
4  算 战略 ， 宣布 开源 服务器 操作系统 、 GaussDB OLTP 单机版 数据库 ， 开放 鲲鹏 主板 。
5  华为 希望 通过 硬件 开放 、 软件 开源 、 使能 合作伙伴 ， 开拓 万亿 级 计算 产业 大 蓝海 。
6
7  华为 轮值 董事长 胡厚 崑 宣布 华为 正式 启动 计算 战略 。
8  在 计算 方面 ， 华为 已经 投入 10 年 时间 。
9  面向未来 ， 华为 越来越 意识 到 ， 计算 是 必须 持续 投入 的 领域 。
10 进入 智能 时代 ， 计算 将 无处不在 。
11
12 为了 实现 这个 追求 ， 华为 将 打造 “ 一云 两翼 、 双引擎 ” 的 产业布局 ， 构筑 开放 的 产业 生态 。
```

图 5: 预处理后分词文件实例

三、模型训练

在本次实验中，我使用预处理后的新闻数据训练了两个语言模型，一个为3-gram，另一个为LSTM，详细说明如下。

¹之所以不使用多线程，是因为Python的多线程机制非常鸡肋，同时没有线程池可以管理。而多进程相对比较成熟，且有进程池Pool统一进行管理。

1. n-gram模型

n-gram模型为统计语言模型，我将其分成以下三个阶段进行模型生成及预测。完整代码可见ngram.py。

(i) 去除停止词及加句子标记

n-gram模型能够正常工作很重要一点在于去除停止词，这里我采用了 [1]中的中文常用停用词列表。通过判断某一单词是否在停用词列表中，然后决定是否将其保留。

同时，每读入一个句子需要在句子前面加上起始标记<BOS>，在句末加上终止标记<EOS>。

(ii) 生成n-gram列表

将上述去除停用词及加句子标记的文本重新读入后，调用generate_ngram函数生成n-gram列表。这里我采用了一种非常快速且巧妙的方法，通过对单词列表移位然后重新组合，即可得到对应的n元组。再将这些n元组用空格连接，即可得到对应的n-gram。

```
def generate_ngram(text,n):
    token = text.split()
    ngrams = zip(*[token[i:] for i in range(n)])
    return [" ".join(ngram) for ngram in ngrams]
```

在实际实施中，会在这个阶段生成三个n-gram列表，如下所示。同时用Counter对列表内容进行计数。将生成的计数结果用pickle序列化处理，保存为pkl方便后续直接读入使用。

- 1-gram: 实际上就是词表，在选词填空时将会从中进行选择
- n-gram: 核心的n-gram模块，保存着所有的n-gram文本
- (n-1)-gram: 用于实际计算中历史字串的索引

(iii) 模型预测

对于每一个需要预测的句子，读入后先确定[MASK]标记的地方，然后分为前后两个部分进行分词。由于在模型生成中将停用词去除并添加了句首句末标记，因此在实际预测中也需要对预测的文本进行同样的处理。

通过遍历词表，每次选择一个词填入[MASK]标记中，然后利用加性平滑(additive smoothing)通过下式计算最大似然概率

$$p(w_i | w_{i-n+1}^{i-1}) = f(w_i | w_{i-n+1}^{i-1}) = \frac{c(w_{i-n+1}^i)}{\sum_{w_i} c(w_{i-n+1}^i)}$$

其中， $\sum_{w_i} c(w_{i-n+1}^i)$ 为历史串 w_{i-n+1}^{i-1} 在给定语料中出现的次数，即 $c(w_{i-n+1}^i)$ ，这也是为什么前面需要提前计算出(n-1)-gram。从所有计算出的概率值中选择最大的那一个，其对应的词语即n-gram模型预测应该填入的词，核心代码如下。

```
# calculate the maximum probability
```

```
rank = []
for mask in word_counter.keys():
    if mask in ["<BOS>", "<EOS>"]:
        continue
    # only focus on the n-1 words before and after [MASK]
    new_text = seg_pre[-N+1:] + [mask] + seg_post[:N-1]
    text_gram = generate_ngram(new_text, N)
    gram_val = cal_ngram(text_gram)
    rank.append((gram_val, mask))
```

另外，注意到其实句子的其他部分对最终计算出来的概率值没有影响，真正对概率值有贡献的只有[MASK]前后 $n - 1$ 的词的范围。因此只考虑这 $2n - 1$ 个词将大大缩减预测时间。

具体实验结果可见第4.1节。

2. LSTM模型

LSTM模型为神经网络模型，包括数据生成、模型搭建、训练和预测。采用Pytorch v1.1进行模型搭建，并使用1块Titan V GPU进行训练。由于一开始在Jupyter Notebook中进行代码编写及运行，因此建议直接访问lstm.ipynb查看交互式数据。另外，也可直接查看jupyter生成的代码文件lstm.py。

(i) 数据生成

这里我编写了自己的TextDataset，其继承了torch.utils.data.Dataset。完整的类声明如下。

```
class TextDataset(data.Dataset):
    """
    My own text dataset
    """
    def __init__(self, file_path, batch_size, seq_size):
        """
        Generate word indices and dictionary used for training

        file_path: The path of the folder storing all the news
        batch_size: size of one batch (used for batch training)
        seq_size: size of the sequence
        """
        text = []
        for i, file_name in enumerate(os.listdir(file_path), 1):
            with open("{} / {}".format(file_path, file_name), "r", encoding="utf-8") as \
                infile:
                for j, line in enumerate(infile):
                    text += line.split()
        word_counts = Counter(text) # {word: count}
        # sort based on counts, but only remain the word strings
```

```

sorted_vocab = sorted(word_counts, key=word_counts.get, reverse=True)

# make embedding based on the occurrence frequency of the words
self.int_to_word = {k: w for k, w in enumerate(sorted_vocab)}
self.word_to_int = {w: k for k, w in self.int_to_word.items()}
self.n_word = len(self.int_to_word)
print('Vocabulary size', self.n_word)

# turn all the words in the text to int
int_text = [self.word_to_int[w] for w in text]
num_batches = int(len(int_text) / (seq_size * batch_size))
in_text = int_text[:num_batches * batch_size * seq_size]

# shift right for one position to generate the 'label' Y
out_text = np.zeros_like(in_text)
out_text[:-1] = in_text[1:]
out_text[-1] = in_text[0]

# reshape X and Y (# of seq, seq_size)
self.in_text = np.reshape(in_text, (-1, seq_size))
self.out_text = np.reshape(out_text, (-1, seq_size))
self.seq_size = seq_size

def __len__(self):
    """
    Return the total number of samples
    """
    return len(self.in_text)

def __getitem__(self, idx):
    """
    Generate one sample of the data
    """
    x = self.in_text[idx]
    y = self.out_text[idx]
    return x, y

```

TextDataset中主要有三个函数：

- `__init__`：将预处理过的所有新闻文本文件读入（注意这里采用了和ngram预处理同样的方法，删除了标点符号和停止词，并添加句首句末标记），然后构建词表，并对词频排序与单词之间建立一个一一映射，此时相当于把每一个单词都映射到一个整数序号值。将输入文本(X)右移一位即得到输出文本(Y)，同时用`reshape`对输入输出向量的维度进行改变，最后一维为序列长`seq_size`。

- `__len__`: 返回序列数目。
- `__getitem__`: 根据idx获得对应的序列。

封装好TextDataset后，结合DataLoader即可实现批训练数据的生成。

```
train_set = TextDataset(flags.train_file_path, flags.batch_size, flags.seq_size)
train_loader = data.DataLoader(dataset=train_set, batch_size=flags.batch_size,
    ↪ shuffle=False)
```

(ii) 模型搭建

在pytorch中搭建神经网络模型非常容易，只需定义好相关变量，并写好前向传播过程即可²，代码模块如下。

```
class RNNModule(nn.Module):
    """
    A basic LSTM model for text generation
    """
    def __init__(self, n_word, seq_size, embedding_size, lstm_size):
        super(RNNModule, self).__init__()
        self.seq_size = seq_size
        self.lstm_size = lstm_size

        # embed = nn.Embedding(vocab_size, vector_size)
        # vocab_size is the number of words in your train, val and test set
        # vector_size is the dimension of the word vectors you are using
        # you can view it as a linear transformation
        # the tensor is initialized randomly
        self.embedding = nn.Embedding(n_word, embedding_size)

        self.lstm = nn.LSTM(embedding_size, lstm_size, batch_first=True)
        self.linear = nn.Linear(lstm_size, n_word)

    def forward(self, x, prev_state):
        """
        Forward propagation
        """
        embedding = self.embedding(x)
        # used for next layer
        output, state = self.lstm(embedding, prev_state)
        # used for output
        logits = self.linear(output)
        return logits, state

    def zero_state(self, batch_size):
```

²后向传播由计算流图自动求导生成

```

"""
Used to make the state all zeros
"""
return (torch.zeros(1, batch_size, self.lstm_size),
        torch.zeros(1, batch_size, self.lstm_size))

```

这里将输入向量做了词嵌入，映射到维度为`vector_size`空间中的一个向量，然后用LSTM模型进行训练。

LSTM的模型如图6所示³。

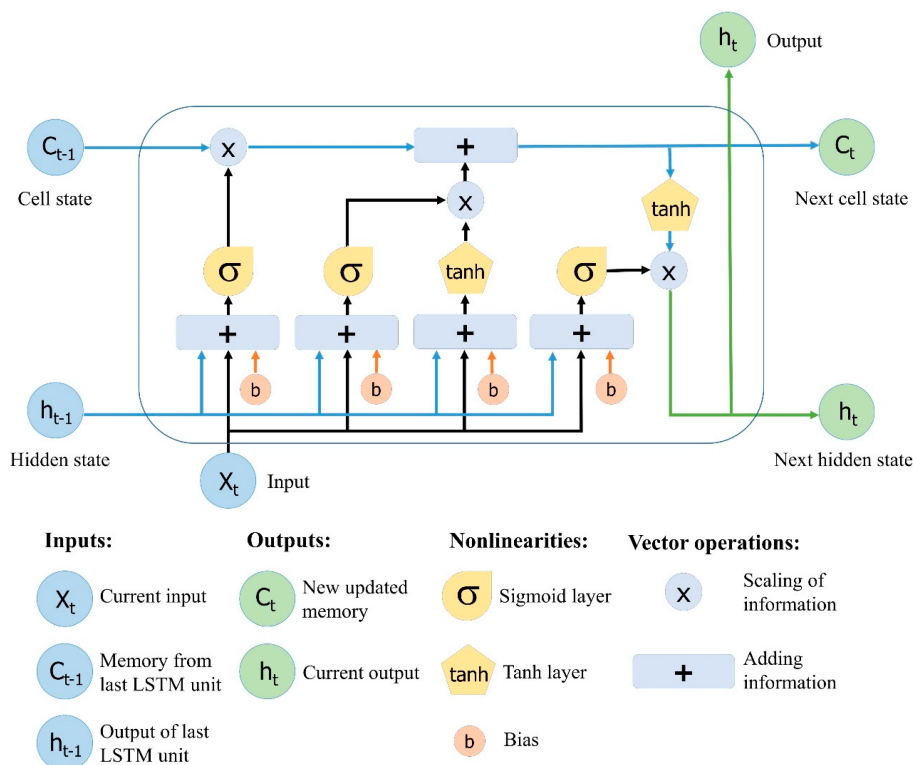


图 6: LSTM单元

因此前向传播主要包括横向纵向的传播，横向传播状态，纵向生成输出，即对应着下面这条语句。

```
output, state = self.lstm(embedding, prev_state)
```

为提升模型的泛化能力，输出会再通过一个线性层，得到最终的输出向量。

(iii) 模型训练

模型训练的核心代码如下所示，在每一轮训练中都会将所有数据进行批训练。

³图源自Application of Long Short-Term Memory (LSTM) Neural Network for Flood Forecasting, <https://www.mdpi.com/2073-4441/11/7/1387>

```
for e in range(flags.num_epochs):
    state_h, state_c = net.zero_state(flags.batch_size)
    state_h = state_h.to(device)
    state_c = state_c.to(device)

    for step, (x, y) in enumerate(train_loader):
        iteration += 1
        net.train()

        optimizer.zero_grad()

        x = torch.tensor(x).to(device)
        y = torch.tensor(y).to(device)

        logits, (state_h, state_c) = net(x, (state_h, state_c))
        loss = criterion(logits.transpose(1, 2), y)

        # avoid delivering loss from h_t and c_t
        # thus need to remove them from the computation graph
        state_h = state_h.detach()
        state_c = state_c.detach()

        loss_value = loss.item()

        loss.backward()

        # avoid gradient explosion
        _ = torch.nn.utils.clip_grad_norm_(net.parameters(), flags.gradients_norm)

        optimizer.step()
        losses.append(loss_value)
```

这里采用了交叉熵作为模型好坏的评价指标，并使用Adam优化器对目标进行优化。从代码中可以看出在每一次批迭代中主要过程如下：

1. 将优化器的梯度置零，在当轮训练中才进行梯度累加
2. 从DataLoader中读取批数据，转换为tensor，并放置到CPU/GPU上
3. 利用LSTM进行前向传播，并计算损失(loss)
4. 将 h_t 和 c_t 从计算流图中分离，再进行梯度回传
5. 为避免梯度爆炸，采用梯度裁剪(gradient clip) [5]的方式对每轮迭代后的梯度进行重新缩放处理
6. 优化器在梯度方向上更进一步，进入下一轮迭代

以上几个步骤不断迭代，直到模型收敛或达到最大训练轮数。

(iv) 模型预测

预测的方式与ngram模型类似，只是将核心部分改为神经网络的推断，核心代码如下。

```
def predict(device, net, question_str, n_word, word_to_int, int_to_word, top_k=5):
    """
    Use net to do the prediction
    Each time only one question_str is input
    """
    net.eval() # set in evaluation mode
    # find out the blank
    q_index = question_str.index("[MASK]")
    question_pre, question_post = question_str[:q_index], question_str[q_index+len
        ↪ (" [MASK] "):]

    # cut the sentence
    seg_pre = generate_seg_lst(question_pre)
    seg_post = generate_seg_lst(question_post)
    seg_pre.insert(0, "<BOS>")
    seg_post.insert(len(seg_post), "<EOS>")

    # LSTM inference
    state_h, state_c = net.zero_state(1)
    state_h = state_h.to(device)
    state_c = state_c.to(device)
    for w in seg_pre:
        index = word_to_int.get(w, word_to_int["<BOS>"])
        ix = torch.tensor([[index]]).to(device)
        output, (state_h, state_c) = net(ix, (state_h, state_c))

    # get the topk prediction
    _, top_ix = torch.topk(output[0], k=top_k)
    choices = top_ix.tolist()

    # return the corresponding words
    return [int_to_word[x] for x in choices[0]]
```

步骤如下：

1. 对句子进行分割和分词，删除停止词、标点符号，添加句首句末标记
2. 将输入文本（[MASK]前面的部分）转换为对应的词向量，输入LSTM进行前向推理，每一次都更新状态 h_t 和 c_t
3. 预测输出[MASK]处的词向量，选取最高的 k 个的值作为预测
4. 将词向量映射回对应的单词输出

(v) 其他实施细节

- 采用`argparse`进行命令行参数的读入与操作（主要是一些模型的超参数及文件路径）
- 采用`logging`模块对训练过程进行日志记录，以便跟踪存在的问题
- 采用`time`模块对模型训练时间进行记录，同时可以预测剩余训练时间
- 采用`torch.save(net.state_dict())`的方法保存模型参数，一方面可以防止训练过程中的模型丢失，另一方面又可以避免将整个模型存储下来的空间开销

四、实验结果

在具体实验中我将数据集进行了扩增，共采用2296条新闻进行模型生成、训练与预测。

1. n-gram模型

实验中我分别使用了 $n = 2, 3, 4$ 的几种模型进行预测，完整的预测正确率可见图13，完整结果可见`myanswer-ngram.txt`。执行样例如图7和图8所示。

```

chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/NaturalLanguageProcessing/Project1
90 [MASK] = ['|'] (2.2895120202768376e-17) - 损失
91 ✓ [MASK] = ['合作'] (6.45023305964232e-11) - 合作
92 [MASK] = ['若干'] (1.1445512581826454e-16) - 速度
93 [MASK] = ['金融'] (2.2890700079675637e-16) - 人工智能
94 [MASK] = ['零售'] (6.867775505735471e-17) - 智能机
95 [MASK] = ['|'] (2.2895120202768376e-17) - 产品
96 [MASK] = ['董事会'] (4.5789590296976635e-17) - 无线电
97 [MASK] = ['人民币'] (2.4724413006235383e-15) - 营收
98 [MASK] = ['|'] (2.2895120202768376e-17) - 安全
99 [MASK] = ['包括'] (6.86834103036689e-17) - 价格
100 ✓ [MASK] = ['企业'] (1.693114063169075e-10) - 企业
Accuracy: 19.00%
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/NaturalLanguageProcessing/Project1$

```

图 7: 3-gram模型(Top 1)执行过程与预测准确率

```

chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/NaturalLanguageProcessing/Project1
90 [MASK] = ['|', 'G', '8', '5', '2'] (2.2895120202768376e-17) - 损失
91 ✓ [MASK] = ['合作', '互鉴', '需求', '互动', '融合'] (6.45023305964232e-11) - 合作
92 [MASK] = ['若干', '迈入', '支撑', '倡议书'] (1.1445512581826454e-16) - 速度
93 [MASK] = ['金融', '成长', '分水岭', '发展', '上市公司'] (2.2890700079675637e-16) - 人工智能
94 [MASK] = ['零售', 'IT', '四化', '服务', '消费'] (6.867775505735471e-17) - 智能机
95 [MASK] = ['|', 'G', '8', '5', '2'] (2.2895120202768376e-17) - 产品
96 [MASK] = ['董事会', '|', 'G', '8', '5'] (4.5789590296976635e-17) - 无线电
97 [MASK] = ['人民币', 'G', '亿元', '收入', '成交额'] (2.4724413006235383e-15) - 营收
98 [MASK] = ['|', 'G', '8', '5', '2'] (2.2895120202768376e-17) - 安全
99 [MASK] = ['包括', '60GB', '深市', '包含', '5G'] (6.86834103036689e-17) - 价格
100 ✓ [MASK] = ['企业', '平台', '合作', '频道', '行业'] (1.693114063169075e-10) - 企业
Accuracy: 19.00%
chhzh123@DESKTOP-PV2UBJL: /mnt/d/Assignments/NaturalLanguageProcessing/Project1$

```

图 8: 3-gram模型(Top 5)执行过程与预测准确率

这里以3-gram的预测结果进行分析。从上述图中可以看出3-gram的预测准确率是非常高的，达到了19%，而且这19个全部在Top 1命中。

像预测值中出现的“|, G, 8, 5, 2”等是因为词表不够大，导致没有找到合适的词语填入，因此所有词语得到的预测概率都近似相同。但可以预计，如果我们有足够大的词表，那么基于传统的统计模型进行文本预测依然可以做得很好。

接下来我们着重分析那些有正常的预测值的情况。先看预测正确的情况，以第100条句子为例。

100 携程的目标是三年内成为亚洲最大的国际旅游企业，五年内成为全球最大的国际旅游企业，十年内成为最具价值和最受尊敬的在线旅游[MASK]。

在这个空中，n-gram模型直接将在线旅游与企业/平台联系在一起，并且给出了较高的预测值。但其实这个空填平台也没有太大问题，因为携程确实也可以被称为在线旅游平台，只是为了跟前文的国际旅游企业照应，因此这里填企业较优。

接下来再看看错误的情况，以第93、94条句子为例，这两个例子明显体现出统计模型的“缺陷”。

93 互联网和人工智能技术为全世界的各个国家都带来巨变。在这一方面，中国做得非常优秀，中国将互联网与[MASK]技术结合得“炉火纯青”，这些经验值得各个国家学习。

第93句中由于大量语料都将互联网和金融相提并论，因此这里直接得出金融为最高的预测值。但是，n-gram的窗口明显没有将前一句中的人工智能纳入考虑，故属于不能具体情况具体分析，而只是生搬硬套以前的结论。

94 总之，5G将开启一个新智能机时代，而创新将是新智能机时代的第一发展动力，期待国产品牌在新[MASK]时代大有作为。

同样，对于第94句来说，新零售时代和IT时代都是没有语病的，但是放在句子中却会产生不对应，因为前文提及的是新智能机时代，因此后面也应该填智能时代。

2. LSTM模型

LSTM模型中使用的超参数如表1所示，训练时间与超参数选择相关，由8分钟到18分钟不等。

表 1: LSTM模型超参数

参数	变量名	数值
序列长	seq_size	32
批大小	batch_size	64
词嵌入维度	embedding_size	128
LSTM隐态维度 [4]	lstm_size	128
梯度裁剪阈值 [5]	gradients_norm	5
训练轮数	num_epochs	40
优化器学习率	learning_rate	0.001

训练过程中的损失函数变化如图9所示，可以看到损失函数在不断下降，说明模型在不断变好。

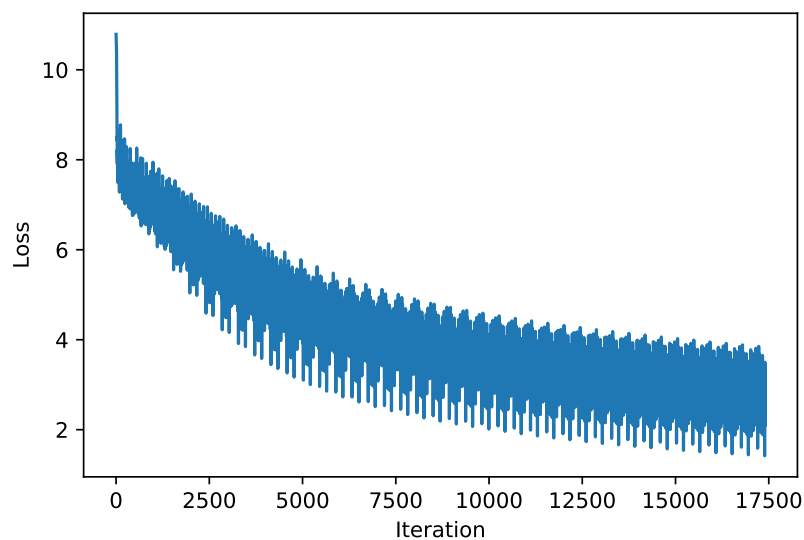


图 9: LSTM训练损失(loss)变化

对应的Top 1、Top 3、Top 5预测准确率如图10所示，训练到后期模型已经开始有过拟合的趋势（Top 3和Top 5准确率下降）。

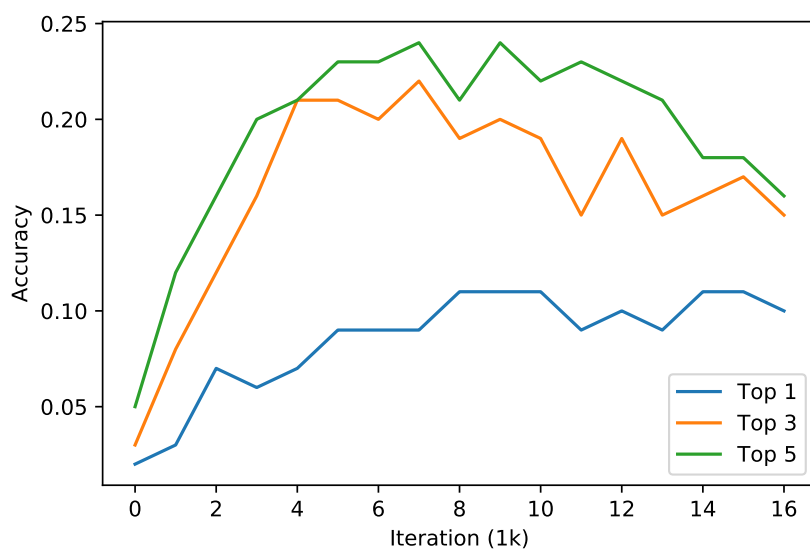


图 10: LSTM训练预测准确率变化

实际推断样例如图11和图12所示（完整结果可见myanswer-lstm.txt），可以看到LSTM模型的Top 1的准确率并不高，只有11%；而Top 5的预测准确率才超过传统的统计模型。另外，由于LSTM模型是预训练的，而n-gram模型是在线推断，因此相比起n-gram，LSTM的预测速度远超n-gram模型。

```

90 [MASK] = ['<EOS>'] - 损失
91 [MASK] = ['<EOS>'] - 合作
92 [MASK] = ['<EOS>'] - 速度
93 [MASK] = ['发展'] - 人工智能
94 [MASK] = ['技术'] - 智能机
95✓ [MASK] = ['产品'] - 产品
96 [MASK] = ['<EOS>'] - 无线电
97 [MASK] = ['收购'] - 营收
98 [MASK] = ['<EOS>'] - 安全
99 [MASK] = ['<EOS>'] - 价格
100✓ [MASK] = ['企业'] - 企业
Accuracy: 11.00%

```

图 11: LSTM模型(Top 1)执行过程与预测准确率

```

90 [MASK] = ['<EOS>', '公司', '投资', '中', 'We'] - 损失
91 [MASK] = ['<EOS>', '落地', '人类', '构建', '更'] - 合作
92 [MASK] = ['<EOS>', '倡议书', '挑战', '机会', '阶段'] - 速度
93 [MASK] = ['发展', '技术', '场景', '互联网', '领域'] - 人工智能
94 [MASK] = ['技术', '机会', '模式', '产业', '趋势'] - 智能机
95✓ [MASK] = ['产品', '<EOS>', '德国', '尺寸', '充电'] - 产品
96 [MASK] = ['<EOS>', '5G', '激进', 'VR', '技术'] - 无线电
97 [MASK] = ['收购', '<EOS>', '提出', '公司', '信贷'] - 营收
98 [MASK] = ['<EOS>', '互联网', '网络', '用户', '技术'] - 安全
99✓ [MASK] = ['<EOS>', '价格', 'GB', '更', '以内'] - 价格
100✓ [MASK] = ['企业', '行业', '公司', '服务', '时'] - 企业
Accuracy: 24.00%

```

图 12: LSTM模型(Top 5)执行过程与预测准确率

在实际操作中如果某个词语在词表中找不到，则我将其直接映射为<BOS>对应的向量，因此这里出现的<BOS>和<EOS>同样是由于词表太小不够导致的预测错误，而非模型本身的问题。

接下来同样进行样例分析。

95 面对山河日下的情况，漫步者推出旗下定位于高端无线便携音响的[MASK]，聚焦打造售价、利润率更高的高端产品线，以期提高利润率。

第95条句子可以将音响与产品联系在一起，这是非常不容易的，在传统的统计模型中就很难做到这一点。同时后面的预测值中还有充电等字眼，这是与无线进行了匹配。

98 腾讯安全平台部作为专注腾讯企业内部安全的团队，也把十余年腾讯自身安全最佳实践对外开放赋能，并逐步把[MASK]能力向腾讯云上开放，做好产业互联时代的数字化安全助手。

第98条句子虽然整句话都在提及互联网相关的内容，但是神经网络显然将其定位得太宽泛，互联网、网络、用户、技术确实在相关语料中常常一起出现，但是具体到这一个句子中却不是正确的匹配项。因为前面提及到安全，因此后文也应该用安全相匹配。

从这些例子中也可以看出神经网络模型更具灵活性，即使对于同样的输入值，由于处在不同的状态（上下文），因此也会预测出不同的结果。

3. 综合比较

所有模型的预测准确率如图13所示。

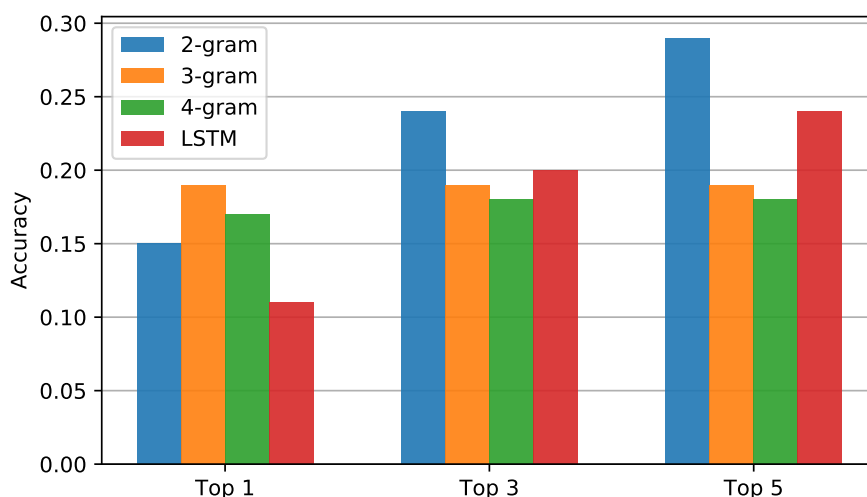


图 13: 预测准确率比较

从图13中，我们可以得到以下结论：

- 在Top 1情况下，3-gram的预测精度最高，达到19%；Top 3情况下，2-gram的预测精度最高，达到24%；Top 5情况下，2-gram预测精度最高，达到29%。
- 传统的统计模型并非一无是处，从这个实验结果来看似乎是反直觉的——2-gram模型的预测精度竟然始终高于神经网络模型，同样也比3-gram和4-gram模型要好很多。虽然2-gram采用非常极端的贪心算法（断章取义），但是对于这种小样本预测效果反而是最好的。
- n-gram模型采用更大的窗口 n 将使得模型更加稳定，基本上能够在Top 1预测出来的结果都是十分确定的，而其他没预测出来的通常就是词表中找不到的词语。
- LSTM的Top 1、Top 3、Top 5的预测精度分别为11%、20%和24%。虽然在这个实验中并不是很惊艳，但是从预测结果来看，LSTM的泛化能力最强，推断时间更快，且更能依照不同句子的特性做出适应性的推断。
- 随着Top k 的 k 值不断增加，各个模型的预测准确率都在不断提升。这说明模型很多时候并没有办法做到精确匹配，但是如果放宽条件，其预测准确率还是可以接受的。

五、总结与思考

这次项目让我对整个数据收集、处理、分析、建模的全过程有了深入的了解，同时也对自然语言处理的原理和理论有了更进一步的理解。

从这次项目中得到最大的体会就是**有多大的人工，就有多大的智能**。在软件2.0时代 [8]，我们输入给程序的不再是算法，更加关键的是训练数据；以前程序编译出来的是机器指令，而现在则变成了网络的学习参数。以前算法决定了模型的性能，而现在则是**数据规模、模型复杂性**决定了其性能。随着输入数据规模的不断增大，神经网络训练出来的效果也会变得更好。

从这次实验中也可以看出，无论是统计模型还是神经网络模型其实都是无关语法的，不需太多前置知识，更加容易搭建且达到很高的精度。相比起n-gram，神经网络模型更加是小白都可以搭。只要模型没有问题，什么数据丢进去训练就好了（甚至不需要做停止词消除或其他的一些预处理，尽管会有影响），神经网络总能拟合到一个不错的结果，关键在于**超参数的选择与调整**上。这体现了神经网络强大的拟合能力，也体现了软件2.0时代与软件1.0时代的巨大区别。

关于这两个模型的问题在前文实验中已经提及，这里再进行总结：

- **准确率**：这是衡量模型性能最重要的指标，由于训练样本太小，测试样本也太小，其实是很难看出什么的。但有一个明显的趋势在于，当数据量加大时，对于两个模型来说都有着巨大的提升空间。只要语料及词表足够大，这两种模型应该都能达到很好的预测效果。
- **推理能力**：无论是n-gram还是LSTM其实都是按照既定的统计方法进行预测，它并不具有真正的智慧，或者说不具有推理能力。比如前文举的几个案例分析，都是句子中的前半句提及了某样事物，后半句就进行复现，对于人类来说这是很明显的事情，但对于机器来说却非常困难。即便是LSTM已经将过往历史很好地进行记录，但是遇到这种问题还是有些捉襟见肘。不过这有点偏阅读理解的范畴，可能有一些更好的语言模型进行处理，这可以是后续的研究方向。
- **泛化能力**：虽然2-gram在本次实验中取得很好的成绩，但这明显是一种鼠目寸光、断章取义的行为，只通过前后两个词而不是通过整个句子的意思来填空显然是不可取的。虽然LSTM能够对整个句子进行分析，但一旦句子变得很长，它也很难分析清楚前后关系。另外，对于这两个模型来说，一个很重要的问题都是没有办法很好解决不在词表中的词语，但显然我们也不可能建立一个足够大的词库涵盖所有的情况，因此需要有更好的方法来处理这种没出现过的情况。泛化也和推理密切相关，想像“太阳从东边升起”这一句子，将中间一词“东”挖去，如果机器通过足够多的统计数据知道太阳和东必然一起出现，那之后的例子中这两者在一起的概率都会被设得很大。但是如果在这一个句子后面加一些内容，如“太阳从[MASK]边升起是错的”，这时就不能填东了，而需要通过后文的推理

得出填写其他的词语。(不过这可能扯得有点远了, 会涉及到知识表示与推理的内容)

总的来说, 本次实验既让我熟悉了大数据分析的整个流程, 又加深了对自然语言处理语言模型的理解。虽然步骤非常繁杂, 但整个过程还是非常有趣的。

参考文献

- [1] 中文常用停止词列表, <https://github.com/goto456/stopwords>
- [2] Colah, <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>
- [3] Pytorch tutorial, https://pytorch.org/tutorials/beginner/nlp/sequence_models_tutorial.html
- [4] Pytorch nn.LSTM, <https://pytorch.org/docs/stable/nn.html#torch.nn.LSTM>
- [5] How to Avoid Exploding Gradients With Gradient Clipping, <https://machinelearningmastery.com/how-to-avoid-exploding-gradients-in-neural-networks-with-gradient-clipping/>
- [6] Text Generation With Pytorch, <https://machinetalk.org/2019/02/08/text-generation-with-pytorch/>
- [7] Language Modelling and Text Generation using LSTMs - Deep Learning for NLP, <https://medium.com/@shivambansal36/language-modelling-text-generation-using-lstms-deep-learning-for-nlp-ed36b224b275>
- [8] Kunle Olukotun, *Designing Computer Systems for Software 2.0*, ISCA Keynote, 2018