Seq2Seq 小说文本生成

姓名 鲍宇翔

学号 ZY2103104

一、实验要求

基于 Seq2seq 模型来实现文本生成的模型，输入可以为一段已知的金庸小说段落，来生成新的段落并做分析。

二、实验原理

2.1 seq2seq 模型

Seq2seq 模型就是一种能够根据给定的序列，通过特定的方法生成另一个序列的方法。它在许多领域产生了一些运用。目前，它主要的应用场景有：机器翻译、聊天机器人、文本生成等。

Seq2seq 模型主要由编码器和解码器两部分构成，在这个结构中，输入一个句子后，生成语义向量 c，编码过程比较简单；解码时，每个 c、上一时刻的 yi-1，以及上一时刻的隐藏层状态 si-1 都会作用到 cell，然后生成解码向量。

编码器端往往采用序列模型，如 RNN，LSTM 等。在编码的每个时刻，模型的输入除了上一时刻产生的隐层状态编码，还有当前时刻的输入字符，并将最后模型最后一个时刻的隐层状态做为整个序列的编码表示，传递给解码器。

解码器端与编码器端近乎相同，不过解码器端需要保存模型的输出用于产生输出序列。在模型的训练阶段，模型的输入是文本内容以及上一刻的状态变量，模型输出为预测的下一个序列变量。在模型的预测阶段，模型输入为上一个时刻的输出以及状态，来预测下一个时刻的输出。

整个编码-解码器的预测阶段的工作流程如下图所示

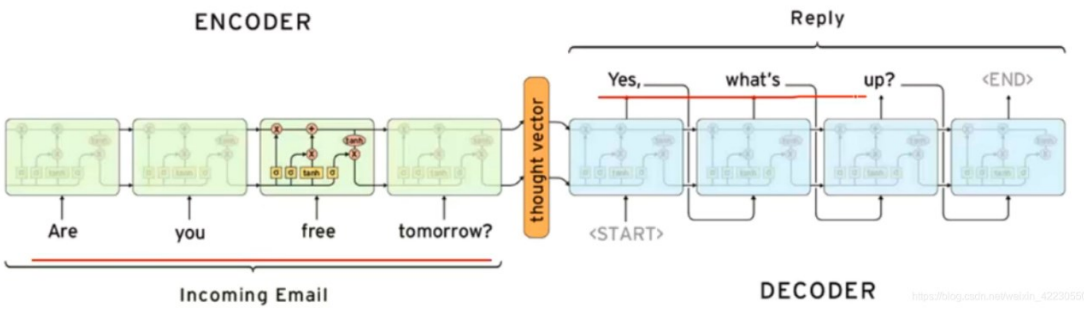

图 1 seq2seq 模型

三、实验过程

3.1 文本预处理

过程与前几次实验大体相同，包括文本的读取，去除特殊标点符号，去除停词，分词等操作。为了让分词更准确，在网站上下载了人名、门派、武功的专有词汇，用于分词过程中。

3.2 模型定义

模型的定义与训练包括 Word2Vec 模型以及 Seq2Seq 模型。

在对 seq2seq 模型进行训练前，采用基于 CBOW 方法的 Word2Vec 模型，通过对金庸小说文本进行训练，生成文本信息的编码，用词向量来表示文本信息。

Seq2Seq 模型编码器和解码器均采用 LSTM，在模型的输入和输出前增加线性映射层。

3.3 模型的训练和预测

简单起见，模型的训练 loss 采用计算余弦相似度的方法，即通过衡量预测词向量与目标词向量之间的余弦相似度，若相似度较大，则损失较小，反之亦然。

在模型的预测过程中，通过设定预测结束的条件，即对输出的总词数以及输出句子的数量进行限制，得到最后的输出。该部分参考了[1]的实现方法。

采用《天龙八部》的全部内容作为训练数据，对模型进行训练，共训练 100epoch，采用 SGD 优化器，学习率为 0.01。测试过程中挑选书中的某半句话作为测试输入。

3.4 模型效果

采用《天龙八部》对模型进行训练，并摘取其中某一句话作为引导词，观察模型的输出。

引导词：段誉望望

原文语句：段誉望望王语嫣，又望望阿朱、阿碧，只见三个少女都笑眯眯的听着，显是极感兴味。

模型输出：段誉望望朱四哥，再运羊儿，缝套无意之中吵醒丁老怪。吵醒闪进小虫，这倒确天堂。

引导词：虚竹恍然

原文语句：虚竹心下恍然，知道童姥为了恼他宁死不肯食荤，却去掳了一个少女来，诱得他破了淫戒，不由得又是悔恨，又是羞耻，突然间纵起身来，脑袋疾往坚冰上撞去，砰的一声大响，掉在地下。

模型输出：虚竹心下恍然，铁丑怕羞。朱四哥缝套粗心，腐骨丸无法无天，无意之中痛快小贼，毒得朱四哥饮水。

总体来看，模型的输出语句与金庸风格比较相近，学会了基本的形容词-名词，动词-副词等语法，并且学会了书中的一些特有词汇，比如腐骨丸、铁丑等词的词性和用法。但是，内容上缺乏实际含义，前后语言不搭，说明模型还没有理解语言背后的深层含义。

四、体会感悟

本次大作业用到了 word embedding、seq2seq 等模型，可以说是前几次作业的总结与提升。从传统的词袋模型、LDA 模型到深度学习中的 seq2seq 模型，让对自然语言处理领域的经典算法有了一定的了解。随着 transformer、bert 等更大规模，更强表征的模型的出现，给自然语言处理领域带来了很大的发展变化，机器翻译、语音助手等产品越来越普及，效果也越来越好，越来越体现出自然语言处理的实际生产生活中的作用和价值。

五、参考目录

[1] https://blog.csdn.net/shzx_55733/article/details/117338742

六、代码实现

5.1 main.py

```
import jieba
from sklearn import svm
import numpy as np
from sklearn.model_selection
import train_test_split as ts
import re
from gensim.models import Word2Vec
import torch
import torch.nn as nn
from model import Seq2Seq
from tqdm import trange
import os
import re
import logging
import pdb
logging.basicConfig(format='%(asct
ime)s : %(levelname)s : %(message)
s', level=logging.INFO)

class TraversalFun():

    # 1 初始化
    def __init__(self,
rootDir="../LDA/datasets/"):
        self.rootDir = rootDir

    def TraversalDir(self):
        return
self.getCorpus(self.rootDir)

    def get_sentences(self,
bookname,
rootDir="../LDA/datasets/"):
        if
os.path.exists(os.path.join(rootDi
r, bookname)):
            corpus =
self._getCorpus(bookname, rootDir)
            return
self._get_sentences(corpus)

    def _getCorpora(self,
rootDir):
        '''
        corpus:长度为 16 的 list,每个
元素为储存书内容的字典
        count:书的数量
        '''
        corpus = []
        r1 = u'[a-zA-Z0-
9’!"#$%&\'()*+,-./:：;<=>?@，。?★
、…【】《》？“”‘’！
[\\]^_`{|}~]+'  # 用户也可以在此进
行自定义过滤字符
        listdir =
os.listdir(rootDir)
        count=0
        for file in listdir:
            path  =
os.path.join(rootDir, file)
            if
os.path.isfile(path):
                with
open(os.path.abspath(path), "r",
encoding='gbk', errors="ignore")
as file:
                    filecontext =
file.read()
                    filecontext =
re.sub(r1, '', filecontext)
                    # filecontext =
filecontext.replace("\n", '')
                    filecontext =
filecontext.replace(" ", '')
                    filecontext =
filecontext.replace("\u3000", '')
```

```
                    filecontext =
filecontext.replace("本书来自
www.cr173.com 免费 txt 小说下载站\n
更多更新免费电子书请关注
www.cr173.com", '')
                    filecontext =
filecontext.replace("本书来自免费小
说下载站\n 更多更新免费电子书请关注",
'')
                    #seg_list =
jieba.cut(filecontext,
cut_all=True)
                    #corpus +=
seg_list
                    count +=
len(filecontext)
                    corpus.append(f
ilecontext)
        return corpus,count

    def _getCorpus(self, bookname,
rootDir="../LDA/datasets/"):
        bookpath =
os.path.join(rootDir, bookname)
        r1 = u'[a-zA-Z0-
9’"#$%&\'()*+,-./:: ;<=>?@，★、…
【】《》“”‘’[\\]^_`{|}~]+'  # 用户
也可以在此进行自定义过滤字符
        if
os.path.isfile(bookpath):
            with
open(os.path.abspath(bookpath),
"r", encoding='gbk',
errors="ignore") as file:
                filecontext =
file.read()
                filecontext =
re.sub(r1, '', filecontext)
                filecontext =
filecontext.replace("\n", '')
                filecontext =
filecontext.replace(" ", '')
                filecontext =
filecontext.replace("\u3000", '')
                filecontext =
filecontext.replace("本书来自
www.cr173.com 免费 txt 小说下载站\n
更多更新免费电子书请关注
www.cr173.com", '')
                filecontext =
filecontext.replace("本书来自免费小
说下载站\n 更多更新免费电子书请关注",
'')
                #seg_list =
jieba.cut(filecontext,
cut_all=True)
                #corpus +=
seg_list
        return filecontext

    def _get_sentences(self,
corpus, length=30):
        '''
        sentences:列表，每个元素为句
子
        '''
        sybl = "。|？|！|……"
        sentences_tmp=re.split(syb
l, corpus)
        sentences = list()
        for sentence in
sentences_tmp:
            sentence_split =
list(jieba.cut(sentence,
cut_all=False))
```

```
            if len(sentence_split)
< 5:
                continue
            sentence_split.append(
"END")
            sentences.append(sente
nce_split)
        print("已获得句子与分词")
        return sentences



class W2V(object):
    def __init__(self, sentences,
sg=0, embed_size=300, min_count=1,
window=10, iter=20):
        super().__init__()
        if not
os.path.exists('w2v.model'):
            self.model =
Word2Vec(sentences=sentences,
sg=sg, size=embed_size,
min_count=min_count,
window=window, iter=iter)
            self.model.save('w2v.m
odel')
        else:
            self.model =
Word2Vec.load('w2v.model')
    def get_model(self):
        return self.model

def train(sentences, w2v_model,
seq2seq_model, embed_size=300,
epochs=100, end_num=10):
    optimizer =
torch.optim.SGD(params=seq2seq_mod
el.parameters(), lr=0.01)
    for epoch_id in range(epochs):
        for idx in trange(0,
len(sentences) // end_num - 1):
            seq = []
            for k in
range(end_num):
                seq +=
sentences[idx + k]
            target = []
            for k in
range(end_num):
                target +=
sentences[idx + end_num + k]
            input_seq =
torch.zeros(len(seq), embed_size)
            for k in
range(len(seq)):
                input_seq[k] =
torch.tensor(w2v_model.wv[seq[k]])
            target_seq =
torch.zeros(len(target),
embed_size)
            for k in
range(len(target)):
                target_seq[k] =
torch.tensor(w2v_model.wv[target[k
]])
            all_seq =
torch.cat((input_seq, target_seq),
dim=0)
            optimizer.zero_grad()
            out_res =
seq2seq_model(all_seq[:-1])
            f1 = ((out_res[-
target_seq.shape[0]:] **
2).sum(dim=1)) ** 0.5
```

```
            f2 =
((target_seq.cuda() **
2).sum(dim=1)) ** 0.5
            loss = (1 - (out_res[-
target_seq.shape[0]:] *
target_seq.cuda()).sum(dim=1) / f1
/ f2).mean()
            loss.backward()
            optimizer.step()
        if idx % (epochs-1) == 0:
            print("loss: ",
loss.item(), " in epoch ",
epoch_id, " res: ",out_res[-
target_seq.shape[0]:].max(dim=1).i
ndices,
target_seq.max(dim=1).indices)

    torch.save(seq2seq_model.state
_dict(), "model/" +
"Seq2Seq.pth.tar")

def test(sentences, w2v_model,
seq2seq_model, embed_size=300):
    seqs = list()
    for s in sentences:
        seqs += s
    input_seq =
torch.zeros(len(seqs),
embed_size).cuda()
    result = ""
    with torch.no_grad():
        for k in range(len(seqs)):
            try:
                input_seq[k] =
torch.tensor(w2v_model.wv[seqs[k]]
)
            except:
                continue
        end_num = 0
        length = 0
        while end_num < 10 and
length < 200:
            print("length: ",
length)
            out_res =
seq2seq_model(input_seq)[-1:]
            key_value =
w2v_model.wv.most_similar(positive
=np.array(out_res.cpu()), topn=20)
            key=key_value[0][0]
            if key == "END":
                result += "。"
                end_num += 1
            else:
                result += key
            length += 1
            input_seq =
torch.cat((input_seq, out_res),
dim=0)
    print(result)

def init():
    stop_words = get_words("./人物
武功门派和停词/stop_words.txt");
stop_words.remove('。')
    menpai = get_words("./人物武功
门派和停词/金庸小说全门派.txt")
    renwu = get_words("./人物武功门
派和停词/金庸小说全人物.txt")
    wugong = get_words("./人物武功
门派和停词/金庸小说全武功.txt")
    add_words(menpai);
add_words(renwu);
add_words(wugong)

def add_words(words):
```

```
    for word in words:
        jieba.add_word(word)
    return

def get_words(filename):
    with open(filename, "r", encoding='gbk', errors="ignore") as f:
        words = [word.strip() for word in f.readlines()]
    return words


def main(unse_checkpoint=1):
    init()
    tra = TraversalFun()
    sentences = tra.get_sentences("天龙八部.txt")
    embed_size=300
    w2v_model = W2V(sentences, embed_size=embed_size).get_model()
    seq2seq_model = nn.DataParallel(Seq2Seq(embed_size)).cuda()
    if unse_checkpoint and os.path.exists("./model/Seq2Seq.pth.tar"):
        checkpoint = torch.load("./model/Seq2Seq.pth.tar")
        seq2seq_model.load_state_dict(checkpoint)
    else:
        train(sentences, w2v_model, seq2seq_model, embed_size=embed_size)

    test(tra.get_sentences("test.txt", "./"), w2v_model, seq2seq_model, embed_size=embed_size)


if __name__ == "__main__":
    main(1)
```

5.2 model.py

```
class Seq2SeqEncoder(nn.Module):
    def __init__(self, embed_size, num_hiddens, num_layers,
                 dropout=0, **kwargs):
        super(Seq2SeqEncoder, self).__init__(**kwargs)
        self.linear = nn.Linear(embed_size, embed_size)
        self.rnn = nn.LSTM(embed_size, num_hiddens, num_layers,
                           dropout=dropout)

    def forward(self, X, *args):
        X = self.linear(X)
        X = X.permute(1, 0, 2)
        output, state = self.rnn(X)
        return output, state
```

```
class Seq2SeqDecoder(nn.Module):
    def __init__(self, embed_size,
num_hiddens, num_layers,
                 dropout=0,
**kwargs):
        super(Seq2SeqDecoder,
self).__init__(**kwargs)
        self.rnn =
nn.LSTM(embed_size, num_hiddens,
num_layers,
                          dropout=dro
pout)
        self.dense =
nn.Linear(num_hiddens, embed_size)

    def init_state(self,
enc_outputs, *args):
        return enc_outputs[1]

    def forward(self, X, state):
        X =
self.embedding(X).permute(1, 0, 2)
        output, state = self.rnn(X,
state)
        output =
self.dense(output).permute(1, 0, 2)
            return output, state
```