NLP-5

PT2100565 罗志钊

NLP-5
- 问题描述
- 预备知识
  - seq2seq框架
    - Encoder
    - Decoder
    - 加入attention机制后
- 实验过程
- 实验结果
- 心得
- 参考内容
- 附录
  - 实验代码
    - modelT.py
    - textI.py

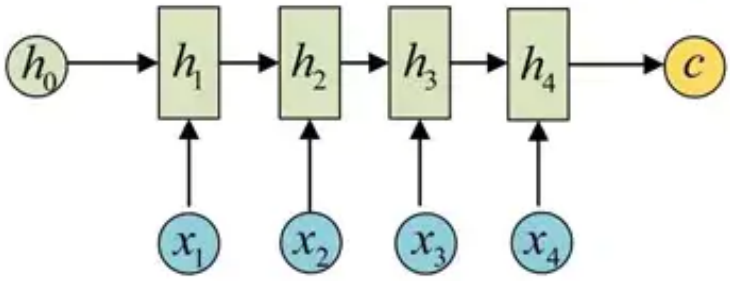
问题描述

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。

预备知识

seq2seq框架

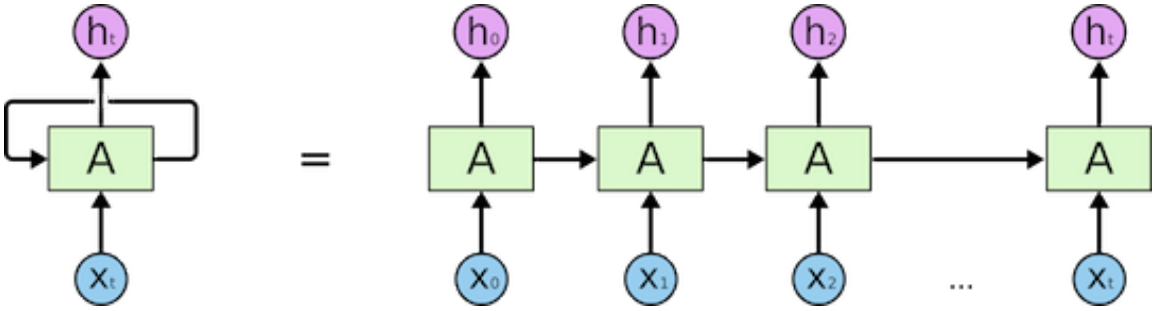
seq2seq是sequence to sequence的缩写，作用是将一个序列映射成另一个序列。seq2seq有两种应用NMT和RNN。同时，seq2seq框架中包含两个模块，encoder模块和decoder模块。在这里我主要用rnn，所以基于rnn我会展开说一下encoder和decoder。



Encoder

在前向计算中，我们传入源语句，并使用嵌入层将其转换为密集向量，然后应用dropout。 然后将这些嵌入传递到RNN。 当我们将整个序列传递给RNN时，它将为我们自动对整个序列进行隐藏状态的递归计算。请注意，我们没有将初始的隐藏状态或单元状态传递给RNN。

一个典型的RNN结构如下图



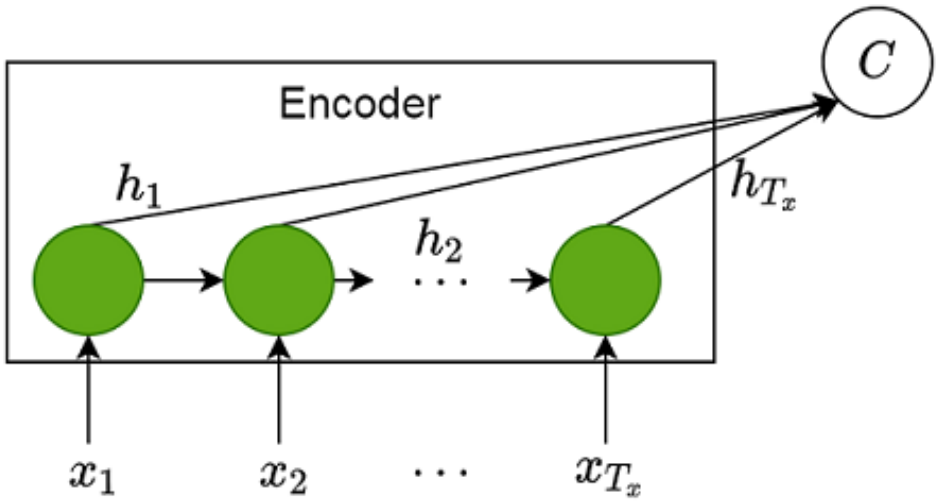
在RNN中，当前时刻 t 的隐含层状态h_t 是由上一时刻 t−1的隐含层状态h_{t-1}和当前时刻的输入x_t共同决定的，可由下式表示：

$$h_t = f(h_{t-1}, x_t)$$

假设在seq2seq框架中，输入序列为x={x1,x2.....x_Tx}，其中x_i属于自然数，输出序列为y={y1,y2,y3.....y_Ty}，其中y_i也属于自然数。在编码阶段，rnn通过学习到每个时刻的隐含层状态后，最终得到所有隐含层状态序列：

$$h_1, h_2, \ldots, h_{T_x}$$

具体过程可由下图表示：



通过对这些隐藏层的状态进行汇总，得到上图中固定长度的语义编码向量C，如下式所示：

$$C = f(h_1, h_2, \ldots, h_{T_x})$$

其中 f 表示某种映射函数。通常取最后的隐含层状态h_Tx 作为语义编码向量C ，即

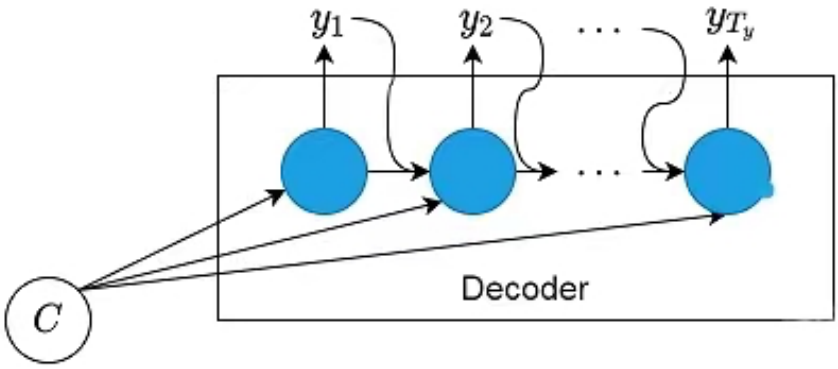
$$C = f(h_1, h_2, \ldots, h_{T_x}) = h_{T_x}$$

Decoder

解码器执行解码的单个步骤，即，每个时间步输出单个token。 第一层将从上一时间步中接收到一个隐藏的单元格状态，并将其与当前嵌入的token一起通过LSTM馈送，以产生一个新的隐藏的单元格状态。 后续层将使用下一层的隐藏状态，以及其图层中先前的隐藏状态和单元格状态。

Decoder的参数与encoder类似，但要注意它们的hid_dim要相同，否则矩阵无法运算。

在解码阶段，在当前时刻t tt，根据在编码阶段得到的语义向量c和已经生成的输出序列y1,y2...y_t-1 ，来预测当前的输出的 y_t，其具体过程可由下图表示：



上述过程可用下面这个公式表示

$$y_t = argmax P(y_t) = \prod_{t=1}^{T} p(y_t|y_1, y_2, \ldots y_{t-1}, c)$$

简化可得：

$$y_t = f(y_t|y_1, y_2, \ldots, y_{t-1}, c)$$

其中f表示某种映射函数。在RNN中，上面公式继续简化为：

$$y_t = f(y_{t-1}, s_{t-1}, c)$$

其中y*{t-1}表示t-1时刻的输出，*s{t-1}表示decoder中rnn在t-1时刻的神经元的隐含层的状态，c代表的是encoder网络生成的语义向量。

加入attention机制后

与原始的Encoder-Decoder模型相比，加入Attention机制后最大的区别就是原始的Encoder将所有输入信息都编码进一个固定长度的向量之中。而加入Attention后，Encoder将输入编码成一个向量的序列，在Decoder的时候，每一步都会选择性的从向量序列中挑选一个集合进行进一步处理。这样，在产生每一个输出的时候，都能够做到充分利用输入序列携带的信息。

实验过程

1. 实现解码器

   我在这里直接使用编码器最后的一个时间步的隐藏状态来初始化。同时要求rnn编码器和rnn解码器具有相同数量的层和隐藏单位。同时，为了对输出令牌的概率分布进行预测，在此使用完全连接的层来转换rnn解码器最后一层的状态。

```
class Net(nn.Module):
    def __init__(self, onehot_num):
        super(Net, self).__init__()
        onehot_size = onehot_num
        embedding_size = 256
        n_layer = 2
        self.lstm = nn.LSTM(embedding_size, embedding_size, n_layer,
batch_first=True)
```

```
        self.encode = torch.nn.Sequential(nn.Linear(onehot_size,
embedding_size),nn.Dropout(0.5),nn.ReLU())
        self.decode = torch.nn.Sequential(nn.Linear(embedding_size,
onehot_size),nn.Dropout(0.5),nn.Sigmoid())

    def forward(self, x):
        em = self.encode(x).unsqueeze(dim=1)
        out, (h, c) = self.lstm(em)
        res = 2*(self.decode(out[:,0,:])-0.5)
        return res
```

2. 读取数据

   将主要分割句子的标点符号（逗号，感叹号和问号）全部转换成句号，用来划分整个句子。其他和之前实验相同，把其余标点符号用空格替换。同时，为了避免过短的段落影响，在此把字数少于200字的段落去掉，保留大段。

```
def dataR():
    with open('./data/射雕英雄传.txt', "r", encoding='utf-8') as f:
        file_read = f.readlines()
        data0 = ""
        for line in file_read:
            # 由于标点符号过多，生成训练效果不好，所以将所有结束的标点符号，比如感叹号问号全部替
换成句号
            line = re.sub('! ', '。', line)
            line = re.sub(', ', '。', line)
            line = re.sub('? ', '。', line)
            line = re.sub('\s','', line)
            line = re.sub('[\u0000-\u3001]', '', line)
            line = re.sub('[\u3003-\u4DFF]', '', line)
            line = re.sub('[\u9FA6-\uFFFF]', '', line)
            if len(line) < 200:
                continue
            data0 += line
        f.close()
    return data0
```

3. word2vec将分词转化为向量

```
# 获得word2vec模型
model = Word2Vec(sentences=wordInText,sg=0, vector_size=1024, min_count=1,
window=10, epochs=10)
model.save('model.model')
```

4. 载入向量模型进行训练

   进行LSTM神经网络训练

```
# lstm训练
sequences = wordInText
vectorInM = Word2Vec.load('model.model')
model = Net(1024).to(device)
optimizer = torch.optim.AdamW(params=model.parameters(), lr=0.0001)

for epochID in range(30):
    #迭代30次训练模型
```

5. 根据上文输出下文

调用最后迭代生成的模型进行seq2seq输出，输入：

```
黄药师不答，向陆冠英一指道："他是你儿子？"陆乘风道："是。"陆冠英不待父亲吩咐，忙上前恭恭敬敬的磕了四个头，说道："孙儿叩见。"黄药师道："罢了！"并不俯身相扶，却伸左手抓住他后心一提，右掌便向他肩头拍落。陆乘风大惊，叫道："恩师，我就只这个儿子……"黄药师这一掌劲道不小，陆冠英肩头被击后站立不住，退后七八步，再是仰天一交跌倒，但丝毫损伤，怔怔的站起身来。黄药师对陆乘风道："你很好，没把功夫传他。这孩子是仙霞派门下的吗？"陆乘风才知师父这一提一推，是试他儿子的武功家数，忙道："弟子不敢违了师门规矩，不得恩师准，决不敢将恩师的功夫传授旁人。这孩子正是拜在仙霞派枯木大师的门下。"黄药师冷笑一声，道："枯木这点微末功夫，也称甚么大师？你所学胜他百倍，打从明天起，你自己传儿子功夫罢。仙霞派的武功，跟咱们提鞋子也不配。"陆乘风大喜，忙对儿子道："快，快谢过祖师爷的恩典。"陆冠英又向黄药师磕了四个头。黄药师昂起了头，不加理睬。
```

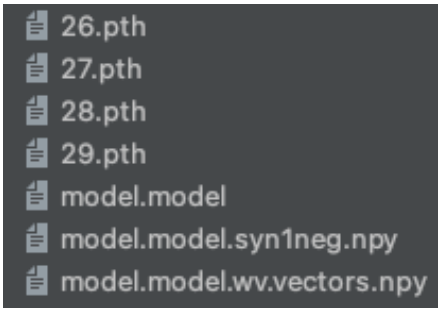
测试模型

由于训练模型迭代次数有限，为了使得最后输出的句子不都是相同的词。这里选取顶端30个分词进行挑选输出。

```
with torch.no_grad():
    for i in range(len(seqs)):
        input_seq[i] = torch.tensor(vec_model.wv[seqs[i]])
    end_num = 0
    length = 0
    while end_num < 10 and length < 100:
        out_res = model(input_seq.to(device))[-2:-1]
        key_value = vec_model.wv.most_similar(positive=np.array(out_res.cpu()),
topn=30)
        print(key_value)
        key = key_value[np.random.randint(30)][0]
        if key == "stopRead":
            result += "。"
            end_num += 1
        else:
            result += key
        length += 1
        input_seq = torch.cat((input_seq, out_res), dim=0)
print("下文：")
print(result)
```

实验结果

1. 训练内容



2. 测试内容及输出文本

```
Building prefix dict from the default dictionary ...
Loading model from cache /var/folders/0z/9vcjblg50qqcrgw4bnr11tkw0000gn/T/jieba.cache
Loading model cost 0.444 seconds.
Prefix dict has been built successfully.
上文:
黄药师不答。向陆冠英一指道他是你儿子。陆乘风道是。陆冠英不待父亲吩咐。忙上前恭恭敬敬的磕了四个头。说道孙儿叩见。黄
[('发', 0.9912098050117493), ('穆易', 0.9911790490150452), ('互相', 0.9911445379257202), ('小孩',
[('发', 0.9912098050117493), ('穆易', 0.9911790490150452), ('互相', 0.9911445379257202), ('小孩',
```

输出文本（result）：

> 适才已到心头。要害闪避香珠重。反过来暗器不免屋子里。分一夜决。不早就这场。一口气别的这场北地。凡是居然决不一口气。一口气此后分必定怪。不易别的必定凡是奸人。此后分一夜。此后就此是否一场全数吃惊。耳朵早就早就一场不易。居然内讧。是否居然一口气必定奸人事先此后。不但决不无人一场。早就分一夜。难以蕴藉怪练习。凡是就此吃惊。必定全数北地。

心得

1. 由于我电脑中没有gpu，这里选用cpu进行模型训练，考虑到时间问题，这里只迭代训练了30次，因而模型学习的内容还稍有不足。
2. 对于标点符号过多，这里把全部句子用句号分割，最后生成文本也是由句号分割开，虽然能确定是句子，但是可读性稍差。
3. 从输出的结果来看，和射雕英雄传文风有一定相似之处。同时语段中出现的部分词，例如暗器、奸人、一夜、练习、吃惊等，都和输入文本有关联，虽然整段意义不明，但一些关键词一定程度上可以体现模型的精确。
4. 对于优化，这里我有两个方向，第一个方向首先可以提高迭代次数，与此同时可以减少顶端输出的词数量（例如测试模型中的topn=30，可以修改为topn=5，甚至更小）。因为随着迭代次数的增多，模型的学习更为可信，理想情况下可以设置为1，更为精准。

   第二个方向是，选用textgenrnn包，可惜macos平台不能使用，如果是其他平台可以尝试。textgenrnn工具包是一种现代的神经网络结构，利用了attention-weighting和skip-embedding等技术；同时可以配置rnn大小、层数和是否使用双向rnn。

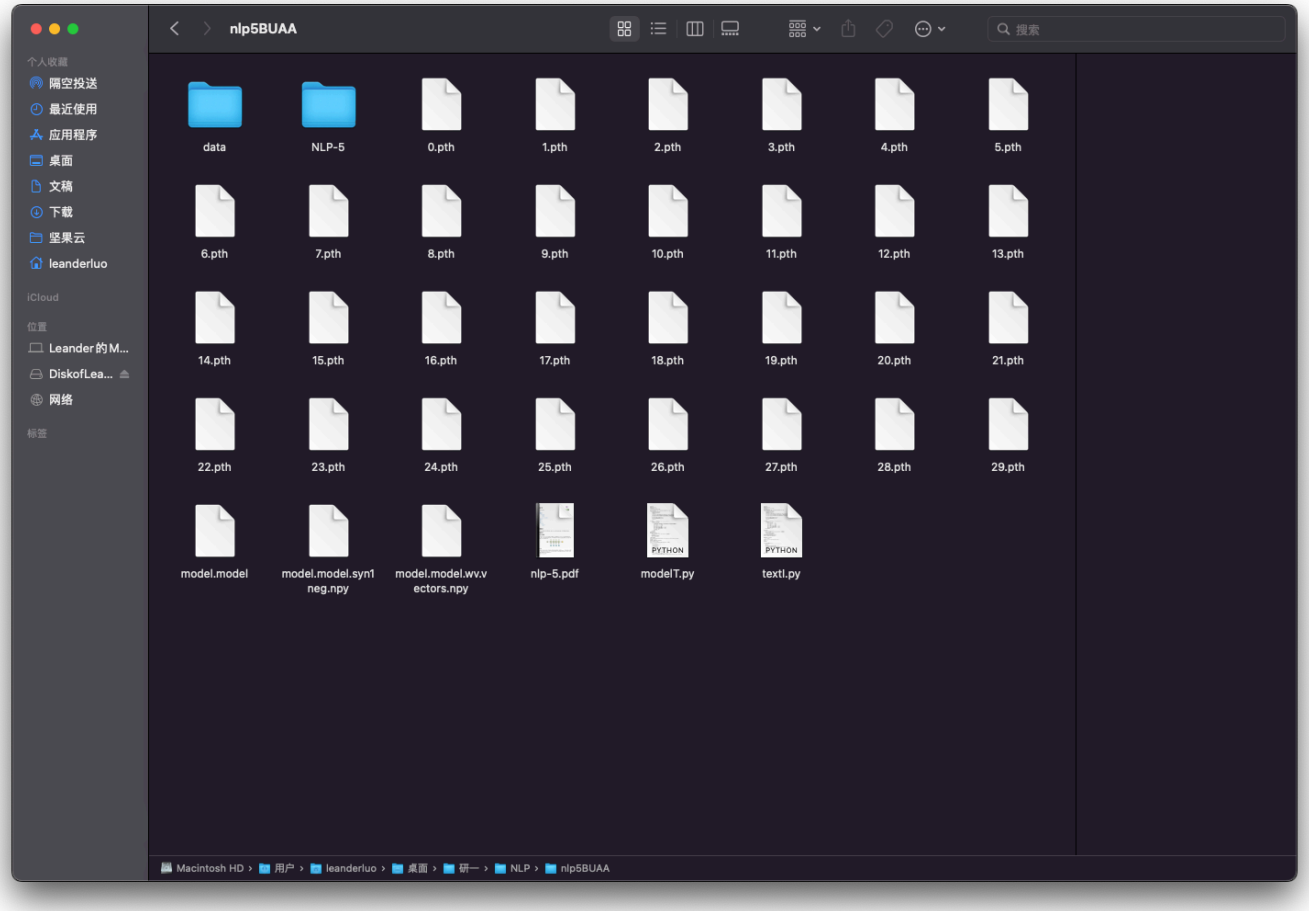
参考内容

1. https://blog.csdn.net/weixin_42663984/article/details/117068473

2. https://blog.csdn.net/google19890102/article/details/108785950

附录

部分生成模型文件过大，超过了100m，故不上传github，原始文件夹列表如下。



实验代码

modelT是用来生成模型；testI用来对input的text进行处理生成对应的小说语段。

modelT.py

```
import torch
import torch.nn as nn
import re
import jieba
from tqdm import trange
from gensim.models import Word2Vec

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
class Net(nn.Module):
    def __init__(self, onehot_num):
        super(Net, self).__init__()
        onehot_size = onehot_num
```

```
        embedding_size = 256
        n_layer = 2
        self.lstm = nn.LSTM(embedding_size, embedding_size, n_layer, batch_first=True)
        self.encode = torch.nn.Sequential(nn.Linear(onehot_size,
embedding_size),nn.Dropout(0.5),nn.ReLU())
        self.decode = torch.nn.Sequential(nn.Linear(embedding_size,
onehot_size),nn.Dropout(0.5),nn.Sigmoid())

    def forward(self, x):
        em = self.encode(x).unsqueeze(dim=1)
        out, (h, c) = self.lstm(em)
        res = 2*(self.decode(out[:,0,:])-0.5)
        return res

def dataR():
    with open('./data/射雕英雄传.txt', "r", encoding='utf-8') as f:
        file_read = f.readlines()
        dataO = ""
        for line in file_read:
            # 由于标点符号过多，生成训练效果不好，所以将所有结束的标点符号，比如感叹号问号全部替换成句
号
            line = re.sub('! ', '。', line)
            line = re.sub(', ', '。', line)
            line = re.sub('? ', '。', line)
            line = re.sub('\s','', line)
            line = re.sub('[\u0000-\u3001]', '', line)
            line = re.sub('[\u3003-\u4DFF]', '', line)
            line = re.sub('[\u9FA6-\uFFFF]', '', line)
            if len(line) < 200:
                continue
            dataO += line
        f.close()
    return dataO
dataO = dataR()
wordInText = list()
for textL in dataO.split('。'):
    segmentL = list(jieba.cut(textL, cut_all=False))  # 使用精确模式
    if len(segmentL) < 5:
        continue
    segmentL.append("stopRead")
    wordInText.append(segmentL)

# 获得word2vec模型
model = Word2Vec(sentences=wordInText,sg=0, vector_size=1024, min_count=1, window=10,
epochs=10)
model.save('model.model')
# lstm训练
sequences = wordInText
vectorInM = Word2Vec.load('model.model')
```

```
model = Net(1024).to(device)
optimizer = torch.optim.AdamW(params=model.parameters(), lr=0.0001)

for epochID in range(30):
    for i in trange(0, len(sequences) // 10 - 1):
        seq = []
        for j in range(10):
            seq += sequences[i + j]
        target = []
        for h in range(10):
            target += sequences[i + 10 + h]
        seqInput = torch.zeros(len(seq), 1024)
        for m in range(len(seq)):
            seqInput[m] = torch.tensor(vectorInM.wv[seq[m]])
            seqTar = torch.zeros(len(target), 1024)
        for n in range(len(target)):
            seqTar[n] = torch.tensor(vectorInM.wv[target[n]])
        seqTotal = torch.cat((seqInput, seqTar), dim=0)
        optimizer.zero_grad()
        resOutput = model(seqTotal[:-1].to(device))
        funcA = ((resOutput[-seqTar.shape[0]:] ** 2).sum(dim=1)) ** 0.5
        funcB = ((seqTar.to(device) ** 2).sum(dim=1)) ** 0.5
        lossFun = (1 - (resOutput[-seqTar.shape[0]:] * seqTar.to(device)).sum(dim=1) /
funcA / funcB).mean()
        lossFun.backward()
        optimizer.step()
        if i % 50 == 0:
            print("ID: ", epochID,"LossFun: ", lossFun.item(), " Res: ",resOutput[-
seqTar.shape[0]:].max(dim=1).indices, seqTar.max(dim=1).indices)
    state = {"models": model.state_dict()}
    torch.save(state, str(epochID) + ".pth")
```

textI.py

```
import torch
import torch.nn as nn
import re
import jieba
from gensim.models import Word2Vec
import numpy as np
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
class Net(nn.Module):
    def __init__(self, onehot_num):
        super(Net, self).__init__()
        onehot_size = onehot_num
        embedding_size = 256
        n_layer = 2
```

```
        self.lstm = nn.LSTM(embedding_size, embedding_size, n_layer, batch_first=True)#
编码
        self.encode = torch.nn.Sequential(nn.Linear(onehot_size,
embedding_size),nn.Dropout(0.5),nn.ReLU())# 解码
        self.decode = torch.nn.Sequential(nn.Linear(embedding_size,
onehot_size),nn.Dropout(0.5),nn.Sigmoid())

    def forward(self, x):# 入
        em = self.encode(x).unsqueeze(dim=1)# 出
        out, (h, c) = self.lstm(em)
        res = 2*(self.decode(out[:,0,:])-0.5)
        return res

def dataR():
    with open('./data/test.txt', "r", encoding='utf-8') as f:
        file_read = f.readlines()
        textInput = ""
        for line in file_read:
            line = re.sub('\s', '', line)
            line = re.sub(', ', '。', line)
            line = re.sub('! ', '。', line)
            line = re.sub('? ', '。', line)
            line = re.sub('[\u0000-\u3001]', '', line)
            line = re.sub('[\u3003-\u4DFF]', '', line)
            line = re.sub('[\u9FA6-\uFFFF]', '', line)
            textInput += line
    return textInput

text = dataR()
wordInText = []
for text_line in text.split('。'):
    seg_list = list(jieba.cut(text_line,cut_all=False))  # 使用精确模式
    if len(seg_list) < 5:
        seg_list.append("stopRead")
        wordInText.append(seg_list)
finalItera = torch.load(str(29) + ".pth")
model = Net(1024).eval().to(device)
model.load_state_dict(finalItera["models"])
vec_model = Word2Vec.load('model.model')

seqs = []
for sequence in wordInText:
    seqs += sequence

input_seq = torch.zeros(len(seqs), 1024).to(device)
result = ""
print("上文: ")
print(text)
```

```
with torch.no_grad():
    for i in range(len(seqs)):
        input_seq[i] = torch.tensor(vec_model.wv[seqs[i]])
    end_num = 0
    length = 0
    while end_num < 10 and length < 100:
        out_res = model(input_seq.to(device))[-2:-1]
        key_value = vec_model.wv.most_similar(positive=np.array(out_res.cpu()),
topn=30)
        print(key_value)
        key = key_value[np.random.randint(30)][0]
        if key == "stopRead":
            result += "。"
            end_num += 1
        else:
            result += key
        length += 1
        input_seq = torch.cat((input_seq, out_res), dim=0)
print("下文：")
print(result)
```