

机器学习实验报告

学院: 软件学院

专业: 软件工程

|  |  |
| --- | --- |
| *作者:*  周治平，谭嘉展，宋文林 | *指导教师:*  吴庆耀 |
|  |  |
| *学号:*  202030483178  202030481488  202030484052 | *班级:*  20软件工程1班 |

2020-10-24

# 基于序列模型的中英文翻译机

**摘要**

RNN的经典的用例之一就是将文本从一种语言翻译成另一种语言。过去，这是使用手工制作的功能以及许多复杂的条件来完成的，这些条件花费很长时间才能创建并且理解起来很复杂。因此我们需要建立机器学习模型来完成机器翻译的工作。

这里使用简单的模型（[编码器](https://so.csdn.net/so/search?q=%E7%BC%96%E7%A0%81%E5%99%A8&spm=1001.2101.3001.7020)-解码器模型），其最简单的想法是使用基本的RNN

编码器：我们首先计算输入热词向量的词嵌入。 然后我们发送嵌入的单词进行翻译。我们的（嵌入词向量）乘以一些权重矩阵W（hx）。我们先前计算的隐藏状态（这是RNN节点的先前输出）乘以不同的权重矩阵W（hh）。然后将这两个乘法的结果相加，并应用非线性（如Relu / tanh）。现在这是我们的下一个隐藏状态h。然后在输入句子的整个过程中重复此过程。 显然，在第一个输入字x0上没有先前的隐藏状态，因此我们仅将此h0设置为全零。

另请注意，我们的句子的长度可能不同，因此我们还必须有一个停止标记（例如句号），该标记表示我们已经到达了句子的结尾。 我们希望我们的模型将学习何时预测输出的停止令牌。 停止令牌基本上只是我们训练数据中的一个额外“单词”。

解码器：一旦到达停止令牌，便进入解码器RNN节点以开始产生输出矢量。为了在每个时间步长从解码器RNN获得输出y，我们有另一个权重矩阵W（S），我们将其隐藏状态h相乘以获得矢量输出。 然后将softmax应用于此，这将为我们提供最终输出。 此最终输出告诉您在该时间步预测了什么单词向量。

该模型非常简单，实际上只能用于非常排序的句子（2-3个单词）。 这是因为这些基本的RNN难以记住过去的多个步骤（消失的梯度问题）。

## **介绍**

1. 探索自然语言处理
2. 理解经典的Sequence-to-Sequence机器翻译模型
3. 掌握Attention机制在机器翻译模型上的应用
4. 搭建机器翻译模型，在简单小规模数据集上验证模型性能，培养工程能力
5. 了解Transformer在机器翻译任务上的应用

## **方法和理论**

1.RNN循环神经网络模型

1.1模型概述

RNN是一种特殊的神经网络结构, 它是根据"人的认知是基于过往的经验和记忆"这一观点提出的. 它与DNN,CNN不同的是: 它不仅考虑前一时刻的输入,而且赋予了网络对前面的内容的一种'记忆'功能.

RNN之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式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，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。

1.2模型结构

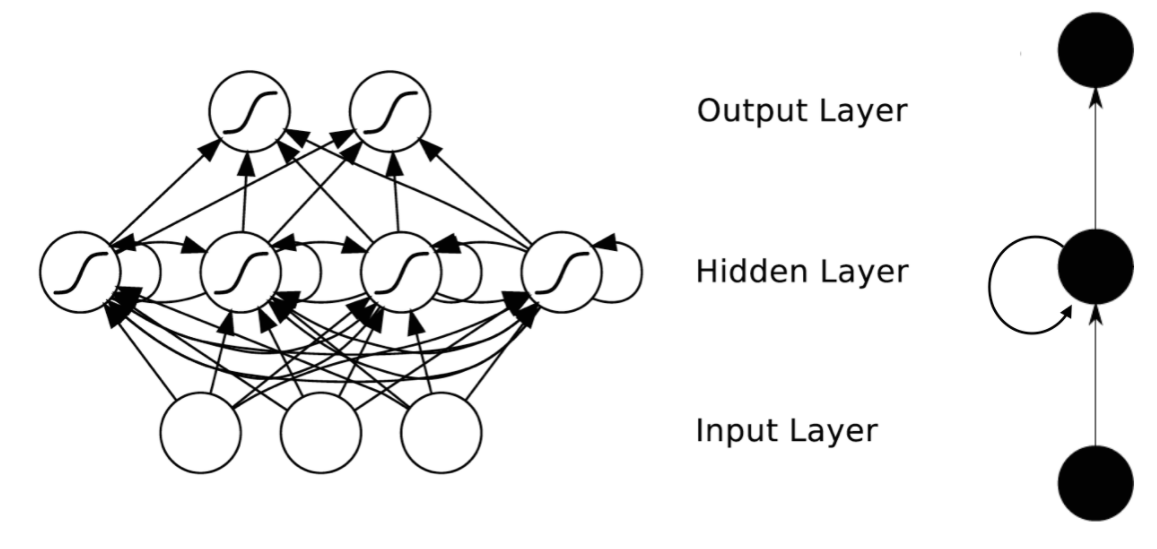


Figure. 1. RNN网络结构

如Figure.1所示, 我们可以看到RNN层级结构较之于CNN来说比较简单, 它主要有输入层,Hidden Layer, 输出层组成。

并且会发现在Hidden Layer 有一个箭头表示数据的循环更新, 这个就是实现时间记忆功能的方法.

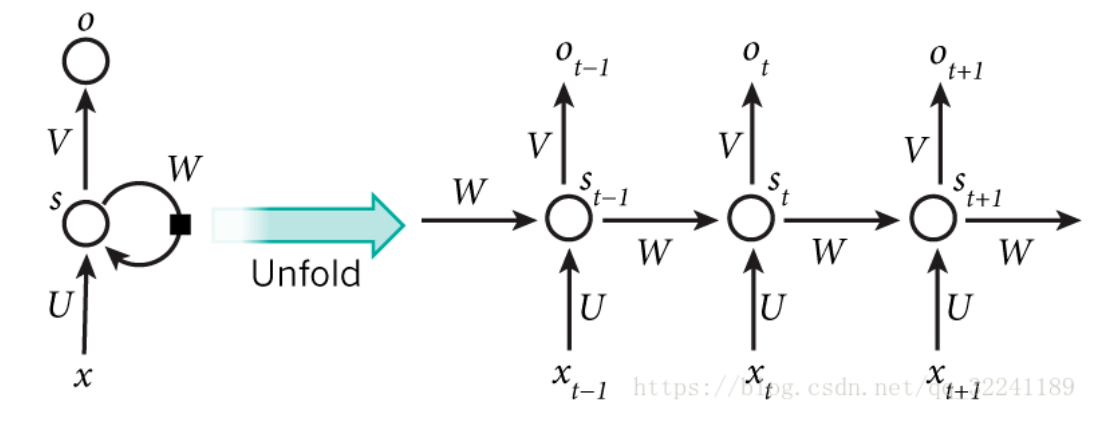


Figure. 2. RNN隐藏层层级展开

如Figure.2所示为Hidden Layer的层级展开图. t-1, t, t+1表示时间序列. X表示输入的样本. St表示样本在时间t处的的记忆,。 W表示输入的权重, U表示此刻输入的样本的权重, V表示输出的样本权重。

最终可推导出：



1.3模型反向传播

每一次的输出值都会产生一个误差值, 则总的误差可以表示为: 

则损失函数可以使用交叉熵损失函数也可以使用平方误差损失函数.

由于每一步的输出不仅仅依赖当前步的网络，并且还需要前若干步网络的状态，那么这种BP改版的算法叫做Backpropagation Through Time(BPTT) , 也就是将输出端的误差值反向传递,运用梯度下降法进行更新。



2 Encoder-Decoder模型

2.1模型介绍

Encoder-Decoder（编码-[解码](https://so.csdn.net/so/search?q=%E8%A7%A3%E7%A0%81&spm=1001.2101.3001.7020)）是深度学习中非常常见的一个模型框架，比如无监督算法的auto-encoding就是用编码-解码的结构设计并训练的；比如这两年比较热的image caption的应用，就是CNN-RNN的编码-解码框架；再比如神经网络机器翻译NMT模型，往往就是LSTM-LSTM的编码-解码框架。因此，准确的说，Encoder-Decoder并不是一个具体的模型，而是一类框架。Encoder和Decoder部分可以是任意的文字，语音，图像，视频数据，模型可以采用CNN，RNN，BiRNN、LSTM、GRU等等。所以基于Encoder-Decoder，我们可以设计出各种各样的应用算法。

Encoder-Decoder框架有一个最显著的特征就是它是一个End-to-End学习的算法；本文将以文本-文本的例子作为介绍，这样的模型往往用在[机器翻译](https://so.csdn.net/so/search?q=%E6%9C%BA%E5%99%A8%E7%BF%BB%E8%AF%91&spm=1001.2101.3001.7020" \t "_blank)中，比如将中文翻译成英语。这样的模型也被叫做 Sequence to Sequence learning。所谓编码，就是将输入序列转化成一个固定长度的向量；解码，就是将之前生成的固定向量再转化成输出序列。 如Figure.3所示：

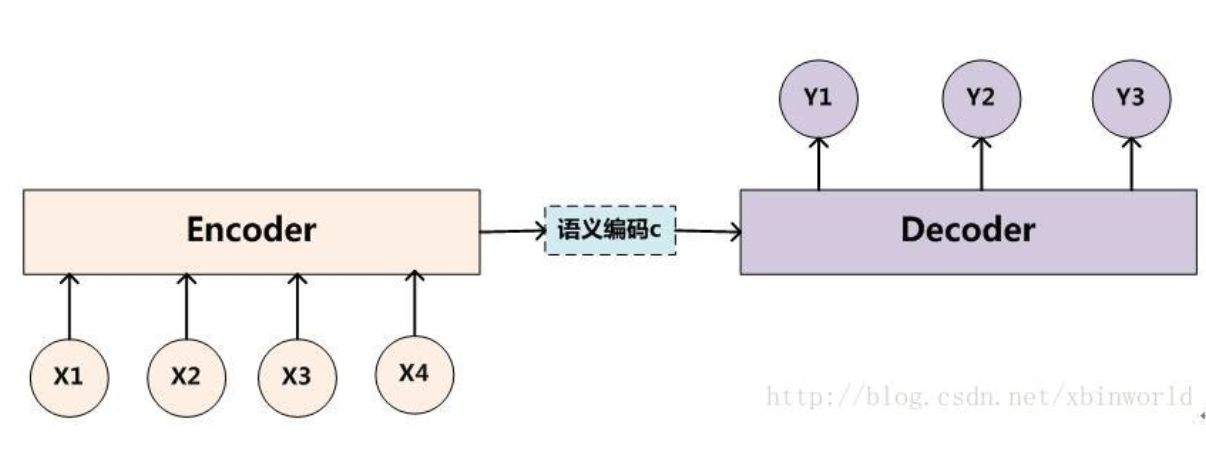


Figure. 3. Encoder-Decoder模型框架

2.2模型结构

在RNN中，当前时刻隐层状态是由上一时刻的隐层状态和当前时刻的输入决定的，也就是：



获得了各个时刻的隐层状态以后，再将信息汇总，生成最后的语义编码c：



q表示某种非线性函数。在[1]中，作者采用LSTM网络作为Encoder网络，实际上在LSTM或者基本的RNN网络中，当前时刻计算完后是看不见前面时刻的隐层状态的了，所以就是用最后一个时刻的隐层状态作为语义编码c，即：



解码过程我们要根据给定的语义编码c和已经生成的输出序列y1,y2,……,yt-1来预测下一个输出的单词yt，实际上就是把生成句子y={y1,y2,..yt-1}的联合概率分解成按顺序的条件概率：



而每一个条件又可以写成：



其中是输出RNN中的隐藏层，c代表之前提过的语义向量，表示上个时刻的输出。g表示一种非线性变换，往往就是指一种多层的函数，可以输出的概率(比如多层RNN后接softmax)。

所以，在文本序列的Encoder-Decoder模型中，原本RNN(LSTM)语言模型是要估计，给定一串输入x，得到一串输出y(不需要等长)，但是因为Encoder-Decoder中间用语义编码c把前后两部分隔开了，所以输出句子y只需要和c相关即可。

3Attention模型

3.1模型介绍

该模型在产生输出的时候，还会产生一个“注意力范围”表示接下来输出的时候要重点关注输入序列中的哪些部分，然后根据关注的区域来产生下一个输出，如此往复。模型的大概示意图如Figure.4所示

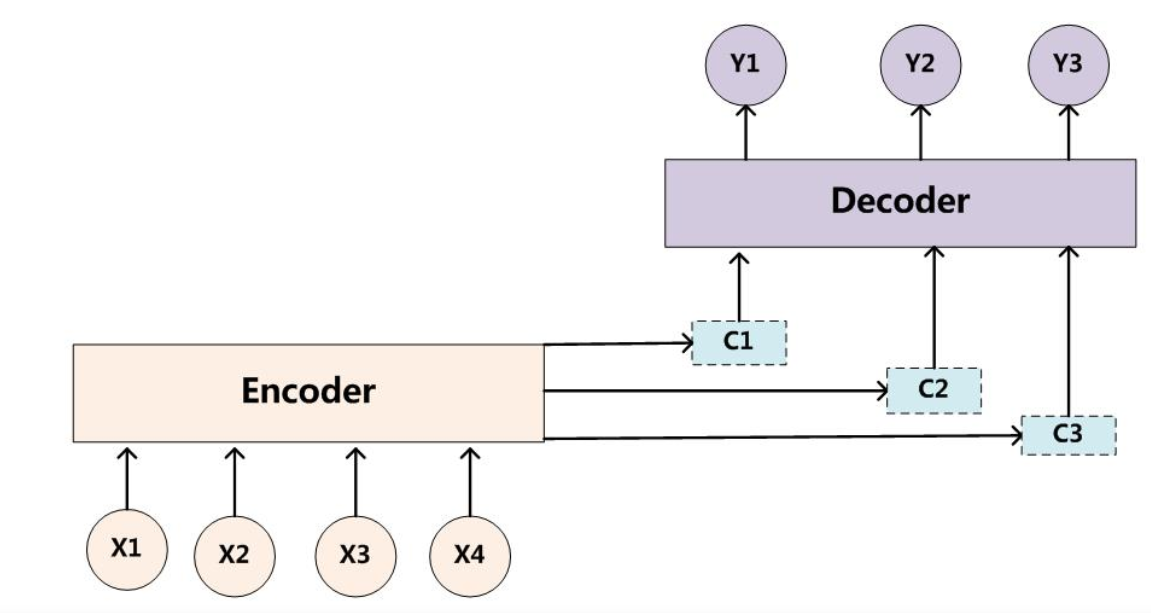


Figure. 4. Attention模型框架

相比于之前的encoder-decoder模型，attention模型最大的区别就在于它不在要求编码器将所有输入信息都编码进一个固定长度的向量之中。相反，此时编码器需要将输入编码成一个向量的序列，而在解码的时候，每一步都会选择性的从向量序列中挑选一个子集进行进一步处理。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。而且这种方法在翻译任务中取得了非常不错的成果。

3.2模型结构

解码部分使用了attention模型。类似的，我们可以将之前定义的条件概率写作：



上式解码器表示解码器t时刻的隐藏状态，计算公式是：



这里的条件概率与每个目标输出相对应的内容向量有关。而在传统的方式中，只有一个内容向量C。那么这里的内容向量 由编码时的隐藏向量序列按权重相加得到：



由于编码使用了双向RNN，因此可以认为中包含了输入序列中第t个词以及前后一些词的信息。将隐藏向量序列按权重相加，表示在生成第j个输出的时候的注意力分配是不同的。的值越高，表示第t个输出在第j个输入上分配的注意力越多，在生成第t个输出的时候受第j个输入的影响也就越大。

此时：





也就是说，先跟每个h分别计算得到一个数值，然后使用softmax得到t时刻的输出在Tx个输入隐藏状态中的注意力分配向量。这个分配向量也就是计算的权重。我们现在再把公式按照执行顺序汇总一下：



最终可解释为如Figure.5示意图

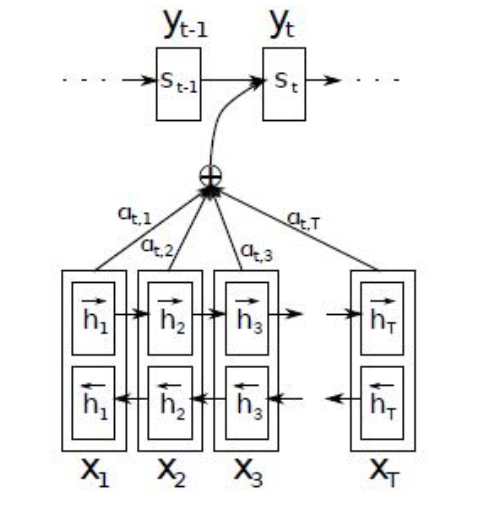


Figure. 5. Attention模型的图形化解释

4损失函数定义

在本次实验中，编码器与解码器均运用了NLLLOSS函数作为损失函数：



其中x是输入，y是输出，w是权重，N是批处理大小，默认是mean处理方法。本实验中使用的是mean处理方法。

5参数优化器定义

本实验中对编码器与解码器均使用SGD优化方法。

输入：



输出：

随机挑选一批样本为依据的损失函数对的导数。



## **实验**

### 数据集

1数据内容：

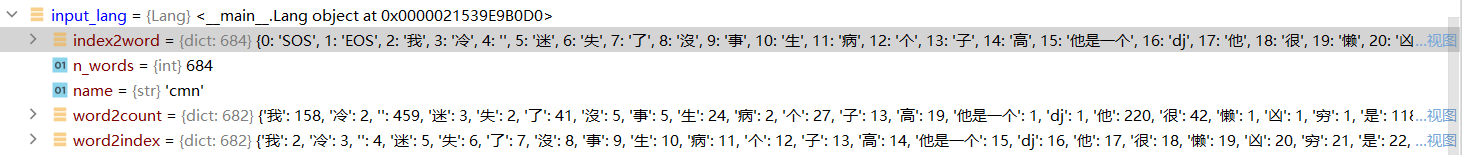


Figure. 6. 中文序列数据集

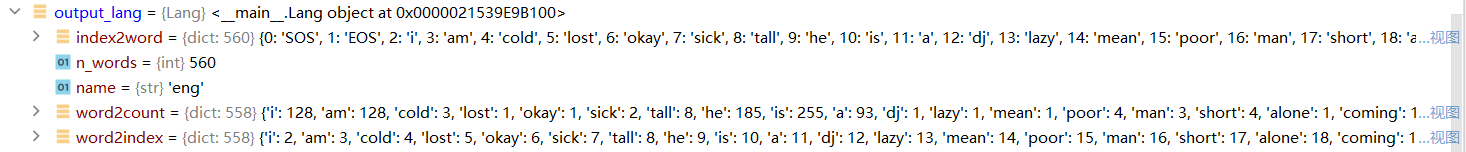


Figure. 7. 英文序列数据集

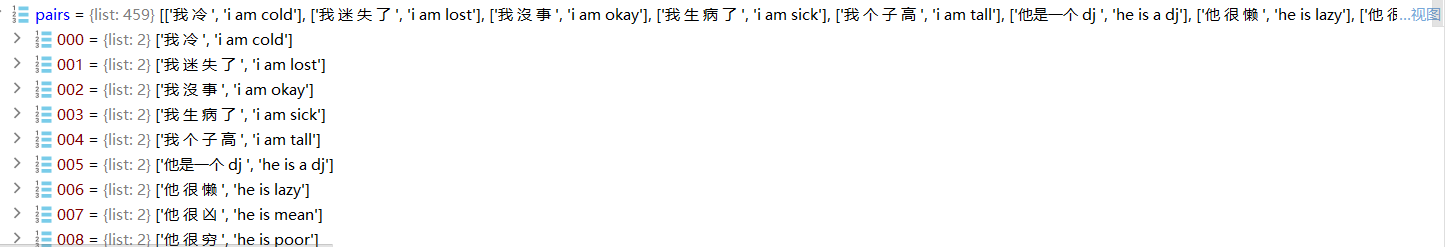


Figure. 8. 中翻英序列数据集

2数据数量

中文单词数据集为684个样本，英文单词数据集为560个样本，中文语句-英文语句转换序列有459个样本。

3训练数据集

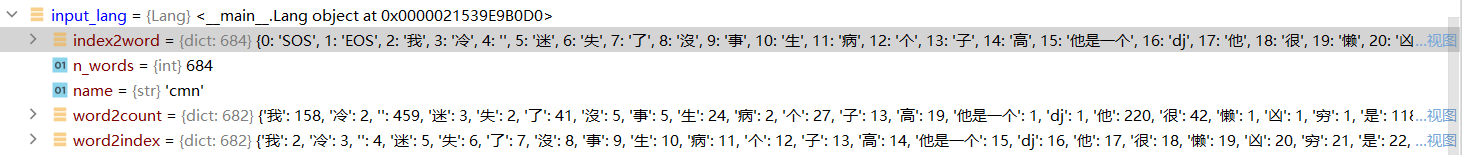


Figure. 9. 中文序列数据集

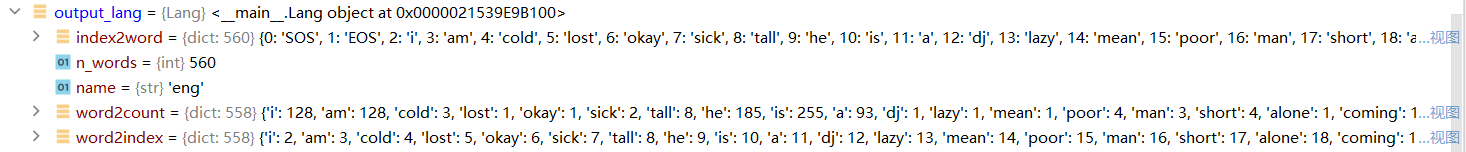


Figure. 10. 英文序列数据集

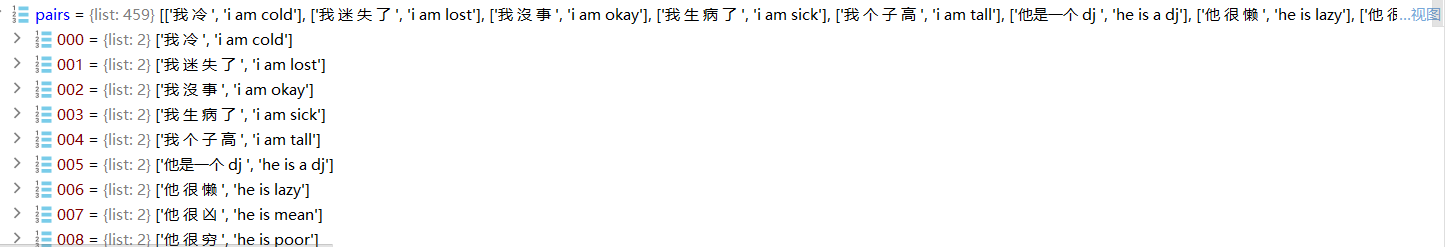


Figure. 11. 中翻英序列数据集

4测试数据集

测试数据集为从训练数据集中（459个中翻英样本）中随机挑选100个中翻英样本进行测试。

### 实现

1导入环境包

**from** \_\_future\_\_ **import** unicode\_literals, print\_function, division  
**from** io **import** open  
**import** unicodedata  
**import** string  
**import** re  
**import** random  
  
**import** torch  
**import** torch.nn **as** nn  
**from** torch **import** optim  
**import** torch.nn.functional **as** F  
  
**import** numpy **as** np  
**import** jupyter  
**import** matplotlib  
**import** matplotlib.pyplot **as** plt  
**import** nltk  
**from** nltk.translate.bleu\_score **import** sentence\_bleu  
device = torch.device(**"cuda"** **if** torch.cuda.is\_available() **else** **"cpu"**)

2导入数据

2.1定义数据加载类

与字符级RNN教程中使用的字符编码类似，我们将把语言中的每个单词表示为一个热向量，或零的巨型向量，除了单个单词（在单词索引处）。与一种语言中可能存在的几十个字符相比，单词要多得多，因此编码向量要大得多。然而，我们会稍微作弊，并将数据修改为每种语言只使用几千个单词。

我们需要每个单词都有一个唯一的索引，以便以后用作网络的输入和目标。为了跟踪所有这些，我们将使用一个名为Lang的助手类，它有单词→ 索引（word2index）和索引→ 单词（index2word）字典，以及每个单词的计数word2count，稍后将用于替换稀有单词。

SOS\_token = 0  
EOS\_token = 1  
  
  
**class** Lang:  
    **def** \_\_init\_\_(self, name):  
        self.name = name  
        self.word2index = {}  
        self.word2count = {}  
        self.index2word = {0: **"SOS"**, 1: **"EOS"**}  
        self.n\_words = 2  *# Count SOS and EOS*  
  
    **def** addSentence(self, sentence):  
        **for** word **in** sentence.split(**' '**):  
            self.addWord(word)  
  
    **def** addWord(self, word):  
        **if** word **not** **in** self.word2index:  
            self.word2index[word] = self.n\_words  
            self.word2count[word] = 1  
            self.index2word[self.n\_words] = word  
            self.n\_words += 1  
        **else**:  
            self.word2count[word] += 1

2.2数据格式转换

这些文件都是Unicode格式，为了简化，我们将把Unicode字符转换为ASCII，使所有内容都小写，并删掉大部分标点符号。

**def** unicodeToAscii(s):  
    **return** **''**.join(  
    c **for** c **in** unicodedata.normalize(**'NFD'**, s)  
    **if** unicodedata.category(c) != **'Mn'**  
    )  
  
  
*# 其中normalizeString函数中的正则表达式需对应更改，否则会将中文单词替换成空格*  
**def** normalizeString(s):  
    *#变成小写，去掉前后空格*  
    s = s.lower().strip()  
    **if** **' '** **not** **in** s:  
        s = list(s)  
        s = **' '**.join(s)  
    s = unicodeToAscii(s)  *#将unicode变成ascii*  
    s = re.sub(**r"([.。!！?？])"**, **""**, s)  
    **return** s

2.3语言翻译定义

为了读取数据文件，我们将文件拆分为行，然后将行拆分成对。这些文件都是英文的→其他语言，所以如果我们想从其他语言翻译→英语，需要加上reverse flag来逆转这对。

**def** readLangs(lang1, lang2, reverse=**False**):  
    print(**"Reading lines..."**)  
  
    *# Read the file and split into lines*  
    file\_path = **"./data/eng-cmn.txt"**  
    **with** open(file\_path, encoding=**'utf-8'**) **as** file:  
        lines = file.readlines()  
      
  
    *# Split every line into pairs and normalize*  
    pairs = [[normalizeString(s) **for** s **in** l.split(**'\t'**)[:2]] **for** l **in** lines]  
  
  
    *# Reverse pairs, make Lang instances*  
    **if** reverse:  
        pairs = [list(reversed(p)) **for** p **in** pairs]  
        input\_lang = Lang(lang2)  
        output\_lang = Lang(lang1)  
    **else**:  
        input\_lang = Lang(lang1)  
        output\_lang = Lang(lang2)  
  
    **return** input\_lang, output\_lang, pairs

2.4简化数据集

由于有很多例句，我们想快速训练一些东西，所以我们将数据集精简为相对简短的句子。这里的最大长度是10个单词（包括结尾标点符号），我们过滤到翻译成“我是”或“他是”等形式的句子（考虑到之前替换的撇号）。

MAX\_LENGTH = 10  
  
eng\_prefixes = (  
    **"i am "**, **"i m "**,  
    **"he is"**, **"he s "**,  
    **"she is"**, **"she s "**,  
    **"you are"**, **"you re "**,  
    **"we are"**, **"we re "**,  
    **"they are"**, **"they re "**  
)  
  
  
**def** filterPair(p):  
    **return** len(p[0].split(**' '**)) < MAX\_LENGTH **and** \  
        len(p[1].split(**' '**)) < MAX\_LENGTH **and** \  
        p[1].startswith(eng\_prefixes)  
  
  
**def** filterPairs(pairs):  
    **return** [pair **for** pair **in** pairs **if** filterPair(pair)]

2.5定义数据准备过程

准备数据的整个过程是：

1. 读取文本文件并拆分成行，将行拆分成对
2. 规范化文本，按长度和内容筛选
3. 成对从句子中列出单词

**def** prepareData(lang1, lang2, reverse=**False**):  
    input\_lang, output\_lang, pairs = readLangs(lang1, lang2, reverse)  
    print(**"Read %s sentence pairs"** % len(pairs))  
    pairs = filterPairs(pairs)  
    print(**"Trimmed to %s sentence pairs"** % len(pairs))  
    print(**"Counting words..."**)  
    **for** pair **in** pairs:  
        input\_lang.addSentence(pair[0])  
        output\_lang.addSentence(pair[1])  
    print(**"Counted words:"**)  
    print(input\_lang.name, input\_lang.n\_words)  
    print(output\_lang.name, output\_lang.n\_words)  
    **return** input\_lang, output\_lang, pairs

2.6导入数据

input\_lang, output\_lang, pairs = prepareData(**'eng'**, **'cmn'**, **True**)  
print(random.choice(pairs))

3定义Encoder模型

seq2seq网络的编码器是一个RNN，它为输入句子中的每个单词输出一些值。对于每个输入字，编码器输出矢量和隐藏状态，并将隐藏状态用于下一个输入字。

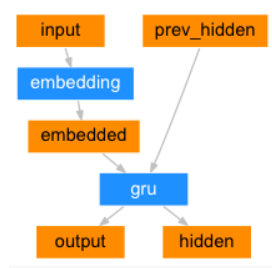


Figure. 12. Encoder模型处理流程

**class** EncoderRNN(nn.Module):  
    **def** \_\_init\_\_(self, input\_size, hidden\_size):  
        super(EncoderRNN, self).\_\_init\_\_()  
        self.hidden\_size = hidden\_size  
  
        self.embedding = nn.Embedding(input\_size, hidden\_size)  
        self.gru = nn.GRU(hidden\_size, hidden\_size)  
  
    **def** forward(self, input, hidden):  
        embedded = self.embedding(input).view(1, 1, -1)  
        output = embedded  
        output, hidden = self.gru(output, hidden)  
        **return** output, hidden  
  
    **def** initHidden(self):  
        **return** torch.zeros(1, 1, self.hidden\_size, device=device)

4定义Attention Decoder模型

使用解码器的输入和隐藏状态作为输入，使用另一个前馈层attn来计算注意力权重。因为训练数据中有各种大小的句子，要实际创建和训练这一层，我们必须选择它可以应用的最大句子长度（输入长度，用于编码器输出）。最大长度的句子将使用所有的注意力权重，而较短的句子将只使用前几个。

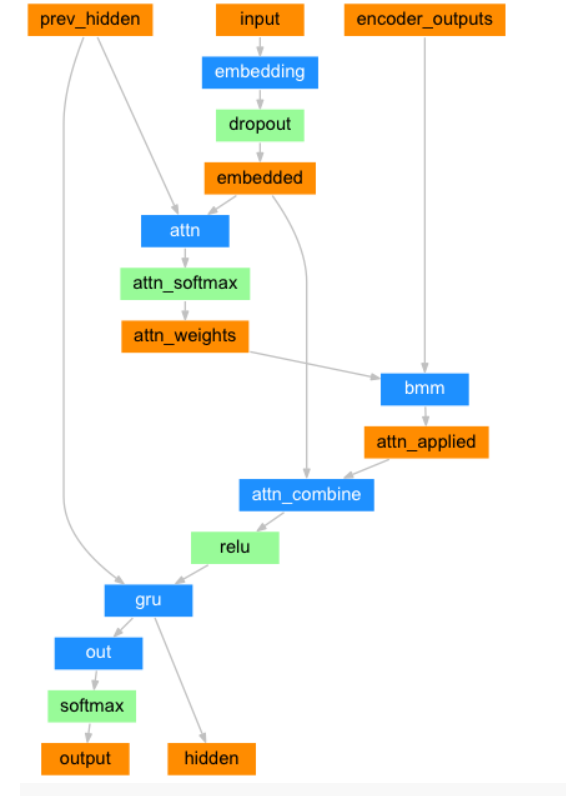


Figure. 13. Attention Decoder模型处理流程

**class** AttnDecoderRNN(nn.Module):  
    **def** \_\_init\_\_(self, hidden\_size, output\_size, dropout\_p=0.1, max\_length=MAX\_LENGTH):  
        super(AttnDecoderRNN, self).\_\_init\_\_()  
        self.hidden\_size = hidden\_size  
        self.output\_size = output\_size  
        self.dropout\_p = dropout\_p  
        self.max\_length = max\_length  
  
        self.embedding = nn.Embedding(self.output\_size, self.hidden\_size)  
        self.attn = nn.Linear(self.hidden\_size \* 2, self.max\_length)  
        self.attn\_combine = nn.Linear(self.hidden\_size \* 2, self.hidden\_size)  
        self.dropout = nn.Dropout(self.dropout\_p)  
        self.gru = nn.GRU(self.hidden\_size, self.hidden\_size)  
        self.out = nn.Linear(self.hidden\_size, self.output\_size)  
  
    **def** forward(self, input, hidden, encoder\_outputs):  
        embedded = self.embedding(input).view(1, 1, -1)  
        embedded = self.dropout(embedded)  
  
        attn\_weights = F.softmax(  
            self.attn(torch.cat((embedded[0], hidden[0]), 1)), dim=1)  
        attn\_applied = torch.bmm(attn\_weights.unsqueeze(0),  
                                 encoder\_outputs.unsqueeze(0))  
  
        output = torch.cat((embedded[0], attn\_applied[0]), 1)  
        output = self.attn\_combine(output).unsqueeze(0)  
  
        output = F.relu(output)  
        output, hidden = self.gru(output, hidden)  
  
        output = F.log\_softmax(self.out(output[0]), dim=1)  
        **return** output, hidden, attn\_weights  
  
    **def** initHidden(self):  
        **return** torch.zeros(1, 1, self.hidden\_size, device=device)

3模型训练

3.1准备训练数据

为了训练，对于每一对，我们需要一个输入张量（输入句子中单词的索引）和目标张量（目标句子中单词索引）。在创建这些向量时，我们会将EOS标记附加到两个序列。

**def** indexesFromSentence(lang, sentence):  
    **return** [lang.word2index[word] **for** word **in** sentence.split(**' '**)]  
  
  
**def** tensorFromSentence(lang, sentence):  
    indexes = indexesFromSentence(lang, sentence)  
    indexes.append(EOS\_token)  
    **return** torch.tensor(indexes, dtype=torch.long, device=device).view(-1, 1)  
  
  
**def** tensorsFromPair(pair):  
    input\_tensor = tensorFromSentence(input\_lang, pair[0])  
    target\_tensor = tensorFromSentence(output\_lang, pair[1])  
    **return** (input\_tensor, target\_tensor)

4.2定义模型训练函数

为了训练，我们通过编码器运行输入语句，并跟踪每个输出和最新的隐藏状态。然后，解码器被赋予＜SOS＞标记作为其第一输入，编码器的最后一个隐藏状态作为其第一隐藏状态。

“teacher\_forcing\_ratio”是指使用实际目标输出作为下一个输入，而不是使用解码器的猜测作为下一输入。使用teacher\_forcing\_ratio使其收敛得更快，但当训练的网络被利用时，它可能会表现出不稳定性。

我们可以随机选择使用教师强制或不使用简单的if语句。将teacher\_forcing\_ratio调大以使用更多。

teacher\_forcing\_ratio = 0.5  
  
  
**def** train(input\_tensor, target\_tensor, encoder, decoder, encoder\_optimizer, decoder\_optimizer, criterion, max\_length=MAX\_LENGTH):  
    encoder\_hidden = encoder.initHidden()  
  
    encoder\_optimizer.zero\_grad()  
    decoder\_optimizer.zero\_grad()  
  
    input\_length = input\_tensor.size(0)  
    target\_length = target\_tensor.size(0)  
  
    encoder\_outputs = torch.zeros(max\_length, encoder.hidden\_size, device=device)  
  
    loss = 0  
  
    **for** ei **in** range(input\_length):  
        encoder\_output, encoder\_hidden = encoder(  
            input\_tensor[ei], encoder\_hidden)  
        encoder\_outputs[ei] = encoder\_output[0, 0]  
  
    decoder\_input = torch.tensor([[SOS\_token]], device=device)  
  
    decoder\_hidden = encoder\_hidden  
  
    use\_teacher\_forcing = **True** **if** random.random() < teacher\_forcing\_ratio **else** **False**  
  
    **if** use\_teacher\_forcing:  
        *# Teacher forcing: Feed the target as the next input*  
        **for** di **in** range(target\_length):  
            decoder\_output, decoder\_hidden, decoder\_attention = decoder(  
                decoder\_input, decoder\_hidden, encoder\_outputs)  
            loss += criterion(decoder\_output, target\_tensor[di])  
            decoder\_input = target\_tensor[di]  *# Teacher forcing*  
  
    **else**:  
        *# Without teacher forcing: use its own predictions as the next input*  
        **for** di **in** range(target\_length):  
            decoder\_output, decoder\_hidden, decoder\_attention = decoder(  
                decoder\_input, decoder\_hidden, encoder\_outputs)  
            topv, topi = decoder\_output.topk(1)  
            decoder\_input = topi.squeeze().detach()  *# detach from history as input*  
  
            loss += criterion(decoder\_output, target\_tensor[di])  
            **if** decoder\_input.item() == EOS\_token:  
                **break**  
  
    loss.backward()  
  
    encoder\_optimizer.step()  
    decoder\_optimizer.step()  
  
    **return** loss.item() / target\_length

4.3定义辅助监测函数

这是一个辅助函数，用于打印给定当前时间和进度%的已用时间和估计剩余时间

*#This is a helper function to print time elapsed and estimated time remaining given the current time and progress %.*  
  
**import** time  
**import** math  
  
  
**def** asMinutes(s):  
    m = math.floor(s / 60)  
    s -= m \* 60  
    **return** **'%dm %ds'** % (m, s)  
  
  
**def** timeSince(since, percent):  
    now = time.time()  
    s = now - since  
    es = s / (percent)  
    rs = es - s  
    **return** **'%s (- %s)'** % (asMinutes(s), asMinutes(rs))

4.4定义损失函数，优化方法，迭代次数

1. 启动计时器
2. 初始化优化器和条件
3. 创建一组训练对
4. 开始绘制空损失数组

**def** trainIters(encoder, decoder, n\_iters, print\_every=1000, plot\_every=100, learning\_rate=0.01):  
    start = time.time()  
    plot\_losses = []  
    print\_loss\_total = 0  *# Reset every print\_every*  
    plot\_loss\_total = 0  *# Reset every plot\_every*  
  
    encoder\_optimizer = optim.SGD(encoder.parameters(), lr=learning\_rate)  
    decoder\_optimizer = optim.SGD(decoder.parameters(), lr=learning\_rate)  
    training\_pairs = [tensorsFromPair(random.choice(pairs))  
                      **for** i **in** range(n\_iters)]  
    criterion = nn.NLLLoss()  
  
    **for** iter **in** range(1, n\_iters + 1):  
        training\_pair = training\_pairs[iter - 1]  
        input\_tensor = training\_pair[0]  
        target\_tensor = training\_pair[1]  
  
        loss = train(input\_tensor, target\_tensor, encoder,  
                     decoder, encoder\_optimizer, decoder\_optimizer, criterion)  
        print\_loss\_total += loss  
        plot\_loss\_total += loss  
  
        **if** iter % print\_every == 0:  
            print\_loss\_avg = print\_loss\_total / print\_every  
            print\_loss\_total = 0  
            print(**'%s (%d %d%%) %.4f'** % (timeSince(start, iter / n\_iters),  
                                         iter, iter / n\_iters \* 100, print\_loss\_avg))  
  
        **if** iter % plot\_every == 0:  
            plot\_loss\_avg = plot\_loss\_total / plot\_every  
            plot\_losses.append(plot\_loss\_avg)  
            plot\_loss\_total = 0  
  
    showPlot(plot\_losses)

5定义绘图函数

使用matplotlib进行绘图，使用训练时保存的损失值plot\_loss数组

**import** matplotlib.pyplot **as** plt  
**import** matplotlib.ticker **as** ticker  
**import** numpy **as** np  
  
  
**def** showPlot(points):  
    plt.figure()  
    fig, ax = plt.subplots()  
    *# this locator puts ticks at regular intervals*  
    loc = ticker.MultipleLocator(base=0.2)  
    ax.yaxis.set\_major\_locator(loc)  
    plt.plot(points)  
    plt.show()

6模型评估

6.1定义测试函数

评估与训练基本相同，但没有目标，因此我们只需将解码器的预测反馈给每一步的自身。每次它预测一个单词时，我们都会将其添加到输出字符串中，如果它预测的是EOS标记，我们就到此为止。我们还存储解码器的注意力输出，以便稍后显示。

**def** evaluate(encoder, decoder, sentence, max\_length=MAX\_LENGTH):  
    **with** torch.no\_grad():  
        input\_tensor = tensorFromSentence(input\_lang, sentence)  
        input\_length = input\_tensor.size()[0]  
        encoder\_hidden = encoder.initHidden()  
  
        encoder\_outputs = torch.zeros(max\_length, encoder.hidden\_size, device=device)  
  
        **for** ei **in** range(input\_length):  
            encoder\_output, encoder\_hidden = encoder(input\_tensor[ei],  
                                                     encoder\_hidden)  
            encoder\_outputs[ei] += encoder\_output[0, 0]  
  
        decoder\_input = torch.tensor([[SOS\_token]], device=device)  *# SOS*  
  
        decoder\_hidden = encoder\_hidden  
  
        decoded\_words = []  
        decoder\_attentions = torch.zeros(max\_length, max\_length)  
  
        **for** di **in** range(max\_length):  
            decoder\_output, decoder\_hidden, decoder\_attention = decoder(  
                decoder\_input, decoder\_hidden, encoder\_outputs)  
            decoder\_attentions[di] = decoder\_attention.data  
            topv, topi = decoder\_output.data.topk(1)  
            **if** topi.item() == EOS\_token:  
                decoded\_words.append(**'<EOS>'**)  
                **break**  
            **else**:  
                decoded\_words.append(output\_lang.index2word[topi.item()])  
  
            decoder\_input = topi.squeeze().detach()  
  
        **return** decoded\_words, decoder\_attentions[:di + 1]

6.2随机从训练样本中挑选测试样本

**def** evaluateRandomly(encoder, decoder, n=100):  
    sum\_scores = 0  
    **for** i **in** range(n):  
        pair = random.choice(pairs)  
        print(**'>'**, pair[0])  
        print(**'='**, pair[1])  
        output\_words, attentions = evaluate(encoder, decoder, pair[0])  
        output\_sentence = **' '**.join(output\_words)  
        print(**'<'**, output\_sentence)  
        print(**''**)  
        w = []  
        words = pair[1].strip(**' '**).split(**' '**)  
        words.append(**'<EOS>'**)  
        w.append(words)  
        bleu\_score = sentence\_bleu(w, output\_words)  
        sum\_scores += bleu\_score  
    print(**'The bleu\_score is '**, sum\_scores/n)

7函数启动

hidden\_size = 256  
encoder1 = EncoderRNN(input\_lang.n\_words, hidden\_size).to(device)  
attn\_decoder1 = AttnDecoderRNN(hidden\_size, output\_lang.n\_words, dropout\_p=0.1).to(device)  
  
trainIters(encoder1, attn\_decoder1, 75000, print\_every=5000)

evaluateRandomly(encoder1, attn\_decoder1)

8结果可视化

plt.rcParams[**'font.sans-serif'**] = [**'KaiTi'**] *# 指定默认字体*  
plt.rcParams[**'axes.unicode\_minus'**] = **False** *# 解决保存图像是负号'-'显示为方块的问题*

output\_words, attentions = evaluate(  
    encoder1, attn\_decoder1, **"你 只 是 玩"**)  
print(output\_words)  
plt.matshow(attentions.numpy())  
plt.show()

9单独的数据测试与结果可视化

**def** showAttention(input\_sentence, output\_words, attentions):  
    *# Set up figure with colorbar*  
    fig = plt.figure()  
    ax = fig.add\_subplot(111)  
    cax = ax.matshow(attentions.numpy(), cmap=**'bone'**)  
    fig.colorbar(cax)  
  
    *# Set up axes*  
    ax.set\_xticklabels([**''**] + input\_sentence.split(**' '**) +  
                       [**'<EOS>'**], rotation=90)  
    ax.set\_yticklabels([**''**] + output\_words)  
  
    *# Show label at every tick*  
    ax.xaxis.set\_major\_locator(ticker.MultipleLocator(1))  
    ax.yaxis.set\_major\_locator(ticker.MultipleLocator(1))  
  
    plt.show()  
  
  
**def** evaluateAndShowAttention(input\_sentence):  
    output\_words, attentions = evaluate(  
        encoder1, attn\_decoder1, input\_sentence)  
    print(**'input ='**, input\_sentence)  
    print(**'output ='**, **' '**.join(output\_words))  
    showAttention(input\_sentence, output\_words, attentions)  
  
  
evaluateAndShowAttention(**"他 和 他 的 邻 居 相 处 "**)  
  
evaluateAndShowAttention(**"我 肯 定 他 会 成 功 的 "**)  
  
evaluateAndShowAttention(**"他 總 是 忘 記 事 情"**)  
  
evaluateAndShowAttention(**"我 们 非 常 需 要 食 物 "**)

10实验结果分析

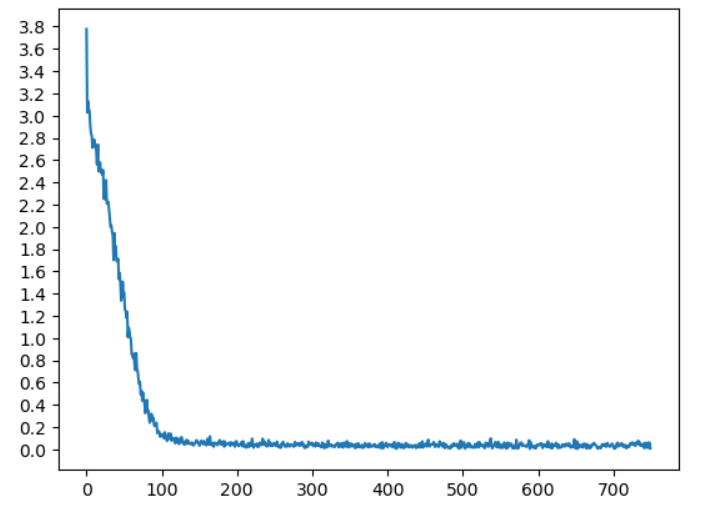


Figure. 14. 模型训练误差迭代过程截图

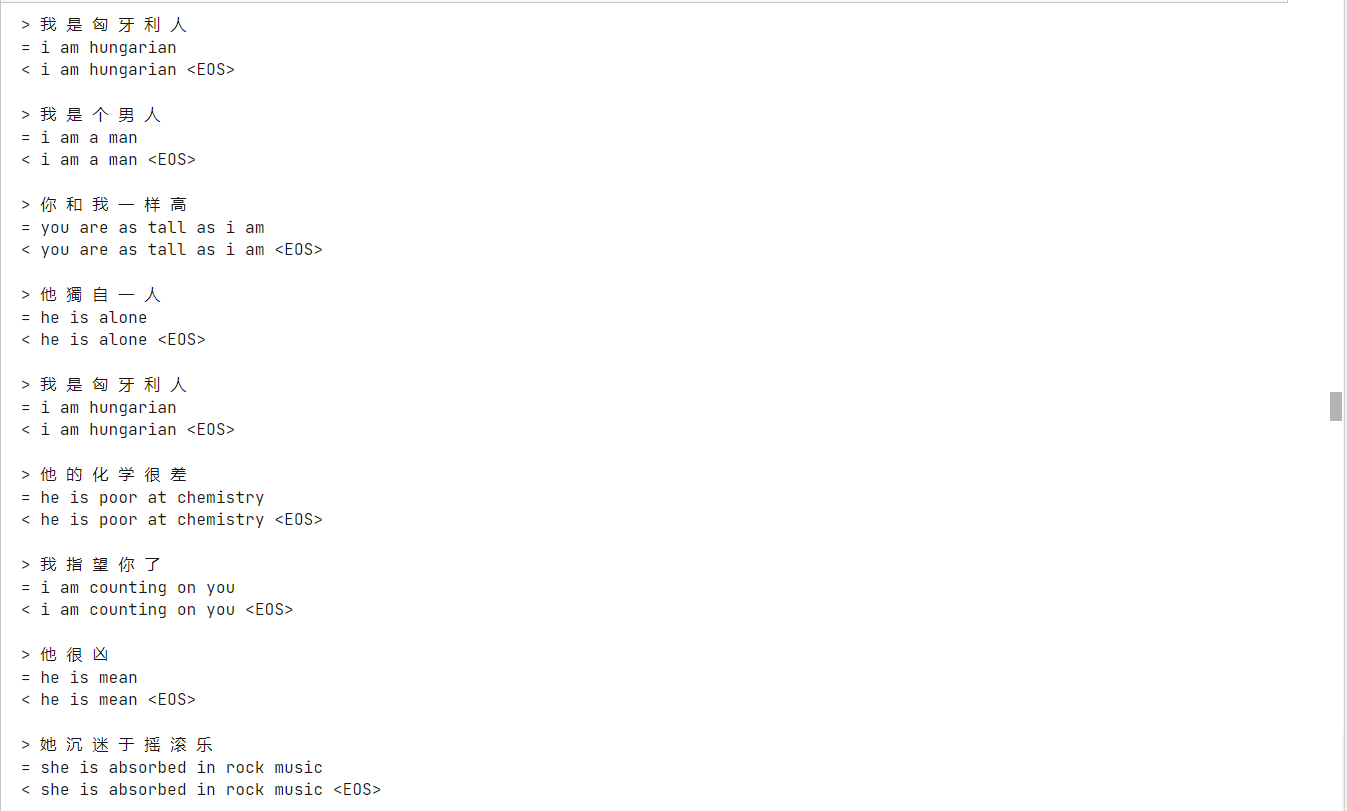


Figure. 15.1模型测试结果截图1

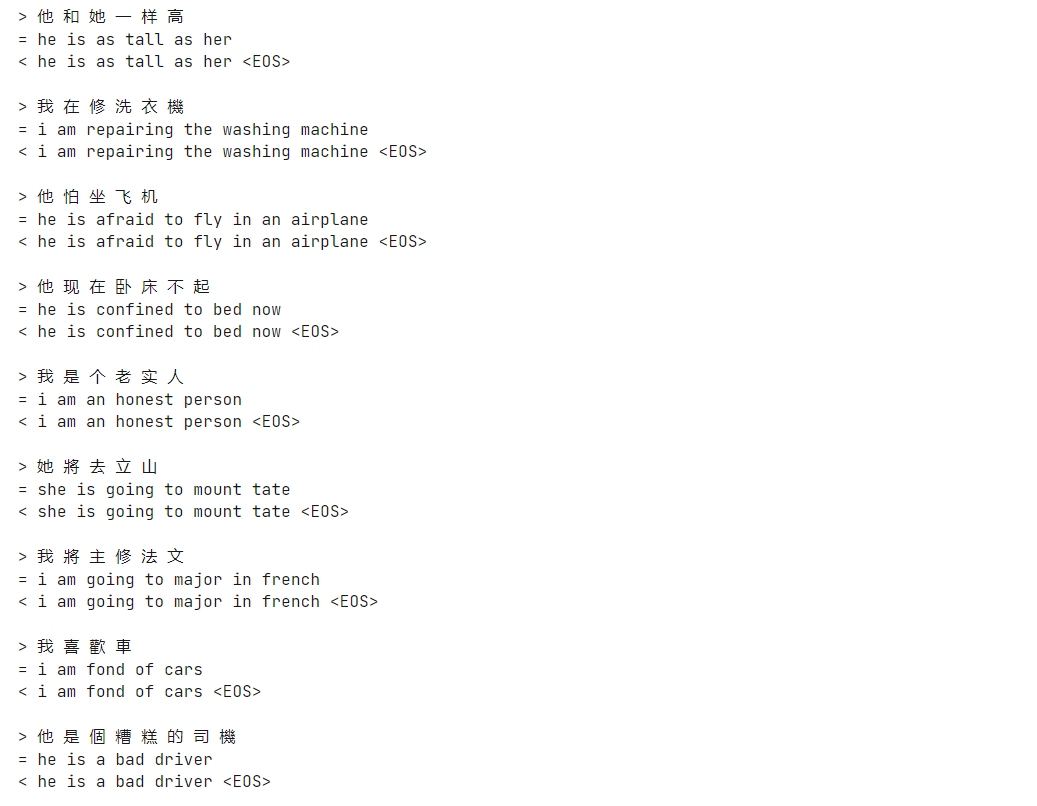


Figure. 15.2模型测试结果截图2

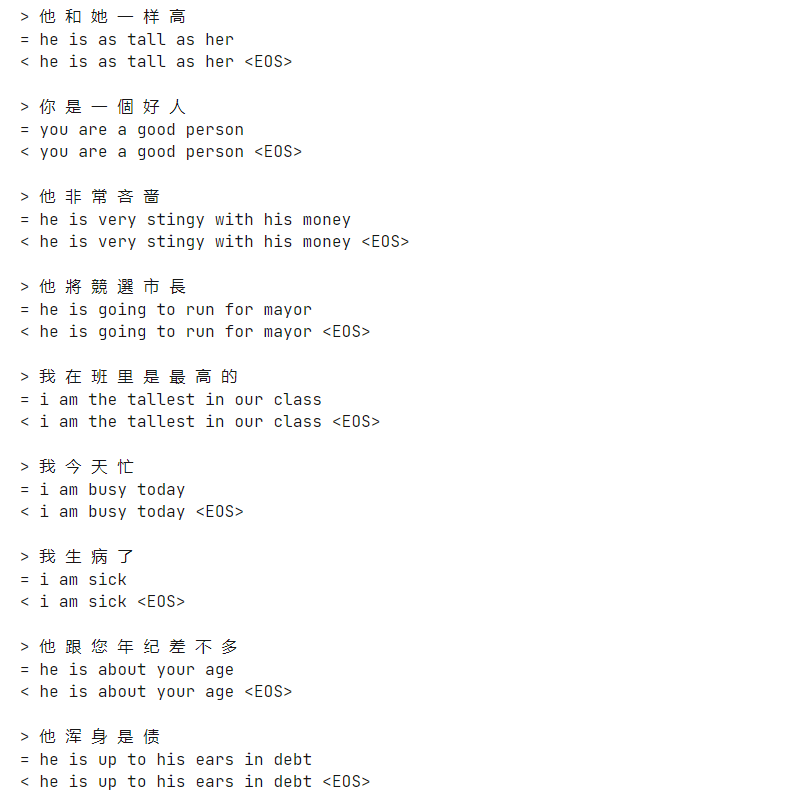


Figure. 15.3模型测试结果截图3

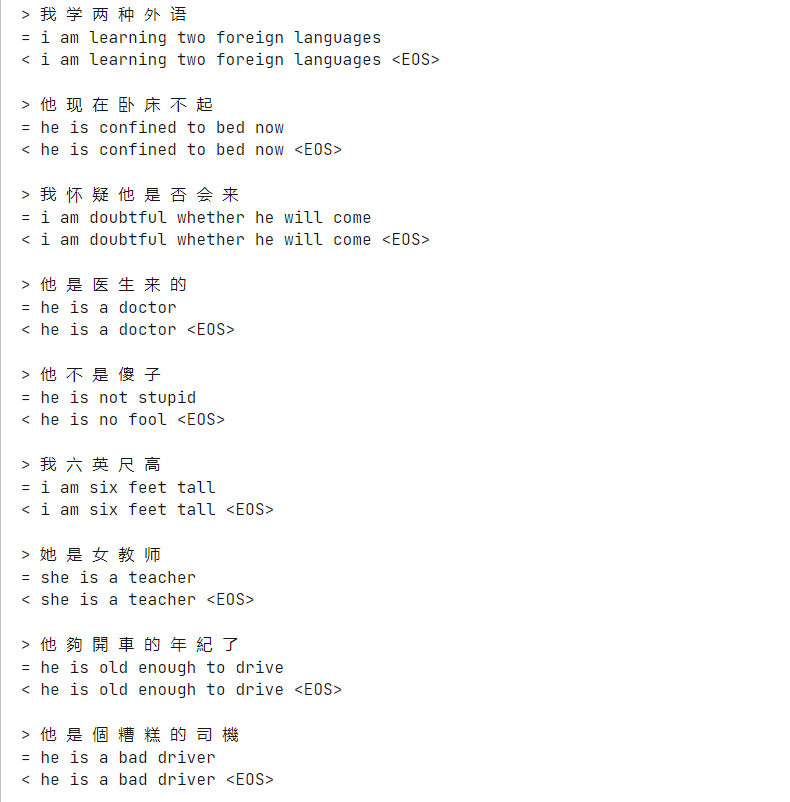


Figure. 15.4模型测试结果截图4

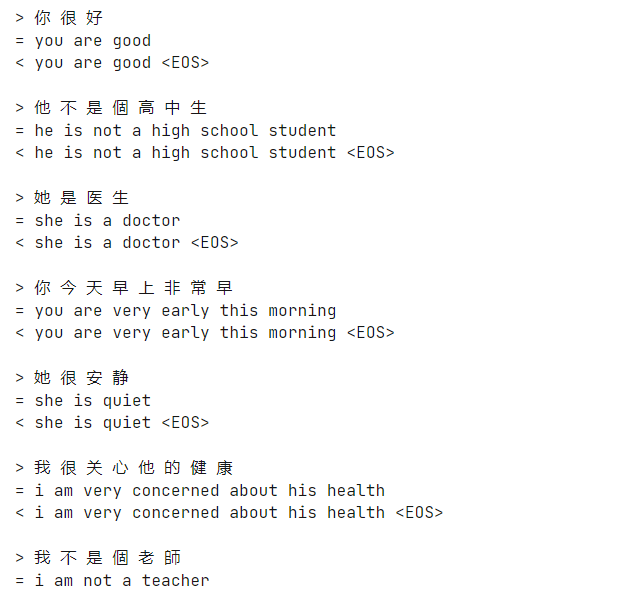


Figure. 15.5模型测试结果截图5

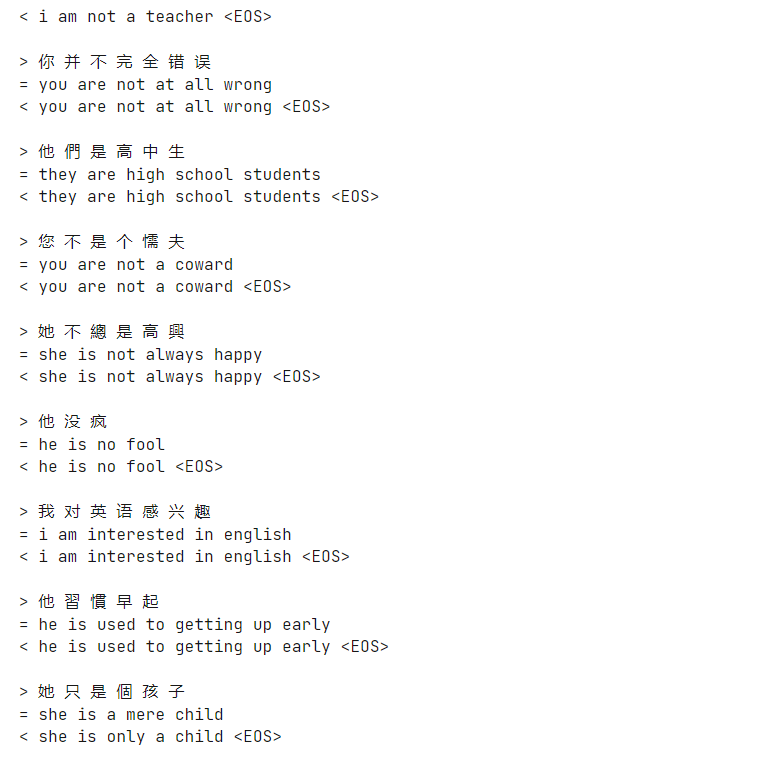


Figure. 15.6模型测试结果截图6

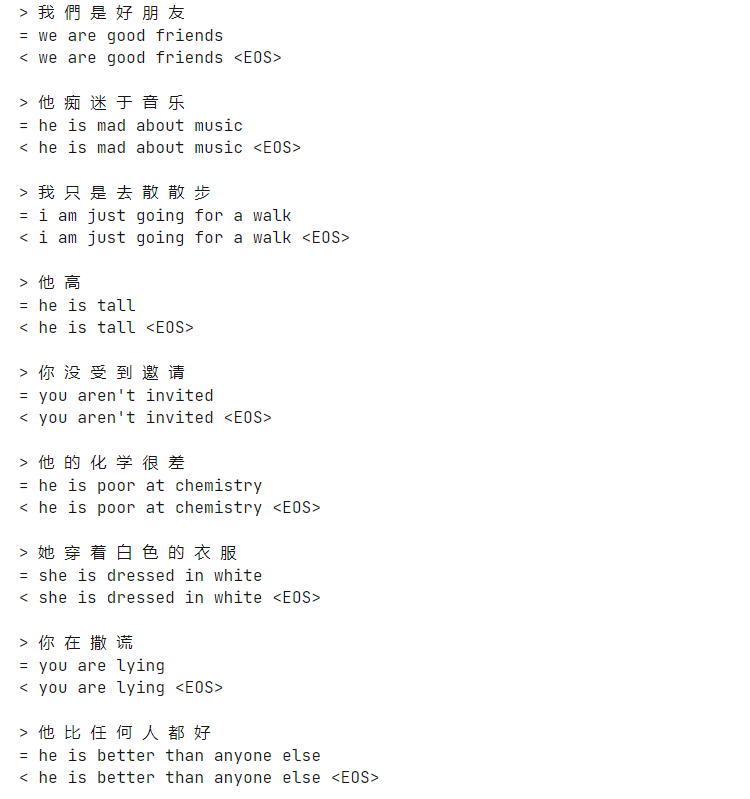


Figure. 15.7模型测试结果截图7

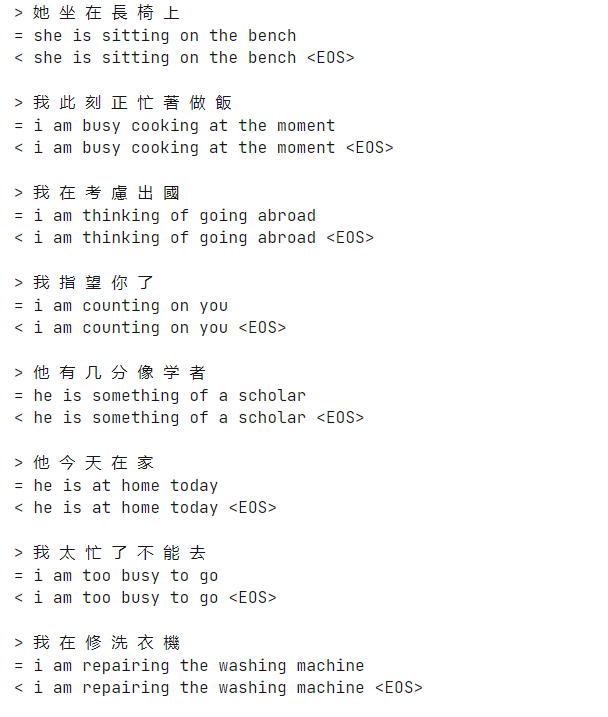


Figure. 15.8模型测试结果截图8

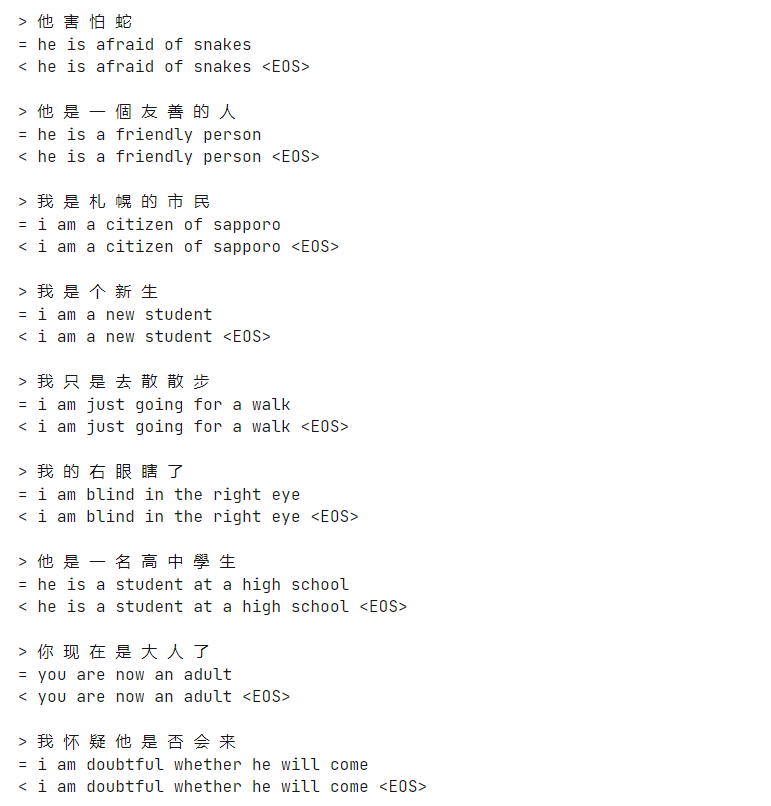


Figure. 15.9模型测试结果截图9

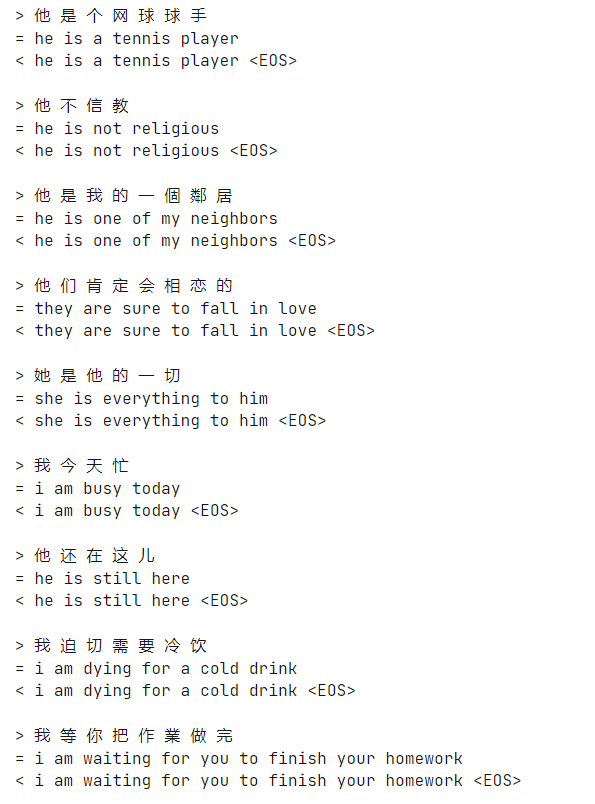


Figure. 15.10模型测试结果截图10

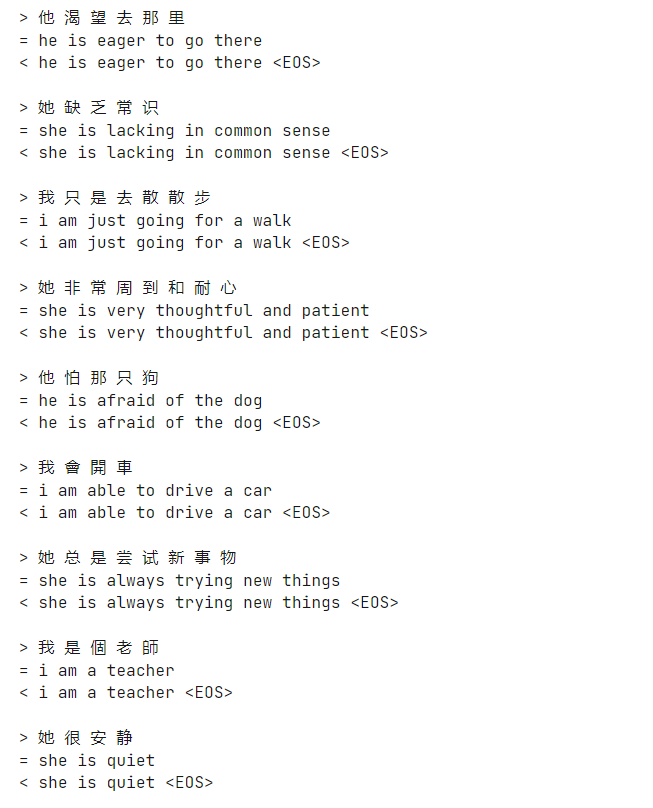


Figure. 15.11模型测试结果截图11

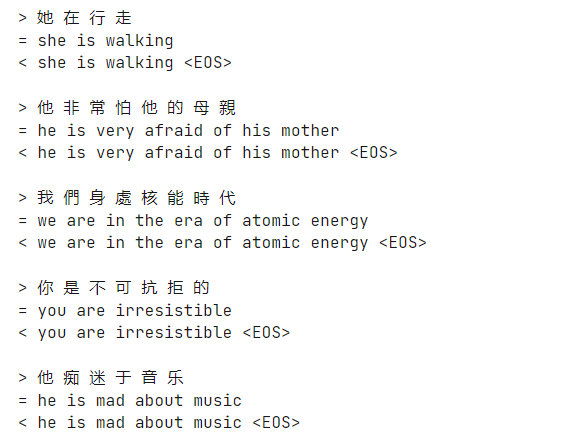


Figure. 15.12模型测试结果截图12

TABLE I

模型预测准确率

|  |  |
| --- | --- |
| bleu\_score | 0.98 |

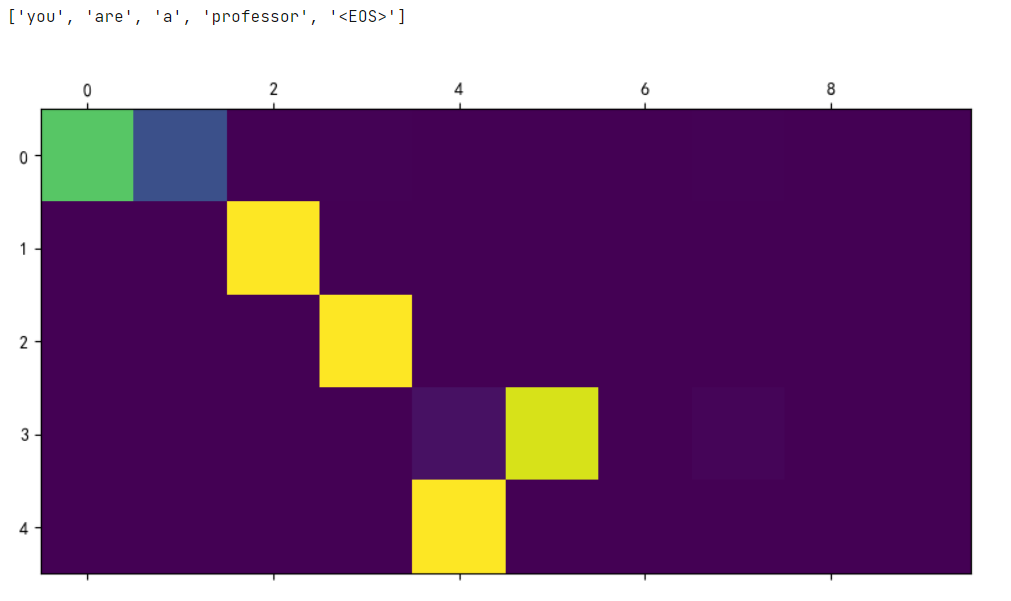


Figure. 16模型测试（“你 只 是 玩”）

TABLE II

模型预测

|  |  |
| --- | --- |
| Input | Output |
| 他 和 他 的 邻 居 相 处 | he is getting along with his neighborhood |
| 我 肯 定 他 会 成 功 的 | i am sure of his success |
| 他 總 是 忘 記 事 情 | he is always forgetting things |
| 我 们 非 常 需 要 食 物 | we are badly in need of food |

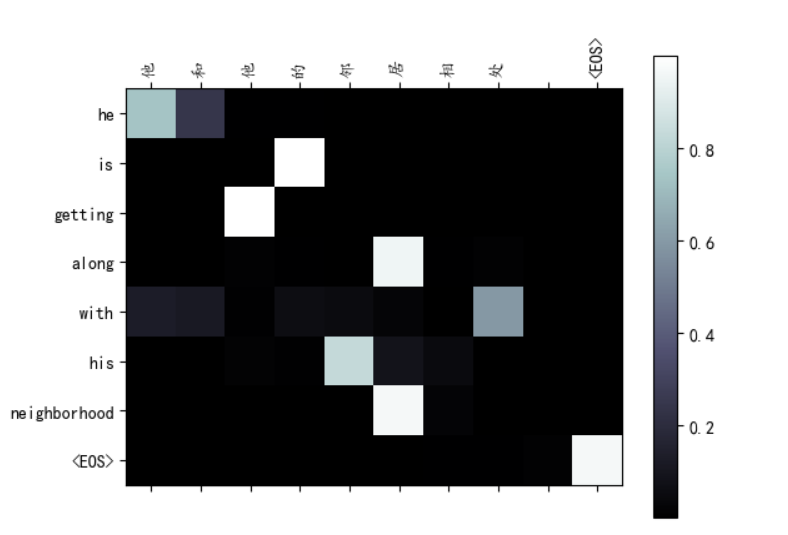


Figure. 17.1.模型测试（“他 和 他 的 邻 居 相 处”）

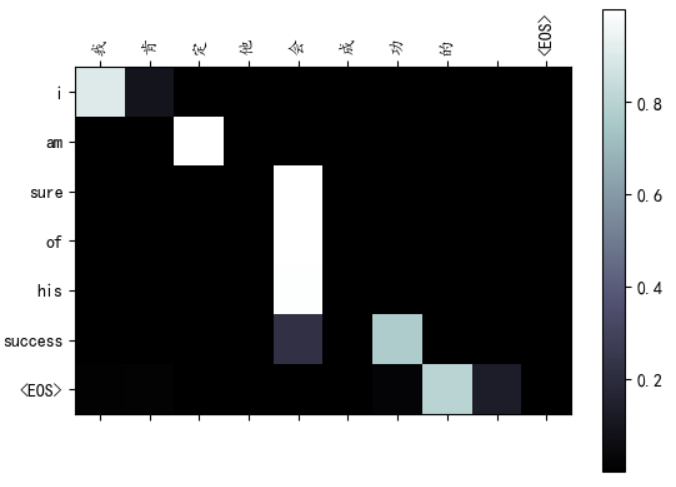


Figure. 17.2.模型测试（“我 肯 定 他 会 成 功 的”）

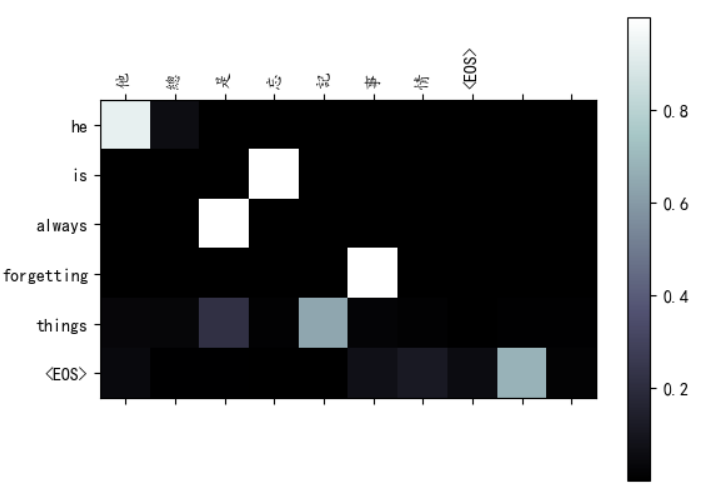


Figure. 17.3.模型测试（“他 總 是 忘 記 事 情”）

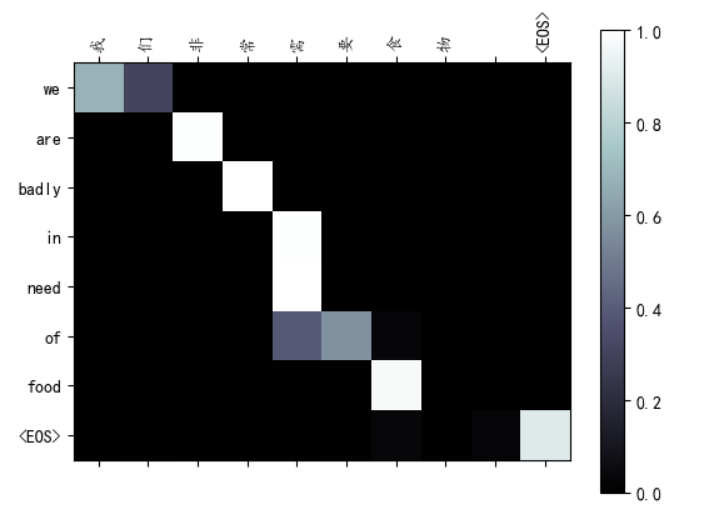


Figure. 17.4.模型测试（“我 们 非 常 需 要 食 物”）

## **结论**

**周治平：**

在本次实验中，主要实现了基于注意力机制的编码器-解码器中英文序列翻译机，这个模型是经典的RNN框架的模型，其主要步骤是先导入中英文翻译数据集，通过一系列的预处理，处理成模型训练所需要的数据集（数据张量），然后定义Encoder模型和基于注意力机制的AttnDecoder，然后定义NLLLOSS损失函数，然后通过随机梯度下降法对模型进行训练，进行参数优化。最后通过多次迭代，训练结果输出与绘图就完成了。

由上述图可以看到，模型经过训练之后对中文的翻译都是较为准确地。总体的预测准确率到达了98%。最后用其预测4条语句的翻译结果，都得到了很好地表现。说明该模型对于中英文序列的翻译是较为合适的。

对于本次实验而言，主要有代码参考，虽然基于编码器-解码器框架的RNN模型较为复杂，但是通过了解其内核的计算公式，再结合代码去理解，便能明白RNN模型的运算机制。仅仅看懂模型的公式是远远不够的，与真正的代码实现相比，其相差还是比较大。此外还学习到了许多pytorch的库中的函数接口，以及多层深度学习模型的建立。总体而言本次实验的收获是挺大的。

**谭嘉展：**

通过本次实验，我们实现了基于注意力机制的编码器-解码器中英文序列翻译机，通过实验探索自然语言处理。

理解经典的Sequence-to-Sequence机器翻译模型，

seq2seq模型是以编码（Encode）和解码（Decode）为代表的架构方式，seq2seq模型是根据输入序列X来生成输出序列Y，在翻译，文本自动摘要和机器人自动问答以及一些回归预测任务上有着广泛的运用。以encode和decode为代表的seq2seq模型，encode意思是将输入序列转化成一个固定长度的向量，decode意思是将输入的固定长度向量解码成输出序列。其中编码解码的方式可以是RNN,CNN、LSTM、GRU等。缺点是：1、定长编码是信息瓶颈2、长度越长，前面输入RNN的信息就被越稀释。

掌握Attention机制在机器翻译模型上的应用：attention机制能够让decoder关注与当前解码相关的encoder源句子核心部分，大大提高了神经机器翻译模型效果。对梯度消失有一定的帮助，因为attention机制提供了源句子信息与解码过程的直接连接，缩短了路径距离attention机制提供了一些可解释性，通过attention分值，可以直观看到每一步的解码过程，decoder更关注encoder源句子的那部分。

通过搭建机器翻译模型，在简单小规模数据集上验证模型性能，培养工程能力，了解Transformer在机器翻译任务上的应用。

## **附录：实验任务分配**

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 任务分配 |
| 周治平 | 202030483178 | 模型建模+代码注释添加 |
| 谭嘉展 | 202030481488 | 代码注释添加，运行实验代码 |
| 宋文林 | 202030484052 | 代码注释添加，实验报告整理 |