中山大學
SUN YAT-SEN UNIVERSITY

自然语言处理

期末大作业

姓名：刘斯宇
学号：17341110

Contents

这次实验主要包括数据处理，模型搭建，模型训练，模型预测，结果分析几个部分。

1 数据处理

1.1 实验数据介绍

这次实验给了我们两个数据集，但是由于计算资源的限制，我只使用了数据集 dataset_10000，下面我们来看看数据集 dataset_10000 的构成。

train_source_8000.txt:训练集的源语言集

train_target_8000.txt:训练集的目标语言集

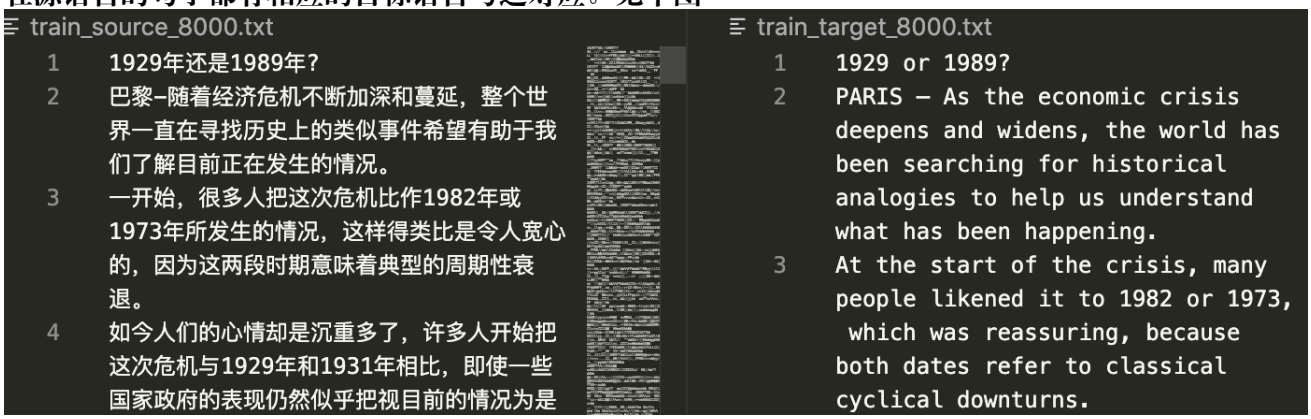
dev_source_8000.txt:验证集的源语言集

dev_target_8000.txt:验证集的目标语言集

test_source_8000.txt:测试集的源语言集

test_target_8000.txt:测试集的目标语言集

每个在源语言的句子都有相应的目标语言与之对应。见下图



1.2 清洗数据

在实验中我们发现数据集并不是完全干净的，还是有一些乱码的情况的，特别是对于中文的情况，在这种情况下，我们可以通过中文 unicode 编码找到这样的字符，当然我们保留中文和英文字母以及数字还有标点符号，所以出了上述的其他编码统统都要用 <UNK> 来代替

判断是否是有效的字符

根据我们所规定的有效字符的集合，分别得到他们的 Unicode 编码的范围来判断某个字符是否是有效的。

```
    #判断是否是汉字
    def is_chinese(uchar):
        if uchar >= u'\u4E00' and uchar <= u'\u9FA5':
            return True
        else:
            return False
    #判断是否是数字
    def is_digit(uchar):
        if uchar >= u'\u0030' and uchar <= u'\u0039':
            return True
        else:
            return False
    #判断是否是英文字母
    def is_alpha(uchar):
        if (uchar >= u'\u0041' and uchar <= u'\u005A') or
         (uchar >= u'\u0061' and uchar <= u'\u007A'):
            return True
        else:
            return False
```

1.3 词表

1.3.1 中文分词

中文分词，还是使用的 jieba 库进行分词，对于每个句子的分词结果进行词频的计算，然后根据词频进行统计，词表中只保留一部分的单词，这里主要是为了训练的方便，全部都进行训练的话，效果是很慢并且是十分的不好的，所以我选择词频前 8000 的单词作为中文的词表，对于不在词表中的一切单词我们都统一当作 <UNK>

- **逐行读取句子**
- **jieba 分词**
- **将分词后的非有效字符剔除**

- **将单词放入词表中**
- **取词表中前 8000 个频率比较高的词作为最后的词表**

```
def div_chinese(filename):
    f = open('ch.txt','w')
    res = []
    wordlist = []
    word2fre={}
    wordlist.append('<pad>')
    lines = open(filename, 'r')
    for line in lines:
        line = ' '.join(jieba.cut(line, cut_all=False))
        line = line.strip()
        line = line.split(' ')
        for index, words in enumerate(line):
            flag = True
            for word in words:
                if word not in pun and not is_chinese(word) and not
                    is_digit(
                        word) and not is_alpha(word):
                    flag = False
                    break
            if not flag:
                line[index] = 'UNK'
                #print(line)
        line.insert(0, '<BOS>')
        line.append('<EOS>')
        res.append(line)
    for lines in res:
        for word in lines:
            if word not in wordlist:
                wordlist.append(word)
```

```
                word2fre[word]=1
            else:
                word2fre[word]+=1
    word2fre = sorted(word2fre.items(), key=lambda item: item[1],
        reverse=True)
    ch2index={}
    cnt=0
    for item in word2fre[:8000]:
        ch2index[item[0]]=cnt
        cnt+=1
        f.write(item[0])
        f.write('\n')
```

1.3.2 英文词表

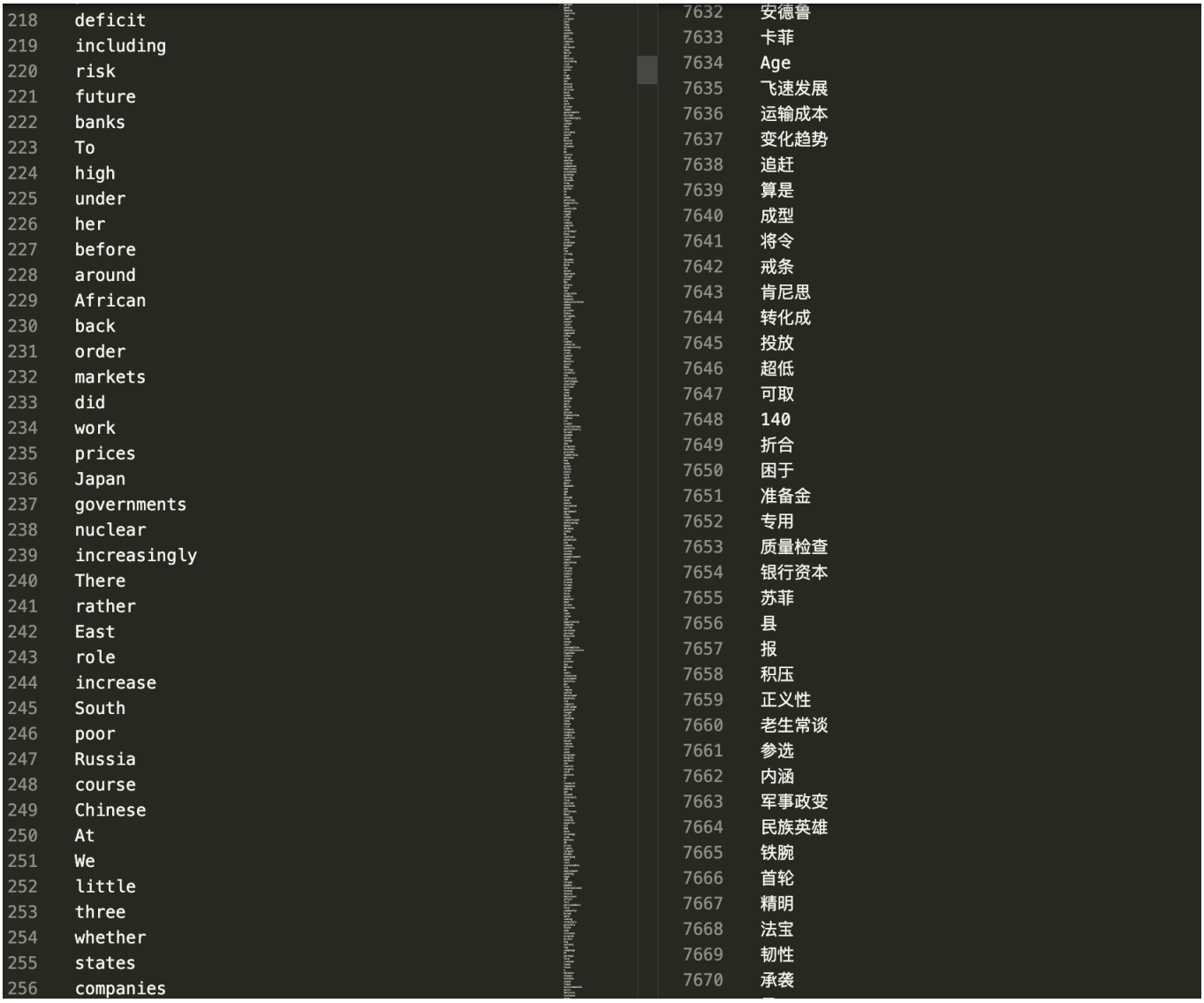
英文分词主要可以根据空格来进行分词，因为英文的单词之间都是用的空格隔开的，所以完全可以使用空格来进行分词，不过英文标点有一点比较特殊的是每个句子的标点符号是会直接跟着单词后面的，对于这种情况，我们只需要在这样的词中加一个空格与标点符号分开就行了。

- 按照空格进行分词
- 将单词存入词表中
- 同样的，为了提高训练速度，我们只取词频前 4000 的单词当作词表，其他的所有都作为 <UNK>

```
def div_English(filename):
    f = open('en.txt','w')
    res = []
    word_list = []
    word_list.append('<pad>')
    word2fre={}
    lines = open(filename, 'r')
    for line in lines:
        line = line.strip('\n')
```

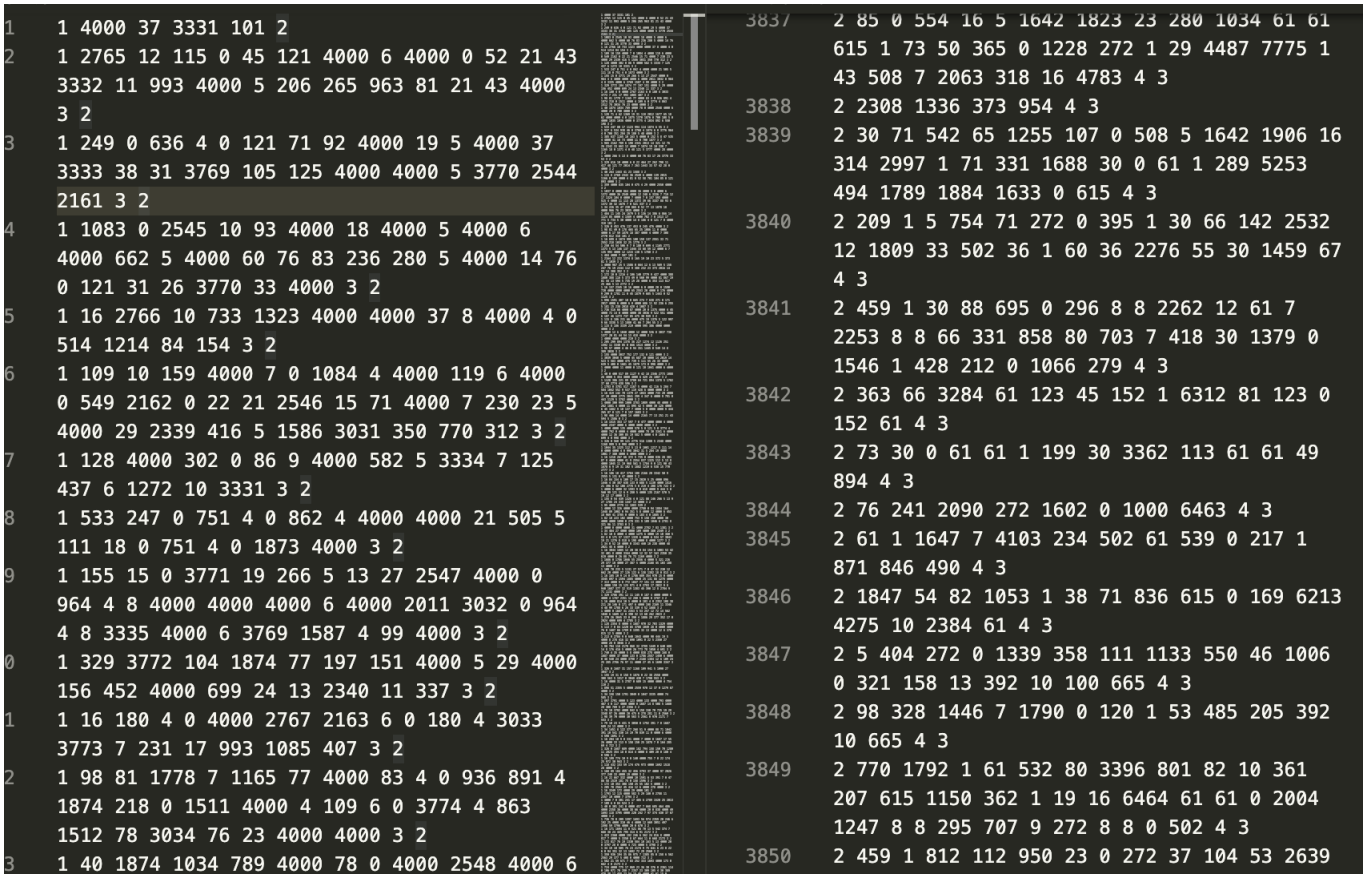
```
        line = line[:-2] + ' ' + line[-1]
        line_list = line.split(' ')
        line_list.insert(0, '<BOS>')
        line_list.append('<EOS>')
        res.append(line_list)
    for line in res:
        for word in line:
            if len(word)==0:
                continue
            if len(word)>1 and word[-1] in enpun:
                word=word[:-1]
            if word not in word_list:
                word_list.append(word)
                word2fre[word]=1
            else:
                word2fre[word]+=1

    word2fre = sorted(word2fre.items(), key=lambda item: item[1],
       reverse=True)
    en2index={}
    cnt=0
    for item in word2fre[:4000]:
        en2index[item[0]]=cnt
        cnt+=1
        f.write(item[0])
        f.write('\n')
```

通过上面的过程，我们可以得到两个词表，这两个词表如下图所示



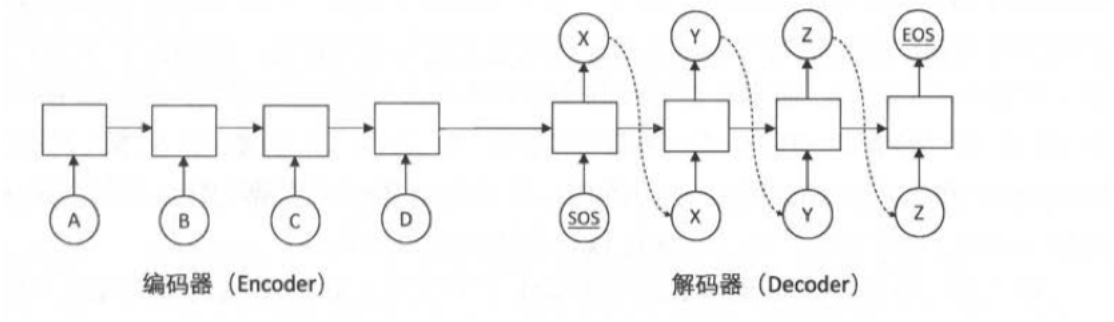
1.4 语料转换

根据上面生成的词表，我们要生成数据集，这个数据集就是单词到词表中 index 的映射，代码的实现是比较简单的，就不展示了，下面展示一下最后的结果



2 模型搭建

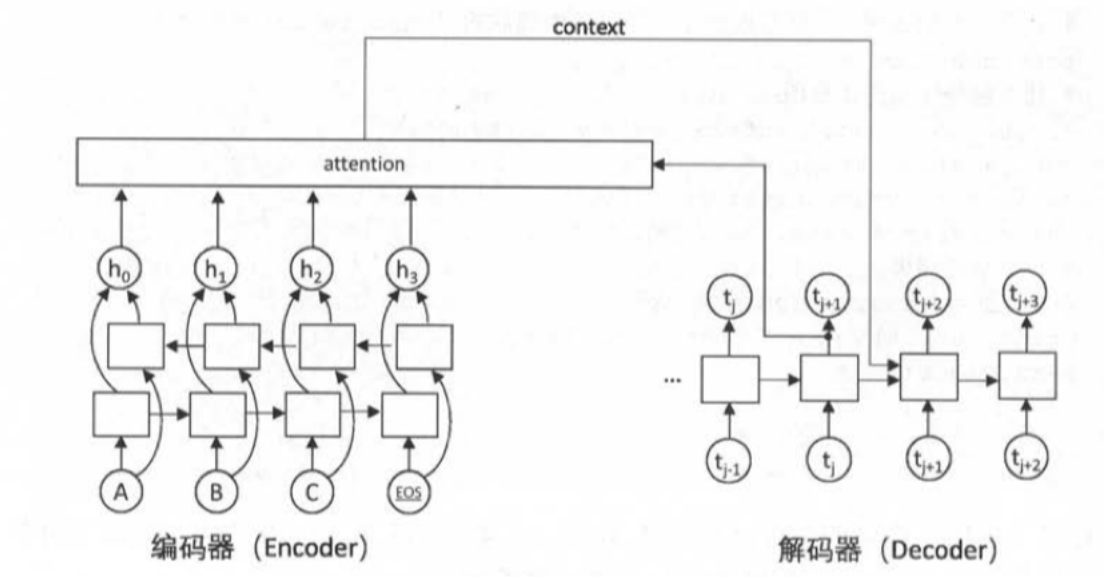
2.1 seq2seq 模型



Seq2Seq 模型的基本思想非常简单一一使用一个循环神经网络读取输入句子，将整个句子的信息压缩到一个固定维度的编码中；再使用另一个循环神经网络读取这个编码，将其“解压”为目标语言的一个句子。这两个循环神经网络分别称为编码器（Encoder）和解码器（Decoder），这个模型也称为 encoder-decoder 模型。解码器部分的结构与语言模型几乎完全相同：输入为单词的词向量，输出为 softmax 层产生的单词概率，损失函数为 log perlexity。

2.2 具有注意力机制的 seq2seq 模型

根据老师的要求，不能够用简单的 seq2seq 模型，要用具有带注意力机制的模型，于是我们接下来就详细的介绍每个部件的实现细节



2.2.1 注意力机制的优点

在 Seq2Seq 模型中，编码器将完整的输入句子压缩到一个维度固定的向量中，然后解码器根据这个向量生成输出句子。当输入句子较长时，这个中间向量难以存储足够的信息，就成为这个模型的一个瓶颈。注意力（“Attention ”）机制就是为了解决这个问题而设计的。注意力机制允许解码器随时查阅输入句子中的部分单词或片段，因此不再需要在中间向量中存储所有信息。这个过程可以类比于人的翻译过程：在翻译句子时，人们经常回头查阅原文的某个词或者片段，来提高别于这个词或者片段的翻译精确度。举个例子，假如一个人要把 “The sea is blue ", 翻译成中文，当他翻译出 “大海的颜色是蓝色” 的时候，如果突然想不起来原文中写的是什么颜色了，就会回头到原文相关部分去查阅。这时如果不允许他查询原文（类比于 Seq2Seq 模型），那么他就只能根据常理推断来选择一个最可能的颜色，准确率就会下降。

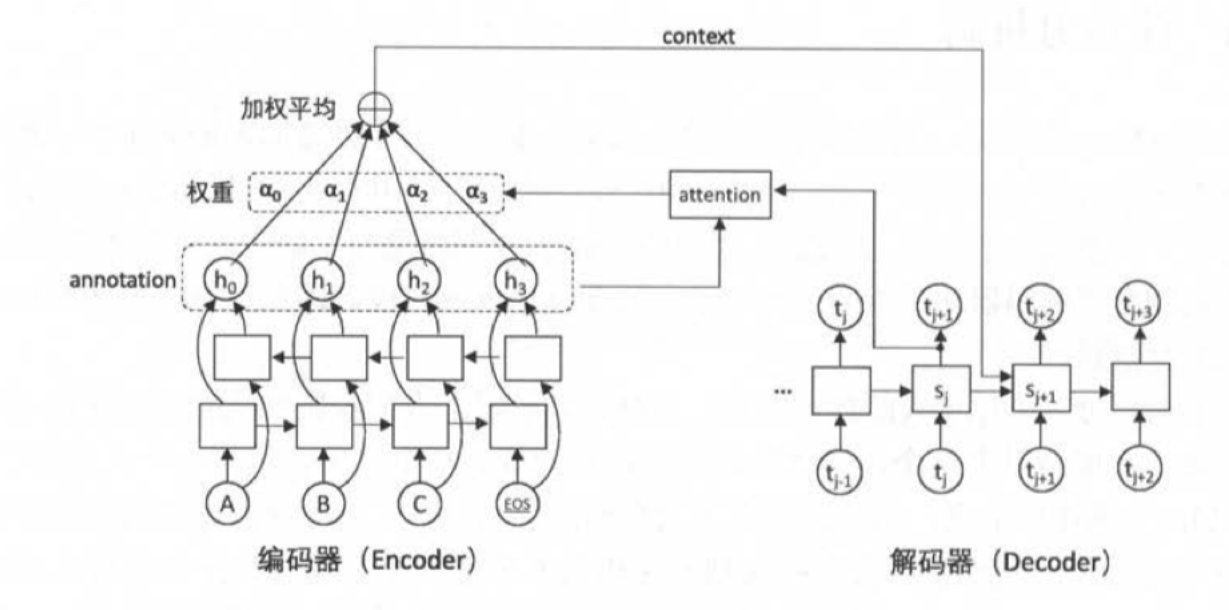
2.2.2 Encoder 的设计

- 用 embedding_lookup 对输入进 embedding
- 编码器采用了一个双向循环网络, 双向循环网络使得每个单词的可以同时包含左右两侧的信息, 使用 bidirectional_dynamic_rnn 构造双向循环网络.

- **Encoder 的输出是两个 LSTM 输出的拼接**

```
    # 定义编码器所使用的两个LSTM
    self.encLSTM1 = tf.nn.rnn_cell.BasicLSTMCell(HIDDEN_SIZE)
    self.encLSTM2 = tf.nn.rnn_cell.BasicLSTMCell(HIDDEN_SIZE)

    #定义源语言的词向量
    self.srcEmbedding = tf.get_variable("srcEmb",
                                        [CH_WORD_SIZE, HIDDEN_SIZE
                                            ])
    #将输入单词编号转为词向量。
    srcEmb = tf.nn.embedding_lookup(self.srcEmbedding, src_input)

with tf.variable_scope("encoder"):
    # 构造编码器时，使用bidirectional_dynamic_rnn构造双向循环网络。
    # 双向循环网络的顶层输出enc_outputs是一个包含两个张量的tuple，每个
        张量的维度都是[batch_size, max_time, HIDDEN_SIZE]，代表两个LSTM
        在每一步的输出。
    encOutputs, encState = tf.nn.bidirectional_dynamic_rnn(
        self.encLSTM1,
        self.encLSTM2,
        srcEmb,
        src_size,
        dtype=tf.float32)
    # 将两个LSTM的输出拼接为一个张量作为最后的输出
    encOutputs = tf.concat([encOutputs[0], encOutputs[1]], -1)
```

2.2.3 Decoder-Attention 机制



Attention 把 Decoder RNN 的每个隐层和 Encoder RNN 的每个隐层直接连起来了，还是“减少中间商赚差价”的思路，由于有这个捷径，Encoder RNN 的最后一个隐状态不再是“信息”瓶颈，信息还可以通过 Attention 的很多“直连”线路进行传输。下面将重点介绍注意力机制里面的数学公式以及意义

- **相关计算**

  当我们预测到第 j 个单词的时候：

  –

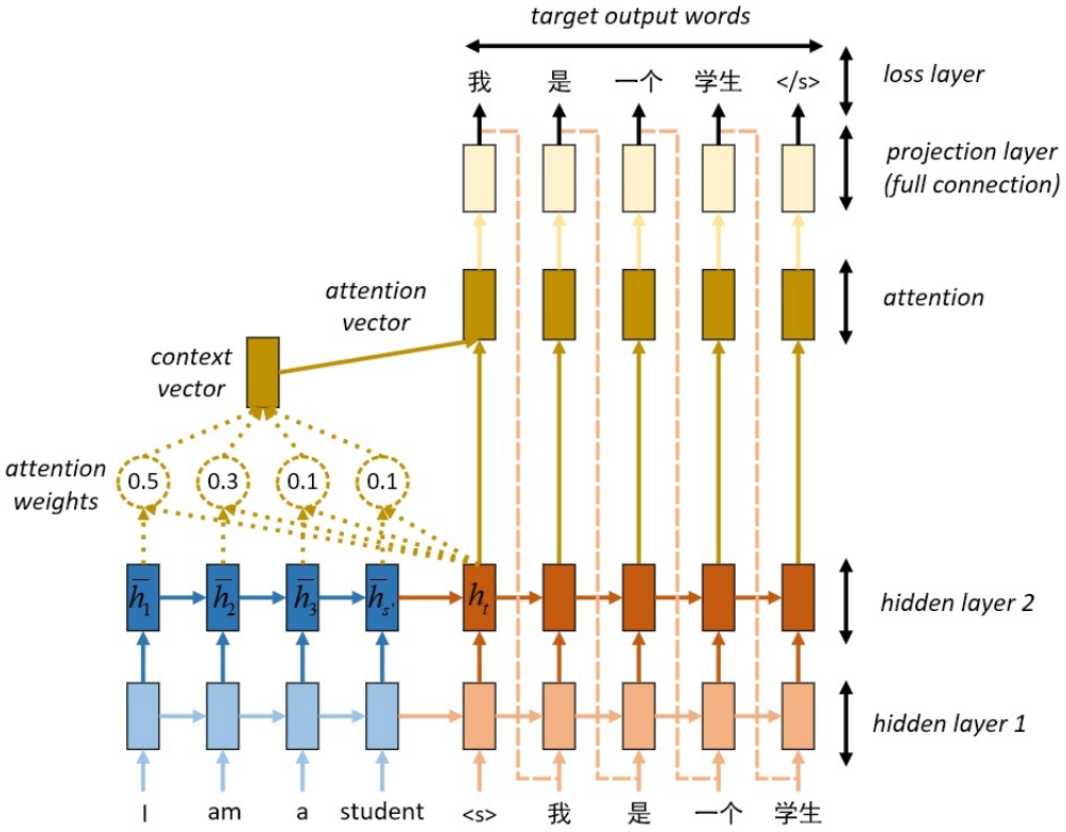
  $$\alpha_{i,j} = \frac{\exp\left(score\left(h_i, s_j\right)\right)}{\sum_i \exp\left(score\left(h_i, s_j\right)\right)}$$

  –

  $$context_j = \sum_i \alpha_{i,j} h_i$$

  其中 $score(h_i, s_j)$ **是计算原文各单词与当前解码状态的相关度的函数，最常用的 e(h, s) 函数定义是一个带有单个隐藏层的前馈神经网络：** $score(h, s) = Utanh(Vh + Ws)$**，**$score(h, s$ **构成了一个包含一个隐藏层的全连接神经网络**

为了更好的说明过程，我们用下面的图来解释



如图所示，图中右边的输入部分的实线表示是训练时的输入，虚线表示预测时的输入。注意向量 (attention vector) 是由解码部分每个时刻的计算产生的，此处以计算第一个时刻为例。

- **首先将当前时刻的状态 同编码部分的记忆状态逐个进行相似度计算，然后进行一个正则化**
- **在计算得到 attention weights 之后就开始计算 context vector**
- **第三步就是计算 attention vector,$\alpha_t = f(c_t, h_t) = \tanh(W_c[c_t; h_t])$, 完成 attention vector 的计算之后就是全连接层，然后与正确 label 构造得到损失函数做分类处理，将预测得到的输出喂给下一时刻得到下一个 ，之后就又是重复上面 3 个步骤了。**

但是在实际实现中，我们只需要调用库函数就好了，不需要手动实现，否则的话出错的概率太大了，所以直接调库是最保险的做法。

```
# 选择注意力权重的计算模型。BahdanauAttention是使用一个隐藏层的前馈神经网络。
# memory_sequence_length是一个维度为[batch_size]的张量，代表batch
# 中每个句子的长度，Attention需要根据这个信息把填充位置的注意力权重设置为0。
```

```
    attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(
        HIDDEN_SIZE, encOutputs, memory_sequence_length=src_size)

    # 将解码器的循环神经网络self.decLSTM和注意力一起封装成更高层的循环
      神经网络。
    attention_cell = tf.contrib.seq2seq.AttentionWrapper(
        self.decLSTM,
        attention_mechanism,
        attention_layer_size=HIDDEN_SIZE)

    # 使用attention_cell和dynamic_rnn构造编码器。
    # 这里没有指定init_state，也就是没有使用编码器的输出来初始化输入，
      而完全依赖
    # 注意力作为信息来源。
    DecOutputs, _ = tf.nn.dynamic_rnn(attention_cell,
                                      tarEmb,
                                      trg_size,
                                      dtype=tf.float32)
```

3 模型训练

- 对于每一个 epoch 在 model 上面训练，重复训练直到遍历完 Dataset 里面的所有数据。
- 运用 log perplexity 计算误差，通过交叉熵计算 loss。

```
# 计算解码器每一步的log perplexity。
output = tf.reshape(DecOutputs, [-1, HIDDEN_SIZE])
logits = tf.matmul(output, self.softmax_weight) + self.softmax_bias
loss = tf.nn.sparse_softmax_cross_entropy_with_logits(
    labels=tf.reshape(trg_label, [-1]), logits=logits)

# 在计算平均损失时，需要将填充位置的权重设置为0，以避免无效位置的预测干
  扰
```

```
# 模型的训练。
label_weights = tf.sequence_mask(trg_size,
                                 maxlen=tf.shape(trg_label)[1],
                                 dtype=tf.float32)
label_weights = tf.reshape(label_weights, [-1])
cost = tf.reduce_sum(loss * label_weights)
cost_per_token = cost / tf.reduce_sum(label_weights)

# 定义反向传播操作。
trainable_variables = tf.trainable_variables()

# 控制梯度大小，定义优化方法和训练步骤。
grads = tf.gradients(cost / tf.to_float(batch_size),
                     trainable_variables)
grads, _ = tf.clip_by_global_norm(grads, MAX_GRAD_NORM)
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.9)
train_op = optimizer.apply_gradients(zip(grads, trainable_variables))
return cost_per_token, train_op
```

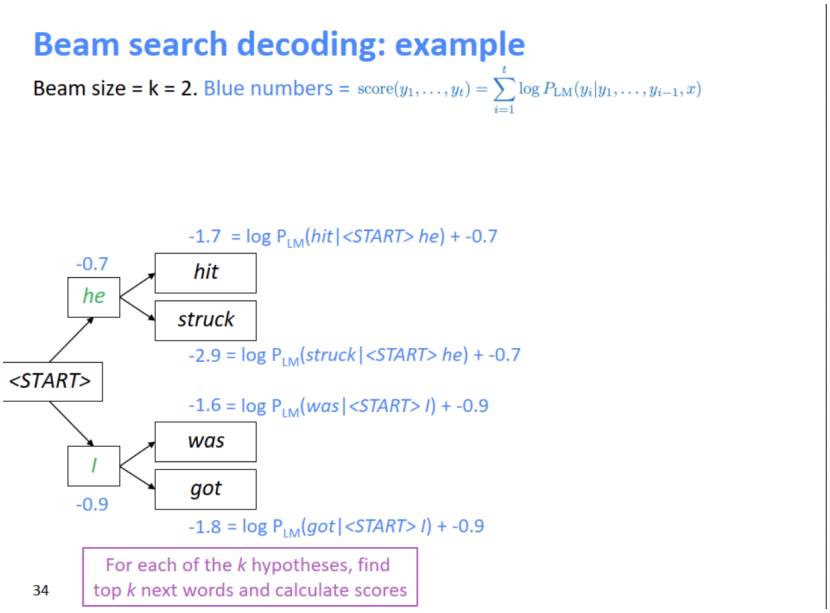
```
def train_net(session, cost_op, train_op, saver, step):
    # 训练一个epoch。
    # 重复训练步骤直至遍历完Dataset中所有数据。
    while True:
        try:
            # 运行train_op并计算损失值。训练数据在main()函数中以Dataset
               方式提供。
            cost, temp = session.run([cost_op, train_op])
            if step % 10 == 0:
                print("After %d steps,loss is %.3f" % (step, cost))
            # 每100步保存一个checkpoint。
            if step % 100 == 0:
                saver.save(session, CHECKPOINT_PATH, global_step=step)
```

```
            step += 1
        except tf.errors.OutOfRangeError:
            break
    return step
```

4 模型预测

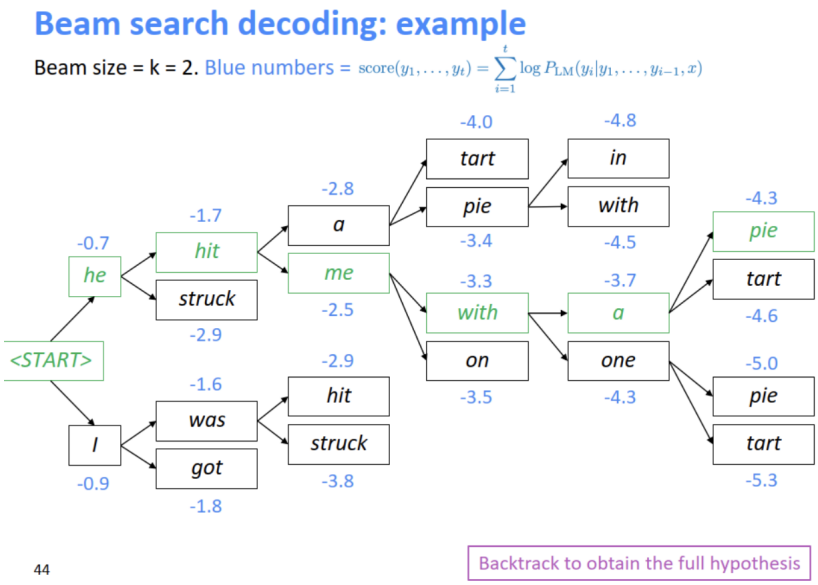
4.1 Beam Search

Beam search 搜索策略是贪心策略和穷举策略的一个折中方案，它在预测的每一步，都保留 Top-k 高概率的词，作为下一个时间步的输入。k 称为 beam size，k 越大，得到更好结果的可能性更大，但计算消耗也越大。请注意，这里的 Top-k 高概率不仅仅指当前时刻的 y t 的最高概率，而是截止目前这条路径上的累计概率之和.



举例如下，假设 k=2，第一个时间步保留 Top-2 的词为"he" 和"I"，他们分别作为下一个时间步的输入。"he" 输入预测输出前两名是"hit" 和"struck"，则"hit" 这条路的累加概率是"he" 的概率加上"hit" 的概率 =-1.7，类似的可以算出其他几个词对应路径的概率打分。最后在这 4 条路上保留 k=2 条路，所以"hit" 和"was" 对应路径保留，作为下一个时间步的输入；"struck" 和"got" 对应路径被剪枝。最终的搜索树如下图所示，可以看到在每个时间步都只保留了 k=2 个节点往下继续搜索。最后"pie" 对应的路径打分最高，通过回溯法得到概率最高的翻译句子。请注意，beam search

作为一种剪枝策略，并不能保证得到全局最优解，但它能以较大的概率得到全局最优解，同时相比于穷举搜索极大的提高了搜索效率。



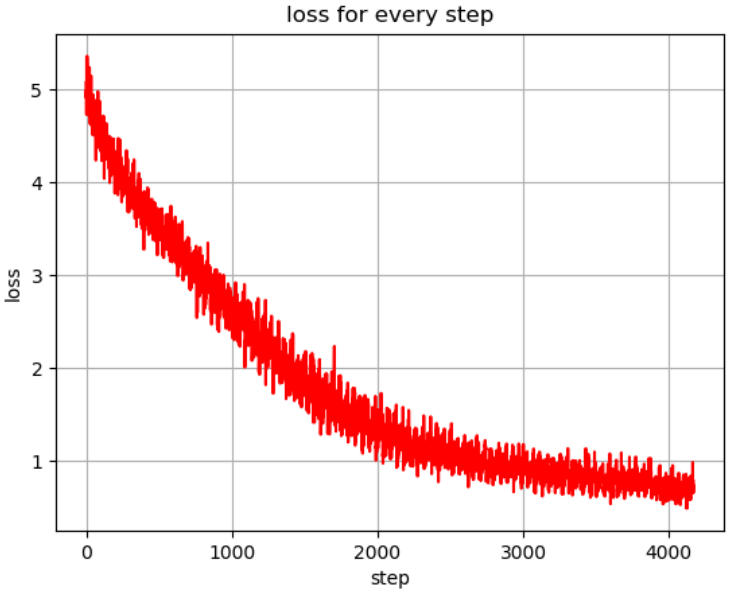
在 beam search 的过程中，不同路径预测输出结束标志符 <END> 的时间点可能不一样，有些路径可能提前结束了，称为完全路径，暂时把这些完全路径放一边，其他路径接着 beam search。beam search 的停止条件有很多种，可以设置一个最大的搜索时间步数，也可以设置收集到的最多的完全路径数。 当 beam search 结束时，需要从 n 条完全路径中选一个打分最高的路径作为最终结果。由于不同路径的长度不一样，而 beam search 打分是累加项，累加越多打分越低，所以需要用长度对打分进行归一化，如下图所示。那么，为什么不在 beam search 的过程中就直接用下面的归一化打分来比较呢？因为在树搜索的过程中，每一时刻比较的两条路径的长度是一样的，即分母是一样的，所以归一化打分和非归一化打分的大小关系是一样的，即在 beam search 的过程中就没必要对打分进行归一化了。

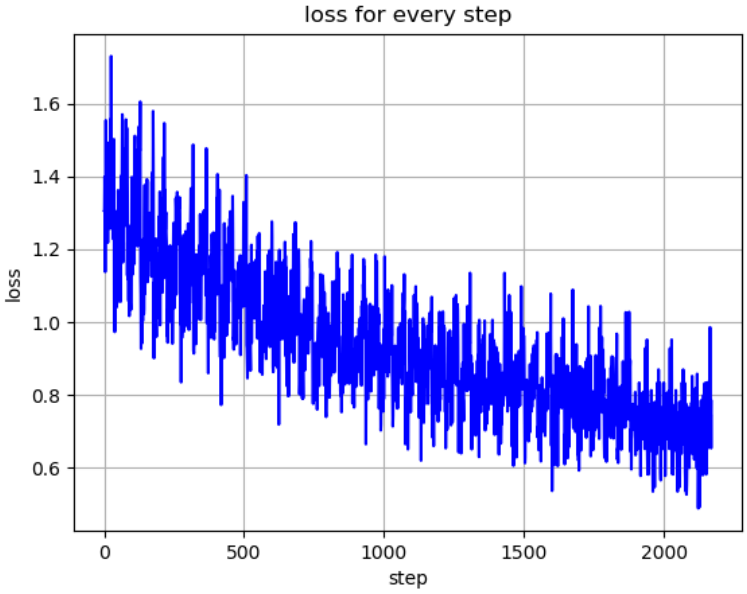
```
for round in range(100) :
    if k <=0 :
        break
    # 所有可能的序列
    all_candidates = []
    for i in range(len(sequences)):
        # 获取句子的下标和分数
        seq, score, h, c = sequences[i]
        # 获取输入单词的词向量表示
```

```
10          pre_word_embed=self.english_embedding(keras.tensor(seq[-1]).to(
                device))
11          # 输入神经网络中进行预测
12          o , h, c=self.decoder(pre_word_embed,h,c)
13          # 将输出变成0-1的
14          output=F.softmax(o,dim=2)
15          for index in range(output.shape[2]):
16              prob=-math.log(output[0][0][index])
17              candidate=[seq+[index], score * prob, h, c]
18              all_candidates.append(candidate)
19      # 所有候选根据分值从大到小排序
20      ordered = sorted(all_candidates, key=lambda tup: tup[1],reverse=
            True)
21      # 选择前k个
22      sequences = ordered[:k]
23      # 判断是否有句子到了EOS
24      for i in range(len(sequences)):
25          if sequences[i][0][-1]==EOS_ID:
26              result_sequences.append(sequences[i])
27              k-=1
28      if len(result_sequences)==0:
29      result_sequences=sequences
30  return sorted(result_sequences, key=lambda tup: tup[1],reverse=True)
        [0][0]
```

5 实验结果

5.1 训练过程





通过上面的图可以发现 loss 确实是一直在减少，但是最后下降的程度就逐渐平缓了，我想了想这可能跟我之前将很多的单词都变成了 UNK 有关，还有一个原因就是我的计算资源真的有限，并不能跑出来很好的效果，没有 GPU，只能用电脑的 CPU 硬跑，导致其实不仅训练的时间长，效果也没有想象中的好。

5.2 案例分析

下面抽取一些训练结果来进行说明

1

原文：埃 博拉 之外 的 非洲
标准译文：Africa Beyond Ebola
模型输出译文：africa <UNK> to produce the <UNK>
Bleu：0.20

2

原文：尽管如此，事后 看来，遏制 给 美国 外交政策 带来 了 如今 所 缺乏 的 秩序 和 组织
标准译文：Nonetheless, in hindsight, containment infused an order and organization on US foreign policy that is absent today.
模型输出译文：nonetheless , it was virtually to imagine the usand <UNK> to advance the <UNK> over the cold presidential , and just before the <UNK> and republic of the iraq
Bleu：0.26

3

原文：对 叙利亚 总统 巴沙尔 阿萨德 政权 的 坚定 支持者 伊朗 来说，美国 沙特 关系 的 削弱 是 改变 地区 实力 平衡 的 关键 特别 是 伴随 着 西方 经济 制裁 放松 的话。
标准译文：For Iran, a staunch supporter of Syrian President Bashar al-Assad’ s regime, a weaker US-Saudi relationship is key to shifting the regional balance of power – especially if it is accompanied by the easing of Western economic sanctions .
模型输出译文：for iran , the need for such nuclear weapons regime is another <UNK> anger , nor regard of the leadership of iran s role in the powers
Bleu：0.30

5.3 Teacher Forcing 效果

Teacher Forcing: 直接使用训练数据的标准答案 (ground truth) 的对应上一项作为当前时间步的输入；我在实验的过程中发现如果 Teacher Forcing 的值太小的话，会导致我的模型不能收敛，同时也会出现很大的震荡，如果调大 Teacher Forcing 的值的话，就能够比较正常，保证收敛

5.4 集束搜索效果

集束搜索效果在我的实验过程中除了增加了代码量，增加了运算时间，似乎并没有提升准确率，所以最后我最后是去掉了集束搜索，让他能够比较快的得到结果。

6 实验心得

为期一学期的自然语言处理终于结束了，终于结课了，这期间真的是很痛苦，因为没有机器学习的基础，所以一上来就要搭网络，这样的跨度实在是太大了。导致刚开始的时候差点想放弃，所幸坚持下来了，完成了这门课的学习，在学习的阶段我住不体会到了什么叫做大数据时代，什么是数据驱动模型的时代，这就是现在我们生活的时代，如何教会计算机识别我们的文字系统确实不是一件容易的事情，这是一个 AI 大火的时代，不过有时候我也在想 AI 的大火引起所以的人的关注不是一件偶然的事情，因为他跟我们的生活实在是太息息相关了，没有人现在能够人心摒弃 AI 带给我们生活的便利，这次的实验让我体会到了其实 AI 就是按照我们小时候学习的方法教会计算机学习的方法，任何一种机制的提出都是按照这个过程来的，还有就是我深刻的体会到了参数的重要性，真的是极其重要的，之前参数没有设置好，导致训练的时候会出现各种各样的问题，还有就是最开始的时候训练不到位的时候训练结果全部都是 the us 真的是很让人郁闷，之后修改了一些参数，调整了一下网络之后，虽然还是会有很大的问题，但是基本不再是之前的 the，us 一样重复的词，如何能够更好的设计模型，如何能够更好的设置好的参数，是我需要在今后的学习生涯中重点掌握的。通过这门课的学习我学会了一些自然语言处理的基本方法，知道了传统的统计方法和深度学习方法的区别，同时也了解到深度学习在大数据时代的重要性，不管怎么样，在今后的生活中还是需要多了解这些前沿知识，为今后的人生打好基础。路漫漫其修远兮，吾将上下而求索。

7 运行环境

```
python 3.7
tensorflow ==1.13.2
jieba ==0.42
nltk ==3.4.5
```

8 参考文献

References

1. https://web.stanford.edu/class/cs224n/ 斯坦福 CS224N
2. 《Tensorflow 实战 Google 深度学习框架》
3. https://zhuanlan.zhihu.com/p/62813990 带注意力机制的 Seq2Seq 翻译模型
4. https://zhuanlan.zhihu.com/p/36440334 Seq2seq 模型及注意力机制
5. http://bitjoy.net/2019/08/02/cs224n%ef%bc%881-31%ef%bc%89translation-seq2seq-attention/ 机器翻译
6. https://zhuanlan.zhihu.com/p/53036028 深度学习中的注意力机制