1. **基于词的IBM Model1模型理解**

对于翻译模型来说，如果直接求把一个句子翻译成另外一个句子的概率，那么需要列举出所有的可能翻译出的句子的概率，并且比较得到最大的概率，这个计算量是十分巨大的，因为句子之间的组合数量太多，并且很多句子的出现次数可能只有一次甚至为0，所以无法准确的获得最准确的翻译。因此，提出了基于词的翻译，也就是在翻译句子的时候，并不是直接求最终的句子，而是求原句子中每个词的最优可能翻译出的词，之后对翻译的词进行组合成最终的翻译句子。

对于IBM Model1模型来说，首先，输入是两段平行语料，输出是两段语料中词的对应关系。可以把这种对应关系设定为一种参数为a，把输入的外语设为f，把对应的英语设为e。那么实际IBM Model1模型就是要求。

在IBM Model1模型中，把这个概率定义为

其中是一个补充项，是为了补充最终翻译出的一个句长的概率。是一个归一化项，表示从f翻译成e的所有可能性。而后面的这是表示当前这种对应关系的相应的出现次数。那么这个式子能够很好的表示的概率。那么接下来的问题就变成了如何来求这个模型所要表达的概率。

这个模型可以看成是一个含有隐变量的问题，把a看作是隐变量，那么也就是可以使用EM算法来解决这么模型的参数估计。

1. **IBM Model 1的EM算法**

EM算法分为2步，第一步是估计这个概率的期望，第二步就是使得这个估计的期望最大化。因此最终是要求，也就是每个单词e和单词f的映射关系：

其中：

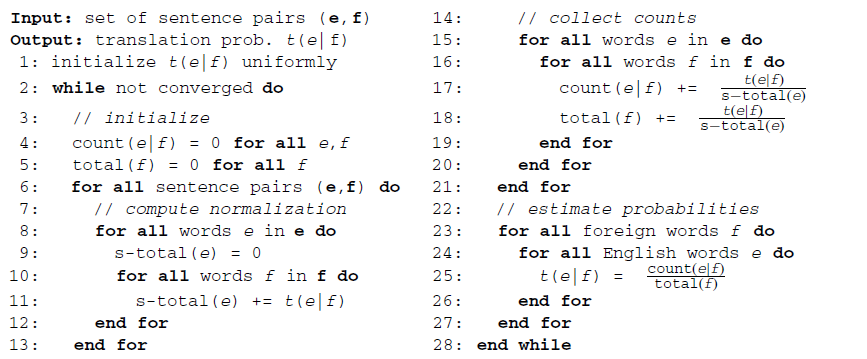
因此可以得到：

再看上面的式子，通过公式推导，可以得到:

在这其中，使用到了一个技巧，把连乘的连加变成了连加后的连乘，能够大大降低计算的复杂度。

1. **EM算法实现**

把上述的过程转化为算法，可以得到：



在使用代码实现过程中，基本思路和本算法相同，只是在计算之前对数据进行了预处理，没有使用文字，而是对每个文字进行了编码，使用一个数字来代替每个单词，这样，能够大大的降低系统对内存的消耗。

假定当t值的平均变化小于0.001停止训练，那么整个模型训练需要419s，每一次的平均修改值为：0.00212410092247，0.00246488671093，0.00195776723896，0.00137608143849。由此可见收敛速度十分快。

1. **源代码附录**

data\_processing.py

DataProcess类对平行语料进行预处理，主要进行将语料读取到内存，将单词转换为索引，将句子转换为索引的序列。

class DataProcess:  
 def \_\_init\_\_(self, src\_file):  
 *"""  
 初始化  
 src\_file为语料文件  
 sentences为语料中的句子  
 sentences\_embedding为语料转化为索引后句子的list  
 word\_to\_index为单词到索引的dict  
 index\_to\_word为索引到单词的dict* ***:param*** *src\_file:  
 """* self.\_\_src\_file = src\_file  
 self.\_\_sentences = []  
 self.\_\_word\_to\_index = {}  
 self.sentences\_embedding = []  
 self.index\_to\_word = {}  
  
 def \_\_read\_data(self):  
 *"""  
 读取语料* ***:return****:  
 """* with open(self.\_\_src\_file) as fp:  
 self.\_\_sentences = fp.readlines()  
  
 def \_\_word\_index(self):  
 *"""  
 将单词转化为索引* ***:param*** *sentences: 语料* ***:return****:  
 """* index = 0  
  
 for sentence in self.\_\_sentences:  
 for word in sentence.split():  
 word = word.lower()  
 if word not in self.\_\_word\_to\_index.keys():  
 self.\_\_word\_to\_index[word] = index  
 self.index\_to\_word[index] = word  
 index += 1  
  
 def \_\_sentence\_to\_embedding(self):  
 *"""  
 将句子转换为索引的序列* ***:return****:  
 """* for sentence in self.\_\_sentences:  
 sentence\_index = []  
 for word in sentence.split():  
 sentence\_index.append(self.\_\_word\_to\_index[word.lower()])  
 self.sentences\_embedding.append(sentence\_index)  
  
 def process(self):  
 *"""  
 运行读取数据，转换索引，以及将句子转换为embedding* ***:return****:  
 """* self.\_\_read\_data()  
 self.\_\_word\_index()  
 self.\_\_sentence\_to\_embedding()

EM\_algorithm.py

EMProcessing类主要实现EM算法，包含初始化t的参数，以及主程序。

class EMprocessing:  
 def \_\_init\_\_(self, foreign\_sentences\_embedding, native\_sentences\_embedding, foreign\_index\_to\_word,  
 native\_index\_to\_word):  
 self.foreign\_sentences\_embedding = foreign\_sentences\_embedding  
 self.native\_sentences\_embedding = native\_sentences\_embedding  
 self.foreign\_index\_to\_word = foreign\_index\_to\_word  
 self.native\_index\_to\_word = native\_index\_to\_word  
  
 self.t = {}  
 self.denominator = {}  
  
 for i in range(len(self.foreign\_sentences\_embedding)):  
 for native\_word in self.native\_sentences\_embedding[i]:  
 for foreign\_word in self.foreign\_sentences\_embedding[i]:  
 if (native\_word, foreign\_word) in self.t.keys():  
 self.t[(native\_word, foreign\_word)] += 1  
 else:  
 self.t[(native\_word, foreign\_word)] = 1  
 if foreign\_word in self.denominator.keys():  
 self.denominator[foreign\_word] += 1  
 else:  
 self.denominator[foreign\_word] = 1  
  
 for key in self.t.keys():  
 self.t[key] = (1.0 / self.denominator[key[1]])  
  
 def em\_algorithm(self):  
 threshold = 1e-3  
 avg\_change = 1  
  
 s = {} # key is native embedding, value is number  
 count = {} # key is (native\_embedding, foreign\_embdding), value is number  
 total = {} # key is foreith\_embdding, value is number  
  
 while avg\_change > threshold:  
 print(avg\_change)  
 sum\_change = 0.0  
 count.clear()  
 total.clear()  
  
 for i in range(len(self.foreign\_sentences\_embedding)):  
 s.clear()  
 for native\_word in self.native\_sentences\_embedding[i]:  
 s[native\_word] = 0.0  
 for foreign\_word in self.foreign\_sentences\_embedding[i]:  
 s[native\_word] += self.t[(native\_word, foreign\_word)]  
  
 for native\_word in self.native\_sentences\_embedding[i]:  
 for foreign\_word in self.foreign\_sentences\_embedding[i]:  
 if (native\_word, foreign\_word) in count.keys():  
 count[(native\_word, foreign\_word)] += self.t[(native\_word, foreign\_word)] / s[native\_word]  
 else:  
 count[(native\_word, foreign\_word)] = self.t[(native\_word, foreign\_word)] / s[native\_word]  
 if foreign\_word in total.keys():  
 total[foreign\_word] += self.t[(native\_word, foreign\_word)] / s[native\_word]  
 else:  
 total[foreign\_word] = self.t[(native\_word, foreign\_word)] / s[native\_word]  
  
 for foreign\_word in self.foreign\_index\_to\_word:  
 for native\_word in self.native\_index\_to\_word:  
 if (native\_word, foreign\_word) in count.keys():  
 new\_t = count[(native\_word, foreign\_word)] / total[foreign\_word]  
 sum\_change += abs(new\_t - self.t[(native\_word, foreign\_word)])  
 self.t[(native\_word, foreign\_word)] = new\_t  
  
 avg\_change = sum\_change / len(self.t)

run.py

主要控制程序的运行流程以及最终结果的存储。

import os  
import importlib  
import sys  
from data\_processing import DataProcess  
from EM\_algorithm import EMprocessing  
  
importlib.reload(sys)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 foreign\_file = os.path.join(os.path.abspath('..'), 'IBM-model-1/data', 'fbis.en.10k')  
 native\_file = os.path.join(os.path.abspath('..'), 'IBM-model-1/data', 'fbis.zh.10k')  
  
 # 数据预处理  
 foreign\_data\_process = DataProcess(foreign\_file)  
 native\_data\_process = DataProcess(native\_file)  
  
 foreign\_data\_process.process()  
 native\_data\_process.process()  
  
 foreign\_sentences\_embedding, foreign\_index\_to\_word = foreign\_data\_process.sentences\_embedding, foreign\_data\_process.index\_to\_word  
 native\_sentences\_embedding, native\_index\_to\_word = native\_data\_process.sentences\_embedding, native\_data\_process.index\_to\_word  
  
 # em算法执行  
 em\_process = EMprocessing(foreign\_sentences\_embedding, native\_sentences\_embedding, foreign\_index\_to\_word,  
 native\_index\_to\_word)  
  
 em\_process.em\_algorithm()  
 t = sorted(em\_process.t.items(), key=lambda kv: -kv[1])  
  
 # 结果存储  
 if not os.path.exists(os.path.join(os.path.abspath('..'), 'IBM-model-1/result')):  
 os.mkdir(os.path.join(os.path.abspath('..'), 'IBM-model-1/result'))  
  
 result\_file\_src = os.path.join(os.path.abspath('..'), 'IBM-model-1/result', 't.txt')  
  
 with open(result\_file\_src, 'w') as file\_write:  
 for k, v in t[:1000]:  
 native\_word = native\_index\_to\_word[k[0]]  
 foreign\_word = foreign\_index\_to\_word[k[1]]  
 file\_write.write("%s\t%s\t%s\n" % (native\_word, foreign\_word, v))

1. **结果附录**

摘取了结果中概率最高的50个词对，分别为中文词\t英文词\t对齐概率\n

市场 market 0.950454653271199  
国际 international 0.9460383800981826  
改革 reform 0.9433844447999319  
投资 investment 0.9428341006795227  
积极 actively 0.9428171616659434  
进一步 further 0.9366889758305861  
竞争 competition 0.9353408168340339  
新 new 0.9349349747692974  
活动 activities 0.9348036963474369  
历史 historical 0.9325934759193465  
积极 positive 0.9310327634522666  
希望 hope 0.9309010036148633  
文化 cultural 0.9285013823432247  
! ! 0.9282958153958407  
强调 stressed 0.9249548857499078  
建设 construction 0.9228821157207905  
群众 masses 0.9211502174482347  
人口 population 0.9206176267300762  
非法 illegal 0.9185454270891651  
社会 society 0.9181880336910272  
有关 relevant 0.9165049158650957  
农业 agriculture 0.9159733170449628  
公司 company 0.9142917569693766  
教育 education 0.9134970150802203  
教授 professor 0.9126189029101606  
机制 mechanism 0.912473285238869  
社会主义 socialist 0.9122358155031688  
中央 central 0.9117605780644665  
农民 peasants 0.9112641183893078  
继续 continue 0.9111124387268246  
市场 markets 0.9098176799299479  
政府 government 0.9093596831561632  
社会 social 0.9091636739511668  
指出 pointed 0.9054674185857723  
信息 information 0.9048279700626487  
农村 rural 0.9047739340075834  
坚决 resolutely 0.9043990340045692  
特别是 especially 0.9017066463268741  
潜艇 submarine 0.9013305475288788  
经济 economic 0.8997383457061934  
民主 democratic 0.8993081521325277  
管理 management 0.8990843276309775  
压力 pressure 0.8989877252008361  
建设 building 0.8961361609757444  
传统 traditional 0.8946135310056449  
基本 basically 0.8940870279319401  
部门 departments 0.8893660661548717  
直接 directly 0.8891820450521529  
落后 backward 0.8883521236116854  
北约 nato 0.8870235486943236