拼音输入法实验报告

计91 刘松铭 2018011960

摘要

在本实验中，我尝试了**二元模型**，**三元+二元模型**，**三元+二元修正模型**，**注意力模型**等四个模型实现了拼音输入法，即从拼音序列转换为汉字序列的算法。我对各个模型进行了测试和对比，结果显示效果最好的是三元+二元模型，在众包测试集上的准确率达到**0.9402**(字)/**0.7274**(句)。

注1：根目录为`pinyin`，众包测试集的输入文件在`/data/test_input.txt`，标准输出文件在`/data/test_std_output.txt`，算法输出文件在`/data/test_output.txt`。此外，原众包测试集有很多输入错误(人为因素造成的)，我和汪子涵同学都对此进行了校对，在此非常感谢汪子涵同学的工作。本实验中的测试集使用的是**校对后**的版本。

注2：本实验的测试环境为11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz 16.0GB RAM Win10 x64。

使用说明

项目文件夹的**根目录**为`pinyin`，该文件夹的目录树如下：

```
pinyin/
|  main.py (主程序入口)
|  test.py (参数调优代码)
|  report.pdf (实验报告, 即本文件)
├─data
|      input.txt
|      output.txt
|      test_input.txt
|      test_output.txt
|      test_std_output.txt
|
├─database
|      *.json (json文件)
|      *.db (数据库文件)
|
├─model
|     viterbi.py
|     viterbi_3.py
|     viterbi_att.py (模型代码)
|
└─source
    |  load_data.py
    |  train.py
    |
    └─preprocess
       *.py (预处理代码)
```

- `/data`目录存放输入输出的txt文件。
- `/database`目录存放本地的数据库，即算法运行所需的一些概率数据。
- `/model`目录存放了模型代码。
- `/source`目录存放了一些预处理和训练的代码。

本程序的**使用方式**是在根目录下运行`main.py`文件，具体的使用方法如下：

- 模式一，无命令行参数。如`python main.py`。

  该模式下直接进入交互输出，即用户在标准输入中输入拼音，标准输出返回汉字序列。

- 模式二，指定输入输出文件路径。

  如`python main.py ./data/input.txt ./data/output.txt`。

  第一位是输入文件路径

  第二位是输出文件路径

  第三位为编码格式, 默认utf-8, 可不输入

- 模式三，指定输入输出标准输出文件路径, 并与标准答案对比，打印正确率。

  如`python main.py ./data/test_input.txt ./data/test_output.txt ./data/test_std_output.txt 1`。

  第一位是输入文件路径

  第二位是输出文件路径

  第三位是标准文件路径

  第四位是标志位, 请输入"1"

  第五位为编码格式, 默认utf-8, 可不输入

若要**更换模型**(`viterbi`对应二元模型，`viterbi_3`对应三元+二元模型(已修正)，`viterbi_att`对应注意力模型，默认为三元+二元模型)，可以直接修改`/main.py`中的import部分，例如：

```
# 本程序是测试入口
from model.viterbi import viter
# from model.viterbi_3 import viter
# from model.viterbi_att import viter
import sys
import time
```

此时`/main.py`采用的是二元模型，若要切换为三元模型只需修改注释即可：

```
# 本程序是测试入口
# from model.viterbi import viter
from model.viterbi_3 import viter
# from model.viterbi_att import viter
import sys
import time
```

二元模型

二元模型是完整实现拼音输入法的初版模型，只采用二元模型进行实现，没有加入任何的优化，使用的数据集是原始新闻数据集。

算法思路

本模型采用维特比算法求解给定拼音序列对应的概率最大的汉字序列。具体算法介绍如下：

隐式马尔科夫模型(HMM)状态空间$\mathbb{S}=\{s_1,s_2,\ldots,s_K\}$，观察值空间为$\mathbb{O}=\{o_1,o_2,\ldots,o_N\}$。在本问题的语境下，状态空间即是全体国标一二级汉字组成的空间，观察值空间则是全体拼音组成的空间。

给定观察值序列$Y=\{y_1,y_2,\ldots,y_T\}$，即用户输入的拼音序列。算法输出对应的概率最大的状态序列$X=\{x_1,x_2,\ldots,x_T\}$，即用户得到的对应汉字序列。这个概率最大的汉字序列可由如下的递推方程得出：

$$V_{1,k}=E_{k,1}\cdot\pi_k$$
$$V_{t,k}=\max_{1\le x\le K}\left(E_{k,t}\cdot A_{x,k}\cdot V_{t-1,x}\right)$$

其中：

- $E_{k,i}=P(y_i|s_k)$是发射矩阵，指的是状态s_k的条件下观测到y_i的概率。在本题语境下，也就是某个字对应其拼音的概率。例如对于汉字`参`，它有四个拼音：`can`，`shen`，`cen`，`san`。得到发射矩阵的方法是利用`pypinyin`库对语料库中`参`出现的位置进行拼音转换，得到其对应的拼音，进而统计出`参`对应不同拼音的概率。考虑发射概率就相当于考虑了**多音字**。
- π_k指的是初始状态为s_k的概率，即每个字作为一句话开头的概率。要知道这个值，我们只需要统计语料库每个字作为一句话句话开头出现的频率即可。
- $A_{x,k}=P(s_k|s_x)$是由状态s_x转移到s_k的概率，也就是当前一个字为s_x，后一个字为s_k的概率。这就是**二元模型**的一个基本假设，即后一个字出现仅依赖前一个字。
- $V_{t,k}$指的是观测序列的前t个，对应最后一个状态为s_k的状态序列的概率。

根据上述递推方程，我们就可以采用动态规划的方法求解X。即：

$$x_T=\arg\max_{1\le x\le K}(V_{T,x})$$
$$x_{t-1}=\mathrm{ptr}(x_t,t)$$

其中：

- $\mathrm{ptr}()$函数总是返回累计概率最大路径的后向指针。

在具体实现，我们可以使用滚动数组填表计算$V_{t,k}$，并用链表维护路径(后向指针)，将路径头存在path字典中。这样计算出最大概率路径后，只需依次遍历链表输出各个汉字即可得到X。

实现细节

整体算法流程如下：

- 预读取状态集合。读取所有的汉字，并且存为json文件。同时读取汉字对应的拼音，建立拼音到汉字的映射(即给出拼音`can`，返回含有这个拼音的所有汉字，如`参`等)，保存为json文件。保存为json文件的目的是便于后续的处理和筛选。
- 数据清洗。从json文件中读取语料库的文本，利用正则表达式将特殊符号以及数字等进行清洗(如去掉双引号，将双引号中的句子提取出来)，同时以逗号，句号，问号，感叹号等标点符号为界切分成句子。数据清洗的代码在`source\preprocess`下。**数据清洗**是非常重要的一环，直接影响了最后的模型预测准确率。
- 训练模型。对处理好的句子进行统计，算出发射概率，转移概率，初始状态概率等。
- 建立数据库。为了**减少内存开销**，以及**提高可扩展性**(如果之后要扩展到四元乃至更复杂的模型，数据量会急剧膨胀可能达十几G，届时直接读入内存是不现实的)，我将转移矩阵等多个统计数据存入了本地**sqlite**数据库中，在程序运行时通过**SQL**语句从数据库中读取需要的数据进入内存。由于数据库在本地，因此可以预见程序运行效率会有一定的下降。为了进一步提升程序运行效率，我采用了**数据库索引**以及**预缓存**的方法提升效率，避免过多的访问本地数据库。
- 根据上述算法部分的思想，利用动态规划实现维特比算法进行预测。

一些实现细节：

- 由于词的总量很多，因此会出现概率值非常小(接近0的浮点数)的情况。一方面要存储这样的浮点数需要高精度的浮点类型，另一方面在实际迭代中，概率是相乘的关系，多次相乘后浮点数会变得非常的小，不便于比较计算。因此本程序的所有概率值都取了**负对数**，相乘变成了相加，取最大变成取最小。

- 对于转移概率和初始状态概率，会出现有些字**从来没有**出现在句子开头或者有些字的组合从来没有作为转移对出现过，这样会造成概率的不良定义。解决这个问题的办法是采用laplace平滑。
    - 首先统计每个字出现的频率$P(s_k)$，统计该概率时采用加一平滑。
    - 将π_k替换为$\pi_k^* = (1-\alpha)P(s_k) + \alpha \cdot \pi_k$.
    - 将$A_{x,k}$替换为$A_{x,k}^* = (1-\beta)P(s_k) + \beta \cdot P(s_k|s_x)$

实验结果

好的例子

> cai fa xian zi ji mei you xiang hao yao zuo shen me cheng wei zen yang de ren
>
> 才发现自己没有想好要做什么成为怎样的人
>
> jing ji jian she he wen hua jian she tu chu le shi ba da jing shen de zhong yao xing
>
> 经济建设和文化建设突出了十八大精神的重要性
>
> yi xi jin ping tong zhi wei zong shu ji de dang zhong yang
>
> 以习近平同志为总书记的党中央
>
> zhong guo pin kun di qu shi xian wang luo fu wu quan fu gai
>
> 中国贫困地区实现网络服务全覆盖
>
> da liang can chu la ji yu qi ta la ji hun he tian mai huo fen shao
>
> 大量餐厨垃圾与其他垃圾混合填埋或焚烧

注意到，输入法在一些长且用词并不浅显的句子上也有比较好的表现。但这样的句子有两个明显的特点：1. 主体部分由**固定的词组或短语**组成；2. 语体上明显接近**新闻语体**。这说明输入法的表现严重依赖于语料库。

不好的例子

这部分将结合具体例子分析输入法的局限性，并且给下一步工作指明方向。

> **他**是我的母亲
>
> 正解：**她**是我的母亲

这是典型的**语义错误**，说明输入法无法基于语义进行筛选判断。值得注意的是，搜狗输入法也有这样的问题(见下图)。

类似的例子还有：

> 我和你**问**别
>
> 正解：我和你**吻**别

以下是另一些错误例子：

> 微笑着面对**他**
>
> 正解：微笑着面对**它**

这是**固有歧义**，也就是按两种汉字序列都可以解释拼音序列，并且语义上也没有问题。

> 我发现网可降的这些我听不**动**
>
> 正解：我发现网课讲的这些我听不**懂**

这是由于**二元模型的局限性**导致的错误。当识别最后一个拼音 dong 的时候，二元模型只看到了 不，跟其概率最大的pair确实有可能是 动。但如果采用三元模型，则会看到 听不，此时识别概率最大的应该就是 懂。

> 请不要输入奇怪的**巨资**
>
> 正解：请不要输入奇怪的**句子**

这是由于**语料库**导致的错误。由于训练采用的是新闻语料库，因此 巨资 出现的概率要比 句子 高很多，而这在日常用语中是不合理的。

下面是一些难以解决的错误，这些错误暂时不在改进考虑范围内：

> 精装体有同心的纤维细胞层组成
>
> 正解：晶状体由同心的纤维细胞层组成

这是涉及到专有名词的场景，暂时不考虑扩充这样的语料库。

> 此成为寄存亡只求也
>
> 正解：此诚危急存亡之秋也

这是涉及到古文的场景，暂时不考虑扩充这样的语料库。

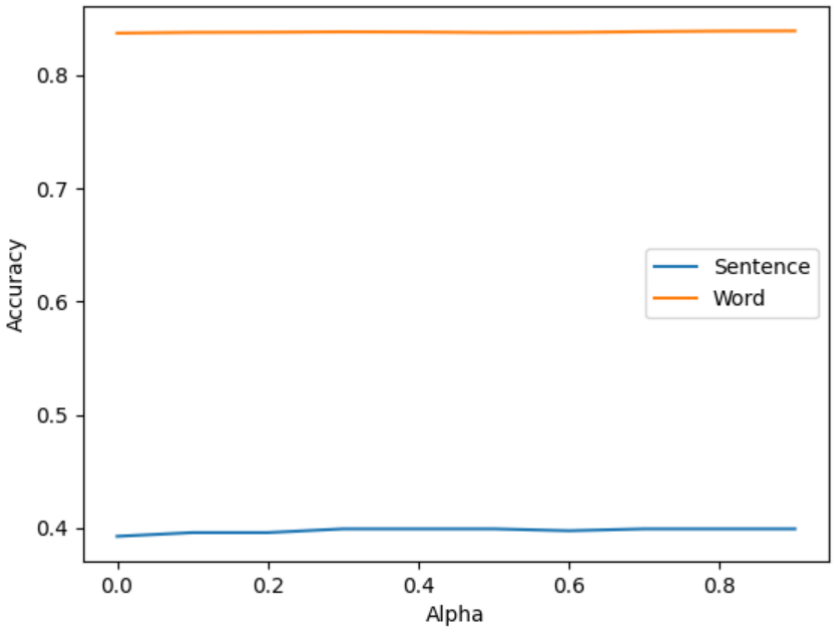
综上所述，我们得到了下一步的可能改进方向：

- 改进语料库。
- 尝试多元的模型。

参数调优及性能分析

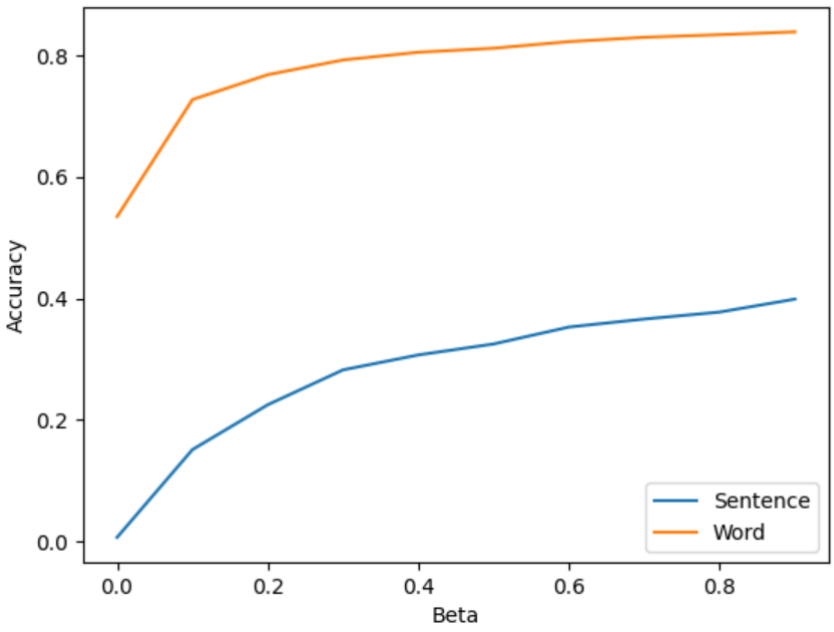
由于α和β这两个参数较为独立，因此我们单独对其调整优化。

首先固定$\beta = 0.9$，调整α

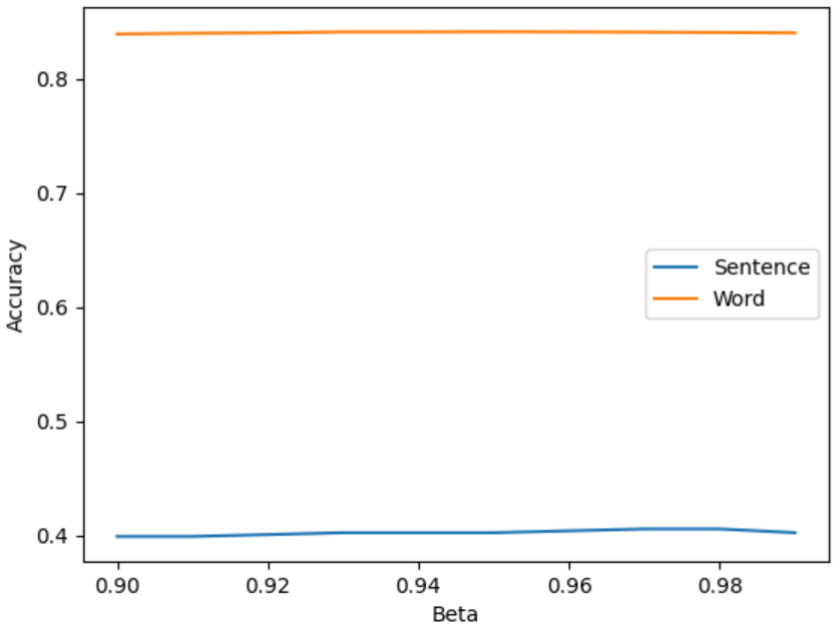


由图可知模型对α不敏感，我们之后不再调节这个参数，并且就取$\alpha = 0.9$。

首先固定$\alpha = 0.9$，粗调整β



从中可以看出准确率基本随β单调递增，下面进行细调整



从中可以看出到0.9之后就变化不大了。

综合上述结果最终选择$\alpha = 0.9, \beta = 0.97$作为最终参数。

在这个参数下测试集上的表现为：

0.8410(字)/**0.4056**(句)

经过多次重复测试，在测试集上的平均运行时间为：

36.30s

注意力模型

通过简单的推理不难发现，人在想一句话的时候，并不是一个一个字去想的，而是先形成一个大致的意思，然后才形成一句话，甚至需要反复迭代修改。

例如，我们说

> 他是一个英俊的男孩

这里我们用了`英俊`这个词，而这个可能跟前文没有任何关系(`他是一个`并不能决定什么，他可以是任何东西，并不一定是英俊的男孩)，相反`英俊`可能和`男孩`有莫大的关系。我们可能原本想说`他是一个男孩`，但是觉得应该加入什么修饰，所以才加上了`英俊`。

因此，每个字单纯决定于前文或许是不完整的，应该加入“当后文出现时，前文应该出现的概率”这一分量。不过可以预见的是，这个分量的权重应该比较小，因为绝大多数情况前文是更重要的。

因此，我们可以从考虑前文变成考虑**上下文**，修改转移概率和初始概率：

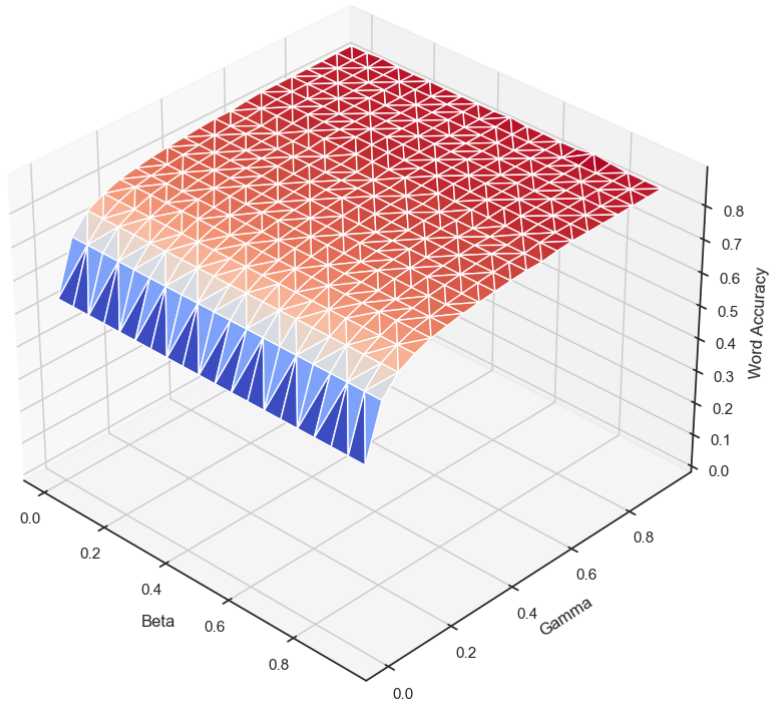
$$A^*_{x,k,t} = (1-\beta-\gamma)P(s_k) + \beta \cdot P(s_k|s_x) + \gamma \cdot P(s_k|y_{t+1})$$

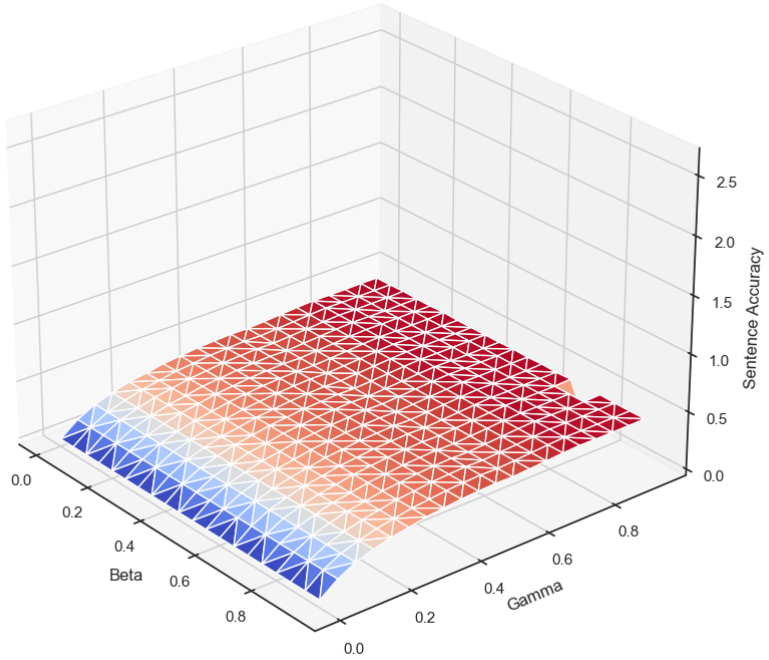
$$\pi^*_k = (1-\alpha-\alpha_2)P(s_k) + \alpha \cdot \pi_k + \alpha_2 \cdot P(s_k|y_2)$$

如序列长度为1，则$P(s_k|y_2)$取0.

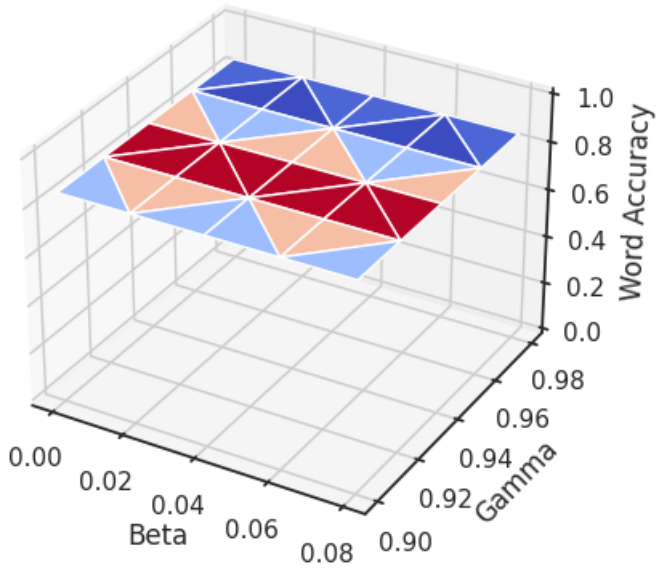
加入注意力模型后的参数调优及性能分析结果如下：

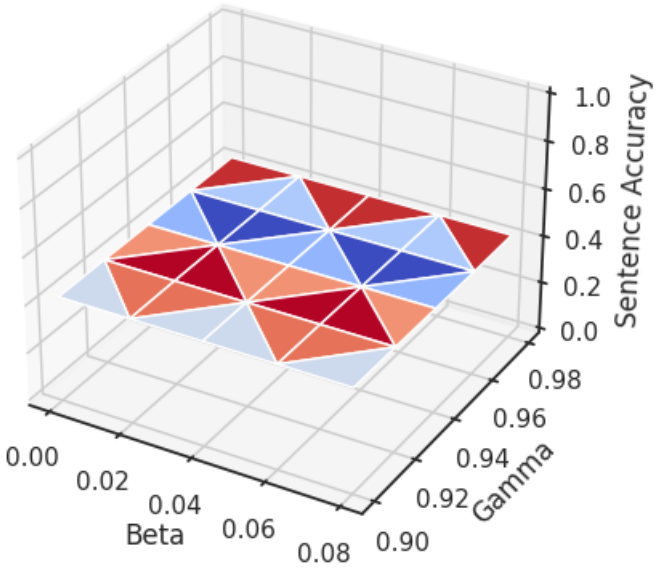
由于β, γ写在同一个式子中，因此不能单独调参，我们对其联合调参(固定$\alpha = 0.1, \alpha_2 = 0.8$)。首先进行粗调，结果如下：





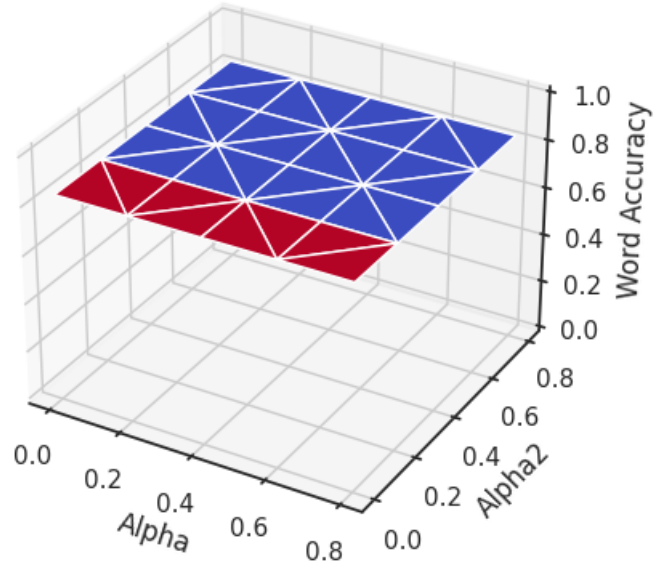
可以看出在γ超过0.9之后的正确率比较高，接着进行细调：

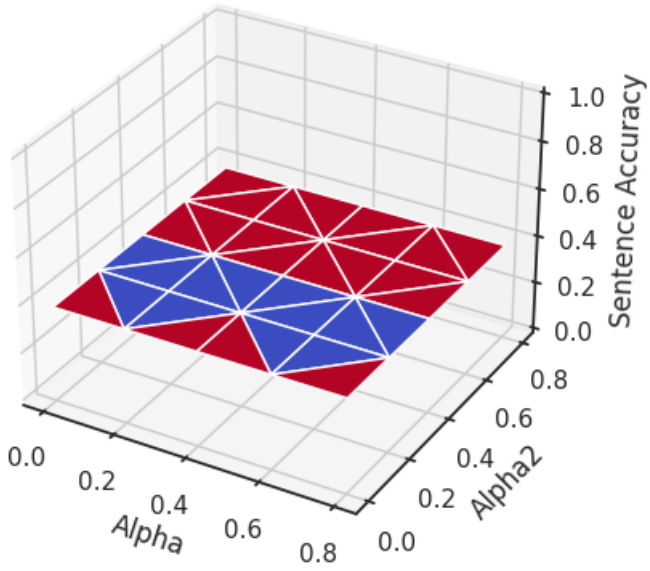




综合上述结果选择$\gamma = 0.93, \beta = 0.03$作为参数。

固定$\gamma = 0.93, \beta = 0.03$，下面进行$\alpha, \alpha_2$的调参，首先进行粗调：





发现α, α_2对准确率的影响不大。

最终选择$\gamma = 0.93, \beta = 0.03, \alpha = 0.5, \alpha_2 = 0.1$作为最终参数。

在这个参数下测试集上的表现为：

0.8336(字)/**0.3760**(句)

经过多次重复测试，在测试集上的平均运行时间为：

35.04s

相比二元模型而言准确率有所下降，说明注意力机制可能需要更加恰当的方式融入二元模型中。

语料库扩增

从上述实验结果分析中可以看出，语料库是影响输入法准确率非常重要的因素。经过仔细比对测试集上输入法的输出结果及标准输出结果发现：有差不多一半的错句是通过扩增语料库可以避免的(换句话说改变语料库可以带来准确率上的巨大提升，这一点会在下文介绍)。

此外，随着时间的变化，网路流行语也在不断地演进，过去的语料并不一定适用于现在，而且新闻的语料比较正式官方，与日常生活使用的语言相去甚远。为此，我从网络上获取大量语料结合网上的nlp训练集筛选整理清洗出了本程序的语料库。

网上的nlp语料库选自git仓库(https://github.com/brightmart/nlp_chinese_corpus)，内容包括：

- 2019年的维基百科词条，约100万个词条。
- 2019年的社区问答语料。
- 2016年的新闻语料。
- 2018年的百科问答。

我从上述语料库中选取部分作为我的语料库的底料。

然而，这个语料库还是有点偏旧，故另外爬取语料库作为补充：

- 2019-2020年的新浪新闻
- 2020年知乎问答
- 2020年豆瓣数据

- 2020年贴吧数据
- 2020年的微博语料
- 2020年部分中文维基资料

我从以上语料库中筛选整理出近20个G的句子库，从中训练概率模型，写入数据库作为算法的依据。由于原句子库太大，故没有包含在根目录中。

在扩增语料库之后，**重新测试**二元模型(保持参数选择不变)，得到的结果如下：

准确率：**0.8702**(字)/**0.4795**(句)

平均运行时间：**45.12s**

平均运行时间有所增加，可能是新的语料库中含有更多的二元组(二元模型中的转移对)，因此消耗更多的时间查询。

接下来的所有模型都是基于**扩增后的语料库**展开的。

二元+三元模型

从二元模型的实验结果可以看出，不少错误都是由二元模型的局限性导致的。如果将三元模型结合进来，则有可能大幅提升准确率。

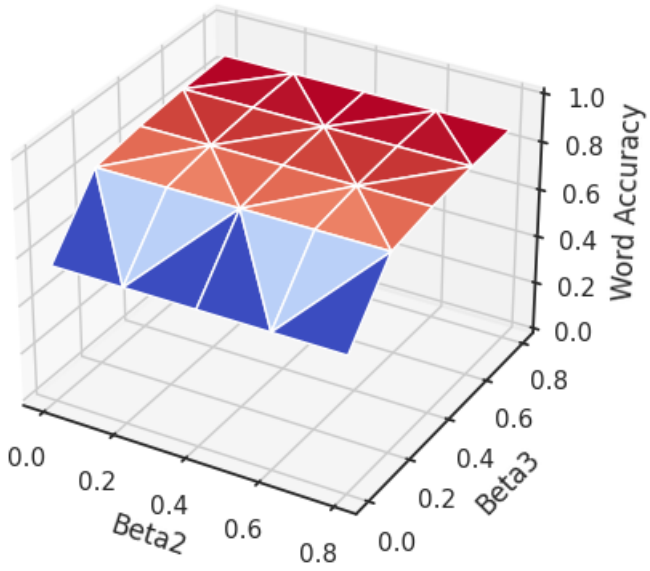
具体来说，就是将转移概率重新定义如下：

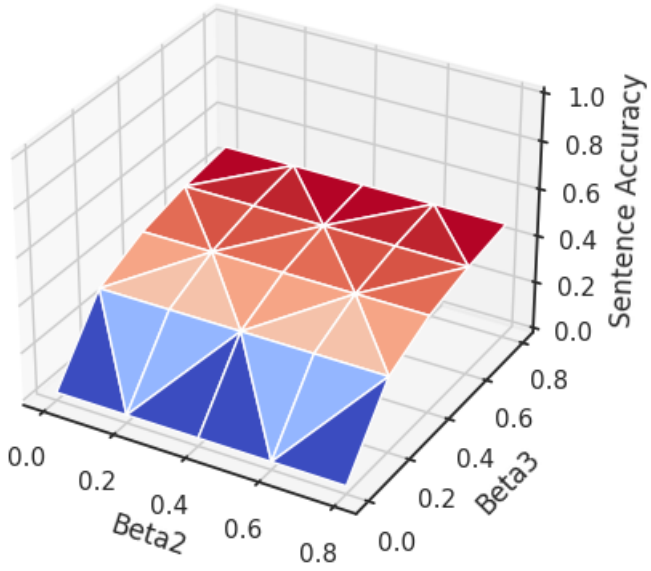
$$A^*_{x,k} = (1-\beta_2-\beta_3)P(s_k) + \beta_2 \cdot P(s_k|s_x) + \beta_3 \cdot P(s_k|s_{x'}\,s_x)$$

其中$s_{x'}$为s_x在其所在路径中的上一个汉字。

而$P(s_k|s_{x'}\,s_x)$则主要通过统计每一个字与其前两个字构成的pair出现的概率获得。

加入三元模型后的参数调优及性能分析结果如下(保持二元模型的$\alpha = 0.9$不变)：





综合上述结果最终选择$\beta_3 = 0.9, \beta_2 = 0.09$作为最终参数。

在这个参数下测试集上的表现为：

0.9402(字)/**0.7274**(句)

经过多次重复测试，在测试集上的平均运行时间为：

186.17s

二元+三元修正模型

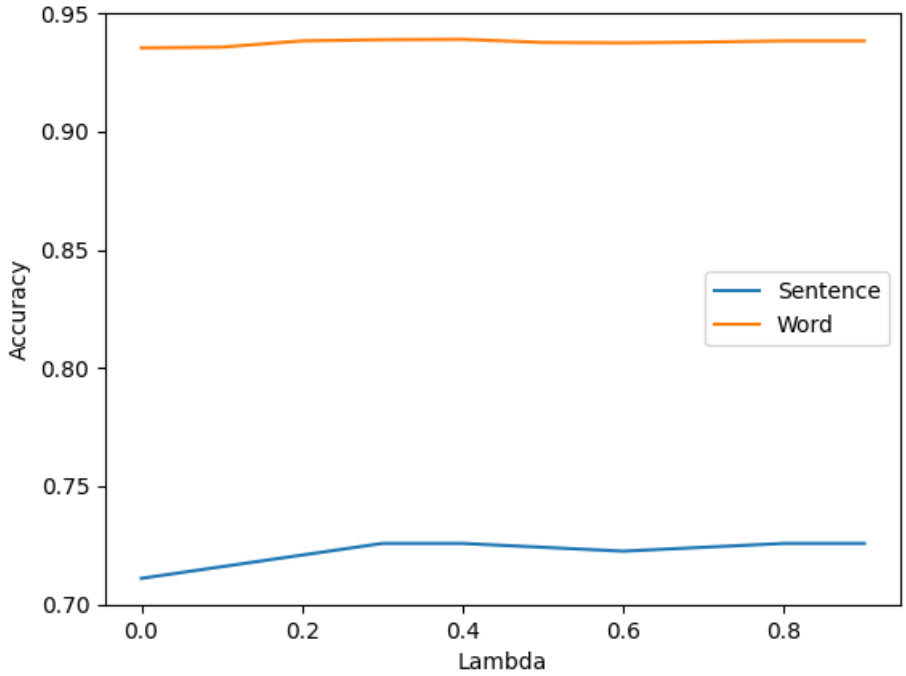
二元+三元模型已经取得了不错的效果，但是看它的转移公式：

$$V_{t,k} = \max_{1 \le x \le K} \left(E_{k,t} \cdot A^*_{x,k} \cdot V_{t-1,x} \right)$$

我们发现发射概率和转移概率是1:1相乘的，这可能缺了一个参数。据此我们可以加入新的参数λ，修改转移公式如下：

$$V_{t,k} = \max_{1 \le x \le K} \left(E^{\lambda}_{k,t} \cdot A^*_{x,k} \cdot V_{t-1,x} \right)$$

对λ进行参数调优结果如下：



最终选择$\lambda = 0.3$作为参数。

在这个参数下测试集上的表现为：

0.9404(字)/**0.7258**(句)

经过多次重复测试，在测试集上的平均运行时间为：

185.89s

模型性能比较

将上述各个模型(**二元模型**，**三元+二元模型**，**三元+二元修正模型**，**注意力模型**)经过调参后的最优结果总结为下表：

模型	字准确率	句准确率
二元模型	**0.8702**(字)	**0.4795**(句)
三元+二元模型	**0.9402**(字)	**0.7274**(句)
三元+二元修正模型	**0.9404**(字)	**0.7258**(句)
注意力模型	**0.8336**(字)	**0.3760**(句)

从上表可以看出在**当前测试集**上最优的模型：

若按字准确率，则是：

三元+二元修正模型

若按句准确率，则是

三元+二元模型

改进方向

- 尝试更多元的模型。从实验结果看出，从二元模型到三元模型有了质的飞跃。或许更多元模型还能进一步拉高正确率。此外，由于本程序采用数据库，即使模型更多，也不用担心内存不足的情况。
- 扩充专业语料库，并且让用户选择领域。通过语料库扩增部分我们可以发现一个真理——“隔行如隔山”。不同的语料库可以给输入法带来完全不同的增益。然而再厉害语料库也不能覆盖所有领域，因此有必要针对不同的领域单独训练语料库，让用户自己选择他需要的领域。
- 加入句子语法分析。我们发现有些错单纯通过概率很难解决。这时就需要基于语法的纠错。比如“他是我的母亲”在语法上就是错的，“母亲”是女性，“他”是男性，这在性别上是不符的。因此我们需要使用nlp训练基于词向量的模型，从语义语法的角度分析句子的合理性。python的`spacy`库或许是一个可以尝试的选择。
- 注意力模型可以进一步扩展。目前注意力模型并没有正面增益。为了可以将其与三元模型结合，可以考虑在生成整个句子之后再用注意力模型进行纠错。
- 加速处理。目前的模型的运算速度还比较慢，未来可能会在速度上对输入法进行优化，可能的方案：
    - 优化数据库查询，设置更多级缓存，避免频繁地访问本地
    - 改用c++实现，将代码迁移到c++上换取更高的速度

实验收获

通过这次实验，我实践了人工智能课上学到一些方法，增加了对知识的理解。此外我通过自己的实践，包括**数据爬取**，**数据清洗**，**模型设计**，**参数调优**，**模型测试评估**，**模型改进优化**等，体会了训练出一个好的人工智能模型是多么的不容易。这些工作让我的实践能力，理论能力和创新能力都有了显著的提高。

参考文献

1. https://zh.wikipedia.org/wiki/%E7%BB%B4%E7%89%B9%E6%AF%94%E7%AE%97%E6%B3%95
2. https://github.com/brightmart/nlp_chinese_corpus