根据白话文生成古诗

杨永祎
17300240038

1 概述

目前已经有许多生成古诗的系统，也有一些系统可以允许用户自己对生成的诗的内容进行定制，但是大多数系统都只能允许较弱的人为定制。比如清华大学的九歌系统[1]，就只能给定关键字进行生成，也可以输入段落等，但无法完全本着给定段落来生成，只能把输入段落作为氛围、基调等的提示[2]。而一个有趣的话题是，能否让用户输入一个白话文段落，让机器生成一首能尽可能地保留这个段落的内容的古诗（等于说是把这个段落『翻译』成古诗）？

从白话文生成古诗，这个主题目前好像还没有太多人研究。我曾看到过一篇用无监督机器翻译系统来做这个任务的文章（Z. Yang et al.，2019）[3]，这篇论文是今年发表的，并且他们声称自己是第一个提出此任务的，所以我想这个主题大概还没有太多可以直接参考的前人成果。

这个任务很少人做是可以理解的：最主要的困难是，几乎找不到白话文-古诗的平行语料，这就导致很难直接将现有的深度学习方法应用于这个任务，而且古诗语料本来就比较少（相对机器翻译等而言），即使做无监督学习效果也不一定好。另外，目前也很难说有什么适合的方法可以对结果进行评价，机器翻译通常使用 BLEU 值评价，对于日常语言而言，这还勉强可行，但是古诗的形式和表达非常多变，而且强调『言外之意』，因此我认为用基于 n-gram 匹配的 BLEU 值评价并不合理。

当然，本文只是一个比较浅的尝试，而且我也不是为了与其他人比拼结果，所以我姑且不考虑评价上的困难。而对于缺少平行语料的困难，不同于 Z. Yang et al.，我这里尝试利用现成的中英翻译系统来生成（比较弱的）平行语料作为监督文本。这也是我主要想探究的内容：能否在完全不使用平行语料的情况下，使用另一门语言作为中间语言来生成『伪平行语料』并将这种语料作为训练数据？

如果能得到这样的伪平行语料，就可以把这个任务看成是一个经典的 seq2seq 任务，可以直接使用经典的机器翻译模型进行有监督的学习。

---

[1] https://jiuge.thunlp.cn/

[2] 比方说，输入『荷塘的四面，远远近近，高高低低都是树，而杨柳最多。这些树将一片荷塘重重围住，只在小路一旁，漏着几段空隙，像是特为月光留下的。』，得到的输出是『古寺多幽寂，萧然四壁围。松风时有响，云雨夜来稀。』，生成内容和原文其实没有什么关系，只能说是把原文当做某种提示。

[3] Z. Yang et al.; Generating Classical Chinese Poems from Vernacular Chinese

2　数据获取

之前已经提到过，白话文-古诗的平行语料是非常缺少的。我这里采用一种比较妥协的方法：首先将古诗（作为中文输入）翻译成英文，然后将英文翻译回汉语。因为翻译系统大多是在现代汉语文本上训练的，所以可以预期其会翻译出更符合现代汉语的习惯的句子，也就是可以把这个翻译的结果作为这首诗的白话文版本。当然，可以预期的是，这种翻译的质量并不会太好。

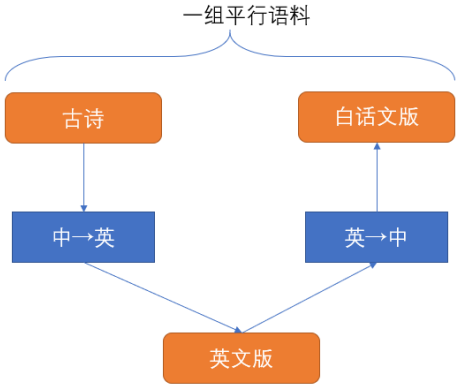

图 1 数据生成过程

顺便说一下，百度翻译其实有一个文言文翻译白话的功能，用那个的话效果应该会好一些，但是我做这个项目的目的之一就是探究使用现成的中英翻译系统将古诗转化为白话，并将其作为训练语料的可行性。因此我希望尽量做到完全不使用平行语料，而百度的文言文转白话的系统中可能用了一些古诗-白话文的平行语料，所以我选择不使用这个功能，只使用中英文翻译的功能。

目前比较常用的翻译系统中，就我所知，Google 翻译和百度翻译是免费开放了 API 的。这两个翻译系统都是著名的高质量的翻译系统，但是很可惜的是，从 API 生成的结果和从网页端生成的结果并不一样，事实上是差很多（当然，这是可以理解的，否则开发者完全可以比如调用 Google 翻译的 API 自己做一个翻译网站，那就把 Google 的知名度分走了）。因此，我事先对这两个系统的古诗-白话翻译性能做一个简单的人工评价。

这里为了展示他们的效果，我展示两首唐诗调用翻译 API 翻译英语后再翻回汉语的结果：

翻译系统 / 古诗	百度翻译	谷歌翻译
床前看月光，疑是地上霜。 举头望山月，低头思故乡。	看着床前的月光，我怀疑是地上的霜。仰望月亮和群山，我俯视我的家乡。	床上看月亮，疑是地上霜。 举头望山月，低头思故乡。
梅发柳依依，黄鹂历乱飞。 当歌怜景色，对酒惜芳菲。 曲水浮花气，流风散舞衣。 通宵留暮雨，上客莫言归。	梅毛刘毅，黄鹂乱飞。当歌怜惜风景，珍爱酒香。水漂浮在空气中，风吹动舞衣。下了一整夜的雨，我回不了家。	梅柳依依头发，黄雀飞行经验。当这首歌可惜风光，酒和糖果可惜。曲水浮花气流风散的服装。夜宿木鱼，登莫言的回报。

可以看出，虽然都有很多出错的地方，百度翻译的质量明显高于 Google，这可能是由于，Google 的 API 是不限制使用的，所以为了防止之前我提到的滥用的情况，他把 API 的能力降低了很多。而百度翻译 API 则采用了一种更折中的方案：在不付费的情况下，每秒只能调用一次，这样就限制了开发者利用翻译 API 分走百度翻译的用户的做法。因此他可以放心的对 API 开放更强大的翻译系统。

所以，最终我采用百度翻译来生成数据。因为他限制了翻译速度，翻译特别慢，我做 PJ 的时间有限，所以我也无法容忍过大的数据量。最终我采用全唐诗[4]作为古诗数据集，用正则表达式删去了每首诗的作者出处注释等信息，只保留诗歌内容（但事实上还是有一些残留的额外内容，但数量不大）。另外因为唐诗中有很多长诗，为了防止数据过长导致训练时内存崩溃等问题，我把每首诗都截取到前 96 个字，对于五言诗而言，这是 16 句话，对于七言诗而言，这是 12 句话，这样保证对五言和七言诗而言诗句都是相对完整的。然后调用百度翻译 API 依次将每首诗翻译成英文再翻回中文并记录下来，生成一个新的数据集，我称之为 Verna_Tangshi 数据集。

每首诗翻译需要两秒种，全唐诗一共 5 万多首，生成这个数据集一共花了一天多的时间，算是可接受的。

3　模型

之前提到过，有了平行语料之后这个任务就可以视为一个经典的 seq2seq 的任务，而机器翻译是 seq2seq 任务中的典型，因此我考虑使用在机器翻译中表现比较好的模型。这里使用 2017 年 Vaswani et al.提出的 Transformer[5]。

3．1 Transformer 介绍

Transformer 的大致框架如下：

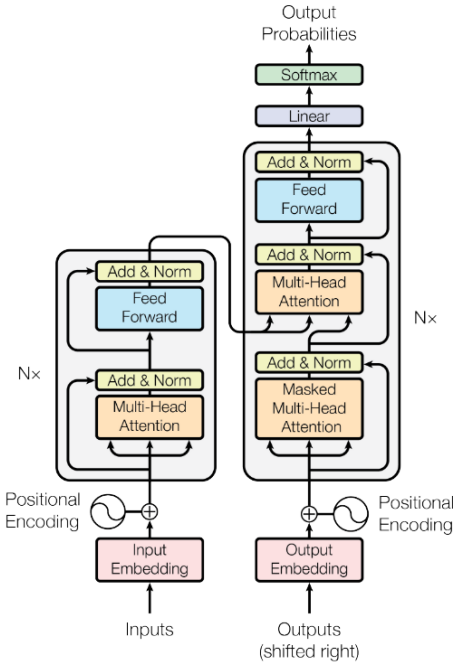

图 2 Transformer 结构

[4] https://github.com/chinese-poetry/chinese-poetry

[5] Vaswani et al.; Attention Is All You Need

左侧是编码器（Encoder），用于把长度为 n 的源语言（白话）句子，编码成 n 个向量，理论上，第 i 个输出向量包含了第 i 个词的信息，并且包含了整个句子的部分信息，右侧是解码器，根据编码器生成的向量以及一串输入字符，生成目标语言（古诗）每个词对应的向量表示，而目标语言每个词的向量表示被用于预测其下一个词（因此，Decoder 中信息不能从右侧向左侧流动，只能从左侧向右侧流动，在 Self-Attention 中使用了一个下三角形的 mask 矩阵来保证这一点）。

编码器以及解码器都是由相同的许多层构成。编码器中每一层包含一个 Self-Attention 层，用来建立任意两个单词之间的信息交流，之后是一个前馈层（Feed-Forward），用来对每个单词的向量进行一定的变换。解码层的结构类似，只不过加了一个对 Encoder 的输出的 Attention，用于建立任意从源语言单词到目标语言单词的信息交流。

相比于传统的 LSTM，这个模型虽然复杂度更高（LSTM 的时间复杂度是$O(d^2n)$，而 Transformer 是$O(d^2n + dn^2)$），但是其能充分利用 GPU 的并行性。因为 LSTM 的每个单词的输出都依赖前面所有单词的输出，因此处理一个单词之前不得不先处理前边的所有单词，而这个模型的任意两个单词的交互都是同时发生的，因此可以充分并行化。因此面对大规模数据，尤其是短数据，Transformer 通常比 LSTM 快很多。

为了方便对模型和超参数等进行各种细微的调整（当然，后来事实证明我其实并没有太多时间调参 T_T），我在这个项目中自己实现了 Transformer。

3．2　生成：束搜索

训练好了一个 seq2seq 模型后，如何从输入生成一个完整的句子并不是显然的，因为 seq2seq 模型被训练于从一个完整的源语言串和一个目标语言前缀，生成目标语言的下一个字的分布。但是实际使用的时候，目标语言要从空开始生成，如果我们每一步都取概率最大的字，最后的句子不一定是概率最大的那个。因此需要一种算法来生成一个联合概率最大或至少是较大的句子。

束搜索是一种常用的生成算法，其结合了广度优先搜索和贪心两种策略，在几乎不损失能力的情况下保证了较快的速度（Transformer 的生成是$O(n^3)$的，因此速度特别重要）。简单来说，因为每个位置的单词分布只与其前缀有关，因此可以用广度优先搜索来以一定顺序遍历所有可能的答案：第一步枚举所有长度为 1 的前缀，第二步从每个长度为 1 的前缀生成所有长度为 2 的前缀，以此类推。但是可能的句子数太多了，这个算法的时间复杂度是不能容忍的。因此考虑使用贪心算法剪枝：在 BFS 的每一步中都只保留概率前k大的前缀。假设单词表大小为V，句子长度为n，这个算法把时间复杂度从 BFS 的$O(V^n)$降低到了$O(k^2n)$。

3．3 超参数和其他

我这里用了 4 层，512 维的 Transformer。使用 Adam 优化器进行优化，学习率取 1e-4，batch size 取 128，训练 10 个 epoch。按照原论文的建议使用了 warm-up 机制，但是因为数据量比较小，warm-up steps 取的 400。

3．4 BERT vs Static

一个单词是一个符号序列，要让其可以被神经网络处理，需要进行一些操作，使之转化为向量序列，这个步骤称为词嵌入。通常，模型要么从随机向量开始训练一个词嵌入方法，要么直接利用在其他大规模文本上训练过的词嵌入方法（称为预训练）。后一种方法的好处是，对于规模较小的文本，通常不足以训练出优秀的词向量。而且在刚开始训练时，如果词向量是随机的，则模型接受的实际上是随机输入，则其有可能向错误的方向优化并且难以复

原，最后导致无法收敛至较优的解之类的训练上的困难[6]。

而对于预训练的词嵌入，现在通常也有两种方法，一种是静态的词嵌入，即对每个单词生成一个词向量，使用时直接查表得出词向量。这种方式的好处是快速、简便、资源占用较小，但是不足之处是，每个单词根据其上下文不同，语义也可能不同，但我们对同一个单词在不同的上下文中分配的词向量都是一样的，这样就可能有歧义之类的问题。

另一种使用预训练词向量的方法是以 BERT 为代表的，上下文有关的向量。BERT 得出的不是一个单词-词向量表，而是一个模型及其参数，实际上就是一个编码器，和 3.1 节中提到的类似，会对每个单词输出一个向量表示，而这个向量表示是经过这个单词和其上下文交互之后的表示，因此可以预期的是，这种方法得到的词向量会带有消歧信息，并且也会包含各种句法信息，因此通常比静态的词嵌入方法更强。当然这种方法相对于静态方法的坏处是比较慢，而且资源占用比较大（因为需要先经过一个大型的模型来编码）。

当然，实际上在我的模型中使用 BERT 是有一定困难的，之前说过，解码器必须保证信息不能从后向前流动，因为解码器的任务是对于每个词，预测其后一个词，而如果这个词在训练中本身就可以得知后一个词的信息，那么训练就无效了。但是 BERT 中，信息就是双向流动的，因此 BERT 无法用于解码器。所以在我的模型中如果要使用 BERT，就只能在编码器中使用，在解码器中还是只能使用静态的词向量。但是这样实际上会造成词嵌入的割裂——输入和输出端都是中文，但是他们却不共用词向量编码，这有可能破坏模型效果。

在我的模型中。我对以上提到的三种词嵌入方法都进行了实验，其中静态方法是使用的 fastNLP 提供的在中文文学作品上训练的中文词向量[7]，而 BERT 方式也是 fastNLP 提供的中文 BERT 模型[8]。

3．5 译后编辑（post-editing）

在一些预实验中我发现，或许是因为数据集过小，模型生成的诗有两个明显的问题：一个是常常不能做到押韵，另一个是有些字常常会重复出现。这使得生成的诗读起来很怪异，为了解决这两个问题，我在生成时加入两个机制，一个是对先前出现过的字，把它再次出现的概率调整为原来的 1/10，另一个是对于句末的字，只保留押韵的候选字（为了不让候选字太少，我用了一种比较宽松的押韵规则，把 in、en、un 等都视为互相押韵的，对于 ng 结尾的韵母也类似）。

4 实验结果

我从数据中随机挑选出 1000 个白话-古诗对，作为验证集。通过观察模型在验证集上的交叉熵损失函数的平均值来评判模型好坏。我想这种方法是能大概能反映出模型的优劣的。

三种不同的词嵌入方式的效果和速度如下：

方式 \ 指标	验证集损失	训练用时(s)
Random	3.731	993.133
Static	3.465	990.689
Encoder Bert Decoder Static	3.633	1729.272

表中 Random，Static，Encoder Bert Decoder Static 分别代表之前提到的随机初始化词向量、静态的预训练词向量、输端用 BERT 输出端用静态词向量的方法。

可以看到，静态词向量初始化的词向量不仅速度快，而且效果最好。

---

[6] Xavier et al.; Understanding the difficulty of training deep feedforward neural networks

[7] https://github.com/fastnlp/fastNLP，然后参数实际上来源于 Shen Li et al.; Analogical Reasoning on Chinese Morphological and Semantic Relations

[8]参数实际上来源于 Y. Cui et al.; Pre-Training with Whole Word Masking for Chinese BERT

5 评价

这个任务无法进行通常的定量评价，因为我其实没有金数据，也不知道应该用什么评价标准。我这里姑且就取效果最好的模型（使用静态预训练词向量的模型），用几首现代诗文片段作为输入，看看其生成的效果如何吧：

输入	输出
荷塘的四面，远远近近，高高低低都是树，而杨柳最多。这些树将一片荷塘重重围住，只在小路一旁，漏着几段空隙，像是特为月光留下的。[9]	荷塘四面远，高低柳色多。 此树重莲户，唯闻晓漏过。
我辨认了星月的光明，草的青，花的香，流水的殷勤。我能忘记那初春的睥睨呢。曾经有多少个清晨我独自冒着冷去薄霜铺地的林子里闲步，为听鸟语，为盼朝阳，为寻泥土里渐次苏醒的花草，为体会最微细最神妙的春信。[10]	我见星辰月草绿，花香流水泛清流。 忘忧始忆春初起，曾是经年少小游。 独步寒林惊鸟语，自闻残日梦魂休。
寻梦，撑一支长蒿，向青草更青处漫溯，满载一船星辉，在星辉斑斓里放歌。但我不能放歌，悄悄是别离的笙箫，夏虫也为我沉默，沉默是今晚的康桥。[11]	寻梦长汀洲草绿，更通星斗满楼船。 却携歌响未能歇，不得歌声是别弦。
东风不来，三月的柳絮不飞，你的心如小小的寂寞的城，恰若青石的街道向晚。跫音不响，三月的春帷不揭，你的心是小小的窗扉紧掩。[12]	东风不来柳絮飞，无心寂寞小城头。 晚钟声咽三湘曲，春色未得晓窗愁。
拂袖起舞于梦中徘徊，相思蔓上心扉。她眷恋梨花泪，静画红妆等谁归，空留伊人徐徐憔悴。[13]	舞袖鸳鸯梦，相思一片心。 惆怅梨花静，空留客愁人。
青山隐隐约约绿水千里迢迢，秋时已尽江南草木还未枯凋。二十四桥明月映照幽幽清夜，你这美人现在何处教人吹箫？[14]	青山临水碧千秋，已尽江南草木凋。 十二桥边明月照，美人今夜教吹箫。

[9] 朱自清《荷塘月色》片段
[10] 徐志摩《我所知道的康桥》片段
[11] 徐志摩《再别康桥》片段
[12] 郑愁予《错误》片段
[13] 《卷珠帘》歌词片段
[14] 杜牧《寄扬州韩绰判官》的白话翻译。原诗是『青山隐隐水迢迢，秋尽江南草未凋。二十四桥明月夜，玉人何处教吹箫。』

可以看出，大部分翻译勉强算是能抓住原文的部分意象，并且对于原文的情感等比较明显常见的白话文，这个模型也能把情感大致重复出来。比如『寻梦...』这首，就截取的这一段来说感情其实比较微妙，原文的情绪先是有些激昂的，之后转为离别的淡淡的寂寞，翻译的古诗中大致也表现出了这种情感的转折，这是很好的。

但是对于有些使用的修辞在唐诗中比较少见的原文，比如像『你的心是小小的窗扉紧掩』，好像也很少在唐诗中看到这种比喻，这一句就没能准确地翻译。

不过令人欣喜的是，即使模型没有理解原文，它也能很大程度上把原诗的意象重复出来，比如『跫音不响，三月的春帷不揭，你的心是小小的窗扉紧掩。』这一段，被翻译成了『晚钟声咽三湘曲，春色未得晓窗愁。』，虽然完全没有保留原诗的意思，但是原诗中出现的意象，如声音（模型大概不认识『跫』这个字，所以以为就是声音）、春、窗等，都在诗中出现了，这说明模型至少能准确抓住原文的主要意象。而且原文中虽然是写愁的，却没有出现『愁』这个字，而模型则点出了这个字，说明模型不是在简单地复写原诗，而是在理解了诗意的基础上的改写。

5．1 虎头蛇尾问题

从这些翻译中可以观察到一个明显的现象，我称之为虎头蛇尾问题，就是译诗的开头通常比较合理，而越到后面，越不能保留原诗的意象和情感，机器越倾向于自己创造诗句，而且创造得不好。比如『寻梦...』这一首的翻译中，前两句『寻梦长汀洲草绿，更通星斗满楼船。』算是写得很好了，但到了第四句『不得歌声是别弦』就读来不是很通顺了。再比如像『东风不来...』这首的翻译中，第一句『东风不来柳絮飞』还算合理，但是到了后面第三句的『晚钟声咽三湘曲』就完全是模型自己创造的诗句了（虽然这句创造地还算合适，但是毕竟没有本着原诗）。

事实上，Z. Yang et al.的文章的系统也有类似的问题，他们发现了一种现象叫 Under Translation（欠翻译），而且从论文里给的例子来看，大多数欠翻译都发生在诗末（文中诗也提到了这一点）。所以总得来说，说译诗越到结尾，越难保留原诗特征还是合理的。

我想这可能和 seq2seq 模型的曝光偏差[15]有关，因为训练的时候是直接用金数据作为输入的，而翻译时却是用之前的结果作为输入，这种训练和生成的不一致导致模型越生成到后面，越容易出错。

5．2 字词改写问题

有些时候模型会莫名奇妙的加减一些不合适的字词，导致意思完全被改写。比如『漏着几段空隙，像是特为月光留下的。』中，译诗确实把『漏』字翻译了出来，却翻译成了『晓漏』（意思是拂晓时铜壶滴漏之声），就完全不合理了。

另外有些即使是他见过的词，也会被莫名其妙地改写，比如『二十四桥明月映照』被翻译成了『十二桥边明月照』，把『二十四桥』写错了。但是『二十四桥』这个词在训练集中是不止一次出现的，模型是见过这个词的，竟然也没有翻译对。

我想这种错乱可能是和我使用的基于字的模型有关，或许先分词再做会好一些。另外有些模型会加入允许解码器直接从源语言复制单词的机制，我想如果加入这个机制或许也会好一些。

这种错误也有可能是来源于监督文本太弱，因为古诗经过翻译软件改写成的现代文，有些时候就会丢失一些意象，这样生成的数据，在模型看来就是可以在诗中添加一些输入中没有的意象或者单词，这就导致了他在生成时也会有类似的行为。

---

[15] T. He et al.; Quantifying Exposure Bias for Neural Language Generation

6 总结

总得来说这个模型的效果还是出乎我的意料的。我本来想把标题取作『带白话文提示的古诗生成』，因为觉得效果不会太好，只能算是把白话文当做提示。没想到训练下来，他生成得还算有模有样。

因此可见，通过中英翻译系统来生成白话文翻译古诗的监督文本还是一种比较有效的方法（虽然确实比较弱），或许可以用于增强现有的系统之类的。不过诗和白话的差异还是太大，我想如果用同样的方法做比如文言文转写之类的，效果或许会更好（不过文言文的长度通常比诗要长很多，这也是一个挑战）。

这个模型的一个主要问题反而是语言模型不够好，像押韵做不好、重复出现字之类的问题，都是语言模型（也就是解码器）的问题，我认为这个问题还是来源于训练语料太少了，如果能使用更大的语料，我相信这些问题能缓解很多。

一个有趣的现象是，虽然这个模型的押韵做得不好，但是在平仄上做得意外的好，我并没有加入任何规则来限制平仄，但是模型生成的诗（即使是那种让人看不懂的诗）往往能做到只有一两处平仄错误甚至没有平仄错误，这暗示对于模型而言，学到平仄是比学到韵要简单的。而对我们人类来说正相反，我们押韵很容易，纠结平仄却很难，我想这是因为我们人对于韵有更强的直观，而平仄我们则常常察觉不到（事实上就诗歌的发展而言也确实是这样，从晋代开始才有人注意到平仄问题，而早在诗经中人们写诗就已经押韵了），但对于模型来说，平仄的规则比押韵更简单（因为平仄只有四种，韵母却有几十种），所以模型更容易学习到平仄。我想这也揭示了机器学习与人类学习的一个不同之处。

本文的实验代码和 Verna_Tangshi 数据集都可以在我的 github 页面 github.com/fftyyy/poem 上找到。我在 README 中写了我的名字，以证明这确实是我的 github 仓库。