**现代诗自动生成器**

**报告结构**

1，综述和选题动机

2，GRU算法应用

3，数据预处理，实现思路和部分代码解释

4，实验结果

5，结论

**1、综述和选题动机**

本次自然语言处理期末project我选择了一个文本生成的任务，也就是生成现代诗。

我选择这个任务的原因其实有好一些。首先是诗歌生成这个任务的背景，有很多之前的项目都做了唐诗宋词元曲的一些生成，这些诗歌都比较有格式化的规律可以描写，而且古诗中每一个字都是意蕴丰富的，甚至一个字可以有多种属性，比如又可以是动词又可以是名词，甚至还可以是形容词，只要可以解释得通。但是现代诗相比古诗，一方面在格式上具有了更多的自由度，另一方面却对语法的完整性要求更高。如何清楚地描述和产生一句话，是现代白话文本的一大难题，而我选择了rnn里面的gru结构来应对这个问题。总的来说gru在这方面表现还不错，而且比较容易训练。

当然另外一个出于私心的原因是为了给女朋友写诗，固然可以自己动手写，甚至写得不错，但是很好奇人工智能写现代诗会是什么样的结果，会很不通顺呢还是别有一番风味，也是我很想知道的。

整个代码的完全实现我没有一字一句地自己实现，而是基于前人对于古诗生成的开源代码基础上加以修改、完善和训练。确实现代诗和古诗的生成是有好些区别的，跳脱格式韵律的束缚，又保证不会语无伦次。

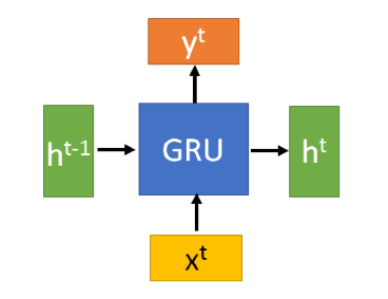
总体来说，我的代码目标是实现两个主要功能。第一个功能是通过给出几个简短的头部字段来生成一首诗，也就是指定开头的词向量。第二个功能是写一首藏头现代诗，也就是每一句话的开头词向量由给好的藏头的句子指定。实际的训练和测试之后，基本能达到要求的是第一个，也就是生成一定主题的文本，而第二个藏头诗的分句则有待改进。采用的gru算法能够基本满足要求，当然有些比较生僻的字词没有出现在语料库的话就比较难生成，但是我们所常见的一些自此，如“你”，“我”，“在”，“爱”，这些诗人们高频出现的词语，还是处理得比较好的。

**2、GRU算法应用**

实际上整个模型的实现就是依赖于2014年提出来的GRU的框架，所以作为该项目整体实现的核心的神经网络技术，在此单独进行阐述。

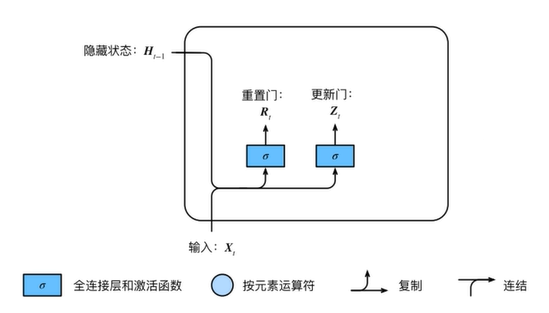
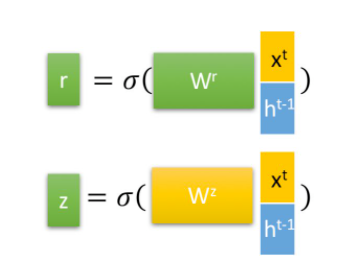
GRU全称是gate recurrent unit，门控循环神经网络。他是rnn的一种，基本上和rnn类似，可以让连续的文本生成有着一个被称之为长期记忆性的东西。它的实现也就是每一层都有一个新的输入参数，与此同时还会有一个隐藏层hidden layer，上一层隐藏层的ht-1会影响到当前的输出和下一层隐藏层ht. 正是因为隐藏层gru有了长期记忆性。

总体来说就和下图一致，每层的输入是xt和ht-1，然后gru会生成下一层所需要的ht和当前层的输出yt. hidden layer的延续性保证了记忆。



为了解决标准RNN的梯度消失问题，GRU使用了所谓的更新门和重置门。基本上，这两个向量决定了应该将什么信息传递给输出。他们的特别之处在于，他们可以被训练来保存很久以前的信息，而不需要经过时间的洗涤或删除与预测无关的信息。

那么gru的内部结构主要为两个门控辅助提供。如下两个图。r为reset重置门，而z为update更新门。输入的xt和上一个隐藏层ht-1经过这两个门得到两个中间输出。每个门相当于是一组参数wxt+wht-1+bias. 然后再放到我们熟悉的sigmod函数里面。

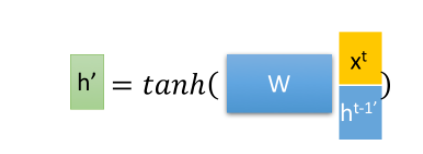


下一步，我们已经通过两个门得到了门控信号r和z。

首先用ht-1 x r = ht-1’，这里的x号指的是矩阵乘法。然后类似之前计算r和z的方法如下图，一组参数w和xt还有ht-1’通过线性组合得到一个中间结果，再放入tanh激活函数中，把数据压缩到-1到1的范围里面，结果就是被称为候选隐藏状态的h’.

那我们看这里的ht-1’是由r和上一层ht-1共同得到的，假设r为0，那么就相当于完全丢弃掉上一个隐藏层拿来的结果，仅仅使用当前的输入xt。所以r相当于是控制之前隐藏层对当前层的影响，可以丢弃掉一部分对未来的预测没有帮助的历史信息。

这个阶段也被称为选择性记忆阶段。

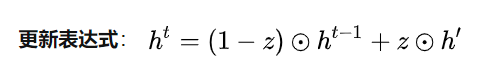


下面是使用更新门z来组合过去的隐藏状态。也被称为更新记忆的阶段。

更新表达式如下图所示。

可以看到，更新表达式里面包含了上一隐藏层ht-1和当前隐藏候选状态h’的信息。取极端情况，假设z=0，前一项完全保留，ht=ht-1，完全抛弃当前层的输入对于下一个隐藏层的影响，之前得来的长期记忆被继承下去。假设z=1，前一项为0，后一项完全保留，相当于抛弃之前隐藏层，只对当前的输入进行记忆。

通过一个z和1-z的组合其实是达到了选择和遗忘的效果，z越大对当前状态的选择性越高，越小则对以往状态的记忆性更好。

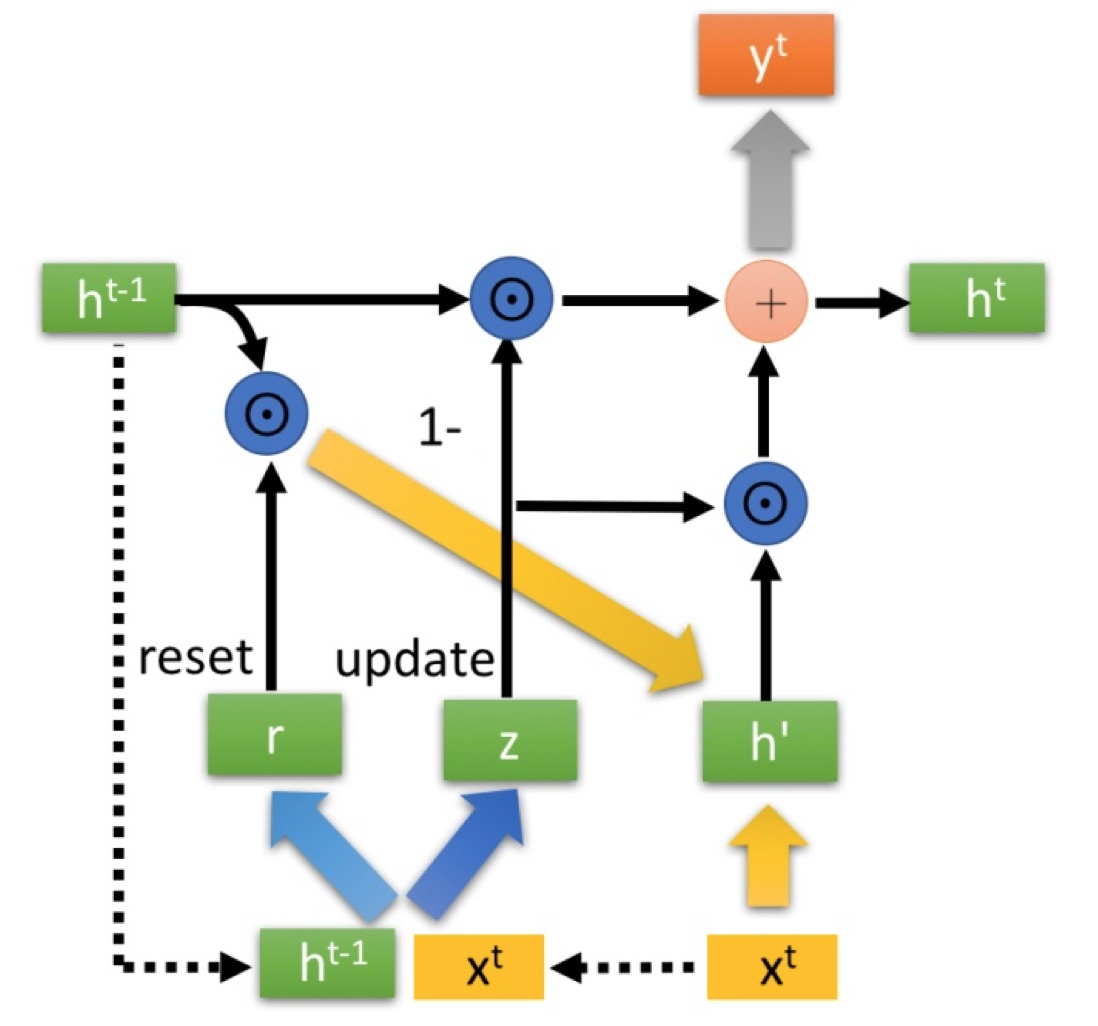


总体来说，r更新门是捕捉当前序列数据中的短期关系，而z更新门是为了获得时序数据中的长期关系。

最后输出的yt=σ(W · ht)就是结果了。

普遍情况下，采取GRU而不是LSTM的原因是，GRU与LSTM相似，只是少了一些参数，但是训练一段时间后结果是类似的。而参数少的情况下训练成本就低一些，可以花更少的时间，也适合我这种机器性能不是很好的情况。

现在，我们就能理解GRU是如何使用更新和重置门来存储和过滤信息的。这消除了梯度消失的问题，因为模型不是每次都洗掉新的输入，而是保留相关的信息，并将其传递到网络的下一个时间步骤。如果仔细训练，他们甚至可以在复杂的情况下表现得非常好。



**3、数据预处理和代码实现**

**1）数据预处理**

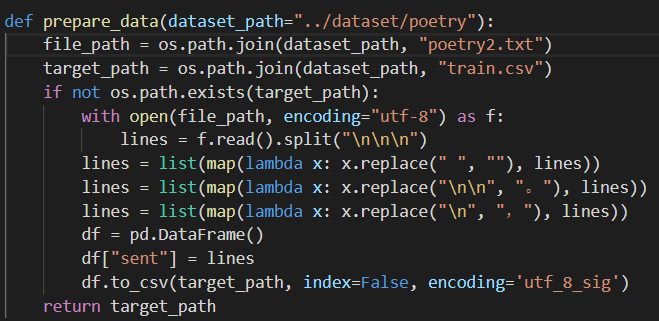
数据来源于著名诗人的现代诗代表作。具体地说我采用的是中华诗库里的近代诗人的作品。链接在<https://www.shigeku.org/shiku>，都能分别找到。我倾向于选择了朦胧诗派的作家，一共五个作家北岛、顾城、舒婷、徐志摩、余光中。他们都属于近当代诗的代表诗人。

平均每个人我大概选取了一万字的诗歌文本，总字符大概50000多一些，不是很大的语料库，这一方面限于gru算法的计算速度和我的计算机的计算能力，一开始训练的文本太长导致时间非常久，只能缩减长度，但是效果还是十分显著的。

首先第一个也是最重要的问题是把每个诗歌格式化。我采取的做法是把诗歌手动复制粘贴下来，然后在python里面处理。其实也很简单，就是打开file，然后file.replace掉一些不需要的地方，我删去了所有的不合适的空格、预先删掉所有的逗号和句号。然后放入到一个txt文件中。放入之后我保证每首诗的间隔是三个”\n”字符，然后诗中的每一段落间隔是两个”\n”字符。由于我预处理几乎删掉了所有的标点，在生成dataset的时候我又加上了标点以便于分割。

如图所示的代码中，我先按照三个换行分离每一首诗，分别用逗号和句号来替换了原来的一个和两个换行，把一首诗就这样放到同一个标记序列中。最后把这些诗歌放入excel可以打开的csv文件中。

我在提交的文件中包括了诗集的word/txt/csv三种形式，都是一步步处理的。最后的训练过程使用的是train.csv文件。



得到了train.csv之后就可以借助touchtext来构造训练集。

touchtext的预处理流程分为：

1，定义Field，表明如何处理数据

代码里面用了sequential=True实现序列化，通过两个init和eos的token定义了头尾字符。tokenize指定返回一个list

2，定义dataset，获得数据集，每一个样本是经过Field处理后的list

利用TabularDataset，加载之前的train.csv

3，构造vocab，建立词汇表，里面就有了词向量。到这一步，我们已经可以把词转为数字，数字转为词，词转为词向量了。里面可以调用stoi，itos，互相转换词和数字。

4，构造迭代器iterator，用来分批次放入gru训练我们的模型。使用了BucketIterator

到这里就是数据已经处理完成，可以开始核心的利用gru训练的部分了。

**2）核心代码实现**

实验环境：

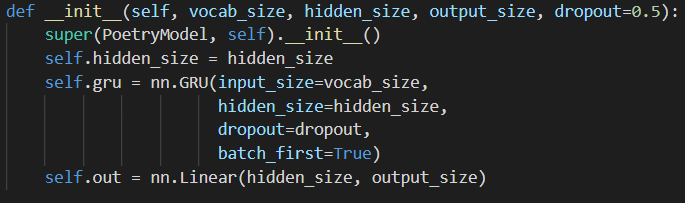
python 3.8.0

pytorch 1.7.0

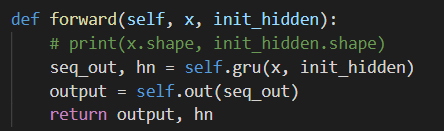
代码一共包括dataset里面的也就是上文提到的几个数据文件，dataloader是数据预处理，model是模型描述，main是主要的调控和交互。运行时只需要跑main.py就可以了。其实pytorch能够很智能地使用gru模型，就不需要用户自己构造了。调用的时候就只需要touch.nn.GRU然后输入参数。

首先是model的部分，其实结构很简单，就是构造了poetrymodel的类。里面包括自身初始化，向前传播和生成结果三个部分。

1）初始化就是给出词向量大小，隐藏层大小，输出大小的参数，调用nn.GRU. pytorch在这一方面十分简洁。dropout如果非零，则在除最后一层以外的GRU层的输出上引入一个Dropout层，Dropout的概率等于Dropout.

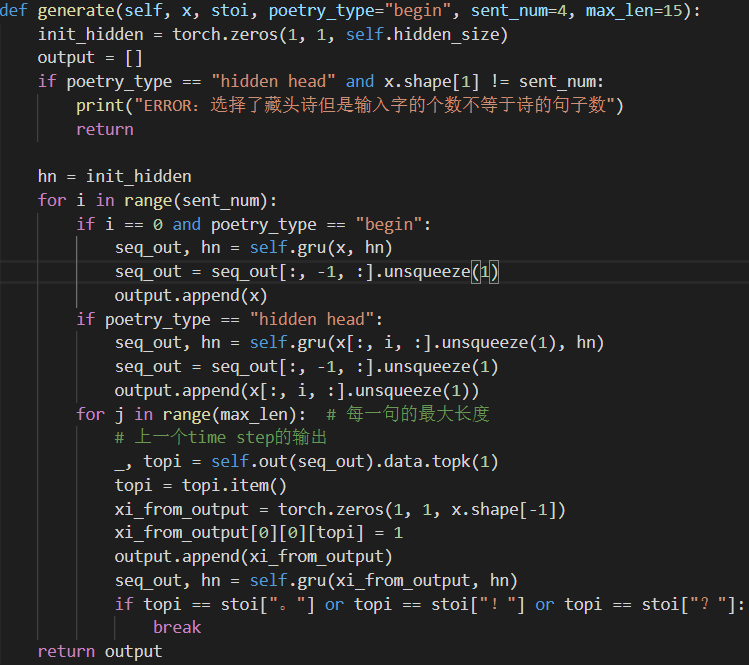


2）前向传播也因此变得很简单，每层只需要输入x和隐藏层init\_hidden，然后再调用线性带偏差的组合可以得到output了。这里的output和hn实际上对相应的是之前gru原理里面提到的下一层yt和ht.



3）generate的部分。根据选择“begin”或者“hidden head”进行结果的生成。这是在训练好之后的参数集上生成现代诗。

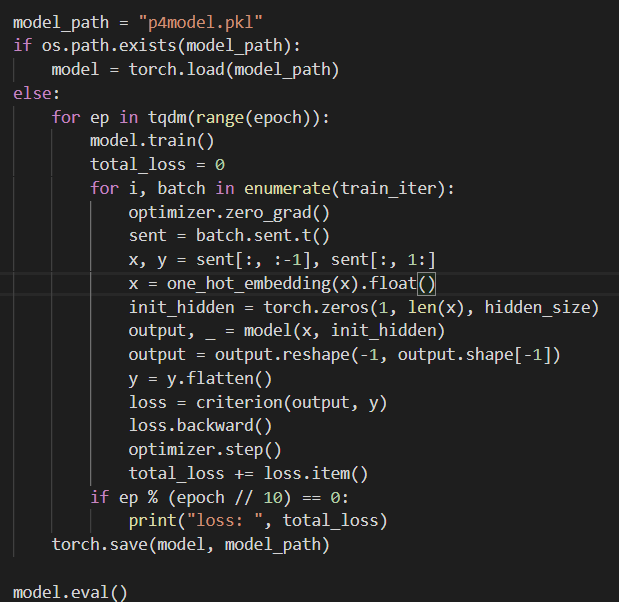
通过seq\_out控制每个字的输出，hn为隐藏层，然后把结果放在output这个list里面。此时output是词向量的数字集合，后面输出需要通过itos来转换成中文文字。



model就这几个部分了，剩下的是在main里面的具体训练计算，也就是如下代码。

使用了batch size为32，学习率为0.005。优化器optimizer是用adam优化器，损失criterion用交叉熵函数计算。

对于每一个epoch，直接很方便地调用model.train训练，然后计算损失，通过之前提到过的iterator来迭代，方便对batch进行处理。y是标准输出，output是训练的结果。y和output进行交叉熵计算回归分析。



以上就是核心代码的解释了，内容不是很多，有了touch.nn.gru后模型的描述变得比较简洁。但是由于数据的一些小杂乱，还有输出格式的问题，实验的结果也会有一些格式问题，但是大体上完成了现代诗的生成任务。

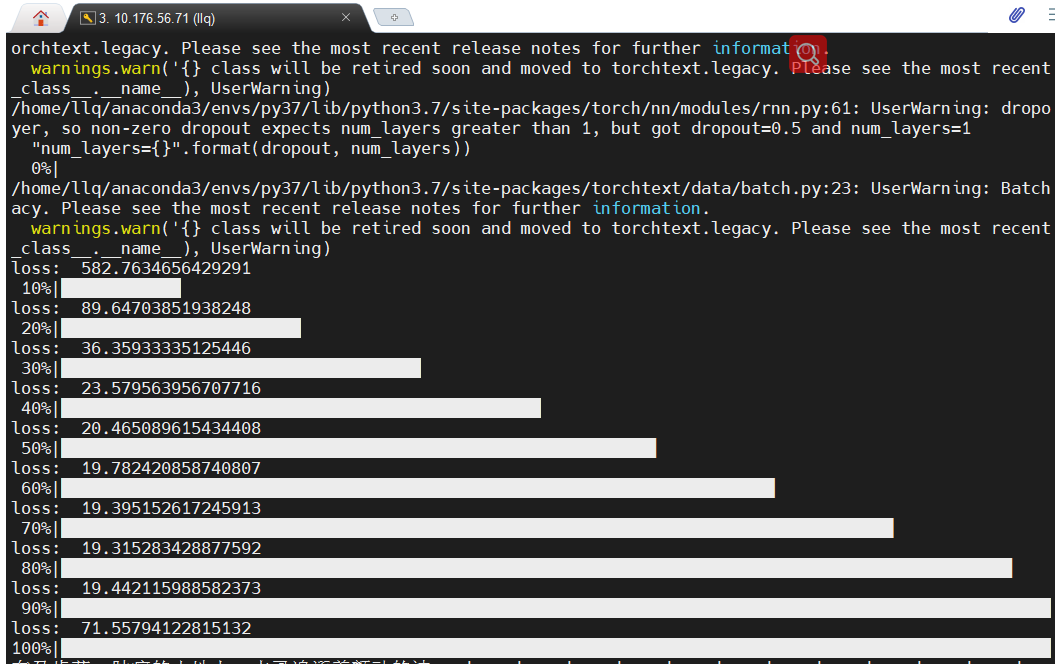
**4、实验以及实验结果**

我没有一个很量化的标准来做实验结果的评估，不像文本分类这样的任务可以有testset来测试是不是准确命中，正确率多少。因此，我的实验结果就是生成的比较好的诗的举例。当然不同的输入可能会导致不同的输出，有一些语法上和语义连贯的问题，可能是gru本身参数不够多，我的训练文本不够大，训练时间，epoch次数不够多都相关。但至少保证了诗的基本生成和诗意的存在，尤其是朦胧诗派的特点，充满着丰富的意象，注重场景的描绘和让人感觉到画面的感情色彩，就是朦胧不清晰的感觉。

由此推断出只要加入不同类别的语料就可以产生不同类式的诗歌。

我前后制作了三个训练集，p2model，p3model和p4model。p2是200次epoch，p3是1000次epoch，p4是200次epoch但把每个序列的头尾<start>和<end>标志去除。p4后来我借用了服务器训练所以训练结果截图有一点不一样。

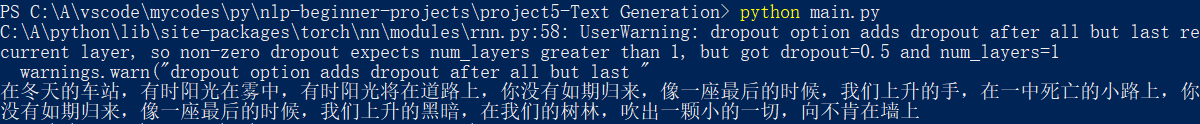
训练过程如下图，只记录了p4训练的时候。一定数量的epoch之后交叉熵算出来的总损失会显示在屏幕上。



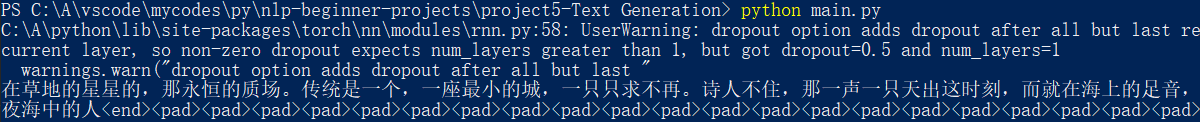
**1）只给出头部词语的实验结果**

这里我设定了一共是七句诗句。给出的开头是一个字“**在**”。p2测试结果。

第一句诗的开头有一种鲁迅先生的家门前有两棵树，一颗是枣树，另一颗也是枣树的手法感觉。并且很有冬季早晨雾气的画面感。

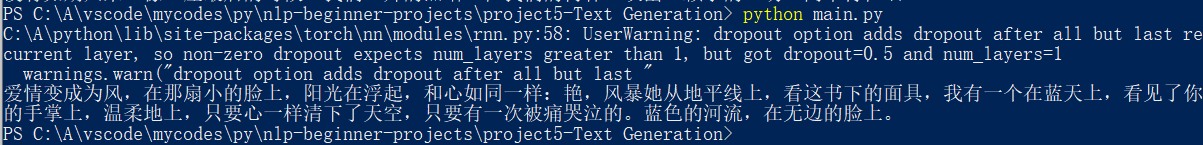


p3的测试结果

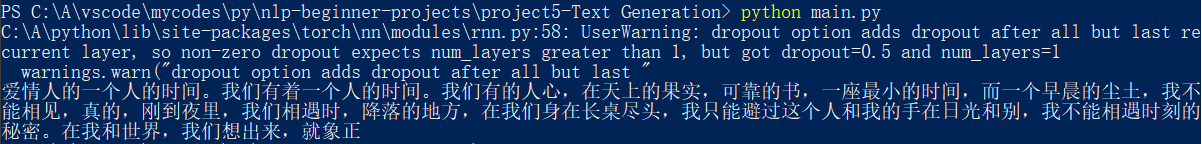


给出的开头是“**爱情**”，句数为7，p2测试结果。

第一句诗“爱情变成为风，在那扇小的脸上” ，有一种处于热恋状态的少女，小小的脸庞被风儿扑打得红扑扑的感觉，一般的人类读者都能有一些画面感。



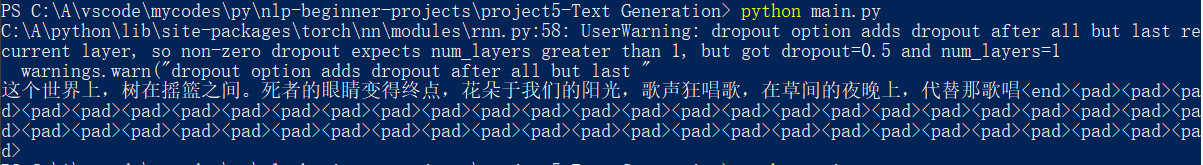
p3测试结果，这里我设置了更长的句数。



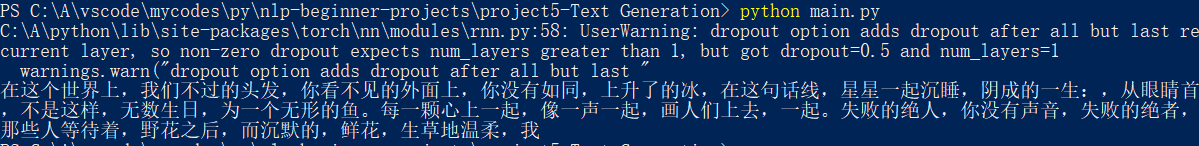
可以看到给出头部文字生成大段语料的时候，一开始的诗句都会上下句之间比较逻辑联系密切，而后面就有一种自由发挥的感觉。这也和gru的记忆能力有关，到后面会通过重置门r和更新门z选择性地遗忘一些之前的内容，也就没有那么连贯。这样的情况是必然的，越远离给出的开头的词，就越容易偏离最开始的输入。但是值得庆幸的是，每一段小诗句大都是挺融洽并且能够被人理解的。

下一个是输入“**在这个世界上**”为开头。p2测试结果

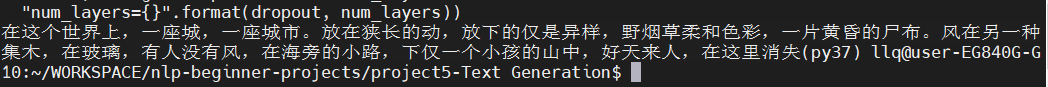
这也是我的模型的一个尚未完全完善的弊端，因为头字段输入过长，可能生成能力下降，很早地就遇到<end>的结尾，就不能继续下去了。考虑的解决方法是在生成训练数据集的时候，取消掉数据处理时touchtext.Field每一句的init\_token和eos\_token，限制就会减少了，也就是采用p4。



p3测试结果



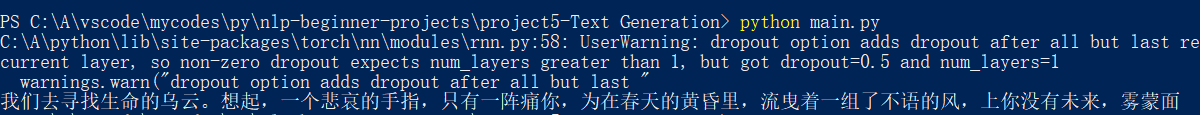
p4测试结果



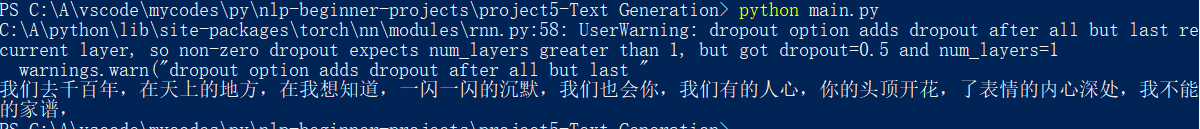
**2）藏头诗的实验结果**

藏头诗的实验结果并不是十分理想。我猜测是数据集的断句原因导致了测试的时候的断句不佳。没有严谨地实现藏头诗，而是把藏头的几个字放在了整首诗的诗句中间，比较平均的分布。因为结果呈现一般就不作过多展示了。

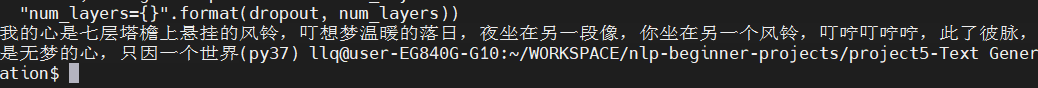
下图输入的是“**我想你了**”和seq\_num=4的四句诗。p2测试结果



p3测试结果



p4测试结果



藏头诗部分还比较原始和粗犷，效果不是很好，有待改进。

**5、结论**

整个项目结构比较分明，就和文件一样主要分为数据预处理、模型和训练三个部分，先是把txt数据处理成可以训练的词向量，然后描述模型，最后在主函数里面进行循环迭代训练，计算损失，得到比较好的一个训练结果并存放在pkl文件中。

比较好的部分是实现了给一些开头词语或者完整句子生成指定长度的诗，基本的语法问题不是很大。有待改进的地方是藏头诗的分割有些不够符合逻辑，也不能称之为藏头诗，可以视为多个开头诗糅杂在一起。总体上能达到有诗的韵味，具有一定的画面感，有朦胧诗派的特点，虽然缺少一些逻辑上的延申。

由于采用的是gru模型，我并没能在模型本身上进行很大程度的优化，所以很多工作实际上是在处理粗糙的数据集还有生成表达上。训练时间也比较久，平均是15秒一个epoch，1000个epoch大概将近四个小时的训练时间。虽然数据集量不是很大，训练次数也不够高，但是效果还是能显著体现的。

总的来说现代诗的生成任务基本能完成，现代诗的语法要求确实比古诗要高很多，这也就注定了gru在参数上的不足，我的数据集的规模，还有计算硬件的水平没办法达到一个非常完美的水准。但是有着很鲜明的改进方向，寻求更优质的模型，更大是数据规模，更好的计算能力的机器都会给结果增色。

参考链接

数据来源：<https://www.shigeku.org/>

代码参考：<https://github.com/positivepeng/nlp-beginner-projects>

touchtext：<https://www.jianshu.com/p/71176275fdc5>

GRU模型：<https://zhuanlan.zhihu.com/p/32481747>