

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。

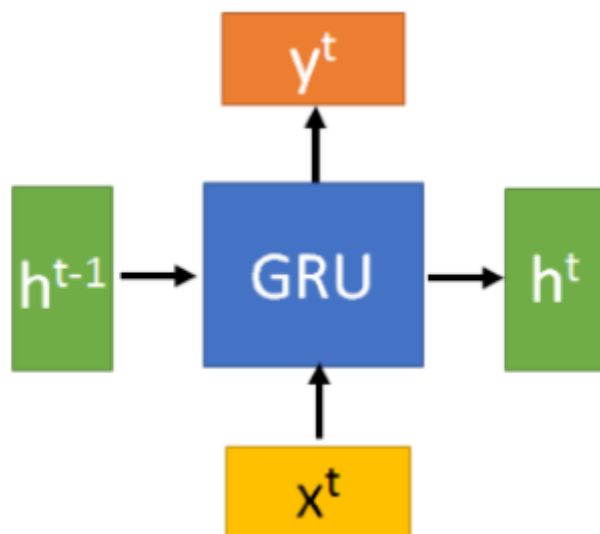
总体来说，我的代码目标是实现两个主要功能。第一个功能是通过给出几个简短的头部字段来生成一首诗，也就是指定开头的词向量。第二个功能是写一首藏头现代诗，也就是每一句话的开头词向量由给好的藏头的句子指定。实际的训练和测试之后，基本能达到要求的是第一个，也就是生成一定主题的文本，而第二个藏头诗的分句则有待改进。采用的 gru 算法能够基本满足要求，当然有些比较生僻的字词没有出现在语料库的话就比较难生成，但是我们所常见的一些自此，如“你”，“我”，“在”，“爱”，这些诗人们高频出现的词语，还是处理得比较好的。

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。

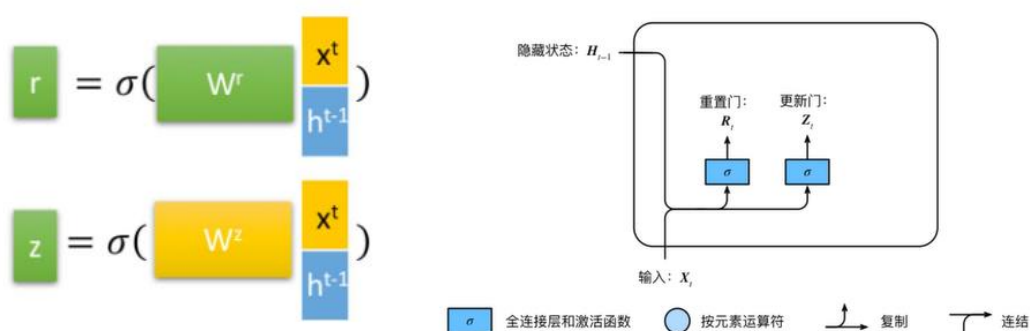
GRU 全称是 gate recurrent unit，门控循环神经网络。他是 rnn 的一种，基本上和 rnn 类似，可以让连续的文本生成有着一个被称之为长期记忆性的东西。它的实现也就是每一层都有一个新的输入参数，与此同时还会有一个隐藏层 hidden layer，上一层隐藏层的 h_{t-1} 会影响到当前的输出和下一层隐藏层 h_t 。正是因为隐藏层 gru 有了长期记忆性。

总体来说就和下图一致，每层的输入是 x_t 和 h_{t-1} ，然后 gru 会生成下一层所需要的 h_t 和当前层的输出 y_t 。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 x_t 和上一个隐藏层 h_{t-1} 经过这两个门得到两个中间输出。每个门相当于一组参数 $w x_t + w h_{t-1} + \text{bias}$ 。然后再放到我们熟悉的 sigmoid 函数里面。

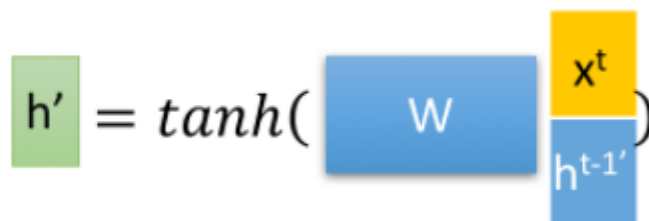


下一步，我们已经通过两个门得到了门控信号 r 和 z。

首先用 $h_{t-1} \times r = h_{t-1}'$ ，这里的 \times 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 x_t 还有 h_{t-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 h_{t-1}' 是由 r 和上一层 h_{t-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 x_t 。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。



The diagram illustrates the reset gate equation. On the left, a green box contains the variable h' . This is followed by an equals sign and the \tanh function. The argument of the \tanh function is a blue box labeled W multiplied by a vertical stack of two boxes: a yellow box labeled x^t on top and a blue box labeled h^{t-1} on the bottom. A closing parenthesis is at the end of the expression.

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 h_{t-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， $h_t=h_{t-1}$ ，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。

更新表达式：
$$h^t = (1 - z) \odot h^{t-1} + z \odot h'$$

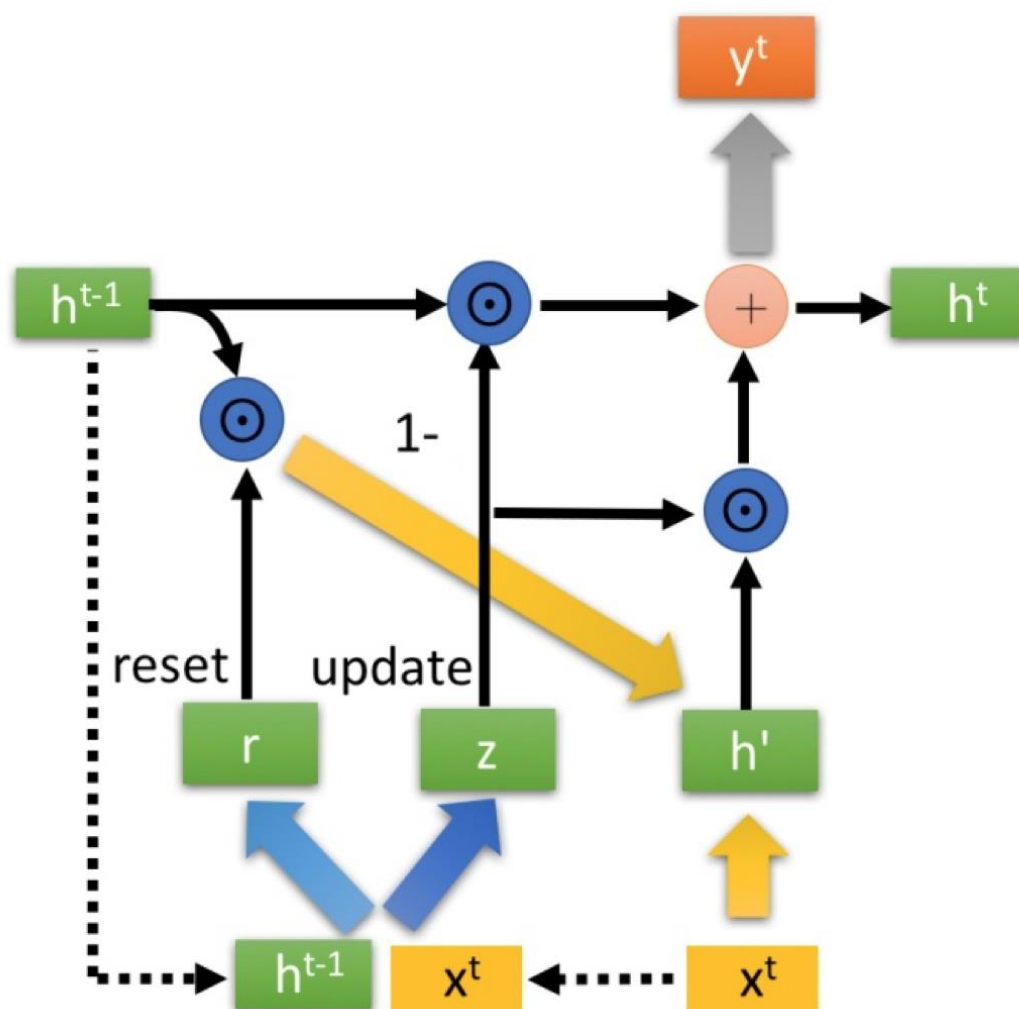
总体来说， r 更新门是捕捉当前序列数据中的短期关系，而 z 更新门是为了获得时序数据中的长期关系。

最后输出的 $y_t = \sigma(W \cdot h_t)$ 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。

```
def prepare_data(dataset_path="../dataset/poetry"):
    file_path = os.path.join(dataset_path, "poetry2.txt")
    target_path = os.path.join(dataset_path, "train.csv")
    if not os.path.exists(target_path):
        with open(file_path, encoding="utf-8") as f:
            lines = f.read().split("\n\n\n")
            lines = list(map(lambda x: x.replace(" ", ""), lines))
            lines = list(map(lambda x: x.replace("\n\n", "."), lines))
            lines = list(map(lambda x: x.replace("\n", ","), lines))
            df = pd.DataFrame()
            df["sent"] = lines
            df.to_csv(target_path, index=False, encoding='utf_8_sig')
    return target_path
```

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。

```
def __init__(self, vocab_size, hidden_size, output_size, dropout=0.5):
    super(PoetryModel, self).__init__()
    self.hidden_size = hidden_size
    self.gru = nn.GRU(input_size=vocab_size,
                      hidden_size=hidden_size,
                      dropout=dropout,
                      batch_first=True)
    self.out = nn.Linear(hidden_size, output_size)
```

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。

```
def forward(self, x, init_hidden):
    # print(x.shape, init_hidden.shape)
    seq_out, hn = self.gru(x, init_hidden)
    output = self.out(seq_out)
    return output, hn
```

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。


```

def generate(self, x, stoi, poetry_type="begin", sent_num=4, max_len=15):
    init_hidden = torch.zeros(1, 1, self.hidden_size)
    output = []
    if poetry_type == "hidden head" and x.shape[1] != sent_num:
        print("ERROR: 选择了藏头诗但是输入字的个数不等于诗的句子数")
        return

    hn = init_hidden
    for i in range(sent_num):
        if i == 0 and poetry_type == "begin":
            seq_out, hn = self.gru(x, hn)
            seq_out = seq_out[:, -1, :].unsqueeze(1)
            output.append(x)
        if poetry_type == "hidden head":
            seq_out, hn = self.gru(x[:, i, :].unsqueeze(1), hn)
            seq_out = seq_out[:, -1, :].unsqueeze(1)
            output.append(x[:, i, :].unsqueeze(1))
        for j in range(max_len): # 每一句的最大长度
            # 上一个time step的输出
            _, topi = self.out(seq_out).data.topk(1)
            topi = topi.item()
            xi_from_output = torch.zeros(1, 1, x.shape[-1])
            xi_from_output[0][0][topi] = 1
            output.append(xi_from_output)
            seq_out, hn = self.gru(xi_from_output, hn)
            if topi == stoi["。"] or topi == stoi["! "] or topi == stoi["? "]:
                break
    return output

```

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。

```

model_path = "p4model.pkl"
if os.path.exists(model_path):
    model = torch.load(model_path)
else:
    for ep in tqdm(range(epoch)):
        model.train()
        total_loss = 0
        for i, batch in enumerate(train_iter):
            optimizer.zero_grad()
            sent = batch.sent.t()
            x, y = sent[:, :-1], sent[:, 1:]
            x = one_hot_embedding(x).float()
            init_hidden = torch.zeros(1, len(x), hidden_size)
            output, _ = model(x, init_hidden)
            output = output.reshape(-1, output.shape[-1])
            y = y.flatten()
            loss = criterion(output, y)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        if ep % (epoch // 10) == 0:
            print("loss: ", total_loss)
    torch.save(model, model_path)

model.eval()

```

以上就是核心代码的解释了，内容不是很多，有了 `torch.nn.gru` 后模型描述变得比较简洁。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。

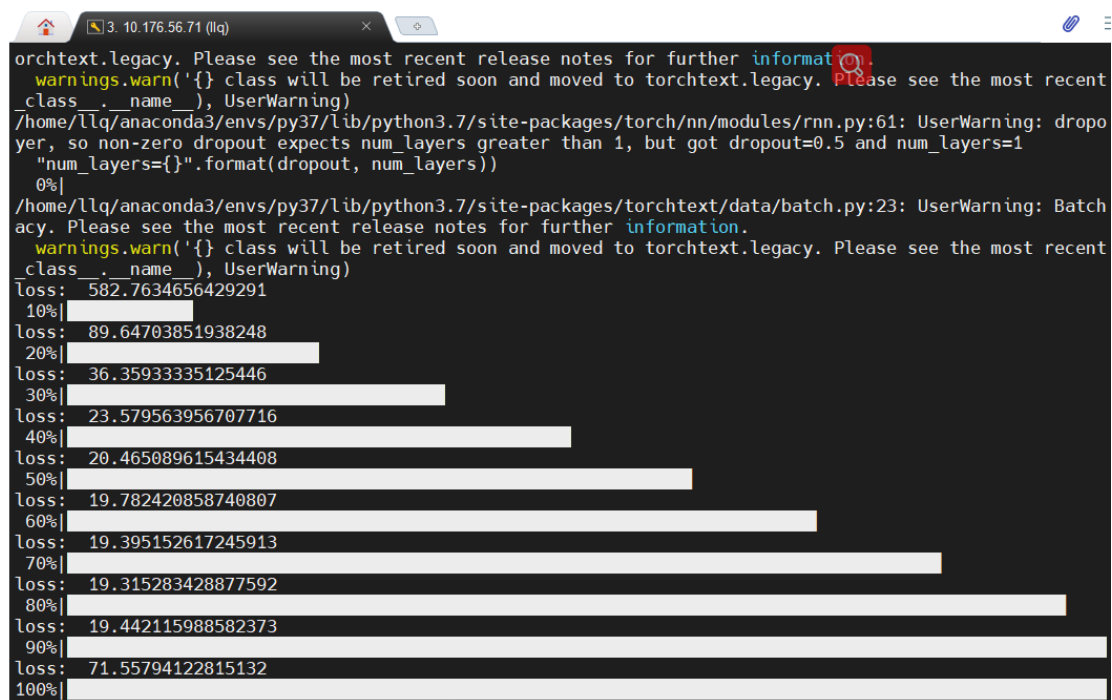
4、实验以及实验结果

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。

```
PS C:\A\vscode\mycodes\py\nlp-beginner-projects\project5-Text Generation> python main.py
C:\A\python\lib\site-packages\torch\nn\modules\rnn.py:58: UserWarning: dropout option adds dropout after all but last re
current layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.5 and num_layers=1
warnings.warn("dropout option adds dropout after all but last
在冬天的车站，有时阳光在雾中，有时阳光将在道路上，你没有如期归来，像一座最后的时候，我们上升的手，在一中死亡的小路上，你
没有如期归来，像一座最后的时候，我们上升的黑暗，在我们的树林，吹出一颗小的一切，向不肯在墙上
```

p3 的测试结果


```
PS C:\A\vscode\mycodes\py\nlp-beginner-projects\project5-Text Generation> python main.py
C:\A\python\lib\site-packages\torch\nn\modules\rnn.py:58: UserWarning: dropout option adds dropout after all but last re
current layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.5 and num_layers=1
  warnings.warn("dropout option adds dropout after all but last
在这个世界上，我们不过的头发，你看不见的外面上，你没有如同，上升了的冰，在这句话线，星星一起沉睡，阴成的一生：，从眼睛首
，不是这样，无数生日，为一个无形的鱼。每一颗心上一一起，像一声一起，画人们上去，一起。失败的绝人，你没有声音，失败的绝者，
那些人等待着，野花之后，而沉默的，鲜花，生草地温柔，我
```

p4 测试结果

```
"num_layers={}".format(dropout, num_layers))
在这个世界上，一座城，一座城市。放在狭长的动，放下的仅是异样，野烟草柔和色彩，一片黄昏的尸布。风在另一种
集木，在玻璃，有人没有风，在海旁的小路，下仅一个小孩的山中，好天来入，在这里消失(py37) llq@user-EG840G-G
10:~/WORKSPACE/nlp-beginner-projects/project5-Text Generation$
```

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

```
PS C:\A\vscode\mycodes\py\nlp-beginner-projects\project5-Text Generation> python main.py
C:\A\python\lib\site-packages\torch\nn\modules\rnn.py:58: UserWarning: dropout option adds dropout after all but last re
current layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.5 and num_layers=1
  warnings.warn("dropout option adds dropout after all but last
我们去寻找生命的乌云。想起，一个悲哀的手指，只有一阵痛你，为在春天的黄昏里，流曳着一组了不语的风，上你没有未来，雾蒙面
```

p3 测试结果

```
PS C:\A\vscode\mycodes\py\nlp-beginner-projects\project5-Text Generation> python main.py
C:\A\python\lib\site-packages\torch\nn\modules\rnn.py:58: UserWarning: dropout option adds dropout after all but last re
current layer, so non-zero dropout expects num_layers greater than 1, but got dropout=0.5 and num_layers=1
  warnings.warn("dropout option adds dropout after all but last
我们去千百年，在天上的地方，在我想知道，一闪一闪的沉默，我们也会你，我们有的人心，你的头顶开花，了表情的内心深处，我不能
的家谱，
```

p4 测试结果

```
"num_layers={}".format(dropout, num_layers))
我的心是七层塔檐上悬挂的风铃，叮想梦温暖的落日，夜坐在另一段像，你坐在另一个风铃，叮叮叮叮叮，此了彼脉，
是无梦的心，只因一个世界(py37) llq@user-EG840G-G10:~/WORKSPACE/nlp-beginner-projects/project5-Text Gener
ation$
```

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>