题目：基于LSTM的古诗文生成系统

### 完成人：王纵横 学号：SA19225391

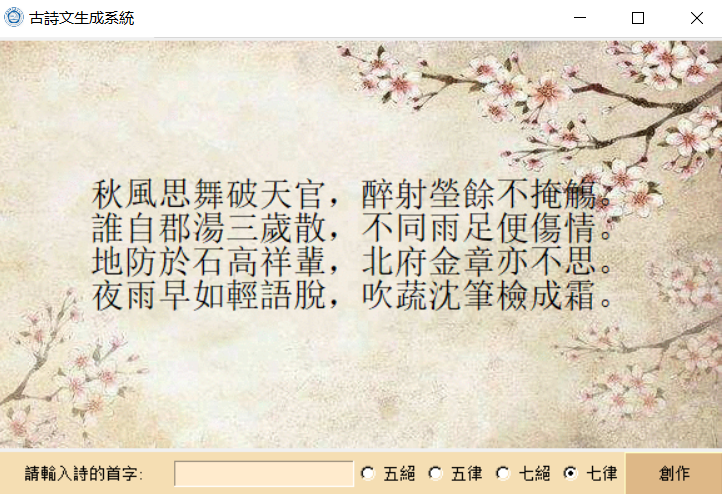
### 一、Demo运行

GitHub: <https://github.com/ArrogantW/Ancient-Chinese-poetry-Creator>

1、运行前请将model文件夹中四个文件夹里的checkpoint文件打开，将其中的文件路径根据当前保存路径进行修改。

2、运行create\_poetry.py文件，系统即可运行。

3、dispose\_data.py为数据预处理文件

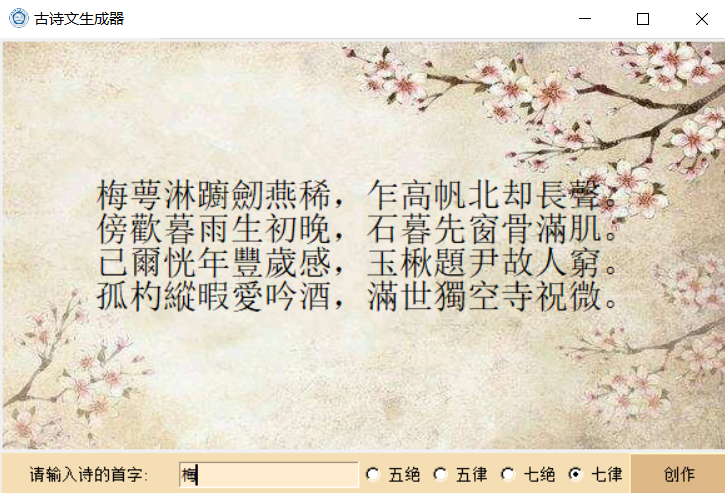
4、train.py为模型训练文件，训练时可根据需要修改model\_path和file\_path以及process\_poetry方法的第二个参数。

### 二、设计背景

百度在自动作诗方向做了很多探索。13 年手机百度 APP 推出了「为你写诗」功能，用户拍摄或上传一张图片，系统可以根据图片内容自动生成一首四句的古诗。它的核心算法是统计机器翻译技术（简称 SMT），是为写诗 1.0 版本，可以生成通顺、押韵的古诗，在对仗方面做的也比较好，但是在主题相关性方面较差。2016 年，百度在手机百度 APP 和度秘 APP 上先后推出了新版「为你写诗」功能，可以让用户任意输入题目生成古诗，这个版本可以称为写诗 2.0 版本。该版本使用了一种基于主题规划的序列生成框架，很好地解决了上一版中主题相关性差的问题。这里借鉴它的实现方法，使用动态双层单向LSTM网络，实现古诗文的生成。

### 三、设计目标

在系统界面的输入框中输入诗的首字，点击四个radiobutton中的任何一个，从而选择诗的类型，之后点击创作（也可直接点击），诗文显示在背景图片中。



### 四、技术路线

环境配置：TensorFlow1.14.1

Python3.7

CPU: intel CORE i7 9th Gen

1.首先，进行数据预处理，将诗文中的字转化为特定的数字，用其构建训练集。

2.其次，根据不同数据集生成的不同训练集，训练适用于生成不同类型诗文的模型并将训练好的模型保存。

3.最后，用户选择诗文的类型从而选择对应的模型，并将诗文的首字输入网络，或系统自动生成首字输入网络，从而实现诗文的生成。

### 五、关键原理

处理序列数据任务最有效的神经网络是循环神经网络（RNN），它是对时间序列上的变化进行建模的一种神经网络，可以基于之前的运行结果或者时间点，进行当前的预测。然而，RNN在处理长期依赖（时间序列上距离较远的节点）时，距离较远的节点之间的联系时会涉及雅可比矩阵的多次相乘，这会带来梯度消失（经常发生）或者梯度膨胀（较少发生）的问题。而LSTM（长短期记忆网络）可以解决梯度消失的问题，适合于处理和预测时间序列中间隔和延迟相对较长的重要事件。而让LSTM具有解决梯度消失问题能力的主要原因是以下两点：

遗忘机制：一个循环结束时，模型重置循环的相关信息，例如位置、时间等。当有新的输入时，模型会忘掉那些用不上的长期记忆信息。

保存机制：当有新的输入时，模型首先启动遗忘机制，之后会学习新输入的数据中包含的值得使用的信息，然后将其存入长期记忆中。

使用LSTM生成诗文时，模型要通过用户或系统输入的一个字，找出数据集中可能性最大的一个字作为后接字（这既是一次预测操作），然后将该后接字作为输入值输入模型，通过多次的预测操作，生成足够多的字就可以生成一首诗。这要求训练过程中，模型需要尽可能的记住每句诗中各个字之间的前后衔接关系，从而使生成的诗更有涵义，具备对仗、押韵的特征。

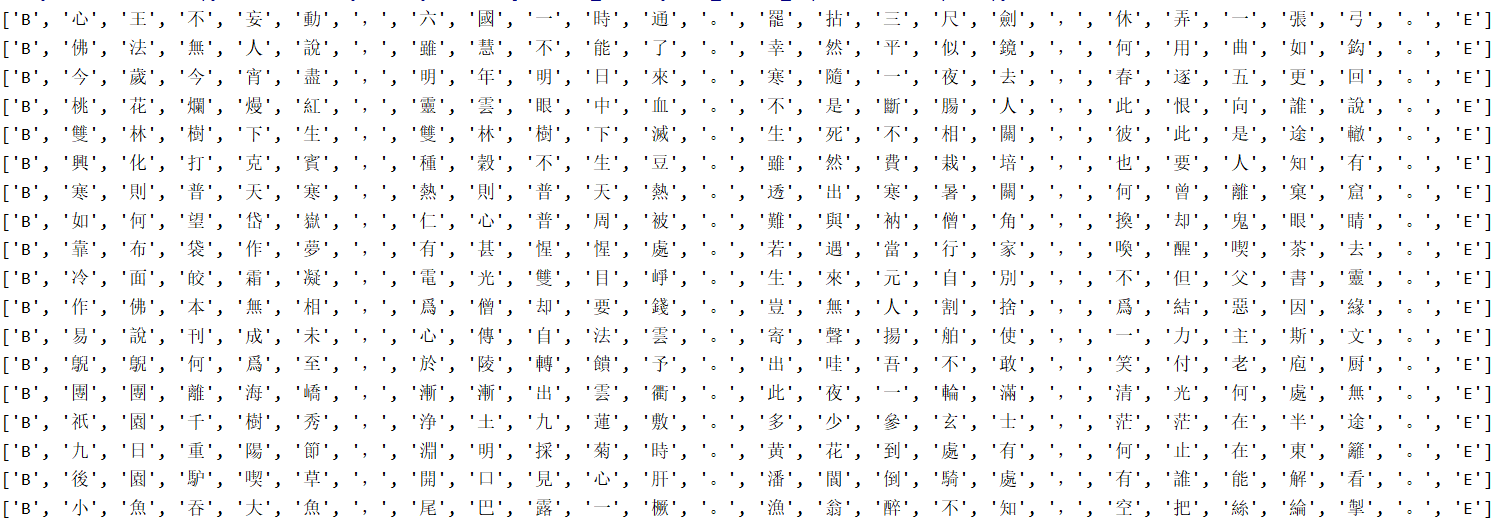
1. **数据预处理**

模型不会处理汉字，所以需要将汉字文本转化为数字。本项目所使用的数据集包括四个txt文件，包含了四种类型的诗文，即：五言绝句、五言律诗、七言绝句、七言律诗。数据集的来源为：<https://github.com/jinfagang/tensorflow_poems/tree/master/data>以五言绝句为例，文件如下图所示：



可以发现诗文都是在冒号之后的，且每行都是一首诗，这四个文件各包含了16995、60178、83069、67832首诗，各由6068、8283、8043、8731种字符组成。

我们逐行读取冒号之后的内容，将每行读取字数根据诗的类型进行限制，如五言绝句加上标点符号一共24个字，我们就只读取冒号后的24个字，当字数超出或字数不够时，放弃该行转而读取下一行。在读取完一行的内容后，在该诗的首尾位各插入一个标记字符（用’B’，’E’表示），用于将数据集中的每首诗隔绝开，则此时，每首诗共有26个字符。最后将读取的所有内容存在一个列表中，我称之为初始列表。将此列表每26个字符为一行表示，效果如下图：

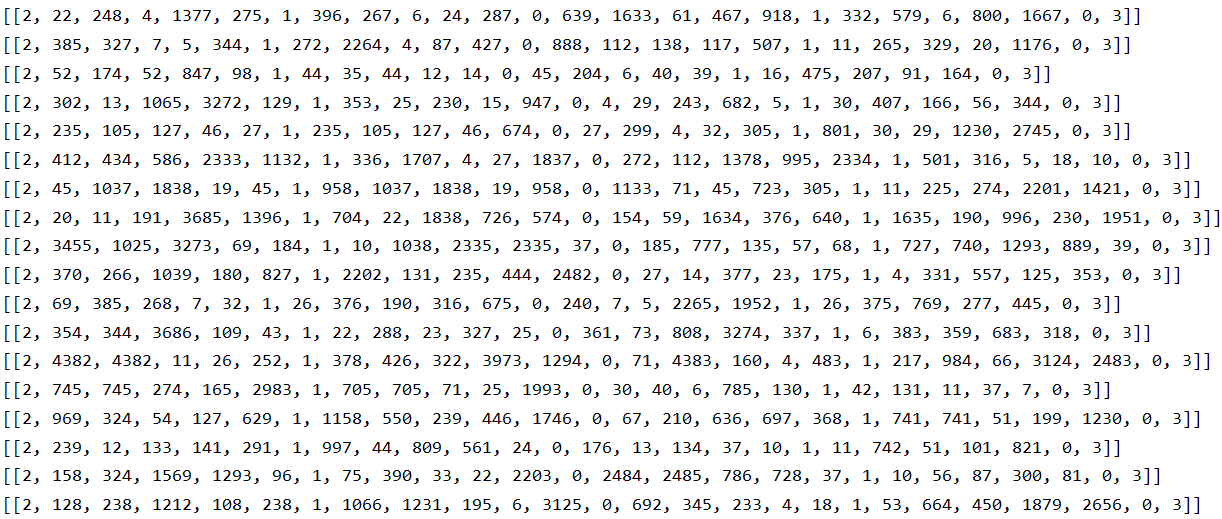


之后是将此列表中的汉字数字化的过程。

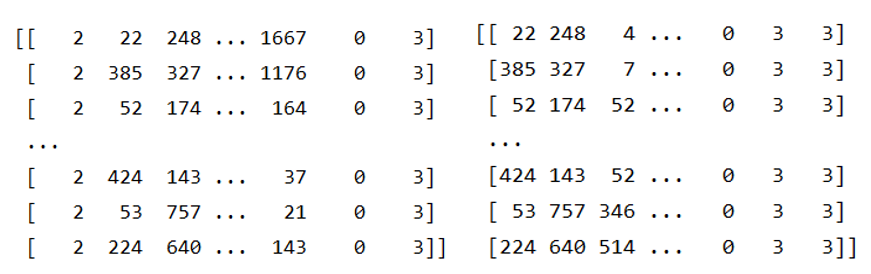
首先，我们统计此列表中每个字出现的次数，之后构建一个新的列表，将出现次数最多的字符放在最前，最少的放在最后，我称其为排序列表。

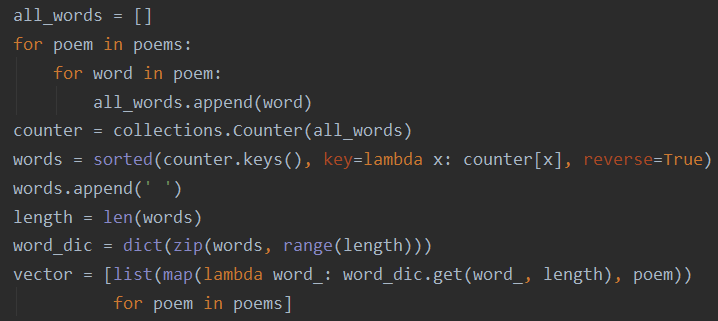
然后，将字符自身作为‘键’，它在排序列表中的索引作为‘值’，构建一个字典。

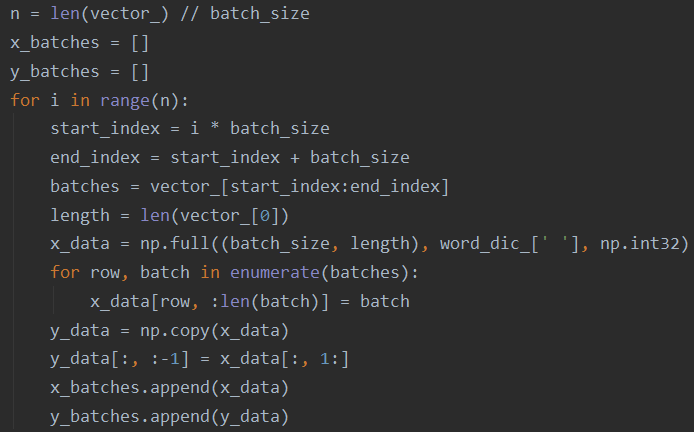
最后，根据字典，将初始列表中的每首诗用它们在字典中的‘值’（即数字）表示，之后放入一个列表中。以五言绝句为例，该操作会生成16995个数字列表，我们再将这些列表都放入一个新列表中，则列表是一个二维列表，形状为(16995,26)，我称之为操作列表。该列表即为数字处理后的初始列表，将其按第一维展开表示，效果如下（可与上图做对比）：



在将文本数据数字化以后，就开始构建训练集了。以五言绝句为例，由于模型每批次的训练集大小为64，所以我们在操作列表中每次取64行（即64首诗），使用numpy构建一个形状为(64, 26)的矩阵，此即为一个批次的训练集。因为，我们希望模型能够预测出每个字的后一个字，所以该训练集的标签矩阵是将每句诗中的字符向左移动一位，最后一位空缺处补3。最后，我们将16995//64个训练集矩阵和标签矩阵各自存储在一个列表中，这两个列表为总的训练集和标签。第一个训练集矩阵和标签矩阵如下图所示：







1. **定义模型并训练**

我尝试了很多不同结构的模型比如：动态单/双层双向LSTM，静态单/双层双向LSTM，但它们都出现了过拟合问题，预测出的诗文很糟糕。我也使用了LSTM的一些变体，比如：使用peephole的LSTM，GRU等，但它们的表现均没有LSTM好；而其他的一些更改，比如：使用单层或更多层LSTMCell、将tanh激活函数替换为relu激活函数等，其结果都差不多。故最终还是选择了动态双层单向LSTM模型。它包含了一个Embedding层，两层LSTMCell，每个隐藏层包含了128个神经元。其中Embedding层用来将训练集矩阵中的每一个元素处理为128个服从(-1,1)之间均匀分布的数，从而输入隐藏层中的128个神经元。模型的输入张量形状为(64, 26, 128)

之后是训练过程，由于四个数据集的大小不同，模型的训练次数也不同，对于五言绝句模型，由于其数据集相对较小，模型共训练15个轮次，每个轮次共训练了16995//64个批次。其余3个模型各训练5次。

LSTM的处理过程如下：

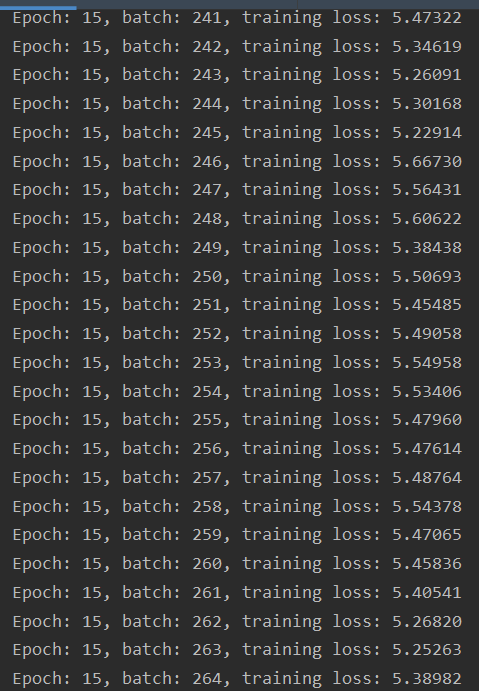
1. 输入被embedding处理后的数字，它会首先和上一个循环的输出结果进入遗忘门（功能为sigmoid函数），处理后输出一个介于0与1之间的遗忘信息，通过与上个循环输出的细胞状态中的每个数字相乘来决定每个字的遗忘程度，当它的值为1时，该字被完全保留；值为0时，该字被完全遗忘。遗忘信息的计算公式如下：
2. 之后和会被送入输入门（功能为sigmoid函数），处理后会输出一个更新信息，用来决定我们要更新什么值。于此同时，二者也被送入了一个tanh激活函数的隐藏层，它会创建一个候选值向量：。通过将该向量与更新信息相乘，就可以得到我们在本次循环要更新的值。更新信息与候选值向量的计算公式如下。
3. 此时，整合以上两个步骤，遗忘该遗忘的，保存该保留的，输出本次循环的细胞状态,其计算公式如下：
4. 最后，我们需要确定本轮循环的输出值。第一步，将x\_t和h\_(t-1)送入输出门（功能还是sigmoid函数），处理后得到一个决定信息，用来决定将细胞状态的哪一部分输出。于此同时，将细胞状态通过tanh激活函数处理得到一个(-1,1)之间的值，将它与决定信息相乘得到该轮循环的输出值。该值会被送入下一个循环，也被会送入下一层的LSTMCell重复以上步骤最后输出模型。决定信息和输出值的计算公式如下：

模型输出的同样是一个形状为(64,26,128)的三维张量，将其变形为形状为(1664,128)的二维张量，通过与一个服从正态分布的形状为(128,6068)的权重矩阵相乘得到一个形状为(1664,6068)的二维张量，我称其为logits。

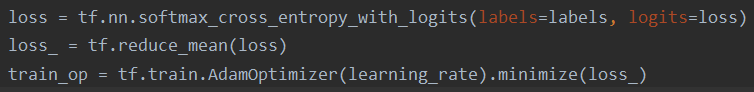
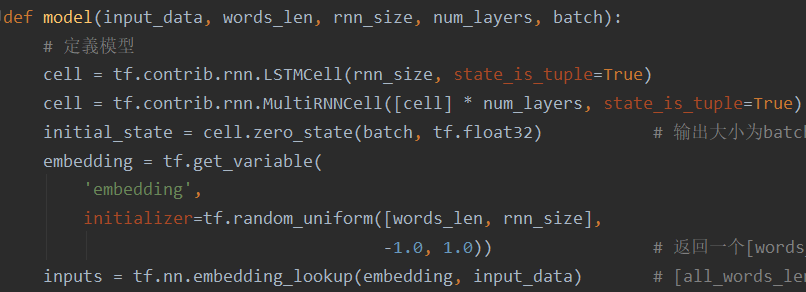
接下来就是通过损失函数来评估模型的训练结果了。我们首先将形状为(64,26)的标签张量独热化处理成形状为(1664,6068)的稀疏矩阵，我称其为labels。之后，结合使用tensorflow的softmax\_cross\_entropy\_with\_logits函数和reduce\_mean函数，将logits和labels作为参数，求出损失值loss。其中，前一个函数首先对logits进行softmax操作，将其第二维中的元素全部变为(0,1)之间的概率，而后输出其概率概率最大的数字与labels中对应的数字求交叉熵，输出一个长度为1664的交叉熵向量。后一个函数求出该向量中元素的平均值，作为损失值loss。softmax函数和交叉熵公式如下所示：

公式中，的最大值为6068，为预测的数字，为标签。

训练时的损失值输出如下所示：

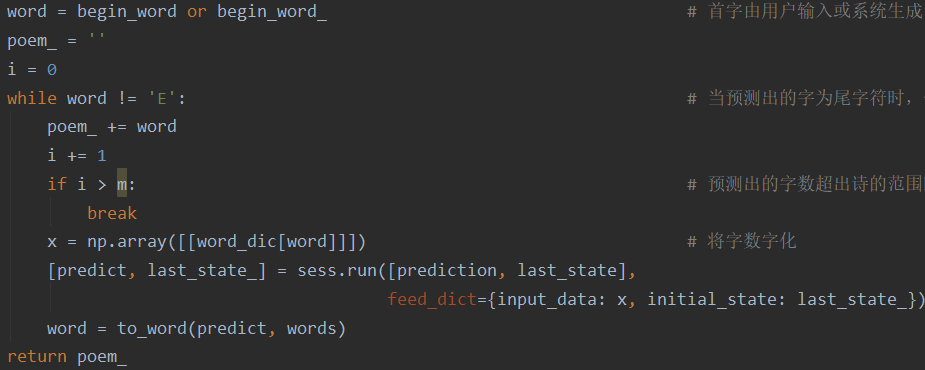


最后使用tensorflow的AdamOptimizer优化器对损失值进行梯度下降优化，学习率设置为1e-4。在训练结束后保存模型，用于预测。

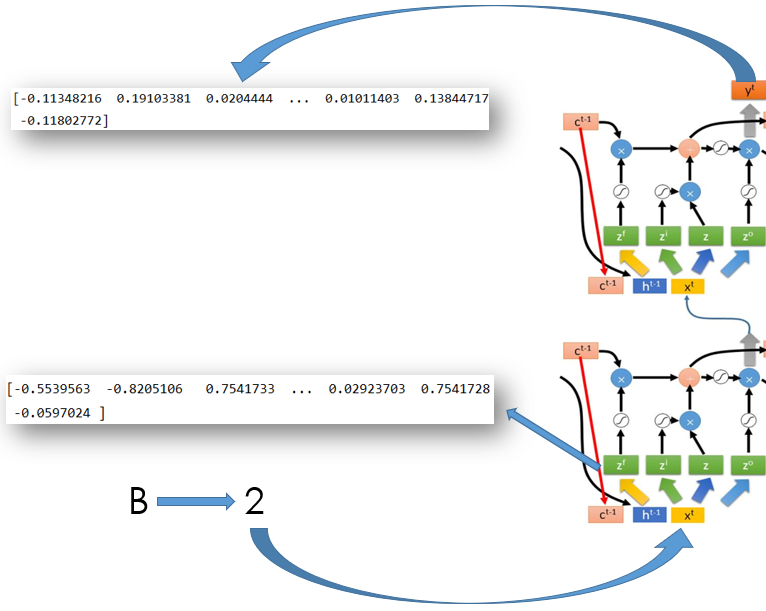


1. **诗文生成**

用户输入诗的首字，或者系统从数据集中随机选择一个字，通过字典找到其对应的数字，将其作为模型预测的第一个测试集。以五言绝句的模型为例，模型在得到该字后会输出一个形状为(1,6068)的logits,将其通过softmax函数处理，得到概率最大的数字，最后通过排序列表找出其对应的汉字。之后，再将该字输入模型，继续预测，直到输出足够多的字，组成一首诗。



### 四、具体实现

中心目标是构键下图所示的模型结构及训练过程。

其中，右侧的双层LSTM网络模型，源码体现如下：

def model(input\_data, words\_len, rnn\_size, num\_layers, batch):  
 cell = tf.contrib.rnn.LSTMCell(rnn\_size, state\_is\_tuple=True)  
 cell = tf.contrib.rnn.MultiRNNCell([cell] \* num\_layers, state\_is\_tuple=True)  
 initial\_state = cell.zero\_state(batch, tf.float32)   
 embedding = tf.get\_variable(  
 'embedding',  
 initializer=tf.random\_uniform([words\_len, rnn\_size],  
 -1.0, 1.0))   
 inputs = tf.nn.embedding\_lookup(embedding, input\_data)   
  
 outputs, last\_state = tf.nn.dynamic\_rnn(cell, inputs, initial\_state=initial\_state)   
 output = tf.reshape(outputs, [-1, rnn\_size])   
 weights = tf.Variable(tf.truncated\_normal([rnn\_size, words\_len]))   
 bias = tf.Variable(tf.zeros(shape=[words\_len]))  
 losses = tf.nn.bias\_add(tf.matmul(output, weights), bias=bias)   
 return losses, initial\_state, last\_state

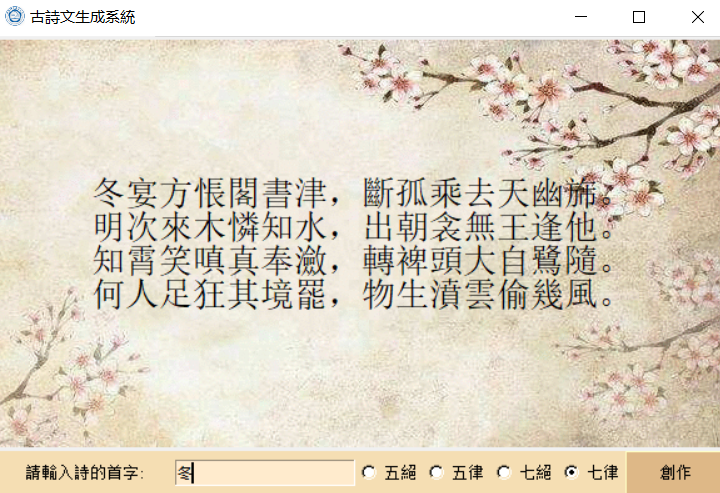
训练过程的源码体现如下：

def training():  
 vector, word\_dic, words = process\_poems(file\_path, '7jue')  
 batches\_inputs, batches\_outputs = get\_batch(batch\_size, vector, word\_dic)  
 input\_data = tf.placeholder(tf.int32, [batch\_size, None])  
 output\_targets = tf.placeholder(tf.int32, [batch\_size, None])  
 loss, \_, last\_state = model(input\_data=input\_data, words\_len=len(words), rnn\_size=128, num\_layers=2, batch=64)  
 labels = tf.one\_hot(tf.reshape(output\_targets, [-1]), depth=len(words))  
 loss = tf.nn.softmax\_cross\_entropy\_with\_logits(labels=labels, logits=loss)  
 loss\_ = tf.reduce\_mean(loss)  
 train\_op = tf.train.AdamOptimizer(learning\_rate).minimize(loss\_)  
 saver = tf.train.Saver(tf.global\_variables())  
 init\_op = tf.group(tf.global\_variables\_initializer(), tf.local\_variables\_initializer())  
 with tf.Session() as sess:  
 sess.run(init\_op)  
 n = len(vector) // batch\_size  
 for epoch in range(0, epochs+1):  
 m = 0  
 for batch in range(n):  
 losses, \_, \_ = sess.run([loss\_, last\_state, train\_op],  
 feed\_dict={input\_data: batches\_inputs[m],  
 output\_targets: batches\_outputs[m]})  
 m += 1  
 print('Epoch: %d, batch: %d, training loss: %.5f' % (epoch, batch, losses))  
 if epoch % 5 == 0:  
 saver.save(sess, os.path.join(model\_path, 'poems'), global\_step=epoch)

预测过程的源码如下：

### def gen\_poem(begin\_word, file, models, poem\_type): tf.reset\_default\_graph() n = [24, 48, 32, 64] List = ['5jue', '5lv', '7jue', '7lv'] if poem\_type in List: m = n[List.index(poem\_type)] batch\_size = 1 print('loading model from %s' % models) vector, word\_dic, words = process\_poetry(file, poem\_type) input\_data = tf.placeholder(tf.int32, [batch\_size, None]) loss, initial\_state, last\_state = model(input\_data=input\_data, words\_len=len( words), rnn\_size=128, num\_layers=2, batch=batch\_size) prediction = tf.nn.softmax(loss) saver = tf.train.Saver(tf.global\_variables()) init\_op = tf.group(tf.global\_variables\_initializer(), tf.local\_variables\_initializer()) with tf.Session() as sess: sess.run(init\_op) checkpoint = tf.train.latest\_checkpoint(models) saver.restore(sess, checkpoint) x = np.array([list(map(word\_dic.get, 'B'))]) [predict, last\_state\_] = sess.run([prediction, last\_state], feed\_dict={input\_data: x}) begin\_word\_ = to\_word(predict, words) word = begin\_word or begin\_word\_ poem\_ = '' i = 0 while word != 'E': poem\_ += word i += 1 if i >= m: break x = np.array([[word\_dic[word]]]) [predict, last\_state\_] = sess.run([prediction, last\_state], feed\_dict={input\_data: x, initial\_state: last\_state\_}) word = to\_word(predict, words) return poem\_

### 五、结果展示





### 六、存在的问题

问题：

生成诗文的语义不太明朗，考虑是训练不足，模型欠拟合。

模型训练20个轮次的预测效果：



模型训练30次预测的效果：



模型训练40次预测的效果：



可见，效果逐渐变好。

### 完成时间：2019/11/14