# 机器翻译大作业

# 基于RNN的古诗词生成

姓名：周正道

学号：2101871

# 1数据集及预处理

1.1数据集中数据基本样式

首春:寒随穷律变，春逐鸟声开。初风飘带柳，晚雪间花梅。碧林青旧竹，绿沼翠新苔。芝田初雁去，绮树巧莺来。首春:寒随穷律变，春逐鸟声开。初风飘带柳，晚雪间花梅。碧林青旧竹，绿沼翠新苔。芝田初雁去，绮树巧莺来。

根据要求，因为我们想训练的是写诗的内容，因此等下训练的时候只需要诗的内容即可。另外，我们的数据中可能存在部分符号的问题，例如中英文符号混用、每行存在多个冒号、数据中存在其他符号等问题，因此我们需要对数据进行清洗每首诗词包括诗的名字和诗句，在数据文件中每一行是一首古诗。所以首先我们按行来划分古诗。其次将冒号后面的部分保留下来。由于模型的设置，排除正文长度大于128的古诗词。最后去掉一些低频词构建词汇表。我们需要一些特殊字符，以完成特定的功能。这里使用的特殊字符有四个，为’[PAD]’, ‘[UNK]’, ‘[CLS]’, ‘[SEP]’，它们分别代表填充字符、低频词、古诗开始标记、古诗结束标记。代码中有一个类——Tokenizer，这是为了方便我们完成字符转编号、编号转字符、字符串转编号序列、编号序列转字符串等操作而编写的一个辅助类。我们需要一个词典形式的结构来讲词与编号一一对应，数据处理时将词变为编号，将编号数据放入到模型中训练，将训练出来的编号串在通过词典转译成我们想看到的古诗词形式

token\_to\_id(self, token)

"""

给定一个编号，查找词汇表中对应的词

:param token\_id: 带查找词的编号

:return: 编号对应的词

"""

id\_to\_token(self, token\_id)

"""

给定一个词，查找它在词汇表中的编号

未找到则返回低频词[UNK]的编号

:param token: 带查找编号的词

:return: 词的编号

"""

encode(self, tokens)

"""

给定一个字符串s，在头尾分别加上标记开始和结束的特殊字符，并将它转成对应的编号序列

:param tokens: 待编码字符串

:return: 编号序列

"""

decode(self, token\_ids):

"""

给定一个编号序列，将它解码成字符串

:param token\_ids: 待解码的编号序列

:return: 解码出的字符串

"""

好的，我们已经对数据进行了处理，并构建了分词器。为了方便训练，我们还需要再编写一个生成器，对数据集进行封装：

sequence\_padding(self, data, length=None, padding=None):

"""

将给定数据填充到相同长度

:param data: 待填充数据

:param length: 填充后的长度，不传递此参数则使用data中的最大长度

:param padding: 用于填充的数据，不传递此参数则使用[PAD]的对应编号

:return: 填充后的数据

"""

上述过程代码在程序文件dataset.py中。

# 2模型的建立

2.1

写成生成器的形式，主要出于内存方面的考虑。训练时需要对数据进行填充、转one-hot形式等操作，会占用较多内存。如果提前对全部数据都进行处理，内存可能会溢出。而以生成器的形式，可以只在要进行训练的时候，处理相应batch size的数据即可。

yield batch\_data[:, :-1], tf.one\_hot(batch\_data[:, 1:], tokenizer.vocab\_size)前面部分是数据x,后面部分是标签y。将诗的内容错开一位分别作为数据和标签，举个例子，假设有诗是“[CLS]床前明月光，疑是地上霜。举头望明月，低头思故乡。[SEP]”，则数据为“[CLS]床前明月光，疑是地上霜。举头望明月，低头思故乡。”，标签为“床前明月光，疑是地上霜。举头望明月，低头思故乡。[SEP]”，两者一一对应，y是x中每个位置的下一个字符。

当然了，以字符的形式举例是为了方便理解，实际上不论是数据还是标签，都是使用tokenizer编码后的编号序列。

还有一点不同的是，标签部分使用了one-hot进行处理，而数据部分没有使用。原因在于，数据部分准备输入词嵌入层，而词嵌入层的输入不需要进行one-hot；而标签部分，需要和模型的输出计算交叉熵，输出层的激活函数是softmax，所以标签部分也要转成相应的shape，故使用one-hot形式。

因为不涉及自定义程度比较高的层，所以我选择使用TensorFlow 2.0的高级接口tf.keras构建模型。使用tf.keras构建模型非常简单快捷。

以下是其中的主要结构：

import tensorflow as tf

from dataset import tokenizer

# 构建模型

model = tf.keras.Sequential([

# 不定长度的输入

tf.keras.layers.Input((None,)),

# 词嵌入层

tf.keras.layers.Embedding(input\_dim=tokenizer.vocab\_size, output\_dim=128),

# 第一个LSTM层，返回序列作为下一层的输入

tf.keras.layers.LSTM(128, dropout=0.5, return\_sequences=True),

# 第二个LSTM层，返回序列作为下一层的输入

tf.keras.layers.LSTM(128, dropout=0.5, return\_sequences=True),

# 对每一个时间点的输出都做softmax，预测下一个词的概率

tf.keras.layers.TimeDistributed(tf.keras.layers.Dense(tokenizer.vocab\_size, activation='softmax')),

])

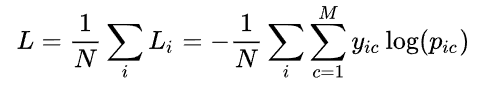
# 查看模型结构

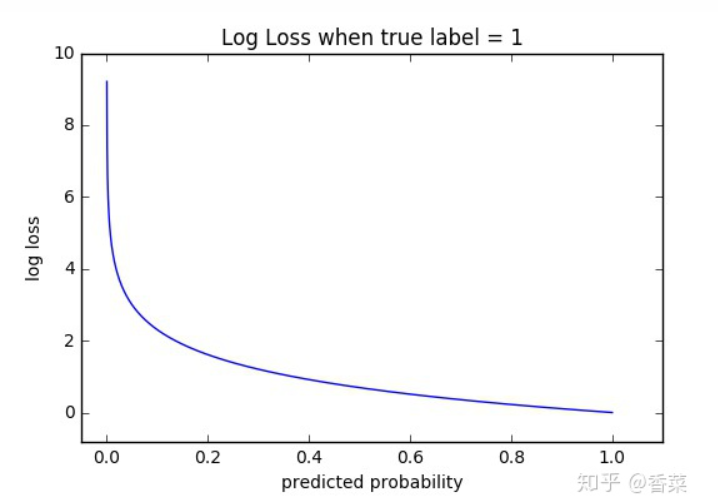
model.summary()

# 配置优化器和损失函数

model.compile(optimizer=tf.keras.optimizers.Adam(), loss=tf.keras.losses.categorical\_crossentropy)

使用tf.keras.Sequential构建了一个顺序的模型，选择Adam作为优化器，交叉熵作为损失函数。这里面损失函数的类型选择的考量是我们每次预测下一个诗词是根据概率来的，所以我们的损失函数选择了交叉熵损失函数。





可以看出，该函数是凸函数，求导时能够得到全局最优值。

模型的主要结构：

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, None, 128) 439552

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm (LSTM) (None, None, 128) 131584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

lstm\_1 (LSTM) (None, None, 128) 131584

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

time\_distributed (TimeDistri (None, None, 3434) 442986

Total params: 1,145,706

Trainable params: 1,145,706

Non-trainable params: 0

其中一些重要参数存储在setting.py的文件中

# 禁用词，包含如下字符的唐诗将被忽略

DISALLOWED\_WORDS = ['（', '）', '(', ')', '\_\_', '《', '》', '【', '】', '[', ']']

# 句子最大长度

MAX\_LEN = 64

# 最小词频

MIN\_WORD\_FREQUENCY = 8

# 训练的batch size

BATCH\_SIZE = 16

# 数据集路径

DATASET\_PATH = './poetry.txt'

# 每个epoch训练完成后，随机生成SHOW\_NUM首古诗作为展示

SHOW\_NUM = 5

# 共训练多少个epoch，20个epoch太大，调小一点比较快，但是效果明显较差

TRAIN\_EPOCHS = 5

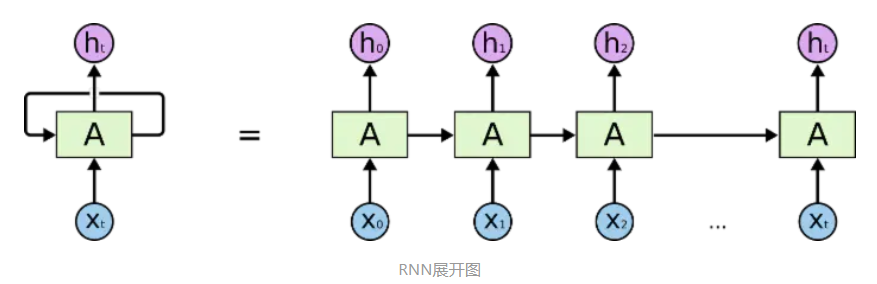
# 最佳权重保存路径

BEST\_MODEL\_PATH = './best\_model.h5'

## 2.2RNN

RNN是两种神经网络模型的缩写，一种是递归神经网络（Recursive Neural Network），一种是循环神经网络（Recurrent Neural Network）。虽然这两种神经网络有着千丝万缕的联系，但是本文主要讨论的是第二种神经网络模型——循环神经网络（Recurrent Neural Network）。

循环神经网络是指一个随着时间的推移，重复发生的结构。在自然语言处理（NLP），语音图像等多个领域均有非常广泛的应用。RNN网络和其他网络最大的不同就在于RNN能够实现某种“记忆功能”，是进行时间序列分析时最好的选择。如同人类能够凭借自己过往的记忆更好地认识这个世界一样。RNN也实现了类似于人脑的这一机制，对所处理过的信息留存有一定的记忆，而不像其他类型的神经网络并不能对处理过的信息留存记忆。



等号右边是RNN的展开形式。由于RNN一般用来处理序列信息，因此下文说明时都以时间序列来举例，解释。等号右边的等价RNN网络中最初始的输入是x0，输出是h0，这代表着0时刻RNN网络的输入为x0，输出为h0，网络神经元在0时刻的状态保存在A中。当下一个时刻1到来时，此时网络神经元的状态不仅仅由1时刻的输入x1决定，也由0时刻的神经元状态决定。以后的情况都以此类推，直到时间序列的末尾t时刻。

# 3训练模型

数据和模型都已就位，可以开始训练了。在这之前，先考虑一个问题，如何评价训练结果呢？除了loss以外，人工观察模型生成的古体诗的质量也是一个选择。让我们先编写几个工具方法，用于使用模型随机生成古体诗。这里有两个重要功能函数

def generate\_random\_poetry(tokenizer, model, s=''):

"""

随机生成一首诗

:param tokenizer: 分词器

:param model: 用于生成古诗的模型

:param s: 用于生成古诗的起始字符串，默认为空串

:return: 一个字符串，表示一首古诗

"""

def generate\_acrostic(tokenizer, model, head):

"""

随机生成一首藏头诗

:param tokenizer: 分词器

:param model: 用于生成古诗的模型

:param head: 藏头诗的头

:return: 一个字符串，表示一首古诗

"""

我们在每次训练完成一个epoch之后进行随机古诗的生成，生成五次。这样就能够比较直观的感受模型编写古诗的能力。

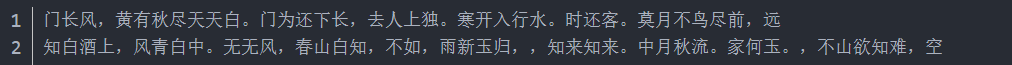
class Evaluate(tf.keras.callbacks.Callback):

"""

在每个epoch训练完成后，保留最优权重，并随机生成settings.SHOW\_NUM首古诗展示

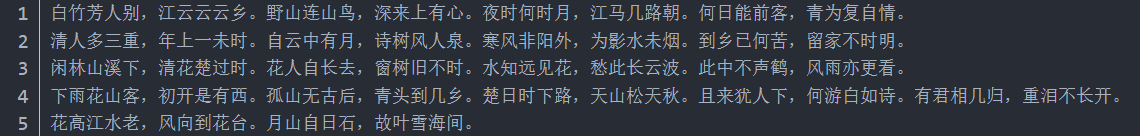
"""

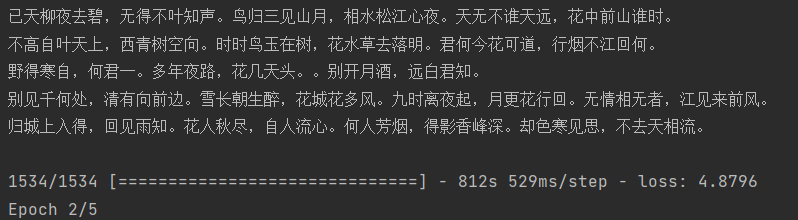
通过执行train.py脚本，即可完成模型的训练。观察模型在每个epoch结束后的表现是一件很有意思的事。刚开始训练，模型的输出有些乱七八糟的。字、标点胡乱的混在一起，完全没有格式。比如（每行是一首）



从第二、三个epoch开始，模型的输出有些格式了。但还是有些很明显的问题，比如标点用错，上下句子长度不同。比如



再几个epoch之后，意境和用词先且不谈，模型算是成功学到了句式：



训练的时候问题比较多，首先是gpu的tensorflow在电脑里运行有点问题，最后换的python3.6和tensflow2.0.0（cpu）

还有就是因为是amd的cpu计算能力不是很强，划片为1534迭代20次需要很久，中间还因为电脑问题重启了一次，最后将epoch设置为10。

# 4模型测试

将我们之前训练好的权重数据导入，试一试随机生成和藏头诗

import tensorflow as tf

from dataset import tokenizer

import settings

import utils

# 加载训练好的模型

model = tf.keras.models.load\_model(settings.BEST\_MODEL\_PATH)

# 随机生成一首诗

print(utils.generate\_random\_poetry(tokenizer, model))

# 给出部分信息的情况下，随机生成剩余部分

print(utils.generate\_random\_poetry(tokenizer, model, s='床前明月光，'))

# 生成藏头诗

print(utils.generate\_acrostic(tokenizer, model, head='海阔天空'))

# 5总结

比较简单的实现了一个古诗词生成模型，训练的过程比较慢，下次争取用gpu训练，还有就是这个古诗词不是很押韵，诗也没有标题，可以说还有很大改进空间。对模型的构建和函数的选择有了比较大的进步和更深刻了解。