



神经网络与深度学习

实验2：循环神经网络

姓 名: 张伯洋

学 号: 11519119

学 院: 人工智能

专 业: 人工智能

指导教师: 翟肇裕

20 21年11月29日

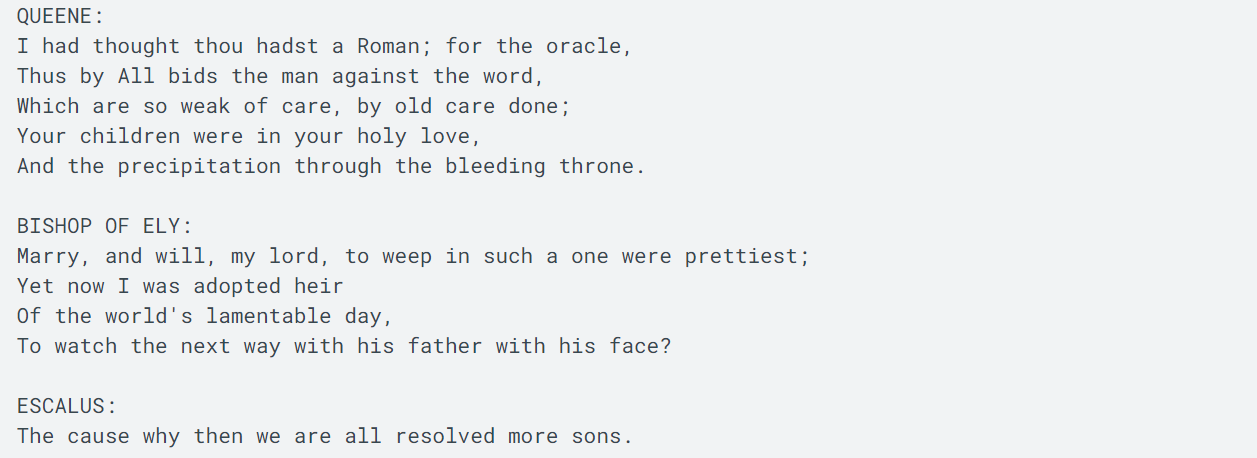
**循环神经网络实验：文本生成**

**（黄色高亮部分需独立编写或补充代码，以完成指定功能）**

**（绿色高亮部分需提供代码截图或实验结果截图）**

**一、实验目的**

在tensorflow深度学习框架下，利用基于字符的循环神经网络（character-based recurrent neural network）完成文本生成任务。本实验将使用威廉·莎士比亚的部分作品作为数据集，训练一个字符级循环神经网络，当给定起始字符（character）后，网络可以预测下一个字符。通过循环执行该网络，生成完整段落的文本。例如，当给定起始字符为“Q”时，网络输出的结果如下图所示（该结果是在训练30个周期后得到的）：



尽管上述文本存在诸多语法和拼写错误，且大多数句子没有意义，这是因为该循环神经网路暂未学习到单词的具体含义。但是考虑到这是一个基于字符的循环神经网络，在开始训练之前，网络甚至都无法拼写英语单词，且网络也并不知道文本的单位是单词。所以能够输出类似上述结果，即可认为达到实验目的。

**二、实验原理**

参考教材P133-156。

**三、实验内容**

本实验包含以下流程：

1. 检验并理解数据集；

2. 数据预处理；

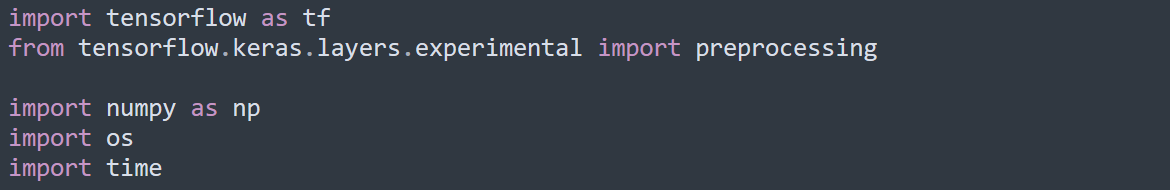
3. 构建循环神经网络；

4. 训练网络；

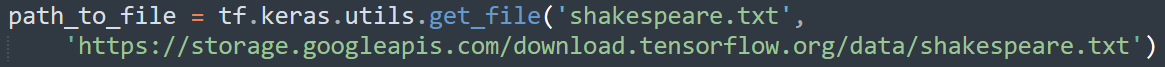
5. 生成文本。

**四、实验过程**

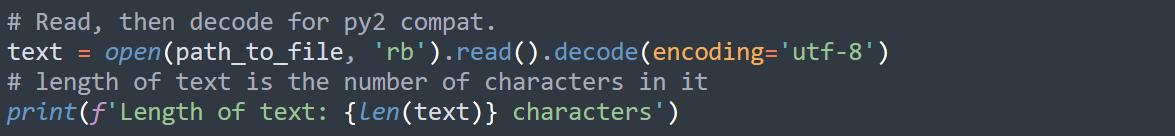
**Step 1. 导库。**导入tensorflow及所需要的库，示例代码如下图所示：



**Step 2. 下载并检验数据集。**本实验所使用的数据集请在学习通平台资料栏目内自行下载，下载完成后，加载数据集，示例代码如下所示（需根据数据集的路径，对示例代码做出更改）：



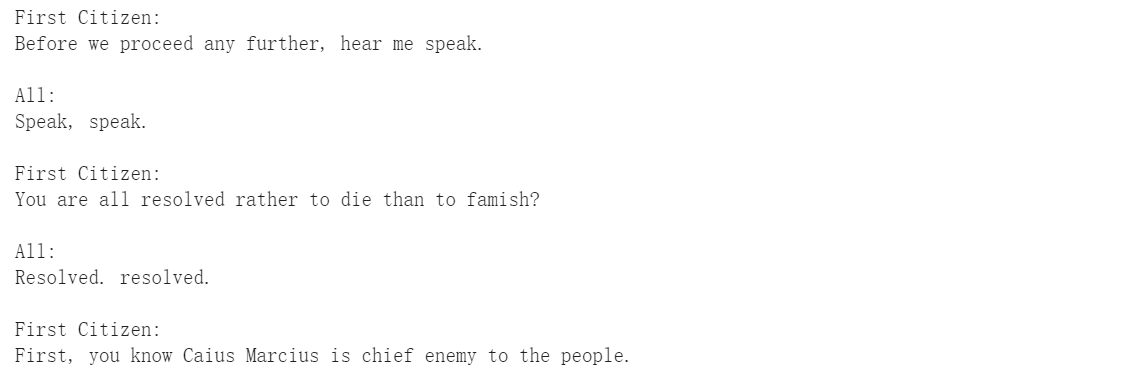
加载完成后，检验数据集，示例代码如下所示：



执行上述代码后，得到如下结果：



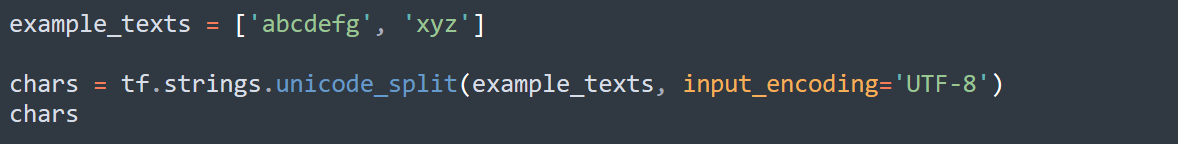
请编写代码，输出该数据集的前250个字符。预期结果如下所示：



**Step 3. 数据预处理。**在训练网络之前，需要将文本数据转换为数值型数据，即文本向量化。我们需要创建两个查找表格（Lookup table），一个将字符映射到数字，另一个将数字映射到字符。在tensorflow中，preprocessing.StringLookup层可以将字符转换成数值ID，也可以将数值ID转换为字符，该API的文档说明请参考如下链接：

<https://www.tensorflow.org/api_docs/python/tf/keras/layers/StringLookup>

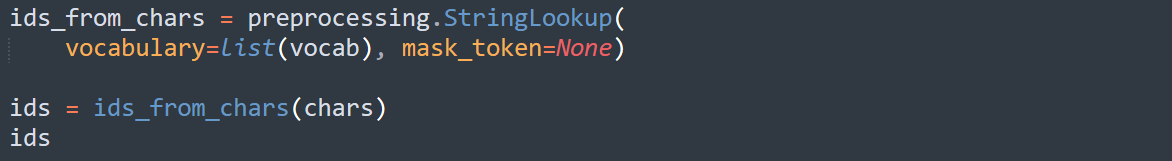
需要注意的是，使用该API时，需要先将文本数据拆分成tokens，以文本数据[’abcdefg’,’xyz’]为例，可使用如下代码将其拆分：



执行上述代码后，得到的输出结果如下图所示：



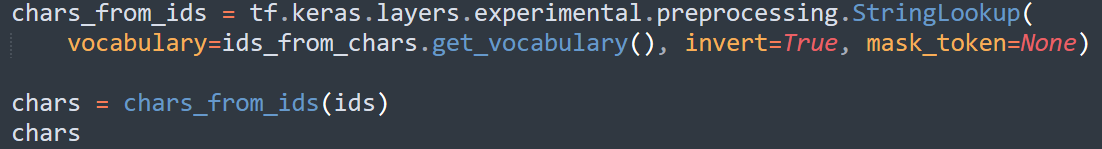
在完成拆分工作后，创建preprocessing.StringLookup层，示例代码如下所示：



执行上述代码后，得到的输出结果如下图所示：



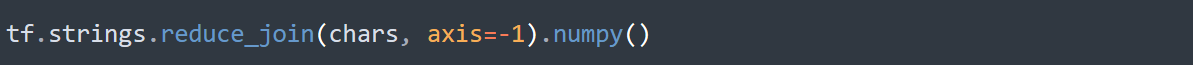
因为本实验的目的在于文本生成，所以需要考虑将数值型数据转换回文本形式。同样地，我们可以使用preprocessing.StringLookup层完成转换操作，示例代码如下所示：



执行上述代码后，得到的输出结果如下图所示：



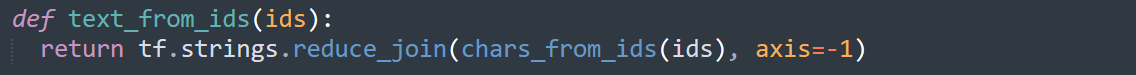
然后，通过调用tf.strings.reduce\_join这个API将字符拼接成字符串，示例代码如下所示：



执行上述代码后，得到的输出结果如下图所示：



为了后续使用方便，定义一个function，如下所示：

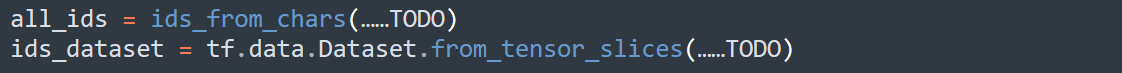


接下来，需要创建训练数据和目标。每个输入的文本序列将包含seq\_length个字符，当给定一个输入的文本序列后，网络需要输出一个与输入序列长度相同的文本序列，即在输入样本的基础上，向右顺移一个字符。例如，当seq\_length被设置为4时，我们想要预测“Hello”这个单词，这时，输入序列是“Hell”，目标序列即为“ello”。

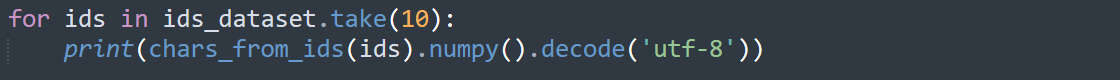
在本实验中，我们将使用tf.data.Dataset.from\_tensor\_slices函数来达成上述目的，该函数可以将文本向量转换为字符索引流（a stream of character indices），该函数的文档说明如下所示：

[https://www.tensorflow.org/api\_docs/python/tf/data/Dataset#from\_tensor\_slices](https://www.tensorflow.org/api_docs/python/tf/data/Dataset" \l "from_tensor_slices)

根据上述提示，请补全以下代码，完成文本数据的拆分和字符索引流转换任务。



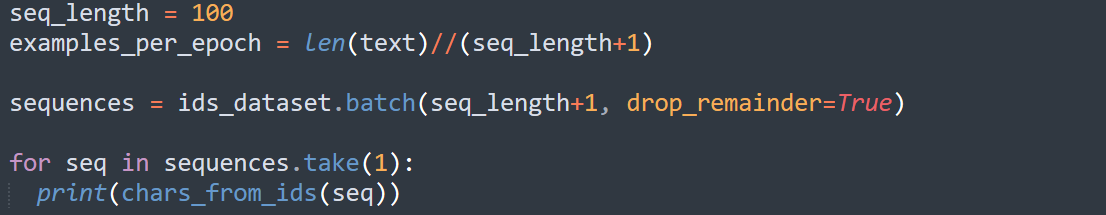
补全上述代码并执行后，查看结果，示例代码如下所示：



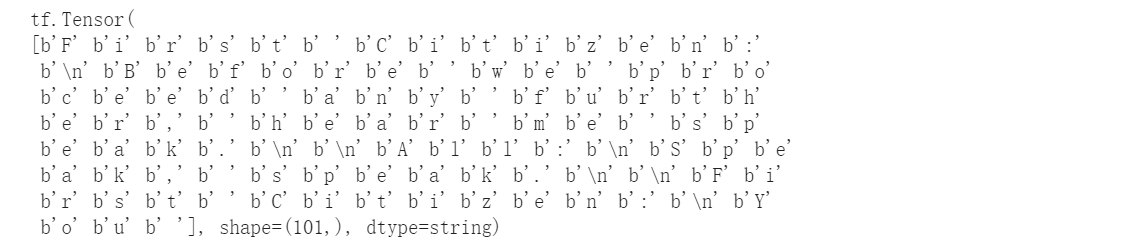
执行上述代码后，预期取得的结果如下图所示：



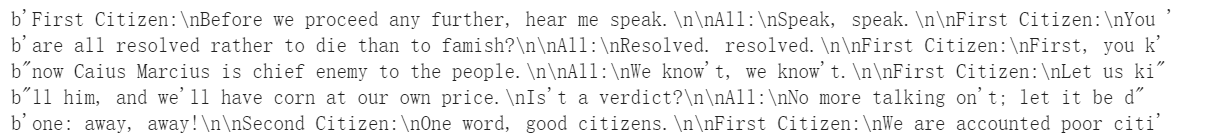
接着，设置seq\_length和examples\_per\_epoch，并使用batch方法将这些字符转换为所需长度的序列数据。示例代码如下所示：



执行完上述代码后，预期取得的结果如下图所示：



请编写代码，将这些字符拼接成字符串。预期取得的结果如下图所示：



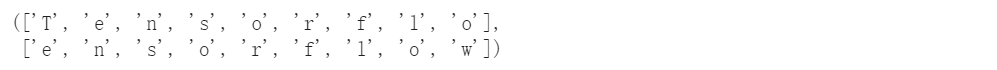
训练数据集通常的组成形式是（input,label），在本实验中，input和label都是以序列的形式存在的，在每一个time step中，输入的是当前的字符，输出的是当前字符的下一个字符。请补充如下代码，将输入的序列转换为input\_text和target\_text：



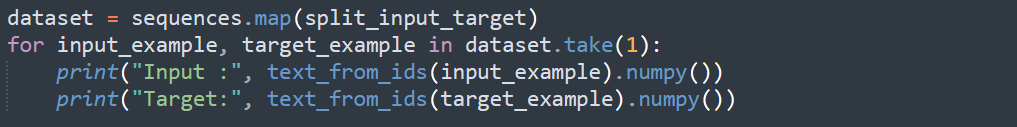
补充完上述代码后，以“Tensorflow”作为测试序列，验证定义的split\_input\_target函数，测试示例代码如下所示：



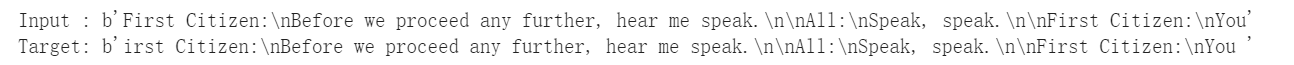
执行上述代码后，预期得到的结果如下图所示：



下面，将本实验所用的数据集通过map方法转换为（input,label）的形式，对于每个序列，map方法通过先复制，再顺移的方式，创建输入文本和目标文本。示例代码如下所示：

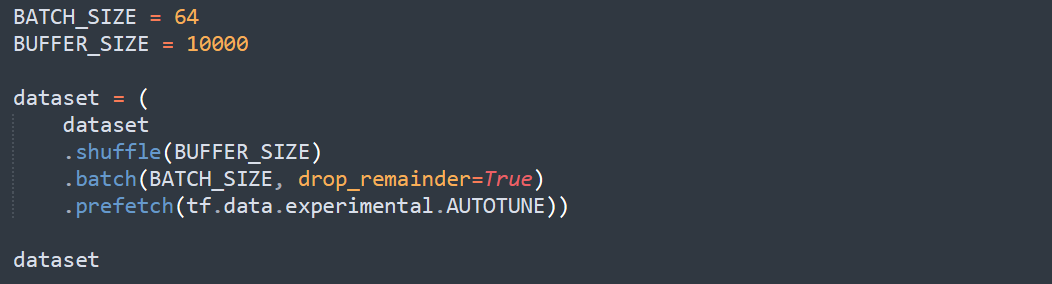


执行上述代码后，预期取得的结果如下图所示：



上图中，需注意的是最后一个单词“You”之后是一个空格字符。

最后，我们需要创建训练批次（Batch）。通过以上的步骤，我们已经成功将文本数据转换为所需序列，但是在将这些序列输入至网络之前，我们需要将这些序列重新排列（shuffle）并打包为批次，示例代码如下所示：



执行上述代码后，预期取得的结果如下图所示：



**Step 4. 构建循环神经网络。**该网络由以下3部分组成：

(1) tf.keras.layers.Embedding: 该层为输入层，在本实验中，该层的作用是使用embedding\_dim将数值索引（Index）映射到一个embedding\_dim维度的向量。tf.keras.layers.Embedding的文档说明请参阅如下链接：

<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Embedding?hl=zh-cn>

(2) tf.keras.layers.GRU: GRU是RNN的一种，在本实验中，GRU的大小为units=rnn\_units，或者，这里也可以使用LSTM。tf.keras.layers.GRU和tf.keras.layers.LSTM的文档说明请分别参阅如下链接：

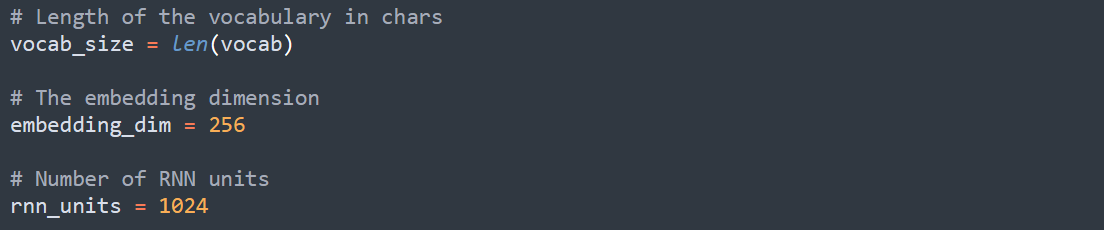
<https://www.tensorflow.org/api_docs/python/tf/keras/layers/GRU?hl=zh-cn>

<https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM>

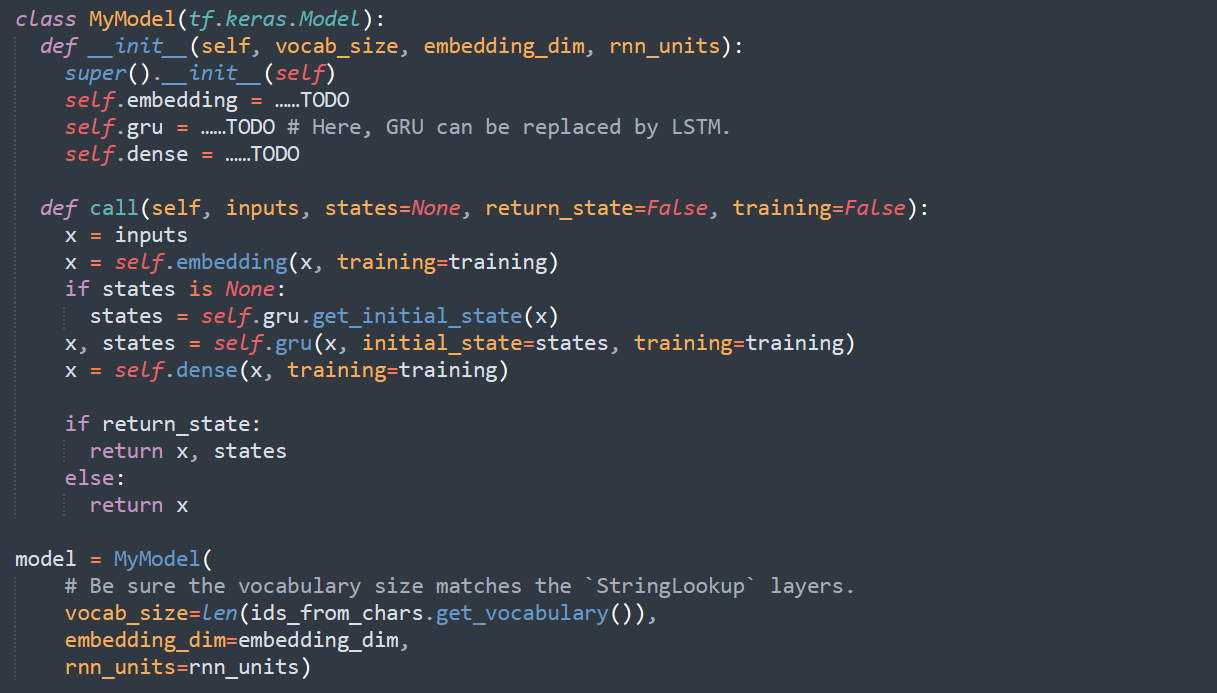
(3) tf.keras.layers.Dense: 该层为输出层，可以预测vocab\_size个输出。tf.keras.layers.Dense的文档说明可参阅如下链接：

<https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense?hl=zh-cn>

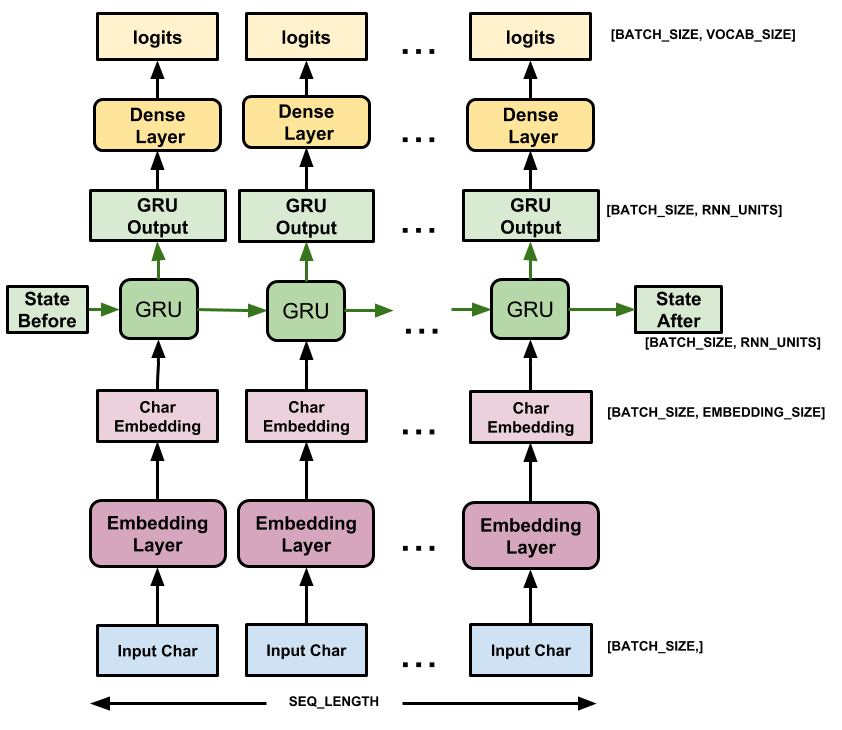
在定义循环神经网络之前，我们首先需要定义一些参数，示例代码如下图所示：



完成以上参数的定义后，请结合(1-3)中提供的文档说明，补充以下代码，完成对网络的定义和实例化：

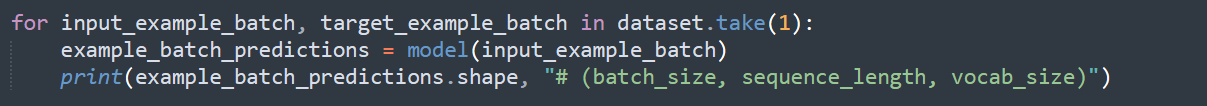


每一个time step中，网络会针对输入的字符寻找相应的embedding，并将找到的embedding作为输入传递至GRU（或LSTM）中，最后使用一个全连接层来预测下一个字符。完整的计算流程如下图所示：



上图中，需要注意的是输出层最后输出的是Logits。

构建完以上循环神经网络后，我们可以运行网络并做一些检查。首先，检查网络的输出形状，示例代码如下所示：



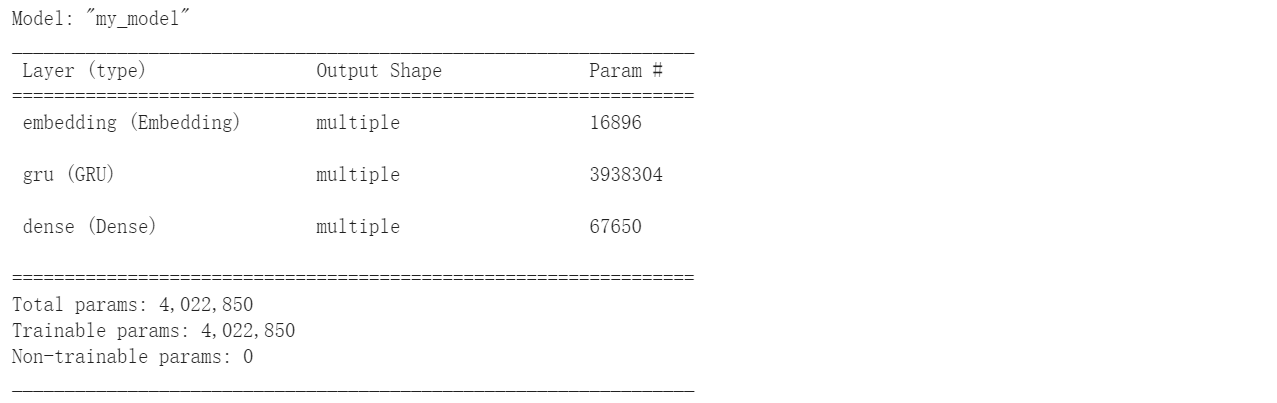
执行完上述代码后，预期取得的结果如下图所示：



同时，请打印网络的详细参数，示例代码如下所示：



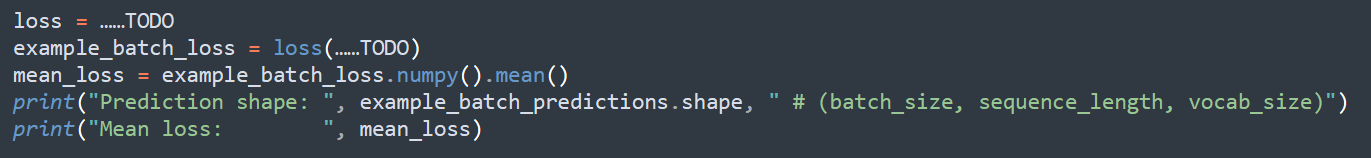
执行完上述代码后，预期取得的结果如下图所示：



**Step 5. 训练网络。**在这一步中，我们需要添加合适的优化器和损失函数。本实验所选用的是损失函数是sparse categorial crossentropy，其说明文档可参阅如下链接：

<https://www.tensorflow.org/api_docs/python/tf/keras/metrics/sparse_categorical_crossentropy?hl=zh-cn>

请阅读上述说明文档，补全以下代码，完成损失函数的定义（请注意from\_logits的设置），并思考为什么使用sparse categorial corssentropy作为本实验的损失函数，是否可以使用其他损失函数：



补全上述代码并执行后，预期取得的结果如下图所示：



一个刚刚初始化的网络通常并不能很好地完成预测任务，所以我们需要配置训练的过程，请阅读tf.keras.Model.compile的文档说明，补全以下代码，完成对网络训练过程的相关配置，链接如下：

[https://www.tensorflow.org/api\_docs/python/tf/keras/Model?hl=zh-cn#compile](https://www.tensorflow.org/api_docs/python/tf/keras/Model?hl=zh-cn" \l "compile)



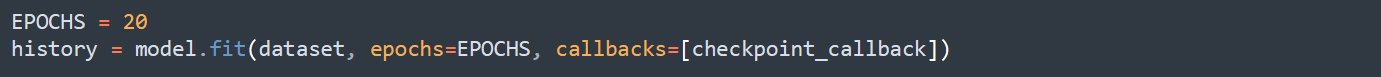
在网络的训练过程中，保存checkpoint也是非常关键的一步，checkpoint可以存储网络所使用的所有tf.Variables对象，可在恢复网络的时候使用。针对保存checkpoint，tensorflow框架提供了tf.keras.callbacks.ModelCheckpoint，其文档说明可参阅如下链接：

<https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint?hl=zh-cn>

保存checkpoint的示例代码如下图所示：



在设置完所有配置后，执行网络的训练。示例代码如下所示：



执行上述代码后，预期取得的结果如下图所示：

Epoch 1/20

172/172 [==============================] - 26s 128ms/step - loss: 2.7395

Epoch 2/20

172/172 [==============================] - 24s 127ms/step - loss: 1.9989

Epoch 3/20

172/172 [==============================] - 23s 127ms/step - loss: 1.7214

…………

Epoch 18/20

172/172 [==============================] - 25s 134ms/step - loss: 0.8387

Epoch 19/20

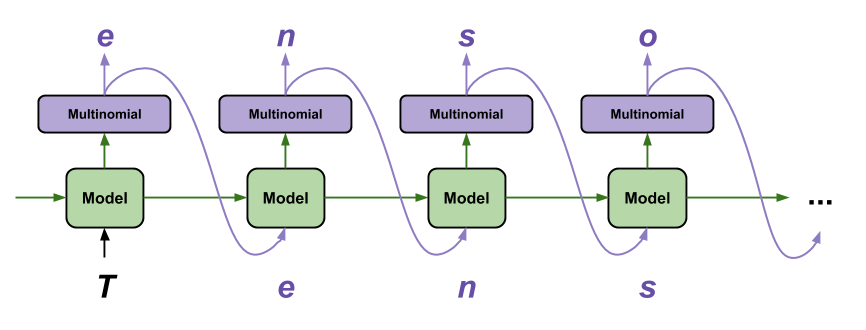
172/172 [==============================] - 24s 133ms/step - loss: 0.7869

Epoch 20/20

172/172 [==============================] - 25s 134ms/step - loss: 0.7384

注：在Google Colab的环境中，使用12GB RAM和GPU的情况下，以上的训练过程耗时10分30秒。如在个人笔记本电脑上训练该网络，训练时间可能会因硬件条件限制而变长，例如使用8GB RAM和CPU运行的情况下，训练过程可能需要耗时1-2个小时。

**Step 6. 生成文本。**在文本生成的过程中，需要重复运行网络，并追踪网络的内部状态（internal state），如下图所示：

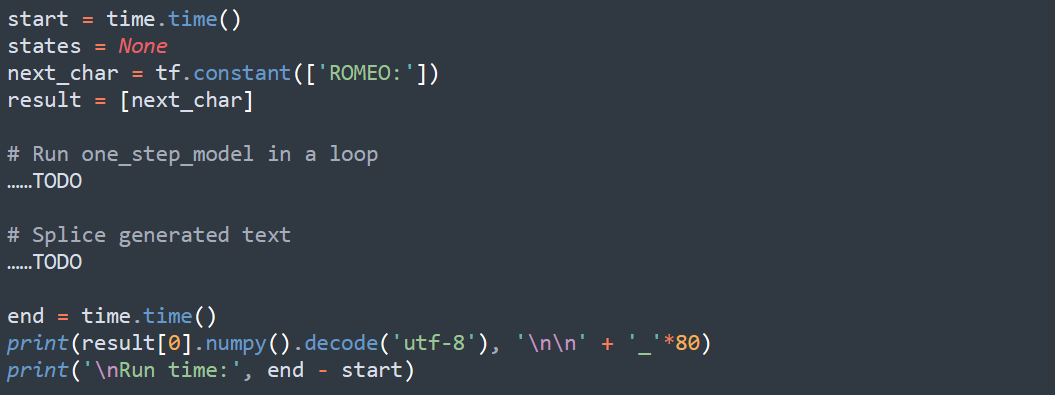


如上图所示，每当我们输入一个新的字符，网络会考虑当前的输入和过去时刻的内部状态，从而给出当前时刻的输出（即下一个最后可能出现的字符）并更新当前时刻的内部状态。通过不断地循环运行该网络，得到完整的文本内容。以下展示了one step训练的过程，请补全代码（注：temperature代表随机性，若temperature值较低，例如0.1，则生成的文本随机性较低，若temperature值较高，例如0.9，则生成的文本随机性较高）：





至此，我们可以通过循环执行的方式生成一些文本。通过观察网络输出的文本内容，可以发现网络现在已经知道何时需要将字母大写，何时创建新的自然段，同时，网络会尽可能地模仿莎士比亚的用词。请补全以下代码，循环执行one\_step\_model，生成一段文本。



执行上述代码后，预期取得的结果如下所示：

ROMEO:

Be it forgive; but they'll just away this boy

Thy due deny, when I quaked? he's money?

I wish you a conclude imageners

Against the king and not anon? whichers:

You are well exile Henry's height of York

Hath power enough: if is his life's curse, longs,

Untogethon of gross time bolong,

And watch Rivave from nothing rages like former'd haste,

And in this rude rounding farther house to purge.

GLOUCESTER:

Come, madam, let's away. The nurse, and But, alas!

CLAUDIO:

No murders, gates sure my brother: where they did

The day is forenunt? at this contract tognts! Pray you, but I can tell you good?

GLOUCESTER:

Unhappy brother, worthy Margaret's curse;

But I say it is a little five worance, in the gods

And waking and our tenth, and by our talk of care

To you a little knee, below the marmest tworn,

And sometime comes for this withwarter:

lay himself and unroof'd inacciquition,

Yet are she and love you well: do you need here,

Will win the heavens for me to tail yields.

SGRGo:

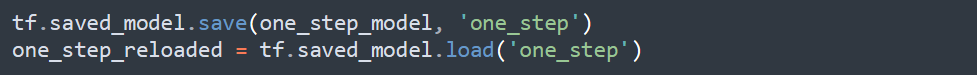
Thy face is mine, L

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

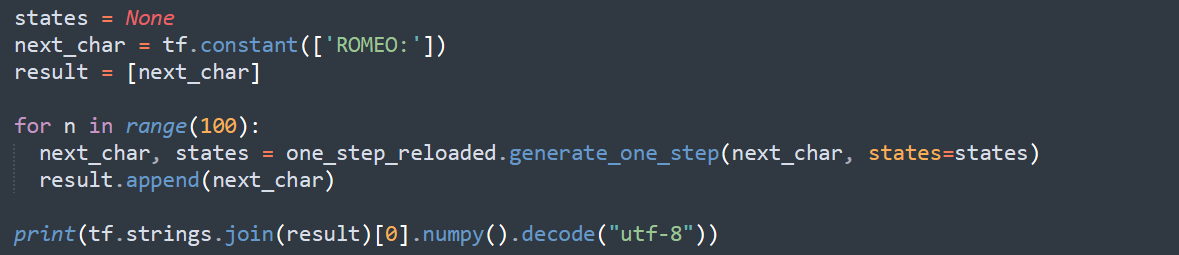
Run time: 4.80143141746521

从以上结果可知，网络已经具备了文本生成的能力，但还是会生成带有语法错误或拼写错误的内容，出现这种现象的原因是欠拟合现象，请思考解决欠拟合问题的可能方案，并做出尝试，给出改进后的文本生成结果。（提示：训练周期是否足够？网络结构是否过于简单？参数设置是否合理？）

完成文本生成任务后，请保存该网络，示例代码如下所示：



除了网络的保存以外，请尝试加载保存的网络，示例代码如下所示：



执行上述代码后，预期得到的结果如下：

ROMEO:

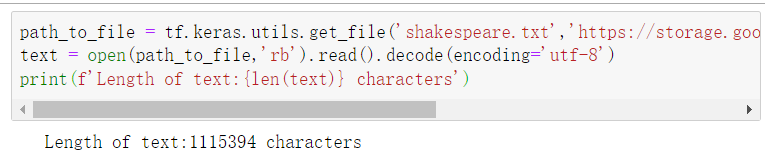
Thou hast hawds charled you as a pair

of the queen's proud with our loves agains:

'Will't plove my

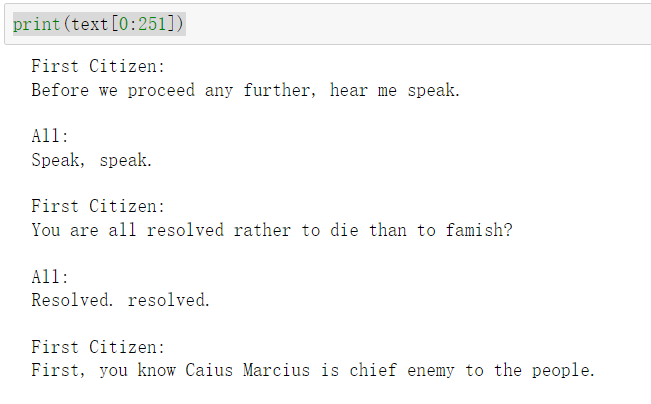
**五、实验结果展示**

1. 在Step 2中，加载数据集后，请展示文本包含的字符数。

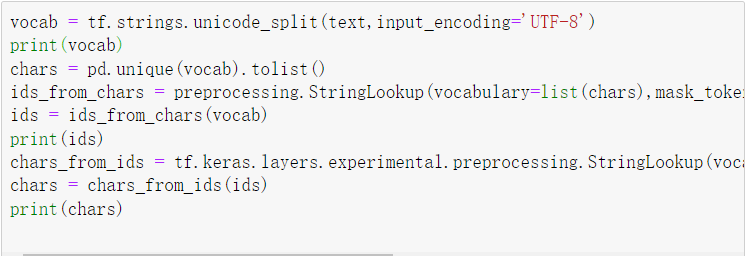


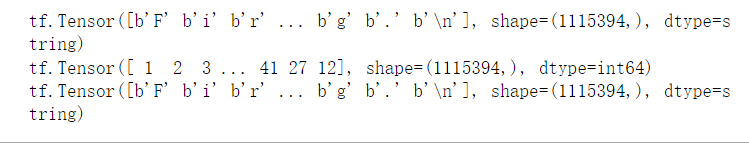
2. 请展示Step 2中输出数据集前250个字符的代码及输出结果。

IMG_256

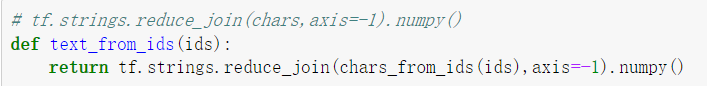


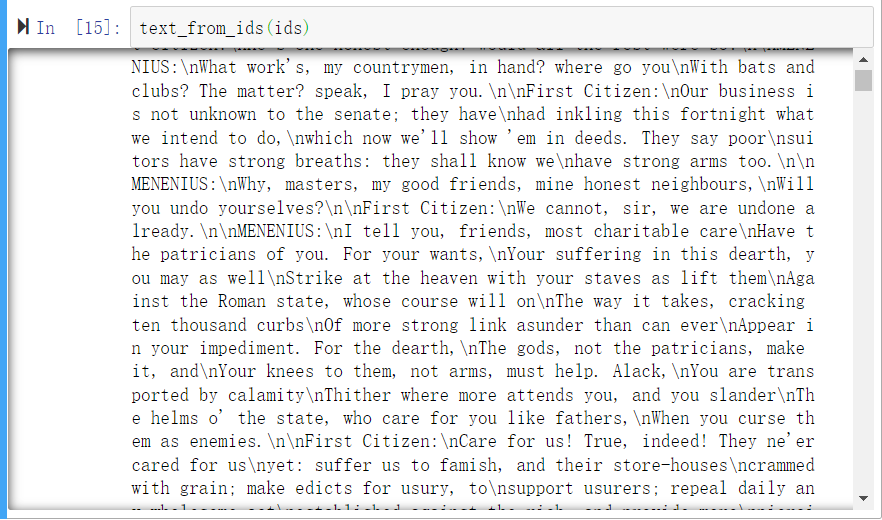
3. Step 3中，要求完成文本数据的拆分和字符索引流转换任务，请展示代码截图及结果。



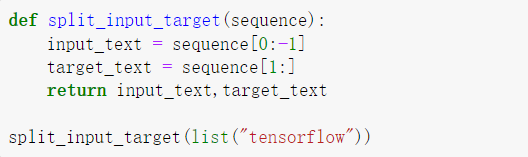


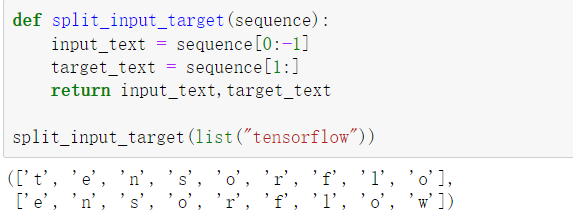
4. Step 3中，要求将字符拼接成字符串，请展示代码截图及拼接结果。



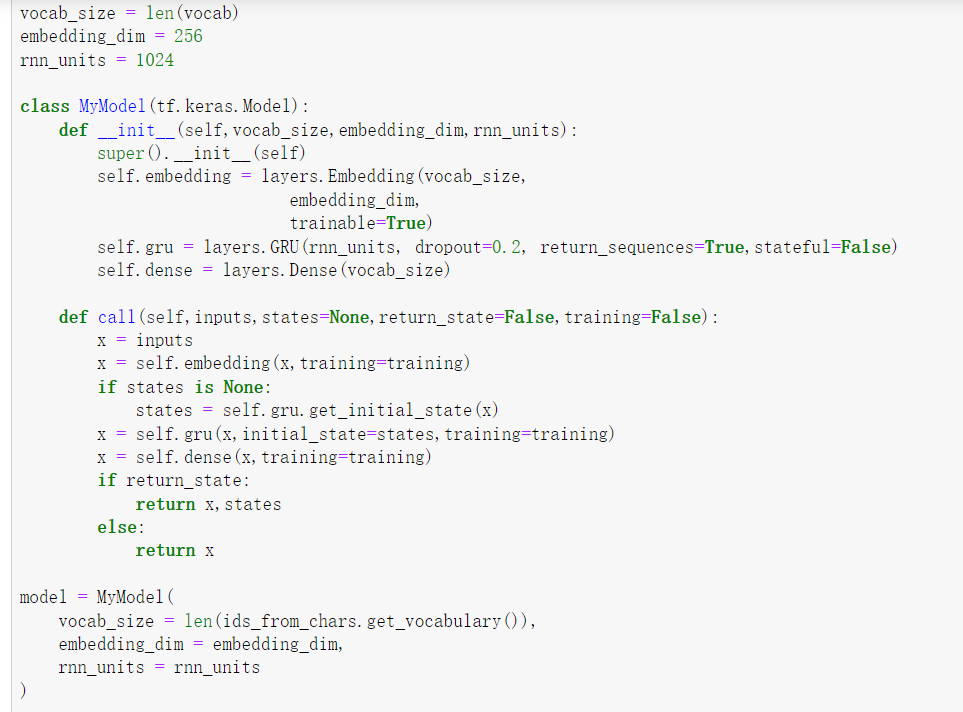


5. Step 3中，要求将输入数据转换为（input，label）形式，请展示代码截图，并以输入数据“Tensorflow”为例，展示测试结果。

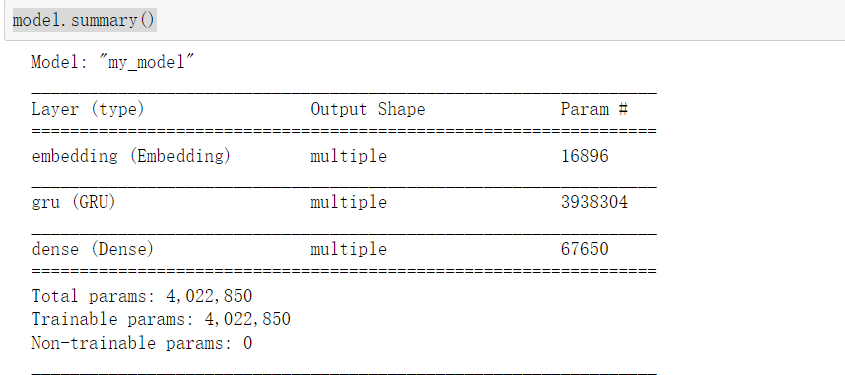




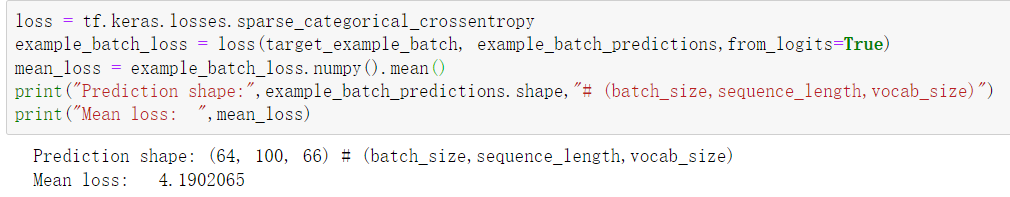
6. Step 4中，要求将代码补充完整，以实现网络的定义和实例化，请展示代码截图。



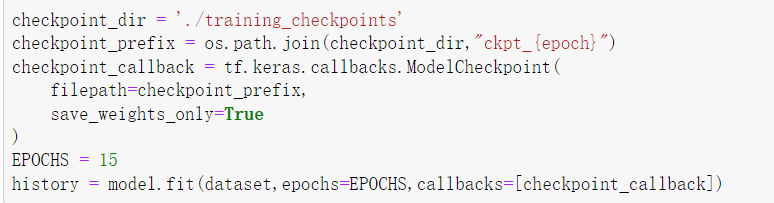
7. Step 4中，请使用model.summary()展示网络的参数，并展示输出结果。



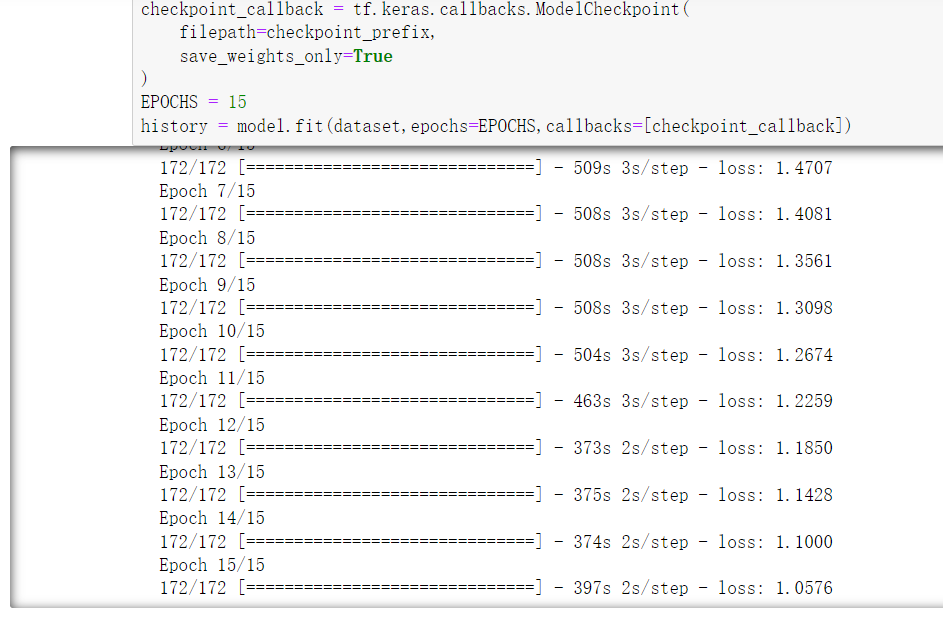
8. Step 5中，要求补全代码，以完成损失函数的定义，请展示代码截图。



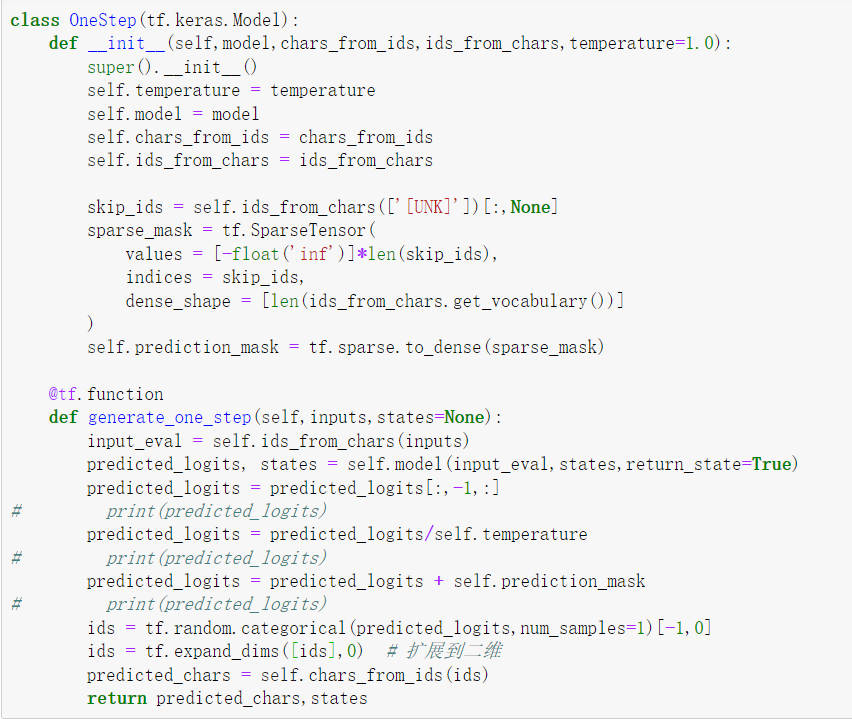
9. Step 5中，要求完成对网络训练过程的相关配置，请展示代码截图。



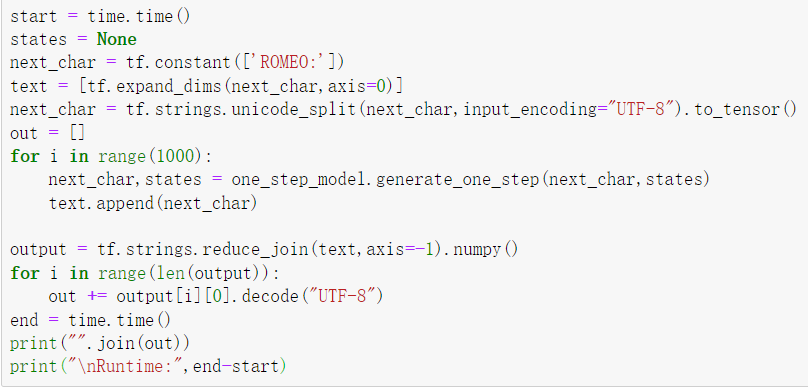
10. Step 5中，要求执行网络的训练，Epoch设置为20，请展示训练的过程（loss的变化）。

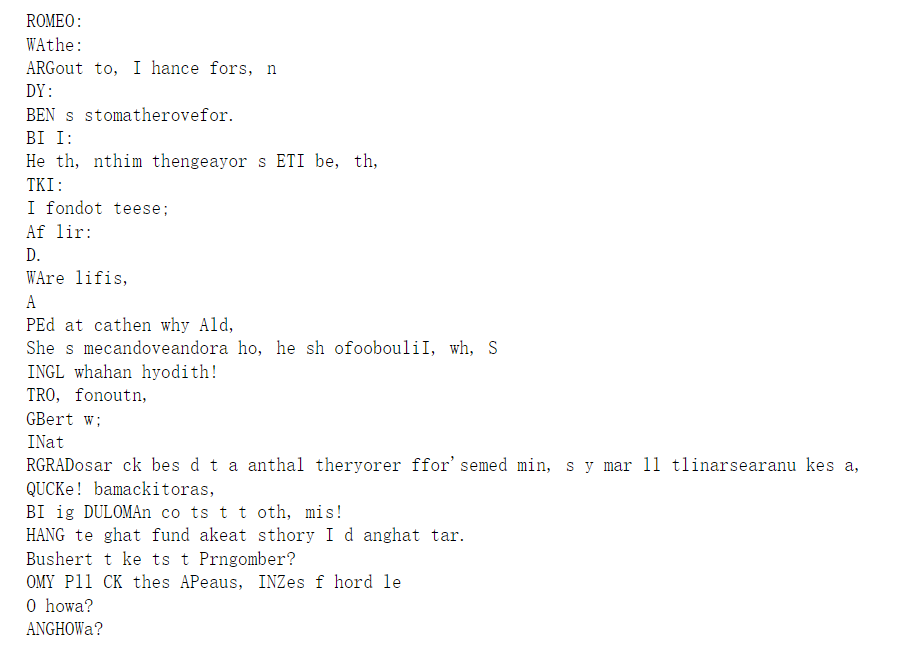


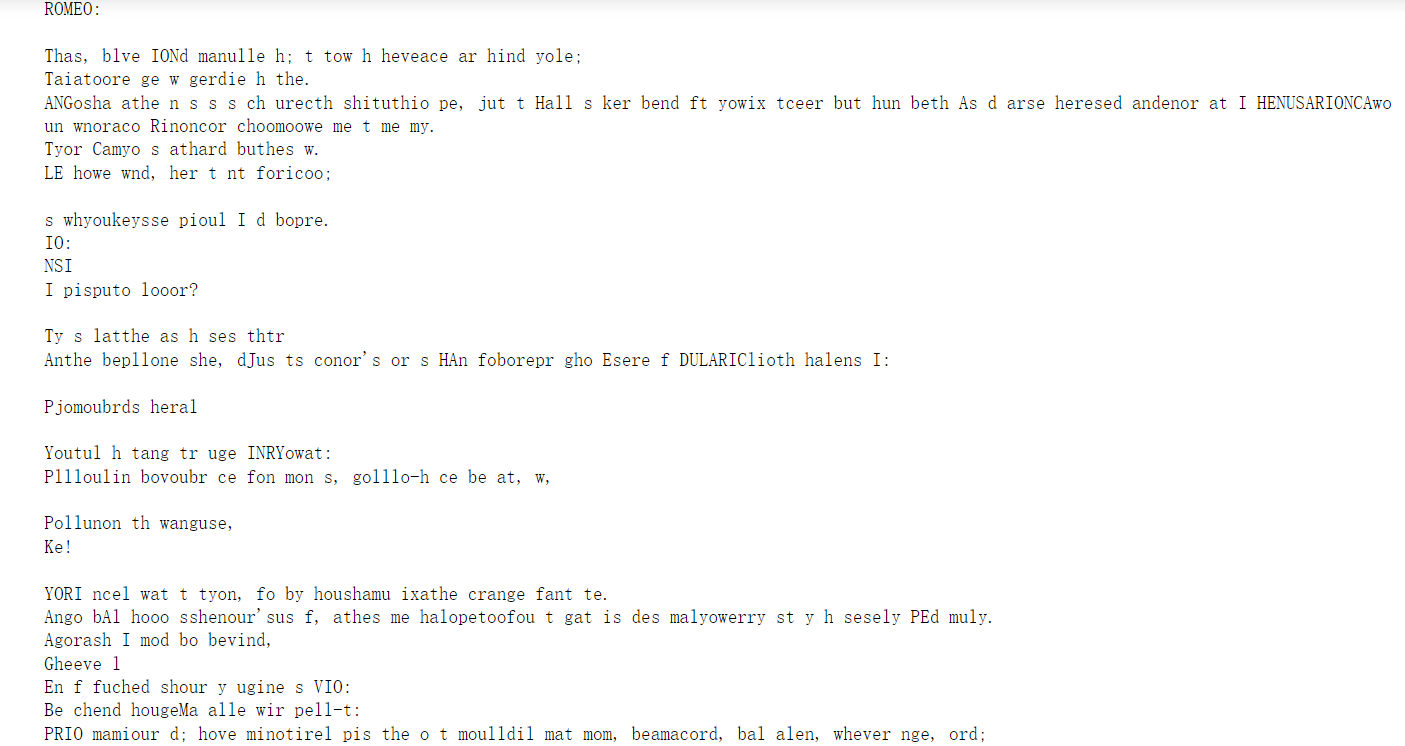
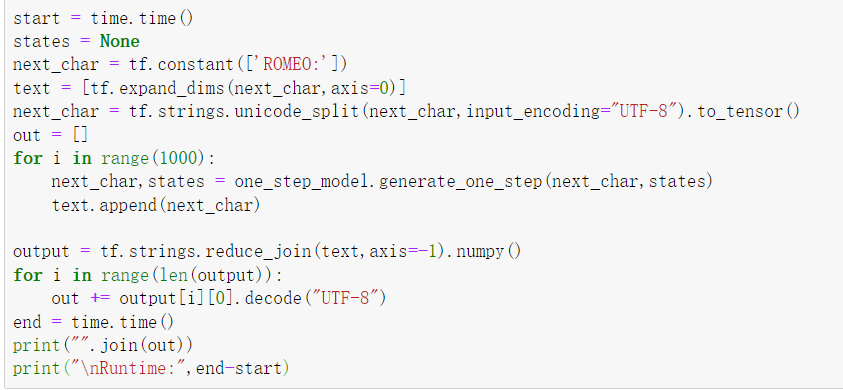
11. Step 6中，要求执行one step训练，请展示代码截图。



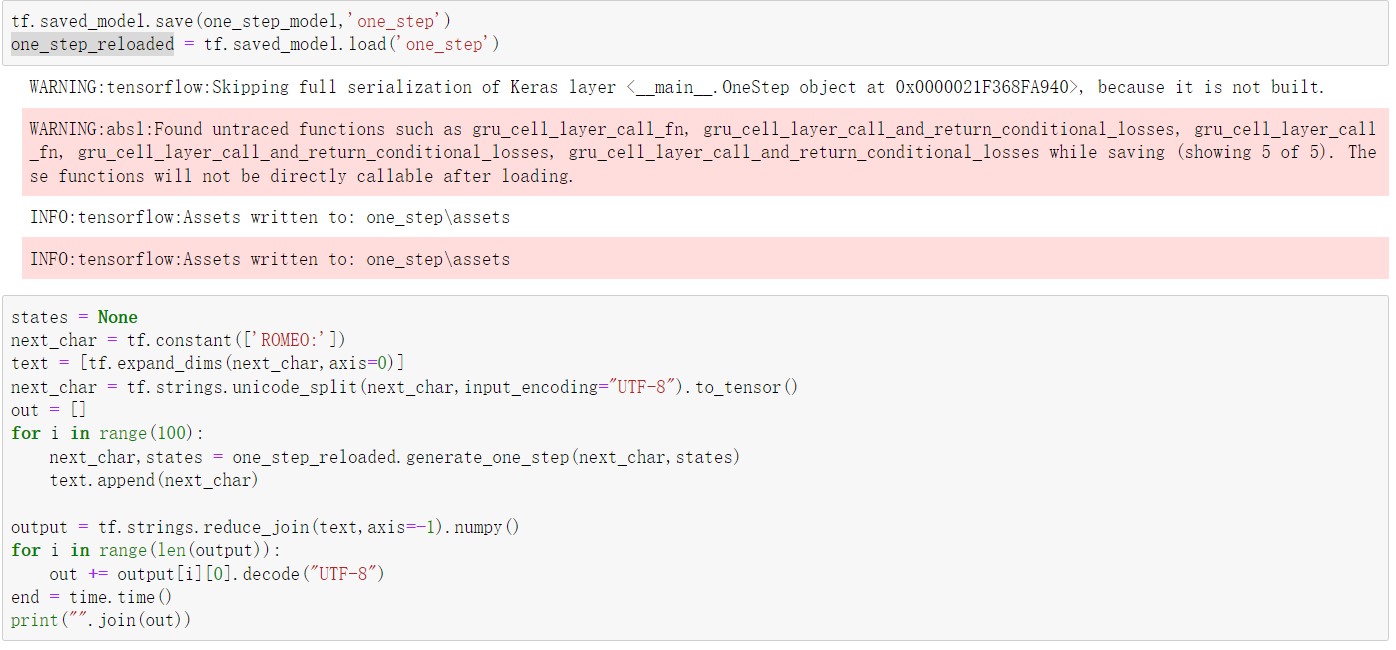
12. Step 6中，要求循环执行one\_step\_model生成文本，请展示代码截图及输出结果。





13. Step 6中，要求解决欠拟合问题，请展示相关解决方案的代码截图，并输出结果。

14. Step 6中，要求保存网络，然后再加载网络，并输出结果。





**六、实验问题回答**

1. 为什么使用sparse categorial crossentropy作为本实验的损失函数？是否可以选择其他损失函数？

Sparse categorial crossentropy用于计算标签和预测之间的交叉熵损失，当有两个或多个标签类别时，使用此交叉熵损失函数，并且标签以整数形式提供。因为这里标签被映射为整数，因此可以选择Sparse categorial crossentropy作为损失函数。

还可以选择categorial crossentropy作为损失函数，在one\_hot表示中提供标签，每个特征都应该有浮点值

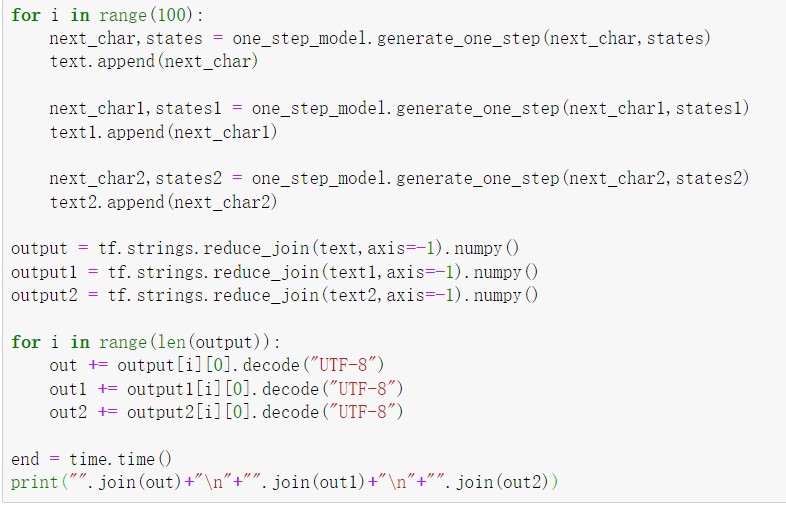
2. 关于欠拟合问题，思考产生该现象的原因，并给出具体的解决方案。

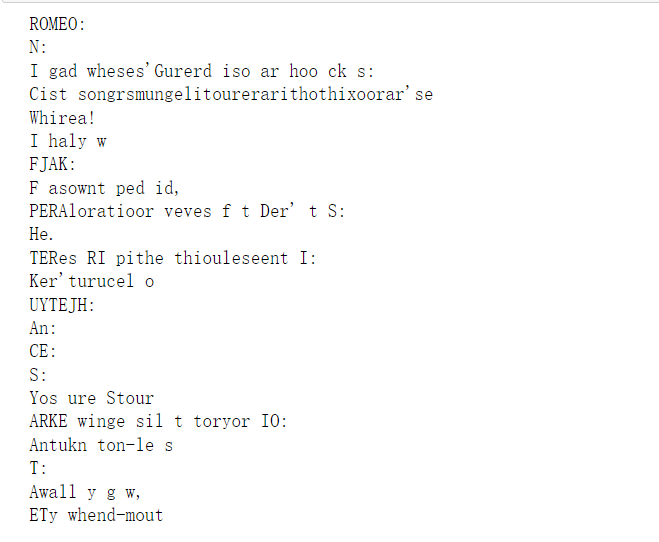
欠拟合现象可能是模型、网络结构过于简单，也可能是训练次数较少不能很好的拟合数据。

这里采用增加训练周期的方法，即增加训练次数，同时可以调整temperature。

3. 在本实验中，网络每次只针对一个起始单词生成一条文本序列。为了加快文本生成的速度，思考是否可以同时给定多个起始单词，一次性生成多条文本序列？请给出相关代码并展示输出结果。







1. 在实验过程中，是否有遇到其他问题？

在保存训练好的模型时有警告，但是可以加载调用

**七、实验心得体会**

本次实验的目的是完成RNN循环神经网络生成文本的功能

首先按照要求完成数据的加载并输出部分数据进行展示，这一步没有问题

然后进行数据的预处理，在这一过程中需要构建两个映射，分别是字符映射为整数和整数映射回字符，按要求先将文本数据拆分成tokens，这里用到unicode\_split方法。然后利用preprocessing.StringLookup层完成映射，这里了解并初步掌握了如何使用preprocessing.StringLookup层以及preprocessing.StringLookup层的一些原理，同样的方法构建整数到字符的映射。最后通过调用tf.strings.reduce\_join来将单个字符拼接，这里初步了解了tf.strings.reduce\_join的原理及用法。

完成预处理后进行训练集和标签的划分，这里通过简单的变换即可完成，在进行完上述步骤后即将文本转化为所需的数据，最后需要打乱其顺序并打包批次。

接着进行网络模型的搭建并训练网络，在训练网络过程中需要保存checkpoint以便后续网络的恢复，因此在训练网络前需配置好tf.keras.callbacks.ModelCheckpoint，完成配置后进行训练，因为这里采用CPU环境因此训练过程极为耗时，因此压缩了一定的次数。

模型训练完成后要进行预测，因为每次的预测师根据上次的输入和过去的状态来判断下次的输出，因此只需实现单次的预测利用循环即可预测长文本，在实现时根据起始字符将其映射为对应的整数，然后利用训练好的model来获得其对应的预测可能和状态，在其所有的可能预测结果中找到可能性最大的将其映射为字符。利用循环给出已知的起始字符，然后不断预测下一个字符将其拼接到一起，由于此时是以二进制字节形式展示，因此需要将其按照UTF-8格式。

转换为字符输出即得到最终的结果。

最后按要求保存训练的模型并加载调用，这里掌握了如何保存已经训练好的模型以及已知模型的调用。

再考虑到同时对多个起始字符生成多个文本，这里减少了单次生成文本的长度，增加了多个列表来存放不同的预测文本，在同一个循环里分别执行添加最后一起输出，即达到与长文本预测同样的效果。

通过本次实验更进一步了解了RNN网络模型的工作原理，对其在编码实现上的一些细节和涉及到的相关函数和方法也有了进一步的了解，但整体感觉还是较为浅层对一些问题的理解不够深刻，碰到部分问题有时会出现无从下手的情况。