人工智能实验4：文本摘要

实验名称：文本摘要	姓名：王溢阳	学号：10204602470
实验时间：2023年12月11日	学院：数据科学与工程学院	实验序号：4
指导老师：李翔	实验成绩：	

实验要求

任务定义

文本摘要是指使用算法从一段较长文本中提取关键信息，生成简短、凝练的摘要。本次实验是文本摘要在医学领域的应用：从详细的病情描述中提取关键信息生成简短清晰的诊断报告。

任务示例

列名	数据类型	示例
description（输入）病情描述	string	左侧顶骨局部骨质缺如；两侧侧脑室旁见点状密度减低。右侧额部颅板下见弧形脑脊液密度影。脑室系统扩大，脑沟、裂、池增宽。中线结构无移位。双侧乳突气化差，内见密度增高。
diagnosis（输出）诊断报告	string	左侧顶骨局部缺如，考虑术后改变；脑内散发缺血灶；右侧额部少量硬膜下积液；双侧乳突炎。

数据集

列名	数据类型	示例
description（输入）病情描述	string	101 47 12 66 74 90 0 411 234 79 175
diagnosis（输出）诊断报告	string	122 83 65 74 2 232 18 44 95

18000条训练数据，自行划分验证集2000条测试数据

任务内容

- 构建seq2seq模型完成本次文本摘要任务
- Encoder/Decoder的选择包括但不限于RNN/ LSTM/ GRU/ Transformer/ BERT/ BART/ T5/ OPT等
- 评估指标的选择：BLEU-4、ROUGE、CIDEr等

实验要求

- 选择一种Encoder-Decoder结构，使用哪种模型作为Encoder或Decoder不受限制，大家可以按照自己的硬件设备自行选择
- 在硬件条件允许的情况下，尽量使用多种不同的模型并进行详细的实验分析；评估指标的使用不受限制，可以仅使用一种或使用多种更加全面的进行结果分析

实验环境

- 由于一轮时间较长且本地运行对电脑CPU要求比较高，本次实验使用恒源云的云主机实例

  以下命令与实验报告内容无关，仅方便个人登录云主机

  ```
  ssh -p 33757 root@i-1.gpushare.com
  ```

	华南 Linux l16d608aff400e01a71 点击设置名称	已关机 显卡空闲可启动	3090-24G * 1 卡 查看详情	系统磁盘 正常 数据盘/hy-tmp 数据保留中	按量付费	最近关机时间 2023-12-11 21:13:23	2024-01-21	-

- 云主机镜像为官方镜像，参数如下

- python3.8
- keras==2.11.0
- keras_preprocessing==1.1.2
- nltk==3.8.1
- numpy==1.23.4
- pandas==1.5.1
- scikit_learn==1.1.3
- tensorflow==2.11.1

实验步骤

主函数

具体代码见code/main.py

> 基于Keras和NLTK库的文本模型的训练和评估

- 导入所需要的库，加载数据，设定随机种子等初始化设置
- 定义命令行参数：运行模式（训练或测试）、训练轮数、模型类型等。
  - 如果运行模式是训练，根据指定的模型类型调用TextModel类中的build_LSTM_model()或build_GRU_model()方法来构建相应的文本模型。
- 数据预处理（详细内容见“数据预处理”），得到最大序列长度、训练数据的源序列和目标序列、测试数据的源序列，以及其他预处理结果。
- 编译模型，定义模型检查点回调函数，以监控验证集上的准确率，并保存最佳模型。
- 使用fit方法训练模型，获取训练过程中的损失和验证集上的损失。（详细内容见“模型训练”）

- 对参考源序列进行预测，得到预测结果
- BLEU-4评估（详细内容见“评估标准”）

数据预处理

具体代码见code/data_processor.py

> 初始化

- 导入库：`numpy`、`tensorflow`、`sklearn`、`keras_preprocessing.sequence`、`os`
- 设置随机种子

> 函数及变量定义

- `data_process`：接收训练和测试数据
  - `train_source_texts`：训练数据的描述文本
  - `train_target_texts`：训练数据的诊断文本
  - `test_source_texts`：测试数据的描述文本
- `train_test_split`：将训练数据划分为训练集和验证集，并将其按比例分割
  - 划分前的训练集：`train_source_texts`和`train_target_texts`
  - 划分后的训练集：`train_source_list`和`train_target_list`
  - 验证集：`ref_source_list`和`ref_target_list`
- `all_texts`：所有文本数据
- `Tokenizer`：创建tokenizer对象，调用`fit_on_texts`方法对`all_texts`进行拟合。
  - 计算tokenizer的词汇表大小
  - `texts_to_sequences`：将训练集、验证集、目标文本序列和测试集的源文本序列转换为整数序列
  - `max_sequence_length`：所有序列中最长的序列长度
  - `pad_sequences`：对训练集、验证集和测试集的源文本序列进行末尾填充，使它们具有相同的长度。对验证集的源文本序列也进行填充，与训练集和测试集保持一致。

模型训练

GRU

具体代码见code/TextModel/build_GRU_model()

> 设计思想

简化LSTM网络的复杂性，保留其主要功能：将输入门、遗忘门和输出门合并为更简单的更新门和输出门。GRU在处理序列数据时表现出色，如文本生成、时间序列预测等任务。

> 计算方法

编码器的更新和重置门（update gate和reset gate）的计算：

- 更新门

$$z_t = \sigma(W_z \cdot x_t + U_z \cdot h_{t-1})$$

- 重置门

$$r_t = \sigma(W_r \cdot x_t + U_r \cdot h_{t-1})$$

x_t:输入序列的当前时间步的向量表示

h_{t-1}: 上一个时间步的隐藏状态

W_z、U_z、W_r、U_r :可学习的权重矩阵

\sigma：Sigmoid函数

隐层状态的更新和生成：

- 计算更新后的候选隐层状态

$$\tilde{h}_t = \tanh(W \cdot x_t + U \cdot (r_t \odot h_{t-1}))$$

- 最终的隐层状态

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$$

W、U: 可学习的权重矩阵

\odot: 逐元素相乘操作

> 代码实现

- `encoder_input`：输入层，形状为 `(self.max_sequence_length,)`，即输入序列的最大长度。
- `encoder_gru`： GRU层，其中 `units=256` 表示 GRU 单元的维度
- `encoder_states`：编码器的输出状态
- `decoder_input`：目标输入层
- `decoder_dense`：全连接层

模型使用了两个 GRU 层，编码器的 GRU 层只返回最后一个时间步的内部状态，解码器的 GRU 层返回每个时间步的输出和最后一个时间步的内部状态。

LSTM

具体代码见code/TextModel/build_GRU_model()

> 设计思想

主要用于处理序列数据。LSTM的主要目标是解决传统RNN中存在的梯度消失和梯度爆炸问题，从而提高模型的记忆能力。

> 计算方法

LSTM的核心结构包括三个门（gate）和一个记忆单元（memory cell）。这三个门分别为输入门、输出门和遗忘门，负责控制信息的输入、输出和遗忘。记忆单元是LSTM的核心部分，用于存储和更新信息。

- 输入门使用当前时间步的输入x_t和上一个时间步的隐藏状态h_{t-1}，计算输入门的输出i_t：

$$i_t = \sigma(W_i \cdot x_t + U_i \cdot h_{t-1} + b_i)$$

- 遗忘门使用当前时间步的输入x_t和上一个时间步的隐藏状态h_{t-1}，计算遗忘门的输出f_t：

$$f_t = \sigma(W_f \cdot x_t + U_f \cdot h_{t-1} + b_f)$$

- 候选记忆单元使用当前时间步的输入x_t和上一个时间步的隐藏状态h_{t-1}，计算候选记忆单元的输出\tilde{C}__t：

$$\tilde{C}_t = \tanh(W_C \cdot x_t + U_C \cdot h_{t-1} + b_C)$$

- 细胞状态使用输入门、遗忘门和候选记忆单元，更新当前时间步的细胞状态C_t：

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

- 输出门使用当前时间步的输入x_t、上一个时间步的隐藏状态h_{t-1}和当前时间步的细胞状态C_t，计算输出门的输出o_t：

$$o_t = \sigma(W_o \cdot x_t + U_o \cdot h_{t-1} + b_o)$$

- 隐层状态使用输出门和通过细胞状态进行激活函数（通常是 tanh 函数）的当前细胞状态，计算当前时间步的隐层状态h_t：

$$h_t = o_t \cdot \tanh(C_t)$$

> 代码实现

- 输入层 `encoder_input`：输入序列的最大长度
- LSTM 层 `encoder_lstm`：将接收嵌入后的输入序列，返回 LSTM 层的输出、当前时刻的内部状态 `state_h` 和记忆状态 `state_c`。
- 目标输入层 `decoder_input`：形状与编码器输入相同
- LSTM 层 `decoder_lstm`：设置 `return_sequences=True` 获取每个时间步的输出，以及 `return_state=True` 来获取最终的内部状态
- 全连接层 `decoder_dense`：用 softmax 激活函数将输出映射到概率分布上。

评估标准

BLEU_4

具体代码见code/main.py/bleu_score

> 总体思想就是准确率，假如给定标准译文reference，神经网络生成的句子是candidate，句子长度为n，candidate中有m个单词出现在reference，m/n就是bleu的1-gram的计算公式。

$$bleu_n = \frac{\sum_{c \in candidates} \sum_{n-gram \in c} Count_{clip}(n-gram)}{\sum_{c' \in candidates} \sum_{n-gram' \in c'} Count(n-gram')}$$

- 分子
  - 神经网络生成的句子是candidate，给定的标准译文是reference。
  - 第一个求和符号:所有的candidate，计算时可能有多个句子
  - 第二个求和符号：一条candidate中所有的n−gram
  - 所以整个分子就是在**给定的candidate中有多少个n-gram词语出现在reference中。**
- 分母释义
  - 前两个求和符号和分子中的含义一样，Count(n-gram')表示n−gram'在candidate中的个数
  - 分母是获得**所有的candidate中n-gram的个数。**

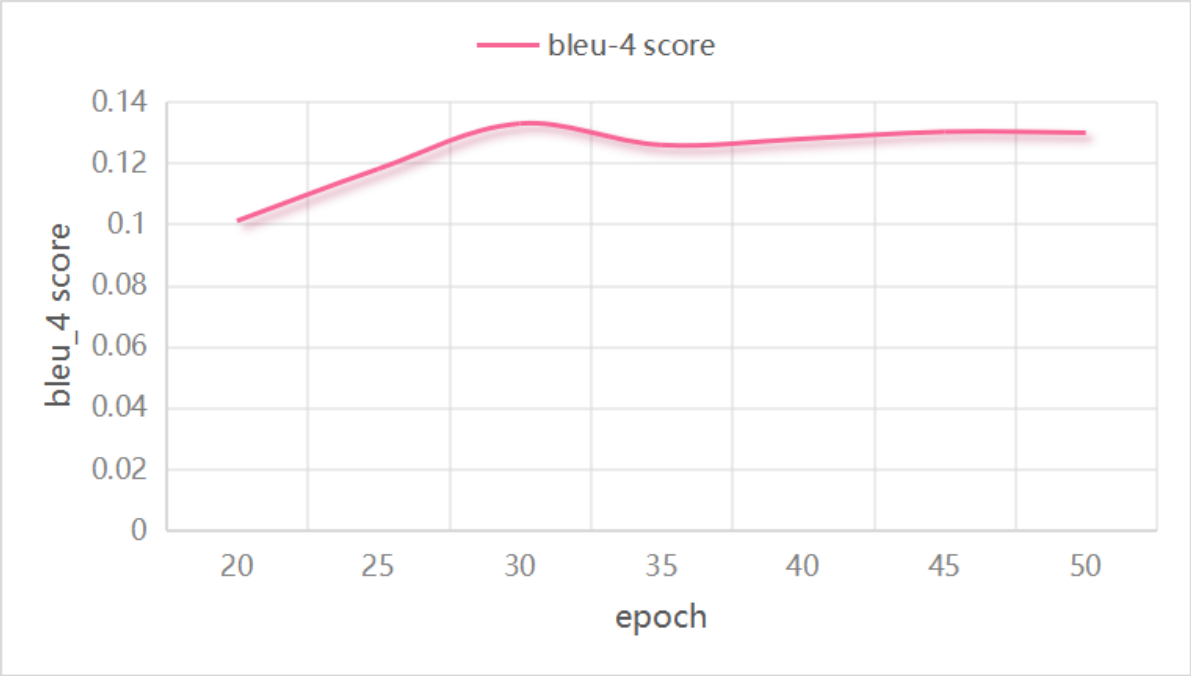
```
# references:参考文本列表
# candidates:候选文本列表
# weights:权重元组，用于给不同长度的n-grams分配不同的权重,四个n-grams的权重都是0.25。
# bleu.corpus_bleu():计算候选文本与参考文本之间的BLEU分数

re = ref_target_list.tolist()
references = [[text.split()] for text in re]
candidates = [text.split() for text in train_predictions['diagnosis']]

bleu_score = bleu.corpus_bleu(references, candidates, weights=(0.25, 0.25, 0.25,
0.25))
print("BLEU-4 评估指标：", bleu_score)
```

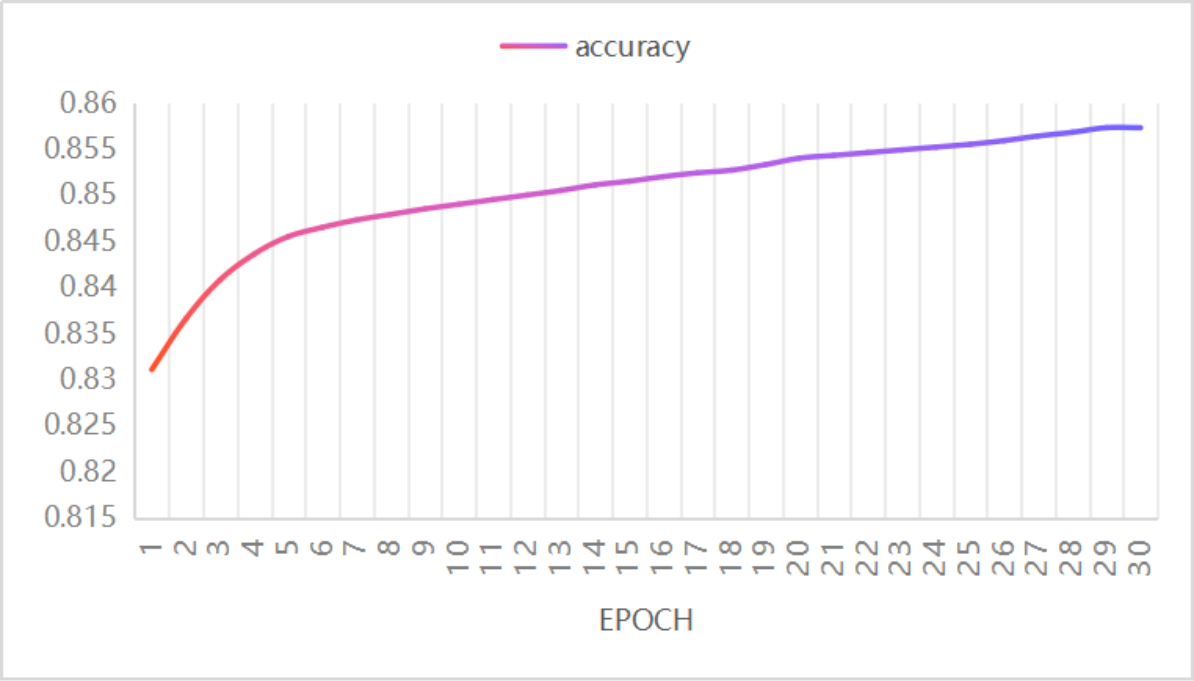
实验结果

GRU

epoch=30 bleu=0.1328

大约在30轮左右出现accuracy峰值，30轮左右出现bleu_4 最大值



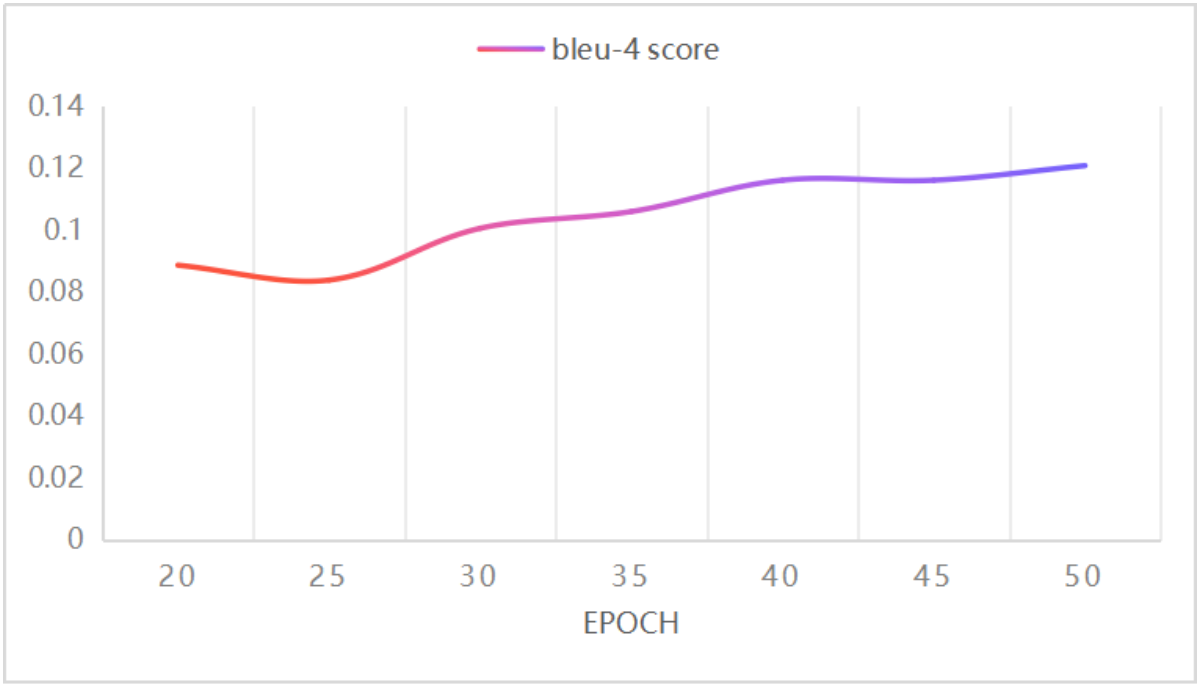


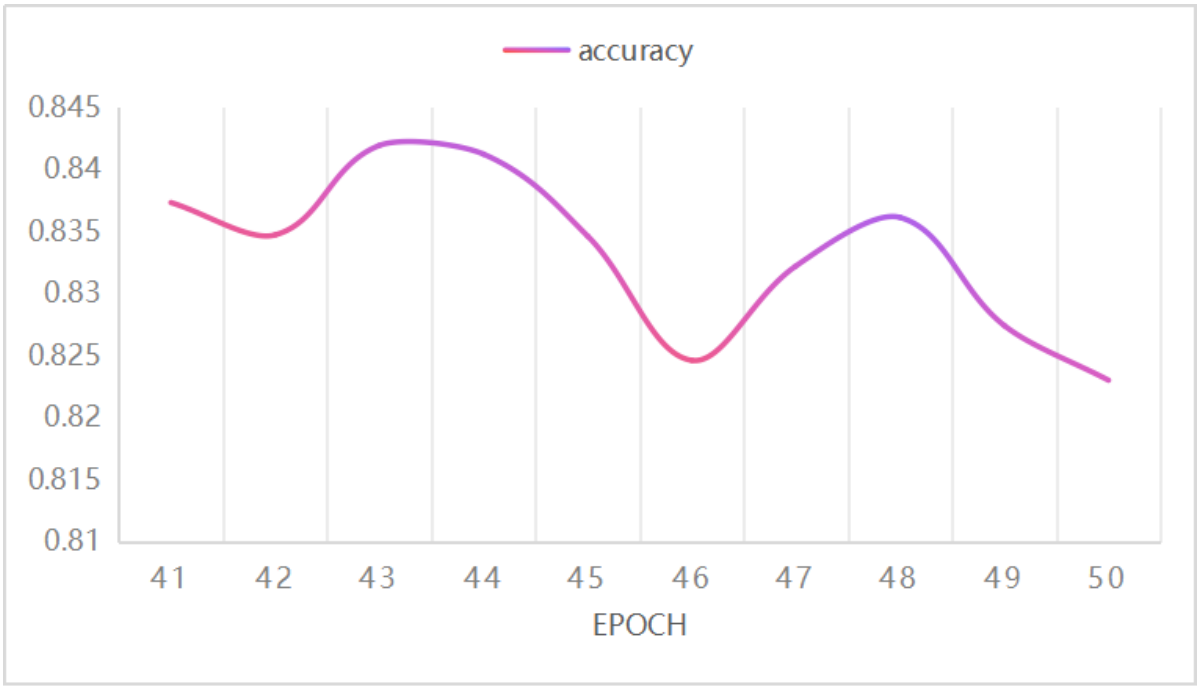

```
Epoch 41/50
180/180 [==============================] - 14s 78ms/step - loss: 0.6385 - accur
acy: 0.8640 - val_loss: 1.0524 - val_accuracy: 0.8373
Epoch 42/50
180/180 [==============================] - 14s 77ms/step - loss: 0.6484 - accur
acy: 0.8632 - val_loss: 1.0240 - val_accuracy: 0.8347
Epoch 43/50
180/180 [==============================] - 14s 77ms/step - loss: 0.6450 - accur
acy: 0.8637 - val_loss: 1.0381 - val_accuracy: 0.8419
Epoch 44/50
180/180 [==============================] - 14s 77ms/step - loss: 0.6290 - accur
acy: 0.8651 - val_loss: 1.0773 - val_accuracy: 0.8412
Epoch 45/50
180/180 [==============================] - 14s 79ms/step - loss: 0.6152 - accur
acy: 0.8664 - val_loss: 1.0721 - val_accuracy: 0.8346
Epoch 46/50
180/180 [==============================] - 14s 77ms/step - loss: 0.6119 - accur
acy: 0.8670 - val_loss: 1.0903 - val_accuracy: 0.8246
Epoch 47/50
180/180 [==============================] - 14s 78ms/step - loss: 0.6150 - accur
acy: 0.8667 - val_loss: 1.0814 - val_accuracy: 0.8322
Epoch 48/50
180/180 [==============================] - 14s 78ms/step - loss: 0.6143 - accur
acy: 0.8665 - val_loss: 1.0817 - val_accuracy: 0.8361
Epoch 49/50
180/180 [==============================] - 14s 77ms/step - loss: 0.6061 - accur
acy: 0.8675 - val_loss: 1.0832 - val_accuracy: 0.8274
Epoch 50/50
180/180 [==============================] - 14s 77ms/step - loss: 0.5968 - accur
acy: 0.8685 - val_loss: 1.1088 - val_accuracy: 0.8230
113/113 [==============================] - 2s 16ms/step
BLEU-4 评估指标:  0.13141371379555436
```

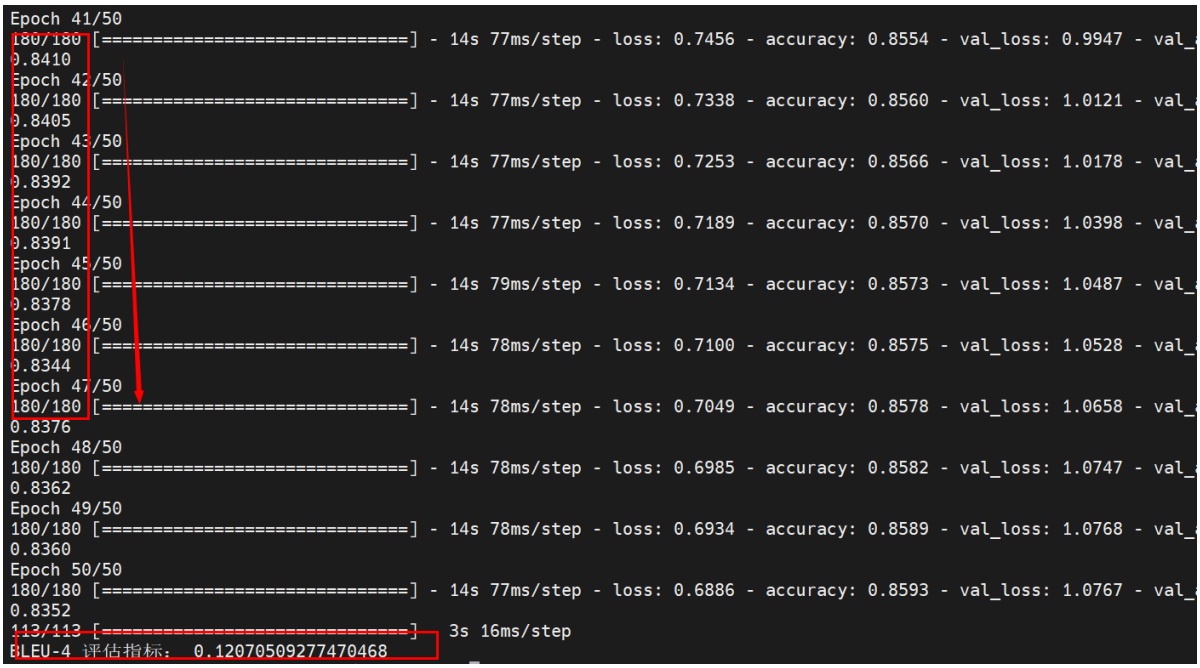
LSTM

epoch=50 bleu=0.12

大约在43轮出现accuracy峰值，在50轮左右出现bleu_4最大值


bleu-4 score
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
20
25
30
35
40
45
50
EPOCH


accuracy
0.845
0.84
0.835
0.83
0.825
0.82
0.815
0.81
41
42
43
44
45
46
47
48
49
50
EPOCH



遇到的问题和总结

> bleu 值很低

- 模型可能没有足够的参考信息来生成高质量的摘要
- 更换编码解码器
- 模型倾向于生成过短或过长的摘要，或者生成的摘要与参考答案在词汇和结构上不一致

> encoder_embedding = layers.Embedding(lens, 256)(encoder_input)
>
> lens = len(self.tokenizer.word_index) + 1

- +1: 留出一个位置表示unknown word