机器翻译实验报告

陈远腾 2020k8009929041

一、环境搭建

1.容器环境准备

由于我们希望训练基于transformer的机器翻译模型，对显存要求较高，本机的GPU无法满足，因此我在autodl平台上租赁了4 x RTX3090 的服务器容器，作为本次实验的平台。

2.依赖库与相关工具安装

(1) fairseq

```
git clone https://github.com/pytorch/fairseq
cd fairseq
pip install --editable ./
```

Fairseq是一个用于序列到序列（Seq2Seq）建模和自然语言处理的开源框架。它由Facebook AI Research开发，旨在为研究人员和开发人员提供一个灵活的、可扩展的平台，用于开发、训练和部署复杂的Seq2Seq模型。

Fairseq支持多种Seq2Seq任务，包括机器翻译、语音识别、文本摘要等。它提供了基于PyTorch的模型和训练工具，可以方便地进行模型开发和训练，并支持分布式训练。此外，Fairseq还提供了一系列预训练模型和预处理工具，可以帮助用户快速构建自然语言处理模型。

> 在本次实验中，我利用fairseq框架与TED数据集训练基于transformer的机器翻译模型。

(2) Moses

Moses是一个用于机器翻译的开源工具包，它提供了一系列用于自然语言处理的工具，包括分词、词性标注、短语提取、语言模型训练等功能，可以用于构建机器翻译系统和其他自然语言处理应用。

Moses的核心组件包括：

- 分词工具：用于将原始文本分割成单词或子词。
- 短语提取工具：用于从平行语料库中提取短语对，用于训练翻译模型。
- 语言模型训练工具：用于训练语言模型，提高机器翻译的准确性。
- 翻译模型训练工具：用于从短语对中训练翻译模型，实现机器翻译。
- 解码器：用于将源语言文本翻译成目标语言文本。

```
git clone https://github.com/moses-smt/mosesdecoder.git
```

> 在本次实验中，我们使用了Moses中提供的一些预处理脚本，如：tokenisation，truecasing，cleaning。

(3) subword-nmt

Subword-nmt是一个用于自然语言处理的开源工具，它提供了一种基于子词（subword）的方法，用于解决单词切分和未登录词等问题，可以用于机器翻译、文本分类、语音识别等各种自然语言处理任务中。

Subword-nmt的主要功能是将单词切分成子词，从而形成一种基于子词的文本表示方式。相比于传统的基于单词的表示方式，基于子词的表示方式可以更好地处理未登录词、低频词等问题，从而提高模型的性能和泛化能力。

```
git clone https://github.com/rsennrich/subword-nmt.git
```

> 在本次实验中，我们使用subword-nmt中提供的BPE算法生成子词的预处理脚本对我们的数据集进行处理。

（4）jieba

jieba这个python库经过前两次NLP大作业，我们已经非常熟悉，主要用于中文分词。

```
pip install jieba
```

3.文件结构组织

环境配置完毕后，根目录下应当已经出现fairseq，mosesdecoder和subword-nmt三个文件夹。接下来我们要准备我们存放原始数据集、后续存放权重文件和其他脚本的文件夹：

```
mkdir nmt && cd nmt
mkdir data && mkdir models && mkdir utils
cd data && mkdir TED
```

- 将课程提供的TED数据集压缩包解压后得到的TED2020.en-zh_cn.en和TED2020.en-zh_cn.zh_cn均copy到data/TED目录下，并分别rename为raw.en和raw.zh。
- models目录下将存放我们后续训练好的模型。
- utils目录下存放了我们用于划分train,test,val集的脚本split.py，后续我们将介绍该脚本。

二、实验过程

1.路径准备

由于我们已经组织好了文件目录结构，为了后续运行脚本方便，我们提前定义好路径如下：(具体内容见path.sh)

```
src=en
tgt=zh
SCRIPTS=/root/autodl-tmp/NLP_hw3/mosesdecoder/scripts
TOKENIZER=${SCRIPTS}/tokenizer/tokenizer.perl
DETOKENIZER=${SCRIPTS}/tokenizer/detokenizer.perl
LC=${SCRIPTS}/tokenizer/lowercase.perl
TRAIN_TC=${SCRIPTS}/recaser/train-truecaser.perl
TC=${SCRIPTS}/recaser/truecase.perl
DETC=${SCRIPTS}/recaser/detruecase.perl
NORM_PUNC=${SCRIPTS}/tokenizer/normalize-punctuation.perl
CLEAN=${SCRIPTS}/training/clean-corpus-n.perl
BPEROOT=/root/autodl-tmp/NLP_hw3/subword-nmt/subword_nmt
MULTI_BLEU=${SCRIPTS}/generic/multi-bleu.perl
MTEVAL_V14=${SCRIPTS}/generic/mteval-v14.pl
data_dir=/root/autodl-tmp/NLP_hw3/nmt/data/TED
model_dir=/root/autodl-tmp/NLP_hw3/nmt/models/TED
utils=/root/autodl-tmp/NLP_hw3/nmt/utils
```

这里我们希望训练英转中的翻译模型，因此src = en && tgt = zh。

/root/autodl-tmp/NLP_hw3/是租赁的服务器上本次作业的根目录。

2.数据预处理

首先观察原始数据集raw.en和raw.zh：

raw.en:

nmt > data > TED > raw.en

```
Thank you so much, Chris.
And it's truly a great honor to have the opportunity to come to this stage twice; I'm extremely grateful.
I have been blown away by this conference, and I want to thank all of you for the many nice comments about what I had to say the other night.
And I say that sincerely, partly because (Mock sob) I need that.
(Laughter) Put yourselves in my position.
(Laughter) I flew on Air Force Two for eight years.
(Laughter) Now I have to take off my shoes or boots to get on an airplane!
(Laughter) (Applause) I'll tell you one quick story to illustrate what that's been like for me.
(Laughter) It's a true story -- every bit of this is true.
Soon after Tipper and I left the -- (Mock sob) White House -- (Laughter) we were driving from our home in Nashville to a little farm we have 50 miles
Driving ourselves.
```

raw.zh:

nmt > data > TED > raw.zh

```
非常谢谢，克里斯。的确非常荣幸
能有第二次站在这个台上的机会，我真是非常感激。
这个会议真是让我感到惊叹不已，我还要谢谢你们留下的 关于我上次演讲的精彩评论
我是非常真诚的，部分原因是因为--(模拟呜咽)--我的确非常需要！ (笑声)
你设身处地为我想想！
我坐了8年的空军二号。
不过现在上飞机前我则要脱掉我的鞋子
（笑声） （掌声） 我给你讲个小故事来形容我现在是什么样的
这是个真实的故事--每一点都是如此。
就在我和Tipper（戈尔妻子）离开白宫稍后不久 --（模拟呜咽） --（笑声） 我们驱车从纳什维尔的家开到 我们在东边50里外的一个小农场--
自己开车
```

> 可以看到，原始数据集并没有对文本做分词等处理，因此我们在训练前首先需要对数据进行预处理。

（1）标点符号标准化

观察整个数据集可以发现，其中中英文语料中的逗号、双引号等标点符号是不相同的，为了同一标点符号，我们使用Mosed库中提供的/tokenizer/normalize-punctuation.perl脚本进行标点符号标准化处理：

```
perl ${NORM_PUNC} -l en < ${data_dir}/raw.en > ${data_dir}/norm.en
perl ${NORM_PUNC} -l zh < ${data_dir}/raw.zh > ${data_dir}/norm.zh
```

经过标准符号标准化处理后，得到norm.en和norm.zh，以下以norm.zh为例展示转换效果：

(转换前)raw.zh:

```
所以在果蝇驾驶舱中的飞行员， “执行者”， 要知道哪一种气味存在 只要看看哪颗蓝色发光二极管亮起来就行了。
“执行者”得到这个讯息之后的行为 取决于它的政策， 这些政策都是根据 气味检测器 与运动神经之间关联的强度来储存的 这驱动了果蝇的逃亡行为。
如果关联性弱，运动神经会保持关上 那只果蝇会继续前进。
如果关联性强，运动神经就会启动 那只果蝇就会作一个转身。
现在试想这样一个情况 就是当运动神经保持关上时， 那只果蝇继续前行 它就会遭受一些痛苦的后果 例如遭到电击。
在这样的情况下， 我们可以预期“批评家”会发表意见 并告诉“执行者” 要它改变它的政策。
我们人工地制造了这样的一个情境 以一束光来启动“批评家”。
```

(转换后)norm.zh:

```
所以在果蝇驾驶舱中的飞行员， "执行者"， 要知道哪一种气味存在 只要看看哪颗蓝色发光二极管亮起来就行了。
"执行者"得到这个讯息之后的行为 取决于它的政策， 这些政策都是根据 气味检测器 与运动神经之间关联的强度来储存的 这驱动了果蝇的逃亡行为。
如果关联性弱，运动神经会保持关上 那只果蝇会继续前进。
如果关联性强，运动神经就会启动 那只果蝇就会作一个转身。
现在试想这样一个情况 就是当运动神经保持关上时， 那只果蝇继续前行 它就会遭受一些痛苦的后果 例如遭到电击。
在这样的情况下， 我们可以预期"批评家"会发表意见 并告诉"执行者" 要它改变它的政策。
我们人工地制造了这样的一个情境 以一束光来启动"批评家"。
```

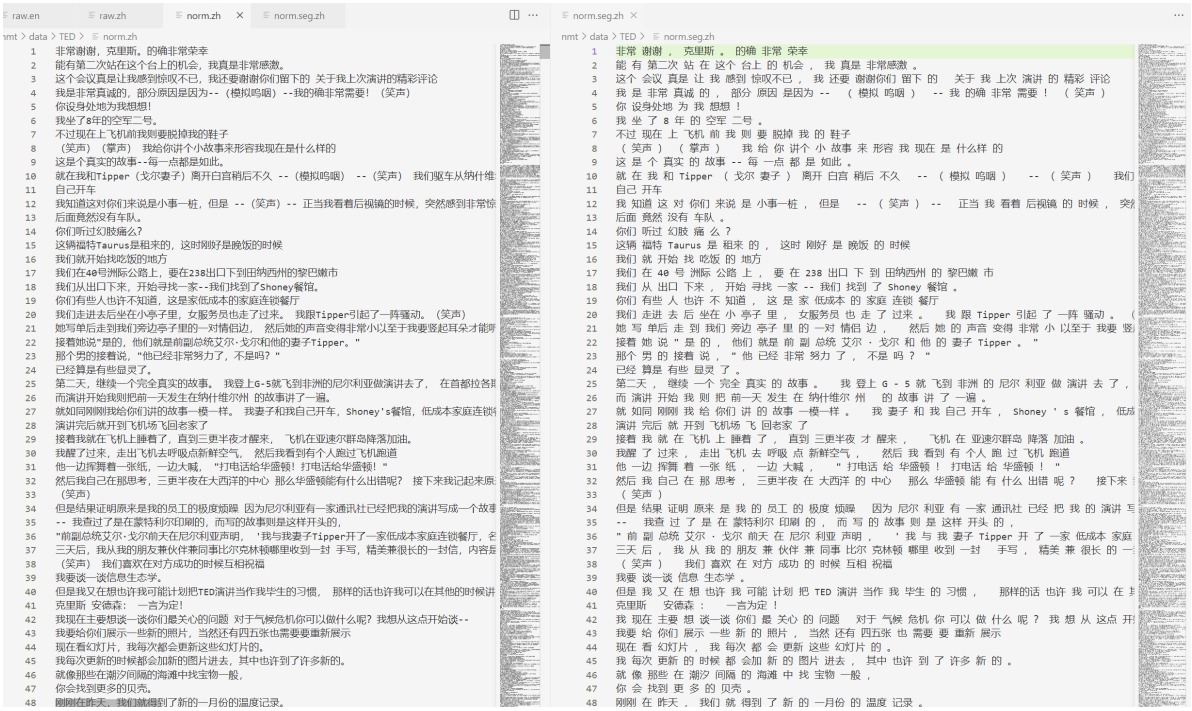
可以看到红框处，已经将中文双引号转换为英文双引号。

（2）分词

上面我们已经观察到，原始数据中并没有对中文语料进行分词，因此这里我们使用jieba库对norm.zh进行处理得到分词结果norm.seg.zh：

```
python -m jieba -d " " ${data_dir}/norm.zh > ${data_dir}/norm.seg.zh
```

分词前后效果对比如下：（左为分词前，右为分词后）



（3）标记化处理

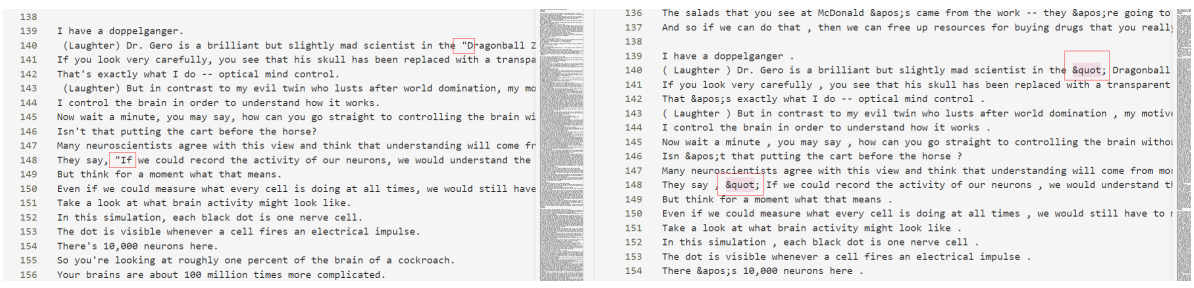
在中文分词后，我对上述处理后的双语文件(norm.en, norm.seg.zh)进行标记化处理，具体来说进行了以下处理：

- 将英文单词与标点符号用空格分开。
- 将多个连续空格简化为一个空格 3.将很多符号替换成转义字符，如：把 " 替换成&quot，把can't替换为

  can &apos t。

```
${TOKENIZER} -l en < ${data_dir}/norm.en > ${data_dir}/norm.tok.en
${TOKENIZER} -l zh < ${data_dir}/norm.seg.zh > ${data_dir}/norm.seg.tok.zh
```

标记化处理前后效果对比如下：（左为分词前，右为分词后，以英文语料为例）



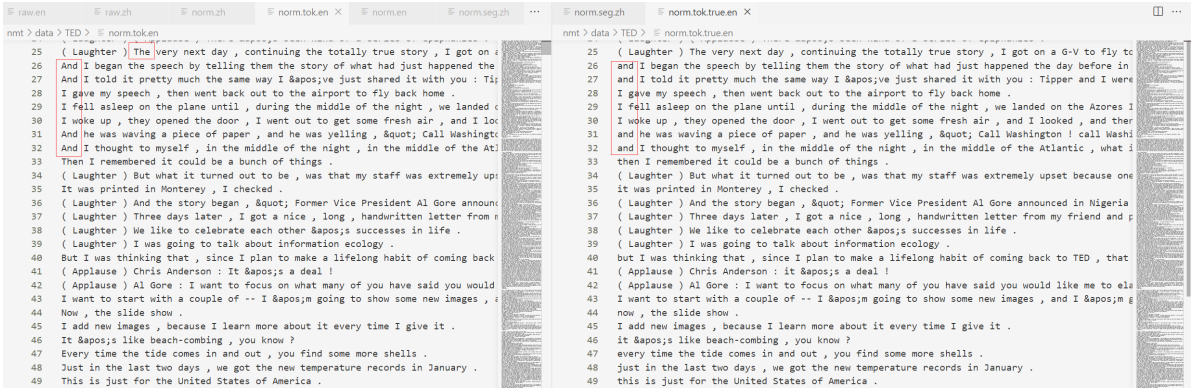
（4）truecase大小写处理

Truecase是指对文本中的单词进行正确的大小写转换。在自然语言文本中，单词可能以大写字母、小写字母或混合大小写的形式出现。Truecase的目的是将单词转换为正确的大小写形式，以提高自然语言处理系统的性能和可读性。Truecase在数据预处理过程中中非常重要，因为单词的大小写形式往往包含了语义和上下文信息。例如，在英语中，单词"I"和"i"的含义是完全不同的。如果在处理自然语言文本时忽略了单词的大小写形式，可能会导致信息丢失和错误的处理结果。

在本次实验中，我们使用Moses库中提供的/recaser/train-truecaser.perl脚本，在我们的数据集上学习适合TED数据集的大小写转换方式，并将训练好的模型直接应用在TED数据集上，完成大小写转换。

```
${TRAIN_TC} --model ${model_dir}/truecase-model.en --corpus
${data_dir}/norm.tok.en
${TC} --model ${model_dir}/truecase-model.en < ${data_dir}/norm.tok.en >
${data_dir}/norm.tok.true.en
```

大小写处理前后效果对比如下：（左为处理前，右为处理后，以英文语料为例）



可以看到红框处，真正属于句首的The和I经处理后仍然保留为大写，而实际并非句首的And，经处理后转换为了小写and，说明我们的转换时有效的。

（5）BPE处理+过滤空白行

对上面的结果进行BPE子词处理，将最频繁出现的字符对合并成一个新的子词。BPE算法通过反复合并字符对来生成子词，从而可以自动学习出适合当前语料库的子词。BPE算法帮助处理未登录词、低频词等问题，提高模型的性能和泛化能力。

BPE算法的主要流程如下：

1. 初始化：将每个字符视为一个初始的子词。
2. 统计字符对频率：统计相邻字符对（或字符与空格的组合）的出现频率。
3. 合并字符对：将频率最高的字符对合并成一个新的子词，并将其添加到词表中。
4. 更新词表：更新词表中的子词出现频率，重新统计相邻字符对的频率。
5. 迭代：重复步骤3和4，直到达到指定的子词数量为止。

在本次实验中，我们直接使用subword-nmt库中的脚本learn_joint_bpe_and_vocab.py和apply_bpe.py完成BPE处理：

```
python ${BPEROOT}/learn_joint_bpe_and_vocab.py --input
${data_dir}/norm.tok.true.en  -s 32000 -o ${model_dir}/bpecode.en --write-
vocabulary ${model_dir}/voc.en
python ${BPEROOT}/apply_bpe.py -c ${model_dir}/bpecode.en --vocabulary
${model_dir}/voc.en < ${data_dir}/norm.tok.true.en >
${data_dir}/norm.tok.true.bpe.en

python ${BPEROOT}/learn_joint_bpe_and_vocab.py --input
${data_dir}/norm.seg.tok.zh  -s 32000 -o ${model_dir}/bpecode.zh --write-
vocabulary ${model_dir}/voc.zh
python ${BPEROOT}/apply_bpe.py -c ${model_dir}/bpecode.zh --vocabulary
${model_dir}/voc.zh < ${data_dir}/norm.seg.tok.zh >
${data_dir}/norm.seg.tok.bpe.zh
```

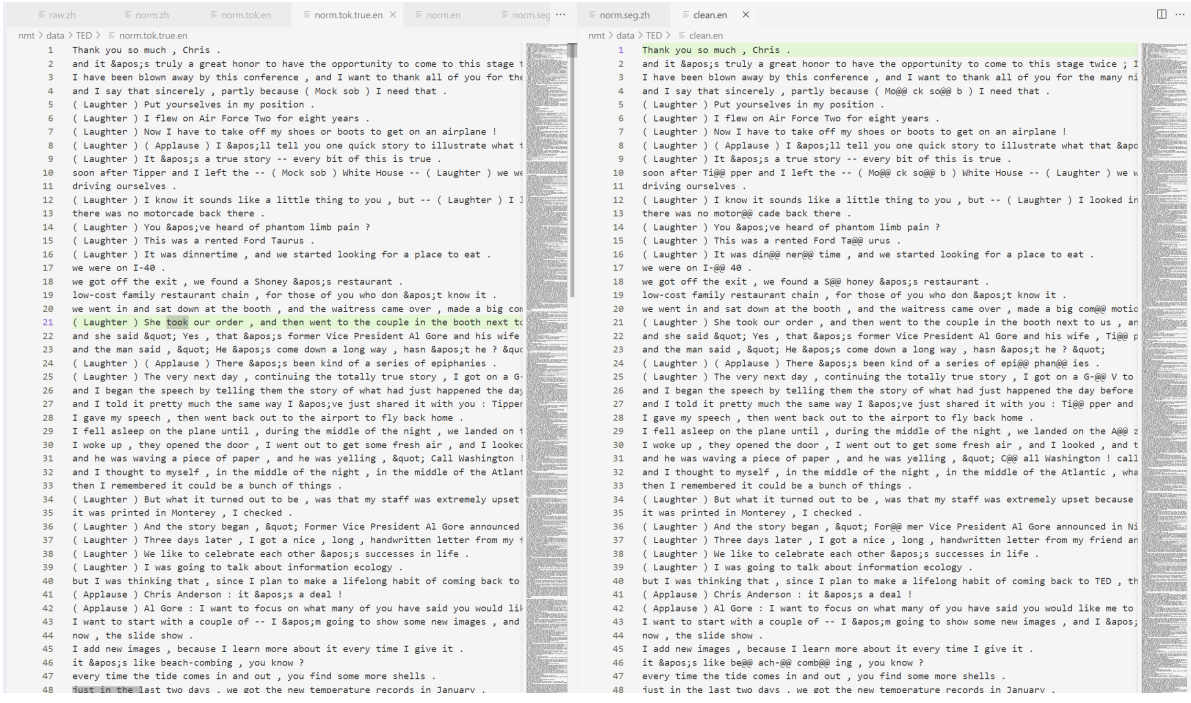
注意这里:

- 我们首先在TED数据集上学习了bpe模型(learn_joint_bpe_and_vocab.py)。
- 而后我们直接在TED数据上应用训练好的bpe模型，完成了bpe处理。

BPE处理后，我们过滤掉空白行以及长度比不合理的句对，将结果输出到clean.en和clean.zh:

```
mv ${data_dir}/norm.seg.tok.bpe.zh ${data_dir}/toclean.zh
mv ${data_dir}/norm.tok.true.bpe.en ${data_dir}/toclean.en
${CLEAN} ${data_dir}/toclean zh en ${data_dir}/clean 1 256
```

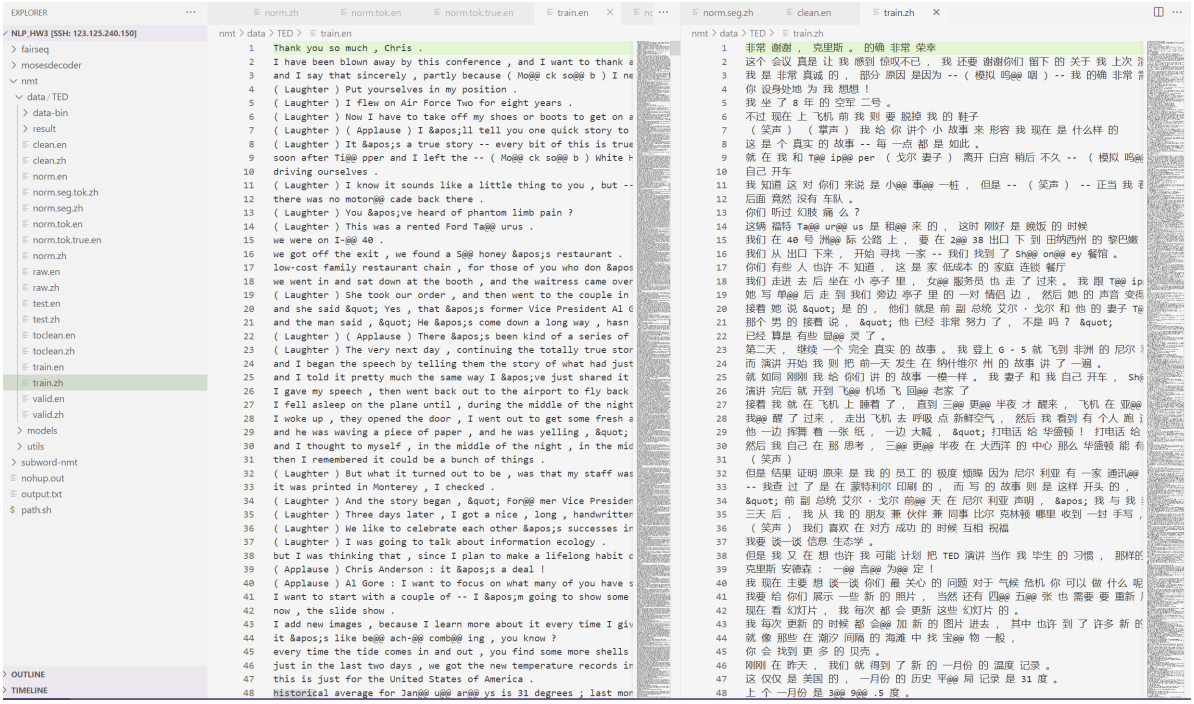
BPE处理+clean处理前后效果对比如下：（左为处理前，右为处理后，以英文语料为例）



（6）划分训练集、测试集、验证集

这里以 train : test : val = 0.95 : 0.025 : 0.025 的比例分别将处理过的clean.en和clean.zh分别划分为train.en, test.en,valid.en 和 train.zh,test.zh,valid.zh。

至此完成了整个数据预处理过程，主要是直接应用了Moses, subword-nmt和jieba三个库提供的脚本直接对原始数据集做了处理，整个处理过程的中间结果及最终处理过后且划分好的训练集、测试集、验证集如下：



3.训练过程(fairseq)

> 经过数据预处理，我们已经得到英文数据(train.en, test.en, valid.en) 和 中文数据(train.zh, test.zh, valid.zh)。接下来我使用fairseq框架用以上数据训练机器翻译模型。

（1）生成词表及二进制文件

在使用fairseq进行训练前，首先需要生成词表并将数据集转换为fairseq可以直接读取的二进制文件，这个过程可以由fairseq提供的fairseq-preprecess脚本直接完成。

fairseq-preprocess: 将文本数据转换为二进制文件，预处理命令首先会从训练文本数据中构建词表，默认情况下将所有出现过的单词根据词频排序。并将排序后的单词列表作为最终的词标。构建的词表是一个单词和序号之间的一对一的映射，这个序号是单词在词表中的下标位置。二进制化的文件会默认保存在data-bin目录下，包括生成的词表，训练数据、验证数据和测试数据，也可以通过destdir参数，将生成的数据保存在其他目录。

fairseq-preprocess中几个重要的超参数解释如下：

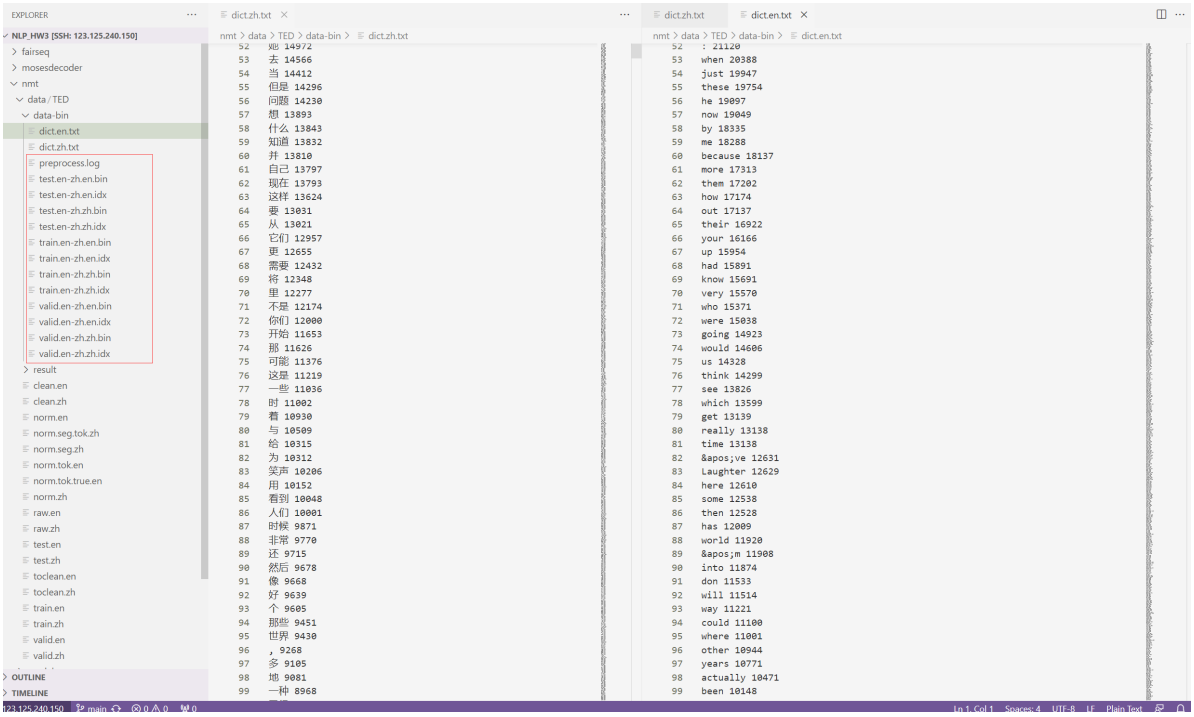
- --destdir： 预处理后的二进制文件会默认保存在data-bin目录下，可以通过destdir参数将生成的数据存放在其他位置
- --thresholdsrc/--thresholdtgt: 分别对应源端（source）和目标端（target）的词表的最低词频，词频低于这个阈值的单词将不会出现在词表中，而是统一使用一个unknown标签来代替。

- --nwordssrc/--nwordstgt，源端和目标端词表的大小，在对单词根据词频排序后，取前n个词来构建词表，剩余的单词使用一个统一的unknown标签代替。
- --source-lang: 源
- --target-lang：目标
- --trainpref：训练文件前缀（也用于建立词典），即路径和文件名的前缀。
- --validpref：验证文件前缀。
- --testpref: 测试文件前缀。

> 我原本的想法是按课件中的要求，只取1000个词生成词表。但后来考虑到既然已经花钱租了服务器，那不如充分利用算力，不具体设置词表长度，直接取全部出现的词作为词表，尝试一下效果。

```
fairseq-preprocess --source-lang ${src} --target-lang ${tgt} \
    --trainpref ${data_dir}/train --validpref ${data_dir}/valid --testpref
${data_dir}/test --destdir ${data_dir}/data-bin
```

经fairseq-preprocess处理后得到的词表以及得到的二进制文件如下：（红框内为二进制文件，左右分别为中文词表和英文词表）。



（2）训练基于transformer的机器翻译模型

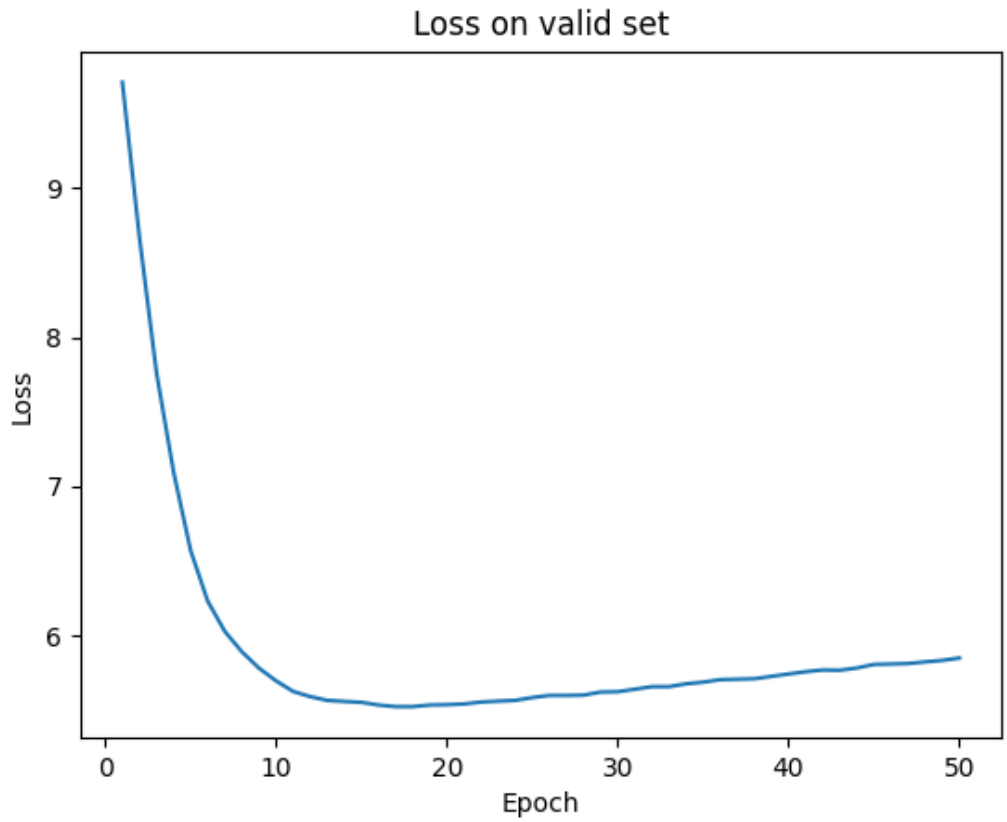
具体训练同样使用fairseq直接提供的fairseq-train脚本，fairseq-train中的一些重要超参说明如下：

- --arch：所使用的模型结构 ——这里我们设置为transformer。
- --optimizer: 可以选择的优化器：adadelta, adafactor, adagrad, adam, adamax, composite, cpu_adam, lamb, nag, sgd——这里我们使用的是adam优化器。
- --lr： 前N个批次的学习率，默认为0.25——这里我们设置为0.001。
- --lr-scheduler： 学习率缩减的方式——这里设置为inverse_sqrt。
- --criterion: 指定使用的损失函数——这里设置为label_smoothed_cross_entropy。
- --max-tokens: 每个batch包含多少个词——这里设置为4096。
- --save-dir: 训练过程中保存中间模型目录——这里设置为${model_dir}/checkpoints。

同时由于我们租的服务器上有4块GPU，因此设置CUDA_VISIBLE_DEVICES=0,1,2,3。同时我们希望将屏幕输出的log文件保存下来，因此在命令结尾加上 | tee output.txt，将屏幕输出保存在output.txt。

```
CUDA_VISIBLE_DEVICES=0,1,2,3 fairseq-train ${data_dir}/data-bin --arch
transformer \
    --source-lang ${src} --target-lang ${tgt}  \
    --optimizer adam  --lr 0.001 --adam-betas '(0.9, 0.98)' \
    --lr-scheduler inverse_sqrt --max-tokens 4096  --dropout 0.3 \
    --criterion label_smoothed_cross_entropy  --label-smoothing 0.1 \
    --max-update 200000  --warmup-updates 4000 --warmup-init-lr '1e-07' \
    --keep-last-epochs 10 --num-workers 8 \
    --save-dir ${model_dir}/checkpoints | tee output.txt
```

一共训练了50个epoch，其中在valid集上的loss曲线如下：



可以看到在17-18epoch处在valid验证集上的loss达到了最低，而后出现了过拟合。因此我们最终保留17epoch后保存的模型作为checkpoints_best。

（3）模型分享链接

由于checkpoint所占空间较大，因此没有在作业提交中直接包含checkpoint，而是选择以百度网盘的形式进行分享，连接如下：

链接：https://pan.baidu.com/s/1sRRO2XPcq3-GA5ZctFlaQg?pwd=cyt1
提取码：cyt1

4.模型评估

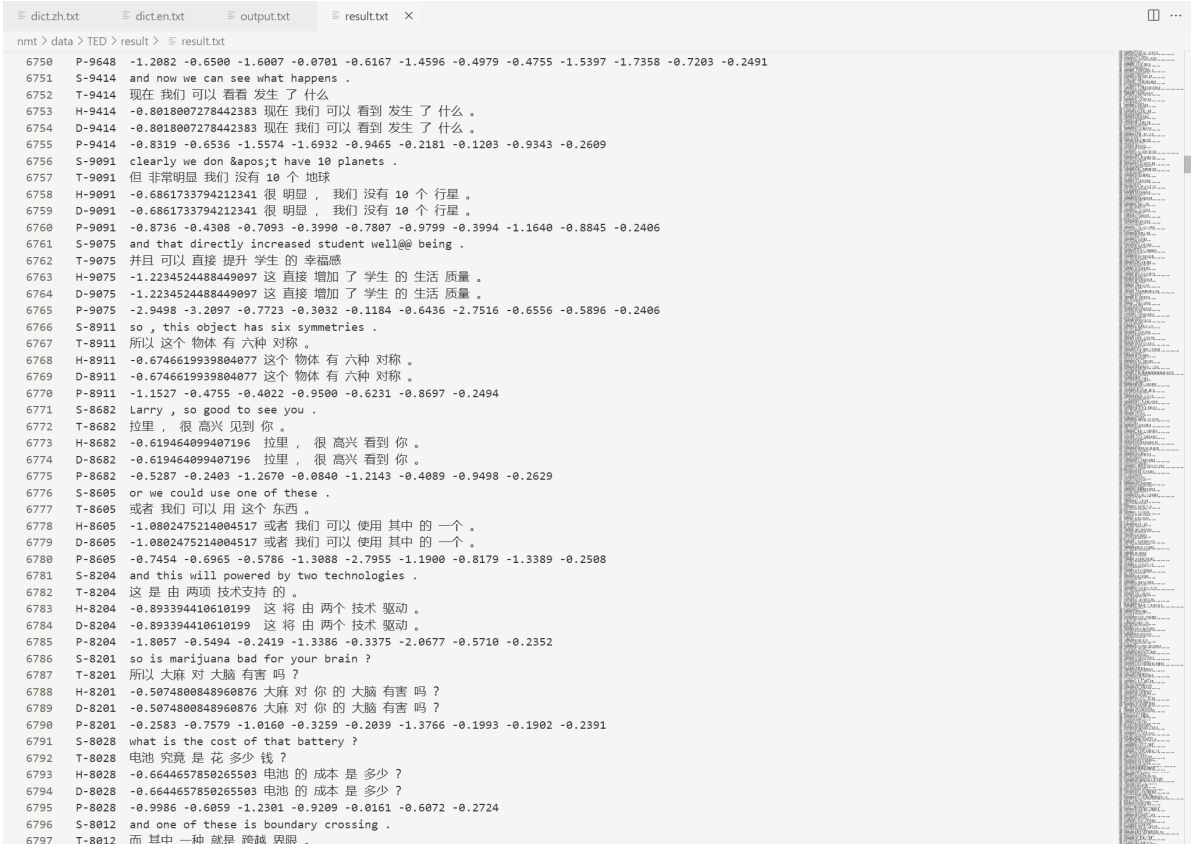
以上经过训练，我们已经得到了训练好的模型，但是我们只知道该模型在valid验证集上的loss值为5.524，但我们并不清楚实际的翻译结果如何。因此我们接下来使用训练好的模型生成valid集上的译文，直观地观察翻译的效果，并使用BLEU等指标来评估翻译结果。

（1）生成译文

使用训练好的模型生成valid验证集上的译文可以直接使用fairseq提供的fairseq-generate脚本：

```
fairseq-generate ${data_dir}/data-bin \
    --path ${model_dir}/checkpoints/checkpoint_best.pt \
    --batch-size 128 --beam 8 > ${data_dir}/result/result.txt
```

生成的译文result.txt如下：



直观观察可以看到，我们生成的译文意思是比较准确的，基本传达了英文原文的意思。但是缺点也很明显：翻译的结果还是比较生硬的。 例如上图中将 “ what is the cost of that batter? " 翻译为了"电池究竟是花多少钱"，可以明显看到意思是对的，但这个"究竟是"翻译的比较生硬，不太符合我们的生活习惯。 但总体上我对模型的性能还是比较满意的。

（2）用BLEU值评估译文效果

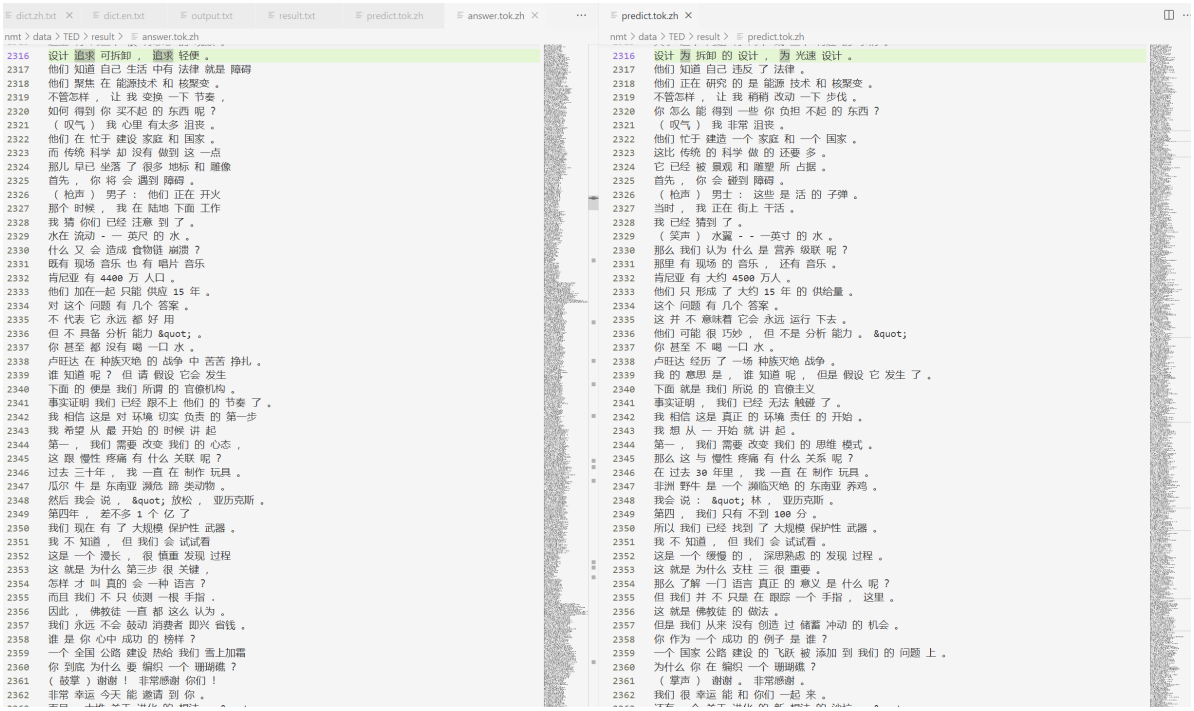
以上我们只是用肉眼直观得观察了模型的翻译效果，接下来我们通过课程中介绍过的BLEU值来评估译文的效果。当然在评估前，我们需要对原始译文result.txt做一些处理。

处理过程实际上就是我们对原始数据集做预处理的逆过程：

- Step1：从result.txt中抽取译文。
- Step2：去除bpe符号。
- Step3：detruecase将大小写恢复正常。

这部分内容不是本次实验的重点，且正向过程已在前面介绍过，因此这里省略，直接展示最终处理后得到的译文：

（左边为参考译文，右边为我们模型生成的译文）



接下来我们使用Moses库提供的/generic/multi-bleu.perl脚本直接对译文做评估，得到译文的BLEU值：

```
${MULTI_BLEU} -lc ${data_dir}/result/answer.tok.zh <
${data_dir}/result/predict.tok.zh
```

结果如下：

```
(bert) root@autodl-container-48f511a7e8-06cfee94:~/autodl-tmp/NLP_hw3# ${MULTI_BLEU} -lc ${data_dir}/result/answer.tok.zh < ${data_dir}/result/predict.tok.zh
BLEU = 16.56, 50.7/22.3/11.2/5.9 (BP=1.000, ratio=1.021, hyp_len=184474, ref_len=180592)
It is not advisable to publish scores from multi-bleu.perl.  The scores depend on your tokenizer, which is unlikely to be reproducible from your paper or consistent across research
 groups.  Instead you should detokenize then use mteval-v14.pl, which has a standard tokenization.  Scores from multi-bleu.perl can still be used for internal purposes when you hav
e a consistent tokenizer.
(bert) root@autodl-container-48f511a7e8-06cfee94:~/autodl-tmp/NLP_hw3#
```

可以看到我们的译文对比label得分为 BLEU = 16.56，其实这个分值并不算高，猜想主要原因是训练所用的数据集规模较小，模型训练不够充分。

三、参考文献

1. 《使用fairseq从头开始训练一个中英神经机器翻译模型》

https://blog.csdn.net/qq_42734797/article/details/112916511?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216880065581680021309238 2%2522%252C%2522scm%2522%253A%252220140713.130102334.pc%255Fall.%2522%257D&request_id=168800655816800213092382&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~first_rank_ecpm_v1~rank_v31_ecpm-1-112916511-null-null.142

2. 《探索Facebook NLP框架Fairseq的强大功能》

https://blog.csdn.net/weixin_42475060/article/details/128484300?ops_request_misc=&request_id=&biz_id=102&utm_term=fairseq&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduweb~default-2-128484300.142