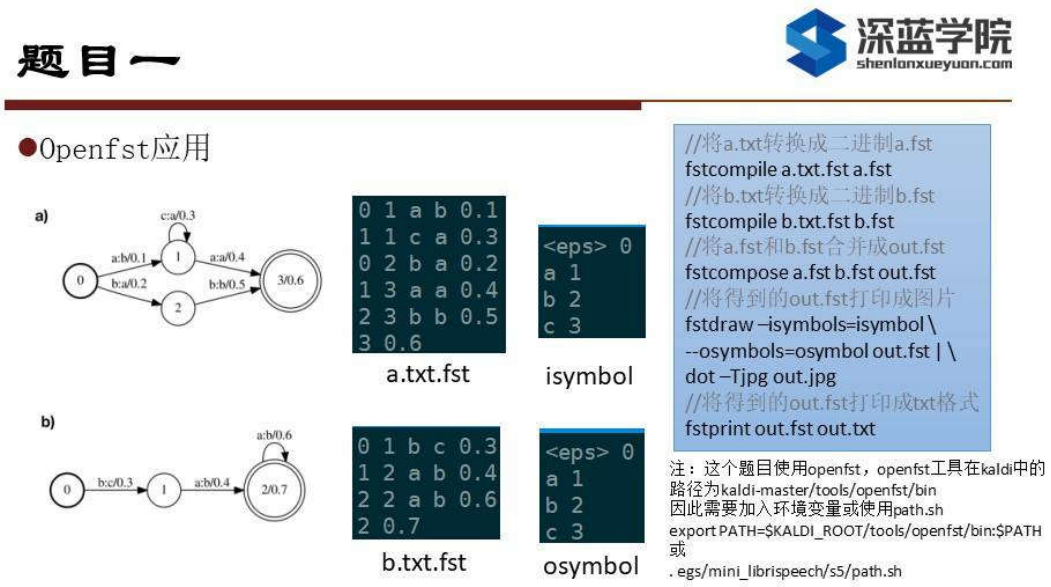
第八章作业点评

By petertsengruihon • 24 天前 • 11 次浏览

●学习讨论总结

● https://www.shenlanxueyuan.com/course/202/thread/732

本章的学习讨论总结在以上链接上，大家如果有问题的话可以链接里面进行查看。



首先我们进行第一道题目的讲解。

本章的作业我们注重实际动手操作，共有三道题目，首先是 Openfst 的应用题目，我们需要将 a 和 b 两个状态图写成 openfst 格式的文件。

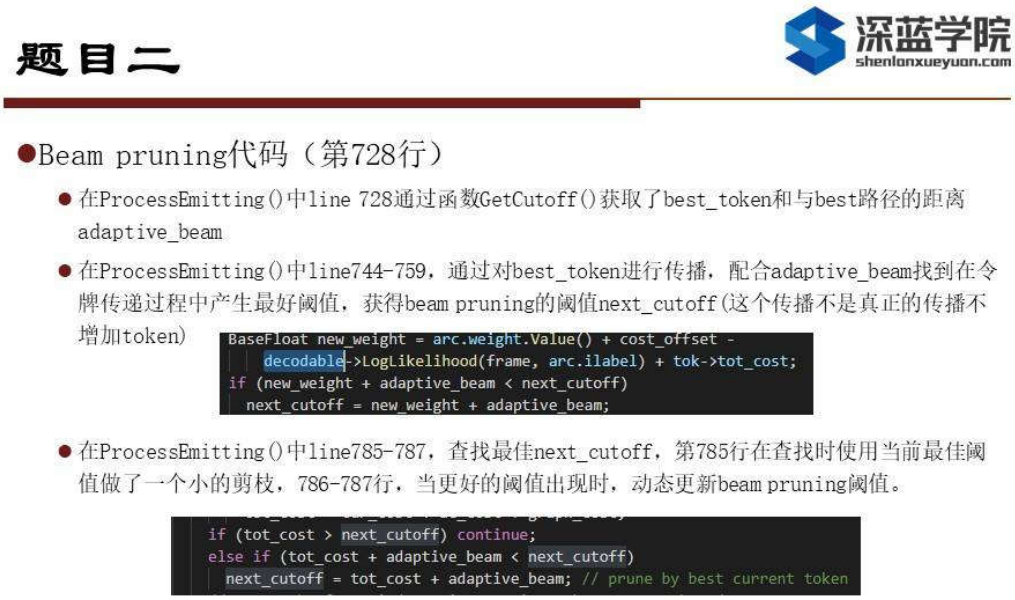
首先我们看到 fst 文件里面的内容，每一行都代表一条边，第一列是出状态节点编号，第二列是入状态节点编号，第三列是输入标签，第四类是输出标签，五列是 arc 的 weight。最后一行表示结束状态和结束状态的 weight，做这个作业我们需要的是 openfst 工具，想要使用这个工具我们首先需要将工具中的可执行文件加入环境变量（PPT 右下角的

export 指令）或者使用 kaldi 样例给的 path.sh 也可以。Fstcompile 是将 txt 格式的 fst 定义转为二进制 fst 文件，fstcompose 可以将两个二进制 fst 文件合并成一个 fst 文件，fstdraw 配合 dot 可以将 fst 打印成图片，或者是使用 fstprint 可以将 fst 输出为 txt 格式，另外在使用 fstdraw 和 fstprint 时使用 isymbol 和 osymbol 可以将输入输出的 id 号转为更容易阅读的文字。

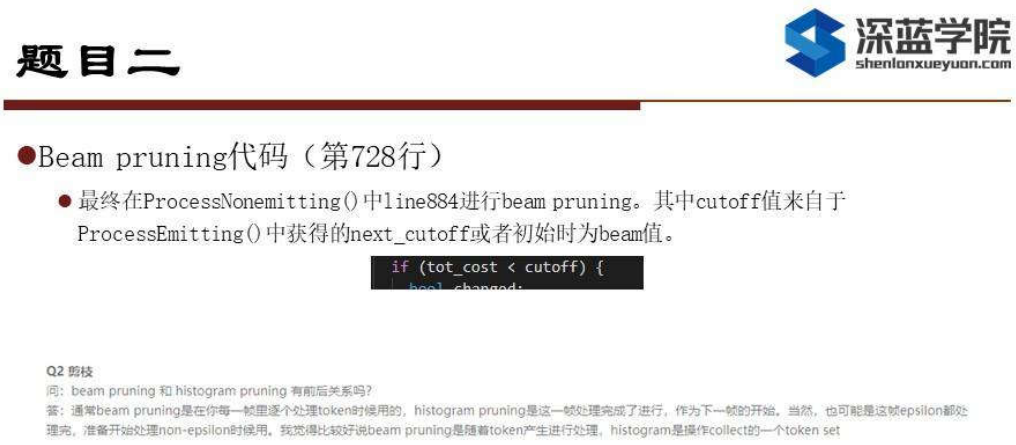
下面是作业二的讲解。

在作业二中我们要找到 kaldi 的两种剪枝方法，这两种剪枝方法主要在 kaldi/src/decoder/lattice-faster-decoder.cc 中的第 728 行 ProcessEmitting()，首

先看 Histogram pruning，代码首先通过函数 GetCutoff()获取 histogram 剪枝阈值，histogram 通过限制最大和最小的 active token 数量来进行剪枝，这个主要体现在 GetCutoff 函数里，这个函数的逻辑是如果对于 max_active 和 min_active 没有限制则返回最小 cost+beam 作为剪枝阈值，如果 max_active 的 cutoff 比 beam pruning 的小则返回 max_active 的 cutoff，如果 min_active 的 cutoff 比 beam pruning 的大则返回 min_active 的 cutoff(histogram pruning)。大家可以看到这个函数的输入，第一个是 token 的指针，第二个是 active token 数量，第三个是 beam 的大小，第四个是包含 cost 最小 token 的 Elem 对象，这个 Elem 对象是包含 token 的。最后判断 token 的 cost 和 cur_cutoff 值来完成 histogram pruning。



再来看 beam pruning，beam pruning 主要是在 token 进行传递的过程中进行阈值计算以及剪枝的，首先对最佳路径进行了试探性的传播，可以注意到这个传播并不是真正的传播，不会增加实际的 token。在这个传播中我们会计算出最佳的新阈值，之后对各个 token 进行遍历时同样也会计算最佳阈值进行更新，这里会对 token cost 做一个小剪枝，当 token 的 cost 大于当前最佳阈值时则跳过。



最终在 ProcessNoneemitting()中进行 beam pruning，这个 cutoff 值就是刚刚计算的 next_cutoff 值，在这个函数中我们实际进行了令牌传递更新了 tokens，因此在讨论区的一个问题 beam pruning 和 histogram pruning 有前后关系吗，事实上我们剪枝的阈值基于 histogram pruning 和 beam pruning 的共同作用，而这两个剪枝方法的特点

就如同下面这句话所说，histogram pruning 是这一帧处理完成了进行，作为下一帧的开始，而 beam pruning 是随着 token 产生进行处理的。

在题目三中我们将实际上手进行解码，我们首先使用 mini_librispeech 的 run.sh 进行下载、准备数据和训练到 tri1，此时我们将 data/local/lm 中的 arpa 文件的压缩包解码并转换成 G.fst，我们在转成成 G.fst 的时候使用 arpa2fst 转换成 G.fst 文件。接下来使用 tri1 模型和 tgsmall 构建的解码图进行解码产生 lattice 文件，指令就是 steps/decode.sh。这个结果应该是在 21-22 之间。我们查看对应 id 的 lattice 可以用这种方法，首先我们 echo 出 id 和 lattice 对应的 scp，通过通道符传给 lattice-copy。这里 scp：-就是把前面给到的通道输入到到 lattice-copy，这样就可以找到 id 对应的 lattice 并且打印成 txt 文件。

题目三

深蓝学院 shenlanxueyuan.com

- 重打分
  - 通过新的G.fst对原本的解码结果进行重打分，kaldi提供了steps/lmrescore.sh工具，使用如下指令
    - steps/lmrescore.sh --cmd “queue.pl” data/lang_nosp_test_tgsmall data/lang_nosp_test_tglarge data/dev_clean_2 exp/tri1/decode_nosp_tgsmall_dev_clean_2 exp/tri1/decode_nosp_tglarge_dev_clean_2
  - 该结果wer在21%-22%之间，可以通过以下指令获取最佳wer
    - grep WER exp/tri1/decode_nosp_tglarge_dev_clean_2/wer* | utils/best_wer.sh

```
%WER 20.98 [ 4224 / 20138, 457 ins, 591 del, 3176 sub ] exp/tri1/decode_nosp_tglarge_dev_clean_2/wer_15_0.0
```

最后我们使用新的 G.fst 对原本的解码结果进行重打分，使用 steps/lmrescore.sh 工具，最后获得 wer，可以使用 kaldi 提供的 best_wer.sh 获得不同 lmwt 和不同 wip 组合中的最佳 wer 作为我们最终的结果，如下图所示我得到的是 lmwt 为 15，wip 为 0.0 的 wer 结果。

QA

Q1：为什么 cur_cutoff 算的是 inf

A1：对于第一帧 GetCutoff 得到 infinity 是正常的，一般 min_active_tokens 会设置一个比较小的数值，比如 20。如果你原本这一帧初始的准备处理的 tokens 比这个 min_active 还要小，就会返回 infinity。比如你的 min_active=20，你这帧最开始只有 10 个 tokens，那么这 10 个 tokens 都会被处理，这个 cutoff=infinity。某一帧 cur_cutoff==inf 并没什么问题，这是正常的，代表不裁剪 tokens，所有的都处理，不会导致程序崩溃。

感谢各位聆听 !
Thanks for Listening

编辑