KALDI安装

建议用Ubuntu的主机或服务器。虚拟机可能会出问题。

网上有教程。

装好后egs/aishell/s5下会有示例脚本。也就是我们的实验脚本。

AIShell ASR By Kaldi

kaldi已经写好了一个训练脚本

模型训练与测试

- **修改cmd配置**

  ```
  cd egs/aishell/s5
  vim cmd.sh
  ```

  改为：

  ```
  export train_cmd=run.pl         #"queue.pl --mem 2G"
  export decode_cmd="run.pl --mem 4G"    #"queue.pl --mem 4G"
  export mkgraph_cmd="run.pl --mem 8G"   #"queue.pl --mem 8G"
  export cuda_cmd="run.pl --gpu 1"
  ```

- **修改run.sh文件中data路径（如果没有数据集的，设置好路径会自动下载解压的，压缩包30G，解压30G左右，所以最好预留个70G的内存吧。解压完了把压缩包删了就行）**

  ```
  data=/home/wang/datasets/AIShell/    # 设置成自己想要的路径
  ```

  该路径下有两个子文件夹：data_aishell以及resource_aishell

  然后需要在两个文件夹下添加 .complete 文件（否则脚本检测不到会去重新下载）

  ```
  touch ./data_aishell/.complete
  touch ./resource_aishell/.complete
  ```

- **开始训练**

  ```
  bash ./run.sh
  ```

kaldi AIShell ASR源码解读

run.sh

```
# 数据路径
data=/home/wang/datasets/AIShell/
# 下载路径（如果上者是空的，就去下载）
data_url=www.openslr.org/resources/33
# 获取到cmd.sh里定义的变量，也就是我们的运行资源以及环境变量
. ./cmd.sh
# 感兴趣的可以去看看download_and_untar.sh源码。这两行的操作是首先看数据文件是否已经存在，不存在就去下载，并自动解压。解压后会添加.complete文件作为完成解压的标识
```

```
local/download_and_untar.sh $data $data_url data_aishell || exit 1;
local/download_and_untar.sh $data $data_url resource_aishell || exit 1;
# 准备词典
# 1、将lexicon.txt全中文发音词典拷贝到"./data/local/dict/"文件夹里
# 2、根据lexicon.txt的内容。将所有非sil的因素放到nonsilence_phones.txt里（无重复）
# 3、将sil单独放在silence_phones.txt以及optional_silence.txt里
# 4、将重音以及音调相关的放在extra_questions.txt里。并把所有因素也放进去
local/aishell_prepare_dict.sh $data/resource_aishell || exit 1;
# 然后准备数据
# /AIShell/data_aishell/transcript/aishell_transcript_v0.8.txt 是标签
# 准备好train和test以及dev的数据路径等文件
local/aishell_data_prep.sh $data/data_aishell/wav $data/data_aishell/transcript || exit
1;
# 音素集、问题集；将相关信息保存在data里
# spk2gender 包含说话人的性别信息
# spk2utt 包含说话人编号和说话人的语音编号的信息
# text 包含语音和语音编号之间的关系
# utt2spk 语音编号和说话人编号之间的关系
# wav.scp 包含了原始语音的路径信息等
utils/prepare_lang.sh --position-dependent-phones false data/local/dict \
    "<SPOKEN_NOISE>" data/local/lang data/lang || exit 1;
# LM training
local/aishell_train_lms.sh || exit 1;

# G compilation, check LG composition
utils/format_lm.sh data/lang data/local/lm/3gram-mincount/lm_unpruned.gz \
    data/local/dict/lexicon.txt data/lang_test || exit 1;

# 提取MFCC以及pitch
# mfccdir should be some place with a largish disk where you
# want to store MFCC features.
mfccdir=mfcc
for x in train dev test; do
  steps/make_mfcc_pitch.sh --cmd "$train_cmd" --nj 10 data/$x exp/make_mfcc/$x $mfccdir
|| exit 1;
  steps/compute_cmvn_stats.sh data/$x exp/make_mfcc/$x $mfccdir || exit 1;
  utils/fix_data_dir.sh data/$x || exit 1;
done

steps/train_mono.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/mono || exit 1;

# 单音素模型
utils/mkgraph.sh data/lang_test exp/mono exp/mono/graph || exit 1;
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/mono/graph data/dev exp/mono/decode_dev
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/mono/graph data/test exp/mono/decode_test

# Get alignments from monophone system.
steps/align_si.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/mono exp/mono_ali || exit 1;

# train tri1 [first triphone pass]
steps/train_deltas.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/mono_ali exp/tri1 || exit 1;

# decode tri1
utils/mkgraph.sh data/lang_test exp/tri1 exp/tri1/graph || exit 1;
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri1/graph data/dev exp/tri1/decode_dev
```

```
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri1/graph data/test exp/tri1/decode_test

# align tri1
steps/align_si.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri1 exp/tri1_ali || exit 1;

# train tri2 [delta+delta-deltas]
steps/train_deltas.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/tri1_ali exp/tri2 || exit 1;

# decode tri2
utils/mkgraph.sh data/lang_test exp/tri2 exp/tri2/graph
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri2/graph data/dev exp/tri2/decode_dev
steps/decode.sh --cmd "$decode_cmd" --config conf/decode.config --nj 10 \
  exp/tri2/graph data/test exp/tri2/decode_test

# train and decode tri2b [LDA+MLLT]
steps/align_si.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri2 exp/tri2_ali || exit 1;

# Train tri3a, which is LDA+MLLT,
steps/train_lda_mllt.sh --cmd "$train_cmd" \
 2500 20000 data/train data/lang exp/tri2_ali exp/tri3a || exit 1;

utils/mkgraph.sh data/lang_test exp/tri3a exp/tri3a/graph || exit 1;
steps/decode.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
  exp/tri3a/graph data/dev exp/tri3a/decode_dev
steps/decode.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
  exp/tri3a/graph data/test exp/tri3a/decode_test

# From now, we start building a more serious system (with SAT), and we'll
# do the alignment with fMLLR.

steps/align_fmllr.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri3a exp/tri3a_ali || exit 1;

steps/train_sat.sh --cmd "$train_cmd" \
  2500 20000 data/train data/lang exp/tri3a_ali exp/tri4a || exit 1;

utils/mkgraph.sh data/lang_test exp/tri4a exp/tri4a/graph
steps/decode_fmllr.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
  exp/tri4a/graph data/dev exp/tri4a/decode_dev
steps/decode_fmllr.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
  exp/tri4a/graph data/test exp/tri4a/decode_test

steps/align_fmllr.sh  --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri4a exp/tri4a_ali

# Building a larger SAT system.

steps/train_sat.sh --cmd "$train_cmd" \
  3500 100000 data/train data/lang exp/tri4a_ali exp/tri5a || exit 1;

utils/mkgraph.sh data/lang_test exp/tri5a exp/tri5a/graph || exit 1;
steps/decode_fmllr.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
   exp/tri5a/graph data/dev exp/tri5a/decode_dev || exit 1;
steps/decode_fmllr.sh --cmd "$decode_cmd" --nj 10 --config conf/decode.config \
   exp/tri5a/graph data/test exp/tri5a/decode_test || exit 1;
```

```
steps/align_fmllr.sh --cmd "$train_cmd" --nj 10 \
  data/train data/lang exp/tri5a exp/tri5a_ali || exit 1;

# nnet3
local/nnet3/run_tdnn.sh

# chain
local/chain/run_tdnn.sh

# getting results (see RESULTS file)
for x in exp/*/decode_test; do [ -d $x ] && grep WER $x/cer_* | utils/best_wer.sh; done
2>/dev/null
for x in exp/*/*/decode_test; do [ -d $x ] && grep WER $x/cer_* | utils/best_wer.sh;
done 2>/dev/null

exit 0;
```

最后可以查看RESULTS文件。查看结果

```
# nnet3 tdnn with online pitch, local/nnet3/tuning/tun_tdnn_2a.sh
%WER 8.64 [ 9050 / 104765, 349 ins, 521 del, 8180 sub ]
exp/nnet3/tdnn_sp/decode_test/cer_15_0.5
%WER 8.72 [ 9135 / 104765, 367 ins, 422 del, 8346 sub ]
exp/nnet3/tdnn_sp_online/decode_test/cer_12_1.0
%WER 9.36 [ 9807 / 104765, 386 ins, 441 del, 8980 sub ]
exp/nnet3/tdnn_sp_online/decode_test_per_utt/cer_13_1.0

# chain with online pitch, local/chain/tuning/run_tdnn_2a.sh
%WER 7.45 [ 7807 / 104765, 340 ins, 497 del, 6970 sub ]
exp/chain/tdnn_2a_sp/decode_test/cer_11_0.5
%WER 7.43 [ 7780 / 104765, 341 ins, 469 del, 6970 sub ]
exp/chain/tdnn_2a_sp_online/decode_test/cer_11_0.5
%WER 7.92 [ 8296 / 104765, 384 ins, 472 del, 7440 sub ]
exp/chain/tdnn_2a_sp_online/decode_test_per_utt/cer_11_0.5
```

可以看到错词率在%8左右。