WeNet实践

洪青阳

WeNet介绍

在端到端语音识别领域中，面向工业应用落地的语音识别工具包WeNet设计简洁，部署方便，解决了不少痛点，因此一经推出就颇受欢迎，迅速得到普及。

[WeNet特色]

① 使用Transformer/Conformer网络结构和CTC/Attention联合优化方案；

② 流式和非流式统一的Unified Conformer模型；

③ 支持n-gram+WFST解码。

[WeNet模型]

- Transformer
- Conformer和Unified Conformer和U2++ Conformer
- RNN-T

WeNet的安装

XIAMEN UNIVERSITY

一、工具安装

1. 用git的方法把WeNet项目保存到本地：git clone https://github.com/wenet-e2e/wenet.git
2. 安装Conda，参照https://docs.conda.io/en/latest/miniconda.html
3. 创建Conda环境：

```
conda create -n wenet python=3.8
conda activate wenet
```

cd到wenet目录

```
pip install -r requirements.txt
conda install pytorch=1.10.0 torchvision torchaudio=0.10.0 cudatoolkit=11.1 -c pytorch -c conda-forge
```

二、Runtime环境安装（可选）

```
cd runtime/libtorch
mkdir build && cd build && cmake -DGRAPH_TOOLS=ON .. && cmake --build .
```

WeNet实践过程（aishell-1）

主要步骤：

stage 0: 映射文件准备

stage 1: 计算CMVN

stage 2: 词典生成

stage 3: 数据打包

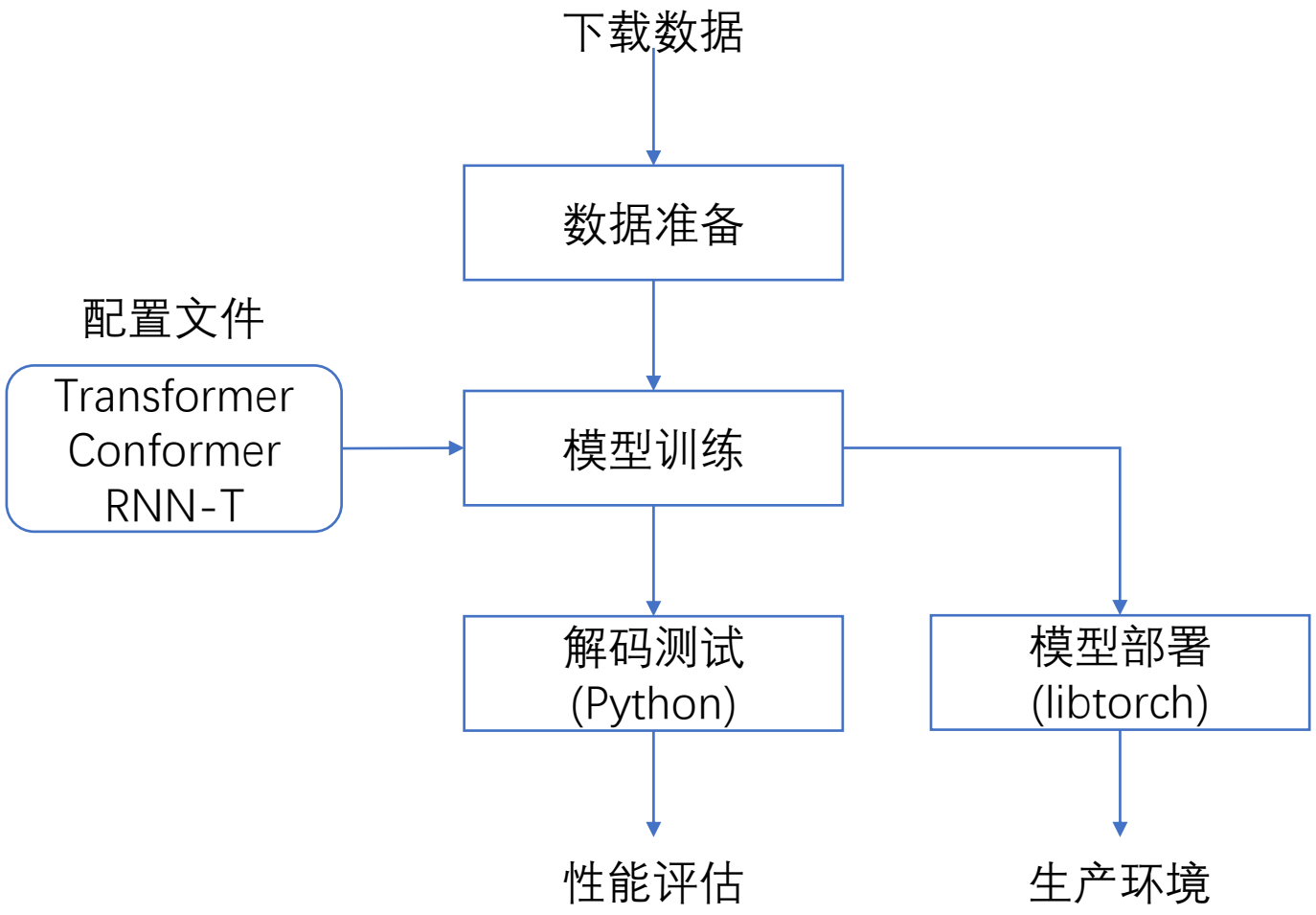
stage 4: 模型训练

stage 5: Python环境解码

stage 6: 模型导出

运行脚本：example/aishell/s0/run.sh

```
cd example/aishell/s0
bash run.sh --stage -1 --stop_stage -1
bash run.sh --stage 0 --stop_stage 0
bash run.sh --stage 1 --stop_stage 1
bash run.sh --stage 2 --stop_stage 2
bash run.sh --stage 3 --stop_stage 3
bash run.sh --stage 4 --stop_stage 4
bash run.sh --stage 5 --stop_stage 5
bash run.sh --stage 6 --stop_stage 6
```



数据准备

1. 映射文件准备

映射文件的准备主要需要两个文件，分别是wav.scp和text。对于不同的数据集，需要自行编写脚本得到这两个映射文件，在WeNet的实验中，使用aishell_data_prep.sh脚本来获得以上2个映射文件。

wav.scp文件保存了语音编号和该语音在系统中的绝对位置路径，格式如下：

<语音编号>　　<语音绝对路径>

BAC009S0002W0124　　/data_aishell/wav/train/S0002/BAC009S0002W0122.wav

该文件的作用在于将语音编号和该语音的绝对路径对应起来，使得在声学特征提取以及数据增强阶段能够访问到该条语音，进而对语音进行处理。

text文件保存了语音编号和该语音对应的转录文本，其格式如下：

<语音编号>　　<语音对应转录文本>

BAC009S0002W0124　　自六月底呼和浩特市率先宣布取消限购后

该文件的作用在于将语音编号与语音对应转录文本关联起来，在计算CTC或者attention损失时能够访问到对应的转录文本。

数据准备

2. 计算CMVN

在语音识别任务中，我们一般需要对特征进行倒谱均值归一化（CMVN）来使特征服从均值为0，方差为1的高斯分布。在WeNet中使用compute_cmvn_stats.py脚本来进行CMVN的计算，这样的处理能够让神经网络更容易对语音特征进行学习。

数据准备

3. 词典生成

在WeNet中，使用脚本text2token.py 通过映射文件中的text文件来生成词典，得到每个字符的唯一表示数字，其格式如下：

<字符> <字符对应的数字>

字 1

在训练过程中，我们会通过这个词典，将每条语音对应的转录文本转换为数字，我们将其称为token。

数据准备

XIAMEN UNIVERSITY

4. 数据打包

在WeNet的模型训练中，不是直接使用wav.scp或者text这些映射文件，而是提供了两种数据打包格式，分别为针对大数据量的shard格式和小数据量的raw格式。对于aishell-1数据集，我们使用raw格式即可。这里使用脚本make_raw_list.py把训练所需映射文件写入data.list文件中，我们以其中一条语音举例，其格式如下：

{"key": "BAC009S0002W0122", "wav": "/data1/data/aishell_1//data_aishell/wav/train/S0002/BAC009S0002W0122.wav", "txt": "而对楼市成交抑制作用最大的限购"}

其中包含音频id、音频路径和其对应的转录文本。

通过以上四个步骤，我们就完成了进行端到端语音识别训练的数据准备，接下来介绍训练过程的一些配置文件。

配置文件

在WeNet的aishell例子中的conf目录下，存放了一系列模型训练的配置文件。

train_conformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_conformer_no_pos.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_transformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_u2++_conformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_u2++_transformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_unified_conformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--
train_unified_transformer.yaml	2KB	YAML 文件	2022/3/9, 23:16	-rw-r--r--

Conformer配置文件: train_conformer.yaml

```yaml
# network architecture
# encoder related
encoder: conformer
encoder_conf:
    output_size: 256    # dimension of attention
    attention_heads: 4
    linear_units: 2048  # the number of units of position-wise feed forward
    num_blocks: 12      # the number of encoder blocks
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    attention_dropout_rate: 0.0
    input_layer: conv2d # encoder input type, you can choose conv2d, conv2d6 and conv2d8
    normalize_before: true
    cnn_module_kernel: 15
    use_cnn_module: True
    activation_type: 'swish'
    pos_enc_layer_type: 'rel_pos'
    selfattention_layer_type: 'rel_selfattn'

# decoder related
decoder: transformer
decoder_conf:
    attention_heads: 4
    linear_units: 2048
    num_blocks: 6
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    self_attention_dropout_rate: 0.0
    src_attention_dropout_rate: 0.0
```

Conformer配置文件: train_unified_conformer.yaml

```yaml
# network architecture
# encoder related
encoder: conformer
encoder_conf:
    output_size: 256    # dimension of attention
    attention_heads: 4
    linear_units: 2048  # the number of units of position-wise feed forward
    num_blocks: 12      # the number of encoder blocks
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    attention_dropout_rate: 0.0
    input_layer: conv2d # encoder input type, you can chose conv2d, conv2d6 and conv2d8
    normalize_before: true
    cnn_module_kernel: 15
    use_cnn_module: True
    activation_type: 'swish'
    pos_enc_layer_type: 'rel_pos'
    selfattention_layer_type: 'rel_selfattn'
    causal: true
    use_dynamic_chunk: true
    cnn_module_norm: 'layer_norm' # using nn.LayerNorm makes model converge faster
    use_dynamic_left_chunk: false

# decoder related
decoder: transformer
decoder_conf:
    attention_heads: 4
    linear_units: 2048
    num_blocks: 6
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    self_attention_dropout_rate: 0.0
    src_attention_dropout_rate: 0.0
```

Conformer配置文件: train_u2++_conformer.yaml

```yaml
# network architecture
# encoder related
encoder: conformer
encoder_conf:
    output_size: 256    # dimension of attention
    attention_heads: 4
    linear_units: 2048  # the number of units of position-wise feed forward
    num_blocks: 12      # the number of encoder blocks
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    attention_dropout_rate: 0.1
    input_layer: conv2d # encoder input type, you can chose conv2d, conv2d6 and conv2d8
    normalize_before: true
    cnn_module_kernel: 8
    use_cnn_module: True
    activation_type: 'swish'
    pos_enc_layer_type: 'rel_pos'
    selfattention_layer_type: 'rel_selfattn'
    causal: true
    use_dynamic_chunk: true
    cnn_module_norm: 'layer_norm' # using nn.LayerNorm makes model converge faster
    use_dynamic_left_chunk: false

# decoder related
decoder: bitransformer
decoder_conf:
    attention_heads: 4
    linear_units: 2048
    num_blocks: 3
    r_num_blocks: 3
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    self_attention_dropout_rate: 0.1
    src_attention_dropout_rate: 0.1
```

RNN-T配置文件: conformer_rnnt.yaml

```yaml
# network architecture
# encoder related
encoder: conformer
encoder_conf:
    output_size: 256    # dimension of attention
    attention_heads: 4
    linear_units: 2048  # the number of units of position-wise feed forward
    num_blocks: 12      # the number of encoder blocks
    dropout_rate: 0.1
    positional_dropout_rate: 0.1
    attention_dropout_rate: 0.1
    input_layer: conv2d # encoder input type, you can chose conv2d, conv2d6 and conv2d8
    normalize_before: true
    cnn_module_kernel: 15
    use_cnn_module: true
    activation_type: 'swish'
    pos_enc_layer_type: 'rel_pos'
    selfattention_layer_type: 'rel_selfattn'

joint_conf:
    join_dim: 512
    prejoin_linear: True
    postjoin_linear: false
    joint_mode: 'add'
    activation: 'tanh'

predictor: rnn
predictor_conf:
    embed_size: 256
    output_size: 256
    embed_dropout: 0.1
    hidden_size: 256
    num_layers: 2
    bias: true
    rnn_type: 'lstm'
    dropout: 0.1

decoder: bitransformer
decoder_conf:
  attention_heads: 4
  dropout_rate: 0.1
  linear_units: 2048
  num_blocks: 3
  positional_dropout_rate: 0.1
  r_num_blocks: 3
  self_attention_dropout_rate: 0.1
  src_attention_dropout_rate: 0.1
```

模型训练

在WeNet中，使用train.py 来进行模型的训练，我们以aishell-1的模型训练举例：

```
python wenet/bin/train.py --gpu $gpu_id \
      --config $train_config \
      --data_type $data_type \
      --symbol_table $dict \
      --train_data data/$train_set/data.list \
      --cv_data data/dev/data.list \
      ${checkpoint:+--checkpoint $checkpoint} \
      --model_dir $dir \
      --ddp.init_method $init_method \
      --ddp.world_size $world_size \
      --ddp.rank $rank \
      --ddp.dist_backend $dist_backend \
      --num_workers 1 \
      $cmvn_opts \
      --pin_memory
```

在train.py 的所有参数中，我们需要准备的文件是4个，第一个就是模型训练的配置脚本，对应上图中的train_config脚本，第二个和第三个分别是训练集数据和验证集数据的data.list，最后一个需要准备的文件是词典，对应上图中的symbol_table，该词典将训练文本和验证文本转换成相应的数字表示进行训练。

Python环境解码

WeNet支持python环境的模型性能评估，方便在正式部署前的模型调试工作。在解码之前，我们会使用脚本average_model.py生成一个平均最优模型进行解码，该脚本使用方法如下图：

```
python wenet/bin/average_model.py \
      --dst_model $decode_checkpoint \
      --src_path $dir  \
      --num ${average_num} \
      --val_best
```

其中，dst_model是指生成的平均最优模型的路径，src_path是指所有训练模型保存路径，num是指选取出来生成平均最优模型的模型个数，val_best是指选取方法为挑选在开发集上表现最优的模型。

Python环境解码

WeNet的语音识别解码使用脚本recognize.py来实现，可以使用GPU进行解码。我们依然以aishell-1的解码为例，该脚本使用方法如下图所示：

```
python wenet/bin/recognize.py --gpu 0 \
      --mode $mode \
      --config $dir/train.yaml \
      --data_type $data_type \
      --test_data data/test/data.list \
      --checkpoint $decode_checkpoint \
      --beam_size 10 \
      --batch_size 1 \
      --penalty 0.0 \
      --dict $dict \
      --ctc_weight $ctc_weight \
      --reverse_weight $reverse_weight \
      --result_file $test_dir/text \
      ${decoding_chunk_size:+--decoding_chunk_size $decoding_chunk_size}
```

其中，gpu是指用来解码的GPU编号，mode是解码模式，WeNet提供了四种解码模式，分别为ctc_greedy_search，ctc_prefix_beam_search，attention和attention_rescoring；config是指模型训练的配置文件，data_type是指数据准备格式，test_data是指测试集数据的data.list文件，checkpoint是指调用的模型，beam_size是beam search的宽度，batch_size是每个batch的音频数量，penalty是长度惩罚参数，dict是词典文件，ctc_weight是CTC解码权重，reverse_weight是从右往左解码权重，result_file是指解码结果文件存放路径，decoding_chunk_size是解码时的chunk大小。

Python环境解码

解码后使用compute-wer.py脚本计算WER，该脚本使用方法如下图：

```
python tools/compute-wer.py --char=1 --v=1 \
      data/test/text $test_dir/text > $test_dir/wer
```

其中char是指是否以字为建模单元做分隔，v是指是否打印计算过程信息。脚本执行完成后，就能在wer文件中找到测试结果。

```
================================================

Overall -> 5.01 % N=307258 C=292198 S=14742 D=318 I=344
Mandarin -> 5.01 % N=307251 C=292198 S=14735 D=318 I=344
Other -> 100.00 % N=7 C=0 S=7 D=0 I=0

================================================
```

模型部署

WeNet中基于LibTorch提供了云端X86和嵌入式端Android的落地方案，同时还引入了语言模型辅助解码。

1. 模型导出

为了能够方便模型部署，WeNet使用脚本export_jit.py将模型进行打包，还提供了一种动态量化方案对模型进行压缩，该脚本使用方法如下图所示：

```
python wenet/bin/export_jit.py \
    --config $dir/train.yaml \
    --checkpoint $dir/avg_${average_num}.pt \
    --output_file $dir/final.zip \
    --output_quant_file $dir/final_quant.zip
```

其中，config是指模型的训练配置参数，checkpoint是进行打包的模型路径，output_file和output_quant_file分别指原模型打包生成路径和量化压缩模型打包生成路径。

模型部署

XIAMEN UNIVERSITY

2. 语言模型训练

WeNet中选择基于 n-gram 的统计语言模型，结合WFST解码技术，实现对定制语言模型的支持。具体语言模型训练方法如下图所示：

```
# Prepare dict
  unit_file=$dict
  mkdir -p data/local/dict
  cp $unit_file data/local/dict/units.txt
  tools/fst/prepare_dict.py $unit_file ${data}/resource_aishell/lexicon.txt \
    data/local/dict/lexicon.txt
  exit 0

# Train lm
  lm=data/local/lm
  mkdir -p $lm
  tools/filter_scp.pl data/train/text \
    $data/data_aishell/transcript/aishell_transcript_v0.8.txt > $lm/text
  local/aishell_train_lms.sh

  # Build decoding TLG
  tools/fst/compile_lexicon_token_fst.sh \
    data/local/dict data/local/tmp data/local/lang
  tools/fst/make_tlg.sh data/local/lm data/local/lang data/lang_test || exit 1;
```

模型部署

3. 结合语言模型的解码

生成TLG后，最后使用decode.sh脚本进行解码，该脚本如下图所示：

```
# Decoding with runtime
  chunk_size=-1
    ./tools/decode.sh --nj 16 \
    --beam 15.0 --lattice_beam 7.5 --max_active 7000 \
    --blank_skip_thresh 0.98 --ctc_weight 0.5 --rescoring_weight 1.0 \
    --chunk_size $chunk_size \
    --fst_path data/lang_test/TLG.fst \
    data/test/wav.scp data/test/text $dir/final.zip \
    data/lang_test/words.txt $dir/lm_with_runtime
```

其中，引入max_active来避免某一时刻state数目过大，blank_skip_thresh是CTC中WFST的空白跳过阈值。

测试结果

[aishell-1]

以下给出了aishell-1的实验结果，对比了Conformer模型不同解码策略的CER%指标。

- Training info: lr 0.002, batch size 16, 240 epochs, dither 0.1
- Decoding info: ctc_weight 0.5, average_num 30

解码策略	CER
ctc_greedy_search	4.87%
ctc_prefix_beam_search	4.87%
attention	4.87%
attention_rescoring	4.60%
lm_with_runtime (TLG)	4.31%

致谢

- 感谢胡文轩同学对WeNet实践过程做了深入细致的整理。
- 感谢厦门大学智能语音实验室其他同学的贡献。