实验2-3 基于Wenet的连续语音识别系统实现

目录

1 实验目的

通过本章学习，掌握语音识别模型的训练数据生成方式和语音识别模型的训练方式。通过主动调整模型训练参数并观察最终的模型性能表现，与其他学生的结果进行对比，分析影响模型训练的重要影响因素，自行平衡模型性能与训练成本之间的关系。

2 预习要求

- 预习《神经网络与深度学习》的第7章“网络优化与正则化”[1]；
- 预习《动手学深度学习》的第3.11 节“模型选择、欠拟合和过拟合”[2];
- 预习《动手学深度学习》的第8.4 节“多GPU计算”[2]；

3 实验原理

3.1 wenet模型框架

3.1.1模型总体架构

该模型使用了一种CTC/AED联合结构，其中CTC Decoder用于计算CTC损失，并通过CTC束解码的方式对编码器的输出进行解码；Attention Decoder是一个标准的Transformer Decoder结构，用于对CTC Decoder的输出进行重打分。两个Decoder共享一个Encoder[2]，其具体结构如图1所示。

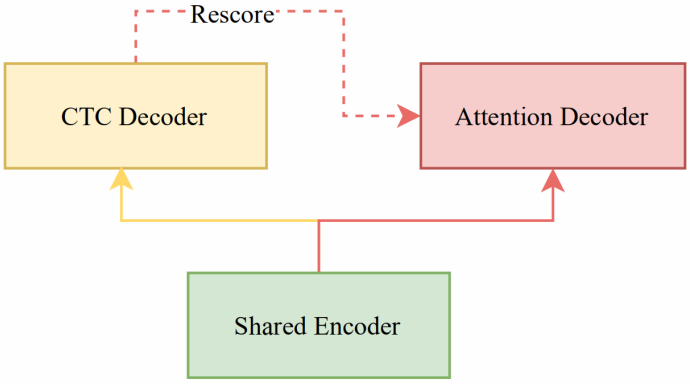

图1 CTC /AED联合结构[2]

该共享Encoder使用80维的Fbank特征 X 作为输入，首先通过CNN进行下采样，降低序列长度，之后使用N个Conformer编码器结构进行特征提取。

在解码过程中，Encoder将首先对输入的特征序列X进行编码

$$\mathrm{hidden_{en}} = \mathrm{Encoder(X)} \quad (1)$$

编码器的编码结果 $\mathrm{hidden_{en}}$ 首先被输入到CTC Decoder，通过束解码的方式进行解码，得到概率最大的N个候选字符序列 $\mathrm{Y_{cand}}$ 以及其对应的CTC评分 $\mathrm{score_{ctc}}$

$$\mathrm{Y_{cand}, score_{ctc}} = \mathrm{Decoder_{CTC}\left(hidden_{en}\right)} \quad (2)$$

候选字符序列 $\mathrm{Y_{cand}}$ 和编码器的编码结果 $\mathrm{hidden_{en}}$ 将进一步作为Attention Decoder的输入，通过教师强制的方式对每个序列进行打分

$$\mathrm{score_{att}} = \mathrm{Decoder_{att}\left(hidden_{en}, Y_{cand}\right)} \quad (3)$$

之后对候选字符序列的两种评分进行加权求和，选择对应评分最大的序列作为模型的最终输出

$$\mathrm{score} = \mathrm{score_{ctc}} * \mathrm{weight_{ctc}} + \mathrm{score_{att}} * \mathrm{weight_{en}} \quad (4)$$

3.1.2 损失函数的设计

CTC/AED联合结构已经在端到端ASR任务中得到广泛运用，与单独的CTC和Encoder-Decoder架构相比，它有效地利用了两种架构在训练和解码方面的优势。

一方面，Encoder-Decoder最早应用于机器翻译领域，编码器和解码器之间通过注意力机制进行衔接，是一种软对齐结构，缺乏从左到右的约束，输入序列和输出序列之间不存在严格的前后对应关系，因此可以适用于不同语言的语法结构。但是语音识别任务与机器翻译任务不同，音频信号和输出字符间存在严格的前后对应关系，因此将其应用于语音识别任务时收敛速度较慢，训练困难较大，对数据量的要求高。通过添加CTC损失作为一种硬对齐策略辅助进行训练可以有效施加音频信号和字符序列先后顺序的强约束，因此可以有效提升收敛速度，降低训练难度。

另一方面，CTC是一种条件独立性约束（Conditional Independence Constraint），前面字符的解码结果不影响后续字符的概率，无法有效利用上下文信息，单独使用进行解码无法充分发挥束解码的优势。通过添加AED损失可以将上下文信息作为补充，对CTC输出的概率分布进行修正。

因此在训练阶段，损失函数设计如下所示，其中 X 为输入的特征， Y 为对应的真实字符序列。

$$\mathrm{L_{combined}(X,Y)}=\lambda\,\mathrm{L_{ctc}(X,Y)}+(1-\lambda)\mathrm{L_{att}(X,Y)} \tag{5}$$

3.2 注意力机制基本原理

注意力机制（Attention Mechanism）是在计算能力有限的情况下，将计算资源分配给更重要的任务，同时解决信息超载问题的一种资源分配方案。在神经网络学习中，一般而言模型的参数越多则模型的表达能力越强，模型所存储的信息量也越大，但这会带来信息过载的问题。通过引入注意力机制，在众多的输入信息中聚焦于对当前任务更为关键的信息，降低对其他信息的关注度，甚至过滤掉无关信息，就可以解决信息过载问题，并提高任务处理的效率和准确性。

例如人做阅读理解任务时，提前确定好一个目标，在繁琐的文章中找出和目标最相关的句子进行分析，而不是无差别的对全文进行分析和研究。这种找出和任务相关的关键信息的能力，就是注意力机制。“对症下药”“因地制宜”“量体裁衣”也包含了注意力机制的思想。

常用的注意力机制为键值对注意力机制，为了从*N*个输入向量 $\boldsymbol{X}=[\boldsymbol{x}_1,\cdots,\boldsymbol{x}_\mathrm{N}]\in\mathrm{R}^{\mathrm{N}\times\mathrm{d}}$ 中选择出和某个特定任务相关的信息，首先通过线性转换分别计算**键向量（Key Vector）** $\boldsymbol{K}=\boldsymbol{X}\boldsymbol{W}_\mathrm{K}=[\boldsymbol{k}_1,\cdots,\boldsymbol{k}_\mathrm{N}]\in\mathrm{R}^{\mathrm{N}\times\mathrm{d}}$ 和**值向量（Value Vector）** $\boldsymbol{V}=\boldsymbol{X}\boldsymbol{W}_\mathrm{V}=[\boldsymbol{v}_1,\cdots,\boldsymbol{v}_\mathrm{N}]\in\mathrm{R}^{\mathrm{N}\times\mathrm{d}}$ ，二者的对应向量之间一一对应，前者用于衡量和指定任务关联性，后者用于对输入信息进行特征提取。此外我们需要引入一个和任务相关的表示 $\boldsymbol{q}$ 称为**查询向量（Query Vector）**，并通过一个打分函数 $\mathrm{s(k,q)}$ 计算注意力评分，该函数输出一个标量，用于衡量查询向量 $\boldsymbol{q}$ 和每个键向量 $\boldsymbol{k}_\mathrm{i}$ 之间的相关性，常用的打分函数如下所示。

表1打分函数

加性模型	$\mathrm{s}(\boldsymbol{k},\boldsymbol{q})=\boldsymbol{v}^{\mathrm{T}}\tanh(\boldsymbol{k}\boldsymbol{W}+\boldsymbol{q}\boldsymbol{U})$
点积模型	$\mathrm{s}(\boldsymbol{k},\boldsymbol{q})=\boldsymbol{q}\boldsymbol{k}^{\mathrm{T}}$
缩放点积模型	$\mathrm{s}(\boldsymbol{k},\boldsymbol{q})=\dfrac{\boldsymbol{q}\boldsymbol{k}^{\mathrm{T}}}{\sqrt{\mathrm{d}}}$

其中 $\boldsymbol{W}$ ， $\boldsymbol{U}$ ， $\boldsymbol{v}$ 为可学习的参数，*d*为输入向量的维度。点积模型与加性模型相比参数量和计算量小，复杂度低，且可以更好的利用矩阵运算，因此计算效率更高。当查询向量和键向量的维度*d*比较高时，点积模型的值通常有比较大的方差，导致Softmax函数的梯度较小甚至发生梯度消失现象，因此通过缩放点积模型根据维度*d*对计算得到的注意力评分进行额外的放缩，降低方差，进而缓解梯度消失的发生。

之后对计算得到的注意力评分 $\boldsymbol{s}=[\mathrm{s}_1,\ldots,\mathrm{s}_\mathrm{N}]$ 使用softmax进行归一化处理，计算注意力分布 $\boldsymbol{\alpha}=[\alpha_1,\ldots,\alpha_\mathrm{N}]$ ，注意力分布 α_i 可以解释为在给定任务相关的查询向量 $\boldsymbol{q}$ 时，对值向量 $\boldsymbol{v}_\mathrm{i}$ 的关注程度，计算方式如下所示。

$$\alpha_\mathrm{i}=\frac{\exp\left(\mathrm{s}_\mathrm{i}\right)}{\sum_{\mathrm{j}=1}^{\mathrm{N}}\exp\left(\mathrm{s}_\mathrm{j}\right)} \quad (6)$$

最终的输出将依据注意力分布 $\boldsymbol{\alpha}$ 和值向量 $\boldsymbol{V}$ 进行计算，主要包括软性注意力机制和硬性注意力机制两种。

软性注意力机制（Soft Attention Mechanism）：以注意力分布 $\boldsymbol{\alpha}$ 为权重对值向量 $\boldsymbol{V}$ 进行加权求和，进而对所有输入的信息进行筛选和聚合，其输出的计算为：

$$\boldsymbol{a}=\boldsymbol{\alpha}\boldsymbol{V}=\sum_{\mathrm{i}=1}^{\mathrm{N}}\alpha_\mathrm{i}\boldsymbol{v}_\mathrm{i} \quad (7)$$

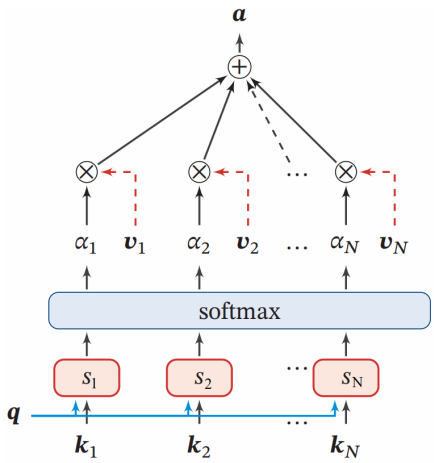

图2软注意力机制[1]

硬性注意力机制（Hard Attention Mechanism）：只关注某一个值向量而非是所有。有两种实现方式：

（1）选取最大的注意力分布对应的值向量，即

$$\mathrm{n}=\underset{\mathrm{i}}{\operatorname{argmax}}\,\alpha_{\mathrm{i}} \quad (8)$$

$$\boldsymbol{a}=\boldsymbol{v}_{\mathrm{n}} \quad (9)$$

2）根据注意力分布作为概率分布随机选择一个对应的值向量作为输出。

在Transformer中使用了多头注意力机制构建编码器和解码器，假设输入的查询向量 $\boldsymbol{Q}_{\mathrm{in}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{d}_{\mathrm{in}}}$ ，键向量 $\boldsymbol{K}_{\mathrm{in}} \in \mathrm{R}^{\mathrm{t}_2 \times \mathrm{d}_{\mathrm{in}}}$ ，值向量 $\boldsymbol{V}_{\mathrm{in}} \in \mathrm{R}^{\mathrm{t}_2 \times \mathrm{d}_{\mathrm{in}}}$ （注意，查询向量和键向量的维度必须相同，以保证可以使用矩阵相乘，但是序列长度无显性限制；键向量和值向量的序列长度必须相同，保证二者一一对应，但是维度无显性限制），共包含H个注意力头，每个注意力头的计算均为键值对软注意力机制，每个注意力头均有相同的输入，但是相互之间不共享参数。其计算流程主要包括以下五个步骤，其中step1至step4在每个注意力头间独立进行，互不干涉，具体流程如图3所示。

step 1:对输入的查询向量 $\boldsymbol{Q}_{\mathrm{in}}$ ，键向量 $\boldsymbol{K}_{\mathrm{in}}$ ，值向量 $\boldsymbol{V}_{\mathrm{in}}$ 进行初步的线性变换，$\boldsymbol{W}_{\mathrm{Qi}} \in \mathrm{R}^{\mathrm{d}_{\mathrm{in}} \times \mathrm{d}}$ ， $\boldsymbol{W}_{\mathrm{Ki}} \in \mathrm{R}^{\mathrm{d}_{\mathrm{in}} \times \mathrm{d}}$ ， $\boldsymbol{W}_{\mathrm{Vi}} \in \mathrm{R}^{\mathrm{d}_{\mathrm{in}} \times \mathrm{d}}$ ， $\boldsymbol{b}_{\mathrm{Qi}} \in \mathrm{R}^{\mathrm{d}}$ ， $\boldsymbol{b}_{\mathrm{Ki}} \in \mathrm{R}^{\mathrm{d}}$ ， $\boldsymbol{b}_{\mathrm{Vi}} \in \mathrm{R}^{\mathrm{d}}$ 为注意力头i的权重，计算方式为

$$\boldsymbol{Q}_{\mathrm{i}}=\boldsymbol{Q}_{\mathrm{in}} \boldsymbol{W}_{\mathrm{Qi}}+\boldsymbol{b}_{\mathrm{Qi}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{d}} \quad (11)$$

$$\boldsymbol{K}_{\mathrm{i}}=\boldsymbol{K}_{\mathrm{in}} \boldsymbol{W}_{\mathrm{Ki}}+\boldsymbol{b}_{\mathrm{Ki}} \in \mathrm{R}^{\mathrm{t}_2 \times \mathrm{d}} \quad (11)$$

$$\boldsymbol{V}_{\mathrm{i}}=\boldsymbol{V}_{\mathrm{in}} \boldsymbol{W}_{\mathrm{Vi}}+\boldsymbol{b}_{\mathrm{Ki}} \in \mathrm{R}^{\mathrm{t}_2 \times \mathrm{d}} \quad (12)$$

step2：计算注意力头i的注意力评分 $\boldsymbol{scores}_{\mathrm{i}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{t}_2}$

$$\boldsymbol{scores}_{\mathrm{i}}=\frac{\boldsymbol{Q}_{\mathrm{i}} \boldsymbol{K}_{\mathrm{i}}^{\mathrm{T}}}{\sqrt{\mathrm{d}_{\mathrm{k}}}} \quad (13)$$

step3：计算注意力分布 $\boldsymbol{attr}_{\mathrm{i}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{t}_2}$

$$\boldsymbol{attr}_{\mathrm{i}}=\operatorname{softmax}\left(\boldsymbol{scores}_{\mathrm{i}}\right) \quad (14)$$

step4：使用软注意力机制计算注意力头*i*的输出 $\boldsymbol{output}_{\mathrm{i}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{d}}$

$$\boldsymbol{output}_{\mathrm{i}}=\boldsymbol{attr}_{\mathrm{i}} \boldsymbol{V}_{\mathrm{i}} \quad (15)$$

step5：将H个注意力头计算得到的结果在特征维度上拼接起来，得到 $\boldsymbol{output}' \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{hd}}$ ，$\boldsymbol{W}_{\mathrm{output}} \in \mathrm{R}^{\mathrm{hd} \times \mathrm{d}_{\mathrm{out}}}$ 和 $\boldsymbol{b}_{\mathrm{output}} \in \mathrm{R}^{\mathrm{d}_{\mathrm{out}}}$ 为可学习参数，通过线性转换将不同注意力头之间的信息进行混合

$$\boldsymbol{output}'=\operatorname{cat}\left(\boldsymbol{output}_1, \boldsymbol{output}_2, \ldots, \boldsymbol{output}_{\mathrm{h}}, \dim=-1\right) \quad (16)$$

$$\boldsymbol{output}=\boldsymbol{output}' \boldsymbol{W}_{\mathrm{output}}+\boldsymbol{b}_{\mathrm{output}} \in \mathrm{R}^{\mathrm{t}_1 \times \mathrm{d}} \quad (17)$$

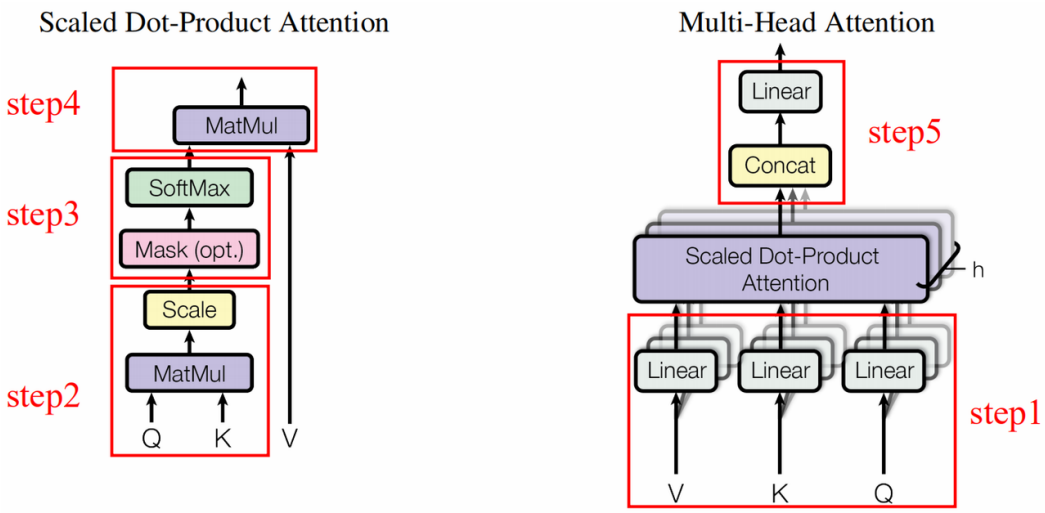

图3多头注意力机制[3]

3.3 conformer模块基本原理

基于Transformer的编码器解码器结构已经成为时间序列相关任务中的主要架构，与传统的RNN和CNN相比，Transformer可以高效的对全局信息进行建模，不存在长时依赖问题，同时其计算方式主要是矩阵乘法，构建简单，并行运算能力强，因此得到越来越多的重视和研究。但是Transformer的长处在于对长序列信息进行建模，对局部特征的提取能力相对较弱。CNN可以实现对局部特征的有效提取，通过局部不变形等特性提高模型鲁棒性和泛化能力，但是单个层的CNN感受野有限，其对全局特征的提取依赖于深度增加带来的感受野的增长，对长序列信息的建模能力有限。

因此Conformer将二者有效结合起来，通过Transformer的注意力机制提取全局特征，通过CNN模块提取局部特征，并在首尾两端各添加一个FFN形成马卡龙形式进行特征提取，其与transformer结构的对比如图4所示。

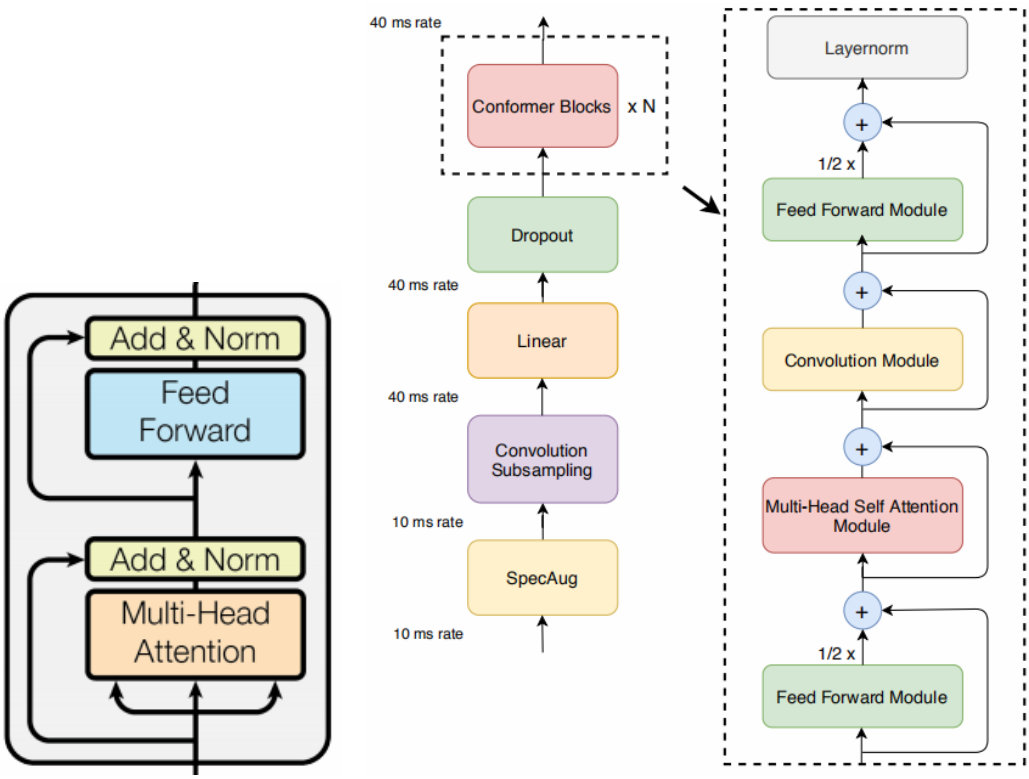

图4Transformer Encoder[3]与Conformer Encoder[4]

其计算方式如下所示，

$$\tilde{x}_i = x_i + \frac{1}{2}\mathrm{FFN}\left(x_i\right) \quad (18)$$

$$x'_i = \tilde{x}_i + \mathrm{MHSA}\left(\tilde{x}_i\right) \quad (19)$$

$$x''_i = x'_i + \mathrm{Conv}\left(x'_i\right) \quad (20)$$

$$y_i = \mathrm{Layernorm}\left(x''_i + \frac{1}{2}\mathrm{FFN}\left(x''_i\right)\right) \quad (21)$$

其中的Convolution Module首先使用一个一维逐点卷积（pointwise convolution）将channel数提升为原本的2倍，并对其使用gated linear unit (GLU)激活函数将channel数重新变为原本的数量，之后分别为一维深度可分离卷积（1-D depthwise convolution layer），批归一化（1-D BatchNorm），Swish激活函数，一维逐点卷积（pointwise convolution）和Dropout，后续步骤中输入和输出的shape一致，其总体架构如图5所示。

4 实验环境

该实验基于huiyuan-dataset数据集，执行run.sh指定训练的起始和终止阶段即可开启训练过程。在该实验中，将详细拆解run.sh每一个步骤的目的和其具体的使用方法，学生需要以此为基础逐步运行，并仔细观察每一步所需的数据和生成的结果，进而掌握完整的模型训练步骤。

该实验使用更容易观察和理解原始音频文件作为训练数据，当进行训练时，需要依据设备情况和需求为该脚本修改部分参数。

如果使用GPU进行模型训练，则需要在此处指定GPU的编号，中间以’,’隔开，例如使用第0和1块GPU卡训练方法如下：

```
export CUDA_VISIBLE_DEVICES="0,1"
```

如果使用CPU进行训练，则指定GPU编号为-1，即：

```
export CUDA_VISIBLE_DEVICES="-1"
```

为了跳过重复的数据准备工作，该脚本将训练分解为6个阶段，通过训练的起始阶段stage和终止阶段stop_stage两个参数进行调控。

```
stage=0 # start from 0 if you need to start from data preparation

stop_stage=5
```

在该实验过程中，将从Stage 0数据准备开始，至Stage 5模型解码测试结束。现将会以s0/run.sh为例对不同的stage分别进行介绍和说明。

实验平台：Linux系统，centos 64, 7.4以上或者ubuntu 64, 18.0以上；
编程环境：python，推荐使用anconda3的环境；在学部平台已为大家配置好python环境和Wenet。
python依赖库如下：

```
Pillow
pyyaml>=5.1
sentencepiece
tensorboard
tensorboardX
typeguard
textgrid
pytest
flake8==3.8.2
flake8-bugbear
flake8-comprehensions
flake8-executable
flake8-pyi==20.5.0
mccabe
pycodestyle==2.6.0
pyflakes==2.2.0
```

5 实验内容

代码路径

```
/root/speech/asr/train_am/
```

在该目录下，wenet文件夹内保存了通过pytorch编写的包括模型架构，训练策略等模型相关代码；tools内中保存了包括数据处理的相关文件；data文件夹下保存了训练所需的数据集；run.sh为训练所需的脚本；conf内保存了该次训练使用的配置文件train_conformer.yaml。wenet和tools部分的代码内容较为复杂，由于课程时间有限，这里将不再对其进行进一步的梳理和分析，如果学生有兴趣，可以自行进行分析和学习。接下来将详细对data文件夹和run.sh进行详细阐述。下文中如无特殊说明，则默认路径为上述代码路径。学生可以根据训练设备和预留的训练时间修改文件train_conformer.yaml，在保证可以完成训练并完成测试的前提下，学生可以调整该文件内的下述参数，训练出可以高性能的模型。

```
encoder_conf:
    output_size: 128    # dimension of attention
    attention_heads: 4
    linear_units: 256  # the number of units of position-wise feed forward
    num_blocks: 6      # the number of encoder blocks
...
decoder_conf:
    attention_heads: 4
    linear_units: 256
    num_blocks: 2
...
dataset_conf:
    batch_conf:
        batch_size: 16 # 训练批次大小
...
max_epoch: 60 # 训练轮数
...
optim_conf:
    lr: 0.001 # 学习率
```

接下来，将以huiyan_speech为例，梳理完整的数据处理和模型训练过程，并进行最终的解码。数据的相对路径为

```
data=data/huiyan_dataset/  # fixme
```

在${data}/raw_wav目录下，包含文件夹wav以及文本文件uttid_wav_text.txt，wav文件夹中保存了训练所需的音频文件。uttid_wav_text.txt中保存了每个wav文件对应的唯一标志uttid、其相对路径和对应的文本信息，中间用\t隔开。

在当前目录下可以通过以下命令查看其前10行

```
head data/huiyan_dataset/raw_wav/uttid_wav_text.txt
```

其对应内容如下所示

```
0001000002	wav/0001/000002.wav	中 央 人 民 广 播 电 台 中 央 人 民 广 播 电 台 全 国 新 闻 联 播
0001000004	wav/0001/000004.wav	全 国 新 闻 联 播 每 天 为 您 汇 总 梳 理 全 天 资 讯 我 是 主 持 人 泰 龙 我 是 玉 蕾 首 先 关 注 今 日 头 条
0001000005	wav/0001/000005.wav	央 广 消 息
0001000006	wav/0001/000006.wav	中 共 中 央 政 治 局 昨 天 下 午 就 我 国 脱 贫 攻 坚 形 式 和 更 好 实 施 精 准 扶 贫
0001000007	wav/0001/000007.wav	进 行 第 三 十 九 次 集 体 学 习
0001000008	wav/0001/000008.wav	中 共 中 央 总 书 记 习 近 平 在 主 持 学 习 时 强 调
0001000009	wav/0001/000009.wav	言 必 信
0001000010	wav/0001/000010.wav	行 必 果
0001000011	wav/0001/000011.wav	农 村 贫 困 人 口 如 期 脱 贫
0001000012	wav/0001/000012.wav	贫 困 县 全 部 摘 帽
```

在开启训练之前，需要依据实际情况修改run.sh的相关配置，其配置阶段主要需要修改下述内容。在后续的训练过程中，如无特殊说明，则出现的路径或文件名均使用该部分参数的引用，不再填写具体的相对或者绝对路径。

检测运行所需必要路径,已在base环境中安装相应依赖，不需要激活环境，直接使用base环境即可。

```
#!/bin/bash
# Copyright 2019 Mobvoi Inc. All Rights Reserved.
. ./path.sh || exit 1;
export PATH=../../../../../experiments/env/wenet/bin:$PATH
```

根据训练情况确定使用的设备，如果有GPU且cuda版本>=9.2，则可以使用GPU进行，否则仅可使用CPU进行训练，CUDA_VISIBLE_DEVICES设置为-1，学部平台只可以使用cpu。

```
# Use this to control how many gpu you use, It's 1-gpu training if you specif
# just 1gpu, otherwise it's is multiple gpu training based on DDP in pytorch
export CUDA_VISIBLE_DEVICES="-1"
# The NCCL_SOCKET_IFNAME variable specifies which IP interface to use for ncc
```

```
# communication. More details can be found in
# https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html
# export NCCL_SOCKET_IFNAME=ens4f1
export NCCL_DEBUG=INFO
```

确定训练的起始和终止阶段，其具体内容将在下面进行详细阐述。请同学们修改run.sh中的参数，逐步进行调试，每次进行一个stage，并观察每个stage的输出。

```
stage=0 # start from 0 if you need to start from data preparation
stop_stage=0
```

确定训练数据${data}和对应的字典${dict}，数据的格式${data_type}，这里使用了原始音频作为训练数据，因此调整${data_type}为raw。

```
# The num of machines(nodes) for multi-machine training, 1 is for one machin
# NFS is required if num_nodes > 1.
num_nodes=1

# The rank of each node or machine, which ranges from 0 to `num_nodes - 1`.
# You should set the node_rank=0 on the first machine, set the node_rank=1
# on the second machine, and so on.
node_rank=0
# The aishell dataset location, please change this to your own path
# make sure of using absolute path. DO-NOT-USE relatvie path!
data=data/huiyan_dataset  # fixme

nj=16
dict=${data}/lang_char.txt

# data_type can be `raw` or `shard`. Typically, raw is used for small datase
# `shard` is used for large dataset which is over 1k hours, and `shard` is
# faster on reading data and training.
data_type=raw
num_utts_per_shard=1000

train_set=train
```

之后确定训练使用的配置文件${train_config}，cmvn配置${cmvn}，训练结果保存路径${dir}，以及使用断点重连的模型路径${checkpoint}。这里使用了cmvn，因此设置为true，否则为false。

```
# Optional train_config
# 1. conf/train_transformer.yaml: Standard transformer
```

```
# 2. conf/train_conformer.yaml: Standard conformer
# 3. conf/train_unified_conformer.yaml: Unified dynamic chunk causal conform
# 4. conf/train_unified_transformer.yaml: Unified dynamic chunk transformer
# 5. conf/train_u2++_conformer.yaml: U2++ conformer
# 6. conf/train_u2++_transformer.yaml: U2++ transformer
train_config=conf/train_conformer.yaml
cmvn=true
dir=exp/conformer
checkpoint=
```

在完成训练后，需要进行测试，其相关配置为是否使用模型权重平均技术${average_checkpoint}，如果设置为true，则会选择最后${average_num}个模型对权重求平均，并将模型保存为${decode_checkpoint}。最终将分别使用${decode_modes}进行解码。

```
# use average_checkpoint will get better result
average_checkpoint=true
decode_checkpoint=$dir/final.pt
average_num=5
decode_modes="ctc_greedy_search ctc_prefix_beam_search attention
attention_rescoring"

. tools/parse_options.sh || exit 1;
```

执行下述命令即可开启训练过程，训练记录将被保存在train.log中。

```
./run.sh >train.log
```

5.1 准备训练数据

对应代码中的stage0，该阶段调用tools/data_split.py对数据进行初步的处理。

如果需要单独执行该阶段代码，需要执行

```
./run.sh  --stage 0 --stop-stage 0
```

该阶段对应代码为：

```
if [ ${stage} -le 0 ] && [ ${stop_stage} -ge 0 ]; then
  # Data preparation
  python3 tools/data_split.py ${data}/raw_wav/uttid_wav_text.txt \
  0.05 ${data} false
fi
```

该阶段将数据${data}/raw_data/uttid_wav_text.txt随机区分为训练集${data}/train，验证集${data}/dev和测试集${data}/test，后二者比例为0.05，其余均为训练集。切分结果被保存到${data}，每个文件夹下包括wav.scp和text两个文件。为了方便理解，这里设置随机切分为false，以便生成统一的文本内容和字典内容。

wav.scp每行保存了两个信息 wav_id 和 wav_path，中间用空格隔开。

以训练集为例，在当前目录下通过下述命令观察前十行

```
head data/huiyan_dataset/train/wav.scp
```

基本格式如下

```
0003000333 data/huiyan_dataset/raw_wav/wav/0003/000333.wav
0004000255 data/huiyan_dataset/raw_wav/wav/0004/000255.wav
0005000068 data/huiyan_dataset/raw_wav/wav/0005/000068.wav
0003000486 data/huiyan_dataset/raw_wav/wav/0003/000486.wav
0003000241 data/huiyan_dataset/raw_wav/wav/0003/000241.wav
0002000087 data/huiyan_dataset/raw_wav/wav/0002/000087.wav
0004000190 data/huiyan_dataset/raw_wav/wav/0004/000190.wav
0002000318 data/huiyan_dataset/raw_wav/wav/0002/000318.wav
0003000328 data/huiyan_dataset/raw_wav/wav/0003/000328.wav
0001000222 data/huiyan_dataset/raw_wav/wav/0001/000222.wav
```

text 每行保存了两个信息 wav_id 和 text_label，中间用空格隔开。通过下述命令观察的前十行

```
head data/huiyan_dataset/train/text
```

基本格式如下

```
0017000214 四要严格依法依规运营
0012000354 自美国与俄罗斯二月二十二号就叙利亚冲突各方停火达成协议以来
0011000338 最高法副院长陶凯元表示
0012000064 美俄外长就叙利亚停火达成一致中方呼吁各方为政治解决叙利亚问题营造积极氛围
0035000294 文朝荣还带头摸索种植中药材
0036000453 十七号有超过二百架次航班取消
0004000006 胡锦涛温家宝发表重要讲话吴邦国贾庆林李长春习近平李克强贺国强周永康出席
0016000217 山西警方追逃小组日前将外逃两年多涉嫌掩饰隐瞒犯罪所得收益罪已被国际刑警组织通缉的犯罪嫌疑人王某押解归案
0022000121 伊朗外交部十九号表示欧洲国家至今未能履行在伊朗核问题全面协议中的相关承诺伊朗正准备进一步减少履行伊核协议义务伊朗外交部发言人穆萨维表示仍有欧洲国家正在进行外交努力
```

```
10 0017000071 就是个别突出的单品可能就一两个单品本身市场进货的价格就是按照这个价格基本上是不赚钱在卖使用softmax进行归一化处理，计算注意力分布，注意力分布可以解释为在给定任务相关的查询向量时，对值向量的关注程度，计算方式如下所示。
```

5.2 提取CMVN特征

对应代码中的Stage 1，该阶段将会对stage0中的text进行进一步处理，将中文字符之间的空格去除。

如果已经设置cmvn配置${cmvn}为true，则将以训练集数据为基准，执行tools/compute_cmvn_stats.py提取倒谱均值和方差归一化global_cmvn，这些统计数据将被用来规范声学特征，并将其保存在${data}/${train_set}/global_cmvn。

如果需要单独执行代该阶段代码，需要执行

```
1 ./run.sh  --stage 1 --stop-stage 1
```

该阶段对应代码为

```
1  if [ ${stage} -le 1 ] && [ ${stop_stage} -ge 1 ]; then
2    # remove the space between the text labels for Mandarin dataset
3    for x in train dev test; do
4      cp ${data}/${x}/text ${data}/${x}/text.org
5      paste -d " " <(cut -f 1 -d" " ${data}/${x}/text.org) \
6        <(cut -f 2- -d" " ${data}/${x}/text.org | tr -d " ") \
7        > ${data}/${x}/text
8      rm ${data}/${x}/text.org
9    done
10
11   python tools/compute_cmvn_stats.py --num_workers 16 --train_config
   $train_config \
12     --in_scp ${data}/${train_set}/wav.scp \
13     --out_cmvn ${data}/${train_set}/global_cmvn
14 fi
```

5.3 生成字典

对应代码中的Stage 2，该阶段将以训练集对应的文本作为基准调用tools/text2token.py生成字典$dict。

如果需要单独执行代该阶段代码，需要执行

```
1 ./run.sh  --stage 2 --stop-stage 2
```

该阶段对应代码为

```
if [ ${stage} -le 2 ] && [ ${stop_stage} -ge 2 ]; then
  echo "Make a dictionary"
  mkdir -p $(dirname $dict)
  echo "<blank> 0" > ${dict}  # 0 is for "blank" in CTC
  echo "<unk> 1"  >> ${dict}  # <unk> must be 1
  python3 tools/text2token.py -s 1 -n 1 ${data}/train/text \
  | cut -f 2- -d" " \
  | tr " " "\n" | sort | uniq | grep -a -v -e '^\s*$' | \
    awk '{print $0 " " NR+1}' >> ${dict}
  num_token=$(cat $dict | wc -l)
  echo "<sos/eos> $num_token" >> $dict
fi
```

字典实现了从建模单元（在huiyan_dataset中使用汉字作为建模单元）向索引的映射，辅助Stage 3生成data.list文件。

可以通过下述命令观察其相关信息

```
cat data/huiyan_dataset/lang_char.txt
```

$dict基本内容如下所示：

```
<blank> 0
<unk> 1
' 2
* 3
+ 4
5 5
a 6
A 7
b 8
B 9
c 10
...
鼻 2900
齐 2901
龄 2902
龙 2903
龚 2904
<sos/eos> 2905
```

其中<blank>表示CTC中的blank字符；<unk>定义不存在于字典中的未知字符，所有OOV(out-of-vocabulary)字符都将被映射为<unk>；<sos/eos>定义了起始字符和终止字符，该字符应用于decoder的重打分机制和训练，二者共享同一个索引。

5.4 生成应用于训练的数据格式

对应代码中的Stage 3，该阶段将数据准备阶段生成的wav.scp和text转换为wenet所需的数据格式，并将其保存在${data}/$x/data.list内。

如果需要单独执行代该阶段代码，需要执行

```
./run.sh  --stage 3 --stop-stage 3
```

该阶段对应代码为

```
if [ ${stage} -le 3 ] && [ ${stop_stage} -ge 3 ]; then
  echo "Prepare data, prepare requried format"
  for x in dev test ${train_set}; do
    if [ $data_type == "shard" ]; then
      python tools/make_shard_list.py --num_utts_per_shard $num_utts_per_sha
        --num_threads 16 ${data}/$x/wav.scp ${data}/$x/text \
        $(realpath ${data}/$x/shards) ${data}/$x/data.list
    else
      python tools/make_raw_list.py ${data}/$x/wav.scp ${data}/$x/text \
        ${data}/$x/data.list
    fi
  done
```

data.list中的每一行将以字典的形式包含key，wav_path，txt三部分内容，其中

key: 样本的唯一标识符。

wav: 音频文本的路径。

txt:依然是文本的形式，在训练阶段将会依照字典自动转换为对应的索引。

以训练集train为例，通过下述命令打开data.list观察前十行

```
head data/huiyan_dataset/train/data.list
```

显示内容如下（由于随机结果不同显示内容不同，但是格式是一致的）

```
step5: 将H个0018000281", "wav":
"data/huiyan_dataset/raw_wav/wav/0018/000281.wav", "txt": "中国之声正在直播稍
```

```
后央广新闻晚高峰为您带来去年国人买走全球百分之四十六的奢侈品"}
{"key": "0006000237", "wav": "data/huiyan_dataset/raw_wav/wav/0006/000237.wav", "txt": "因为呢这条道路呢是有这个视频监控设备对违法停车进行监控的"}
{"key": "0029000060", "wav": "data/huiyan_dataset/raw_wav/wav/0029/000060.wav", "txt": "今年八十一岁的中国科学院院士汪尔康从事分析化学电分析化学研究达六十余年为我国电分析化学研究跻身世界前列作出了杰出贡献"}
{"key": "0030000071", "wav": "data/huiyan_dataset/raw_wav/wav/0030/000071.wav", "txt": "今晚七点三十八焦点访谈将播出依法治国的历史性跨越第三集解读中国的依宪执政与西方的所谓宪政是截然不同的"}
{"key": "0027000047", "wav": "data/huiyan_dataset/raw_wav/wav/0027/000047.wav", "txt": "面对国际地区局势的深刻变化双方要坚持睦邻友好增进战略互信更有效地维护共同利益确保两国关系长期稳定健康发展温家宝指出深化东亚经济融合具有重要的现实和战略意义"}
{"key": "0012000213", "wav": "data/huiyan_dataset/raw_wav/wav/0012/000213.wav", "txt": "各级卫生计生行政部门要进一步加强对违规医疗行为的监管和打击力度"}
业局的冲突，和为可学习参数，通过线性转换将不同注意力头之间的信息进行混合
```

5.5 训练神经网络模型

对应代码中的Stage 4，该阶段将使用stage3生成的data.list进行模型训练。

如果需要单独执行代该阶段代码，需要执行

```
./run.sh  --stage 4 --stop-stage 4
```

该阶段对应代码如下。

需要同学们根据实验硬件设置相应的CUDA_VISIBLE_DEVICES变量。

```
if [ ${stage} -le 4 ] && [ ${stop_stage} -ge 4 ]; then
  echo "begin"
  mkdir -p $dir
  # You have to rm `INIT_FILE` manually when you resume or restart a
  # multi-machine training.
  INIT_FILE=$dir/ddp_init
  init_method=file://$(readlink -f $INIT_FILE)
  echo "$0: init method is $init_method"
  num_gpus=$(echo $CUDA_VISIBLE_DEVICES | awk -F "," '{print NF}')
  echo $num_gpus
  # Use "nccl" if it works, otherwise use "gloo"
  dist_backend="gloo"
```

```
  world_size=`expr $num_gpus \* $num_nodes`
  echo "total gpus is: $world_size"
  cmvn_opts=
  $cmvn && cp ${data}/${train_set}/global_cmvn $dir
  $cmvn && cmvn_opts="--cmvn ${dir}/global_cmvn"

  # train.py rewrite $train_config to $dir/train.yaml with model input
  # and output dimension, and $dir/train.yaml will be used for inference
  # and export.
  for ((i = 0; i < $num_gpus; ++i)); do
  {
    gpu_id=$(echo $CUDA_VISIBLE_DEVICES | cut -d',' -f$[$i+1])
    # Rank of each gpu/process used for knowing whether it is
    # the master of a worker.
    rank=`expr $node_rank \* $num_gpus + $i`
    python /root/Desktop/wenet/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
  } &
  done
  wait
fi
```

可以设置在指定的多个GPU上分别开始训练进程

```
  for ((i = 0; i < $num_gpus; ++i)); do
  {
    gpu_id=$(echo $CUDA_VISIBLE_DEVICES | cut -d',' -f$[$i+1])
    # Rank of each gpu/process used for knowing whether it is
    # the master of a worker.
```

```
    rank=`expr $node_rank \* $num_gpus + $i`
    python3 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
  } &
  done
  wait
fi
```

参数说明：

```
-config：指向一个yaml模型配置文件。
在该文件下可以指定模型配置参数、数据集参数和学习率等训练参数。
在conf/文件夹下，提供了一系列的默认配置，
可以参考conf/train_conformer.yaml作为调整参数的依据。
-train_data：训练集路径，指向通过stage3生成的format.data。
-cv_data：验证集路径，指向通过stage3生成的format.data。
-checkpoint：可以为空。
如果想要利用断点重连技术在已有模型或者继续原本断开的模型进行训练，
指向已经训练好的模型即可。
-model_dir：保存训练好的模型和相关位置的路径。
-ddp.init_method：DDP初始化方法，参见pytorch官方文档。
-ddp.world_size：DDP使用的总的GPU的数量。
-ddp.rank：当前进程的编号。
-ddp.dist_backend：使用的dist_backend。
如果是使用多GPU进行DDP模式(DistributedDataParallel)进行训练，
建议设置 dist_backend="nccl"选择nccl作为dist_backend,该方式下训练速度更优。
如果nccl不可用，则可调整 dist_backend="gloo" 选择gloo作为dist_backend。
-num_workers：读取数据的进程数。
-cmvn_opts：使用的CMVN路径。
-pin_memory：是否读取数据时使用缓存机制。
```

该训练过程将会持续几个小时，这依赖于使用的设备配置。中间的训练过程可以通过train.log进行观察：

```
cat train.log
```

5.6 使用训练好的模型进行识别

对应代码中的Stage 5，该阶段将调用stage4训练好的模型利用wenet/bin/recognize.py对测试集数据进行解码，并利用tools/compute-wer.py计算对应的CER。

如果需要单独执行代该阶段代码，需要执行

```
./run.sh  --stage 5 --stop-stage 5
```

该阶段对应代码如下。

如果设置的average_checkpoint=true，则会对保存stage4训练结果的文件夹src_path中的模型调用wenet/bin/average_model.py进行参数求平均，提高模型鲁棒性，防止过拟合现象的发生，并将结果保存在$decode_checkpoint下。

```
if [ ${stage} -le 5 ] && [ ${stop_stage} -ge 5 ]; then
  # Test model, please specify the model you want to test by --checkpoint
  if [ ${average_checkpoint} == true ]; then
    decode_checkpoint=$dir/avg_${average_num}.pt
    echo "do model average and final checkpoint is $decode_checkpoint"
    python wenet/bin/average_model.py \
      --dst_model $decode_checkpoint \
      --src_path $dir  \
      --num ${average_num} \
      --val_best
  fi
```

通过wenet/bin/recognize.py即可进行模型识别。

```
  decoding_chunk_size=
  ctc_weight=0.5
  reverse_weight=0.0
  for mode in ${decode_modes}; do
  {
    test_dir=$dir/test_${mode}
```

```
    mkdir -p $test_dir
    python3 wenet/bin/recognize.py --gpu -1 \
      --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}
```

recognize.py参数说明

```
-gpu：使用的gpu的编号，如果不使用则调整为-1，即为使用cpu进行解码。
-mode：解码使用的模式。
可以是ctc_greedy_search，ctc_prefix_beam_search，
attention，attention_rescoring之一。
一般来说，attention_resscoring解码效果最好。
-ctc_greedy_search : encoder + CTC greedy search。
-ctc_prefix_beam_search : encoder + CTC prefix beam search。
-attention : encoder + attention-based decoder decoding。
-attention_rescoring : rescoring the ctc candidates from
ctc prefix beam search with encoder output on attention-based decoder。
-config：模型配置文件。
注意这一文件并非是训练时使用的配置文件，
而是在训练时生成的被保存在和模型相同目录下的train.yaml文件。
-test_data：测试集对应的format.data。
-checkpoint：训练好的模型路径。

```

```
-beam_size：束解码使用的beam size。
一般而言该参数设置越大，识别结果约好，但是计算量和耗时也随之上升。
-batch_size：每次进行解码的样本数量。
如果mode设置为ctc_prefix_beam_search或者 attention_rescoring则必须为1。
-penalty：对长度的惩罚项。
-dict：stage3使用的字典，用于将索引转换为字符。
-ctc_weight：当使用attention_rescoring时该参数生效。
用于决定该阶段下的CTC权重。
```

```
-reverse_weight: 使用双向重打分机制时该参数生效，决定反向decoder的评分的权重。
-result_file: 保存解码结果的文件路径。
-decoding_chunk_size: 解码时使用的chunk size。
当设置为-1时为非流式解码，大于0的正整数为流式解码。
```

最终得到识别结果后调用compute-wer.py计算CER。需要为其传递三个位置参数，第一个参数为我们的测试集对应的text文件（参见stage2）${data}/test/text，第二个参数为recognize.py生成的解码结果$test_dir/text，第三个参数为保存计算CER的结果的文件$test_dir/wer。

```
    python3 tools/compute-wer.py --char=1 --v=1 \
      ${data}/test/text $test_dir/text > $test_dir/wer
```

6 实验结果

在解码完成后，请提交对应的配置文件train.yaml和对应的解码结果（CER）

解码方式	解码结果（CER）
attention	Overall -> Mandarin -> Other -> English ->
ctc_greedy_search	Overall -> Mandarin -> Other -> English ->
ctc_prefix_beam_search	Overall -> Mandarin -> Other -> English ->
attention_rescoring	Overall -> Mandarin -> Other -> English ->

7 实验思考

1.常见的导致训练不收敛的原因有哪些?

2.训练过程中发现模型在训练集上的损失值远低于测试集上的损失值，训练集的损失值虽然在下降，但是测试集的损失值呈现上升趋势，这表明什么，该如何解决?

3.梯度消失和梯度爆炸的问题是如何产生的？如何解决?

8 参考文献

[1] 邱锡鹏.《神经网络与深度学习》[M], 机械工业出版社, 2020年；
[2] Zhang A, Li Mu，Lipton C Z, et al.《动手学深度学习》[M], 人民邮电出版社, 2019年；