

基于 MindSpore 框架的 DeepSpeech2 语音识别 模型



华为技术有限公司

版权所有 © 华为技术有限公司 2022。保留一切权利。

非经本公司书面许可，任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部，并不得以任何形式传播。

商标声明



和其他华为商标均为华为技术有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司	
地址：	深圳市龙岗区坂田华为总部办公楼 邮编：518129
网址：	http://e.huawei.com

目录

1 实验介绍	2
1.1 实验背景	2
1.2 实验目的	2
1.3 实验清单	2
1.4 实验开发环境	2
1.5 开发平台介绍	3
2 DeepSpeech2 实现语音识别任务	4
2.1 环境准备	4
2.2 代码及数据下载	4
2.2.1 获取代码文件	4
2.2.2 脚本说明	4
2.2.3 下载 LibriSpeech 数据集	5
2.3 数据预处理	6
2.3.1 安装 python3.9.0	6
2.3.2 安装 MindSpore 和所需要的依赖包	7
2.3.3 下载数据预处理 SeanNaren 脚本	7
2.3.4 LibriSpeech 数据预处理	8
2.4 模型训练与评估	10
2.4.1 模型训练	10
2.4.2 模型评估	18
2.4.3 思考题	19
2.5 实验小结	20
3 附录：开发环境搭建	21
3.1 ECS 弹性服务器	21
3.2 MobaXterm 连接 ECS	24
3.3 FinalShell 连接 ECS	26
4 问题 FAQ	28
4.1 报错“ No modul named '_bz2' ”	28
4.2 没有_lzma 模块	29

1 实验介绍

1.1 实验背景

DeepSpeech2 是一个使用 CTC 损失训练的语音识别模型。它用神经网络取代了整个手工设计的管道，可以处理各种各样的语音，包括嘈杂的环境、口音和不同的语言。

论文: [Amodei, Dario, et al. Deep speech 2: End-to-end speech recognition in english and mandarin.](#)

1.2 实验目的

- 熟悉 MindSpore 训练网络的流程。
- 熟悉 MindSpore 搭建 DeepSpeech2 网络训练和评估过程。
- 了解 Linux 操作命令。

1.3 实验清单

实验	简述	难度	软件环境	开发环境
基于 MindSpore 框架 deepspeech2 语音识别任务	本实验用 LibriSpeech 数据集，实现 MindSpore 在语音识别应用	高级	Python3.9.0、MindSpore1.6、	ECS

1.4 实验开发环境

- 实验平台：ECS
- 框架：MindSpore1.6
- 硬件：CPU

1.5 开发平台介绍

MindSpore 是一种适用于端边云场景的新型开源深度学习训练/推理框架。MindSpore 提供了友好的设计和高效的执行，旨在提升数据科学家和算法工程师的开发体验，并为 Ascend AI 处理器提供原生支持，以及软硬件协同优化。

同时，MindSpore 作为全球 AI 开源社区(<https://www.mindspore.cn/>)，致力于进一步开发和丰富 AI 软硬件应用生态。

2 DeepSpeech2 实现语音识别任务

2.1 环境准备

参考附录完成 ECS（通用计算规格）资源的购买和远程登陆，以创建开发环境。

2.2 代码及数据下载

1.1.1 获取代码文件

使用 git 从 mindspore 下载训练脚本的源码，在 MobaXterm 连接的服务器后，切换到 home 目录，创建工作目录如/work，执行命令如下：

```
git clone https://gitee.com/mindspore/models.git
```

本实验 deepspeech2 项目代码位于 models/official/audio/deepspeech2。

2.2.1 脚本说明

2.2.1.1 脚本脚本及样例代码

```
├─ audio
│   └─ deepspeech2
│       ├── scripts
│       │   ├── run_distribute_train_gpu.sh // gpu8 卡训练脚本
│       │   ├── run_eval_cpu.sh           // cpu 推理脚本
│       │   ├── run_eval_gpu.sh           // gpu 推理脚本
│       │   ├── run_standalone_train_cpu.sh // cpu 单卡训练脚本
│       │   └── run_standalone_train_gpu.sh // gpu 单卡训练脚本
│       ├── train.py                      // 训练文件
│       ├── eval.py                      // 推理文件
│       ├── export.py                    // 将 mindspore 模型转换为 mindir 模型
│       ├── labels.json                  // 可能映射到的字符
│       ├── README.md                   // DeepSpeech2 相关描述
│       ├── deepspeech_pytorch           //
│       │   └── decoder.py               // 来自第三方代码的解码器（MIT 许可证）
│       └─ src
│           ├── __init__.py
│           ├── DeepSpeech.py            // DeepSpeech2 网络架构
│           ├── dataset.py               // 数据处理
│           ├── config.py                // DeepSpeech 配置文件
│           ├── lr_generator.py          // 产生学习率
│           └── greedydecoder.py         // 修改 Mindspore 代码的 greedydecoder
```

```
└─callback.py          // 回调以监控训练
```

2.2.1.2 训练和推理相关参数在 config.py 文件

训练相关参数	
epochs	训练的 epoch 数量，默认为 70
数据处理相关参数	
train_manifest	用于训练的数据文件路径，默认为 'data/libri_train_manifest.csv'
val_manifest	用于测试的数据文件路径，默认为 'data/libri_val_manifest.csv'
batch_size	批处理大小，默认为 8
labels_path	模型输出的 token json 路径，默认为 './labels.json'
sample_rate	数据特征的采样率，默认为 16000
window_size	频谱图生成的窗口大小（秒），默认为 0.02
window_stride	频谱图生成的窗口步长（秒），默认为 0.01
window	频谱图生成的窗口类型，默认为 'hamming'
speed_volume_perturb	使用随机速度和增益扰动，默认为 False，当前模型中未使用
spec_augment	在 MEL 谱图上使用简单的光谱增强，默认为 False，当前模型中未使用
noise_dir	注入噪音到音频。默认为 noise Inject 未添加，默认为 ''，当前模型中未使用
noise_prob	每个样本加噪声的概率，默认为 0.4，当前模型中未使用
noise_min	样本的最小噪音水平，(1.0 意味着所有的噪声，不是原始信号)，默认是 0.0，当前模型中未使用
noise_max	样本的最大噪音水平。最大值为 1.0，默认值为 0.5，当前模型中未使用
模型相关参数	
rnn_type	模型中使用的 RNN 类型，默认为 'LSTM'，当前只支持 LSTM
hidden_size	RNN 层的隐藏大小，默认为 1024
hidden_layers	RNN 层的数量，默认为 5
lookahead_context	查看上下文，默认值是 20，当前模型中未使用
优化器相关参数	
learning_rate	初始化学习率，默认为 3e-4
learning_anneal	对每个 epoch 之后的学习率进行退火，默认为 1.1
weight_decay	权重衰减，默认为 1e-5
momentum	动量，默认为 0.9
eps	Adam eps，默认为 1e-8
betas	Adam betas，默认为 (0.9, 0.999)
loss_scale	损失规模，默认是 1024
checkpoint 相关参数	
ckpt_file_name_prefix	ckpt 文件的名称前缀，默认为 'DeepSpeech'
ckpt_path	ckpt 文件的保存路径，默认为 'checkpoints'
keep_checkpoint_max	ckpt 文件的最大数量限制，删除旧的检查点，默认是 10
	训练的 epoch 数量，默认为 70

2.2.2 下载 LibriSpeech 数据集

下载数据集的链接为：<http://www.openslr.org/12>

训练集：

- train-clean-100: [6.3G] (100 小时的无噪音演讲训练集) (只要下载这个文件)
- train-clean-360.tar.gz [23G] (360 小时的无噪音演讲训练集) (不用下载)
- train-other-500.tar.gz [30G] (500 小时的有噪音演讲训练集) (不用下载)

验证集：

- dev-clean.tar.gz [337M] (无噪音)
- dev-other.tar.gz [314M] (有噪音)

测试集：

- test-clean.tar.gz [346M] (测试集, 无噪音)
- test-other.tar.gz [328M] (测试集, 有噪音)

数据格式：wav 和 txt 文件

注意：数据需要通过 librispeech.py 进行处理

LibriSpeech 数据目录结构,如下：

```

|— L LibriSpeech
    |— train
        |— train-clean-100
    |— val
        |— dev-clean.tar.gz
        |— dev-other.tar.gz
    |— test_other
        |— test-other.tar.gz
    |— test-clean
        |— test-clean.tar.gz
    
```

2.3 数据预处理

2.3.1 安装 python3.9.0

步骤 1 安装 python 依赖以及 gcc 等软件

```

sudo apt-get install -y gcc g++ make cmake zlib1g zlib1g-dev openssl libsqlite3-dev libssl-dev
libffi-dev unzip pciutils net-tools libblas-dev gfortran libblas3 libopenblas-dev libgmp-dev sox
libjpeg8-dev
    
```

步骤 2 检查系统是否安装 python3.9.0 开发环境。

- 开发套件包依赖 python 环境，分别使用命令 **python3.9 --version**、**pip3.9 --version** 检查是否已经安装，如果返回如下信息则说明已经安装，进入下一步。(需要学生安装)

```

Python 3.9.0
pip 19.2.3 from /usr/local/python3.9.0/lib/python3.9/site-packages/pip (python 3.9)
    
```

- 使用 wget 下载 python3.9.0 源码包，可以下载到安装环境的任意目录，命令为：

```

wget https://www.python.org/ftp/python/3.9.0/Python-3.9.0.tgz
tar -zxvf Python-3.9.0.tgz
    
```

- 进入解压后的文件夹，执行配置、编译和安装命令：

```

cd Python-3.9.0
chmod +x configure # configure 文件添加可执行权限
./configure --prefix=/usr/local/python3.9.0 --enable-shared
make
    
```



```
sudo make install
```

- 查询/usr/lib64 或/usr/lib 下是否有 libpython3.9.so.1.0，若有则跳过此步骤或将系统自带的 libpython3.9.so.1.0 文件备份后执行如下命令（需要学生执行）：

```
cp /usr/local/python3.9.0/lib/libpython3.9.so.1.0 /usr/lib
```

- 执行如下命令设置软链接：

```
sudo ln -s /usr/local/python3.9.0/bin/python3.9 /usr/bin/python
sudo ln -s /usr/local/python3.9.0/bin/pip3.9 /usr/bin/pip
sudo ln -s /usr/local/python3.9.0/bin/python3.9 /usr/bin/python3.9
sudo ln -s /usr/local/python3.9.0/bin/pip3.9 /usr/bin/pip3.9
```

- 安装完成之后，执行如下命令查看安装版本，如果返回相关版本信息，则说明安装成功。

```
python3.9 --version
pip3.9 --version
```

2.3.2 安装 MindSpore 和所需要的依赖包

- 安装 mindspore，根据服务器实际架构进行安装，请参考 <https://www.mindspore.cn/install>。

```
pip install https://ms-release.obs.cn-north-4.myhuaweicloud.com/1.6.0/MindSpore/cpu/x86_64/
mindspore-1.6.0-cp39-cp39-linux_x86_64.whl \
--trusted-host ms-release.obs.cn-north-4.myhuaweicloud.com \
-i https://pypi.tuna.tsinghua.edu.cn/simple
```

- Pip 源安装, 依赖包文件较大时可以添加镜像源安装，例如：pip install -i https://pypi.tuna.tsinghua.edu.cn/simple sox。

```
pip3.9 install wget
pip3.9 install tqdm
pip3.9 install sox
```

2.3.3 下载数据预处理 SeanNaren 脚本

- 在 MobaXterm/Finalshell（推荐）连接 ECS 的服务器后，切换到 home 目录，创建工作目录，然后使用 SeanNaren 中的脚本来处理数据。
- SeanNaren 脚本链接：<https://github.com/SeanNaren/deepspeech.pytorch>

```
cd ../home
mkdir work
cd work
#需要访问外网，建议直接下载到本地然后上传 zip 文件再解压
git clone https://github.com/SeanNaren/deepspeech.pytorch.git
```

```
drwxr-xr-x 4 root root 4096 Feb 21 11:27 ../
root@ecs-32c9:/home/work# git clone https://github.com/SeanNaren/deepspeech.pytorch.git
Cloning into 'deepspeech.pytorch'...
remote: Enumerating objects: 2099, done.
remote: Counting objects: 100% (74/74), done.
remote: Compressing objects: 100% (57/57), done.
remote: Total 2099 (delta 33), reused 38 (delta 15), pack-reused 2025
Receiving objects: 100% (2099/2099), 769.60 KiB | 2.07 MiB/s, done.
Resolving deltas: 100% (1307/1307), done.
root@ecs-32c9:/home/work# ll
total 12
drwxr-xr-x 3 root root 4096 Feb 21 11:30 ./
drwxr-xr-x 4 root root 4096 Feb 21 11:27 ../
drwxr-xr-x 9 root root 4096 Feb 21 11:30 deepspeech.pytorch/
```

图2.3.3.1.1.1.1.1 数据预处理脚本下载

2.3.4 LibriSpeech 数据预处理

- 步骤 1 本地通过数据集链接 <http://www.openslr.org/12> 下载 train-clean-100 的训练集，验证集 dev-clean.tar.gz 和 dev-other.tar.gz，测试集 test-clean.tar.gz 和 test-other.tar.gz。
- 步骤 2 上传本地数据集到 MobaXterm 服务器上，通过下图的上传按钮进行上传。

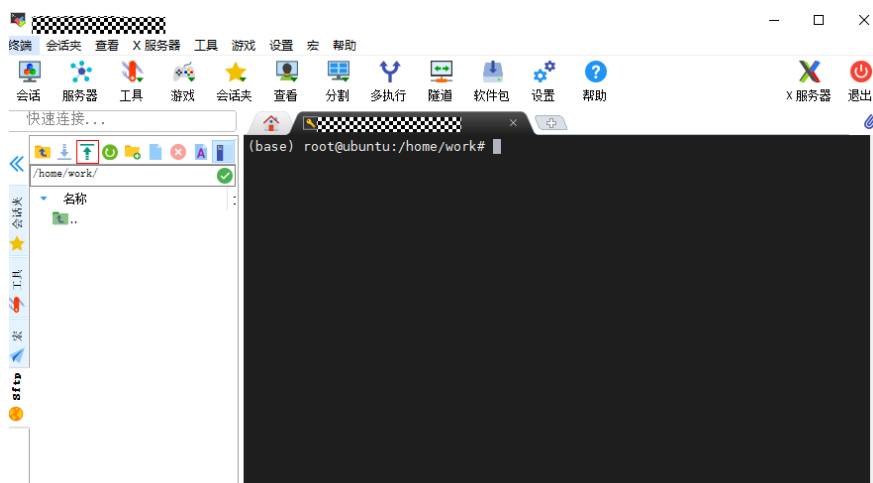


图2.3.4.1.1.2.1.1 上传本地数据集

- 步骤 3 数据集结构如下图：

```
root@ecs-32c9:/home/work/deepspeech.pytorch# tree LibriSpeech_dataset/
LibriSpeech_dataset/
├── test_clean
│   └── test-clean.tar.gz
├── test_other
│   └── test-other.tar.gz
├── train
│   └── train-clean-100.tar.gz
└── val
    ├── dev-clean.tar.gz
    └── dev-other.tar.gz

4 directories, 5 files
root@ecs-32c9:/home/work/deepspeech.pytorch#
```

图2.3.4.1.1.3.1.1 原始数据集结构图

步骤 4 把 deepspeech.pytorch 的 data 目录下的 librispeech.py 复制到 deepspeech.pytorch 目录下，执行命令如下：

```
cd deepspeech.pytorch
cp ./data/librispeech.py ./
```

步骤 5 修改 librispeech.py 代码数据集路径，参照步骤三在当前目录下设置目录结构，代码路径改为数据集实际路径，如下图所示：

```
LIBRI_SPEECH_URLS = {
    "train": ["LibriSpeech_dataset/train/train-clean-100.tar.gz"],
    "val": ["LibriSpeech_dataset/val/dev-clean.tar.gz",
           "LibriSpeech_dataset/val/dev-other.tar.gz"],
    "test_clean": ["LibriSpeech_dataset/test_clean/test-clean.tar.gz"],
    "test_other": ["LibriSpeech_dataset/test_other/test-other.tar.gz"]
}
```

图2.3.4.1.1.5.1.1 代码数据集路径

步骤 6 执行数据集处理命令，执行命令如下

```
python librispeech.py
```

步骤 7 进行数据处理完成后，数据目录结构如下：

```
.
├── LibriSpeech_dataset
│   ├── train
│   │   ├── wav
│   │   └── txt
│   ├── val
│   │   ├── wav
│   │   └── txt
│   ├── test_clean
│   │   ├── wav
│   │   └── txt
│   └── test_other
│       ├── wav
│       └── txt
└── libri_test_clean_manifest.json, libri_test_other_manifest.json, libri_train_manifest.json,
    libri_val_manifest.json
```

步骤 8 json 文件转 csv 文件，在 deepspeech.pytorch 目录下创建 json_to_csv.py，并把代码复制到文件，代码如下：

```
#创建文件
touch json_to_csv.py
```

```
import json
import csv
import argparse

parser = argparse.ArgumentParser(description='Image classification')
parser.add_argument("--json", type=str, default="", help="")
parser.add_argument("--csv", type=str, default="", help="")
config = parser.parse_args()

def trans(jsonfile, csvfile):
    jsonData = open(jsonfile)
    csvfile = open(csvfile, "a")
    for i in jsonData:
        dic = json.loads(i[0:])
        root_path = dic["root_path"]
        for j in dic["samples"]:
            wav_path = j["wav_path"]
            transcript_path = j["transcript_path"]
            res_wav = root_path + '/' + wav_path
            res_txt = root_path + '/' + transcript_path
            res = [res_wav, res_txt]
            writer = csv.writer(csvfile)
            writer.writerow(res)
    jsonData.close()
    csvfile.close()

if __name__ == "__main__":
    trans(config.json, config.csv)
```

步骤 9 json_to_csv 文件参数,参数说明 :

- --json : 数据集 json 文件的实际路径
- --csv: 数据集 csv 文件路径

运行命令如下图所示 :

```
#创建文件
python json_to_csv.py --json libri_test_clean_manifest.json --csv libri_test_clean_manifest.csv
python json_to_csv.py --json libri_test_other_manifest.json --csv libri_test_other_manifest.csv
python json_to_csv.py --json libri_train_manifest.json --csv libri_train_manifest.csv
python json_to_csv.py --json libri_val_manifest.json --csv libri_val_manifest.csv
```

```
root@ecs-6740:~# python json_to_csv.py --json libri_test_clean_manifest.json --csv libri_test_clean_manifest.csv
root@ecs-6740:~# python json_to_csv.py --json libri_test_other_manifest.json --csv libri_test_other_manifest.csv
root@ecs-6740:~# python json_to_csv.py --json libri_train_manifest.json --csv libri_train_manifest.csv
root@ecs-6740:~# python json_to_csv.py --json libri_val_manifest.json --csv libri_val_manifest.csv
root@ecs-6740:~# ll
total 6880
```

三个*.csv 文件存放的是对应数据的绝对路径。

2.4 模型训练与评估

2.4.1 模型训练

步骤 1 切换到 models/official/audio/DeepSpeech2 目录下 ,

步骤 2 模型训练需要在 DeepSpeech2 目录下创建 deepspeech_pytorch 目录，并在 deepspeech_pytorch 目录下创建 decoder.py 文件。

```
mkdir deepspeech_pytorch
cd deepspeech_pytorch
touch decoder.py
```

步骤 3 把代码复制到 decoder.py 文件，代码如下：

```
#!/usr/bin/env python
# -----
# Copyright 2015-2016 Nervana Systems Inc.
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# -----
# Modified to support pytorch Tensors

import Levenshtein as Lev
import torch
from six.moves import xrange

class Decoder(object):
    """
    Basic decoder class from which all other decoders inherit. Implements several
    helper functions. Subclasses should implement the decode() method.

    Arguments:
        labels (list): mapping from integers to characters.
        blank_index (int, optional): index for the blank '_' character. Defaults to 0.
    """

    def __init__(self, labels, blank_index=0):
        self.labels = labels
        self.int_to_char = dict([(i, c) for (i, c) in enumerate(labels)])
        self.blank_index = blank_index
        space_index = len(labels) # To prevent errors in decode, we add an out of bounds index
        for the space
        if ' ' in labels:
            space_index = labels.index(' ')
```

```
self.space_index = space_index

def wer(self, s1, s2):
    """
    Computes the Word Error Rate, defined as the edit distance between the
    two provided sentences after tokenizing to words.
    Arguments:
        s1 (string): space-separated sentence
        s2 (string): space-separated sentence
    """

    # build mapping of words to integers
    b = set(s1.split() + s2.split())
    word2char = dict(zip(b, range(len(b))))

    # map the words to a char array (Levenshtein packages only accepts
    # strings)
    w1 = [chr(word2char[w]) for w in s1.split()]
    w2 = [chr(word2char[w]) for w in s2.split()]

    return Lev.distance("".join(w1), "".join(w2))

def cer(self, s1, s2):
    """
    Computes the Character Error Rate, defined as the edit distance.

    Arguments:
        s1 (string): space-separated sentence
        s2 (string): space-separated sentence
    """

    s1, s2 = s1.replace(' ', ''), s2.replace(' ', '')
    return Lev.distance(s1, s2)

def decode(self, probs, sizes=None):
    """
    Given a matrix of character probabilities, returns the decoder's
    best guess of the transcription

    Arguments:
        probs: Tensor of character probabilities, where probs[c,t]
              is the probability of character c at time t
        sizes(optional): Size of each sequence in the mini-batch
    Returns:
        string: sequence of the model's best guess for the transcription
    """
    raise NotImplementedError
```

```
class BeamCTCDecoder(Decoder):
    def __init__(self,
                  labels,
                  lm_path=None,
                  alpha=0,
                  beta=0,
                  cutoff_top_n=40,
                  cutoff_prob=1.0,
                  beam_width=100,
                  num_processes=4,
                  blank_index=0):
        super(BeamCTCDecoder, self).__init__(labels)
        try:
            from ctctdecode import CTCBeamDecoder
        except ImportError:
            raise ImportError("BeamCTCDecoder requires paddleddecoder package.")
        labels = list(labels) # Ensure labels are a list before passing to decoder
        self._decoder = CTCBeamDecoder(labels, lm_path, alpha, beta, cutoff_top_n, cutoff_prob,
                                         beam_width,
                                         num_processes, blank_index)

    def convert_to_strings(self, out, seq_len):
        results = []
        for b, batch in enumerate(out):
            utterances = []
            for p, utt in enumerate(batch):
                size = seq_len[b][p]
                if size > 0:
                    transcript = ".join(map(lambda x: self.int_to_char[x.item()], utt[0:size]))
                else:
                    transcript = ""
            utterances.append(transcript)
            results.append(utterances)
        return results

    def convert_tensor(self, offsets, sizes):
        results = []
        for b, batch in enumerate(offsets):
            utterances = []
            for p, utt in enumerate(batch):
                size = sizes[b][p]
                if sizes[b][p] > 0:
                    utterances.append(utt[0:size])
                else:
                    utterances.append(torch.tensor([], dtype=torch.int))
            results.append(utterances)
        return results
```

```
def decode(self, probs, sizes=None):
    """
    Decodes probability output using ctcdecode package.
    Arguments:
        probs: Tensor of character probabilities, where probs[c,t]
               is the probability of character c at time t
        sizes: Size of each sequence in the mini-batch
    Returns:
        string: sequences of the model's best guess for the transcription
    """
    probs = probs.cpu()
    out, scores, offsets, seq_lens = self._decoder.decode(probs, sizes)

    strings = self.convert_to_strings(out, seq_lens)
    offsets = self.convert_tensor(offsets, seq_lens)
    return strings, offsets

class GreedyDecoder(Decoder):
    def __init__(self, labels, blank_index=0):
        super(GreedyDecoder, self).__init__(labels, blank_index)

    def convert_to_strings(self,
                          sequences,
                          sizes=None,
                          remove_repetitions=False,
                          return_offsets=False):
        """Given a list of numeric sequences, returns the corresponding strings"""
        strings = []
        offsets = [] if return_offsets else None
        for x in xrange(len(sequences)):
            seq_len = sizes[x] if sizes is not None else len(sequences[x])
            string, string_offsets = self.process_string(sequences[x], seq_len, remove_repetitions)
            strings.append([string]) # We only return one path
            if return_offsets:
                offsets.append([string_offsets])
        if return_offsets:
            return strings, offsets
        else:
            return strings

    def process_string(self,
                      sequence,
                      size,
                      remove_repetitions=False):
        string = "
```



```

offsets = []
for i in range(size):
    char = self.int_to_char[sequence[i].item()]
    if char != self.int_to_char[self.blank_index]:
        # if this char is a repetition and remove_repetitions=true, then skip
        if remove_repetitions and i != 0 and char == self.int_to_char[sequence[i - 1].item()]:
            pass
        elif char == self.labels[self.space_index]:
            string += ' '
            offsets.append(i)
        else:
            string = string + char
            offsets.append(i)
    return string, torch.tensor(offsets, dtype=torch.int)

def decode(self, probs, sizes=None):
    """
    Returns the argmax decoding given the probability matrix. Removes
    repeated elements in the sequence, as well as blanks.

    Arguments:
        probs: Tensor of character probabilities from the network. Expected shape of batch x
        seq_length x output_dim
        sizes(optional): Size of each sequence in the mini-batch
    Returns:
        strings: sequences of the model's best guess for the transcription on inputs
        offsets: time step per character predicted
    """
    _, max_probs = torch.max(probs, 2)
    strings, offsets = self.convert_to_strings(max_probs.view(max_probs.size(0),
max_probs.size(1)),
                                              sizes,
                                              remove_repetitions=True,
                                              return_offsets=True)

    return strings, offsets

```

步骤 4 模型配置

修改 src 下的 config.py。修改完成后，“ctrl+s”进行保存，并退出。

- 修改 batch_size 为 1（一次处理数据量大小，该数据与服务器设备性能相关）。
- 修改 epochs 为 1（大概用时 48h，可根据实际需求调整）；
- 修改 train_manifest 为 libri_train_manifest.csv 实际路径；
- 修改 test_manifest 为 libri_test_clean_manifest.csv 为实际路径；
- 修改 eval_config 的加窗类型把 hanning 改为 hann

```
train_config = ed({
    "TrainingConfig": {
        # "epochs": 70,
        "epochs": 1,
    },
    "DataConfig": {
        # "train_manifest": '/data/libri_train_manifest.csv',
        "train_manifest": '/home/work/deepspeech.pytorch/libri_train_manifest.csv',
        # "val_manifest": 'data/libri_val_manifest.csv',
        "val_manifest": '/home/work/deepspeech.pytorch/libri_val_manifest.csv',
        # "batch_size": 20,
        "batch_size": 1,
        "labels_path": "labels.json",
    },
})

eval_config = ed({
    "save_output": 'librispeech_val_output',
    "verbose": True,
    "DataConfig": {
        # "test_manifest": '/data/libri_test_clean_manifest.csv',
        "test_manifest": '/home/work/deepspeech.pytorch/libri_test_clean_manifest.csv',
        # "test_manifest": 'data/libri_test_other_manifest.csv',
        # "test_manifest": 'data/libri_val_manifest.csv',
        # "batch_size": 20,
        "batch_size": 1,
        "labels_path": "labels.json",
    },
    "SpectConfig": {
        "sample_rate": 16000,
        "window_size": 0.02,
        "window_stride": 0.01,
        "window": "hanning"
    },
})
```

步骤 5 安装模型 python 依赖, :

```
cd /home/work/models/official/audio/DeepSpeech2 #该路径的 requirements.txt
pip3.9 install -r requirements.txt
pip3.9 install Levenshtein
pip3.9 install -i https://pypi.tuna.tsinghua.edu.cn/simple torch==1.7.1
pip3.9 install numpy==1.20.0
pip3.9 install numba==0.53.1
```

步骤 6 下载预训练模型, 下载链接为 <https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ASR/DeepSpeech.ckpt>, 下载命令如下:

```
wget https://ascend-professional-construction-dataset.obs.cn-north-4.myhuaweicloud.com/ASR/DeepSpeech.ckpt
```

步骤 7 修改训练启动脚本 scripts 目录下的 run_standalone_train_cpu.sh, 加载预训练模型修改如下:

```
PATH_CHECKPOINT=$1
python ./train.py --device_target 'CPU' --pre_trained_model_path $PATH_CHECKPOINT
```

步骤 8 在 DeepSpeech2 目录下进行模型训练，输入如下命令

```
bash scripts/run_standalone_train_cpu.sh PATH_CHECKPOINT
# PATH_CHECKPOINT 预训练文件路径

example: 使用 nohup 后台运行
nohup bash scripts/run_standalone_train_cpu.sh
'/home/work/models/official/audio/DeepSpeech2/DeepSpeech.ckpt' > train.log 2>&1 &
```

如：

```
root@asr2022-deepspeech2:/home/work/models/official/audio/DeepSpeech2# bash scripts/run_standalone_train_cpu.sh /home/work/models/official/audio/DeepSpeech2/deepspeech_pytorch/DeepSpeech.ckpt
```

步骤 9 查看训练日志，当前目录下 train.log。

输入命令如下：

```
#时刻查看训练日志结果
tail -f train.log
```

步骤 10 因模型在 cpu 服务器训练实际过长，跑完一个完成的 epoch 需要大约 48 小时，可以在模型训练一定步数后停止，例如 150 个 step 进行停止模型训练。停止模型训练命令如下

```
pkill -9 python
```

步骤 11 完整的一个 epoch 模型训练结果，如下图：

```
epoch: 1 step: 20321, loss is 388.0677795410156
epoch: 1 step: 20322, loss is 492.2615661621094
epoch: 1 step: 20323, loss is 444.7136535644531
epoch: 1 step: 20324, loss is 572.735107421875
epoch: 1 step: 20325, loss is 456.96405029296875
epoch: 1 step: 20326, loss is 543.9458618164062
epoch: 1 step: 20327, loss is 194.89727783203125
epoch: 1 step: 20328, loss is 136.91307067871094
epoch: 1 step: 20329, loss is 599.404541015625
epoch: 1 step: 20330, loss is 376.4102478027344
epoch: 1 step: 20331, loss is 472.4965515136719
epoch: 1 step: 20332, loss is 432.156982421875
....
epoch time: 158699402.968 ms, per step time: 7803.865 ms
```

查看模型文件。在 checkpoint 文件夹下，保存模型文件，如下图：。

```
root@ecs-32c9:/home/xx/deepspeech2# ll checkpoint
total 10149708
drwx----- 2 root root      4096 Feb 22 15:40 ./
drwxr-xr-x  8 root root      4096 Feb 22 15:40 ../
-r-----  1 root root 1039306262 Feb 22 15:35 DeepSpeech-1_100.ckpt
-r-----  1 root root 1039306262 Feb 22 15:35 DeepSpeech-1_105.ckpt
-r-----  1 root root 1039306262 Feb 22 15:36 DeepSpeech-1_110.ckpt
-r-----  1 root root 1039306262 Feb 22 15:36 DeepSpeech-1_115.ckpt
-r-----  1 root root 1039306262 Feb 22 15:37 DeepSpeech-1_120.ckpt
-r-----  1 root root 1039306262 Feb 22 15:38 DeepSpeech-1_125.ckpt
-r-----  1 root root 1039306262 Feb 22 15:38 DeepSpeech-1_130.ckpt
-r-----  1 root root 1039306262 Feb 22 15:39 DeepSpeech-1_135.ckpt
-r-----  1 root root 1039306262 Feb 22 15:40 DeepSpeech-1_140.ckpt
-r-----  1 root root 1039306262 Feb 22 15:34 DeepSpeech-1_95.ckpt
-rw-----  1 root root    182298 Feb 22 15:22 DeepSpeech-graph.meta
root@ecs-32c9:/home/xx/deepspeech2#
```

图2.4.1.1.1.11.1.1 模型训练 ckpt 图

选择最后一个 epoch 的 checkpoint 文件用以评估和模型导出。

2.4.2 模型评估

步骤 1 模型评估，输入如下命令进行评估。

```
# CPU 评估
bash scripts/run_eval_cpu.sh [PATH_CHECKPOINT]
# [PATH_CHECKPOINT] 模型 checkpoint 文件
#参考样例：
bash scripts/run_eval_cpu.sh ./checkpoint/ DeepSpeech-1_140.ckpt
```

查看评估日志，当前目录下 eval.log。输入命令如下：

```
#时刻查看评估日志结果
tail -f eval.log
```

评估结果如下图所示：

```
7-100@114.116.245.222 ~
Hyp: ahe
WER: 1.0 CER: 0.9302325581395349
Ref: young fitzooth had been commanded to his mothers chamber so soon as he had come out from his converse with the squire
Hyp: ahe
WER: 1.0 CER: 0.96875
Ref: we shall be blown up but no the dazzling disk of mysterious light nimbly leaps aside it approaches hans who fixes his blue eye upon it st
Hyp: ahe
WER: 1.0 CER: 0.9840425531914894
Ref: approaching the dining table he carefully placed the article in the centre and removed the cloth
Hyp: ahe
WER: 1.0 CER: 0.9629629629629629
Ref: edison had installed his historic first great central station system in new york on the multiple arc system covered by his feeder and mai
Hyp: ahe
WER: 1.0 CER: 0.9868421052631579
Ref: you left him in a chair you say which chair by the window there
Hyp: ahe
WER: 1.0 CER: 0.94
Ref: on the other hand we are not to regard them as so terrible that we must despair
Hyp: ah
WER: 1.0 CER: 0.9682539682539683
Ref: and to think we can save all that misery and despair by the payment of a hundred and fifty dollars
Hyp: ahe
WER: 1.0 CER: 0.9620253164556962
Ref: he still held on to it with both hands as he rushed into his mothers cottage
Hyp: ahe
WER: 1.0 CER: 0.9508196721311475
Ref: the parliament and the scots laid their proposals before the king
Hyp: ahe
WER: 1.0 CER: 0.9454545454545454
Test Summary      Average WER 100.000      Average CER 96.907
```

图2.4.2.1.1.1.1.1 评估结果图

因为模型只训练了 150step，评估的 WER 暂无变化，但 CER 有所下降，如果需要更加准确的结果，至少需要跑完 2 个 epochs，获取对应的 ckpt 文件，在进行评估。

ASR 是指自动语音识别技术（Automatic Speech Recognition），是一种将人的语音转换为文本的技术。图 2-7 中 WER 为词错率，词错率（Word Error Rate, WER）是一项用于评价 ASR 性能的重要指标，用来评价预测文本与标准文本之间错误率，因此词错率最大的特点是越小越好。因为英文语句中句子的最小单位是单词，而中文语句中的最小单位是汉字，因此在中文语音转文本任务或中文语音识别任务中使用字错率（Character Error Rate, CER）来衡量中文 ASR 效果好坏。

这个模型能够通过深度学习网络识别嘈杂环境下的两种完全不同的语言——英语与普通话，而端到端的学习能够使系统处理各种条件下的语音，包括嘈杂环境、口音及区别不同语种。

步骤 2 模型导出，需要修改 export.py 文件以下代码。

```
config = train_config
context.set_context(mode=context.GRAPH_MODE, device_target="CPU", save_graphs=False)
with open(config.DataConfig.labels_path) as label_file:

    labels = json.load(label_file)
```

步骤 3 输入如下命令用于转换并导出模型文件

```
python export.py --pre_trained_model_path ./checkpoint/DeepSpeech-1_856.ckpt
```

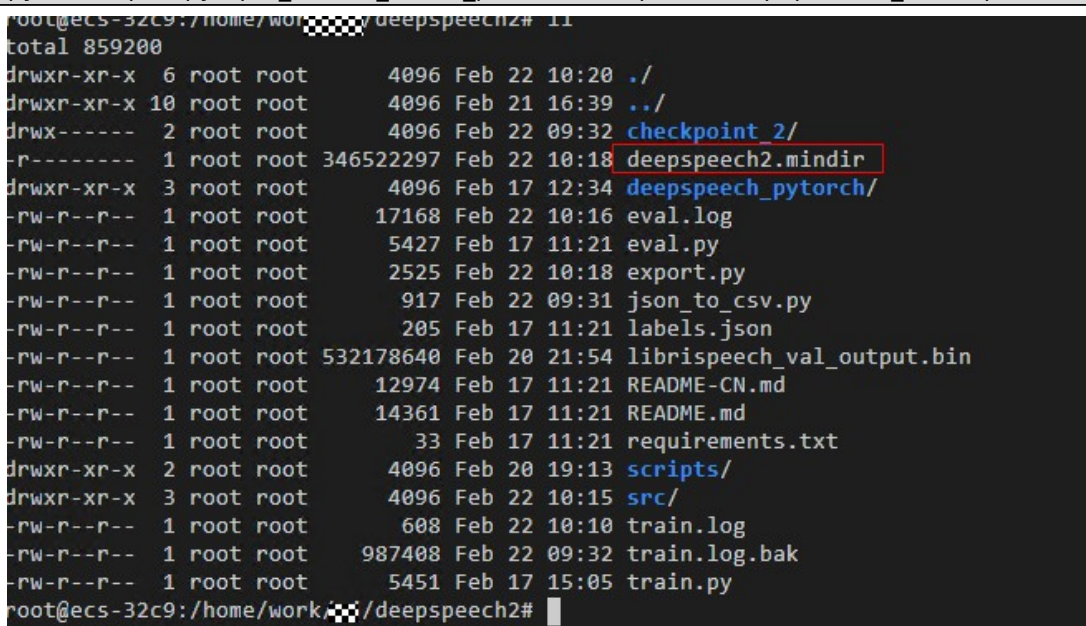


图2.4.2.1.1.3.1.1 模型 mindir 文件

2.4.3 思考题

模型训练 1 轮，需要 50 小时，怎么进行速度提升？

2.5 实验小结

本案例通过使用 MindSpore 框架搭建 deepspeech2 网络采用 LibriSpeech 数据集实现语音生成任务。



3 附录：开发环境搭建

3.1 ECS 弹性服务器

步骤 1 搜索“ECS”，点击“控制台”。

在[华为云 ECS 主页](#)，点击“管理控制台”进入 ECS 的管理页面。



图3.1.1.1.1.1.1 ECS 搜索界面

步骤 2 创建弹性云服务器

控制台区域选择“华北 - 北京四”，在左侧菜单栏中选择“弹性云服务器”，在右上角“购买弹性云服务器”。

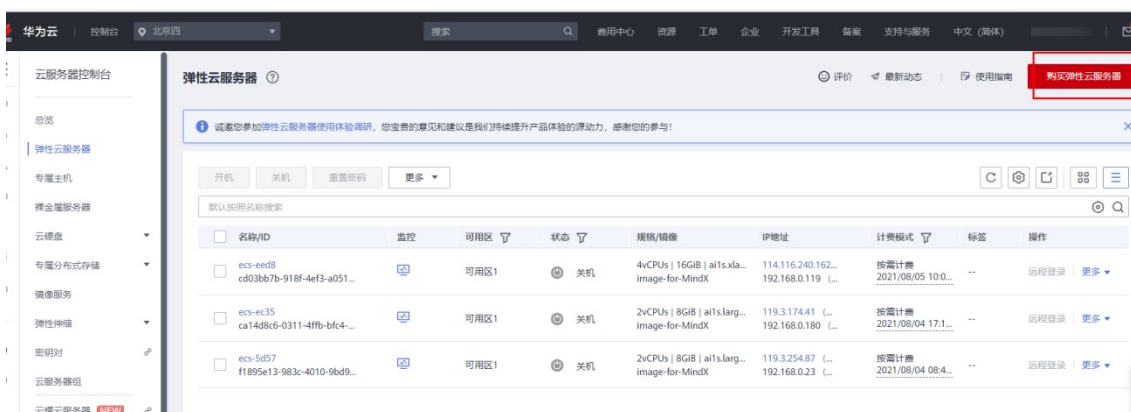


图3.1.1.1.2.1.1 ECS 购买界面

在“基础配置”里，选择如下配置：

- 计费模式：按需计费。
- 区域：华北-北京四。
- 可用区：随机分配。

- CPU 架构：x86 计算。
- 规格：通用计算增强型型 | c7.2xlarge.2 | 8vCPUs | 16GiB
- 镜像：公共镜像，Ubuntu，Ubuntu 18.04 server 64bit
- 系统盘：通用型 SSD，100GB。

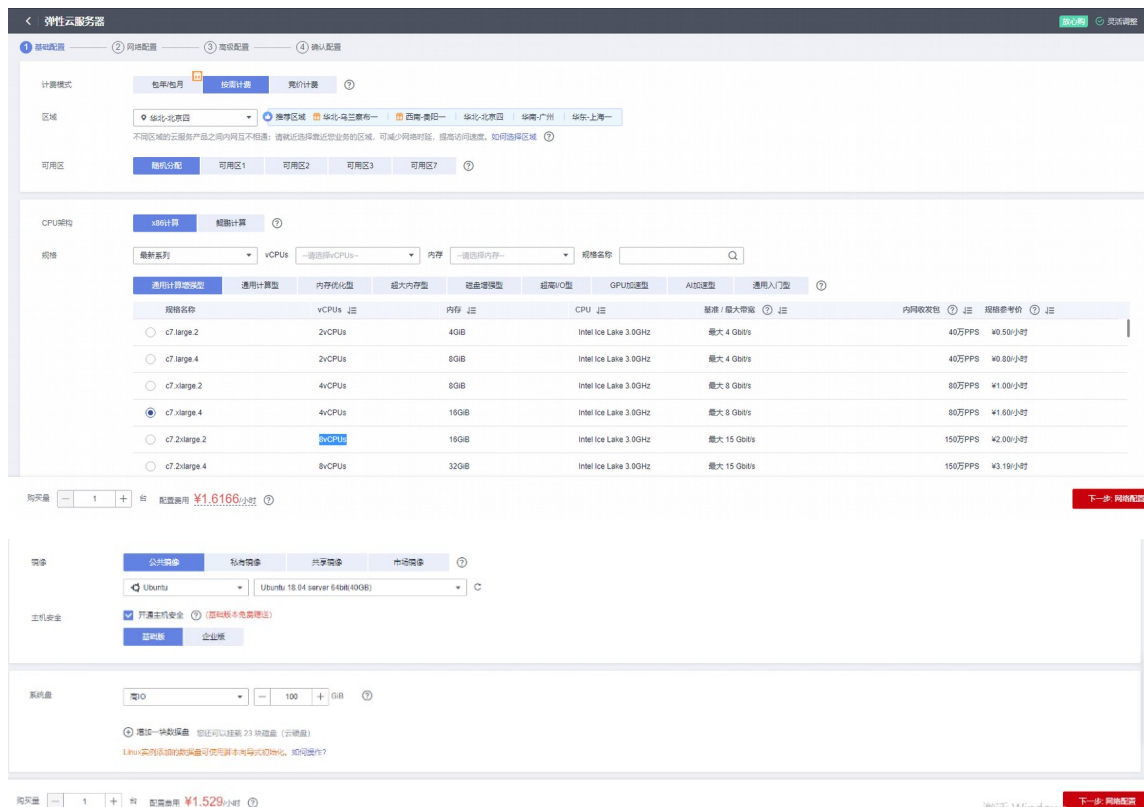


图3.1.1.1.1.2 ECS 基础配置

步骤 3 在窗口右下角中点击“下一步：网络配置”

在“网络配置”里，选择如下配置：

- 网络：可以前往控制台创建新的虚拟私有云。
- 拓展网卡：无。
- 安全组：可以新建安全组。
- 弹性公网 IP：现在购买。
- 线路：全动态 BGP。
- 公网带宽：按流量计费。
- 宽带大小：自定义，200Mbit/s。
- 释放行为：勾选随实例释放。

如图所示：

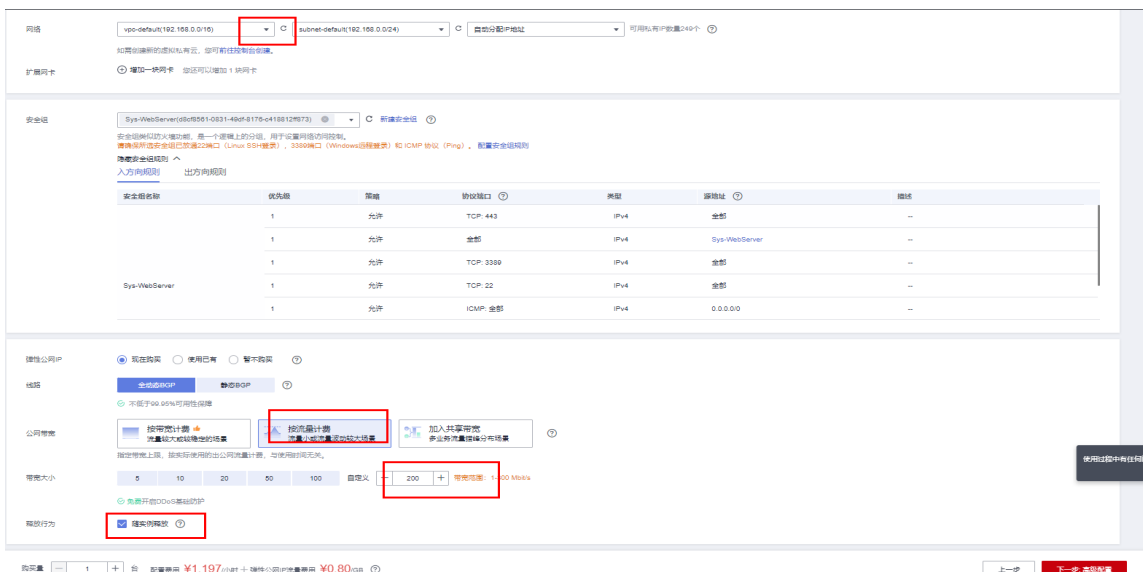


图3.1.1.1.1.3.1.1 ECS 网络配置

步骤 4 在窗口右下角中点击“下一步：高级配置”

在“高级配置”里，选择如下配置：

- 云服务器名称：可以自定义。
- 登录凭证：密码。
- 用户名：root。
- 密码：自定义（后续登录使用，需谨记）。
- 云备份：暂不购买。
- 云服务器组：无。
- 高级选项：无。

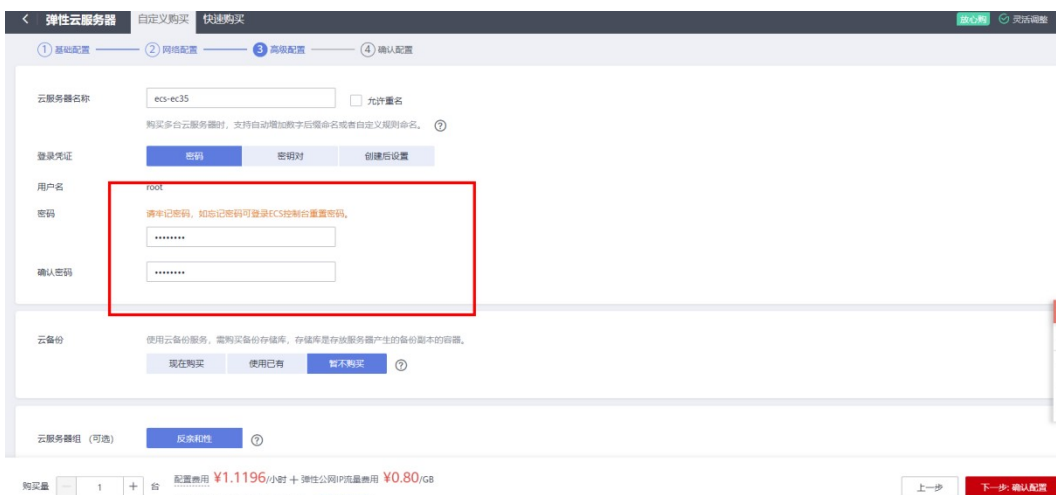


图3.1.1.1.1.4.1.1 高级配置

步骤 5 在窗口右下角中点击“下一步：确认配置”，

在“确认配置”里，选择如下配置：

协议：勾选我已阅读并同意《镜像免责声明》。



图3.1.1.1.5.1.1 ECS 确认配置

步骤 6 在窗口右下角中点击“立即购买”。

“任务提交成功”之后，选择“返回服务器列表”即可回到弹性云服务器的管理控制台，看到已创建的 ECS 弹性云服务器正在运行中。

* 注意在“IP 地址”显示的弹性公网 IP 地址，后续会用到。



图3.1.1.1.6.1.1 配置成功界面

3.2 MobaXterm 连接 ECS

步骤 1 下载 MobaXterm

进入 MobaXterm 的官网主页：<https://mobaxterm.mobatek.net/>

选择“Home Edition”，下载“MobaXterm Home Edition v21.x（Portable edition）”。

下载完成之后解压 MobaXterm_Portable_v21.x.zip 文件。

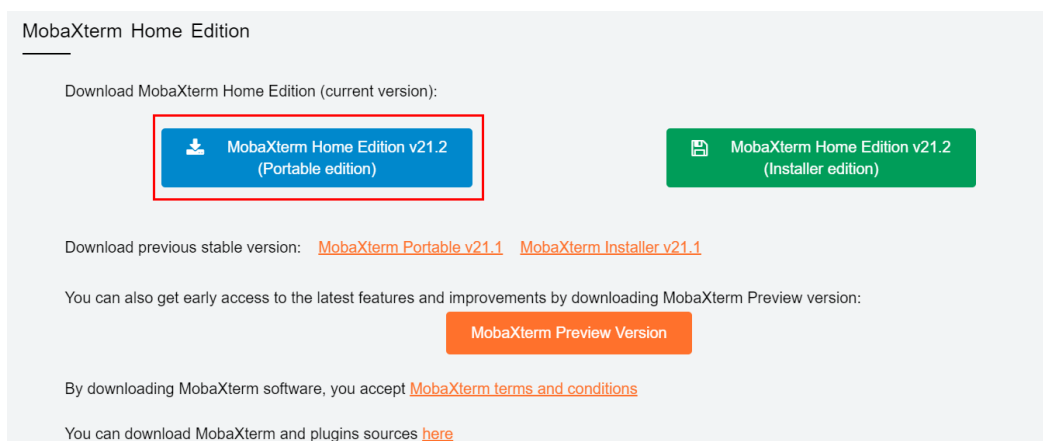


图3.2.1.1.1.1.1 下载界面

步骤 2 使用 MobaXterm 远程连接弹性云服务器

进入解压后的 MobaXterm_Portable_v21.x 文件夹，打开 MobaXterm_Personal_21.x.exe 文件，选择菜单栏的“Session”，之后进入“Session settings”页面，远程链接选择“SSH”协议，输入图 2-12 ECS 弹性云服务器创建成功时显示的弹性公网 IP 地址，选择指定用户名“Specify username”，用户名为“root”，配置完成之后选择“OK”提交。

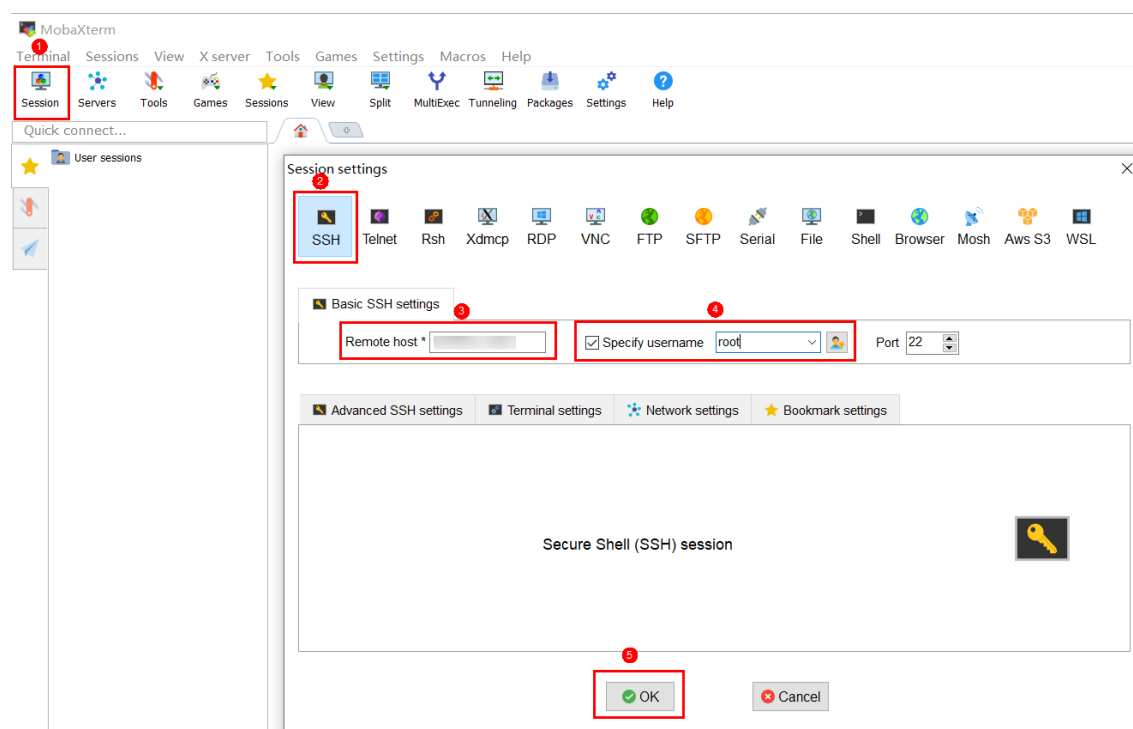


图3.2.1.1.1.2.1.1 MobaXterm 远程连接

MobaXterm 登录 ECS 需输入密码，在 ECS 弹性云服务器的步骤 4 中，高级配置里已自定义了弹性云服务器 root 用户密码，在此输入即可。

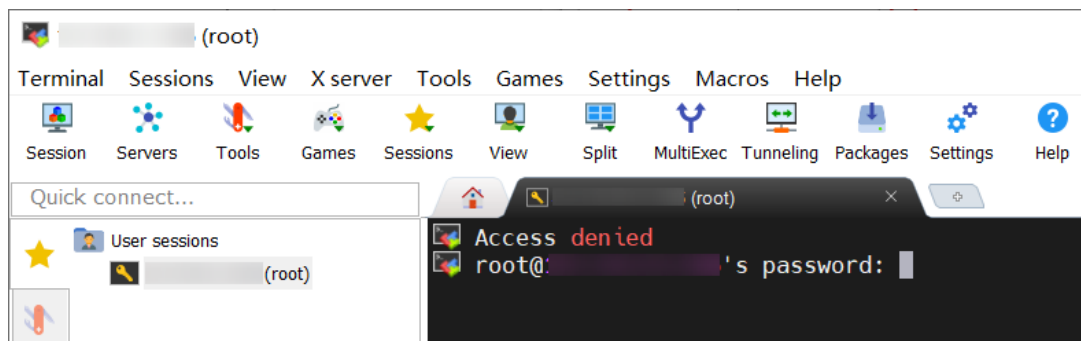


图3.2.1.1.2.1.2 MobaXterm 登录 ECS 需输入密码

MobaXterm 远程链接弹性云服务器成功，后续还需进一步配置弹性云服务器的云上环境。

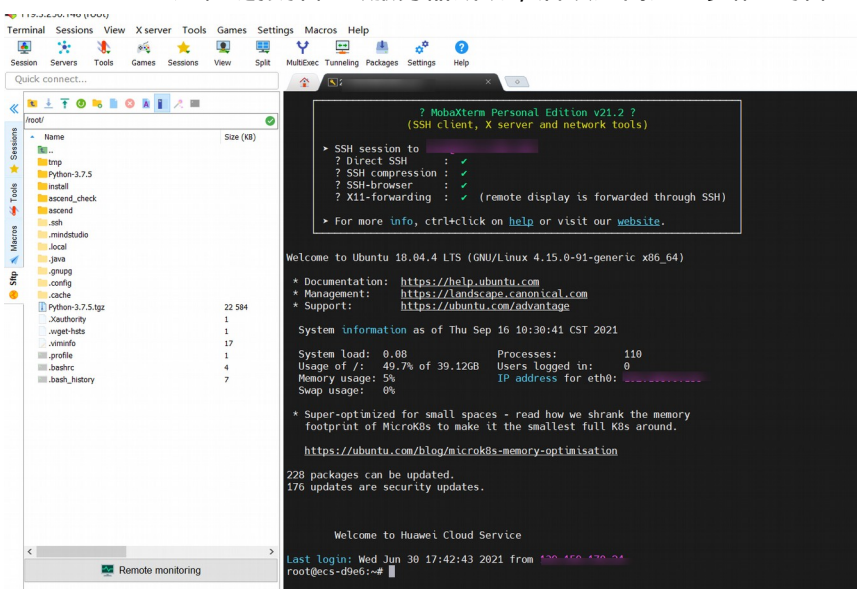


图3.2.1.1.2.1.3 MobaXterm 登录成功界面

3.3 FinalShell 连接 ECS

步骤一：安装 FinalShell；下载链接：<https://www.hostbuf.com/t/988.html>

步骤二：获取主机 ip、ssh 端口信息；



登录Linux弹性云服务器 ?

CloudShell登录（默认使用22端口） 最近使用

支持复制粘贴命令、多会话分区布局，可同时管理多台云服务器，操作更流畅。

立即登录

其他登录方式

云堡垒机登录 RemoteShell客户端登录 VNC登录

远程登录遇到问题时，建议您通过诊断工具或教程自助排查。 [深度诊断](#) [查看教程](#)

区域：华北-北京四

刷新

云服务器：csd

1.94.237.100 (公网) 192.168.0.113 (私网)

端口：22

用户名：root

认证方式：密码认证

密码：

会话名称：root@1.94.237.100

获取主机ip

☒ 打开远程主机文件树

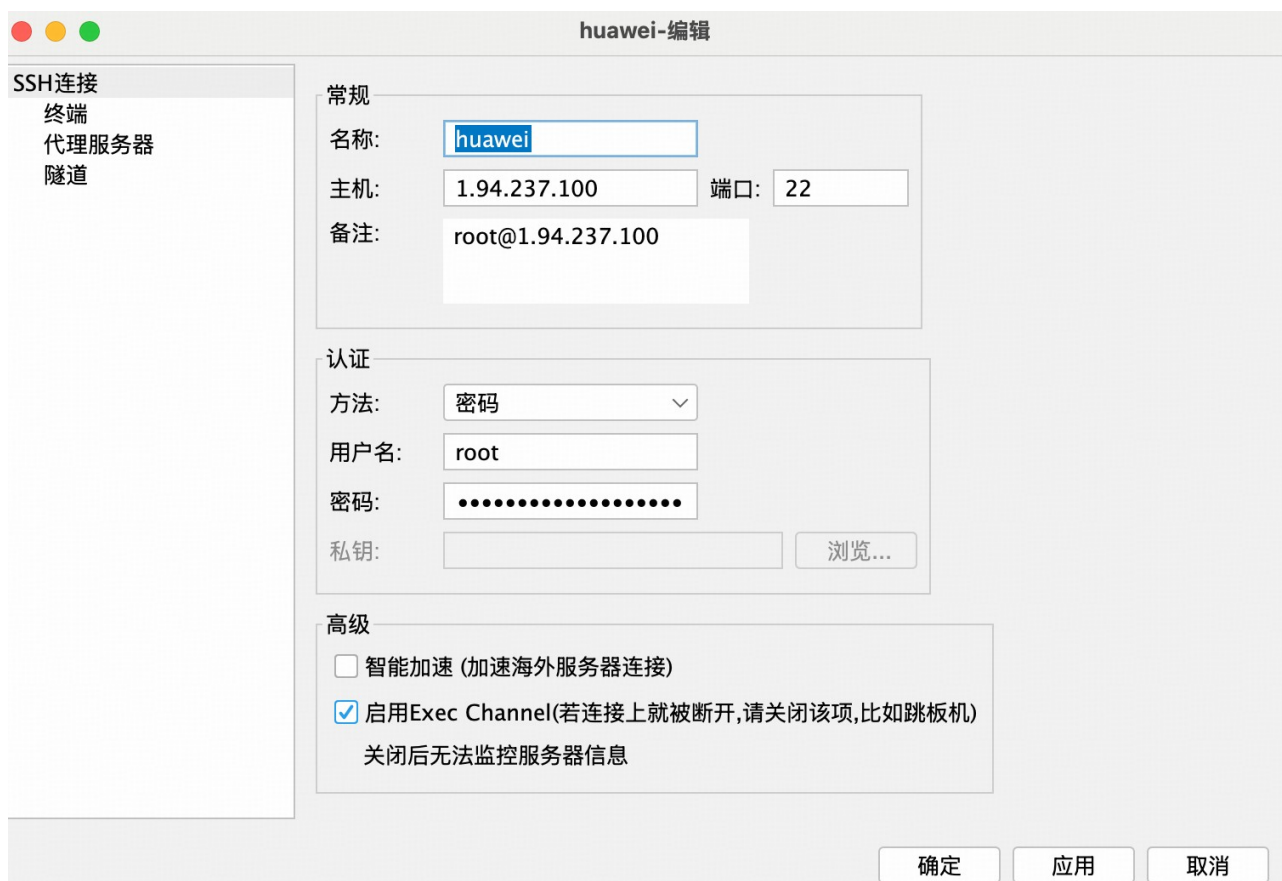
注意：

- 为确保连接的安全性，系统将对超过 20分钟 没有活跃的会话进行自动断开。
- 请确认安全组中来源为CloudShell代理IP的远程端口（SSH默认端口为22）已经允许。
- 当远程登录后操作卡顿时，建议查看一下机器的CPU、内存情况，请定义云监控在主机异常时通过短信等多种方式通知。
- 华为云CloudShell不会保存您的密码，请妥善保管以防丢失。

连接

取消

步骤三：finalshell 远程登录服务器：



步骤四：本地文件拖到对应服务器目录即可；

4 问题 FAQ

4.1 报错“ No modul named '_bz2' ”

如下所示


```
root@ecs-cbea:/home/two/zj/deepspeech2# bash scripts/run_standalone_train_gpu.sh 0
Traceback (most recent call last):
  File "./train.py", line 33, in <module>
    from src.dataset import create_dataset
  File "/home/two/zj/deepspeech2/src/dataset.py", line 21, in <module>
    import librosa
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/__init__.py", line 209, in <module>
    from . import core
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/__init__.py", line 5, in <module>
    from .convert import * # pylint: disable=wildcard-import
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/convert.py", line 7, in <module>
    from . import notation
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/notation.py", line 8, in <module>
    from ..util.exceptions import ParameterError
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/util/__init__.py", line 78, in <module>
    from .files import * # pylint: disable=wildcard-import
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/util/files.py", line 11, in <module>
    import pooch
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pooch/__init__.py", line 19, in <module>
    from .processors import Unzip, Untar, Decompress
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pooch/processors.py", line 12, in <module>
    import bz2
  File "/usr/local/python3.7.5/lib/python3.7/bz2.py", line 19, in <module>
    from bz2 import BZ2Compressor, BZ2Decompressor
ModuleNotFoundError: No module named 'bz2'
```

● 可能原因

出现这个错误的原因是由于运行程序所使用的 python 版本中没有安装_bz2 库所致。

通常是由于运行程序使用的是 python3.9，但是 bz2 这个库是安装到了 python3.6 的路径下，所以找不到。

● 解决方案

将 python3.6 里面的 bz2 库拷贝到 python3.9 下面。

● 操作步骤：

步骤 1 找到 python3.6 路径下的_bz2 库文件，即“_bz2.cpython-36m-x86_64-linux-gnu.so”

```
ll /usr/lib/python3.6/lib-dynload/
```

步骤 2 切换到 python3.9 对应路径，将该文件复制到该目录下

```
cd /usr/local/python3.9.0/lib/python3.9/lib-dynload/
cp /usr/lib/python3.6/lib-dynload/_bz2.cpython-36m-x86_64-linux-gnu.so ./
```

步骤 3 修改文件名称，将“-36m”修改为“-39”

```
mv _bz2.cpython-36m-x86_64-linux-gnu.so _bz2.cpython-39-x86_64-linux-gnu.so
chmod +x _bz2.cpython-39-x86_64-linux-gnu.so #（可选）增加该文件的可执行权限
```

现在运行程序所使用的是 python3.9 的目录下已经有了 bz 库文件



说明：

该问题有多种解决方案，只要保证所使用的 python 版本相关路径下存在_bz2 库即可。例如可以从网上下载“_bz2.cpython-39-x86_64-linux-gnu.so”文件，或从任意其他存在该文件的环境中复制到目标环境的相关路径下，并改成对应版本的 python 文件即可。

4.2 没有_lzma 模块

● 报错 报错“ No module named '_lzma' ”。如下所示

```
>>> exit()
root@ecs-cbea:/home/two/zj/deepspeech2# python
Python 3.7.5 (default, Feb 11 2022, 14:43:18)
[GCC 7.5.0] on linux
Type "help", "copyright", "credits" or "license()" for more
>>> import bz2
>>> exit()
root@ecs-cbea:/home/two/zj/deepspeech2# bash scripts/run_standalone_train_gpu.sh 0
Traceback (most recent call last):
  File "/train.py", line 33, in <module>
    from src.dataset import create_dataset
  File "/home/two/zj/deepspeech2/src/dataset.py", line 21, in <module>
    import librosa
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/__init__.py", line 209, in <module>
    from . import core
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/__init__.py", line 5, in <module>
    from .convert import * # pylint: disable=wildcard-import
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/convert.py", line 7, in <module>
    from . import notation
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/core/notation.py", line 8, in <module>
    from ..util.exceptions import ParameterError
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/util/__init__.py", line 78, in <module>
    from .files import * # pylint: disable=wildcard-import
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/librosa/util/files.py", line 11, in <module>
    import pooch
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pooch/__init__.py", line 19, in <module>
    from .processors import Unzip, Untar, Decompress
  File "/usr/local/python3.7.5/lib/python3.7/site-packages/pooch/processors.py", line 14, in <module>
    import lzma
  File "/usr/local/python3.7.5/lib/python3.7/lzma.py", line 27, in <module>
    from lzma import *
ModuleNotFoundError: No module named 'lzma'
```

解决方案 请参考上一步骤“4.2 没有 bz2 模块”解决,保证所使用的 python 版本相关路径下存在 lzma 库即可。

```
cd /usr/local/python3.9.0/lib/python3.9/lib-dynload/
cp /usr/lib/python3.6/lib-dynload/_lzma.cpython-36m-x86_64-linux-gnu.so ./
mv _lzma.cpython-36m-x86_64-linux-gnu.so _lzma.cpython-39-x86_64-linux-gnu.so
chmod +x _lzma.cpython-39-x86_64-linux-gnu.so
```