

**《软件项目管理》课程项目报告**



题 目 **《智能聊天机器人项目报告》**

学 院  **软件学院**

专 业  **软件工程**

学生姓名  **王世龙**

学 号  **2018141463124**

年 级 **2018级**

二Ο二一 年 六 月

1 绪论

* 1. 项目背景

聊天机器人应用范围广泛，作用明显。预计到2020年，聊天机器人将为85%的客户服务交互提供助力，到2022年，聊天机器人每年将节约80多亿美元的成本。随着聊天机器人技术的不断优化，聊天机器人的应用领域也逐渐广泛，包含金融、电信、旅游甚至体育、医疗等领域，从而为用户打造更好的服务。聊天机器人已开始慢慢渗透到了人们的日常生活中，未来将会有更多细分领域下的应用场景落地，并且在技术、企业和资本的不断助力下，聊天机器人将不断释放其潜力价值。

1.2 国内外研究现状

国外的bot startups种类比较多，各个等级的企业都有，从最上面的应用层来说，slack、messenger、telegram、kik等各个message平台上都有大量的bot，包括各种各样的服务。这类bot门槛较低，缺乏核心技术，通常是一个idea来支撑整个企业，容易同质化，来源可能是各种bot比赛的产物，域名都是.ai，稍微大一点的支持多个平台，很多都是只在slack上使用，有一种bot成海的感觉，什么样的服务都可以用bot来做，强行改变交互方式。有的slack bot服务于team，有的是将slack与其他服务，比如google analytics，以bot的形式进行桥接。目前，我国企业聊天机器人重点布局客服领域。埃森哲与蚂蚁金服发布《新客服白皮书》显示，目前新客服创新中核心技术之一便是对话机器人。报告显示，大量资本将聊天机器人持续投入智能客服领域。截至2018年12月31日，国内共有73家提供数字客服解决方案的企业获得融资，26.03%的企业获得A轮融资，部分公司融资到C轮以上。

1.3 项目的主要工作

我们的项目是用python实现智能聊天机器人，现阶段定下来的主要目标为，实现中文的语音识别，并且能够根据语音完成三种不同情境下的聊天需求和智能助理服务，包括任务完成类型，例如：人：今天天气如何 机器人：今天深圳天气晴朗，25~33度；闲聊类型，人：哇，天气好好，做点什么好呢。机器人：不如出去公园散散步玩玩吧~；问答类型，人：哪个地方？机器人：给你推荐附件的市民中心吧。

可以将任务划分为以下几个方面：

声学语音部分包括：1.语音识别功能(ASR) 2.语音合成(TTS)

自然语音理解部分包括:1.语言理解(NLU) 2.对话状态管理（DST) 3.动作候选排序（Policy） 4.语音生成（NLG）

进阶目标是实现更加智能化的语音对答，包括更多情境下的回复，以及支持更多语言。

1.4 项目组成员及分工

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 技术水平 | 角色 | 工作描述 |
| 蒋祎 |  | 项目管理、前期分析、设计 | 分析系统需求、项目计划、项目团队管理、检查进度 |
| 娄泰宇 |  | 分析、设计 | 分析新功能、软件框架扩展、代码模块分配、设计说明书 |
| 张天翊 |  | 设计、编码 | 数据交换、模块设计、接口设计、模块编写 |
| 谭兴成 |  | 编码、测试 | 模块编写、加载程序编写、安装程序编写、软件测试 |
| 王世龙 |  | 测试、文档编写 | 软件测试、软件测试文档、用户操作手册 |

2 相关技术介绍

本项目语音识别部分通过采用卷积神经网络（CNN）和连接性时序分类（CTC）方法，使用大量中文语音数据集进行训练，将声音转录为中文拼音，并通过语言模型，将拼音序列转换为中文文本。

我们使用的的声学模型采用了深度全卷积神经网络，直接将语谱图作为输入。模型结构上，借鉴了VGG，这种网络模型有着很强的表达能力，可以看到非常长的历史和未来信息，相比RNN在鲁棒性上更出色。在输出端，这种模型可以和CTC方案可以完美结合，以实现整个模型的端到端训练，将声音波形信号直接转录为中文普通话拼音序列。在语言模型上，通过最大熵隐含马尔可夫模型，将拼音序列转换为中文文本。

3 设计工作和进展情况

3.1 项目概况

3.1 项目概况

3.1.1 项目特点

参见项目技术介绍

3.1.2 功能需求

SRS-0010 语音转文字需求（核心功能）

在该系统中将输入的语音转换文字，并能够输出成一定格式和规范的文字数据，提供给下一个功能进行识别和转换。

#

SRS-0020 文字转拼音需求（核心功能）

系统能够将语音转文字中转换出来的文字进行识别，并将其转换为拼音数据，拼音数据将提供给下一个拼音转语音的功能进行识别和转换。

#

SRS-0030 拼音转语音需求（核心功能）

系统能够将语音转文字中转换出来的文字进行识别，并将其转换为拼音数据，拼音数据将提供给下一个拼音转语音的功能进行识别和转换。

SRS-0030 NLP需求（自然语言处理）

让计算机接受用户自然语言形式的输入，并在内部通过人类所定义的算法进行加工、计算等系列操作，以模拟人类对自然语言的理解，并返回用户所期望的结果。

拓展功能：针对不同情境 做出不同的回应

#

SRS-0040 问答类型聊天需求

系统应提供用户的问答型对话需求，例如：XXX，今天天气如何？机器人：今天天气晴，体感温度16摄氏度，建议您多加一件衣服。等等类似的对话。

#

SRS-0050 闲聊类型聊天需求

系统应提供用户的闲聊类型聊天需求，例如：机器人：XXX，您今天看起来心情很好啊，是发生了什么好事情么。等等类似的对话。

#

SRS-0060 情感分析

系统应可以从与用户的对话当中，提取有效信息并分析出用户当前的心情概率，并以此为依据进行闲聊类型聊天的方向选择。例如：当判断用户表现出明显的不耐烦或者其他消极情绪时，要及时停止话题结束对话，等

#

未来发展方向：

SRS-0070 智能家居

系统应在不同问答类型聊天的基础上，进行与家居设备的操作互动。例如，XXX，我想听《xxxx》音乐，当用户提出这样的目的性问答聊天时，系统需要进行回应，并进行音乐播放的操作。类似的还有，开门，关灯等，达到智能家居的效果。

SRS-0080 微信小程序聊天机器人

系统应搭建在微信小程序平台上，能够支持语音输入和键盘输入，并且能够在不同情境下进行反馈。

3.2 项目开发计划及阶段性完成情况

3.2.1 工作内容

模型学习：

CTC是一种让网络自动学会对齐的好方法，十分适合语音识别和书写识别。

为了描述地更形象一些，我们可以把输入序列（音频）映射为X=[x1,x2,…,xT]，其相应的输出序列（转录）即为Y=[y1,y2,…,yU]。这之后，将字符与音素对齐的操作就相当于在X和Y之间建立一个准确的映射。

如果想直接用非监督学习算法，我们还有3个难点：

X和Y的长度是可变化的；

X和Y的长度比例也是可变化的；

X和Y没有严格对齐。

而CTC算法刚好能解决这些问题，给定一个X，它能基于所有可能是准确映射的Y给出输出分布。根据这个分布，我们可以推理最可能的输出，或计算分布内各字符的可能性概率。

CTC在损失计算和推理上十分高效。

模型搭建：

batch\_size = tf.shape(inputs)[0]

max\_time = tf.shape(inputs)[1]

encoder\_outputs, final\_state = self.encoder(

inputs, inputs\_seq\_len, keep\_prob, is\_training)

self.encoder\_outputs = encoder\_outputs

# Reshape to apply the same weights over the timesteps

output\_dim = encoder\_outputs.shape.as\_list()[-1]

outputs\_2d = tf.reshape(encoder\_outputs, shape=[batch\_size \* max\_time, output\_dim])

if self.bottleneck\_dim is not None and self.bottleneck\_dim != 0:

with tf.variable\_scope('bottleneck') as scope:

outputs\_2d = tf.contrib.layers.fully\_connected(

outputs\_2d,

num\_outputs=self.bottleneck\_dim,

activation\_fn=tf.nn.relu,

weights\_initializer=tf.truncated\_normal\_initializer(stddev=self.parameter\_init),

biases\_initializer=tf.zeros\_initializer(),

scope=scope)

outputs\_2d = tf.nn.dropout(

outputs\_2d, keep\_prob, name='dropout\_bottleneck')

with tf.variable\_scope('output') as scope:

logits\_2d = tf.contrib.layers.fully\_connected(outputs\_2d,

num\_outputs=self.num\_classes,

activation\_fn=None,

weights\_initializer=tf.truncated\_normal\_initializer(

stddev=self.parameter\_init),

biases\_initializer=tf.zeros\_initializer(),

scope=scope)

if self.time\_major:

# Reshape back to the original shape

logits = tf.reshape(logits\_2d, shape=[max\_time, batch\_size, self.num\_classes])

else:

# Reshape back to the original shape

logits = tf.reshape(logits\_2d, shape=[batch\_size, max\_time, self.num\_classes])

# Convert to time-major: `[T, B, num\_classes]'

logits = tf.transpose(logits, [1, 0, 2])

return logits

模型训练：

# Tell TensorFlow that the model will be built into the default graph

with tf.Graph().as\_default(), tf.device('/cpu:0'):

global\_step = tf.Variable(0, name='global\_step', trainable=False)

# Set optimizer

#设置优化器

learning\_rate\_pl = tf.placeholder(tf.float32, name='learning\_rate')

optimizer = model.\_set\_optimizer(params['optimizer'], learning\_rate\_pl)

# Calculate the gradients for each model tower

#定义梯度和损失

total\_grads\_and\_vars, total\_losses = [], []

decode\_ops, ler\_ops = [], []

#loss平均

total\_losses = tf.concat(axis=0, values=total\_losses)

loss\_op = tf.reduce\_mean(total\_losses, axis=0)

#ler平均

ler\_ops = tf.concat(axis=0, values=ler\_ops)

ler\_op = tf.reduce\_mean(ler\_ops, axis=0)

#梯度平均

average\_grads\_and\_vars = average\_gradients(total\_grads\_and\_vars)

train\_op = optimizer.apply\_gradients(average\_grads\_and\_vars,global\_step=global\_step)

summary\_train = tf.summary.merge(model.summaries\_train)

summary\_dev = tf.summary.merge(model.summaries\_dev)

#一次初始化所有参数

init\_op = tf.global\_variables\_initializer()

saver = tf.train.Saver(max\_to\_keep=None)

#统计所有的参数

parameters\_dict, total\_parameters = count\_total\_parameters(tf.trainable\_variables())

for parameter\_name in sorted(parameters\_dict.keys()):

print("%s %d" % (parameter\_name, parameters\_dict[parameter\_name]))

print("Total %d variables, %s M parameters" % (len(parameters\_dict.keys()),

"{:,}".format(total\_parameters / 1000000)))

#获取数据

train\_dataset = tf.data.TFRecordDataset(params['train\_data\_file'])

train\_dataset = train\_dataset.map(parse\_function)

train\_dataset = train\_dataset.shuffle(1000)

train\_dataset = train\_dataset.padded\_batch(params['batch\_size'], padded\_shapes=([None, None],

[None], []), padding\_values=(0.0, tf.cast(-1, tf.int64), tf.cast(0, tf.int32)))

train\_dataset = train\_dataset.repeat(1)

iterator = train\_dataset.make\_initializable\_iterator()

#获取一个batch

batch\_feat, batch\_label, batch\_seq\_len = iterator.get\_next()

s\_indices, s\_value, s\_shape = tf.py\_func(dense\_to\_sparse, [batch\_label], [tf.int64, tf.int32, tf.int64])

batch\_label = tf.SparseTensor(s\_indices, s\_value, s\_shape)

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True,log\_device\_placement=False)) as sess:

summary\_writer = tf.summary.FileWriter(model.save\_path, sess.graph)

sess.run(init\_op)

start\_time\_train = time.time()

learning\_rate = float(params['learning\_rate'])

step\_id = 0

last\_id = 0

#获取数据

for i\_gpu in range(len(gpu\_indices)):

new\_feat, new\_label, new\_seq\_len = sess.run([batch\_feat, batch\_label, batch\_seq\_len])

feed\_dict\_train[model.inputs\_pl\_list[i\_gpu]] = new\_feat

feed\_dict\_train[model.labels\_pl\_list[i\_gpu]] = new\_label

feed\_dict\_train[model.inputs\_seq\_len\_pl\_list[i\_gpu]] = new\_seq\_len

feed\_dict\_train[model.keep\_prob\_pl\_list[i\_gpu]] = float(params['dropout'])

feed\_dict\_train[learning\_rate\_pl] = learning\_rate

#一次训练

step\_loss, step\_ler, \_, \_ = sess.run([loss\_op,

ler\_op, global\_step, train\_op], feed\_dict=feed\_dict\_train)

step\_id += 1

end\_time\_step = time.time()

step\_time = end\_time\_step - start\_time\_step

测试：

#!/bin/bash

. ./cmd.sh

. ./path.sh

# step 1: generate fbank features

obj\_dir=data/chat001

for x in test; do

rm -rf fbank/$x

mkdir -p fbank/$x

# compute fbank without pitch

steps/make\_fbank.sh --nj 1 --cmd "run.pl" $obj\_dir/$x exp/make\_fbank/$x fbank/$x || exit 1;

# compute cmvn

steps/compute\_cmvn\_stats.sh $obj\_dir/$x exp/fbank\_cmvn/$x fbank/$x || exit 1;

done

# #step 2: offline-decoding

test\_data=data/chat001/test

dir=exp/chain/tdnn

steps/nnet3/decode.sh --acwt 1.0 --post-decode-acwt 10.0 \

--nj 1 --num-threads 1 \

--cmd "$decode\_cmd" --iter final \

--frames-per-chunk 50 \

$dir/graph $test\_data $dir/decode\_chat001\_test

3.2.2 工作进度安排

第1周-第2周：学习网络上的模型

1. 第5周：模型初步建立

第6-第8周：逐步完善模型

1. 第12周：连接三个模块整合成为最终项目

第12-第13周：测试，进行最终完善

最终提交：可运行代码和项目文档

3.3 项目的开发内容和结果

3.3.1 系统需求分析

见《需求规格说明书》

3.3.2 系统设计

体系架构设计

智能聊天机器人系统总共分为三大模块，分别为语音识别模块：识别用户的语音输入并进行拼音处理；AI图灵模块：进行内层逻辑运算，包括语音分析，情感分析等；语音合成模块：将得到的结果通过语音合成的方式输出。

人机界面设计



系统接口设计

内层逻辑使用使用图灵AI接口，外层界面使用QT接口UIform界面设计

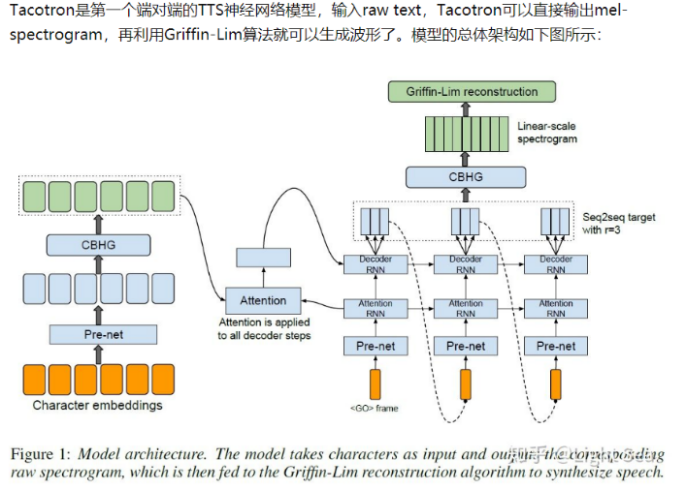
3.3.3 系统具体实现

语音识别部分通过采用卷积神经网络（CNN）和连接性时序分类（CTC）方法，使用大量中文语音数据集进行训练，将声音转录为中文拼音，并通过语言模型，将拼音序列转换为中文文本。

我们使用的的声学模型采用了深度全卷积神经网络，直接将语谱图作为输入。模型结构上，借鉴了VGG，这种网络模型有着很强的表达能力，可以看到非常长的历史和未来信息，相比RNN在鲁棒性上更出色。在输出端，这种模型可以和CTC方案可以完美结合，以实现整个模型的端到端训练，将声音波形信号直接转录为中文普通话拼音序列。在语言模型上，通过最大熵隐含马尔可夫模型，将拼音序列转换为中文文本。

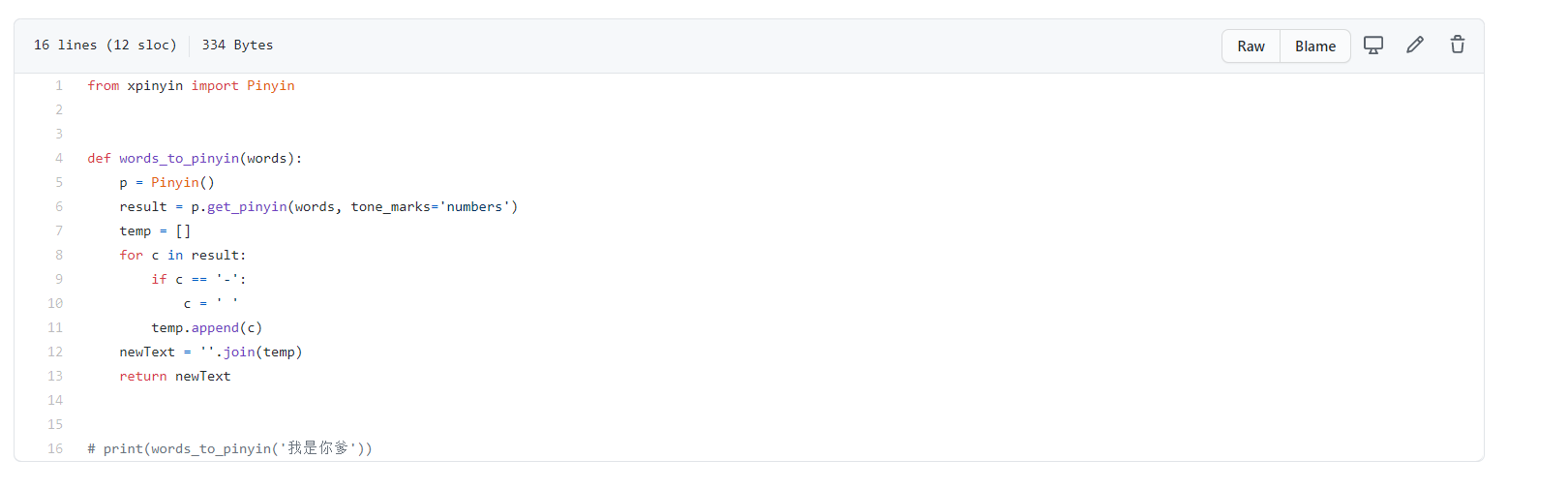
内层逻辑直接调用图灵AI的api，从而实现内部的回答逻辑

采用TACOTRON 模型，TACOTRON是一个端到端的TTS模型，模型核心是seq2seq + attention。模型的输入为一系列文本字向量，输出mel-spectrogram, 然后在使用Griffin\_lim算法生成对应音频。模型结构如下图：

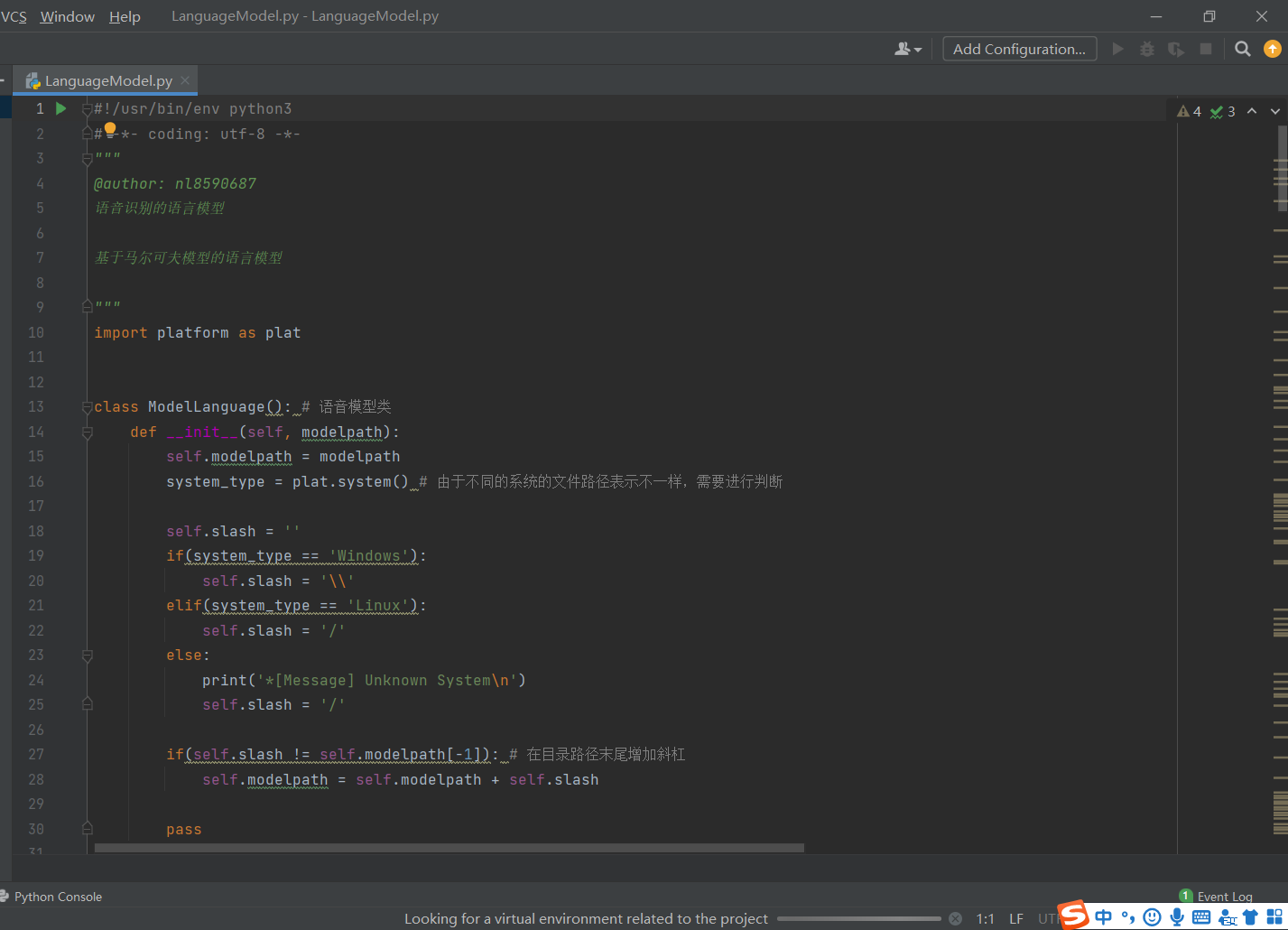


系统接口设计

转接口，文字转拼音部分桥接代码 文字转拼音.py文件：



语音识别模型代码 LanguageModel.py文件



3.3.4 系统测试

安装 kaldi， 然后，下载 http://kaldi-asr.org/models/m2 并解压到egs/cvte，保证文件 kaldi/egs/cvte/s5 文件存在。添加新的语音文件并进行识别测试。

主要测试内容：

#!/bin/bash

. ./cmd.sh

. ./path.sh

# step 1: generate fbank features

obj\_dir=data/chat001

for x in test; do

rm -rf fbank/$x

mkdir -p fbank/$x

# compute fbank without pitch

steps/make\_fbank.sh --nj 1 --cmd "run.pl" $obj\_dir/$x exp/make\_fbank/$x fbank/$x || exit 1;

# compute cmvn

steps/compute\_cmvn\_stats.sh $obj\_dir/$x exp/fbank\_cmvn/$x fbank/$x || exit 1;

done

# #step 2: offline-decoding

test\_data=data/chat001/test

dir=exp/chain/tdnn

steps/nnet3/decode.sh --acwt 1.0 --post-decode-acwt 10.0 \

--nj 1 --num-threads 1 \

--cmd "$decode\_cmd" --iter final \

--frames-per-chunk 50 \

$dir/graph $test\_data $dir/decode\_chat001\_test

# # note: the model is trained using "apply-cmvn-online",

# # so you can modify the corresponding code in steps/nnet3/decode.sh to obtain the best performance,

# # but if you directly steps/nnet3/decode.sh,

# # the performance is also good, but a little poor than the "apply-cmvn-online" method.

在解码阶段，执行的脚本如下：

# nnet3-latgen-faster --frame-subsampling-factor=3 --frames-per-chunk=50 --extra-left-context=0 --extra-right-context=0 --extra-left-context-initial=-1 --extra-right-context-final=-1 --minimize=false --max-active=7000 --min-active=200 --beam=15.0 --lattice-beam=8.0 --acoustic-scale=1.0 --allow-partial=true --word-symbol-table=exp/chain/tdnn/graph/words.txt exp/chain/tdnn/final.mdl exp/chain/tdnn/graph/HCLG.fst "ark,s,cs:apply-cmvn --norm-means=true --norm-vars=false --utt2spk=ark:data/chat001/test/split1/1/utt2spk scp:data/chat001/test/split1/1/cmvn.scp scp:data/chat001/test/split1/1/feats.scp ark:- |" "ark:|lattice-scale --acoustic-scale=10.0 ark:- ark:- | gzip -c >exp/chain/tdnn/decode\_chat001\_test/lat.1.gz"

# Started at Sat Aug 1 16:21:14 CST 2020

#

nnet3-latgen-faster --frame-subsampling-factor=3 --frames-per-chunk=50 --extra-left-context=0 --extra-right-context=0 --extra-left-context-initial=-1 --extra-right-context-final=-1 --minimize=false --max-active=7000 --min-active=200 --beam=15.0 --lattice-beam=8.0 --acoustic-scale=1.0 --allow-partial=true --word-symbol-table=exp/chain/tdnn/graph/words.txt exp/chain/tdnn/final.mdl exp/chain/tdnn/graph/HCLG.fst 'ark,s,cs:apply-cmvn --norm-means=true --norm-vars=false --utt2spk=ark:data/chat001/test/split1/1/utt2spk scp:data/chat001/test/split1/1/cmvn.scp scp:data/chat001/test/split1/1/feats.scp ark:- |' 'ark:|lattice-scale --acoustic-scale=10.0 ark:- ark:- | gzip -c >exp/chain/tdnn/decode\_chat001\_test/lat.1.gz'

LOG (nnet3-latgen-faster[5.5.765-f88d5]:RemoveOrphanNodes():nnet-nnet.cc:948) Removed 1 orphan nodes.

LOG (nnet3-latgen-faster[5.5.765-f88d5]:RemoveOrphanComponents():nnet-nnet.cc:847) Removing 2 orphan components.

LOG (nnet3-latgen-faster[5.5.765-f88d5]:Collapse():nnet-utils.cc:1472) Added 1 components, removed 2

lattice-scale --acoustic-scale=10.0 ark:- ark:-

apply-cmvn --norm-means=true --norm-vars=false --utt2spk=ark:data/chat001/test/split1/1/utt2spk scp:data/chat001/test/split1/1/cmvn.scp scp:data/chat001/test/split1/1/feats.scp ark:-

LOG (nnet3-latgen-faster[5.5.765-f88d5]:CheckAndFixConfigs():nnet-am-decodable-simple.cc:294) Increasing --frames-per-chunk from 50 to 51 to make it a multiple of --frame-subsampling-factor=3

CHAT001\_20200801\_001 上海 浦东机场 入境 房 输入 全 闭 环 管理

LOG (nnet3-latgen-faster[5.5.765-f88d5]:DecodeUtteranceLatticeFaster():decoder-wrappers.cc:375) Log-like per frame for utterance CHAT001\_20200801\_001 is 2.19918 over 208 frames.

LOG (apply-cmvn[5.5.765-f88d5]:main():apply-cmvn.cc:162) Applied cepstral mean normalization to 2 utterances, errors on 0

CHAT001\_20200801\_002 北京 地铁 宣武门 站 综合 改造 新增 换乘 通道

LOG (nnet3-latgen-faster[5.5.765-f88d5]:DecodeUtteranceLatticeFaster():decoder-wrappers.cc:375) Log-like per frame for utterance CHAT001\_20200801\_002 is 2.19511 over 333 frames.

LOG (nnet3-latgen-faster[5.5.765-f88d5]:main():nnet3-latgen-faster.cc:256) Time taken 10.9386s: real-time factor assuming 100 frames/sec is 0.673972

LOG (nnet3-latgen-faster[5.5.765-f88d5]:main():nnet3-latgen-faster.cc:259) Done 2 utterances, failed for 0

LOG (nnet3-latgen-faster[5.5.765-f88d5]:main():nnet3-latgen-faster.cc:261) Overall log-likelihood per frame is 2.19668 over 541 frames.

LOG (nnet3-latgen-faster[5.5.765-f88d5]:~CachingOptimizingCompiler():nnet-optimize.cc:710) 0.00447 seconds taken in nnet3 compilation total (breakdown: 0.00219 compilation, 0.00168 optimization, 0 shortcut expansion, 0.000385 checking, 1.1e-05 computing indexes, 0.000209 misc.) + 0 I/O.

LOG (lattice-scale[5.5.765-f88d5]:main():lattice-scale.cc:107) Done 2 lattices.

# Accounting: time=53 threads=1

# Ended (code 0) at Sat Aug 1 16:22:07 CST 2020, elapsed time 53 seconds

4 讨论与体会

4.1 对项目过程的体会

在AI潮流席卷全球时，围绕语音交互的产品之争正愈演愈烈，苹果siri、亚马逊echo、微软小冰这些产品风靡全球的同时，国内外科技巨头、创业团队也在暗流涌动，各种智能音箱以及语音解决方案层出不穷。

这种顺势发展的方式，造就了语音交互已经成为人工智能领域最成熟也是落地最快的技术。尤其是深度学习的起势，让语音识别、语音合成以及自然语言处理的发展速度提升到了一个新的高度。因此我们小组最终选择了语音识别和语音合成领域进行学习并决定开发一个基于语音识别和语音合成的智能聊天机器人

经过几个月有关语音识别学习的摸爬滚打，勉强算是了解了这个领域的基本知识了。写下这篇学习历程和心得，来记录和总结所学以让自己更好地前进，

语音识别其实是基于深度神经网络学习的，所以要整体上明白深度学习的应用形式是通过程序实现，深度学习可以说是一类算法。而深度学习的核心部分是数学理论。在学习其理论知识时，我阅读了《人工神经网络理论、设计及应用》这本韩力群的书，同时在网易云课堂上观看吴恩达的微专业课程《深度学习工程师》

有了一定的深度学习理论知识后，就可以着手学习语音识别的相关知识了。我在网上了解到语音识别模型主要有Connectionist Temporal Classification、sequence to sequence speech recognition with attention、Listen Attend and Spell等主流模型，最终我选择了Connectionist Temporal Classification模型来进行学习。CTC由Alex Graves等人在ICML 2006上提出来，目的是实现RNN直接对序列数据进行学习，而无需事先标注好训练数据中输入序列和输入序列的映射关系，使得RNN模型在语音识别等序列学习任务中取得更好的效果。传统的语音识别的声学模型训练，对于每一帧的数据，需要知道对应的label才能进行有效的训练，在训练数据之前需要做语音对齐的预处理。而语音对齐的过程本身就需要进行反复多次的迭代，来确保对齐更准确，这本身十分耗时。与传统的声学模型训练相比，采用CTC作为损失函数的声学模型训练，是一种完全端到端的声学模型训练，不需要预先对数据做对齐，只需要一个输入序列和一个输出序列即可以训练。这样就不需要对数据对齐和一一标注，并且CTC直接输出序列预测的概率，不需要外部的后处理。CTC引入了blank（该帧没有预测值），每个预测的分类对应的一整段语音中的一个spike（尖峰），其他不是尖峰的位置认为是blank。对于一段语音，CTC最后的输出是spike（尖峰）的序列，并不关心每一个音素持续了多长时间。这种神经网络+CTC的结构除了可以应用到语音识别的声学模型训练上以外，也可以用到任何一个输入序列到一个输出序列的训练上（要求：输入序列的长度大于输出序列）。

比如，OCR识别也可以采用RNN+CTC的模型来做，将包含文字的图片每一列的数据作为一个序列输入给RNN+CTC模型，输出是对应的汉字，因为要好多列才组成一个汉字，所以输入的序列的长度远大于输出序列的长度。而且这种实现方式的OCR识别，也不需要事先准确的检测到文字的位置，只要这个序列中包含这些文字就好了。

至于编程,我学习和使用的框架是TensorFlow，它由谷歌开发，是一个基于数据流编程（dataflow programming）的符号数学系统，被广泛应用于各类机器学习（machine learning）算法的编程实现，其前身是谷歌的神经网络算法库DistBelief，也是目前最受欢迎的深度学习框架。在TensorFlow内的编程是用python高级程序语言，python因为其灵活性和少量代码就能实现复杂功能而成为许多深度学习框架的语言。我学习这个框架在一开始主要通过观看学习视频

除了通过看上述视频，同时阅读技术书籍也是必要的。黄文坚、唐源编著的《TensorFlow实战》知名度很高，它除了简要介绍了TensorFlow编程模型等基础知识，也主要从如何编写代码角度分析了如何写手写数字识别程序，自编码器，多层感知器，简单以及进阶的卷积网络。

以上关于用TensorFlow实现深度学习理论的书籍和视频教程其根本来源于TensorFlow官网（占主要部分）以及相对应的Github库：tensorflow官网；tensorflow的Github网址。我花了将近一周的时间也没有把代码调通，最后通过看官网的技术文档，不到半天就把代码调通了。这是因为博客会因为TensorFlow的API、版本不断升级拓展而过时，而官网的技术文档则会随时更新，由此可见阅读原始官网技术文档的重要性，即使英文有点难阅读。

由于在学习深度学习时我有python编程基础，因此学习TensorFlow时会相对简单一点， 在学习的过程中，因为会面临大量代码和文件，也会记下很多学习笔记，这个时候管理和记录的能力就显得非常重要了，我的方法是用平板记下心得和学习笔记，因为相对于纸质笔记本，平板的笔记本软件能更好地进行管理。

4.2 对项目的评价

我们小组的项目实现了基本的功能，也就是的人机通过简单自然语言进行交互，其中语音识别和语音合成模块由我们小组自主实现，人机交互的人工智能处理部分调用的是图灵的api，总的来说项目实现的功能和效果还算符合预期，能满足基本的需求，没有特别惊艳但也还不错。

5 小结

自3月份开始，我一直在跟进项目管理课程的智能聊天机器人系统开发工作，至此为止已近四个月时间， 从语音识别的相关知识学习到系统的开发以及小组内部系统测试、验收测试等等。从开始到项目即将结束，一步步走过来。本次项目中，我作为系统开发和测试的主力人员之一，对此项目中的工作进行总结。

项目的开发进度主要是按照项目计划进行的，完全按照项目组计划要求完成开发任务、提交开发相关文档，包括项目需求的完善、制定开发计划、执行开发计划、以及测试等。这次开发过程中我认真配合项目工作，和小组其他成员一起共同保证项目质量，按照规划时间完成系统开发的工作。

项目组内部成员关系处理。在项目工作的这几个月里大家相处非常融洽，项目组内部共同探讨解决问题的方法，向各模块负责人学习模块功能处理方式，共同进行系统功能模块的开发。鉴于之前对其他系统进行的开发经历，我也积极向项目组提出了一些完善性意见。

系统开发期间，让我学到了很多东西，不仅使我在理论上对IT领域有了全新的认识，在实践能力上也得到了提高，真正地做到了学以致用，更学到了很多做人的道理，对我来说受益非浅。除此以外，我还学会了如何更好地与别人沟通，如何更好地去陈述自己的观点，如何说服别人认同自己的观点。第一次亲身感受到理论与实际的相结合，让我大开眼界。也是对以前所学知识的一个初审吧!这次实训对于我以后学习、找工作也真是受益菲浅，在短短几个月中相信这些宝贵的经验会成为我今后成功的重要的基石。

作为一名大三的学生，经过差不多三年的在校学习，对软件项目有了理性的认识和理解。在校期间，一直忙于理论知识的学习，比较少有机会来参与项目的开发。所以在上这门课之前，我对项目开发的经验还有所欠缺，一个完整的项目要怎么来更好地分工以及完成该项目所要的基本步骤也需要更深刻地了解。而经过这次课程开发实践，让我明白一个完整项目的开发，必须由团队来分工合作，并在每个阶段中进行必要的总结与论证。一个完整项目的开发它所要经历的阶段包括:规划和用例说明、项目结构和风险评估、业务功能说明书、详细设计说明书、代码实现、测试、操作手册。一个项目的开发所需要的财力、人力都是大量的，如果没有一个好的远景规划，对以后的开发进度会产生很大的影响，严重时导致在预定时间内不能完成该项目或者完成的项目跟原先计划所要实现的项目功能不符合。一份好的项目结构、业务功能和详细设计说明书对一个项目的开发有明确的指引作用，它可以使开发人员对这个项目所要实现的功能在总体上有具体的认识，并能减少在开发过程中出现不必要的脱节。代码的实现是一个项目开发成功与否的关键，可以说，前面所做的事情就是为代码的实现做铺垫。感谢老师的教导,在短短几个月的学习和开发实践中老师能充分将自己的知识传授给我们，并且有耐心的给我们讲解所有问题，帮助每位同学了解项目开发的技能。

参考资料

附录1项目开发计划

1 引言

2 项目概述

3 实施计划

4 支持条件

5 专题计划要点

附录2需求规格说明书（包括需求分析模型-功能、数据、行为模型等）

1 引言

2 任务概述

3 需求规定

4 运行环境规定

附录3设计文档（包括设计模型-软件架构、数据结构、用户界面、功能模块等）

概要设计说明书

1 引言

2 总体设计

3 接口设计

4 运行设计

5 系统出错处理设计

详细设计说明书

1 引言

2 系统的结构

3模块1设计说明

4模块2设计说明

5模块3设计说明

。。。

附录4用户文档说明书（包括软件的描述、安装和使用说明等）

1 引言

2 软硬件环境

3 安装说明

4 操作说明

5 功能列表