第十四届中国研究生电子设计竞赛

技术论文

中文题目：基于 TACOTRON 端到端语音合成模型的改进方案

英文题目：An improved scheme based on TACOTRON end-to-end speech synthesis model

参赛单位：南京航空航天大学

队伍名称：奔跑吧小白

指导老师：李海林

参赛队员：杨凌辉、鲍悦、张嘉纹、高璇、毛健

完成时间：（2019-6-17）

摘要

TTS（文本到语音）模型包含许多模块，例如文本分析，声学模型，音频合成等。这些模块的构建需要大量专业相关的知识及特征工程，将花费大量的时间与精力，同时各个模块之间组合在一起也会产生很多新的问题。TACOTRON 是一个直接从文本合成语音的神经网络架构，它将各模块放入一个黑箱，我们无需花费大量时间了解 TTS 中的各模块或者专业领域知识，可直接通过深度学习训练出一个 TTS 模型。

传统 TACOTRON 模型后端通过 Griffin-Lim 算法合成语音，合成语音保真度相较于传统算法较好，但是依然保持了 Griffin-Lim 从频谱到语音时域信号的转换方法，机器味依然存在。可实时高质量合成音频的 WaveRNN 网络模型能够解决这一问题。WaveRNN 在质量和计算资源消耗方面获得平衡，是语音合成方面一类有效算法。

本文首先分析了语音合成技术目前的难点，在此基础上提出新的思路。我们选取 Thchs30 中文语料库和 LJSpeech 英文语料库进行模型训练，希望实现中、英文语音合成。通过对比分析各类语音合成算法的合成效果，我们选择 TACOTRON 模型作为基础架构，并对其进行一定的改进，在后处理网络添加 WaveRNN 模块实现语音合成。我们摒弃了声码器传统合成算法，设计了一个直接从文本合成语音的端到端神经网络架构，并设计人机交互界面方便使用。本文详细介绍了 TACOTRON 原理及 WaveRNN 算法，展示了改进的 TACOTRON 模型，基于此实现了软件设计。最后进行系统测试，结果表明，本文采用的方案具有一定的可行性、准确性。

关键词：TTS（文本到语音）、TACOTRON、WaveRNN、深度学习

Abstract

The TTS (text-to-speech) model contains many modules， For example, text analysis, acoustic models, audio synthesis, etc. The construction of these modules requires a lot of professional-related knowledge and feature engineering, which will take a lot of time and effort, and the combination of modules will also create many new problems. TACOTRON is a neural network architecture that synthesizes speech directly from text. It puts each module into a black box. We don't need to spend a lot of time to understand the modules or professional domain knowledge in TTS, instead ,we can train a TTS model directly through deep learning.

The traditional TACOTRON model backend synthesizes speech through the Griffin-Lim algorithm. The synthesized speech fidelity is low, and the "machine flavor" is still strong. The WaveRNN network model that can synthesize audio in real time can solve this problem. WaveRNN balances quality and computing resource consumption and is an effective algorithm for speech synthesis.

This paper first analyzes the current difficulties of speech synthesis technology, and proposes new ideas on this basis. We selected Thchs30 Chinese corpus and LJSpeech English corpus for model training, hoping to achieve Chinese and English speech synthesis. By comparing and analyzing the synthesis effects of various speech synthesis algorithms, we choose the TACOTRON model as the infrastructure and make some improvements to it. We add WaveRNN module to the speech synthesis in the post-processing network. We have designed an end-to-end neural network architecture that directly synthesizes speech from text, and designed a human-computer interaction interface for ease of use. This paper introduces the TACOTRON principle and the WaveRNN algorithm in detail, shows the improved TACOTRON model, and implements the software design based on this. Finally, the system test is carried out. The results show that the scheme adopted in this paper has certain feasibility and accuracy.

Keywords:TTS(Text-To-Speech)、TACOTRON、WaveRNN、Deep Learning

目　　录

第 1 章 作品难点与创新

1.1 作品难点

从文本生成自然语音（语音合成，TTS）经历了几十年的研究，仍是一项有挑战的任务。在基于统计学的模型中通常需要实现一个文本前端来抽取丰富的语言学特征，持续时间模型，声学特征预测模型和一个复杂的基于信号处理的声码器。这些子模块不仅需要领域知识，在设计实现上也很困难。此外，由于这些子模型进行相互独立的训练，每个组件的错误可能会复合。上述 TTS 系统的设计复杂性导致在构建新模型的时候必须要进行大量的工程方面的努力。单元挑选和拼接式合成方法，是一项把预先录制的语音波形的小片段缝合在一起的技术，过去很多年中一直代表了最高水平。统计参数语音合成方法，是直接生成语音特征的平滑轨迹，然后交由声码器来合成语音，这种方法解决了拼接合成方法中出现的边界人工痕迹的很多问题。然而由这些方法构造的系统生成的语音与人类语音相比，经常模糊不清并且不自然。

TACOTRON 作为集成的端到端 TTS 系统具有许多优点：它可以减少繁重的特征工程需要，更容易对各种属性（如说话者或语言）或情绪等高级功能进行丰富的调节，对新数据的适应也可能更容易，单个模型可能比多阶段模型更稳健。TACOTRON 使用 Griffin-Lim 算法，会产生特有的人工痕迹并且合成的语音保真度较低。另一方面，Griffin-Lim 算法是基于 CPU 实现，无法实现实时语音合成。

WaveRNN 是一种强大的音频生成模型。它适用于 TTS，但由于其样本级别的自回归性质，训练速度会变得很慢。它还需要对现有 TTS 前端的语言特征进行调节，因此不是端到端：它只取代声码器和声学模型。

因此，我们需要设计一个保真度高的实时端到端 TTS 系统。

1.2 作品创新点

本文结合了 TACOTRON 和 WaveRNN，组成统一的神经网络语音合成模型。在实现端到端合成 TTS 系统的优点同时，后接 WaveRNN 声码器，提高了合成的语音的自然度。

在全球一体化的今天，汉语与英语成为大部分人日常工作中使用最频繁的语言，为了使得系统具有更高的普适性，我们选用了中文语料库 Thchs30 和英文语料库 LJSpeech 进行模型训练，同时实现了中文及英文的语音合成。

为了提高用户体验，我们为该系统设计了友好的人机交互界面，让语音合成的实现接口化，操作简单。

第 2 章 方案论证与设计

2.1 语料库的选择

我们选用了中文语料库 Thchs30 和英文语料库 LJSpeech。两种数据中，汉语数据属于相对难以处理的数据，汉字数量庞大，因此考虑到用汉语拼音作为网络训练的字符集，可以将字符集限制在 a-z 范围内，很好的和英文字符集统一到了一起，为同时识别中文以及英文提供了可能在汉语的处理过程中，由于同音字较多，因此在处理过程中还需要通过分析上下文信息以确定相应的同音字。

Thchs30(Tsinghua Chinese 30 hour database)是清华大学语言与语言技术中心发布的开放式中文语音数据库。Thchs30 语料库是在安静的办公室环境下，通过单个碳粒麦克风录取的，总时长约 33.5 h，共 13388 句话，每句平均 20 个字，平均时长 9 s。其选取了大容量的新闻，由 30 个会讲流利普通话的大学生录制。采样率为 16kHz，数据为 16bit，这些录音根据其文本内容分为了四个部分，A（句子的 ID 是 1~250），B（句子的 ID 是 251~500），C（501~750），D（751~1000）。ABC 三组包括 30 个人的 10893 句发音，用来做训练，D 包括 10 个人的 2496 句发音，用来做测试。

LJSpeech 英语语料库是一个开放式的英文语音数据集，由 13100 个简短的音频剪辑组成，其中的演讲者阅读 7 本非小说类书籍的段落。LJSpeech 语料库的总时长约 24 h，共 13 100 句话，每句平均 17 个单词，平均时长 6.6 s。语料库中的每个音频文件都是单通道 16 位 PCM WAV，采样率为 22050 Hz。将本文实验需用的两种语料库对比如下：

表 2-1 语料库对比

	Thchs30	LJSpeech
语言	中文	英文
采样率（kHz）	22	16
语量	13388	13100
时长（h）	33.5	24
平均时长（s）	9	6.6

2.2 语音数据特征

对于语音数据而言，从自然界中直接获取的语音含有噪声，且包含了大量的冗余信息，需对其进行一系列的预处理，然后将音频信号中具有辨识度的特性提取出来,以压缩数据处理量。基于有效性和可靠性两个方面，特征参数一定要有很好的区分性，并且要具有较强的鲁棒性。最常用的语音特征是梅尔倒谱系数(Mel Frequency Cepstral Coefficient, MFCC)，其是在 Mel 标度频率域提取出来的倒谱参数，Mel 标度描述了人耳频率的非线性特性，更符合人耳的听觉特性。梅尔倒

谱系数是在 1980 年由 Davis 和 Mermelstein 提出的，从那时起，作为一种简单可用并且切合人主观感受的音频特性，MFCC 一直广泛地应用于语音识别领域。

（1）预加重

研究表明，人在发声过程中发声器会对频谱造成影响，使得信号高频部分受到压制，通常当频率高于 800 Hz 的频段会有 6dB 的衰减。因此，预加重的目的在于提升高频部分，使信号的频谱变得平坦，保持在低频到高频的整个频带中，能用同样的信噪比求频谱。同时，也是为了消除发生过程中声带和嘴唇的效应，来补偿语音信号受到发音系统所抑制的高频部分，突出高频的共振峰。预加重操作的时域表达式与 z 域表达式如下：

$$x_n = s_n - \alpha s_{n-1} \tag{2.1}$$

$$H(z) = 1 - \alpha z^{-1} \tag{2.2}$$

其中 s_n 为原始信号的第 n 个采样点，α 为预加重系数，其值一般介于 0.94-0.97 之间。

（2）分帧与加窗

由于语音信号的非平稳性，其在不同的时间点由于说话内容的不同而形成不同的波形，因此将语音信号分割成若干段短时间信号，更利于对其波形进行观察。分帧是将语音信号分成若干帧，把 N 个采样点作为一个观测单位为一帧，对信号以固定长度（通常为 10ms-30ms）进行分割，此时我们可以认为短时间内的语音信号是平稳的。为了确保帧与帧之间更加平滑的过渡，采样时让两个相邻的帧之间有一定程度的重叠，这一重叠部分即为帧移，其长度通常取每一帧长度的 1/2 或 1/3。

加窗即让每帧语音信号与窗函数相乘，其目的主要有两个：一是降低两相邻帧间的不连续性，避免出现吉布斯效应；二是使原本没有周期的语音信号呈现出周期函数的部分特征。若选用汉明窗:

$$w(n) = 0.54 - 0.46\cos\frac{2\pi n}{N-1} \quad , \quad 0 \le n \le N-1 \tag{2.3}$$

假设分帧后的信号为 $x(n), n = 0,1,...,N-1$，N 为帧的大小，则经过加窗后的信号为

$$y(n) = x(n) \times w(n) \tag{2.4}$$

（3）快速傅里叶变换（Fast Fourier Transform，FFT）

由于信号在时域上的变换很难看出信号的特性，所以通常将它转换为频域上的能量分布来观察，不同的能量分布代表不同语音信号的特性。所以在乘上汉明窗后，每帧还必须再经过快速傅里叶变换以得到在频谱上的能量分布。语音信号的 DFT 及语音信号的功率谱分别为：

$$Y(k)=\sum_{n=0}^{N-1}y(n)e^{-j2\pi k/N}\quad ,\quad 0\le k\le N \tag{2.5}$$

$$E(k)=\left|Y(k)\right|^2 \tag{2.6}$$

式中 y(n)为分帧加窗后的语音信号，N 表示傅里叶变换的点数。

（4）Mel 滤波器组

人的听觉系统是一个特殊的非线性系统，它响应不同频率信号的灵敏度是不同的。因此需要将傅里叶谱变换到梅尔频谱，梅尔频率的变换比较符合人的直观感受。傅里叶频率和梅尔频谱的关系可表示为：

$$f_{mel}=1120\cdot\ln(1+\frac{f}{700}) \tag{2.7}$$

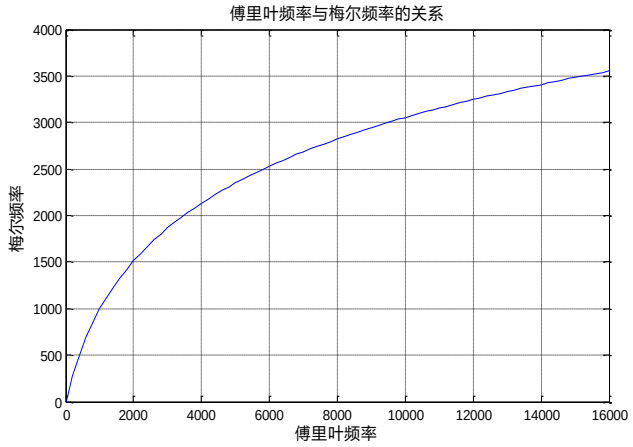


图 2-1 傅里叶频率与梅尔频率的关系

将梅尔频率上等间隔采样的点映射回频率域后可获得多个梅尔滤波器，此滤波器在梅尔频率内是等宽的，但在频率域上则表现为随着频率增大而逐渐变宽的三角形。此三角带通滤波器能够对频谱进行平滑化，并消除谐波的作用，突显出原先语音的共振峰。定义一个有 M 个滤波器的滤波器组，通常取 22-26，本文取 M=24，其每一个子滤波器的频率响应可定义为：

$$H_m(k)=\begin{cases}\dfrac{2(k-f(m-1))}{(f(m+1)-f(m-1))(f(m)-f(m-1))}, & f(m-1)<k\le f(m)\\ \dfrac{2(f(m+1)-k)}{(f(m+1)-f(m-1))(f(m)-f(m-1))}, & f(m)\le k\le f(m+1)\\ 0, & others\end{cases} \tag{2.8}$$

将（2.3）中的频谱通过 Mel 滤波器组得到 Mel 频谱，将线形的自然频谱转换为体现人类听觉特性的 Mel 频谱。

（5）计算每个滤波器组输出的对数能量

将能量谱与梅尔滤波器相乘后进行数值积分后取对数可得到：

$$S(m)=\ln(\sum_{k=0}^{N}E(k)H_m(k))=\ln(\sum_{k=0}^{N}|Y(k)|^2H_m(k)),0\le m\le M \tag{2.9}$$

（6）离散余弦变换（Discrete Cosine Transform，DCT）

经离散余弦变换得到 MCFF 系数：

$$C(n)=\sum_{m=0}^{M}S(m)\cos(\frac{\pi n(m-0.5)}{M}),n=1,2,...,L \tag{2.10}$$

将上述的对数能量带入离散余弦变换，求出 L 阶的 Mel-scale Cepstrum 参数。L 阶指 MFCC 系数阶数，通常取 12-16。

2.3 TTS 算法对比

2.3.1 Griffin-Lim 算法

Griffin-Lim 算法是在仅已知幅度谱、未知相位谱的条件下重建语音的算法。Griffin-Lim 算法将 Seq2Seq 的输出转化成被合成为波形的目标表达，使得估计得到的信号傅里叶变换的幅度值与原始信号傅里叶变换的幅度值的平方误差达到最小。通过迭代重构信号的相位信息和已知的幅度信息，得到语音信号的估算值。当两者的差值达到一个比较小的值后，则停止迭代，就认为第 i 次迭代后所获得的语音信号是从信号傅里叶变换幅度值重构的原始语音信号。对于不同的迭代次数，将原始波形与重构波形对比如下图，可得到在这种迭代算法下 D 会不断下降，即使原来不依赖原始相位，我们也能很大程度上还原波形。

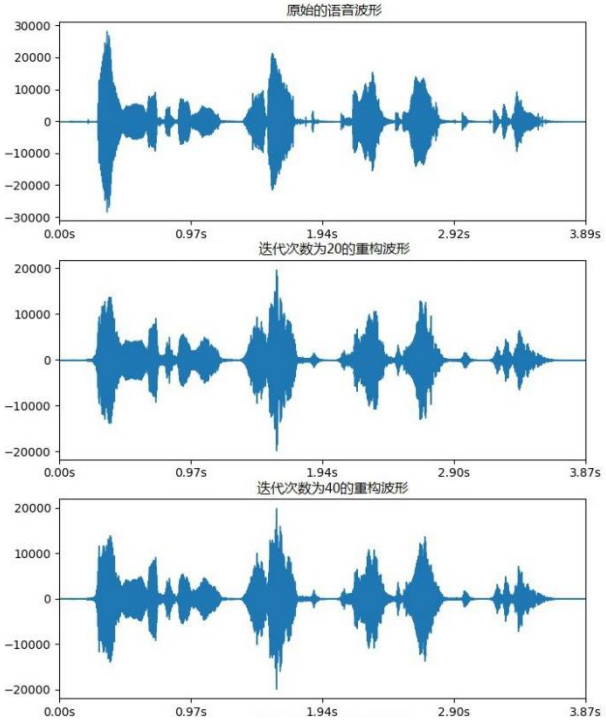


图 2-2 原始波形与重构波形对比

2.3.2 WaveRNN

WaveRNN 模型是具有双 softmax 层的单层 RNN，旨在有效地预测 16 位原始音频样本。对 WaveRNN 模型进行质量与采样性能测评如下：

在单人北美英语文字转语音数据库上进行测试，输入是语言特征向量，输出是 24kHz，16 位的波形。在验证集上进行三项评估：负对数似然函数（NLL），人为评价的平均主观意见分（MOS），以及人为评价模型之间的 A/B 对比测试（分数在-3 非常差和+3 非常好之间）。

表 2-2 给定模型/WaveRNN 模型的对比测试结果

资料来源：Efficient Neural Audio Synthesis

模型/WaveRNN 模型	更好	中立	更差	总计	重要性
WAVENET 512 (60)	145	529	126	0.02 ± 0.08	否
SPARSE WR 384 (2048/96.4%)	139	441	220	-0.14 ± 0.08	是
SPARSE WR MOBILE	71	456	237	-0.40 ± 0.09	是
SUBSCALE WR 1024 (16×)	113	558	129	-0.03 ± 0.05	否

从 A/B 对比试验中可以看出，仅有 896 单元的 WaveRNN 与最大的 WaveNet 达到了相当的 NLL 评分，而且音频保真度之间也没有很大的差距，而且 MOS 评分均很高。WaveRNN 在产生每一个 16 位样本时，仅用了 N=5 次的矩阵矢量相乘，就达到了这样的表现性能。而 WaveNet 用了两层网络，每层 30 个残差块（residual block），即 N=30*2=60 次矩阵矢量相乘。我们在实现过程中将 TACOTRON 模块声码器部分的 Griffin-Lim 实现替换为 WaveRNN 模型的声码器，来高效率地合成高质量语音。

第 3 章 原理分析

3.1 TACOTRON

TACOTRON 系统通过一个循环 Seq2Seq 结构的特征预测网络，把字符向量映射到线性频谱图为了最终合成出幅度谱图，TACOTRON 使用 Griffin-Lim 算法估计相位，然后施加一个短时傅里叶逆变换。

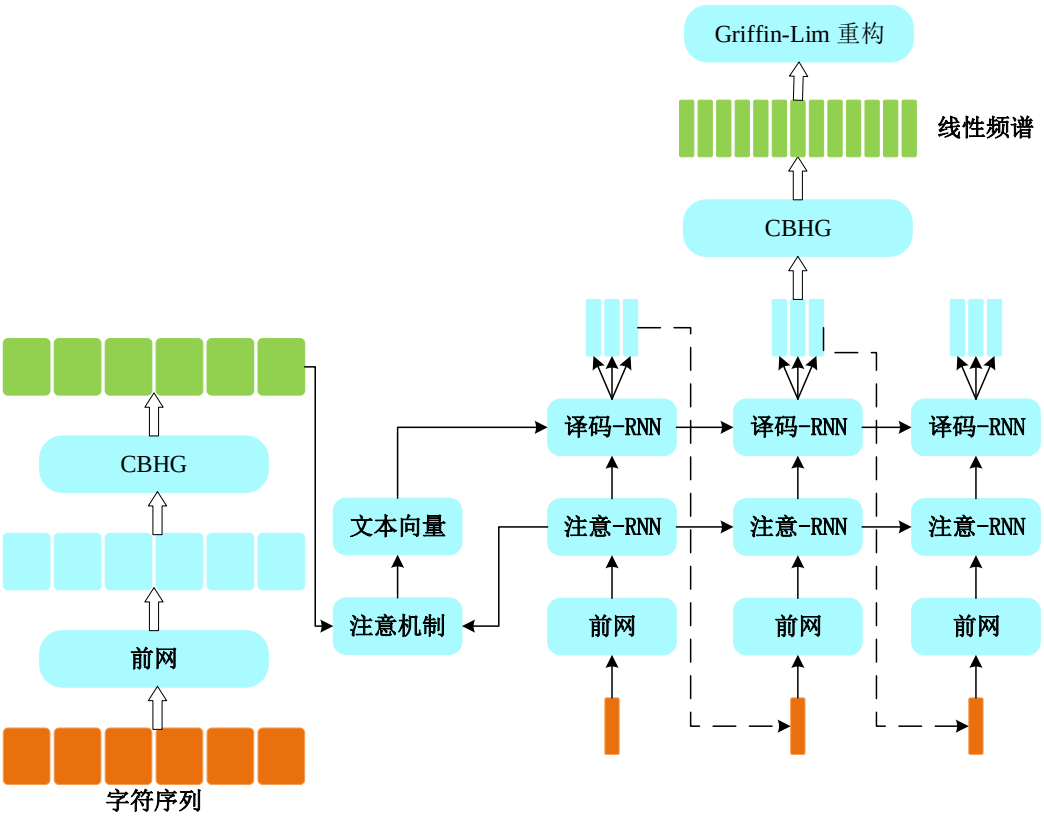


图 3-1 TACOTRON 结构图

3.1.1 Seq2Seq 架构

一般模型的输入特征通常是一个固定大小的矩阵，机器翻译输入的句长不固定，而输入的长度必须一致。Seq2Seq 结构的输入序列和输出序列长度是不固定的，因而可以解决这个问题。通常我们把 RNN 的输入称为“上下文”，通过编码器产生表示此上下文的向量 C。

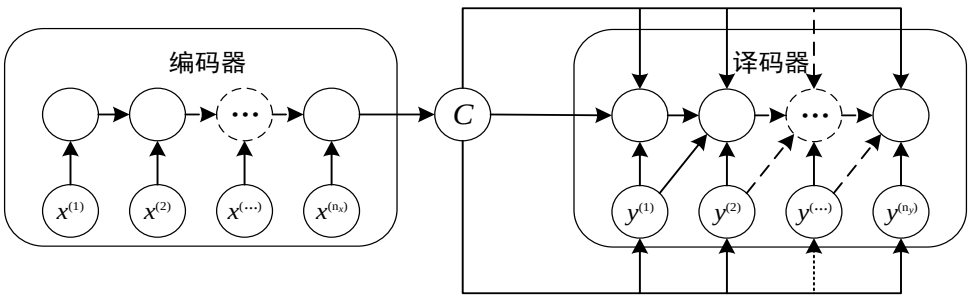


图 3-2 Seq2Seq 网络结构

Seq2Seq 网络通常包括两部分：

编码器：主要用于处理 RNN 的输入序列 $X=\{x^{(1)},x^{(2)}...x^{(n_x)}\}$，将最后一个 RNN 的单元状态来作为最终的输出的上下文 C。

解码器：它以编码器的输出 C 作为输入，以固定长度的向量作为条件，产生输出序列 $Y=\{y^{(1)},y^{(2)}...y^{(n_y)}\}$。

这种架构输入序列的长度不需和输出序列的长度保持一致。而编码器的输出上下文 C 的维度太小，因此它很难概括长序列完整的语义细节信息。引入注意机制并让 C 成为一个可变长度的序列，可以解决这个问题。

3.1.2 注意机制

注意机制主要包括三个部分：

编码器：用于读取原始数据，并将其转化成分布式的表示，其中一个特征向量与一个单词相互对应，这里使用的结构为 RNN；

存储器：主要用于存储编码器输出的特征向量列表；

解码器：利用存储器的内容，顺序的执行任务，每次任务聚焦于某个或几个具有不同权重的内容；

注意机制的结构如下图所示：

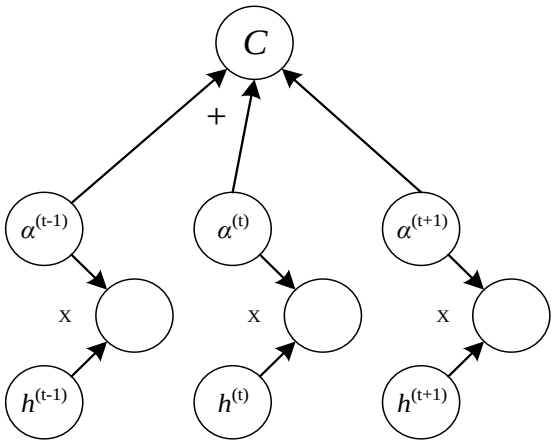


图 3-3 注意机制示意图

对上下文向量 C 有：

$$C=\sum_{t=1}^{T}\alpha^{(t)}h^{(t)} \tag{3.1}$$

其中 $h^{(t)}$为编码器输出的特征向量，在编码器中使用了 RNN，$h^{(t)}$即为 RNN 中隐藏层的状态 S，$\alpha^{(t)}$为权重。权重向量在译码器每次进行预测时都不一样，计算方法如下：

$$a_{ij}=\frac{\exp(e_{ij})}{\sum_{K=1}^{T}\exp(e_{ij})} \tag{3.2}$$

其中的 e_{ij} 代表了编码器的第 j 个输入与译码器的第 i 个输出的匹配程度，

这里把匹配程度转化为了概率来表示。

3.1.3 TACOTRON 结构

如图 3-1 所示，注意机制连接了编码器与译码器模块用于预测线性频谱的帧序列，后处理网络模块通过 Griffin-Lim 合成语音。

(a) 编码器

编码器的目标是提取文本的高鲁棒性序列表示。编码器的输入是字符序列，其中每个字符表示为 one-hot 矢量并嵌入到连续矢量中。然后对每个嵌入施加一组非线性变换，统称为前网。

前网是一个 3 层的网络结构，其主要功能是对输入进行一系列的非线性的变换，这样有助于模型收敛和泛化。它有两个隐藏层，层与层之间的连接均是全连接；第一层的隐藏单元数目与输入单元数目一致，第二层的隐藏单元数目为第一层的一半。

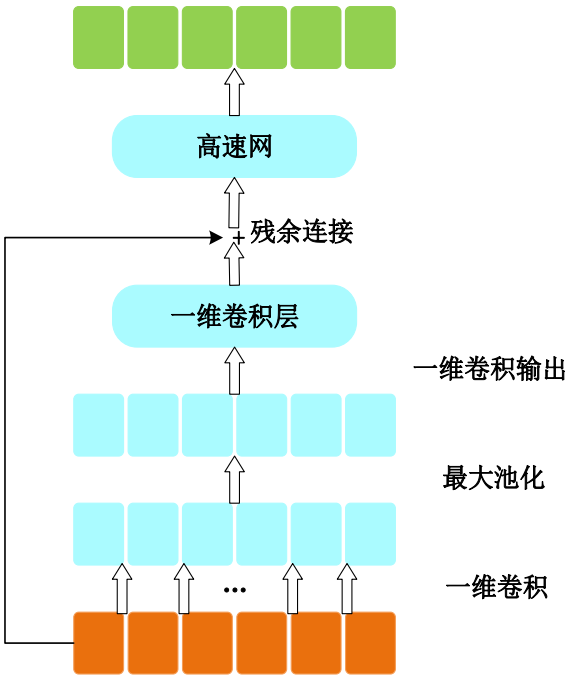


图 3-4 CBHG 结构图

CBHG 模块将前网输出转换为最终编码器表示以便被注意机制模块使用。CBHG 是一个功能强大的模块，用于从序列中提取表示信息。输入序列首先会经过一个卷积层，它有 K 个大小不同的 1 维的滤波器，其大小为 1,2,3…K。这些大小不同的卷积核提取了长度不同的上下文信息。然后，将经过不同大小的 k 个卷积核的输出堆积在一起。进而沿着时间轴对每一个时间步的结果进行最大池化，以增加局部不变性。经过池化之后，会再经过两层一维的卷积层。卷积输出被馈送到多层高速网以提取高级特征。最后，在顶部堆叠双向门控循环单位(GRU)以及 RNN 以提取前向和后向文本特征。

(b) 译码器

译码器是一个自回归的循环神经网络，它从经过编码的输入序列预测输出声谱图，一次预测一帧。译码器模块主要分为三部分：前网、注意-RNN、译码-RNN。

前网的结构与编码器中的前网相同，主要是对输入做一些非线性变换。注意-RNN 将前网的输出和注意机制模块的输出作为输入，经过具有垂直残余连接的堆栈 GRU 单元后输出到译码-RNN 中。译码-RNN 的输出为输入与经过 GRU 单元输出之和。

残余连接可以加速收敛。虽然整段文本的原始频谱图可直接预测，但它是一种高度冗余的表示。可将 Seq2Seq 译码目标设定为不同于原始语谱图的输出目标，之后再进行波形合成。Seq2Seq 译码目标可以是一种高度压缩的信息表示，只要它能为反演过程提供足够的可懂度和韵律信息，即可实现修复或训练。

由于每个字符在发音的时候，可能对应了多个帧，因此每个 GRU 单元输出为多个帧的音频文件。该模型可以减少训练模型的大小，同时减少了训练的时间，收敛速度大大提高。

(c) 后处理网络

后处理网络的任务是将 Seq2Seq 目标转换为可以用于波形合成的目标。我们使用 Griffin-Lim 作为合成器，后处理网络学习如何预测在线性频率范围内采样的频谱幅度，进而把线性频谱特征表达逆变换为时域波形样本。后处理网的另一个作用是它可以看到完整的解码序列。与始终从左到右运行的 Seq2Seq 相比，它具有前向和后向信息以校正每个单独帧的预测误差。在这项工作中，我们使用 CBHG 模块进行后处理网络。

3.2 WaveRNN

3.2.1 WaveRNN 模型概述

端到端的语音合成模型相较于传统的语音合成算法在音质方面得到了前所未有的提高，没有了传统算法合成语音浓厚的“机器味”，同时还可以根据深度网络设计完备的训练字符集，使机器学会有情感的发声。但是高质量的语音合成需要消耗大量的计算资源，对于实时性要求较高的项目任务，可能还需要在质量和计算资源消耗方面做一个均衡。Google 针对实时高质量的音频合成任务提出了 WaveRNN 网络模型架构。

该模型设计的目标为：增加音频序列模型的采样效率，而不损失模型的性能。

$$T(\mathrm{u}) = |u| \sum_{i=1}^{N} (\mathrm{c}(\mathrm{op_i}) + \mathrm{d}(\mathrm{op_i})) \tag{3.3}$$

采样花费时间由上述公式定义，其中：

$T(\mathrm{u})$ 采样花费时间

u 音频样例单元
N 网络模型深度
$c(\mathrm{op_i})$ 样例单元计算时间
$d(\mathrm{op_i})$ 每层网络所需运行时间

3.2.2 模型参数分析与设计

(a) 降低网络深度减少采样花费时间

RNN 的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能无力。例如，你要预测句子的下一个单词是什么，一般需要用到前面的单词，因为一个句子中前后单词并不是独立的。RNNs 之所以称为循环神经网路，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。理论上，RNNs 能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关。可以看出 RNN 网络层不是当前输入的简单的线性输出，该网络的输出取决于前面几个网络层的状态特征，从而可以很好的处理时序关联的文本，语音序列。WaveRNN 的设计就是借鉴了 RNN 网络这样的特点，即单层循环网络层可以输出对文本的高非线性变换。WaveRNN 由单层 RNN 和双 softmax 层构成，softmax 层用于预测 bit 位音频样本。

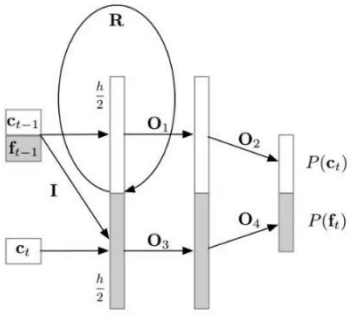


图 3-5 WaveRNN 结构图

c_{t-1} 代表上一时刻状态的粗采样样本数据，f_{t-1} 代表上一时刻的细采样样本数据。每一个部分分别输入对应的 softmax 层，对细样本的预测是基于粗样本的。R 与粗样本和细样本同时相乘，门输出只用于估算粗样本，然后采样得到 c_t。从 $P(\mathrm{c_t})$ 中采样得到 c 后，估算细样本，然后采样得到 f_t。

(b) 权重稀疏化减少采样计算时间

减少网络的参数数量可以减少采样所需要的计算时间，如何在给定参数数量情况下最优化表现性能，WaveRNN 给出了稀疏化权重的策略。在参数固定

时，大型稀疏 WaveRNN 比小型密集 WaveRNN 表现性能要好，而且这一对比关系在稀疏程度高达 96% 时也能保持。

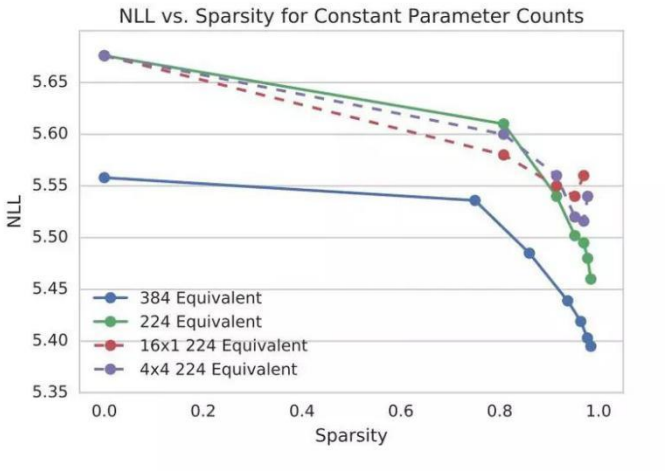


图 3-6 固定参数数量 NLL 与稀疏度变化曲线

资料来源：Efficient Neural Audio Synthesis

(c) 子尺度 WaveRNN 减少单次训练样本数量

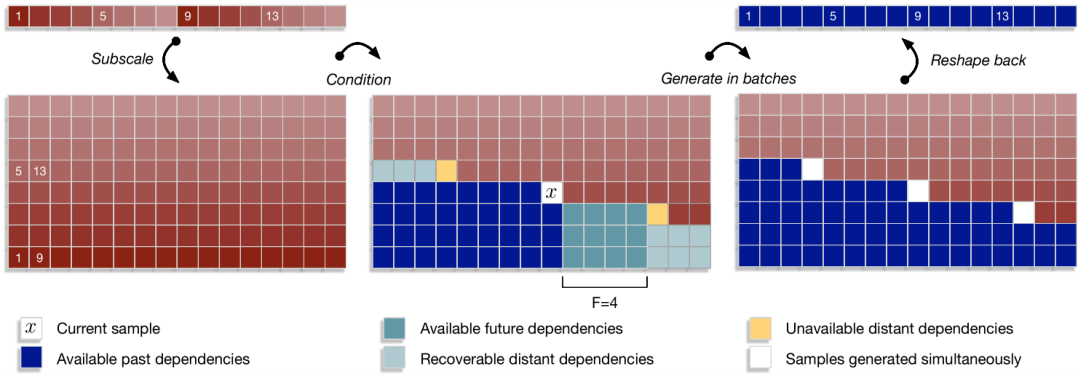


图 3-7 Subscale WaveRNN 依赖机制

资料来源：Efficient Neural Audio Synthesis

图中每一个方块对应一个 bit 位采样样本。子尺度方法首先将张量分割成 B 个交错样本的子张量。每个子张量基于之前生成的子张量的过去和未来样本生成。过去范围不受限制，而未来范围为 F，根据条件网络的接受野而定，然后应用批采样。最后的张量由生成的子张量重组而成。

批采样：

一个尺度为 L 的 张量 折叠成 B 个子张量，每一个尺度为 L/B。子尺度的方法可以在一个 batch 中一次生成多个样本。由于每个子张量的生成基于前一个子张量，在实际中只需要相对较小的未来状态视野（future horizon F）。下一个子张量的生成可以在前一个子张量的前 F 个样本生成后立刻开始。

(d) GRU 网络层解决长序列依赖问题

GRU 即 Gated Recurrent Unit。为了克服 RNN 无法很好处理远距离依赖而提出了 LSTM，而 GRU 则是 LSTM 的一个变体，当然 LSTM 还有有很多其他的变体。GRU 保持了 LSTM 的效果同时又使结构更加简单，所以它非常流行。而 GRU 模型如下，它只有两个门了，分别为更新门和重置门，即图中的 ut 和 rt。更新门用于控制前一时刻的状态信息被带入到当前状态中的程度，更新门的值越大说明前一时刻的状态信息带入越多。重置门用于控制忽略前一时刻的状态信息的程度，重置门的值越小说明忽略得越多。

GRU 模型单元如下图所示：

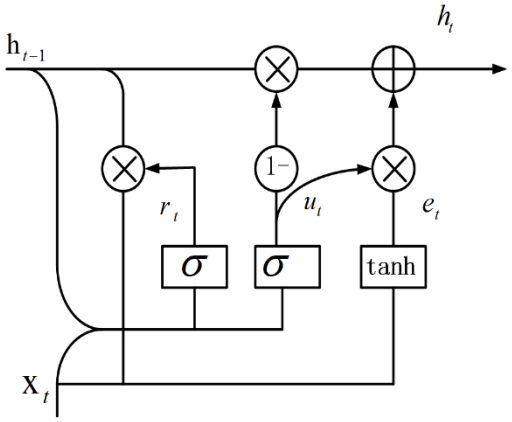


图 3-8 GRU 模型单元

WaveRNN 模型中引入 GRU 网络连接层以更好的解决该网络模型对于长序列依赖问题的解决，可以使得网络在长时间序列即长输入音频采样序列上表现出更好的时间轴上的因素级别的关联。结合 GRU 单元，单层 RNN 网络的数学模型分析如下：

$$Xt = [c_{t-1}, f_{t-1}, c_t] \tag{3.4}$$

$$u_t = \sigma(R_u h_{t-1} + I_u^* x_t) \tag{3.5}$$

$$r_t = \sigma(R_r h_{t-1} + I_r^* x_t) \tag{3.6}$$

$$e_t = \tau(r_t \circ (R_e h_{t-1}) + I_e^* x_t) \tag{3.7}$$

$$h_t = u_t \circ h_{t-1} + (1 - u_t) \circ e_t \tag{3.8}$$

$$P(c_t) = \text{softmax}(O_2 \quad relu(O_1 y_c)) \tag{3.9}$$

$$P(f_t) = \text{softmax}(O_4 \quad relu(O_3 y_f)) \tag{3.10}$$

如图 3-5 所示，当前粗采样数据只送入网络的细采样分析部分，网络的 R 部分即循环部分根据 GRU 单元的更新门和重置门计算响应的向量权重。告知网络单元对此刻输入参数的参量变化，以及对前一时刻网络层次信息的筛选和保留程度，以便更好地对下一时刻的输出做出参数的调整。

3.3 改进的 TACOTRON

TACOTRON 是一个从字符序列生成幅度谱图的 Seq2Seq 架构，它仅用输入数据训练出一个单一的神经网络，用于替换语言学和声学特征的生成模块，从而简化了传统语音合成的流水线。为了最终合成出幅度谱图，TACOTRON 使用 Griffin-Lim 算法估计相位，然后施加一个短时傅里叶逆变换。Griffin-Lim 算法会产生特有的人工痕迹并且合成的语音保真度较低，我们将其替换成 WaveRNN 神经网络声码器。

本文将 TACOTRON 与 WaveRNN 优势相结合组成一个统一的神经网络语音合成模型。基于 Seq2Seq 的 TACOTRON 模型用来生成梅尔声谱图，后接一个 WaveRNN 声码器。该系统允许直接使用字符序列和语音波形数据进行端到端的训练学习语音合成，它合成的语音的自然度接近了真人语音。

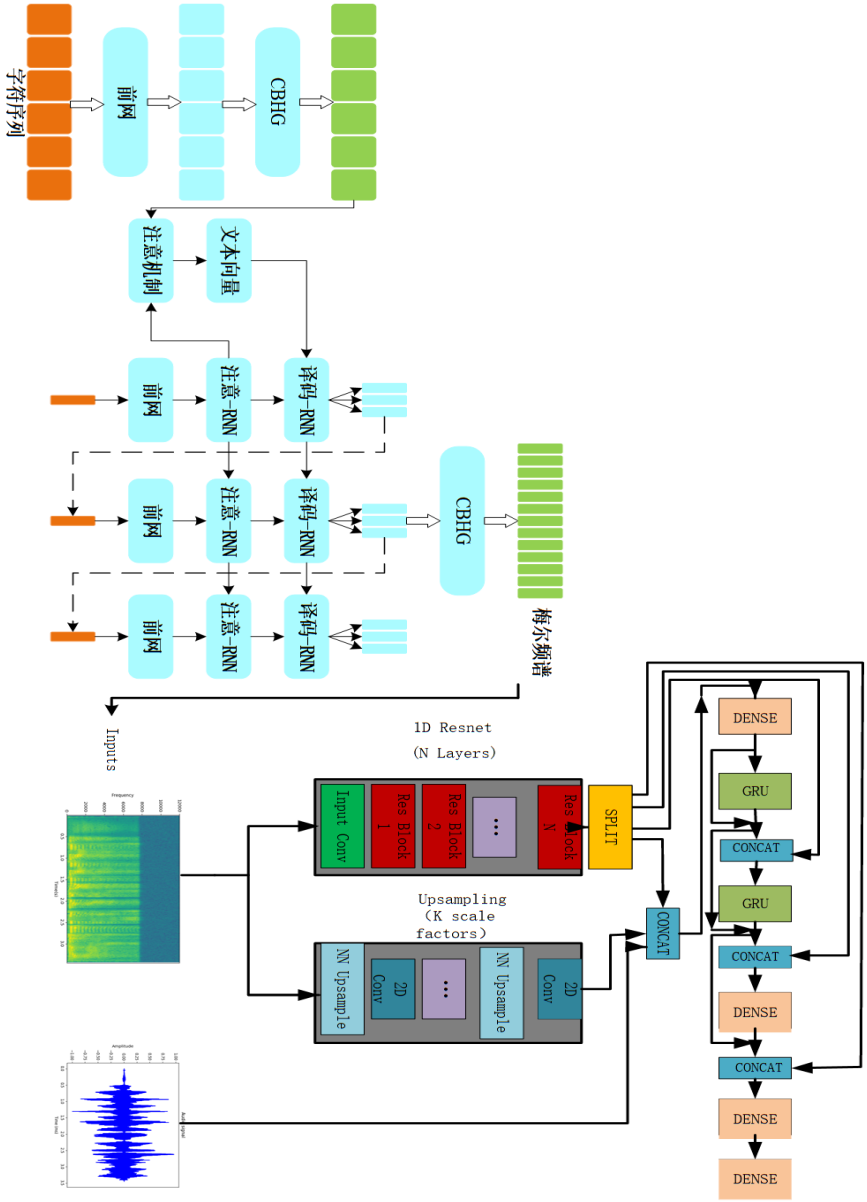


图 3-9 改进的 TACOTRON 结构

第 4 章 软件设计与流程

根据以上原理分析，我们设计了代码实现部分。硬件参数如下：CPU： i7-9750、显卡：GTX1650、内存：8GB。

4.1 前期数据处理

表 4-1 训练音频文件处理参数设置

参数	数值
采样速率	22050
FFT 点数	2048
位深度	9bits
梅尔系数	80

音频预处理方法实现（部分示例）：训练数据的线性频谱和梅尔频谱

```
def spectrogram(y):
    D = stft(y)
    S = amp_to_db(np.abs(D)) - hp.ref_level_db
    return normalize(S)


def melspectrogram(y):
    D = stft(y)
    S = amp_to_db(linear_to_mel(np.abs(D)))
    return normalize(S)
```

4.2 模型设计

4.2.1. TACOTRON 模型参数设计

表 4-2 TACOTRON 模型参数设计

模型参数	数值
词向量嵌入维度	256
编码维度	128
解码维度	256
LSTM 维度	512
Pre-Net 维度	128
Droupout	0.5

代码实现（部分示例）：

TACOTRON 类定义及初始化函数：

```
class Tacotron(nn.Module):
    def __init__(self, embed_dims, num_chars, encoder_dims, decoder_dims, n_mels, fft_bins, postnet_dims,
                 encoder_K, lstm_dims, postnet_K, num_highways, dropout):
        super().__init__()
        self.n_mels = n_mels
        self.lstm_dims = lstm_dims
        self.decoder_dims = decoder_dims
        self.encoder = Encoder(embed_dims, num_chars, encoder_dims,
                               encoder_K, num_highways, dropout)
        self.encoder_proj = nn.Linear(decoder_dims, decoder_dims, bias=False)
        self.decoder = Decoder(n_mels, decoder_dims, lstm_dims)
        self.postnet = CBHG(postnet_K, n_mels, postnet_dims, [256, 80], num_highways)
        self.post_proj = nn.Linear(postnet_dims * 2, fft_bins, bias=False)

        self.init_model()
        self.num_params()

        # Unfortunately I have to put these settings into params in order to save
        # if anyone knows a better way of doing this please open an issue in the repo
        self.step = nn.Parameter(torch.zeros(1).long(), requires_grad=False)
        self.r = nn.Parameter(torch.tensor(0).long(), requires_grad=False)
```

前向传播方法实现：

```
def forward(self, x, m, generate_gta=False):

    self.step += 1

    if generate_gta:
        self.encoder.eval()
        self.postnet.eval()
        self.decoder.generating = True
    else:
        self.encoder.train()
        self.postnet.train()
        self.decoder.generating = False

    batch_size, _, steps = m.size()

    # Initialise all hidden states and pack into tuple
    attn_hidden = torch.zeros(batch_size, self.decoder_dims).cuda()
    rnn1_hidden = torch.zeros(batch_size, self.lstm_dims).cuda()
    rnn2_hidden = torch.zeros(batch_size, self.lstm_dims).cuda()
    hidden_states = (attn_hidden, rnn1_hidden, rnn2_hidden)

    # Initialise all lstm cell states and pack into tuple
    rnn1_cell = torch.zeros(batch_size, self.lstm_dims).cuda()
    rnn2_cell = torch.zeros(batch_size, self.lstm_dims).cuda()
    cell_states = (rnn1_cell, rnn2_cell)

    # <GO> Frame for start of decoder loop
    go_frame = torch.zeros(batch_size, self.n_mels).cuda()

    # Need an initial context vector
    context_vec = torch.zeros(batch_size, self.decoder_dims).cuda()

    # Project the encoder outputs to avoid
    # unnecessary matmuls in the decoder loop
    encoder_seq = self.encoder(x)
```

4.2.2 WaveRNN 模型参数设计

表 4-3 WaveRNN 模型参数设计

模型参数	数值
上采样系数	（5,5,11）
RNN 维度	512
粗/细采样维度	512
计算维度(R)	128

代码实现（部分示例）：

定义 WaveRNN 类，并设计初始化函数：

```
class WaveRNN(nn.Module) :
    def __init__(self, hidden_size=896, quantisation=256) :
        super(WaveRNN, self).__init__()

        self.hidden_size = hidden_size
        self.split_size = hidden_size // 2

        # The main matmul
        self.R = nn.Linear(self.hidden_size, 3 * self.hidden_size, bias=False)

        # Output fc layers
        self.O1 = nn.Linear(self.split_size, self.split_size)
        self.O2 = nn.Linear(self.split_size, quantisation)
        self.O3 = nn.Linear(self.split_size, self.split_size)
        self.O4 = nn.Linear(self.split_size, quantisation)

        # Input fc layers
        self.I_coarse = nn.Linear(2, 3 * self.split_size, bias=False)
        self.I_fine = nn.Linear(3, 3 * self.split_size, bias=False)

        # biases for the gates
        self.bias_u = nn.Parameter(torch.zeros(self.hidden_size))
        self.bias_r = nn.Parameter(torch.zeros(self.hidden_size))
        self.bias_e = nn.Parameter(torch.zeros(self.hidden_size))
```

定义前向传播方法：

```
def forward(self, prev_y, prev_hidden, current_coarse) :

    # Main matmul - the projection is split 3 ways
    R_hidden = self.R(prev_hidden)
    R_u, R_r, R_e, = torch.split(R_hidden, self.hidden_size, dim=1)

    # Project the prev input
    coarse_input_proj = self.I_coarse(prev_y)
    I_coarse_u, I_coarse_r, I_coarse_e = \
        torch.split(coarse_input_proj, self.split_size, dim=1)

    # Project the prev input and current coarse sample
    fine_input = torch.cat(prev_y, current_coarse, dim=1)
    fine_input_proj = self.I_fine(fine_input)
    I_fine_u, I_fine_r, I_fine_e = \
        torch.split(fine_input_proj, self.split_size, dim=1)

    # concatenate for the gates
    I_u = torch.cat(I_coarse_u, I_fine_u, dim=1)
    I_r = torch.cat(I_coarse_r, I_fine_r, dim=1)
    I_e = torch.cat(I_coarse_e, I_fine_e, dim=1)

    # Compute all gates for coarse and fine
    u = F.sigmoid(R_u + I_u + self.bias_u)
    r = F.sigmoid(R_r + I_r + self.bias_r)
    e = F.tanh(r * R_e + I_e + self.bias_e)
    hidden = u * prev_hidden + (1. - u) * e

    # Split the hidden state
    hidden_coarse, hidden_fine = torch.split(hidden, self.split_size, dim=1)

    # Compute outputs
    out_coarse = self.O2(F.relu(self.O1(hidden_coarse)))
    out_fine = self.O4(F.relu(self.O3(hidden_fine)))
```

第 5 章 系统测试与误差分析

英语和汉语前期数据预处理有着些许不同，英文标注主要有 A-Z，a-z 以及数字常用标点符号等组成。而汉字本身有 2-3 万个，穷举的话太多，还有很多同音字，所以我们使用汉语拼音作为字符标注是一种可行方案，其中数字 1-5 分别表示拼音的一声调，二声调，三声调，四声调以及轻声。所谓的端到端<文本，声谱>配对，其实就是要让机器学会将每一个包括空格和标点在内的字符集对应到（梅尔频谱或线性频谱）的某几帧。

5.1 模型训练数据

表 5-1 模型训练数据

数据集	样本数量
LJSpeech	10177
Thchs30	13388

5.2 LJSpeech 模型分析

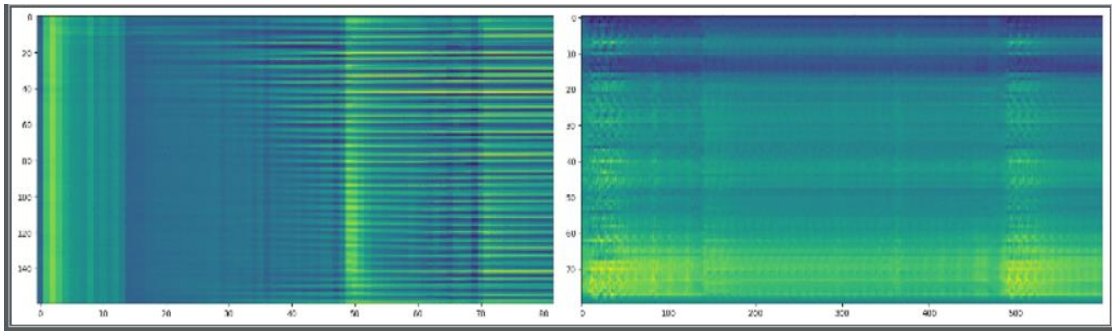


图 5-1 模型训练初期 Alignment 图及梅尔频谱图

左图为训练初期模型的 Alignment 图示，该图线性程度越高以及对应单元颜色亮度越大说明模型训练效果越好。右图为梅尔频谱的显示情况，此刻该模型还没有学会很好分类各音素单元，所以图示频谱混乱没有规则。

我们可以通过 Aligment 图示以及 Loss 指标对模型进行实时的训练监督。

5.2.1 TACOTRON 模型

最后的训练结果图示如下

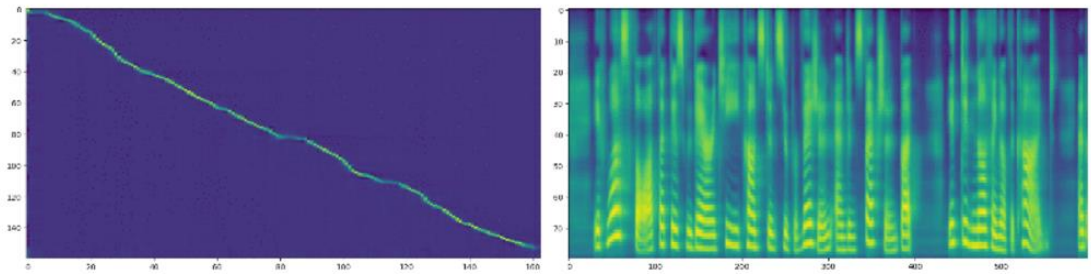


图 5-2 训练结果图

可以看出 Aligment 图示线性程度良好，且音素单元亮度明晰，说明合成效果较好，没有嘈杂的背景噪声。

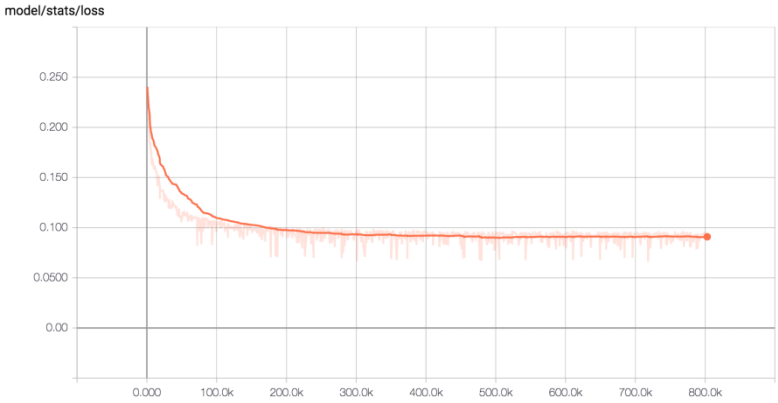


图 5-3 网络训练 loss 图

模型在 20k steps 下可以出现能分辨的语言合成，250k 之后 loss 基本不变，模型训练完成。

示例：输入 “To be or not to be this is a question”

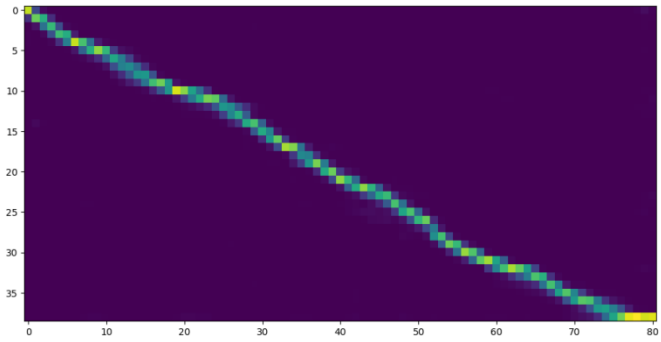


图 5-4 示例训练结果图

语音合成模型参数总量：11.088M。

5.2.2 TACOTRON+WaveRNN 模型

该模型与 TACOTRON 模型最大的区别在于最后语音合成阶段。根据 WaveRNN 模型的原理分析，我们设计了代码实现部分可以自定义设计实现子尺度下 Batch（批采样）精度。可以通过设置 unbatched 参数实现批采样，集保留最高品质的音效，但是相应的要增加合成语音的时长。相反可以根据不同的—batched 参数设计不同数值大小，在合成效率和合成质量上做合理地取舍。

```
parser = argparse.ArgumentParser(description='TTS Generator')
parser.add_argument('--input_text', '-i', type=str, help='[string] Type in something here and TTS will generate it!')
parser.add_argument('--batched', '-b', dest='batched', action='store_true', help='Fast Batched Generation (lower quality)')
parser.add_argument('--unbatched', '-u', dest='batched', action='store_false', help='Slower Unbatched Generation (better quality)')
```

示例：输入“To be or not to be this is a question” unbatched（batch size = 1）

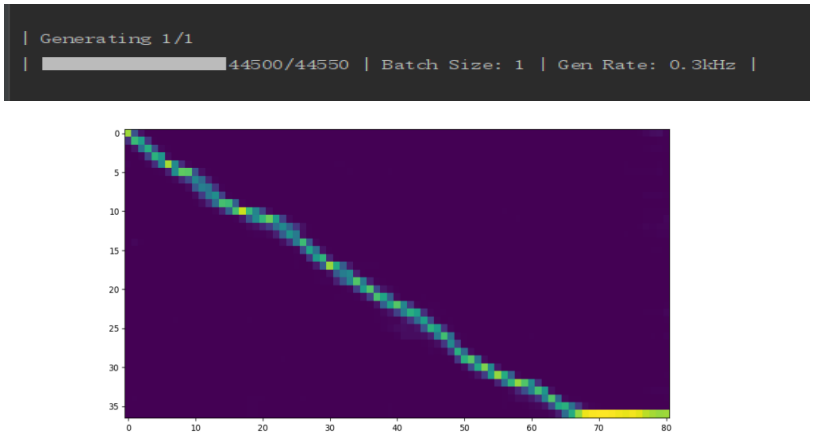
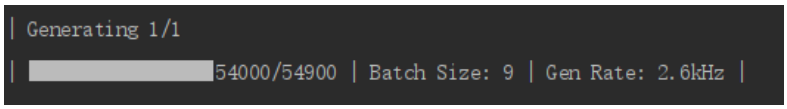


图 5-5 示例设置 unbatched (batch size=1)参数训练结果图

输入“To be or not to be this is a question” batched 9（batch size = 9）



Gen Rate 参数可以清晰地指示合成需要的时间缩短为 unbatched 的 2.6/0.3 = 8.7 倍,大小约等于 batch size 的倍数。同时从 Aligment 图示可以看出，合成效果基本没有差别。

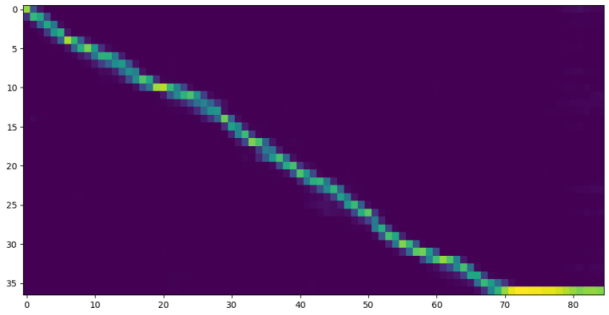


图 5-6 示例设置 unbatched (batch size=9)参数训练结果图

语音合成模型参数总量：4.234M

5.3 Thchs30 汉语言合成

汉语言合成相比于英文语音合成前期数据处理阶段需要对输入汉字进行拼音划分和音调标注，具体实现如下代码：

```
import pypinyin
import re
def to_pinyin(word):
    s = ''
    for i in pypinyin.pinyin(word, style=pypinyin.STYLE_TONE3):
        s = s+''.join(i)
    s1 = (re.findall(r"\D+\d",s))      #正则表达式 分割字符串
    s2 = [' '.join(s1)]

    return s2
if __name__ == "__main__":
  print(to_pinyin("南京航空航天大学电子信息工程学院"))
```

表 5-2 合成语音参数分析

参数	数值
采样速率	24000
通道数量	1
采样宽度	2
帧速率	24000
总帧数	84424
时长（s）	3.518

下图为训练 1000step 时候的 Alignment 图示。

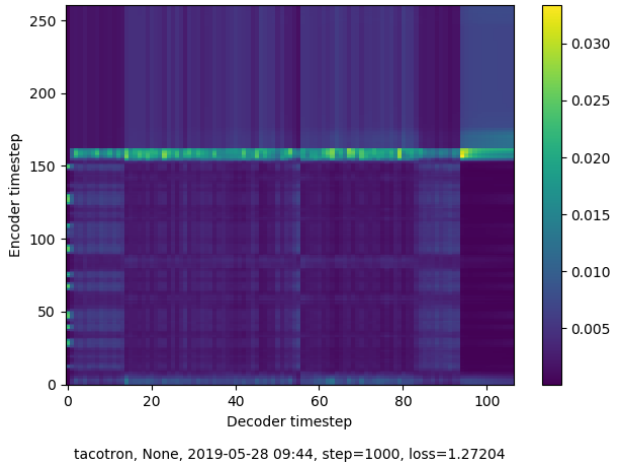


图 5-7 Alignment 图（训练 1000step）

下图为训练 92000step 时候的 Alignment 图示。

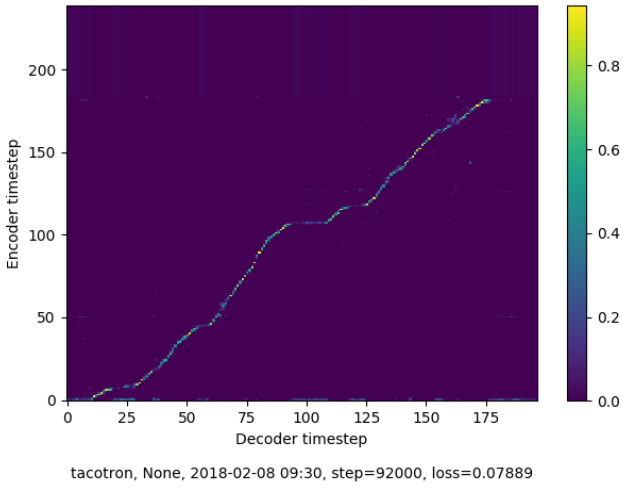


图 5-8 Alignment 图（训练 92000step）

可以看出此时模型线性程度较好，说明模型训练有效。

示例：输入 “南京航空航天大学电子信息工程学院”

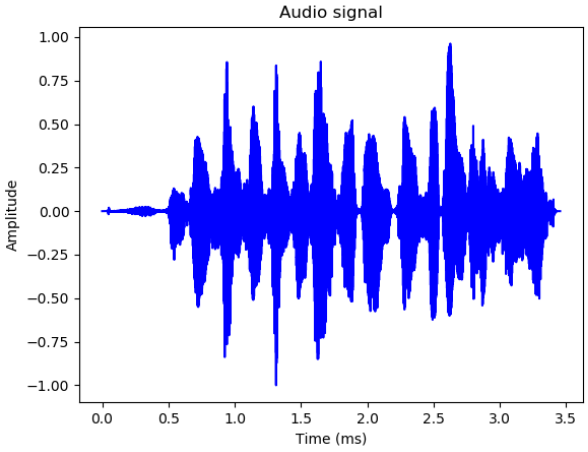


图 5-9 合成时域波形图

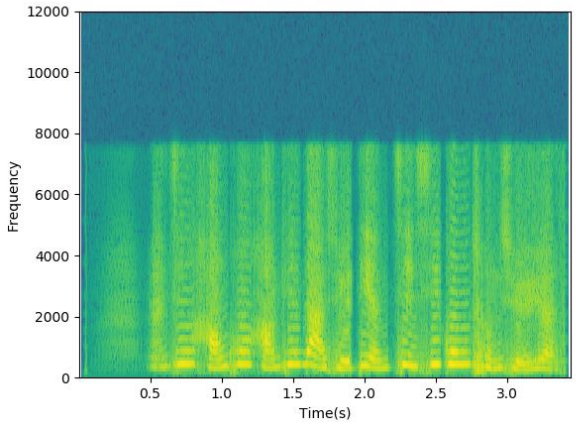


图 5-10 合成语音频谱图

第 6 章 客户端设计

6.1 安卓端 APP 设计

开发工具：Android Studio

开发语言：Java

测试平台：Android9.0

软件设计实现手机客户端可以和同在一局域网下的 Python 服务器端进行信息交互和指令控制，客户端和服务端程序执行流程如下：

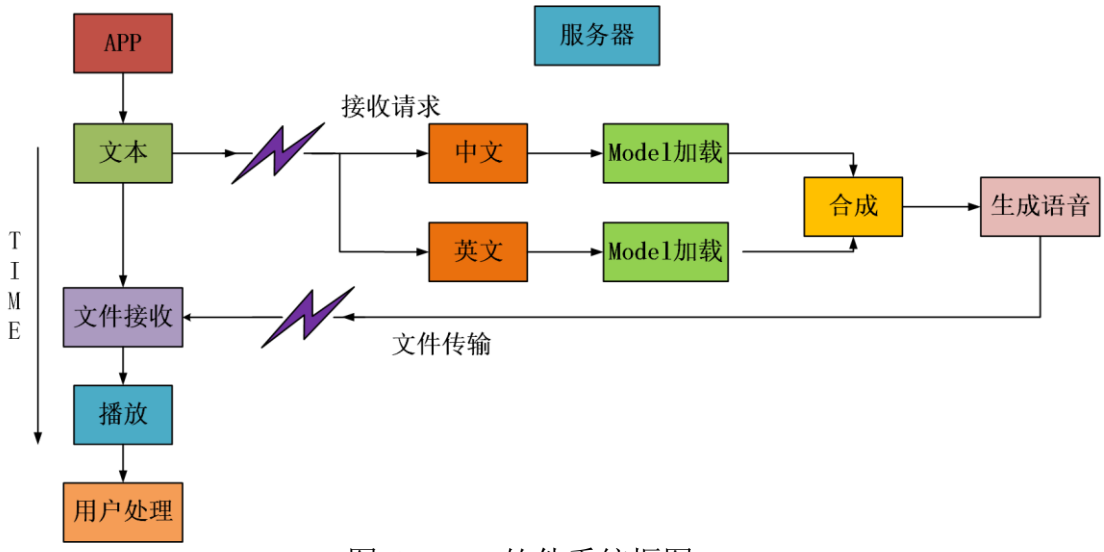


图 6-1 APP 软件系统框图

主要界面展示：

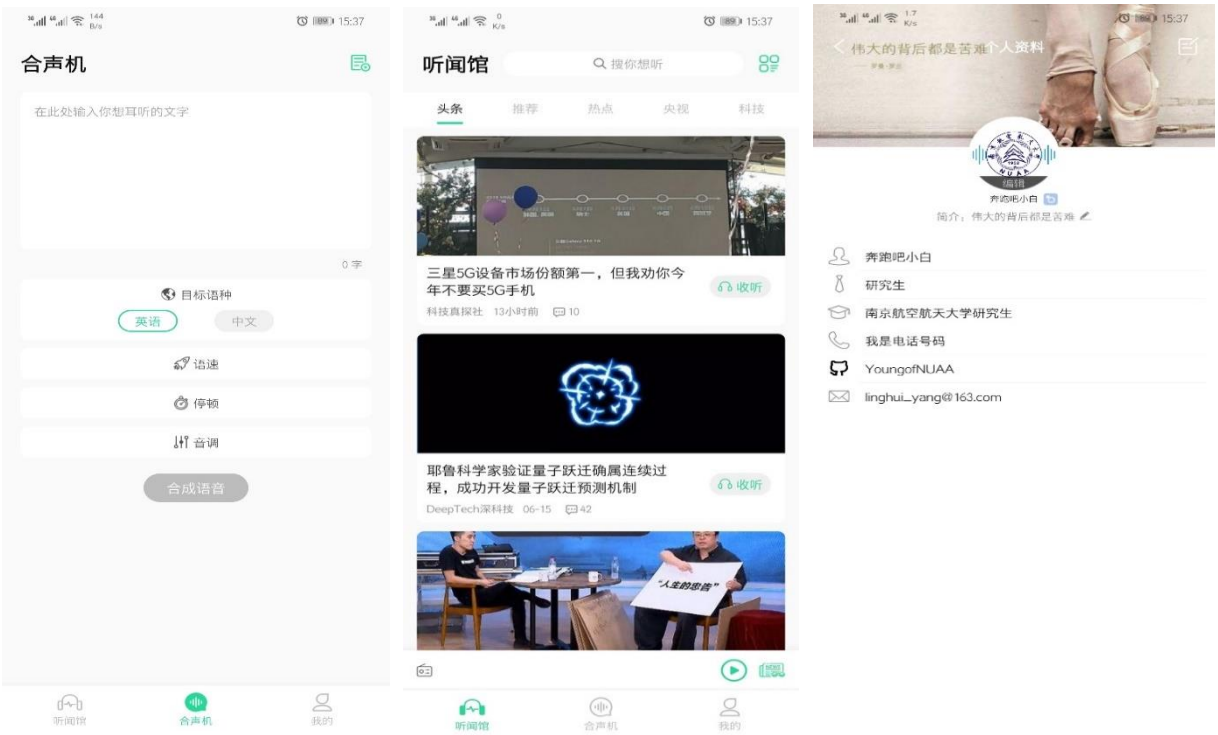


图 6-2 APP 主界面框架设计

主要功能介绍：

软件名称：ITalk

软件主要功能分为三个模块：

听闻馆： 可以实时获取网络新闻头条，进行展示，点击右边收听按钮可以实现新闻标题的语音合成。

合声机： 实现基本的中文英文语音合成，设置简单的语音文件控制按钮可以实现播放语速，停顿和音调的调节。

我的： 该模块主要展示个人信息和 Logo。“伟大的背后都是苦难”。

6.2 电脑端语音合成界面设计

开发工具：Pycharm
开发语言：Python
编译器： python3.7
测试平台：win10
语音合成界面展示：

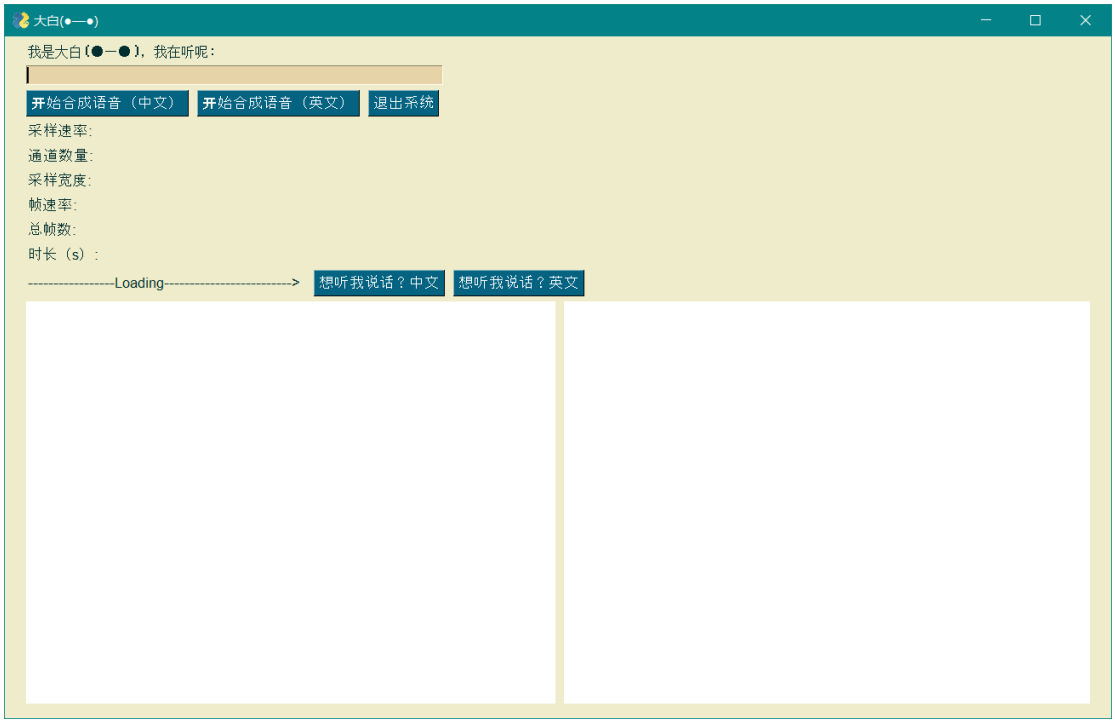


图 6-3 电脑端语音合成用户界面

功能模块描述：

软件名称：大白(●—●)

输入文本框：可以接受输入中文或者英文文本。

中/英合成按钮：根据输入选择正确的合成按钮。

语音参数显示模块：显示合成语音文件的采样速率，通道数量等参数。

语音播放按钮：实现语音的播放功能。

时域/语谱图展示框图：显示语音信号的时域波形图和对应的语谱图。

运行效果展示：

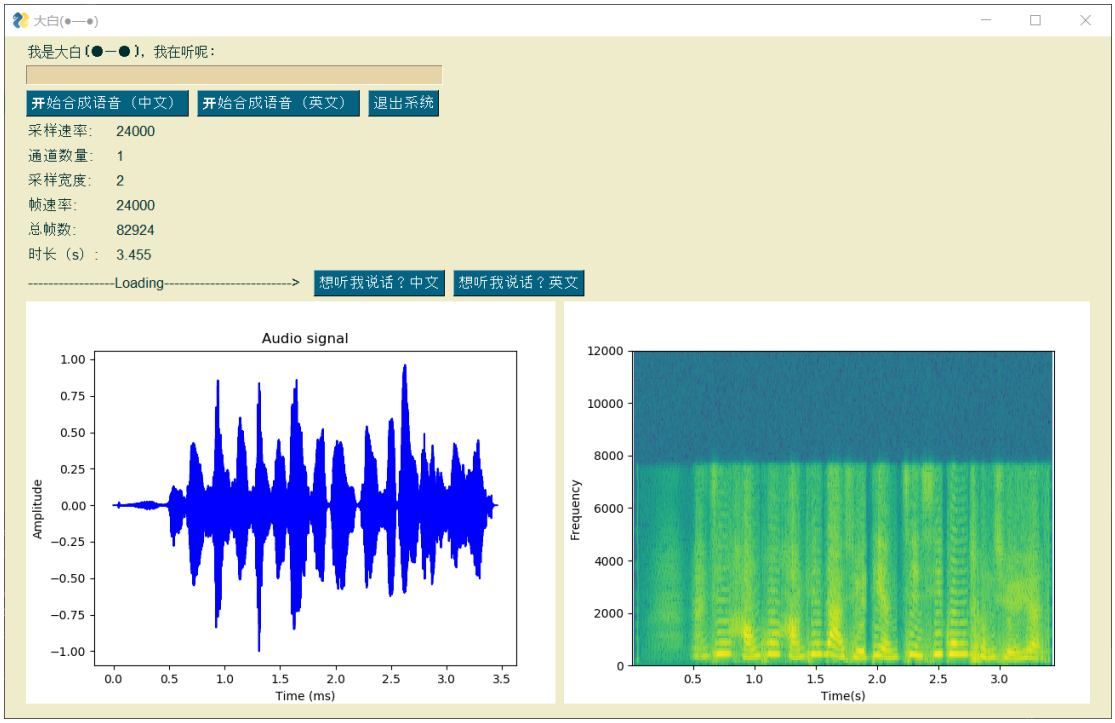


图 6-4 界面运行各模块结果展示

第 7 章 总结

本文详细描述了改进的 TACOTRON 模型设计，它通过两个组件的结合形成了完整的端到端语音合成系统。包含注意机制的 Seq2Seq 循环网络用于预测梅尔声谱图，WaveRNN 用于实时高质量的音频合成。深度学习模型的特点很好的诠释了 TTS，即文本到语音的合成流程，没有复杂的中间处理过程，而且完全的端到端合成模型，使得对各种语言的合成变得简单明晰。只需要送给模型目标语言的符号集以及对应的发音文件，模型就可以作为质量较佳的合成系统，开发者可以将注意力放在模型参数的优化和改良上来。深度学习给各行各业带来了繁荣与解决问题的新思路，语音合成也是人机交互的最基本需求，通过神经网络让语音合成日新月异，机器不但能听会说，还能分析，会思考，这是传统算法模型无法比拟的优点。我们坚信在神经网络的催化下，语音合成正在迈向新的里程碑，也准备好了迎接万物互联，万物智能时代的到来。

参考文献

[1] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, et al. Tacotron: Towards end-to-end speech synthesis. In Proceedings of Interspeech, August 2017a. URL https://arxiv.org/abs/1703.10135.

[2] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, et al. Efficient Neural Audio Synthesis. The 35th International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[3] Jonathan Shen, Ruoming Pang, Ron J. Weiss, et al. Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions. arXiv preprint arXiv:1712.05884, dec 2017. URL https://arxiv.org/abs/1712.05884.