语音合成实验报告

王松宸 2024201594

2025 年 12 月 21 日

1 实验目的

本次实验旨在使用现成的 FastSpeech 2 模型，理解非自回归语音合成模型的原理。通过配置环境、下载预训练模型，最终实现英文和中文的文本到语音（TTS）合成。

2 模型理解

在阅读代码后，我对 FastSpeech 2 有了以下几点直观的理解：

2.1 为什么叫"Fast"？

传统的 TTS 模型（如 Tacotron 2）是“自回归”的，也就是说，它生成第 2 个时刻的声音时，必须依赖第 1 个时刻的结果。

而 FastSpeech 2 是“非自回归”的（Parallel），它像是一个统筹全局的指挥官，一次性就能预测出整句话的所有频谱帧。这使得它的推理速度极快，非常适合实时应用。

2.2 核心组件：Variance Adaptor

查阅资料后我发现，FastSpeech 2 相比一代最大的改进在于引入了 **Variance Adaptor（变量适配器）**。它不再只是简单地把字变成声音，而是试图去预测声音的“抑扬顿挫”。模型中包含了三个关键的预测器：

- **Duration Predictor（时长预测）**：决定每个音素（比如一个拼音）读多长时间。
- **Pitch Predictor（音高预测）**：决定声音的高低起伏。
- **Energy Predictor（能量预测）**：决定声音的响度。

正是这些组件，让合成出来的声音不再是冷冰冰的机器音，而是有了情感和节奏。

3 实验过程

本次实验最大的挑战不在于运行代码，而在于配置好所有环境。

3.1 环境搭建

由于项目依赖较旧（PyTorch 1.7），直接在 Base 环境安装会导致各种冲突。我采取了以下策略：

- **使用 Miniconda 管理环境**：在 WSL 中安装 Miniconda，创建了一个基于 Python 3.8 的独立环境 `fs2`。

- **解决依赖冲突：**在安装 `pyworld` 和 `numpy` 时遇到了严重的编译错误。最终发现必须使用 Conda 安装二进制包来规避源码编译的兼容性问题。
- **解决库冲突（Double Free）：**在合成时遇到了 Linux 下经典的 `free(): double free detected` 错误。我最终通过卸载 pip 版 torch，转而安装官方 CPU 版 torch 解决了这个问题。

3.2 模型准备

- **声码器：**解压了项目自带的 HiFi-GAN 模型（`generator_LJSpeech` 和 `generator_universal`）。
- **声学模型：**下载了 LJSpeech（英文）和 AISHELL3（中文）的预训练 Checkpoint，并按要求重命名为 `restore_step.pth.tar` 格式，放置在 `output/ckpt/` 目录下。

3.3 代码调试

- **修复音频截断问题：**初次合成时,发现音频的首尾部分有明显的吞字现象。我在在音素序列（Phoneme Sequence）的前后添加了 `{sp}`（静音标记），成功解决了截断问题。
- **修复文件名过长导致无法创建文件的问题：**当输入长文本时，Linux 无法创建过长的文件名。我修改了保存逻辑，自动截取文件名的前 15 个字符。

4 实验结果

实验成功生成了多条清晰的 `.wav` 音频文件。

- **英文效果：** LJSpeech 模型生成的英文发音标准，连读自然，HiFi-GAN 声码器还原的音质非常高，几乎听不出底噪。
- **中文效果：** AISHELL3 模型支持多说话人。通过调整 `speaker_id`，我可以生成不同音色的声音。虽然部分长句的韵律感稍显机械，但字正腔圆，清晰度极佳。

5 个人心得

这次实验让我深刻体会到了“配置环境是深度学习的第一道门槛”。

1. **环境隔离：**为了避免破坏我的 base 环境，我专门为此学习了 Miniconda 的使用。通过创建独立环境，可以自由安装各种版本的库，而不担心冲突。

2. **对 End-to-End 的思考：** FastSpeech 2 虽然叫 End-to-End，但其实还是分成了“声学模型”和“声码器”两步。声学模型预测 Mel 频谱，声码器把频谱变回波形。这种解耦的设计让模型训练更稳定，也方便替换不同的声码器。

3. **实践出真知：**只有亲自跑代码，才会遇到各种各样的问题。解决问题的过程正是提高我科研能力的过程。