CS230

Correcting Lecture Audios Using Speech Synthesis

Rongxin Liu
Stanford University
rongxinl@stanford.edu

Abstract

When an online course is published, some mistakes can be found in the recorded audio sentences and affect the learner's experience. Re-recording could be time-consuming and not cost-effective, so a tool that could generate speech using arbitrary text while preserving the target speaker's vocal style is preferable. This project demonstrates the possibility of using Tacotron2 to perform speech synthesis tasks and experiment with extending the model to generate speech from unseen speakers.

1 Introduction

Often, published online courses or recorded lectures may contain a few errors in the speech, potentially affecting learners' experience or unnecessarily confuse them. Current mitigation provides foot-notes to inform the learners, but this breaks the learning experience because it is difficult for learners to override what they have just listened to. Re-recording the lecture is not cost-effective, and replacing the sentences with new recordings breaks the continuity because of the tonal changes in the new audio recordings.

In this project, we focus on implementing Tacotron2, a text-to-speech component of a neural network-based system, which will ultimately be the tool that can replace the original audio containing errors with the new audio while preserving the speaker's vocal style.

2 Related work

The Tacotron2 model, as the name suggests, is based on its predecessor - Tacotron, in which the main differences are in vocoders. Tacotron uses the Griffin-Lim algorithm for phase estimation, followed by an inverse short-time Fourier transform, which is essentially a placeholder for future neural vocoder approaches because Griffin-Lim produces characteristic artifacts and lower audio quality than approaches like WaveNet.

3 Methods

3.1 Overview

In a nutshell, the neural network-based TTS system is consist of three main components: 1) an encoder that generates a fixed-dimensional embedding vector from reference audio (e.g., the target speaker); 2) a text-to-speech synthesis network that generates a mel-spectrogram from the text, conditioned on the speaker embedding. We will be using the Tacotron2 model for our project; 3) a vocoder using WaveNet to convert the mel-spectrogram to a waveform.

CS230: Deep Learning, Winter 2020, Stanford University, CA.

3.1.1 Models

Encoder The speaker encoder is used to condition the synthesis network on a reference speech signal from the desired target speaker. A good generalization is the use of a representation that captures the characteristics of different speakers. The paper, Generalized End-to-End Loss for Speaker Verification (Li, et al), proposed a highly scalable and accurate neural network framework for speaker verification that the embedding generated from a speaker discrimination task is suitable for conditioning the synthesis network.

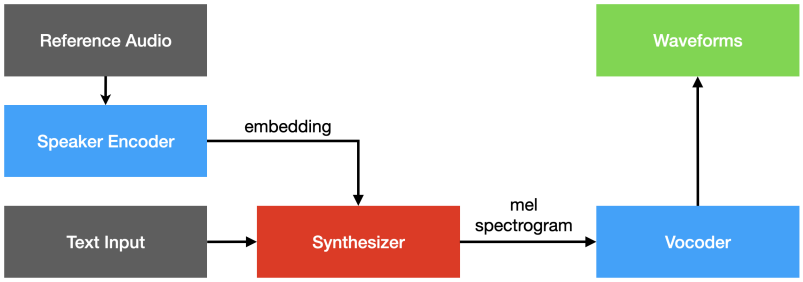
Synthesizer Tacotron2 model forms a text-to-speech system that enables the user to synthesize a natural-sounding speech from raw transcripts without any additional prosody information. The Tacotron 2 model produces mel-spectrograms from input text using encoder-decoder architecture.

Vocoder WaveNet is essentially a deep convolutional neural network (CNN) that takes a raw signal as an input and synthesizes an output one sample at a time. It does so by sampling from a softmax (i.e., categorical) distribution of a signal value encoded using -law companding transformation and quantized to 256 possible values.

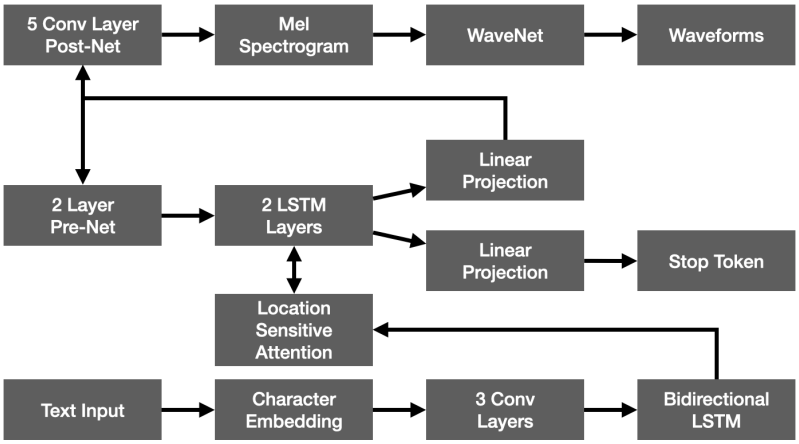
Given the available computing resource and the scope of this project and, we focus only on the implementation details of the Tacotron2 model, provided that it is the core of this multi-speaker TTS system. For the encoder and vocoder, we will be using pre-train models.

3.1.2 Model Architectures

At a high-level, the neural network-based TTS system can be summarized by the following diagram:



And in our project specifically, we're interested in Tacotron2's architecture.



The network is composed of an encoder and a decoder with attention. The encoder converts a character sequence into a hidden feature representation which the decoder consumes to predict a spectrogram. Input characters are represented using a learned 512-dimensional character embedding, which is passed through a stack of 3 convolutional layers, each containing 512 filters with shape 5×1, i.e., where each filter spans 5 characters, followed by batch normalization and ReLU activations.

The final convolutional layer's output is passed into a single bi-directional LSTM layer containing 512 units (256 in each direction) to generate the encoded features.

The encoder output is consumed by an attention network which summarizes the full encoded sequence as a fixed-length context vector for each decoder output step. Attention probabilities are computed after projecting inputs and location features to 128-dimensional hidden representations. Location features are computed using 32 1-D convolution filters of length 31.

The decoder is an autoregressive recurrent neural network that predicts a Mel-frequency spectrogram from the encoded input sequence one frame at a time. The previous time step's prediction is first passed through a small pre-net containing 2 fully connected layers of 256 hidden ReLU units. The pre- net output and attention context vector are concatenated and passed through a stack of 2 uni-directional LSTM layers with 1024 units. The LSTM output concatenation and the attention context vector are projected through a linear transform to predict the target spectrogram frame. Finally, the predicted Mel-frequency spectrogram is passed through a 5-layer convolutional post-net which predicts a residual to add to the prediction to improve the overall reconstruction. Each post-net layer comprises 512 filters with shape 5×1 with batch normalization, followed by Tanh activations on all but the final layer.

4 Dataset and Features

4.1 Dataset

We will be using the LJ Speech dataset to train the Tacotron2 model, and it contains pairs of text transcripts and target audio. LJ Speech is a public domain speech dataset consisting of 13,100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription is provided for each clip. Clips vary in length from 1 to 10 seconds and have a total length of approximately 24 hours. The texts were published between 1884 and 1964 and are in the public domain. The audio was recorded in 2016-17 by the LibriVox project and is also in the public domain.

The dataset can be downloaded here: https://keithito.com/LJ-Speech-Dataset/

We use 12500 entries of data for training, 100 for validation, and 500 for testing.

Dataset samples:

LJ050-0261.wav LJ050-0261|from many Government agencies including the Department of Defense and the President's Office of Science and Technology.

LJ049-0056.wav LJ049-0056|The first duty of the agents in the motorcade is to attempt to cover the President as closely as possible and practicable

4.2 Features

A low-level acoustic representation: mel-frequency spectrograms are used in the model. A Mel-frequency spectrogram is related to the linear-frequency spectrogram, i.e., the magnitude of short-time Fourier transform (STFT). It is obtained by applying a nonlinear transform to the frequency axis of the STFT and summarizes the frequency content with fewer dimensions.

Using such an auditory frequency scale has the effect of emphasizing details in lower frequencies, which are critical to speech intelligibility while de-emphasizing high-frequency details, which are dominated by fricatives and other noise bursts and generally do not need to be modeled with high fidelity.

5 Experiments

5.1 Setup

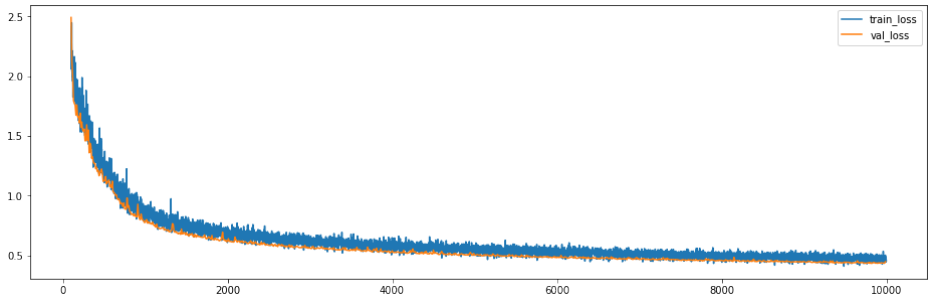
We use the Nvidia Deep Learning container to train our model on a single TITAN RTX GPU. Using the docker container solution can resolve many environment configuration issues and can utilize CUDA correctly for our model training.

The training process involves first training the feature prediction network on its own. We apply the standard maximum-likelihood training procedure to training the feature prediction network by feeding in the correct output instead of the predicted output on the decoder side (a.k.a., teacher-forcing). We use the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$ and a learning rate of $\alpha = 10^{-3}$ exponentially decaying to 10^{-5} starting after 10,000 iterations. We also apply the L_2 regularization with a weight of 10^{-6}.

For every 10 iterations, we will save the spectrogram, gates outputs, and alignment figures to generate animations to visualize the training process.

5.2 Training/Testing

Here we have the model training performance on the first 10000 iterations (excluding the first 50 iterations):

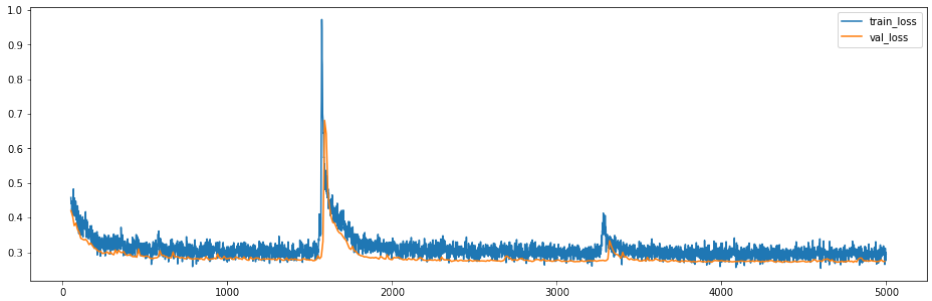


This is a good sign. Clearly, the network is learning and confirms that there is nothing awry with our model configuration. We also attach three generated audio which is corresponding to the model being trained for 1000, 5000, and 10000 iterations. The first generated audio is not very audible that the speaker sounds like mumbling somethings. However, in the second audio, we can start hearing some structural sentences there. For example, we can hear the "The..." at the beginning. For the 3rd audio, albeit still not audible, we can recognize a clear sentence structure (with a seemingly correct pause between words) when listening for the first few seconds.

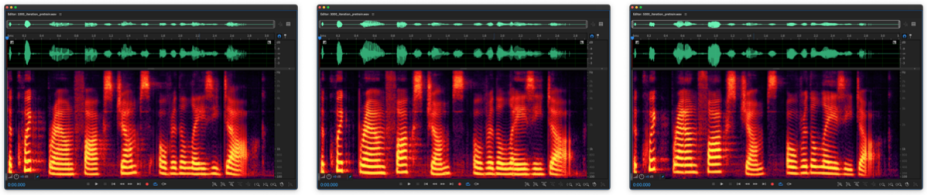
We can also confirm the observation by inspecting the three mel-spectrograms respectively:

The generated audios start to form sentence structures.

For faster convergence on the model, we can also use a published pre-trained model from Nvidia. And here is the result for training 5000 iterations on the pre-trained model:



The model indeed converges at about 1000 iterations, and we also attach 3 audio generated by the model trained on the pre-trained model for 1000, 3000, and 5000 iterations.



The spectrograms suggest a clear sentence structures.

5.3 Further Experiments

By leveraging the knowledge learned by the discriminative speaker encoder, the synthesizer can also generate speech for an arbitrary reference speaker. To synthesize First, we feed the speaker encoder reference audio to generate a speaker embedding to use it in our final speech synthesis. Then, we provide some texts and the speaker embedding to the Tacotron model to have it generate mel-spectrograms. We then feed the mel-spectrograms to the WaveNet model and have it generate the target waveform.

We extracted an audio clip from the Neural Networks and Deep Learning lecture as reference audio (reference_01.wav) and have the TTS system generate the speech using the same content:

The term, Deep Learning, refers to training Neural Networks, sometimes very large Neural Networks. So what exactly is a Neural Network?

Subjectively, the generated audio does resemble the reference speaker's vocal style and proves that the model can generate speech using unseen data. We can also pick one of the audio clips from training data and use it as a reference audio (reference_02.wav) and see how the generated speech sounds like. The result resembles vocal style closer to the speaker, which is expected since we're essentially using training data as test data.

6 Conclusion

In this project, we've covered the implementation details of Tacotron2 and describe a fully neural TTS system that combines a sequence-to-sequence recurrent network with attention to predict mel-spectrogram with a WaveNet vocoder. Furthermore, by extending the Tacotron2 model, a neural network-based TTS system can generate seemingly realistic speech from an unseen speaker, implying that the model is capable of learning to utilize a realistic representation of the space of speaker variation.

7 Acknowledgement

The author would like to thank the CS230 teaching team for delivering this rewarding course during a trying time. The author would also like to thank Jo Chuang for providing suggestions regarding the model evaluation process and assistance on the project in general.

References

[1] Ye Jia et al. Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis *arXiv eprint arXiv:1806.04558*, 2019.

[2] Naihan Li et al. Neural Speech Synthesis with Transformer Network *arXiv eprint arXiv:1809.08895*, 2019.

[3] Aaron van den Oord et al. WaveNet: A Generative Model for Raw Audio *arXiv eprint arXiv:1609.03499*, 2016.

[4] Jonathan Shen et al. Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions *arXiv eprint arXiv:1712.05884*, 2018.

[5] Yuxuan Wang et al. Tacotron: Towards End-to-End Speech Synthesis *arXiv eprint arXiv:1703.10135*, 2017.

CS230 Final Project - Codes

This repository contains the required codes to realize the functionalities described in the final report.

What's inside?

1. A docker-compose file uses a customized docker image, allowing the user to bring up the container that satisfies the runtime environment for this project.
2. Implementations of Tacotron2.
3. Neural network-based system that uses tacotron2 as TTS synthesizer and generates speech that mimics the target speaker's vocal style.

Setup

To resolve the hassles of setting up the environment, the author proposes the Nvidia-Docker solution.

1. Install docker: https://docs.docker.com/get-docker/

   The docker image used in this repository is built on the Nvidia NGC container image and added frequently used software packages. I also added a ngrok container to docker-compose, so it allows users to visit their machine remotely.

2. Install Nvidia Container Toolkit: You will also need to install Nvidia Container Toolkit to run these containers using multiple GPUs.

   https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#linux-distributions

To bring up the containers, run:

```
docker-compose up
```

This will start a docker container with all the required packages installed, and a jupyter notebook server will be running on the assigned address.

For example, when the container is running, you should see the following outputs in the terminal:

```
Starting tensorflow ... done
Starting ngrok      ... done
Attaching to tensorflow, ngrok
ngrok                  | + exec ngrok http nvidia-gpu:8888
tensorflow             |
tensorflow             | ================
tensorflow             | == TensorFlow ==
tensorflow             | ================
tensorflow             |
tensorflow             | NVIDIA Release 20.12-tf1 (build 18410160)
tensorflow             | TensorFlow Version 1.15.4
tensorflow             |
tensorflow             | Container image Copyright (c) 2020, NVIDIA CORPORATION.
```

```
...
tensorflow               | [C 2021-03-17 20:29:13.517 ServerApp]
tensorflow               |
tensorflow               |     To access the server, open this file in a browser:
tensorflow               |         file:///root/.local/share/jupyter/runtime/jpserver-
1-open.html
tensorflow               |     Or copy and paste one of these URLs:
tensorflow               |         http://hostname:8888/lab?
token=a929f2f2a6623ddcae11ccadd5bd3b957564c797f69ff553
tensorflow               |      or http://127.0.0.1:8888/lab?
token=a929f2f2a6623ddcae11ccadd5bd3b957564c797f69ff553
```

User should be able to visit the jupyter notebook using the address:

```
http://127.0.0.1:8888/lab?token=a929f2f2a6623ddcae11ccadd5bd3b957564c797f69ff553
```

If the user want to visit the machine in the same network, you can replace the host with the locally assigned IP address:

```
http://10.0.0.1:8888/lab?token=a929f2f2a6623ddcae11ccadd5bd3b957564c797f69ff553
```

If using ngrok, you can visit the machine via ngrok subdomains, for example:

```
https://titan-rtx-dl.ngrok.io/lab?
token=a929f2f2a6623ddcae11ccadd5bd3b957564c797f69ff553
```

Download Models and Dataset

Once the environment is up-and-running, the user can go ahead and download all the required models and datasets.

Inside the workspace folder, run:

```
./downloads/download_all.sh
```

Usage

To train the Tacotron2 model, inside `tacotron2` folder, run:

```
python3 workspace/tacotron2/train.py --output_directory=outdir --log_directory=logdir
```

If using a pretrain model for faster convergence, run:

```
python3 workspace/tacotron2/train.py --output_directory=outdir --log_directory=logdir
-c saved_models/tacotron2_statedict.pt --warm_start
```

To generate audio from text while preserving target speaker vocal style, run:

```
python3 workspace/nn_system/run.py
```

Example usage:

```
root@056d6d408f41:/mnt/nn_system# python3 run.py
Initializing...
Loaded encoder "pretrained.pt" trained to step 1564501
Synthesizer using device: cuda
Building Wave-RNN
Trainable Parameters: 4.481M
Loading model weights at vocoder/saved_models/pretrained.pt
Target speaker: provide an audio filepath of a vocal style to be synthesized
reference_01.wav
Loaded file succesfully
Created the embedding
Write a sentence (+-20 words) to be synthesized:
The quick brown fox jumps over the lazy dog.
Trainable Parameters: 30.870M
Loaded synthesizer "pretrained.pt" trained to step 295000
+----------+---+
| Tacotron | r |
+----------+---+
|   295k   | 2 |
+----------+---+


| Generating 1/1


Done.

Created the mel spectrogram
Synthesizing the waveform:
{| ████████████████ 47500/48000 | Batch Size: 5 | Gen Rate: 7.0kHz | }
Caught exception: PortAudioError('Error querying device -1')
Continuing without audio playback. Suppress this message with the "--no_sound" flag.

float64

Saved output as demo_output_00.wav
```