Radar-based speech synthesis

Christoph Wagner
christoph.wagner@tu-dresden.de

22. September 2022

This document contains notes on the radar-based speech synthesis.

1 Corpus

The current corpus (version 5) is constructed from a 1000 sentence list taken from the carina corpus. Currently for a single speaker (me). The content is approximately 2 h of parallel speech and radar frames.

1.1 Corpus generation

The corpus was generated to approximate an equal phoneme distribution (script under: `C:/Programming/MATLAB/speech_synthesis/create_phonetically_balanced_sentence_list.mat`)

Phoneme distribution (taken from the `Setup` folder).

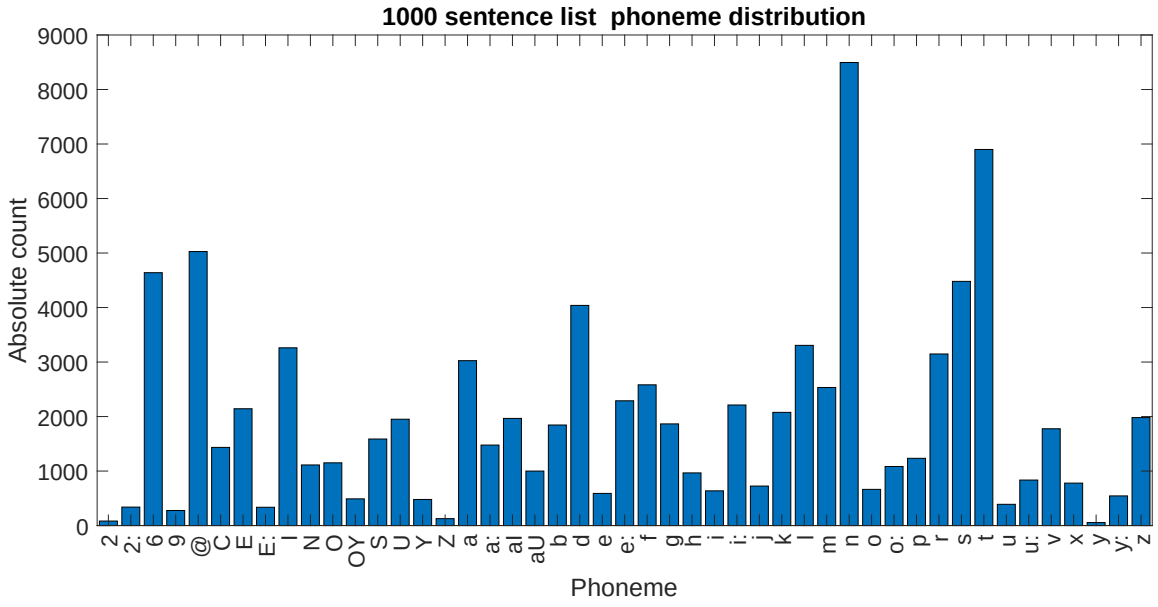

Abbildung 1: Carina corpus five phoneme distribution

2 Vocoder

Currently use the HIFI-GAN Vocoder `https://github.com/jik876/hifi-gan` with the `UNIVERSAL_V1` model. No fine-tuning necessary.

3 Architectures

Currently use an LSTM (4 Layers) with dropout of 10 % and layer size of 400 units, followed by a fully connected layer. Loss function currently is the MAE loss error to reduce smoothing induced by the MSE loss.

4 Pre-processing

- LTAS subtraction from linear magnitude spectrum
- LTAS subtraction from log10 magnitude spectrum (equal to channel normalization)
- TODO: Procrustes matching for each frame?
- TODO: I-vector approach as aux. information during training and testing for speaker normalization?
- TODO: Cepstral smoothing of radar spectra for clearer delta features?

5 Features

Tested features were:

- magnitude spectrum (linear, db)
- both magnitude variants minus the LTAS spectrum (in dB domain this corresponds to long-term channel normalization)
- delta magnitude spectrum (linear, db)
- delta phase spectrum
- impulse response

6 Corpus recording to training, step-by-step

- Record sentences along with aligned audio
- Store both in separate folders: `audio_files` and `radar_files`.
- Create a files list named `*_list.csv` for each folder.
- Run the Python script `normalize_audio_loudness.py`.
- Run the Python script `resample_audio.py` to resample from 44100 Hz to 22000 Hz.
- Run the script `calculate_mel_spectrograms.py` to write out the mel-spectrograms for each audio frame of 220 samples.

7 Evaluation on multi-speaker corpus (22.09.2022

- LTAS calculation: average magnitude spectrum across **all** sessions (non-normalized)
- Frame by frame norm. does not work well and a sequence normalization would be difficult in an actual deployment scenario, where the sequence is not known in advance. However, the test is also on a full sequence. As such, sequence normalization would mimic the case where the user waits and records the spectra and all all derived features for x seconds, which are then used to calculate the corresponding min/max values of each feature (mag, delta-mag, phase etc.).

8 Results

Currently stored under `C:/Programming/GitLab/radarspeech/speech_synthesis/results/results_corpus_5.xlsx`

With respect to MCD, the feature set [mag_db - LTAS, mag_db_delta - LTAS, phase_delta] work best currently, with a close second for the linear magnitude features. As for the delay, a delay of -3 works better than 0. More delay is likely to cause too much latency between speaking and resynthesis.

9 Next steps

10 Development history

10.1 Vocoder

Iteration 1: The used vocoder is the WORLD vocoder, as STRAIGHT is proprietary $\Rightarrow$ apparently not, Code is under `https://github.com/shuaijiang/STRAIGHT`. I will use WORLD nevertheless for now, because STRAIGHT seems quite outdated, does not work immediately in MATLAB and does not have a Python implementation (?).

Implementation switch to `https://github.com/JeremyCCHsu/Python-Wrapper-for-World-Vocoder`.

To match the frame period of the radar speech frames of 100 Hz, the `frame_period` parameter of the `Dio()` f_0 extraction function is set to 10 ms (default is 5 ms, as in [**?**]).

F_0 extraction

The synthesis is quite reliant on a good F0 estimation [**?**]. Also, the sound quality of the training data is critical. Recordings with matlab (and also for the current radar speech recordings) have significantly lower synthesis quality compared to recordings with audacity.

The `harvest()` method requires a 1 ms frame step internally with the option to interpolate to coarser time steps. It outputs both the time instants and the selected f0 candidates (object members).

Spectral envelope extraction

(Currently) uses the `cheapTrick()` function [**?**].

Parameters: q_1 and `fft_size`.

The output is an amplitude spectrogram of size `fft_size`+1. This can be condensed into MFCCs to reduce the number of predicted parameters.

The resynthesized quality from MFCCs is horrible, however. Maybe good quality audio will improve this.

Vocoder spectrograms to MFCCs

MFCC calculation pipeline:

- Start with audio signal if necessary (not if vocoder is used).
- Calculate or use N power spectra $S = |fft()|^2$ on frame basis with windowing or the vocoder output directly.
- Frequency domain filtering with triangle filterbank and summing/binning to get the Mel spectra. use `librosa.feature.melspectrogram()` or create a mel filterbank mfb $[J \times K]$ (with K frequency components, equal to the FFT size +1, and J Mel filter, one filter for each coeff) and compute the matrix product $mfb \cdot S$ $[K \times N]$. This operation is lossy.
- yields J linear mel coefficients **C** (the **Mel power spectrogram**
- Compute $10 \cdot log_{10}(\mathbf{C})$) (the log-power Mel spectrogram)
- Compute the $DCT(\mathbf{C})$ transform of the log-mel spectrogram frame wise.
- NOCH LANGZEITKANALNORMIERUNG?

The WORLD vocoder outputs a power spectrogram (according to the paper [**?**].

Iteration 2: `config.json` Bundles all configurable parameters of the synthesizer.

`train_config`	
`fp16_run`	potentially use 16 fp precision (degrades quality slightly)
`checkpoint_path`	path to saved checkpoints, including the file name (!)
`data_config`	
`segment_length`	sample segments length during training (i think, [**?**], p. 13)

Tabelle 1: waveGlow config file explanation

The sampling rate specified in the config.json needs to match the one from the training data.

Inference

The number of samples L waveGlow produces is

$$L = N_{\text{mels}} \cdot \text{hop\_length} \tag{1}$$

N_{mels} itself depends on the hop_length, fft size and window size during computation.

When calculating the mel spectrograms from a given audio file, the upper corner frequency (fmax) is important (currently at 8kHz).

Troubleshooting

- The modification to the save function (b.c. the training got Killed when saving a checkpoint) was not necessary anymore when using the HPC directly from the uni network.. don't know why, but the default load/save functions work now.

Script to get it to train on the HPC:

```
On alpha partition:
1) allocate ressources
2) module load Python
3) virtualenv --system-site-packages waveglow_env (writing to /scratch/ws/0/not possible (?))
4) source waveglow_env/bin/activate
4) python -m pip install --upgrade pip
4) python -m pip install inflect librosa tensorboardX Unidecode pillow matplotlib
5) module load modenv/hiera
6) module load GCC/10.2.0 CUDA/11.1.1 OpenMPI/4.0.5 PyTorch/1.9.0 tqdm/4.56.2
7) git clone https://github.com/NVIDIA/apex
7) cd apex
7) pip install -v --disable-pip-version-check --no-cache-dir ./
==> WaveGlow actually trains after fixing the save_checkpoint() error! On a second iteration this was
likely an HPC error and not a code error. It also works out of the box without code modifications.
```