Université de Liège

Faculté des Sciences Appliquées

Real-time Voice Cloning

Author:
Corentin Jemine

Supervisor:
Prof. Gilles Louppe

Academic year 2018 - 2019

Graduation studies conducted for obtaining the Master's degree in Data Science by Corentin Jemine

Abstract

Recent advances in deep learning have shown impressive results in the domain of text-to-speech. To this end, a deep neural network is usually trained using a corpus of several hours of professionally recorded speech from a single speaker. Giving a new voice to such a model is highly expensive, as it requires recording a new dataset and retraining the model. A recent research introduced a three-stage pipeline that allows to clone a voice unseen during training from only a few seconds of reference speech, and without retraining the model. The authors share remarkably natural-sounding results, but provide no implementation. We reproduce this framework and open-source the first public implementation of it. We adapt the framework with a newer vocoder model, so as to make it run in real-time.

Contents

1 Introduction

Deep learning models have become predominant in many fields of applied machine learning. Text-to-speech (TTS), the process of synthesizing artificial speech from a text prompt, is no exception. Deep models that would produce more natural-sounding speech than the traditional concatenative approaches begun appearing in 2016. Much of the research focus has been since gathered around making these deep models more efficient, sound more natural, or training them in an end-to-end fashion. Inference has come from being hundreds of times slower than real-time on GPU (van den Oord et al., 2016) to possible in real-time on a mobile CPU (Kalchbrenner et al., 2018). As for the quality of the generated speech, Shen et al. (2017) demonstrate near-human naturalness. Interestingly, speech naturalness is best rated with subjective metrics; and comparison with actual human speech leads to the conclusion that there might be such a thing as "speech more natural than human speech". In fact, some argue that the human naturalness threshold has already been crossed (Shirali-Shahreza and Penn, 2018).

Datasets of professionally recorded speech are a scarce resource. Synthesizing a natural voice with a correct pronunciation, lively intonation and a minimum of background noise requires training data with the same qualities. Furthermore, data efficiency remains a core issue of deep learning. Training a common text-to-speech model such as Tacotron (Wang et al., 2017) typically requires hundreds of hours of speech. Yet the ability to generate speech with any voice is attractive for a range of applications, be they useful or merely a matter of customization. Research has led to frameworks for voice conversion and voice cloning. They differ in that voice conversion is a form of style transfer on a speech segment from a voice to another, whereas voice cloning consists in capturing the voice of a speaker to perform text-to-speech on arbitrary inputs.

While the complete training of a single-speaker TTS model is technically a form of voice cloning, the interest rather lies in creating a fixed model able to incorporate newer voices with little data. The common approach is to condition a TTS model trained to generalize to new speakers on an embedding of the voice to clone (Arik et al., 2017, 2018; Jia et al., 2018). The embedding is low-dimensional and derived by a speaker encoder model that takes reference speech as input. This approach is typically more data efficient than training a separate TTS model for each speaker, in addition to being orders of magnitude faster and less computationally expensive. Interestingly, there is a large discrepancy between the duration of reference speech needed to clone a voice among the different methods, ranging from half an hour per speaker to only a few seconds. This factor is usually determining of the similarity of the generated voice with respect to the true voice of the speaker.

Our objective is to achieve a powerful form of voice cloning. The resulting framework must be able to operate in a zero-shot setting, that is, for speakers unseen during training. It should incorporate a speaker's voice with only a few seconds of reference speech. These desired results are shown to be fulfilled by (Jia et al., 2018).

Their results are impressive[1], but not backed by any public implementation. We reproduce their framework and make our implementation open-source[2]. In addition, we integrate a model based on (Kalchbrenner et al., 2018) in the framework to make it run in real-time, i.e. to generate speech in a time shorter or equal to the duration of the produced speech.

The structure of this document goes as follows. We begin with a short introduction on methods of TTS with machine learning. Follows a review of the evolution of the state of the art for TTS with speech naturalness as the core metric. We then present the work of (Jia et al., 2018) along with our own implementation. We conclude with a presentation of a toolbox we designed to interface the framework.

2 A review of text-to-speech methods in machine learning

2.1 Statistical parametric speech synthesis

Statistical parametric speech synthesis (SPSS) refers to a group of data-driven TTS methods that emerged in the late 90s. In SPSS, the relationship between the features computed on the input text and the output acoustic features is learned by a statistical generative model (called the acoustic model). A complete SPSS framework thus also includes a pipeline to extract features from the text to synthesize, as well as a system able to reconstruct an audio waveform from the acoustic features produced by the acoustic model (such a system is called a vocoder). Unlike the acoustic model, these two parts of the framework may be entirely engineered and make use of no statistical methods. While modern deep TTS models are usually not referred to as SPSS, the SPSS pipeline as depicted in Figure 1 applies just as well to those newer methods.

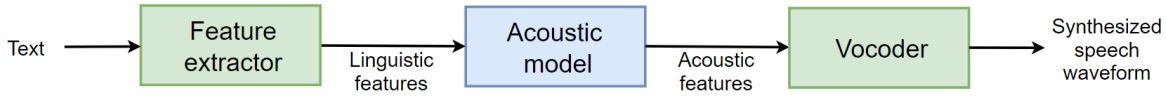

Figure 1: The general SPSS pipeline. The blue box is purely a statistical model while the green boxes can be engineered processes or/and statistical models.

The role of the feature extractor is to provide data that is more indicative of what the speech produced by the model is expected to sound like. Speech is a complex process, and directly feeding characters to a weak acoustic model will prove not to be effective. Providing additional features from natural language processing (NLP) techniques may greatly reduce the extent of the task to be learned by the acoustic model. It may however result in trade-offs when it comes to naturalness, especially

[1] https://google.github.io/tacotron/publications/speaker_adaptation/index.html
[2] https://github.com/CorentinJ/Real-Time-Voice-Cloning

for rare or unknown words. Indeed, manually engineered heuristics do not quite fully characterize all intricacies of spoken language. For this reason, feature extraction can also be done with trained models. The line between the feature extractor and the acoustic model can then become blurry, especially for deep models. In fact, a tendency that is common across all areas where deep models have overtaken traditional machine learning techniques is for feature extraction to consist of less heuristics, as highly nonlinear models become able to operate at higher levels of abstraction.

A common feature extraction technique is to build frames that will integrate surrounding context in a hierarchical fashion. For example, a frame at the syllable level could include the word that comprises it, its position in the word, the neighbouring syllables, the phonemes that make up the syllable, ... The lexical stress and accent of individual syllables can be predicted by a statistical model such as a decision tree. To encode prosody, a set of rules such as ToBI (Beckman and Elam, 1997) can be used. Ultimately, there remains a work of feature engineering to present a frame as a numerical object to the model, e.g. categorical features are typically encoded using a one-hot representation.

The reason why the acoustic model does not directly predict an audio waveform is that audio happens to be difficult to model: it is a particularly dense domain and audio signals are typically highly nonlinear. A representation that brings out features in a more tractable manner is the time-frequency domain. Spectrograms are smoother and much less dense than their waveform counterpart. They also have the benefit of being two-dimensional, thus allowing models to better leverage spatial connectivity. Unfortunately, a spectrogram is a lossy representation of the waveform that discards the phase. There is no unique inverse transformation function, and deriving one that produces natural-sounding results is not trivial. When referring to speech, this generative function is called a vocoder. The choice of the vocoder is an important factor in determining the quality of the generated audio.

As is often the case with tasks that involve generating perceptual data such as images or audio, a formal and objective evaluation of the performance of the model is difficult. In our case, we're concerned with evaluating speech naturalness and voice similarity of the generated audio. Older TTS methods often relied on statistics computed on the waveform or on the acoustic features to compare different models. Not only are those metrics often only weakly correlated with the human perception of sound, but they would also typically be what the model was aiming to minimize. "When a measure becomes a target, it ceases to be a good measure" (Strathern, 1997). A recent tendency adopted in TTS is to perform a subjective evaluation with human subjects and to report their Mean Opinion Score (MOS). The subjects are presented with a series of audio segments and are asked to rate their naturalness (or similarity when comparing two segments) on a Likert scale from 1 to 5. Because subjects do not necessarily rate actual human speech with a 5, it may be possible for TTS systems to surpass humans on this metric in the future. (Shirali-Shahreza and Penn, 2018) argue that MOS is not a metric adapted to evaluate TTS systems, and they advocate using A/B testing instead, where subjects are asked to say which audio segment they prefer

among two (a neutral vote is usually also possible). To account for the variance of opinion within subjects, one should aim to perform these studies at a large scale, with hundreds of subjects. Crowdsourcing services such as Amazon Mechanical Turk[3] are commonly involved.

2.2 Evolution of the state of the art in text-to-speech

The state of the art in SPSS has for long remained a hidden Markov model (HMM) based framework (Tokuda, 2013). This approach, laid out in Figure 2, consists in clustering the linguistic features extracted from the input text with a decision tree, and to train a HMM per cluster (Yoshimura et al., 1999). The HMMs are tasked to produce a distribution over spectrogram coefficients, their derivative, second derivative and a binary flag that indicates which parts of the generated audio should contain voice. With the maximum likelihood parameter generation algorithm (MLPG) (Tokuda et al., 2000), spectrogram coefficients are sampled from this distribution and eventually fed to the MLSA vocoder (Imai, 1983). It is possible to modify the voice generated by conditioning the HMMs on a speaker or tuning the generated speech parameters with adaptation or interpolation techniques (Yoshimura et al., 1997). Note that, while this framework used to be state of the art for SPSS, it was still inferior in terms of the naturalness of the generated speech compared to the well-established concatenative approaches.

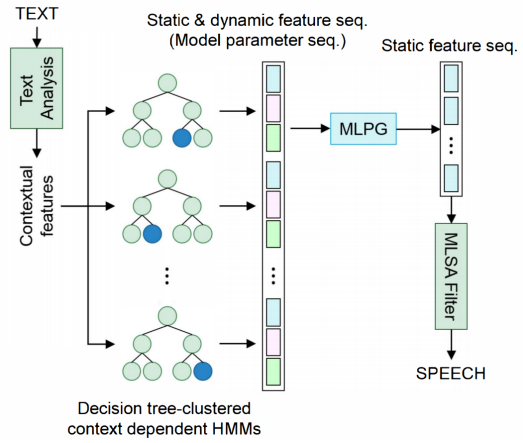

Figure 2: The general HMM-based TTS pipeline. Figure extracted from (Hashimoto et al., 2015).

Method	MOS
HMM+MLPG	3.08 (±0.12)
HMM+DNN	2.86 (±0.12)
DNN+MLPG	**3.53 (±0.12)**
DNN+DNN	3.17 (±0.12)

Table 1: MOS of the different methods explored in (Hashimoto et al., 2015). The first line is the HMM-based framework. For the second and fourth line, the MLPG algorithm is replaced by a fully-connected neural network.

Improvements to this framework were later brought by feed-forward deep neural networks (DNN), as a result of progress in both hardware and software. Zen et al. (2013) proposes to replace entirely the decision tree-clustered HMMs in favor of a DNN. They argue for better data efficiency as the training set is no longer fragmented in different clusters of contexts. They demonstrate improvements over the speech quality

[3] `https://www.mturk.com/`

with a number of parameters similar to that of the HMM-based approach. Later researches corroborate these findings (Qian et al., 2014; Hashimoto et al., 2015). The MOS of different model combinations tried by (Hashimoto et al., 2015) are reported in Table 1

(Fan et al., 2014) support that RNNs make natural acoustic models as they are able to learn a compact representation of complex and long-span functions. As RNNs are fit to generate temporally consistent series, the static features can directly be determined by the acoustic model, alleviating the need for dynamic features and MLPG. They compare networks of bidirectional LSTMs against the HMM and DNN based approaches described previously. Their A/B testing results are conclusive, we report them in Figure 3.

Figure 3: A/B testing of the models. Hybrid_B is a network with fully-connected and bidirectional LSTM layers. Figure extracted from (Fan et al., 2014).

The coming of WaveNet (van den Oord et al., 2016) made a substantial breakthrough in TTS. WaveNet is a deep convolutional neural network that, for a raw audio waveform, models the distribution of a single sample conditionally to previous ones. It is thus possible to directly generate audio by predicting samples one at a time in an autoregressive fashion. WaveNet leverages stacks of one-dimensional dilated convolutions with a dilation factor increasing exponentially with the layer depth, allowing for the very large receptive field and the strong nonlinearity needed to model raw audio. Conditioning the model on linguistic features is required to perform TTS. WaveNet acts thus both as an acoustic model and as a vocoder. Note that without the local conditioning, a trained WaveNet generates sound alike the training data but without structure or semantics (essentially babbling). The authors compare WaveNet to an older parametric approach and to a concatenative approach, the results are reported in Figure 4. The parametric approach is an LSTM-based system while the other is an HMM-driven unit selection concatenative system (not detailed in this document). Notice how the results vary between US English and Mandarin Chinese, showing that TTS performance is not language agnostic.

Follows Tacotron (Wang et al., 2017), a sequence-to-sequence model that produces a spectrogram from a sequence of characters alone, further reducing the need for domain expertise. In this framework, the vocoder is the Griffin-Lim algorithm. Tacotron uses an encoder-decoder architecture where, at each step, the decoder operates on a weighted sum of the encoder outputs. This attention mechanism, described in (Bahdanau et al., 2014), lets the network decide which steps of the input sequence are important with respect to each step of the output sequence. Tacotron achieves

[4]Figure extracted from `https://deepmind.com/blog/wavenet-generative-model-raw-audio/`. Audio samples are available at the same URL.

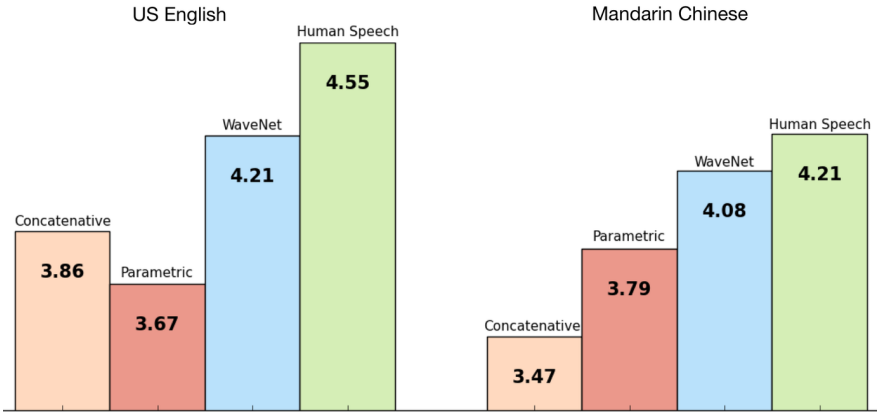

Figure 4: MOS of WaveNet's performance compared with a parametric and concatenative approach, as well as with natural speech.[4]

a MOS of 3.85 on a US English dataset, which is more than the 3.69 score obtained in the parametric approach of (Zen et al., 2016) but less than the 4.09 score obtained by the concatenative approach of (Gonzalvo et al., 2016). The authors mention that Tacotron is merely a step towards a better framework. Subsequently, Tacotron 2 is published (Shen et al., 2017). The architecture of Tacotron 2 remains that of an encoder-decoder with attention although several changes to the type of layers are made. The main difference with Tacotron is the use of a modified WaveNet as vocoder. On the same dataset, Tacotron 2 achieves a MOS of 4.53, which compares to the 4.58 for human speech (the difference is not statistically significant), achieving the all-time highest MOS for TTS. With A/B testing, Tacotron 2 was found to be only slightly less preferred on average than ground truth samples. These ratings are shown in Figure 5.

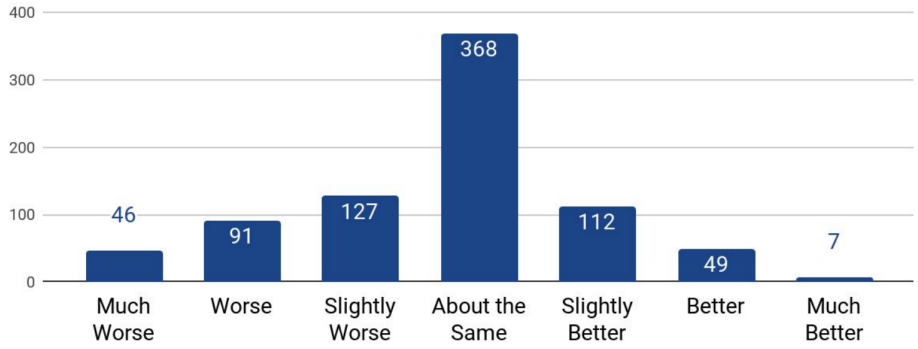

Figure 5: Preference ratings between Tacotron 2 and ground truth samples. There are 800 ratings from 100 items. The labels are expressed with respect to Tacotron 2. Figure extracted from (Shen et al., 2017).

For a global comparison of the techniques described in this section, we report in Figure 6 all the MOS of each state of the art method. Note that we have more than one result for human speech, as at least two studies evaluate it independently. The MOS for human speech is consistent across both, and the difference between them is not statistically significant. However note that the two studies have authors in common,

and thus might share a bias. Slightly different evaluation methods of the MOS may yield different results, which is one of the reasons why MOS is subject to criticism.

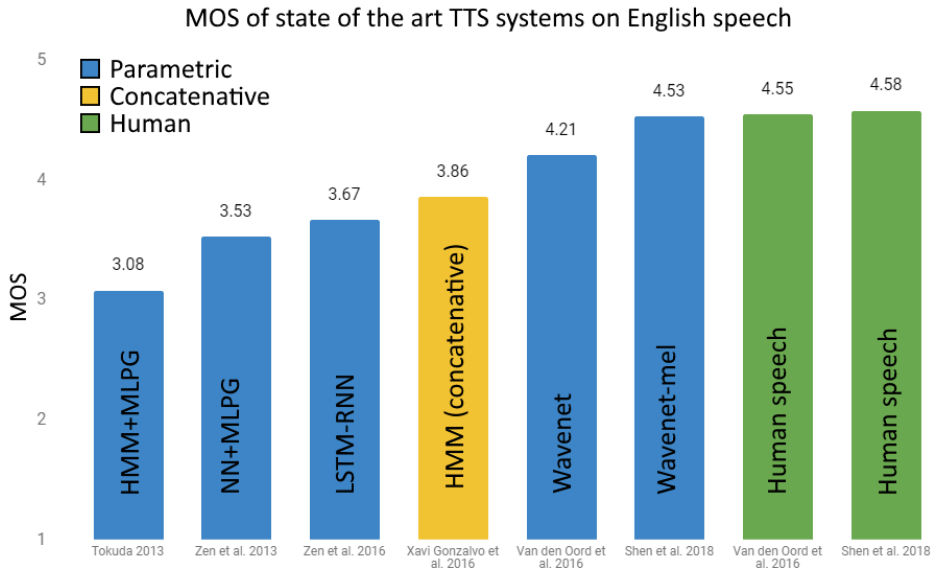

Figure 6: MOS for the methods presented in this section.

3 Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis

3.1 Overview

Our approach to real-time voice cloning is largely based on (Jia et al., 2018) (referred to as SV2TTS throughout this document). It describes a framework for zero-shot voice cloning that only requires 5 seconds of reference speech. This paper is only one of the many publications from the Tacotron series[5] authored at Google. Interestingly, the SV2TTS paper does not bring much innovation of its own, rather it is based on three major earlier works from Google: the GE2E loss (Wan et al., 2017), Tacotron (Wang et al., 2017) and WaveNet (van den Oord et al., 2016). The complete framework is a three-stage pipeline, where the steps correspond to the models listed in order previously. Many of the current TTS tools and functionalities provided by Google, such as the Google assistant[6] or the Google cloud services[7], make use of these same models. While there are many open-source reimplementations of these models online, there is none of the SV2TTS framework to our knowledge (as of May 2019).

The three stages of the framework are as follows:

[5] `https://google.github.io/tacotron/`
[6] `https://assistant.google.com/`
[7] `https://cloud.google.com/text-to-speech/`

- A speaker encoder that derives an embedding from the short utterance of a single speaker. The embedding is a meaningful representation of the voice of the speaker, such that similar voices are close in latent space. This model is described in (Wan et al., 2017) (referred as GE2E throughout this document) and (Heigold et al., 2015).
- A synthesizer that, conditioned on the embedding of a speaker, generates a spectrogram from text. This model is the popular Tacotron 2 (Shen et al., 2017) without WaveNet (which is often referred to as just Tacotron due to its similarity to the first iteration).
- A vocoder that infers an audio waveform from the spectrograms generated by the synthesizer. The authors used WaveNet (van den Oord et al., 2016) as a vocoder, effectively reapplying the entire Tacotron 2 framework.

At inference time, the speaker encoder is fed a short reference utterance of the speaker to clone. It generates an embedding that is used to condition the synthesizer, and a text processed as a phoneme sequence is given as input to the synthesizer. The vocoder takes the output of the synthesizer to generate the speech waveform. This is illustrated in Figure 7.

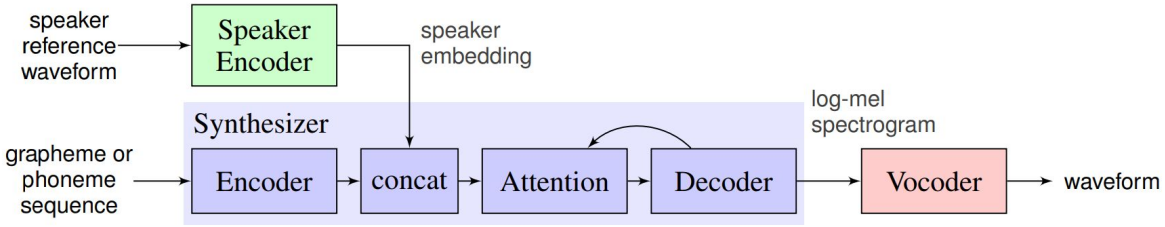

Figure 7: The SV2TTS framework during inference. The blue blocks represent a high-level view of the Tacotron architecture modified to allow conditioning on a voice. Figure extracted from (Jia et al., 2018).

A particularity of the SV2TTS framework is that all models can be trained separately and on distinct datasets. For the encoder, one seeks to have a model that is robust to noise and able to capture the many characteristics of the human voice. Therefore, a large corpus of many different speakers would be preferable to train the encoder, without any strong requirement on the noise level of the audios. Additionally, the encoder is trained with the GE2E loss which requires no labels other than the speaker identity. With GE2E, the task to be learned by the model is a speaker verification task, which by itself has little to do with voice cloning. However, the task is stipulated in way that the network will output an embedding that is a meaningful representation of the voice of the speaker. This embedding is suitable for conditioning the synthesizer on a voice, hence the name of the paper: "Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis". For the datasets of the synthesizer and the vocoder, transcripts are required and the quality of the generated audio can only be as good as that of the data. Higher quality and annotated datasets are thus required, which often means they are smaller in size.

In the following subsection, we formally define the task that SV2TTS aims to solve. In the remaining subsections, we present the three parts of the framework as described in (Jia et al., 2018), our implementation of each and the result of our experiments.

3.2 Problem definition

Consider a dataset of utterances grouped by their speaker. We denote the jth utterance of the ith speaker as $\mathbf{u}_{ij}$. Utterances are in the waveform domain. We denote by $\mathbf{x}_{ij}$ the log-mel spectrogram of the utterance $\mathbf{u}_{ij}$. A log-mel spectrogram is a deterministic, non-invertible (lossy) function that extracts speech features from a waveform, so as to handle speech in a more tractable fashion in machine learning.

The encoder $\mathcal{E}$ computes the embedding $\mathbf{e}_{ij} = \mathcal{E}(\mathbf{x}_{ij}; \mathbf{w}_{\mathcal{E}})$ corresponding to the utterance $\mathbf{u}_{ij}$, where $\mathbf{w}_{\mathcal{E}}$ are the parameters of the encoder. Additionally, the authors define a speaker embedding as the centroid of the embeddings of the speaker's utterances:

$$\mathbf{c}_i = \frac{1}{n} \sum_{j=1}^{n} \mathbf{e}_{ij} \tag{1}$$

The synthesizer $\mathcal{S}$, parametrized by $\mathbf{w}_{\mathcal{S}}$, is tasked to approximate $\mathbf{x}_{ij}$ given $\mathbf{c}_i$ and $\mathbf{t}_{ij}$, the transcript of utterance $\mathbf{u}_{ij}$. We have $\hat{\mathbf{x}}_{ij} = \mathcal{S}(\mathbf{c}_i, \mathbf{t}_{ij}; \mathbf{w}_{\mathcal{S}})$. In our implementation, we directly use the utterance embedding rather than the speaker embedding (we motivate this choice in section 3.4), giving instead $\hat{\mathbf{x}}_{ij} = \mathcal{S}(\mathbf{u}_{ij}, \mathbf{t}_{ij}; \mathbf{w}_{\mathcal{S}})$.

Finally, the vocoder $\mathcal{V}$, parametrized by $\mathbf{w}_{\mathcal{V}}$, is tasked to approximate $\mathbf{u}_{ij}$ given $\hat{\mathbf{x}}_{ij}$. We have $\hat{\mathbf{u}}_{ij} = \mathcal{V}(\hat{\mathbf{x}}_{ij}; \mathbf{w}_{\mathcal{V}})$.

One could train this framework in an end-to-end fashion with the following objective function:

$$min_{\mathbf{w}_{\mathcal{E}}, \mathbf{w}_{\mathcal{S}}, \mathbf{w}_{\mathcal{V}}} L_{\mathcal{V}}(\mathbf{u}_{ij}, \mathcal{V}(\mathcal{S}(\mathcal{E}(\mathbf{x}_{ij}; \mathbf{w}_{\mathcal{E}}), \mathbf{t}_{ij}; \mathbf{w}_{\mathcal{S}}); \mathbf{w}_{\mathcal{V}}))$$

Where $L_{\mathcal{V}}$ is a loss function in the waveform domain. This approach has drawbacks:

- It requires training all three models on a same dataset, meaning that this dataset would ideally need to meet the requirements for all models: a large number of speakers for the encoder but at the same time, transcripts for the synthesizer. A low level noise for the synthesizer and somehow an average noise level for the encoder (so as to be able to handle noisy input speech). These conflicts are problematic and would lead to training models that could perform better if trained separately on distinct datasets. Specifically, a small dataset will likely lead to poor generalization and thus poor zero-shot performance.
- The convergence of the combined model could be very hard to reach. In particular, the Tacotron synthesizer could take a significant time before producing correct alignments.

An evident way of addressing the second issue is to separate the training of the synthesizer and of the vocoder. Assuming a pretrained encoder, the synthesizer can be trained to directly predict the mel spectrograms of the target audio:

$$min_{\mathbf{w}_\mathcal{S}} L_\mathcal{S}(\mathbf{x}_{ij}, \mathcal{S}(\mathbf{e}_{ij}, \mathbf{t}_{ij}; \mathbf{w}_\mathcal{S}))$$

Where $L_\mathcal{S}$ is a loss function in the time-frequency domain.

The vocoder is then trained directly on the spectrograms. Note that both the approaches of training on ground truth spectrograms or on synthesizer-generated spectrograms are valid. The latter requires a pretrained synthesizer.

$$min_{\mathbf{w}_\mathcal{V}} L_\mathcal{V}(\mathbf{u}_{ij}, \mathcal{V}(\mathbf{x}_{ij}; \mathbf{w}_\mathcal{V})) \quad \text{or} \quad min_{\mathbf{w}_\mathcal{V}} L_\mathcal{V}(\mathbf{u}_{ij}, \mathcal{V}(\hat{\mathbf{x}}_{ij}; \mathbf{w}_\mathcal{V}))$$

Remains the optimization of the speaker encoder. Unlike the synthesizer and the vocoder, the encoder does not have labels to be trained on. The task is lousily defined as producing "meaningful" embeddings that characterize the voice in the utterance. One could conceive of a way to train the speaker encoder as an autoencoder, but it would require the corresponding upsampling model to be made aware of the text to predict. Either the dataset is constrained to a same sentence, either one needs transcripts and the upsampling model is the synthesizer. In both cases the quality of the training is impaired by the dataset and unlikely to generalize well. Fortunately, the GE2E loss (Wan et al., 2017) brings a solution to this problem and allows to train the speaker encoder independently of the synthesizer. This is described in section 3.3.

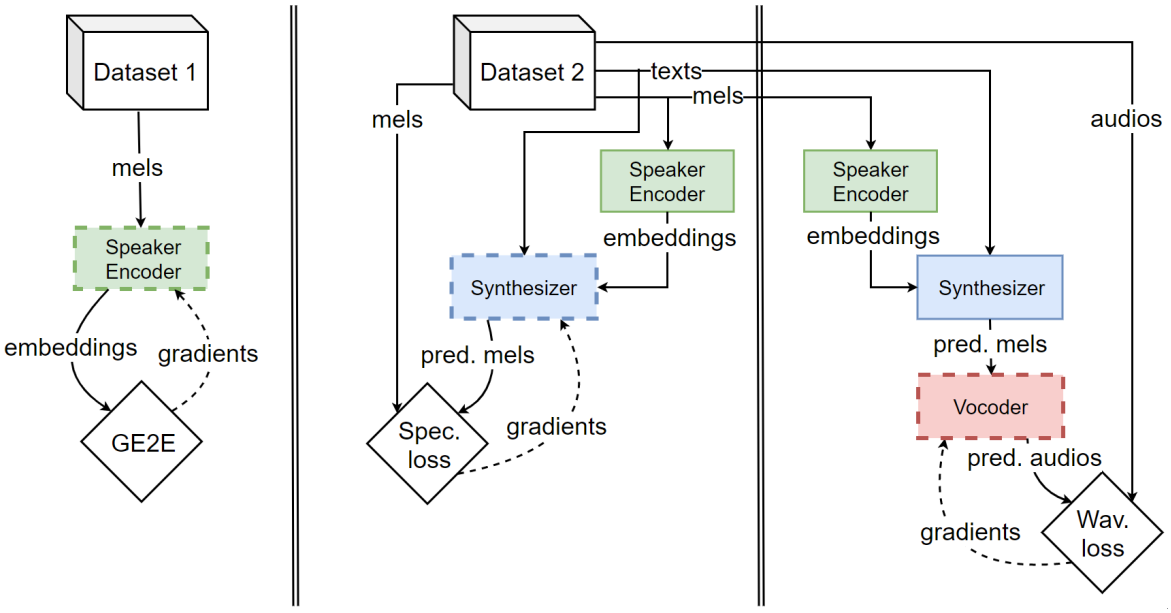

Figure 8: The sequential three-stage training of SV2TTS (following our choices of implementation). Models with solid contour lines are frozen. Note that the mel spectrograms fed to the speaker encoder and those used as target for the synthesizer are created with different parameters.

While all parts of the framework are trained separately, there is still the requirement for the synthesizer to have embeddings from a trained encoder and for the vocoder to have mel spectrograms from a trained synthesizer (if not training on ground truth spectrogram). Figure 8 illustrates how each model depends on the previous one for training. The speaker encoder needs to generalize well enough to produce meaningful embeddings on the dataset of the synthesizer; and even when trained on a common dataset, it still has to be able to operate in a zero-shot setting at inference time.

3.3 Speaker encoder

The encoder model and its training procedure are described over several papers (Jia et al., 2018; Wan et al., 2017; Heigold et al., 2015). We reproduced this model with a PyTorch implementation of our own. We synthesize the parts that are pertinent to SV2TTS as well as our choices of implementation.

3.3.1 Model architecture

The model is a 3-layer LSTM with 768 hidden nodes followed by a projection layer of 256 units. While there is no reference in any of the papers as to what a projection layer is, our intuition is that it is simply a 256 outputs fully-connected layer per LSTM that is repeatedly applied to every output of the LSTM. When we first implemented the speaker encoder, we directly used 256 units LSTM layers instead, for the sake of quick prototyping, simplicity and for a lighter training load. This last part is important, as the authors have trained their own model for 50 million steps (although on a larger dataset), which is technically difficult for us to reproduce. We found this smaller model to perform extremely well, and we haven't found the time to train the larger version later on.

The inputs to the model are 40-channels log-mel spectrograms with a 25ms window width and a 10ms step. The output is the L2-normalized hidden state of the last layer, which is a vector of 256 elements. Our implementation also features a ReLU layer before the normalization, with the goal in mind to make embeddings sparse and thus more easily interpretable.

3.3.2 Generalized End-to-End loss

The speaker encoder is trained on a speaker verification task. Speaker verification is a typical application of biometrics where the identity of a person is verified through their voice. A template is created for a person by deriving their speaker embedding (see equation 1) from a few utterances. This process is called enrollment. At runtime, a user identifies himself with a short utterance and the system compares the embedding of that utterance with the enrolled speaker embeddings. Above a given similarity

threshold, the user is identified. The GE2E loss simulates this process to optimize the model.

At training time, the model computes the embeddings $\mathbf{e}_{ij}$ $(1 \leq i \leq N, 1 \leq j \leq M)$ of M utterances of fixed duration from N speakers. A speaker embedding $\mathbf{c}_i$ is derived for each speaker: $\mathbf{c}_i = \frac{1}{M}\sum_{j=1}^{M} \mathbf{e}_{ij}$. The similarity matrix $\mathbf{S}_{ij,k}$ is the result of the two-by-two comparison of all embeddings $\mathbf{e}_{ij}$ against every speaker embedding $\mathbf{c}_k$ $(1 \leq k \leq N)$ in the batch. This measure is the scaled cosine similarity:

$$\mathbf{S}_{ij,k} = w \cdot cos(\mathbf{e}_{ij}, \mathbf{c}_k) + b = w \cdot \mathbf{e}_{ij} \cdot ||\mathbf{c}_k||_2 + b \tag{2}$$

where w and b are learnable parameters. This entire process is illustrated in Figure 9. From a computing perspective, the cosine similarity of two L2-normed vectors is simply their dot product, hence the rightmost hand side of equation 2. An optimal model is expected to output high similarity values when an utterance matches the speaker $(i = k)$ and lower values elsewhere $(i \neq k)$. To optimize in this direction, the loss is the sum of row-wise softmax losses.

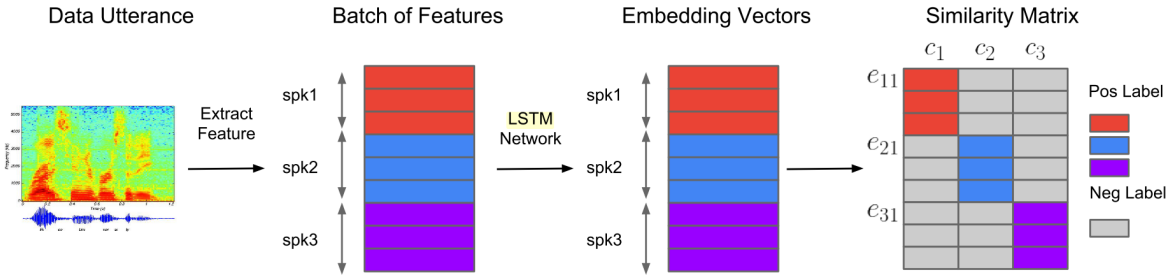

Figure 9: The construction of the similarity matrix at training time. This figure is extracted from (Wan et al., 2017).

Note that each utterance $\mathbf{e}_{ij}$ is included in the centroid $\mathbf{c}_i$ of the same speaker when computing the loss. This creates a bias towards the correct speaker independently of the accuracy of the model; and the authors argue that it also leaves room for trivial solutions. To prevent this, an utterance that is compared against its own speaker's embedding will be removed from the speaker embedding. The similarity matrix is then defined as:

$$\mathbf{S}_{ji,k} = \begin{cases} w \cdot \cos(\mathbf{e}_{ij}, \mathbf{c}_i^{(-j)}) + b & \text{if} \quad i = k \\ w \cdot \cos(\mathbf{e}_{ij}, \mathbf{c}_k) + b & \text{otherwise.} \end{cases} \tag{3}$$

where the exclusive centroids $\mathbf{c}_i^{(-j)}$ are defined as:

$$\mathbf{c}_i^{(-j)} = \frac{1}{M-1} \sum_{\substack{m=1 \\ m \neq j}}^{M} \mathbf{e}_{im} \tag{4}$$

The fixed duration of the utterances in a training batch is of 1.6 seconds. These are partial utterances sampled from the longer complete utterances in the dataset. While the model architecture is able to handle inputs of variable length, it is reasonable to expect that it performs best with utterances of the same duration as those seen in training. Therefore, at inference time an utterance is split in segments of 1.6 seconds overlapping by 50%, and the encoder forwards each segment individually. The resulting outputs are averaged then normalized to produce the utterance embedding. This is illustrated in Figure 10. Curiously, the authors of SV2TTS advocate for 800ms windows at inference time but still 1.6 seconds ones during training. We prefer to keep 1.6 seconds for both, as is done in GE2E.

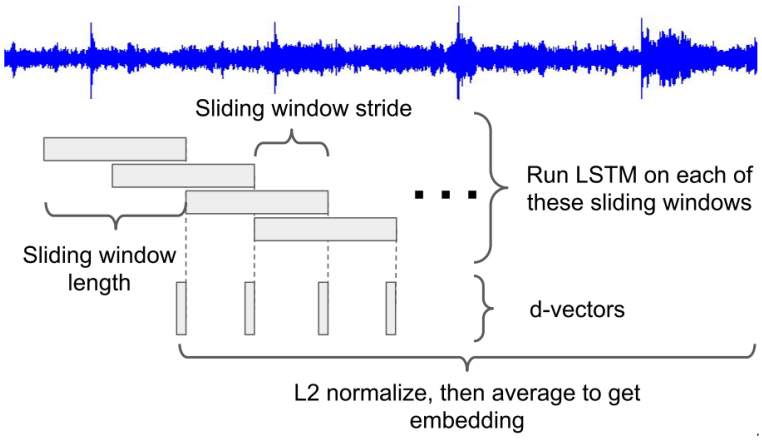

Figure 10: Computing the embedding of a complete utterance. The d-vectors are simply the unnormalized outputs of the model. This figure is extracted from (Wan et al., 2017).

The authors use $N = 64$ and $M = 10$ as parameters for the batch size. When enrolling a speaker in a practical application, one should expect to have several utterances from each user but likely not an order of magnitude above that of 10, so this choice is reasonable. As for the number of speakers, it is good to observe that the time complexity of computing the similarity matrix is $O(N^2M)$. Therefore this parameter should be chosen not too large so as to not slow down substantially the training, as opposed to simply picking the largest batch size that fits on the GPU. It is still of course possible to parallelize multiple batches on the same GPU while synchronizing the operations across batches for efficiency. We found it particularly important to vectorize all operations when computing the similarity matrix, so as to minimize the number of GPU transactions.

3.3.3 Experiments

To avoid segments that are mostly silent when sampling partial utterances from complete utterances, we use the webrtcvad[8] python package to perform Voice Activity

[8] `https://github.com/wiseman/py-webrtcvad`

Detection (VAD). This yields a binary flag over the audio corresponding to whether or not the segment is voiced. We perform a moving average on this binary flag to smooth out short spikes in the detection, which we then binarize again. Finally, we perform a dilation on the flag with a kernel size of $s + 1$, where s is the maximum silence duration tolerated. The audio is then trimmed of the unvoiced parts. We found the value $s = 0.2$s to be a good choice that retains a natural speech prosody. This process is illustrated in Figure 11. A last preprocessing step applied to the audio waveforms is normalization, to make up for the varying volume of the speakers in the dataset.

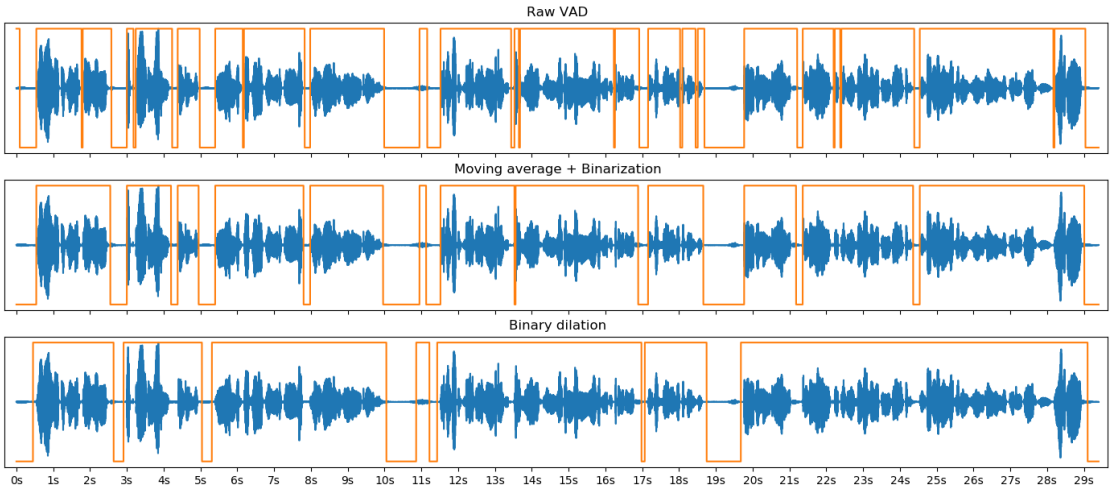

Figure 11: The steps to silence removal with VAD, from top to bottom. The orange line is the binary voice flag where the upper value means that the segment is voiced, and unvoiced when lower.

The authors combined several noisy datasets to make for a large corpus of speech of quality similar to what is found in the wild. These datasets are LibriSpeech (Panayotov et al., 2015), VoxCeleb1 (Nagrani et al., 2017), VoxCeleb2 (Chung et al., 2018) and an internal dataset, to which we do not have access. LibriSpeech is a corpus of audiobooks making up for 1000 hours of audio from 2400 speakers, split equally in two sets "clean" and "other". The clean set is supposedly made up of cleaner speech than the other set, even though some parts of the clean set still contain a lot of noise (Zen et al., 2019). VoxCeleb1 and VoxCeleb2 are made up from audio segments extracted from youtube videos of celebrities (often in the context of an interview). VoxCeleb1 has 1.2k speakers, while VoxCeleb2 has about 6k. Both these datasets have non-English speakers. We used heuristics based on the nationality of the speaker to filter non-English ones out of the training set in VoxCeleb1, but couldn't apply those same heuristics to VoxCeleb2 as the nationality is not referenced in that set. Note that it is unclear without experimentation as to whether having non-English speakers hurts the training of the encoder (the authors make no note of it either). All these datasets are sampled at 16kHz.

The authors test different combinations of these datasets and observe the effect on the quality of the embeddings. They adjust the output size of LSTM model (the size of the embeddings) to 64 or 256 according to the number of speakers. They evaluate

the subjective naturalness and similarity with ground truth of the speech generated by a synthesizer trained from the embeddings produced by each model. They also report the equal error rate of the encoder on speaker verification, which we discuss later in this section. These results can be found in Table 2.

Training Set	Speakers	Embedding Dim	Naturalness	Similarity	SV-EER
LS-Clean	1.2K	64	3.73 ± 0.06	2.23 ± 0.08	16.60%
LS-Other	1.2K	64	3.60 ± 0.06	2.27 ± 0.09	15.32%
LS-Other+VC	2.4K	256	3.83 ± 0.06	2.43 ± 0.09	11.95%
• LS-Other+VC+VC2	8.4K	256	3.82 ± 0.06	2.54 ± 0.09	10.14%
Internal	18K	256	4.12 ± 0.05	3.03 ± 0.09	5.08%

Table 2: Training of the speaker encoder on different datasets, from (Jia et al., 2018). LS is LibriSpeech and VC is VoxCeleb. The synthesizers are trained on LS-Clean and evaluated on a test set. The line with a bullet is the implementation we aim to reproduce.

These results indicate that the number of speakers is strongly correlated with the good performance of not only the encoder on the verification task, but also of the entire framework on the quality of the speech generated and on its ability to clone a voice. The small jump in naturalness, similarity and EER gained by including VoxCeleb2 could possibly indicate that the variation of languages is hurting the training. The internal dataset of the authors is a proprietary voice search corpus from 18k English speakers. The encoder trained on this dataset performs significantly better, however we only have access to public datasets. We thus proceed with LibriSpeech-Other, VoxCeleb1 and VoxCeleb2.

We train the speaker encoder for one million steps. To monitor the training we report the EER and we observe the ability of the model to cluster speakers. We periodically sample a batch of 10 speakers with 10 utterances each, compute the utterance embeddings and projecting them in a two-dimensional space with UMAP (McInnes and Healy, 2018). As embeddings of different speakers are expected to be further apart in the latent space than embeddings from the same speakers, it is expected that clusters of utterances from a same speaker form as the training progresses. We report our UMAP projections in Figure 12, where this behaviour can be observed.

As mentioned before, the authors have trained their model for 50 million steps on their proprietary dataset. While both our dataset and our model are smaller, our model still hasn't converged at 1 million steps. The loss decreases steadily with little variance and could still decrease more, but we are bound by time.

The resulting model yields very strong results nonetheless. In fact, we computed the test set EER to be **4.5**%. This is an astonishingly low value in light of the 10.14% of the authors for the same set with 50 times more steps. We do not know whether our model is actually performing that well or if the EER computation procedure of the authors is different enough than ours to produce values so far apart.

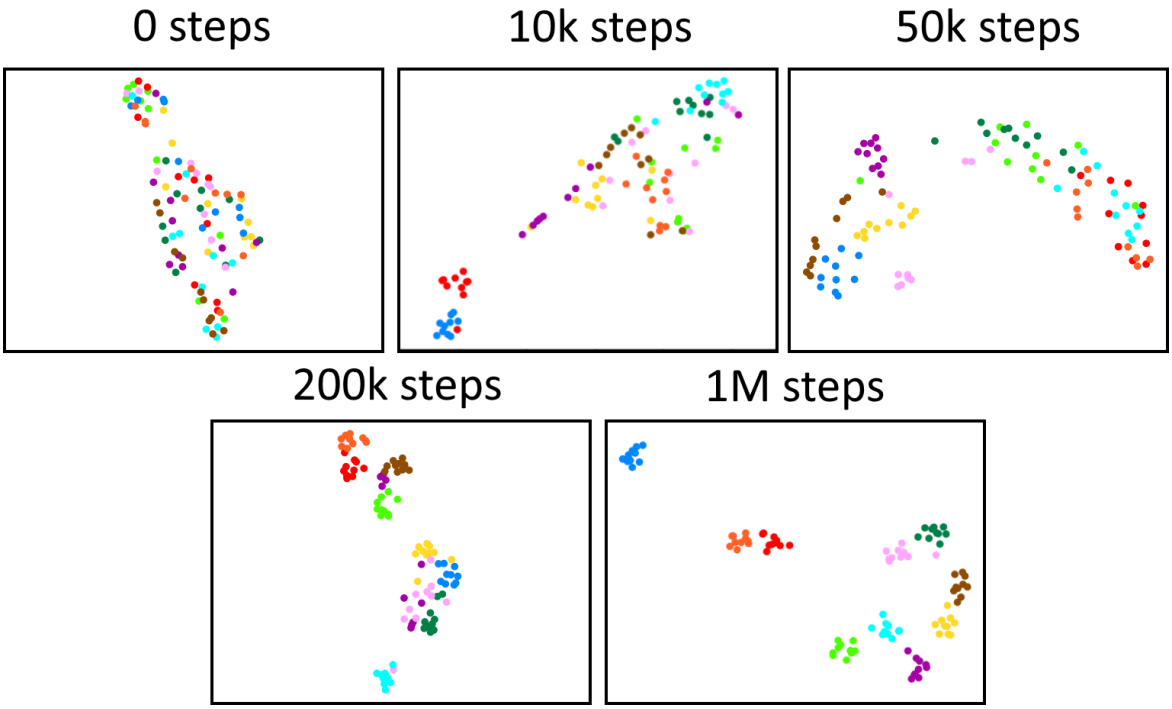

Figure 12: UMAP projections of utterance embeddings from randomly selected batches from the train set at different iterations of our model. Utterances from the same speaker are represented by a dot of the same color. We specifically omit to pass labels to UMAP, so the clustering is entirely done by the model.

We find the clustering in latent space produced by our model to be impressively robust and to generalize well. In all our tests, the UMAP projections perfectly separate utterances from the test set of each of the three datasets, with large inter-cluster distances and small intra-cluster variance. An example is given in Figure 13. The test set used for this evaluation is the combination of the test sets of LibriSpeech, VoxCeleb1 and VoxCeleb2. Speakers annotated with an F are female speakers, those with an M are male speakers. We compare our results with (Jia et al., 2018, Figure 3). We find that our projections also separate the gender of speakers linearly in the projected space. In the author's figure the utterances are from LibriSpeech only. We include VoxCeleb1 and VoxCeleb2 as well, with some of the randomly selected speakers who do not speak English. Note that our clusters are denser than those of the authors, but that this is only a result of using a different dimension reduction technique. We find results similar to theirs when using T-SNE (van der Maaten and Hinton, 2008).

The Equal Error Rate (EER) is a measure typically used in biometric systems to evaluate the accuracy of the system. It is the value of the false positive rate when it is equal to the true negative rate. Equating those terms is done by varying the similarity threshold above which a user is recognized by the biometric system. The authors of SV2TTS do not make mention of their procedure to evaluate the EER. This is problematic as the EER is tricky to compute, and highly depends on the number of enrollment utterances chosen. We refer to GE2E and use 6 utterances for enrollment

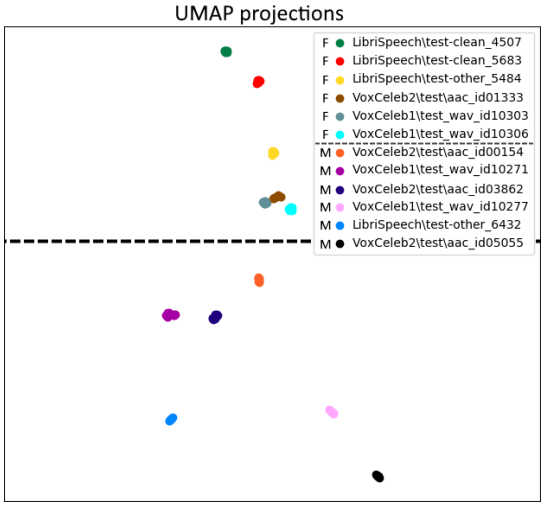

Figure 13: UMAP projections of 120 embeddings, 10 for each of the 12 speakers. Six male and six female speakers are selected at random from the test sets

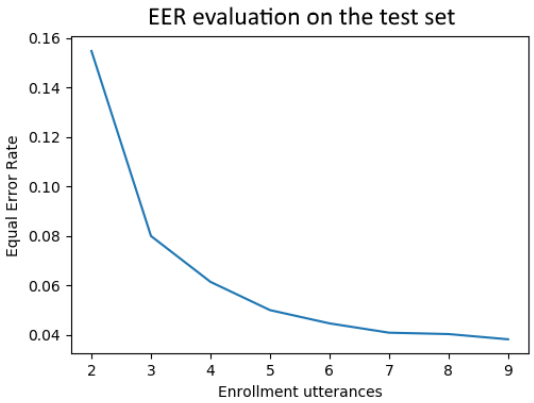

Figure 14: Speaker verification EER on the test set with respect to the number of enrollment utterances. Each speaker is tested against $n + 1$ utterances used for enrollment.

and compare those to 7 utterances. For different numbers of utterances, refer to Figure 14. They authors do not mention either if the utterances they use (both for enrollment and test) are complete or partial utterances. In light of our results, we uses partial utterances; but complete utterances would yield an even lower EER.

Due to the lack of information provided by the authors of SV2TTS and as we found no reliable source indicating how the EER is computed for such a system, we cannot guarantee the correctness nor fairness of the comparison of our results against those of the authors. Independently of them however, our tests certainly demonstrate that our speaker encoder is performing very well on the task of speaker verification. With that taken into account, we are confident that the speaker encoder is generating meaningful embeddings.

On the topic of inference speed, the encoder is by far the fastest of the three models as it operates at approximately 1000× real-time on our GTX 1080 GPU. In fact, the execution time is bounded by the GPU I/O for all utterances of our dataset.

3.4 Synthesizer

The synthesizer is Tacotron 2 without Wavenet (van den Oord et al., 2016). We use an open-source Tensorflow implementation[9] of Tacotron 2 from which we strip Wavenet and implement the modifications added by SV2TTS.

[9] `https://github.com/Rayhane-mamah/Tacotron-2`

3.4.1 Model architecture

We briefly present the top-level architecture of the modified Tacotron 2 without Wavenet (which we'll refer to as simply Tacotron). For further details, we invite the reader to take a look at the Tacotron papers (Shen et al., 2017; Wang et al., 2017).

Tacotron is a recurrent sequence-to-sequence model that predicts a mel spectrogram from text. It features an encoder-decoder structure (not to be mistaken with the speaker encoder of SV2TTS) that is bridged by a location-sensitive attention mechanism (Chorowski et al., 2015). Individual characters from the text sequence are first embedded as vectors. Convolutional layers follow, so as to increase the span of a single encoder frame. These frames are passed through a bidirectional LSTM to produce the encoder output frames. This is where SV2TTS brings a modification to the architecture: a speaker embedding is concatenated to every frame that is output by the Tacotron encoder. The attention mechanism attends to the encoder output frames to generate the decoder input frames. Each decoder input frame is concatenated with the previous decoder frame output passed through a pre-net, making the model autoregressive. This concatenated vector goes through two unidirectional LSTM layers before being projected to a single mel spectrogram frame. Another projection of the same vector to a scalar allows the network to predict on its own that it should stop generating frames by emitting a value above a certain threshold. The entire sequence of frames is passed through a residual post-net before it becomes the mel spectrogram. This architecture is represented in Figure 15.

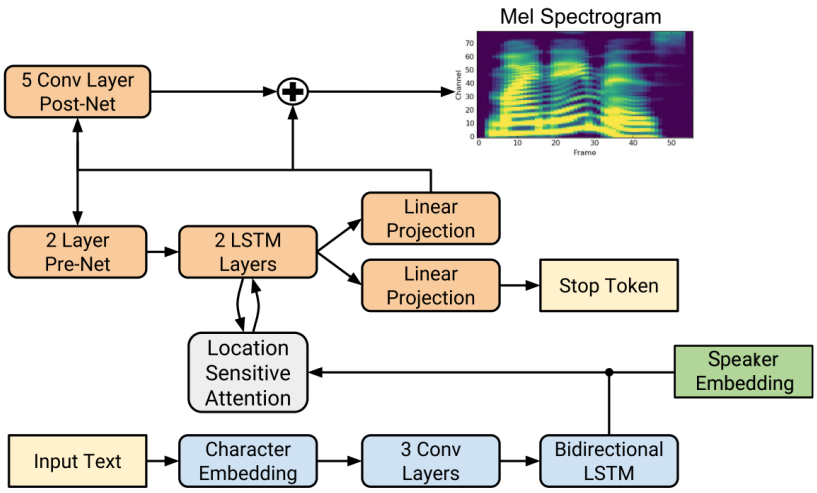

Figure 15: The modified Tacotron architecture. The blue blocks correspond to the encoder and the orange ones to the decoder. This figure was extracted from (Shen et al., 2017) and modified.

The target mel spectrograms for the synthesizer present more features than those used for the speaker encoder. They are computed from a 50ms window with a 12.5ms step and have 80 channels. The input texts are not processed for pronunciation in our implementation, and the characters are fed as they are. There are a few cleaning proce-

dures however: replacing abbreviations and numbers by their complete textual form, forcing all characters to ASCII, normalizing whitespaces and making all characters lowercase. Punctuation could be used, but isn't present in our datasets.

3.4.2 Experiments

In SV2TTS, the authors consider two datasets for training both the synthesizer and the vocoder. These are LibriSpeech-Clean which we have mentioned earlier and VCTK[10] which is a corpus of only 109 native English speakers recorded with professional equipment. The speech in VCTK is sampled at 48kHz and downsampled to 24kHz in their experiments, which is still higher than the 16kHz sampling of LibriSpeech. They find that a synthesizer trained on LibriSpeech generalizes better than on VCTK when it comes to similarity, but at the cost of speech naturalness. They assess this by training the synthesizer on one set, and testing it on the other. These results are in Table 3. We decided to work with the dataset that would offer the best voice cloning similarity on unseen speakers, and therefore picked LibriSpeech. We have also tried using the newer LibriTTS (Zen et al., 2019) dataset created by the Tacotron team. This dataset is a cleaner version of the whole LibriSpeech corpus with noisy speakers pruned out, a higher sampling rate of 24kHz and the punctuation that LibriSpeech lacks. Unfortunately, the synthesizer could not produce meaningful alignments on this dataset for reasons we ignore. We kept the original LibriSpeech dataset instead.

Synthesizer Training Set	Testing Set	Naturalness	Similarity
VCTK	LibriSpeech	4.28 ± 0.05	1.82 ± 0.08
LibriSpeech	VCTK	4.01 ± 0.06	2.77 ± 0.08

Table 3: Cross-dataset evaluation on naturalness and speaker similarity for unseen speakers. This table is extracted from (Jia et al., 2018)

Following the preprocessing recommendations of the authors, we use an Automatic Speech Recognition (ASR) model to force-align the LibriSpeech transcripts to text. We found the Montreal Forced Aligner[11] to perform well on this task. We've also made a cleaner version of these alignments public[12] to save some time for other users in need of them. With the audio aligned to the text, we split utterances on silences longer than 0.4 seconds. This helps the synthesizer to converge, both because of the removal of silences in the target spectrogram, but also due to the reduction of the median duration of the utterances in the dataset, as shorter sequences offer less room for timing errors. We ensure that utterances are not shorter than 1.6 seconds, the duration of partial utterances used for training the encoder, and not longer than 11.25 seconds so as to save GPU memory for training. We do not split on a silence

[10] `https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html`
[11] `https://montreal-forced-aligner.readthedocs.io/en/latest/`
[12] `https://github.com/CorentinJ/librispeech-alignments`

that would create an utterance too short or too long if possible. The distribution of the length of the utterances in the dataset is plotted in Figure 16. Note how long silences already account for 64 hours (13.7%) of the dataset.

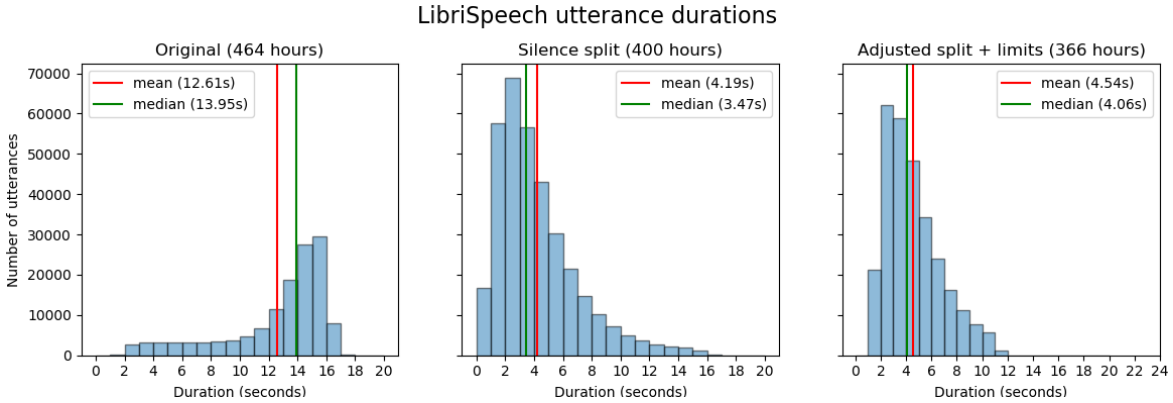

Figure 16: (left) Histogram of the duration of the utterances in LibriSpeech-Clean, (middle) after splitting on silences, (right) after constraining the length and readjusting the splits.

Isolating the silences with force-aligning the text to the utterances additionally allows to create a profile of the noise for all utterances of the same speaker. We use a python implementation[13] of the LogMMSE algorithm (Ephraim and Malah, 1985). LogMMSE cleans an audio speech segment by profiling the noise in the earliest few frames (which will usually not contain speech yet) and updating this noise profile on non-speech frames continuously throughout the utterance. We adapt this implementation to profile the noise and to clean the speech in two separate steps. On par with the authors, we found this additional preprocessing step to greatly help reducing the background noise of the synthesized spectrograms.

In SV2TTS, the embeddings used to condition the synthesizer at training time are speaker embeddings. We argue that utterance embeddings of the same target utterance make for a more natural choice instead. At inference time, utterance embeddings are also used. While the space of utterance and speaker embeddings is the same, speaker embeddings are not L2-normalized. This difference in domain should be small and have little impact on the synthesizer that uses the embedding, as the authors agreed when we asked them about it. However, they do not mention how many utterance embeddings are used to derive a speaker embedding. One would expect that all utterances available should be used; but with a larger number of utterance embeddings, the average vector (the speaker embedding) will further stray from its normalized version. Furthermore, the authors mention themselves that there are often large variations of tone and pitch within the utterances of a same speaker in the dataset, as they mimic different characters (Jia et al., 2018, Appendix B). Utterances have lower intra-variation, as their scope is limited to a sentence at most. Therefore, the embedding of an utterance is expected to be a more accurate representation of the voice spoken in the utterance than the embedding of the speaker. This holds if

[13] `https://github.com/wilsonchingg/logmmse`

the utterance is long enough than to produce a meaningful embedding. While the "optimal" duration of reference speech was found to be 5 seconds, the embedding is shown to be already meaningful with only 2 seconds of reference speech (see table 4). We believe that with utterances no shorter than the duration of partial utterances (1.6s), the utterance embedding should be sufficient for a meaningful capture of the voice, hence we used utterance embeddings for training the synthesizer.

	Reference utterance duration				
	1 sec	2 sec	3 sec	5 sec	10 sec
Naturalness (MOS)	4.28 ± 0.05	4.26 ± 0.05	4.18 ± 0.06	4.20 ± 0.06	4.16 ± 0.06
Similarity (MOS)	2.85 ± 0.07	3.17 ± 0.07	3.31 ± 0.07	3.28 ± 0.07	3.18 ± 0.07
SV-EER	17.28%	11.30%	10.80%	10.46%	11.50%

Table 4: Impact of duration of the reference speech utterance. Evaluated on VCTK. This table is extracted from (Jia et al., 2018).

We train the synthesizer for 150k steps, with a batch size of 144 across 4 GPUs. The number of decoder outputs per step is set to 2, as is done in Tacotron 1. We found the model to either not converge or perform poorly with one output per decoder step. The loss function is the L2 loss between the predicted and ground truth mel spectrograms. During training, the model is set in Ground Truth Aligned (GTA) mode (also called teacher-forcing mode), where the input to the pre-net is the previous frame of the ground truth spectrogram instead of the predicted one. With GTA, the pitch and prosody of the generated spectrogram is aligned with the ground truth, allowing for a shared context between the prediction and the ground truth as well as faster convergence. Without GTA, the synthesizer would generate different variations of the same utterance given a fixed text and embedding input (as is the case at inference time).

As was discussed in section 2.1 is also the case for the vocoder, it is difficult to provide any quantitative assessment of the performance of the model. We can observe that the model is producing correct outputs through informal listening tests, but a formal evaluation would require us to setup subjective score polls to derive the MOS. While some authors we referred to could do so, this is beyond our reach. In the case of the synthesizer however, one can also verify that the alignments generated by the attention module are correct. We plot an example in Figure 17. Notice the number of decoder steps (223) matching the number of frames predicted (446) by the number of decoder outputs per step (2). Notice also how the predicted spectrogram is smoother than the ground truth, a typical behaviour of the model predicting the mean in presence of noise.

Before training the vocoder, we can evaluate some aspects of the trained synthesizer using Griffin-Lim (Griffin and Jae Lim, 1984) as vocoder. Griffin-Lim is not a machine learning model but rather an iterative algorithm that estimates the source audio signal of a spectrogram. Audio generated this way typically conserves few of the

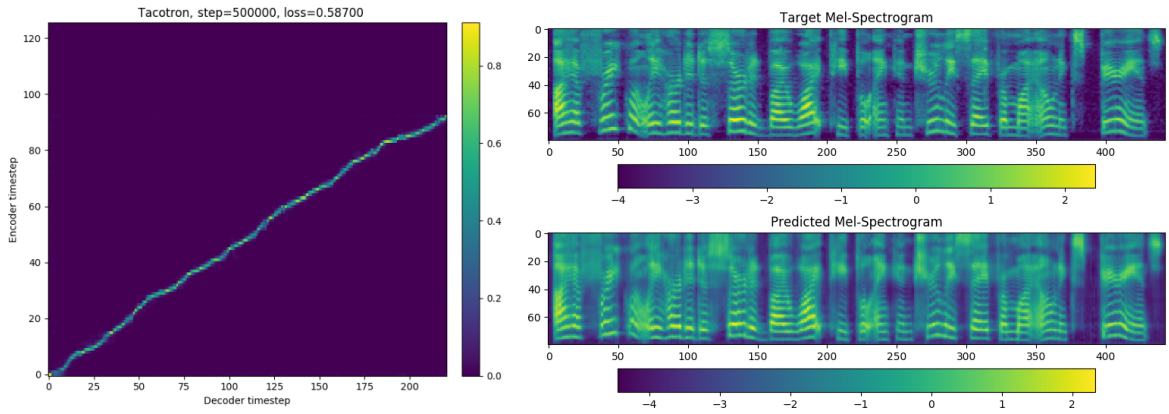

Figure 17: (left) Example of alignment between the encoder steps and the decoder steps. (right) Comparison between the GTA predicted spectrogram and the ground truth spectrogram.

voice characteristics of the speaker, but the speech is intelligible. The speech generated by the synthesizer matches correctly the text, even in the presence of complex or fictitious words. The prosody is however sometimes unnatural, with pauses at unexpected locations in the sentence, or the lack of pauses where they are expected. This is particularly noticeable with the embedding of some speakers who talk slowly, showing that the speaker encoder does capture some form of prosody. The lack of punctuation in LibriSpeech is partially responsible for this, forcing the model to infer punctuation from the text alone. This issue was highlighted by the authors as well, and can be heard on some of their samples[14] of LibriSpeech speakers. The limits we imposed on the duration of utterances in the dataset (1.6s - 11.25s) are likely also problematic. Sentences that are too short will be stretched out with long pauses, and for those that are too long the voice will be rushed. When generating several sentences at inference time, we need to manually insert breaks to delimit where to split the input text so as to synthesize the spectrogram in multiple parts. This has the advantage of creating a batch of inputs rather than a long input, allowing for fast inference.

We can further observe how some voice features are lost with Griffin-Lim by computing the embeddings of synthesized speech and projecting them with UMAP along with ground truth embeddings. An example is given in Figure 18. We observe that the synthesized embeddings clusters are close to their respective ground truth embeddings cluster. The loss of emerging features is also visible, e.g for the pink, red and the two blue speakers the synthesized utterances have a lower inter-cluster variance than their ground truth counterpart. This phenomenon occurs with the gray and purple speakers as well.

Tacotron usually operates faster than real-time. We measure an inference speed of 5× to 10× real-time, by comparing the time of generation with the duration of the generated spectrogram.

[14] `https://google.github.io/tacotron/publications/speaker_adaptation/index.html`

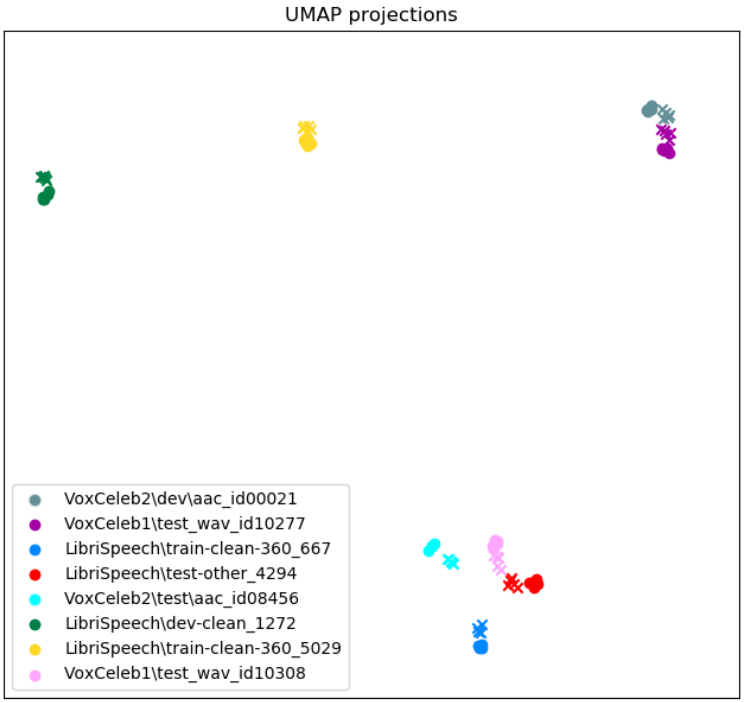

Figure 18: Projections of ground truth embeddings and of Griffin-Lim synthesized speech embeddings generated from the same ground truth embeddings. Ground truth embeddings are drawn with circles and synthesized embeddings with crosses.

3.5 Vocoder

In SV2TTS and in Tacotron2, WaveNet is the vocoder. WaveNet has been at the heart of deep learning with audio since its release and remains state of the art when it comes to voice naturalness in TTS. It is however also known for being the slowest practical deep learning architecture at inference time. Several later papers brought improvements on that aspect to bring the generation near real-time or faster than real-time, e.g. (van den Oord et al., 2017; Paine et al., 2016; Kalchbrenner et al., 2018), with no or next to no hit to the quality of the generated speech. Nonetheless, WaveNet remains the vocoder in SV2TTS as speed is not the main concern and because Google's own WaveNet implementation with various improvements already generates at 8000 samples per second (Kalchbrenner et al., 2018, page 2). This is in contrast with "vanilla" WaveNet which generates at 172 steps per second at best (van den Oord et al., 2017, page 7). At the time of the writing of this thesis, most open-source implementations of WaveNet are still vanilla implementations.

(Kalchbrenner et al., 2018) proposes a simple scheme for describing the inference speed of autoregressive models. Given a target vector $\mathbf{u}$ with $|\mathbf{u}|$ samples to predict, the total time of inference $T(\mathbf{u})$ can be decomposed as:

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

where N is the number of matrix-vector products ($\propto$ the number of layers) required to produce one sample, $c(op_i)$ is the computation time of layer i and $d(op_i)$ is the overhead of the computation (typically I/O operations) for layer i. Note that standard sampling rates for speech include 16kHz, 22.05kHz and 24kHz (while music is usually sampled at 44.1kHz), meaning that for just 5 seconds of audio $|\mathbf{u}|$ is close to 100,000 samples. The standard WaveNet architecture accounts for three stacks of 10 residual blocks of two layers each, leading to $N = 60$.

WaveRNN, the model proposed in (Kalchbrenner et al., 2018), improves on WaveNet by not only reducing the contribution from N but also from $\mathbf{u}$, $c(op_i)$ and $d(op_i)$. The vocoder model we use is an open source PyTorch implementation[15] that is based on WaveRNN but presents quite a few different design choices made by github user fatchord. We'll refer to this architecture as the "alternative WaveRNN".

3.5.1 Model architecture

In WaveRNN, the entire 60 convolutions from WaveNet are replaced by a single GRU layer (Cho et al., 2014). The authors maintain that the high non-linearity of a GRU layer alone is close enough to encompass the complexity of the entire WaveNet model. Indeed, they report a MOS of 4.51 ± 0.08 for Wavenet and 4.48 ± 0.07 for their best WaveRNN model. The inputs to the model are the GTA mel spectrogram generated by the synthesizer, with the ground truth audio as target. At training time the model predicts fixed-size waveform segments. The forward pass of WaveRNN is implemented with only $N = 5$ matrix-vector products in a coarse-fine scheme where the lower 8 bits (coarse) of the target 16 bits sample are predicted first and then used to condition the prediction of the higher 8 bits (fine). The prediction is over the parameters of a distribution from which the output is sampled. We refer the reader to (Kalchbrenner et al., 2018) for additional details.

The authors improve on the factors $c(op_i)$ and $d(op_i)$ by implementing the sampling operation as a custom GPU operation. We do not replicate this. They also sparsify the network with the pruning strategy from (Narang et al., 2017; Zhu and Gupta, 2017). This method gradually prunes weights during training, as opposed to more classical pruning algorithms that operate between several trainings. The algorithm creates a binary mask over the weights that indicates whether the weight should be forced set to 0 or remain as is. The proportion of zero weights compared to the total number of weights in the network is called sparsity. Results from (Narang et al., 2017; Zhu and Gupta, 2017) indicate that large networks with sparsity levels between 90% and 95% significantly outperform their dense versions. The authors of WaveRNN additionally argue that $c(op_i)$ is proportional to the number of nonzero weights. We experiment with this form of pruning and report our results in section 3.5.2.

Finally, they improve on $|\mathbf{u}|$ with batched sampling. In batched sampling, the utterance is divided in segments of fixed length and the generation is done in parallel

[15] `https://github.com/fatchord/WaveRNN`

over all segments. To preserve some context between the end of a segment and the beginning of the subsequent one, a small section of the end of a segment is repeated at the beginning of the next one. This process is called folding. The model then forwards the folded segments. To retrieve the unfolded tensor, the overlapping sections of consecutive segments are merged by a cross-fade. This is illustrated in Figure 19. We use batched sampling with the alternative WaveRNN, with a segment length of 8000 samples and an overlap length of 400 samples. With these parameters, a folded batch of size 2 will yield a bit more than 1 second of audio for 16kHz speech.

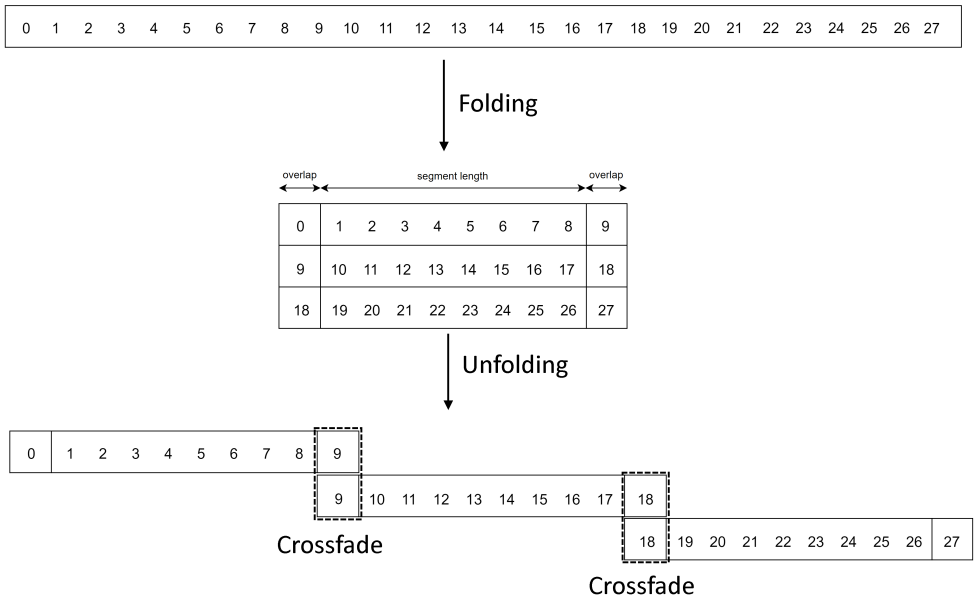

Figure 19: Batched sampling of a tensor. Note how the overlap is repeated over each next segment in the folded tensor.

The alternative WaveRNN is the architecture we use. There is no documentation nor paper for this model, so we rely on the source code and on the diagram of the author (Figure 20) to understand its inner workings. At each training step, a mel spectrogram and its corresponding waveform are cut in the same number of segments. The inputs to the model are the spectrogram segment t to predict and the waveform segment $t-1$. The model is expected to output the waveform segment t of identical length. The mel spectrogram goes through an upsampling network to match the length of the target waveform (the number of mel channels remains the same). A Resnet-like model also uses the spectrogram as input to generate features that will condition the layers throughout the transformation of the mel spectrogram to a waveform. The resulting vector is repeated to match the length of the waveform segment. This conditioning vector is then split equally four ways along the channel dimension, and the first part is concatenated with the upsampled spectrogram and with the waveform segment of the previous timestep. The resulting vector goes through several transformation with skip connections: first two GRU layers then a dense layer. Between each step, the conditioning vector is concatenated with the intermediate waveform. Finally, two dense layers produce a distribution over discrete values that correspond to a 9-bit

encoding of mu-law companded audio.

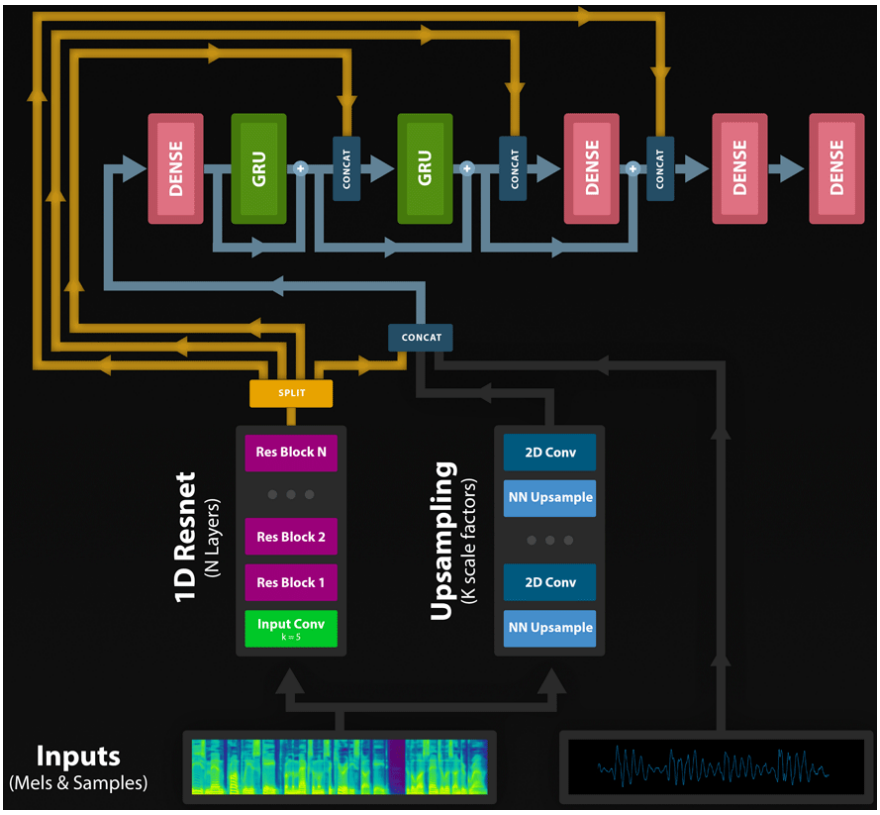

Figure 20: The alternative WaveRNN architecture.

3.5.2 Experiments

When dealing with short utterances, the vocoder usually runs below real-time. The inference speed is highly dependent of the number of folds in batched sampling. Indeed, the network runs nearly in constant time with respect to the number of folds, with only a small increase in time as the number of folds grows. We find it is simpler to talk about a threshold duration of speech above which the model runs in real time. On our setup, this threshold is of 12.5 seconds; meaning that for utterances that are shorter than this threshold, the model will run slower than real-time. It seems that performance varies unexpectedly with some environment factors (such as the operating system) on PyTorch, and therefore we express our results with respect to a single same configuration.

The implementation on our hands does not have the custom GPU operation by (Kalchbrenner et al., 2018), and implementing it is beyond our capabilities. Rather, we focus on the pruning aspect mentioned by the authors. They claim that a large sparse WaveRNN will perform better and faster than a smaller dense one. We have experimented with the pruning algorithm but did not complete the training of a pruned model, due to time limits. This is a milestone we hope to achieve at a later date.

Sparse tensors are, at the time of writing, yet an experimental feature in PyTorch. Their implementation might not be as efficient as the one the authors used. Through experiments, we find that the matrix multiply operation `addmm` for a sparse matrix and a dense vector only breaks even time-wise with the dense-only `addmm` for levels of sparsity above 91%. Below this value, using sparse tensors will actually slow down the forward pass speed. The authors report sparsity levels of 96.4% and 97.8% (Kalchbrenner et al., 2018, Table 5) while maintaining decent performances. Our tests indicate that, at best, a sparsity level of 96.4% would lower the real-time threshold to 7.86 seconds, and a level of 97.8% to 4.44 seconds. These are optimistic lower bounds on the actual threshold due to our assumption of constant time inference, and also because some layers in the model cannot be sparsified. This preliminary analysis indicates that pruning the vocoder would be beneficial to inference speed.

Unfortunately we could not complete with successfully training a final version of the vocoder before submitting this thesis. We did manage to get a prototype working by February, but this model is no longer compatible with changes we've made to the framework. We are determined to provide a working implementation before the defense of this thesis, but we cannot report of new experiments for now. Our impressions of the prototype was that our implementation was successful in creating a TTS model that could clone most voices, but not some uncommon ones. Some artifacts and background noise were present due to the poor quality of our synthesizer, which is why we had to revise the quality of our data and our preprocessing procedures. One drawback of the SV2TTS framework is the necessity to train models in sequential order. Once a new encoder is trained, the synthesizer must be retrained and so must the vocoder. Waiting for models to train so as to know on what to focus next has been a recurring situation in the development of our framework. Nonetheless we believe that we will be able to publish samples on our repository soon, and we invite the reader to follow the developments to come even if they now fall beyond the submission date.

4 Toolbox and open-sourcing

A part of the development effort went into making this project suitable for open-source. You can access the repository through this url: `https://github.com/CorentinJ/Real-Time-Voice-Cloning`. If the repository is currently inaccessible at the time of reading, it should be made open shortly.

Making the project open-source means to expose a clean interface to each model, both for training and inference, as well as documenting code and procedures. We believe that the effort is worth it. Our motivations are:

- to improve on our implementation with the contributions of users
- to learn about managing open-source repositories
- to assure the replicability of our results

- to allow users to perform voice cloning on consumer hardware with little data, time and effort.

For this last point, we aimed to develop a graphical interface allowing users to get their hands on the framework quickly and without having to study it first. We call it the "SV2TTS toolbox". The interface of the toolbox can be seen in Figure 21. It is written in Python with the Qt4 graphical interface, and is therefore cross-platform.

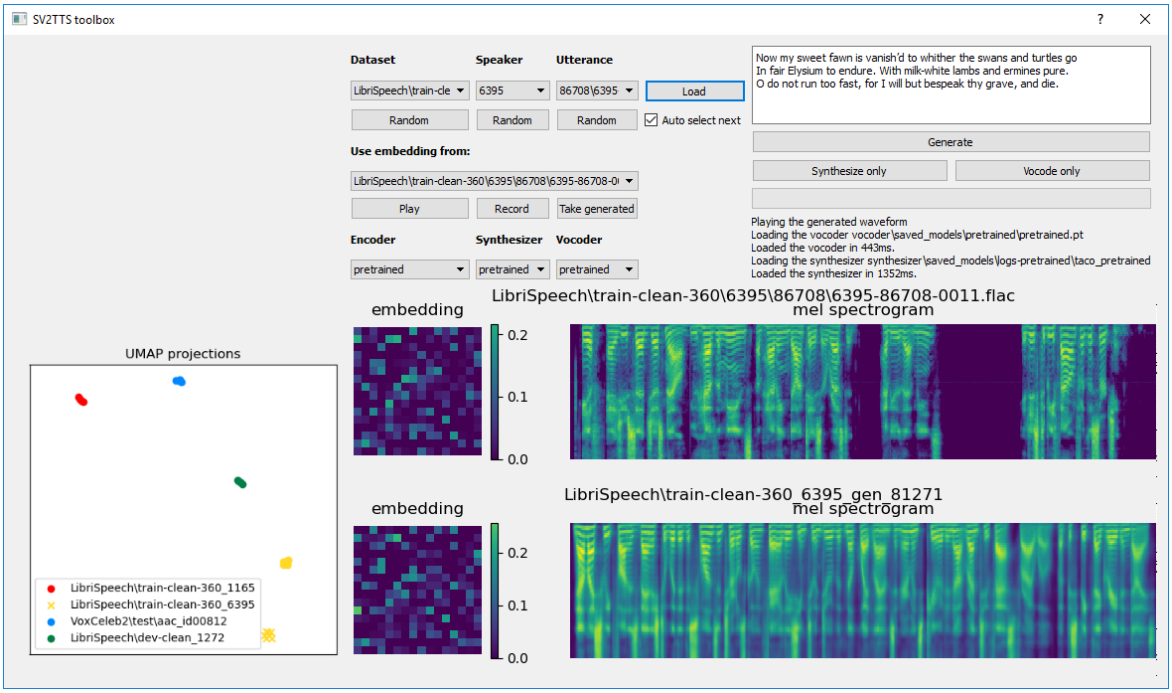

Figure 21: The SV2TTS toolbox interface. This image is best viewed on a digital support.

A user begins by selecting an utterance audio file from any of the dataset stored on their disk. The toolbox handles several popular speech datasets, and can be customized to add new ones. Furthermore, the user can also record utterances so as to clone his or her own voice.

Once an utterance is loaded, its embedding will be computed and the UMAP projections will be updated automatically. The mel spectrogram of the utterance is drawn (middle row on the right), but this is merely for reference, as it is not used to compute anything. We also draw the embedding vector with a heatmap plot (on the left of the spectrogram). Note that embeddings are unidimensional vectors and thus the square shape has no structural meaning about the embedding values. Drawing embeddings gives visual cues as to how two embeddings differ.

When an embedding has been computed, it can be used to generate a spectrogram. The user can write any arbitrary text (top right of the interface) to be synthesized. As a reminder, punctuation is not supported by our model and will be discarded. To tune the prosody of the generate utterance, the user has to insert line

breaks between parts that should be synthesized individually. The complete spectrogram is then the concatenation of those parts. Synthesizing a spectrogram will display it on the bottom right of the interface. Synthesizing the same sentences multiple times will yield different outputs.

Finally, the user can generate the segment corresponding to the synthesized spectrogram with the vocoder. A loading bar displays the progress of the generation. When done, the embedding of the synthesized utterance is generated (on the left of the synthesized spectrogram) and will also be projected with UMAP. The user is free to take that embedding as reference for further generation.

5 Conclusion

We developed a framework for real-time voice cloning that had no public implementation. We find the results to be satisfying despite some unnatural prosody, and the voice cloning ability of the framework to be reasonably good but not on par with methods that make use of more reference speech time. We hope to improve our framework beyond the scope of this thesis, and possibly to implement some of the newer advances in the field that were made at the time of writing. We believe that even more powerful forms of voice cloning will become available in a near future.

Acknowledgments

I would like to thank my advisor Pr. Gilles Louppe for his wise advices, his dedication and his patience.

I would like to thank mr. Quan Wang for answering my questions about his research.

I would like to thank github users Rayhane-mamah[16], fatchord[17] and begeekmyfriend[18] for their contributions to the open source repositories used in this project.

I also wish to thank teaching assistants Joeri Hermans and Romain Mormont for managing the university's deep learning cluster and for providing me with technical assistance.

Finally, I wish to thank my sister Marie and sister-in-law Virginie for proofreading this thesis.

[16] `https://github.com/Rayhane-mamah`
[17] `https://github.com/fatchord`
[18] `https://github.com/begeekmyfriend`

References

Sercan Arik, Gregory Diamos, Andrew Gibiansky, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep voice 2: Multi-speaker neural text-to-speech, 2017.

Sercan O. Arik, Jitong Chen, Kainan Peng, Wei Ping, and Yanqi Zhou. Neural voice cloning with a few samples, 2018.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL `http://arxiv.org/abs/1409.0473`.

Mary E. Beckman and Gayle Ayers Elam. Guidelines for tobi labelling, 03 1997.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, KyungHyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *CoRR*, abs/1506.07503, 2015. URL `http://arxiv.org/abs/1506.07503`.

J. S. Chung, A. Nagrani, and A. Zisserman. Voxceleb2: Deep speaker recognition. In *INTERSPEECH*, 2018.

Y. Ephraim and D. Malah. Speech enhancement using a minimum mean-square error log-spectral amplitude estimator. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(2):443–445, April 1985. ISSN 0096-3518. doi: 10.1109/TASSP.1985.1164550.

Y Fan, Yuang Qian, Feng-Long Xie, and Frank Soong. Tts synthesis with bidirectional lstm based recurrent neural networks. pages 1964–1968, 01 2014.

Xavi Gonzalvo, Siamak Tazari, Chun-an Chan, Markus Becker, Alexander Gutkin, and Hanna Silen. Recent advances in google real-time hmm-driven unit selection synthesizer. In *Interspeech*, pages 2238–2242, 2016.

D. Griffin and Jae Lim. Signal estimation from modified short-time fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2):236–243, April 1984. ISSN 0096-3518. doi: 10.1109/TASSP.1984.1164317.

K. Hashimoto, K. Oura, Y. Nankaku, and K. Tokuda. The effect of neural networks in statistical parametric speech synthesis. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4455–4459, April 2015. doi: 10.1109/ICASSP.2015.7178813.

Georg Heigold, Ignacio Moreno, Samy Bengio, and Noam Shazeer. End-to-end text-dependent speaker verification. *CoRR*, abs/1509.08062, 2015. URL `http://arxiv.org/abs/1509.08062`.

S. Imai. Cepstral analysis synthesis on the mel frequency scale. In *ICASSP '83. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 8, pages 93–96, April 1983. doi: 10.1109/ICASSP.1983.1172250.

Ye Jia, Yu Zhang, Ron J. Weiss, Quan Wang, Jonathan Shen, Fei Ren, Zhifeng Chen, Patrick Nguyen, Ruoming Pang, Ignacio Lopez-Moreno, and Yonghui Wu. Transfer learning from speaker verification to multispeaker text-to-speech synthesis. *CoRR*, abs/1806.04558, 2018. URL `http://arxiv.org/abs/1806.04558`.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis, 2018.

Leland McInnes and John Healy. Umap: Uniform manifold approximation and projection for dimension reduction. 02 2018.

A. Nagrani, J. S. Chung, and A. Zisserman. Voxceleb: a large-scale speaker identification dataset. In *INTERSPEECH*, 2017.

Sharan Narang, Gregory F. Diamos, Shubho Sengupta, and Erich Elsen. Exploring sparsity in recurrent neural networks. *CoRR*, abs/1704.05119, 2017. URL `http://arxiv.org/abs/1704.05119`.

Tom Le Paine, Pooya Khorrami, Shiyu Chang, Yang Zhang, Prajit Ramachandran, Mark A. Hasegawa-Johnson, and Thomas S. Huang. Fast wavenet generation algorithm, 2016.

V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, April 2015. doi: 10.1109/ICASSP.2015.7178964.

Y. Qian, Y. Fan, W. Hu, and F. K. Soong. On the training aspects of deep neural network (dnn) for parametric tts synthesis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3829–3833, May 2014. doi: 10.1109/ICASSP.2014.6854318.

Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, and Yonghui Wu. Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions. *CoRR*, abs/1712.05884, 2017. URL `http://arxiv.org/abs/1712.05884`.

S. Shirali-Shahreza and G. Penn. Mos naturalness and the quest for human-like speech. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, pages 346–352, Dec 2018. doi: 10.1109/SLT.2018.8639599.

Marilyn Strathern. 'improving ratings': audit in the british university system. *European Review*, 5(3):305–321, 1997. doi: 10.1002/(SICI)1234-981X(199707)5:3⟨305::AID-EURO184⟩3.0.CO;2-4.

K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura. Speech parameter generation algorithms for hmm-based speech synthesis. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No.00CH37100)*, volume 3, pages 1315–1318 vol.3, June 2000. doi: 10.1109/ICASSP.2000.861820.

Yoshihiko; Toda Tomoki; Zen Heiga; Yamagishi Junichi; Oura Keiichiro Tokuda, Keiichi; Nankaku. Speech synthesis based on hidden markov models. *Proceedings of the IEEE*, 101, 05 2013. doi: 10.1109/JPROC.2013.2251852.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL `http://arxiv.org/abs/1609.03499`.

Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017. URL `http://arxiv.org/abs/1711.10433`.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008. URL `http://www.jmlr.org/papers/v9/vandermaaten08a.html`.

Li Wan, Quan Wang, Alan Papir, and Ignacio Lopez Moreno. Generalized end-to-end loss for speaker verification, 2017.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc V. Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: A fully end-to-end text-to-speech synthesis model. *CoRR*, abs/1703.10135, 2017. URL `http://arxiv.org/abs/1703.10135`.

Takayoshi Yoshimura, Takashi Masuko, Keiichi Tokuda, Takao Kobayashi, and Tadashi Kitamura. Speaker interpolation in hmm-based speech synthesis system. In *EUROSPEECH*, 1997.

Takayoshi Yoshimura, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi, and Tadashi Kitamura. Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis. In *EUROSPEECH*, 1999.

H. Zen, A. Senior, and M. Schuster. Statistical parametric speech synthesis using deep neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7962–7966, May 2013. doi: 10.1109/ICASSP.2013.6639215.

Heiga Zen, Yannis Agiomyrgiannakis, Niels Egberts, Fergus Henderson, and Przemyslaw Szczepaniak. Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices. *CoRR*, abs/1606.06061, 2016. URL `http://arxiv.org/abs/1606.06061`.

Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J. Weiss, Ye Jia, Zhifeng Chen, and Yonghui Wu. Libritts: A corpus derived from librispeech for text-to-speech. *CoRR*, abs/1904.02882, 2019. URL `http://arxiv.org/abs/1904.02882`.

Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression, 2017.