MSc Artificial Intelligence
Master Thesis

Whisper to Voice

by

Bram Kooiman

11415665

June 24, 2019

36 EC

January - June 2019

Supervisor:
Prof. Paul Boersma

Assessor:
Dr. Zeynep Akata

University of Amsterdam

Abstract

Whisper to Voice

by Bram Kooiman

Speaking in a whispered voice can greatly reduce stuttering. Inspired by this fact, we research whisper-to-voice conversion models that can perform in real-time on phone hardware. We place special interest in paradigms that don't require paired data, transcribed data or data alignment by dynamic time warping. We find it necessary to condition on speaker identity, but show that the model can successfully learn to adapt to new speakers with few examples. The model shows promising results for whisper-to-speech conversion while maintaining the constraint that the model be fast and lightweight.

Keywords: *real-time performance, spectrograms, cycle-consistency, WGAN, speaker embedding.*

Acknowledgements

Special thanks to Joris Castermans of Castermans Connected, Majid Bahrepour of Luminis, Prof. Paul Boersma of the University of Amsterdam and Prof. David van Leeuwen of the Radboud University for providing ideas, guidance and/or support throughout the research.

Contents

Chapter 1

Introduction

1.1 Whispp

Stutterers find it hard to speak naturally, and can get behind in social settings or miss job opportunities because of it. Strangely enough, most stutterers have no problem singing, playing wind instruments or — as it turns out — whispering [17]. In this thesis we explore the possibilities of Deep Learning for whisper-to speech-conversion (W2S). The insights gained by this research will be used to develop Whispp, an app that helps stutterers speak freely. With the app, a stutterers can make a phone call or even speak face-to-face (which would require especially low latency).

Though the causes and therapies for stuttering are outside the scope of this thesis, we hope that Whispp will eventually be incorporated into stutter therapies and teach stutterers confidence of speech, or at least give them a workaround for the time being. Uninhibited speakers may also benefit in situations where they want to make a phone call without disturbing a silent environment.

The app will be downloadable in the play store from late 2020 onward, but data acquisition for Whispp is ongoing

Everyone is encouraged 'donate their voice' via:
www.whispp.com
username: Whispp
password: HelpWhispp

1.2 Speech-to-Text, Text-to-Speech

The first idea that comes to mind is to connect Speech-to-Text and Text-to-Speech. Both methods are well-researched and even available for plugging into a custom application[1][2] [40, 13, 50]. Though it is a very good place to start, it is not free of objections. Most importantly, a textual (or phonemic) representation will get rid of information about the speaker's identity and the prosody of the sentence she speaks. Prosody is the 'melody' of speech, and the rhythm of syllabic emphasis. If we don't want to lose this info, it is possible to learn to recognise it from whisper (parallel to STT), represent it with some latent vector and condition the TTS network on it.

In this thesis we propose to *only* perform the latter strategy and let go of an intermediate symbolic representation. This thesis focuses on the conversion from audio $\rightarrow$ audio, not from audio $\rightarrow$ text $\rightarrow$ audio. As a small bonus, this will free us from requiring phoneme-level labeling on the data.

[1]https://cloud.google.com/speech-to-text/
[2]https://cloud.google.com/text-to-speech/

Translatotron [20] is a speech-to-speech language translator that shows that it is possible to learn all of audio recognition, semantics, grammar and audio synthesis *implicitly*.

1.3 Mathmatical Conventions

- For scalars, we denote x. Vectors (and timeseries of scalars) $\mathbf{x}$ are bold. An entry of a vector is a scalar x_i, and is not bold. Matrices or tensors X are capitals. An entry of a matrix can be a vector and will be denoted by $\mathbf{x}_t$. Nontheless, a single entry of a matrix or tensor will be denoted by a capital X_{ij}.
- $\oslash$ and $\odot$ are element-wise matrix operations (division and multiplication). A matrix squared X^2 denotes an elementwise operation. XY denotes a matrix multiplication.
- $||...||$ is the Euclidean norm.

$$||\mathbf{x}|| = \sqrt{\sum x_i^2}$$

- The imaginary number is i:

$$x = 2 + 3i \; ; \; \Re(x) = 2 \; ; \; \Im(x) = 3 \; ; \; \sqrt{i} = -1$$

- The space of all real numbers is denoted with $\mathbb{R}$ and the space of all complex numbers is denoted with $\mathbb{C}$. $\mathbf{x} \in \mathbb{R}^3$ specifies that $\mathbf{x}$ is a 3-dimensional real-valued vector and $X \in \mathbb{C}^{3\times3}$ specifies that X is a complex-valued 3-by-3 matrix.
- A domain of [0, 1) is continuous and envelops all numbers between 0 and 1, but not 1 itself. A set of integers is denoted with curly brackets {0,1}.
- Sampling from a distribution is denoted by $x \sim U(0, 1)$.
- The Gaussian distribution is denoted by $\mathcal{N}(\mu, \sigma)$
- $\sigma(\cdot)$ denotes the logistic sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Unless otherwise specified, 'logarithmic' refers to the natural logarithm (base e).
- $\nabla_Y X$ and $\frac{\partial X}{\partial Y}$ mean the same thing and denote the gradient of X with respect to Y.
- f_θ is a function parameterized by θ. We'll often use greek symbols for parameters.
- $\mathcal{L}$ is used to denote a loss. It is always the objective to minimize it.

1.4 Report structure

Chapter 2 discusses background information on audio processing and speech. In Chapter 3 we discuss three paradigms for whisper-to-speech conversion, namely LSTMs for sequence-to-sequence translation (W2S-L), LSTMs with RBM-pretraining (W2S-R) and Convolutional U-net with WGAN training (W2S-C). In chapter 4 we encourage the network to output audio with the true spoken voice of the whisperer (W2S-aux and W2S-embed). Throughout the thesis, we map spectrograms to audio with the Griffin-Lim algorithm. In chapter 5 explore alternatives (PhaseNet, W2S-M and an implicit source-filter model), but none improve over the Griffin-Lim algorithm. Chapter 6 discusses areas of future research, primarily alternatives to using spectrograms altogether.

Audio samples are available on google drive via the following link:
https://drive.google.com/open?id=1h3V9SbvBUwIp8GCgroEJSSG3hT6eB3fh.

The GitHub repo of this work is protected and only available to the reviewers of this work.

Chapter 2

Background

This chapter gives a general overview of relevant audio processing concepts. Section 2.1 explains the difference in whispered and voiced speech, and brings to light the challenges of conversion. Sections 2.2 - 2.6 discuss audio features. Sections 2.8 - 2.10 discuss older paradigms for whisper-to-speech conversion. Section 2.11 discusses Dynamic Time Warping (DTW); a method that warps two time series closer together so that each entry in one series can act as a supervised target for each entry in the other series.

2.1 Whispered and Voiced Speech

There are two types of phonemes; voiced and unvoiced phonemes[1]. A voiced phoneme requires that the vocal cords vibrate. A quick check to see if a phoneme is voiced is to press the throat while speaking the phoneme; you will feel the vibration. The difference in pitch (high voice, low voice) comes from how the vocal cords vibrate. The difference between phonemes comes from how that sound is ’filtered’ by the shape of the mouth and tongue.

The main difference between whisper and speech is that during whisper the vocal cords never vibrate. For voiced phonemes — which should be accompanied by vibrating vocal chords — this means that whispering and speaking is very different. By doing various ’tricks’ with the higher frequencies present in whisper we can give the impression of there being a pitch, while in reality there is none. For unvoiced phonemes — which are inherently pitchless — the difference between whisper and speech is not as pronounced.

Prosody is the ’melody’ of speech and the rhythm of syllabic emphasis. In English and Dutch, emphasized phonemes are not monotonous but have a sliding pitch profile (usually downward, but upward for questions). Subtler cues are a raised intensity (volume) and longer duration.

It seems there’s two primary challenges for a whisper-to-speech conversion model: it must know when synthesising pitch is appropriate (only when voiced phonemes are being whispered) and the synthesized pitch must adhere to the prosodic profile of the whisper. Additionally, the voice of the synthesized speech should resemble that of the speaker.

2.2 Spectrograms

Sound consists of changes in air pressure. It can be best represented with a one-dimensional array; air pressure values for discrete points in time. Such arrays are

[1]which phonemes are voiced and unvoiced can be found at http://www.phonemicchart.com/

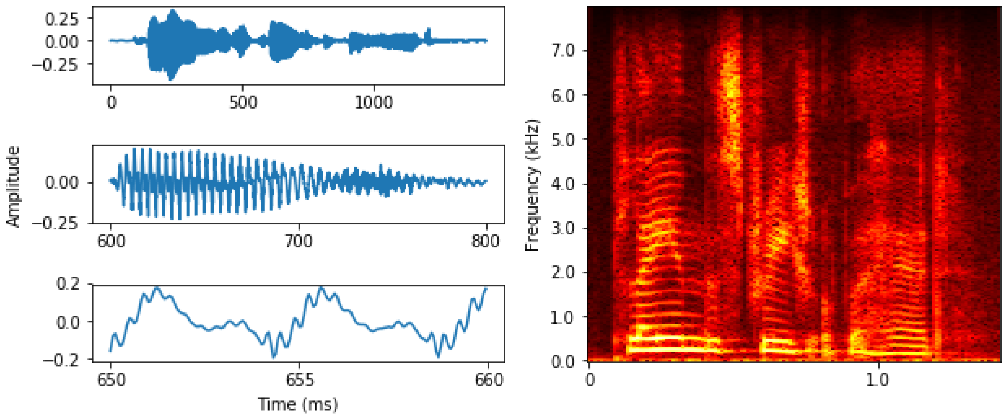


FIGURE 2.1: The raw audio wave and a spectrogram of a short utterance. The waveform is of samplerate 16kHz and the spectrogram has been obtained with a fourier window of 36ms and hop length of 12ms. The spectrogram is enhanced by $\sqrt[3]{10 \cdot \log(x+1)}$ for better visualisation.

very fine-grained and typically have many thousands of entries per second (44.1k entries per second is a common setting). Because there are so many entries and because the entries by themselves encode no high-level semantic meaning it is a complicated matter to perform analysis or manipulation on the raw audio waveform.

The spectrogram is a common alternative representation of audio. It shows the frequencies that are present within a short slice of time. This is similar to how humans experience sound; we can hear high and low frequencies simultaneously. By evaluating slices of time consecutively, we can see how frequency presence changes over time. The spectrogram is thus a two-dimensional representation of sound.

The Math. The spectrogram is derived from the short-time Fourier transform ($STFT$). We evaluate $STFT$ at F frequencies of interest and over T temporal bins. We call the result of the $STFT$, $X \in \mathbb{C}^{F \times T}$, the 'complex spectrogram'. The spectrogram $A \in \mathbb{R}^{F \times T}$ (or its full name the 'magnitude spectrogram') is its absolute value. Suppose we have a discrete signal x. We compute the presence of a frequency f in a small frame of x consisting of N samples, starting at timepoint m.

$$X_{fm} = STFT_{fm}(\mathbf{x}) = \sum_{n=m}^{m+N} x_n w_n e^{i\phi n} \tag{2.1}$$

where

$$\phi = \frac{2\pi f}{N}$$

and w is a windowing function such as the Hamming window. Windowing functions are close to zero near the borders of the window. It is customary to take overlapping windows controlled by a parameter h called the 'hop-size'. This is done to combat the loss of information at the borders of the frames due to the windowing operation.

The complex spectrogram has a real and an imaginary part. The words 'real' and 'imaginary' are unfortunate because of their meaning in everyday life. Engineers sometimes prefer the terms 'in-phase' and 'quadrature phase'. Indeed, if we rewrite

$STFT$ according to Euler's formula we can see the origin of this naming convention.

$$e^{i\theta} = cos(\theta) + i\ sin(\theta)$$

$$X_{fm} = STFT_{fm}(\mathbf{x}) = \sum_{n=m}^{m+N} x_n w_n (cos(\phi n) + i\ sin(\phi n))$$

The 'real' part of the complex value is simply how much the windowed signal agrees with an in-phase perfect cosine of the frequency of interest and the 'imaginary' part is how much the windowed signal agrees with a perfect cosine of the frequency a quarter phase off (a perfect sine).

We can consider the in-phase and quadrature-phase components of the $STFT$ to be the axes of a cartesian coordinate system. The spectrogram value is then the radius of the polar coordinate system. Hence spectrogram values are always real and non-negative.

$$A = |X| \tag{2.2}$$

$$A_{ft} = \sqrt{\Re(X_{ft})^2 + \Im(X_{ft})^2}$$

The magnitude spectrogram A of equation 2.2 is an 'energy spectrogram'. It is also common to take the 'power spectrogram' $|X|^2$ or normalize the spectrogram to decibel scale by $A \leftarrow \log(A + \epsilon)$, where ϵ is a small constant to prevent numerical instability (because $log(0) = -\infty$). We shall be using the normalization strategy $\sqrt[3]{10 \cdot \log(x + 1)}$, where x is a scalar entry of the energy spectrogram, because it maps the values to a non-negative domain where most are between 0 and 1. This normalization strategy works well for visualisation and we've found it also works well as an input to neural networks.

In this work we use the shorthand 'spectrogram' for 'magnitude spectrogram', spectrograms are only complex if explicitly stated so. We shall also coin the term 'radial spectrograms', which is a complex spectrogram expressed in polar coordinates.

FFT. The operation of 2.1 has computational complexity $\mathcal{O}(n^2)$. The Fast Fourier Transform of Cooley and Tukey [5] makes uses of the symmetry of the operation to reduce computational complexity to $\mathcal{O}(n \log n)$. This trick only works if the frequencies are linearly spaced.

The Uncertainty Principle. Because we must assume a signal in a window to be stationary we encounter an uncertainty principle. If we take very small windows we can be very certain in how the frequencies develop temporally, but there will be more uncertainty in exactly what frequencies are present. Vice versa, if we take large windows we have high resolution in the frequency dimension, but low resolution in the temporal dimension.

The Missing-Phase Problem. By extracting spectrograms we lose phase information. The problem with this is most evident in audio synthesis. The inverse of $STFT$ requires a complex spectrogram. If a magnitude spectrogram has been obtained, care must be taken on how to modify its real values to complex before the inverse $STFT$ can synthesize audio from it. Chapter 5 delves deeper into this subject. For now, we resolve the missing-phase problem with the Griffin-Lim algorithm.

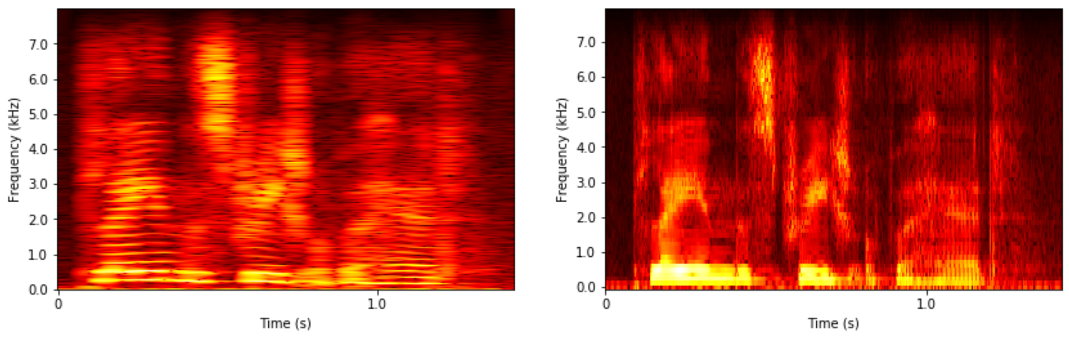


FIGURE 2.2: A spectrogram with large window size (left) is precise in the frequency dimension, but imprecise in the temporal dimension. Vice versa for a spectrogram with small window size (right).

2.3 The Griffin-Lim algorithm

If the phases of a spectrogram are incorrect, its resynthesized audio can contain frequencies that were not present in the original spectrogram. This becomes apparent by taking the spectrogram of the resynthesized audio (an example of this is given in section 5.2).

The Griffin-Lim algorithm [11] converges to a complex spectrogram whose inversion won't change the magnitude spectrogram too much.

Given a magnitude spectrogram $A \in \mathbb{R}^{F \times T}$, we first construct an initial complex spectrogram $X_0 \in \mathbb{C}^{F \times T}$ by imposing random phase values $\Phi \in \mathbb{R}^{F \times T}$ on it.

$$\Phi_{ft} \sim U(-\pi, \pi)$$
$$X_0 = A \odot \cos(\Phi) + iA \odot \sin(\Phi)$$

Next, we iterate $iSTFT$ and $STFT$, all the while maintaining the magnitudes of the original spectrogram.

$$Y_t = STFT(iSTFT(X_t))$$
$$X_{t+1} = A \odot Y_t \oslash |Y_t|$$

20 iterations is enough for noticable improvement, we shall be using 100 iterations in this work. Chapter 5 considers approaches that aim to improve over the Griffin-Lim algorithm.

2.4 The Mel-Scale

The frequencies f that are evaluated for the $STFT$ are by default linearly spaced. However, people do not always perceive linear steps in frequency as equivalent changes in pitch. The mel-scale reflects the human perception of equidistant steps better. Figure 2.3 shows that the mel-scale is linear in the lower frequency range

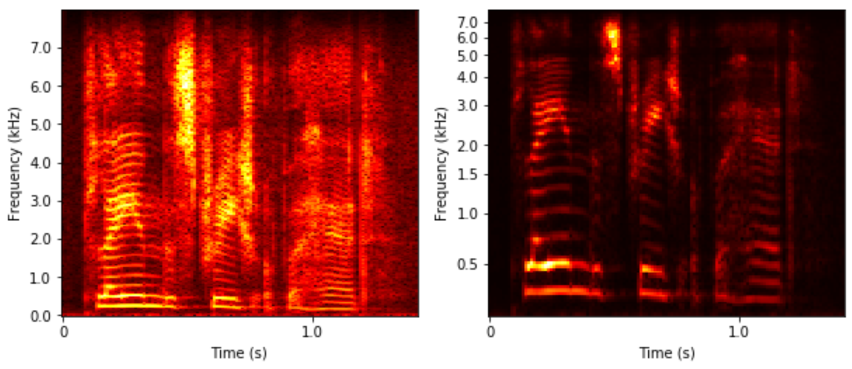


FIGURE 2.3: Linear-scale (left) and Mel-scale (right) spectrograms of speech. The mel scale is linear up to 1 kHz, after which it becomes logarithmic. In the linear spectrum we see overtones as equidistant lines whereas the mel scale shows them as increasingly close together. Audio has been enhanced with pre-emphasis $\alpha = .92$ and spectrograms have been enhanced by $\sqrt[3]{10 \cdot \log(x+1)}$ for better visualisation.

(below 1kHz) and logarithmic in the higher frequency range (above 1kHz)[2].

$$x_m = \begin{cases} x_l & \text{if } x_l < 1000 \\ 1000\log(\frac{x_l}{1000}) + 1000 & \text{if } x_l \geq 1000 \end{cases}$$

As it is derived from perceptual studies with humans, several definitions coexist within the audio processing community. We follow the convention of Librosa[3] and the well-established MATLAB Auditory toolbox of Slaney [51].

Mel-spectrograms are obtained by extracting a linear spectrogram and multiplying it with a matrix — the mel-filterbank, M, shown in figure 2.4. There is no consensus on the proper way to invert this process. In this work, we construct the mel-inversionbank M^* as the normalized transpose of the mel-filterbank. We first normalize the rows to sum to one, and next we normalize the columns to sum to the reciprocal of what the rows of the mel-filterbank sum to.

$$A_m = MA_l$$

Where $A_l \in \mathbb{R}^{F \times T}$ is the linear spectrogram, $A_m \in \mathbb{R}^{N \times F}$ is the mel-spectrogram, $M \in \mathbb{R}^{N \times F}$ is the mel filterbank, F the amount of frequency bins of $STFT$, T timesteps and N the amount of mel-bins. We convert A_m back to linear by multiplying with the mel-inversion bank $M^* \in \mathbb{R}^{N \times F}$:

$$M^*_{ij} = \frac{(M^T)_{ij}}{\sum_k^M (M^T)_{ik} \cdot \sum_l^F (M^T)_{lj}}$$

$$A_l \approx M^* A_m$$

Although the pseudo-inverse $M^* = (M^T M)^{-1} M^T$ can be used as well, figure 2.4 shows that there are some cases where it fails and has extreme and even negative

[2]The relative importance of the high and low frequency domain for speech is of course debatable. It would however be wrong to say that the range between 8kHz and 1kHz is seven times as important as the range below 1kHz.

[3]https://librosa.github.io/librosa/

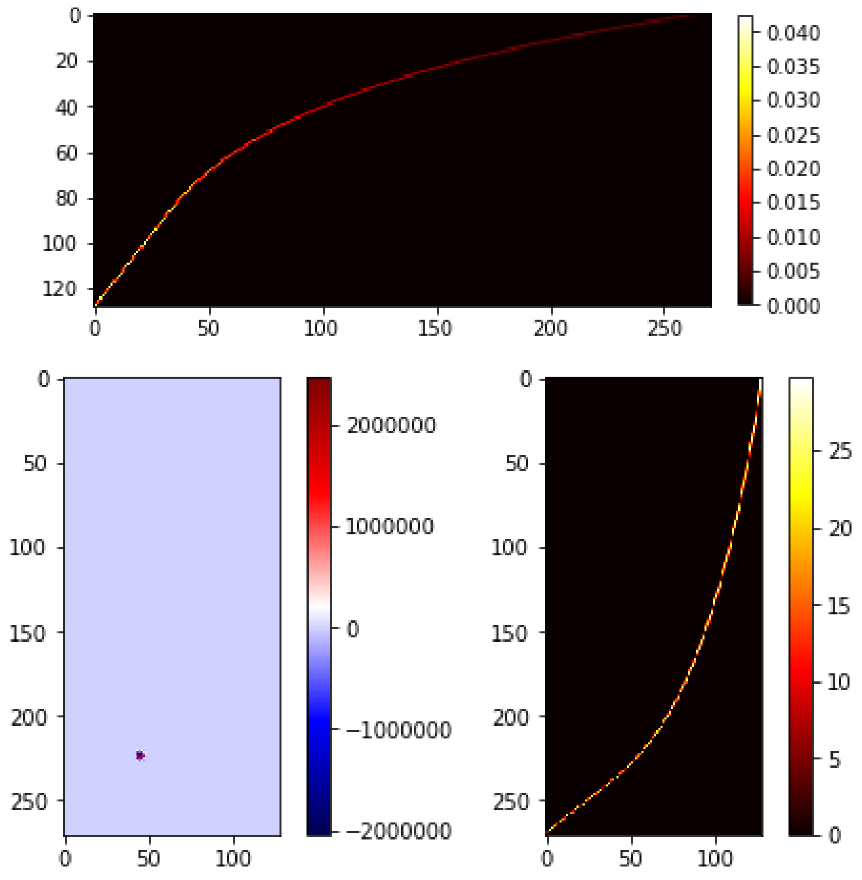


FIGURE 2.4: The mel-filterbank (top), its pseudo-inverse (left) and inversionbank used in this work (right) under $n_{fft} = 540$ and $n_{mels} = 128$. The pseudo-inverse is shown with a different colormap that better shows its problems with value range.

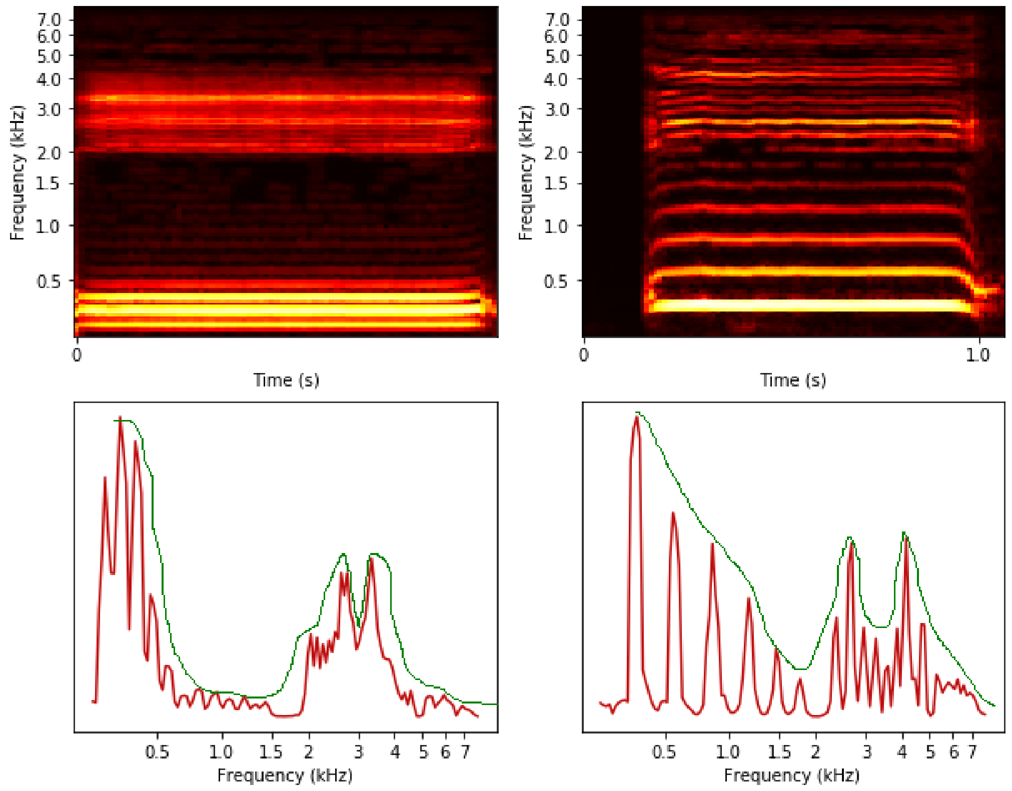


FIGURE 2.5: Mel-spectrogram (top) and spectral envelope (bottom) of the vowel [i] ('ee' in 'sheep') spoken at lower pitch(left) and higher pitch(right). The spectral envelope (in green) is drawn by hand. Energies have been normalized by $\sqrt[3]{10 \cdot \log(x+1)}$.

values, although it should be a non-negative matrix.

Another approach would be to employ a non-negative least squares solver. In chapter 5 we propose to draw mel-spectrograms directly from the audio without an intermediate linear spectrogram and filter- or inversionbank.

2.5 Spectral Envelope

The spectral envelope is a tight bound that wraps the peaks in the spectrogram. The spectral envelope of a phoneme is generally not affected by the pitch with which it is uttered[4]. Figure 2.5 shows the phoneme [i] ('ee' in 'sheep') spoken at low and high pitch. The spectral envelope can be seen by plotting a single column of the spectrogram and connecting the peaks of the spectrum. The pitch is the distance between the peaks of the spectrum, but the height of the peaks is determined by the (relatively unchanging) spectral envelope.

An overtone is a frequency that is an integer multiply of the fundamental frequency. Each of the horizontal lines in top figures of 2.5 (and consequently each of the peaks

[4]Some interplay between pitch and spectral envelope exists, but one of the most succesful vocoders — the STRAIGHT vocoder [22] — is based on the assumption that they are separate, which shows that it is a fairly safe assumption.

in the red curve of the bottom figures) is an overtone. A peak in the spectral envelope — the green curve of figure 2.5 — is a formant.

The fact that a phoneme's spectral envelope does not change under different pitch (or even whisper) is because it is a result of the shape of the mouth and tongue while voicing the phoneme. The formants are the frequencies of resonance of the physical body. Whatever the source, the body will want to resonate under those frequencies. This is why a change in pitch does not impact the spectral envelope and why formants are also present in whisper.

2.6 Mel-Frequency Cepstral Coefficients (MFCCs)

Another common feature used in audio processing are Mel-Frequency Cepstral Coefficients (MFCCs) [33]. They are obtained by taking the log of the energies in a mel-scaled spectrogram and then taking the Discrete Cosine Transform (DCT) over an isolated spectrogram column as if it itself were a signal.

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n+\frac{1}{2})k] \; ; \; k = 0, ..., N-1.$$

The DCT expresses the signal as a weighted sum of k cosines. It is similar to the STFT, but with only real values (and taken over the spectral domain, which moves it up to the 'cepstral' domain). The success of the MFCC feature may be related to its lower frequency components picking up on the shape of the spectral envelope.

2.7 Pre-Emphasis

In speech we typically see that the lower frequencies are of higher energy than the higher frequencies. To counter-balance this and flatten the spectrum, we perform pre-emphasis on the raw audio wave:

$$x_t \leftarrow x_t - \alpha x_{t-1}$$

The factor α is computed by:

$$\alpha = e^{-2\pi F \delta t}$$

Where δt is the duration of a sample and F the frequency above which the spectral slope will increase by 6 decibels per octave. Because we set $F = 214$ Hz at a sample rate of 16 kHz, we find $\alpha = .92$. Other common settings are $\delta t = 2.2676 \cdot 10^{-5}$ (sample rate of 44.1 kHz) and $\alpha = .97$, also implying that $F \approx 214$ Hz. Figure 2.6 shows the effect of pre-emphasis. After new speech is synthesized by the model, the inverse of the pre-emphasis operation is applied over the raw wave to obtain de-emphasised sound.

2.8 Linear Predictive Coding (LPC)

Although originally intended for signal compression, LPC [41] can be used for resynthesis and even Whisper-to-Speech conversion. It requires so little memory that it could be used on 1970's hardware in the text-to-speech synthesis toy Speak&Spell [9]. This was in a day that storing audio of pre-recorded phonemes already required

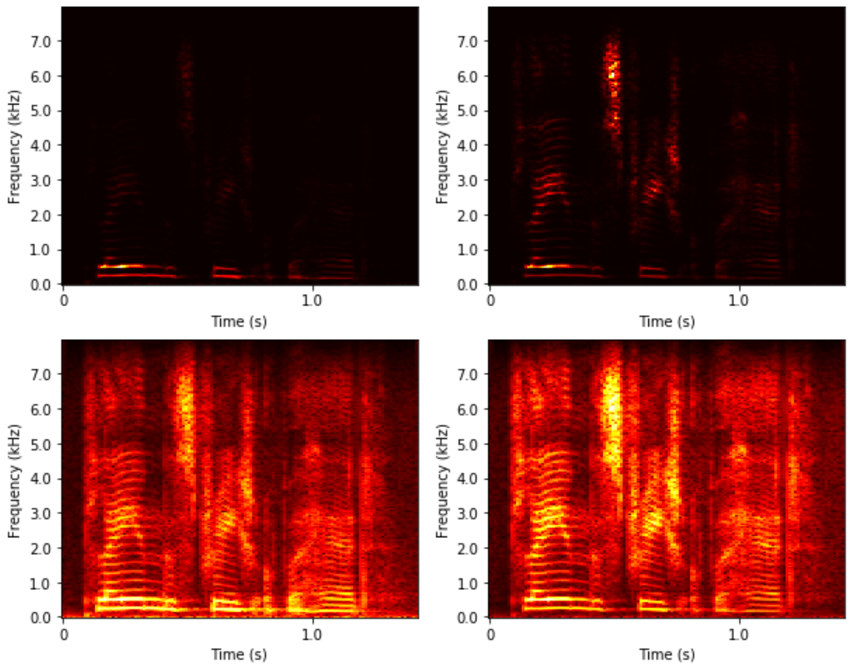


FIGURE 2.6: Spectrograms of speech without pre-emphasis (left) and with pre-emphasis (right) on 16 kHz audio with $\alpha = .92$. Spectrograms on the bottom row have been normalized by $\sqrt[3]{10 \cdot \log(x+1)}$. This normalization strategy primarily boosts low energies, which makes the effect of pre-emphasis less pronounced.

too much memory.

Math. In LPC, we assume that the observed signal **s** is the result of an original signal **u** that has been modified by a filtering mechanism. The filtering mechanism is modelled as auto-regressive, which means that the current value of the signal depends on its own past values. Specifically for LPC, the current value is a *linear* combination of previous values, hence the name 'Linear Predictive Coding'. Mathematically, a source-filter model where **s** is the result of filtering a source signal **u** can be written as:

$$s_n = \sum_{k=1}^{p} a_k s_{n-k} + G \sum_{l=0}^{q} b_l u_{n-l} \tag{2.3}$$

Where G is the 'gain' of the original signal. A signal is assumed to have p poles and $q+1$ zeros, which means a value depends on the past p timesteps of the output signal **s** and on the past q timesteps of the input signal **u**. For LPC we consider an 'all-pole' model, where $q = 0$ and $b_0 = 1$. The coefficients α_k are found by by minimizing the least squares error:

$$E = (s_n - \sum_{k=1}^{p} a_k s_{n-k})^2$$

Setting all factors $\frac{\partial E}{\partial \alpha_k}$ to 0 yields p linear equations with p unknowns.

$$\sum_{k=1}^{p} \alpha_k R_{i-k} = R_i \; ; \; 0 \leq i \leq p$$

Where

$$R_i = \sum_{n=i}^{N-1} s_n s_{n-i}$$

In the case of signal compression, this in itself might not seem so useful. Whereas first we had to represent the signal s by its n values, we now have signal u with n values and p coefficients α_k. However, if we assume u to be a simple signal we can compress it further. If u is gaussian white noise, we can represent it with a simple boolean. If u is a periodic signal, we need only store its frequency f_0. We then need to ship only this single scalar and k factors α_k. On the other end we can resynthesize according to eq 2.3. The periodic source signal $\mathbf{u}$ is usually a pulse train.

Much like spectrograms, it is typical to determine LPC coefficents over a short window of time that has been smoothed by a windowing function. Taking successive windows maps the development of sound over time.

Relation to physiology. The source-filter assumption of LPC match very well with the physiology of speech. The source being either white noise or a periodic signal matches well with speaking either a voiced or unvoiced phoneme. The coefficients α_k determine the filtering mechanism of the mouth and tongue that changes the source sound into recognizable phonemes.

Whisper-to-speech conversion can be achieved by determining the α_k's from the whispered speech. If the phoneme is meant to be unvoiced, the sound can be synthesized from noise. If the phoneme is meant to be voiced, it can be synthesized from a periodic signal. Methods that employ LPC today focus on finding the voiced/unvoiced decision and finden f_0 if the phoneme is meant to be voiced.

Related Work The work of Konno *et al.* [26] learns intended f_0 from recorded whispers. Test subjects were asked to imitate a perfect sine with a whispered vowel. The whisper is represented with Mel-Frequency Cepstral Coefficients (MFCCs) and from this representation condensed vectors are gathered by PCA. From these condensed vectors f_0 prediction is learned by linear regression. When a new whisper arrives its speech can be synthesized using the LPC coefficients and the predicted f_0. This method is elegant in that it does not require paired data; in fact it does not require spoken data at all.

The filter coefficients can sometimes not be 'well-behaved'; taking the average of two sets of filter coefficients will not render the 'average' of the two sounds. And although it doesn't occur in nature, it is easy to by accident construct an unstable filter; one that increases amplitude over time.

2.9 Non-Negative Matrix Factorisation (NMF)

NMF learns to represent a matrix as a linear combination of basis vectors. It can be useful in representing audio as a lower-dimensional vector so that feature matching can be done more efficiently. The constraint that all matrices be non-negative is well-matched with spectrograms, because they themselves are non-negative.

We train a set of basis vectors $B \in \mathbb{R}^{d \times k}$ of which the original spectrogram is approximately a linear combination, weighted by weights $W \in \mathbb{R}^{d \times n}$. The matrix B can be seen as a collection of k basis vectors. The matrix W is then the weights with which the basis vectors must be multiplied in order to approximate $S \in \mathbb{R}^{k \times n}$.

$$S \approx BW$$

NMF can be trained by backpropagating the Mean Squared Error (MSE) between the reconstructed and the original spectrogram. B and W are initialised nonnegative.

$$\mathcal{L} = (S - BW)^2$$

B is updated in the direction of $-\nabla_B \mathcal{L}$ and W is updated in the direction of $-\nabla_W \mathcal{L}$.

Other ways to train NMF include Lee and Seung's Multiplicative Update Rule [27] and Kim and Park's Alternating Non-Negative Least Squares [23].

Related Work The work of Meenakshi *et al.* [32] trains basis vectors from artificial 'colored noises' so that any new whispered noise can be represented as a 5-dimensional vector of weights. This allows for efficient training of a Deep Neural Network (DNN) that can predict whether a sounding whispered phoneme is voiced or unvoiced. Being able to make this distinction is important for LPC synthesis.

2.10 Gaussian Mixture Models (GMM)

GMMs match features of whispered and spoken voice (these can be spectral features, MFCCs, f0 or LPC coefficients) by concatenating pairs of whispered features $\mathbf{x}$ and spoken features $\mathbf{y}$ into a longer vector $\mathbf{z}$. We train a Gaussian Mixture Model that specifies a complex probability distribution over the vectors $\mathbf{z}$. Given a new whispered example $\mathbf{x}$, the problem is reduced to solving

$$\hat{\mathbf{y}} = argmax_{\mathbf{y}} p(Y|X)$$

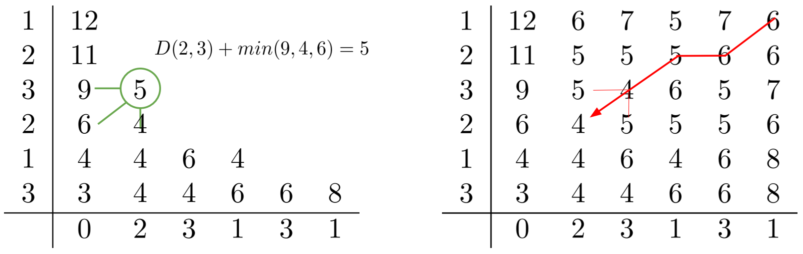
Which can be approximated for example by performing gradient ascent steps, initialized at mixture modes with a high mixing coefficient.

Related Work The work of Sharifzadeh *et al.* [48] focusses on post-laryngectomised speech reconstruction. After enhancement and other pre-processing steps, whispered and spoken features are matched using GMMs.

One problem with the approach is that gradient ascent needs to be performed for each frame. In practice we can get away with only a few gradient steps, not requiring complete convergence. Another problem is that it is required that aligned whisper and speech data is available.

2.11 Dynamic Time Warping (DTW)

Many whisper-to-voice conversion methods assume that paired data is available, meaning that for every frame of a whispered utterance we know which frame of a target spoken utterance we should map to. Humans never speak with reliable timing twice over, certainly not on the level of spectrogram frames, which means that the source and target sequence will be misaligned. Dynamic Time Warping aims [4]



1	12					
2	11					
3	9	5				
2	6	4				
1	4	4	6	4		
3	3	4	4	6	6	8
	0	2	3	1	3	1

1	12	6	7	5	7	6
2	11	5	5	5	6	6
3	9	5	4	6	5	7
2	6	4	5	5	5	6
1	4	4	6	4	6	8
3	3	4	4	6	6	8
	0	2	3	1	3	1

FIGURE 2.7: The Dynamic Time Warping (DTW) algorithm. The first step is to fill the accumulative distance matrix from bottom left to top right and the second step is to backtrack the path that determined the minimal cumulative distance.

to warp two signals closer together.

DTW finds the warping path that minimizes the total distance between two series without breaking continuity. Continuity means that a sample of series **a** can only be matched with a sample of timeseries **b** that comes after the samples in **b** that have already been matched. A sample may be matched more than once, but may not be skipped.

We explain DTW by a simple example, illustrated in figure 2.7. Suppose we have two time series **a** and **b**. They need not be of the same size.

$$\mathbf{a} = \begin{bmatrix}0 & 2 & 3 & 1 & 3 & 1\end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix}3 & 1 & 2 & 3 & 2 & 1\end{bmatrix}$$

The first step of DTW is to construct a cumulative distance matrix M. Every entry (i, j) depends on the entries below it and to its left. We fill the matrix from the bottom left to top right. $D(x, y)$ can be any distance function, but for this example it is simply $|x - y|$.

$$M[i, j] = D(a[i], b[j]) + min(M[i-1, j], M[i, j-1], M[i-1, j-1])$$

Given M, the optimal warping path is found by backtracking through the matrix. This can be done by taking the minimal value of the three options to its left, bottom, or bottom-left, starting at the top-right. If there is a tie it means that both paths leading up to the entry of consideration are equally optimal. The example in figure 2.7 breaks the tie by favoring left over bottom entries, but any heuristic would suffice.

In this example the warped series $\mathbf{a}_w$ and $\mathbf{b}_w$ that match each other best are:

$$\mathbf{a}_w = \begin{bmatrix}0 & \underline{\mathbf{0}} & 2 & 3 & 1 & 3 & 1\end{bmatrix}$$

$$\mathbf{b}_w = \begin{bmatrix}3 & 1 & 2 & 3 & 2 & \underline{\mathbf{2}} & 1\end{bmatrix}$$

Counter-intuitively, both series are warped and can only be longer or equal than their original.

This simple example considers sequences of scalar values, but DTW can also be applied to sequences of vectors — such as spectrograms. The only change is in the distance function D, which can be any measure of similarity. For vectors, we can use the Euclidean distance.

DTW on the waveform can change the nature of the sound considerably. Additionally, the sequence length of even a small piece of audio is many thousands, which incurs the need for a very large matrix M. For this reason DTW is typically performed on spectrograms or MFCCs — where a second is a sequence of a few dozen vectors rather than thousands.

DTW works when the sequences are already somewhat alike. Although a person might experience a whispered and spoken phoneme as very similar, this is not the case for their waveforms or spectrograms.

The work of Sharifzadeh *et al.* [49] artfully combats the problem by artificially whisper<u>is</u>ing spoken voice recordings. This is done by removing pitch of voiced phonemes. The whisper<u>is</u>ed audio is naturally perfectly aligned with the spoken audio it is derived of. DTW can be more succesfully applied between the whisper<u>is</u>ed and whispered voice to obtain the warping path. Applying the warping path to the spoken voice yields a better pairing between whispered and spoken voice. In this work we follow the stategy of McLoughlin *et al.* [31], who perform DTW on 25-order MFCCs. MFCCs are more alike in the whisper- and speech domains than spectrograms are, because they partly encode the spectral envelope.

Cuturi *et al.* [6] formulate soft-DTW, a differentiable version of DTW. It can be used as a loss function and backpropagated to a neural network. When the neural network is close to convergence it is likely that the converted item and the target are similar, which helps DTW. If the cold start problem can be overcome, this is a tremendous benefit over performing DTW offline and regarding its outcome as ground-truth. However, the complexity of the loss metric is $\mathcal{O}(m \times n)$ for sequences of length m and n. For the lengths of sequences we typically encounter in this work (about 500 spectrogram columns), this becomes prohibitively expensive.

In the next chapter we will find that the methods that don't require DTW perform better than those that do. Indeed, one contribution of this thesis is that DTW can be circumvented by using an adversarial learning paradigm rather than a supervised one.

Chapter 3

Whisper-to-Speech on the Spectrogram Level

In this chapter we perform Whisper-to-Speech (W2S) conversion. We constrain ourselves to mel-spectrograms. At test time, we convert them to audio by multiplying with the mel-inversionbank (section 2.4) and performing 100 Griffin-Lim iterations starting with random phases drawn from $\phi \sim U(-\pi, \pi)$.

Section 3.1 discusses the use of an LSTM for sequence-to-sequence translation (W2S-L). In section 3.2 we add RBM pretraining to that paradigm (W2S-R). In section 3.3 we propose to train a CNN with WGAN loss (W2S-C). A benefit of W2S-C is that it doesn't require DTW.

Dataset. We train the models of this thesis on the WTIMIT [29] dataset (consisting of about 15 hrs of parallel data). All speech is in English. Half the speakers are from the United States (US) and half the speakers are from Singapore (SG). We hold out the data of 4 speakers to constitute the test set (US and SG male and female). We assign 85% of the remainder to the training set and 15% to the validation set. The data of each speaker contributes to each partition by the same ratio. For most runs we define 'training epochs', which are loops over a subset of the training set (because a loop over a full epoch would take too long to grant insight in the learning process). The full training set contains about 19k items.

We use pre-emphasis with $\alpha = .92$ on the audio (as discussed in section 2.7). 128-dimensional mel-scaled spectrograms taken with a window size of 540 samples and hop size of 180 (for a samplerate of 16kHz this amounts to roughly 36ms and 12ms). Spectrograms are normalized by $\sqrt[3]{10 \cdot \log(x+1)}$, which maps to a non-negative range where most values are below 1.

W2S-L and W2S-R require that input and target sequences are aligned. We employ the DTW strategy of McLoughlin *et al.* [31] and align by 25-order MFCCs. Figure 3.1 shows one such alignment. Although the columns now match, the source and target audio can suffer some distortion.

3.1 Sequence-to-Sequence Translation with LSTMs (W2S-L)

Long Short-Term Memory [16] networks are a flavour of Recurrent Neural Networks (RNNs). They are commonly used in an autoregressive setting; the network is tasked with extrapolating the input sequence. We denote this as $[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3] \rightarrow [\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4]$. When predicting the t-th item of the target sequence the model has access to all items up to and including the t-th item of the source sequence. However, LSTMs can also be used for sequence-to-sequence translation by $[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3] \rightarrow [\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3]$. In predicting $\mathbf{y}_t$, it is important to have information about $\mathbf{y}_{t-1}$. Although $\mathbf{y}_{t-1}$ is not

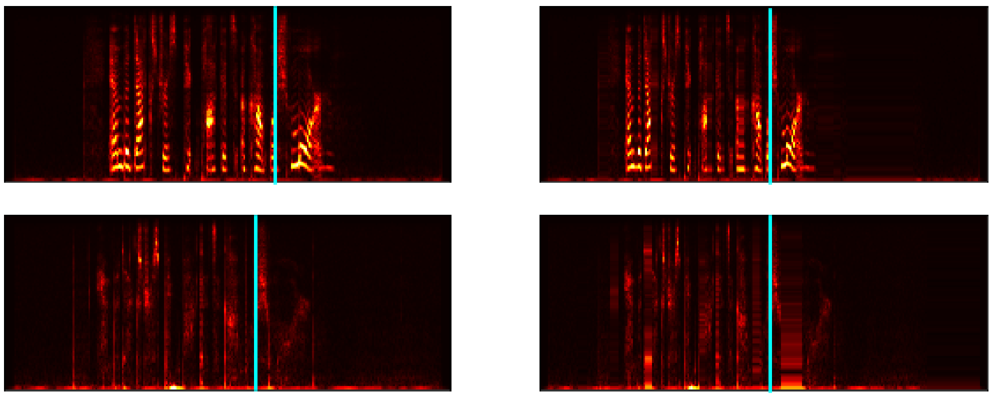


FIGURE 3.1: Dynamic Time Warping for temporal alignment. Blue markers are drawn by hand. Audio can be found in 'drive/DTW/...' and are based on wTIMIT data with the tag 's123u051'

directly an input to the network, the information is implicitly passed on through the hidden- and cell-states of the LSTM (in fact, for a single layer LSTM the output *is* the hidden state).

An LSTM cell can be defined by the following operations:

$$\mathbf{i}_t = \sigma(W_{xi}\mathbf{x}_{t-1} + W_{hi}\mathbf{h}_{t-1} + \mathbf{b}_i)$$

$$\mathbf{f}_t = \sigma(W_{xf}\mathbf{x}_{t-1} + W_{hf}\mathbf{h}_{t-1} + \mathbf{b}_f)$$

$$\mathbf{o}_t = \sigma(W_{xo}\mathbf{x}_{t-1} + W_{ho}\mathbf{h}_{t-1} + \mathbf{b}_o)$$

$$\widehat{\mathbf{c}_t} = tanh(W_{xc}\mathbf{x}_{t-1} + W_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c)$$

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \widehat{\mathbf{c}_t}$$

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Where $\mathbf{x}_t$ is the input at time t and $\mathbf{h}_t$ and $\mathbf{c}_t$ the hidden- and cell-states that are passed on to the next timepoint. $\sigma(\cdot)$ is the logistic sigmoid function. The output of an LSTM cell is also $\mathbf{h}_t$. LSTM-cells can be stacked, in which case the input of the second layer $\mathbf{x}_t^{l=2}$ is the output (or hidden state) of the first layer $\mathbf{h}_t^{l=1}$. Each layer will have its own hidden- and cell-state.

A notable occurrence in training LSTMs on long sequences is that of exploding gradients. When backpropagating $\frac{\partial \mathcal{L}}{\partial W}$ we encounter the term $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial W}$. $\mathbf{h}_t$ depends on W directly, but also indirectly through $\mathbf{h}_{t-1}$, so we must also take $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}}\frac{\partial \mathbf{h}_{t-1}}{\partial W}$ into account. $\mathbf{h}_{t-1}$ in turn depends on $\mathbf{h}_{t-2}$, and so on until the initial hidden state. When sequences are very long, summing these gradients can cause updates that are very large, and these can destabilise learning. Truncated Backprop Through Time (TBPTT) **??** proposes to only update the weights by taking into account terms $\frac{\partial \mathbf{h}_{t-k}}{\partial W}$ where k is smaller than some threshold. In this work we prevent exploding gradients by only showing the network sequences of length 40, which is small enough to prevent exploding gradients, but large enough to model speech on a relevant time scale (about half a second).

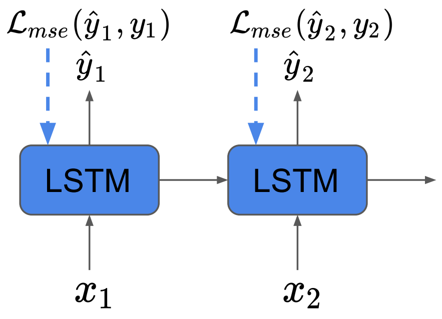

$\mathcal{L}_{mse}(\hat{y}_1, y_1)$
$\mathcal{L}_{mse}(\hat{y}_2, y_2)$
$\hat{y}_1$
$\hat{y}_2$
LSTM
LSTM
x_1
x_2

FIGURE 3.2: Training Schematic for the LSTM model. Dashed arrows represent backpropagation of the loss to the parameters of the model with its color.

We train W2S-L to minimise the Mean Squared Error (MSE) between aligned input sequences $X = [\mathbf{x}_1, ..., \mathbf{x}_T]$ and $Y = [\mathbf{y}_1, ..., \mathbf{y}_T]$.

$$\mathcal{L} = MSE(X, Y) = MSE([\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_T], [\mathbf{y}_1, ..., \mathbf{y}_T])$$

As a unified expression for MSE, we write:

$$MSE(X, Y) = \frac{1}{D} \sum_{i}^{D} (x_i^* - y_i^*)^2$$

Where $\mathbf{x}^* \in \mathbb{R}^D$ is a multidimensional vector, matrix or tensor X vectorised into a single vector.

Network Architecture. The input- and output dimensionality of W2S-L is $\mathbf{x}_t, \hat{\mathbf{y}}_t \in \mathbb{R}^{128}$. The LSTM consists of 3 stacked LSTM cells with $\mathbf{h} \in \mathbb{R}^{256}$ appended with a linear layer that projects back to 128 dimensions. The final layer has a softplus activation.

Training Procedure. The LSTM-based converter is trained with the ADAM [24] optimizer with a learning rate of 10^{-4} and weight decay of 10^{-4} for 100 training epochs (each a quarter of a true epoch). Other parameters of the ADAM optimizer are left at default settings ($\beta_1 = .9$, $\beta_2 = .999$). Its convergence is shown in figure 3.3.

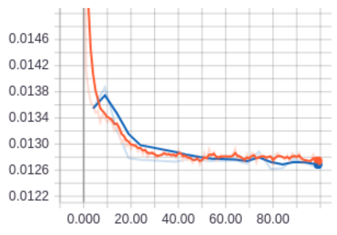


FIGURE 3.3: Convergence of the Mean Squared Error loss of the LSTM model. Training loss in orange, validation loss in blue.

Results and Discussion. Figure 3.4 shows an output of W2S-L. It appears that the

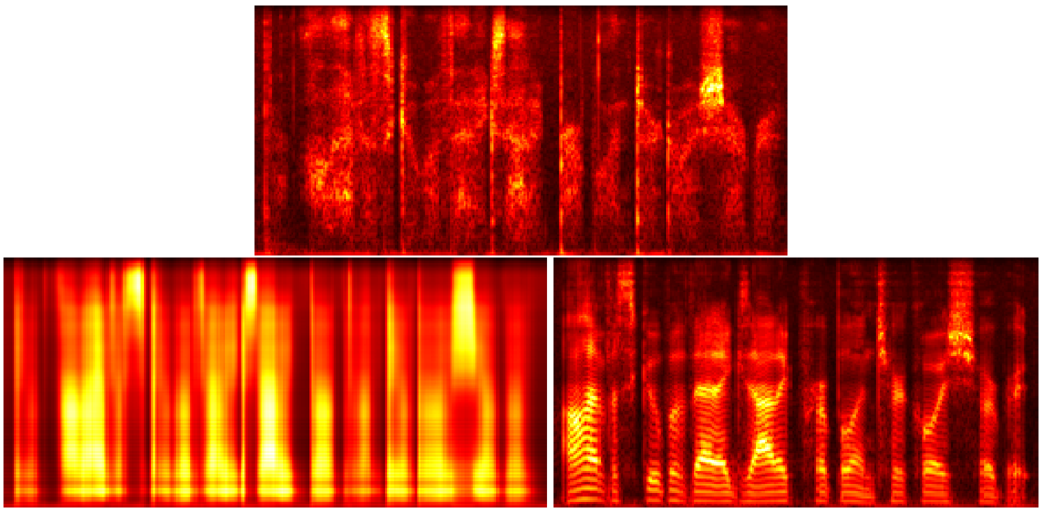


FIGURE 3.4: Whisper (top), converted by W2S-L (left) and ground-truth speech (right). The excerpt tag in the wTIMIT dataset is 's107u348'. Audio can be found on 'drive/W2S-L/...'

model knows to output a fundamental and its first overtone, indicating that W2S-L is starting to synthesize a pitch, but it is quite monotonous. It is likely synthesizing the 'mean pitch' of the data. When listening to the audio, the pitch gets lost in the overall noise caused by the blurriness of the spectrogram. Words are unintelligible.

MSE can drive a network to output smooth spectrograms. If the network can't decide between a pitch of 100 Hz (characterised by overtones of 100, 200, 300, etc...) and a pitch of 110 Hz (characterised by overtones of 110, 220, 330, etc...) the MSE is minimised by outputting a mixture of the two, which is a blurry spectrogram.

3.2 RBM pretraining for LSTM (W2S-R)

In order to prevent smooth spectrograms we employ RBM pretraining and optimize a different objective than MSE.

Restricted Boltzmann Machines. RBMs [15] can be trained as feature extractors in an unsupervised way. The goal of an RBM is to maximize the loglikelihood of the data.

A standard RBM assumes that the input vector **x** and the hidden vector **h** are binary vectors. Conditional probabilities of vector values are:

$$p(\mathbf{h}|\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b})$$

$$p(\mathbf{x}|\mathbf{h}) = \sigma(W^T\mathbf{h} + \mathbf{a})$$

The probability of an $(\mathbf{x}, \mathbf{h})$-pair is proportional to the free energy function E:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{a}^T\mathbf{x} - \mathbf{b}^T\mathbf{h} - \mathbf{x}^T W\mathbf{h}$$

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{x},\mathbf{h})} \tag{3.1}$$

Where **a**, **b** and W are learnable parameters (from here on out referred to as θ). **h** is a latent vector and Z is a the normalizing constant — sometimes called the "partition function" — obtained by summing over all possible $(\mathbf{x}, \mathbf{h})$-pairs:

$$Z = \sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})} \tag{3.2}$$

When properly trained, an RBM can be used to map an input vector $\mathbf{x}$ to a hidden vector $\mathbf{h}$ that encodes in it meaningful features of the data. Additionally, an RBM can also map a hidden vector $\mathbf{h}$ to a vector $\mathbf{x}$ in the original feature space.

Intuitively, the logprobablility of the data can be maximized by updating the parameters to decrease the energy function for points in the dataset and increase it for other points (the latter limits the growth the partition function caused by the first update). The update rule of the RBM reflects this.

$$\frac{\partial \log p(X_{data})}{\partial \theta} = \mathbb{E}_{data}[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \theta}] - \mathbb{E}_{not-data}[\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \theta}]$$

An unbiased sample of the first term can easily be obtained by computing $p(\mathbf{h}|\mathbf{x})$ from a known datapoint $\mathbf{x}$ and sampling $\mathbf{h}$ from it. The second term is harder to obtain. Since no $(\mathbf{x}, \mathbf{h})$-pair can be drawn 'from nothing' (we have no explicit expression of the joint distribution), it would have to be obtained by performing alternating Gibbs-sampling steps for a long time. Instead, Hinton [15] proposes to use a reconstruction $\mathbf{x}_{recon}$ from $p(\mathbf{x}|\mathbf{h})$ where $\mathbf{h}$ is the hidden state that has been used for the first term. This approach is an approximation of the gradient of the loglikelihood and is actually closer to the gradient of another objective called the Contrastive Divergence. Consequently, the algorithm is often referred to as the Contrastive Divergence algorithm (interestingly, Sutskever *et al.* [53] note that it doens't approximate the gradient of the CD either). Despite not being the true gradient of the loglikelihood, the approximation works well in practice. It is customary to sample from $p(\mathbf{h}|\mathbf{x})$ only once, and use for other values only probabilities, not samples [14]. Concretely:

$$\begin{aligned}
\mathbf{h}_{data} &\sim p(\mathbf{h}|\mathbf{x}_{data}) \\
\mathbf{x}_{recon} &= p(\mathbf{x}|\mathbf{h}_{data}) \\
\mathbf{h}_{recon} &= p(\mathbf{h}|\mathbf{x}_{recon})
\end{aligned}$$

$$\frac{\partial \log p(\mathbf{x}_{data})}{\partial \theta} \approx \frac{\partial E(\mathbf{x}_{data}, \mathbf{h}_{data})}{\partial \theta} - \frac{\partial E(\mathbf{x}_{recon}, \mathbf{h}_{recon})}{\partial \theta}$$

$$\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \mathbf{a}} = -\mathbf{x}\ ; \ \frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \mathbf{b}} = -\mathbf{h}\ ; \ \frac{\partial E(\mathbf{x},\mathbf{h})}{\partial W} = -\mathbf{x}\mathbf{h}^T$$

Parameters are updated along gradients of mini-batches (Stochastic Gradient Descent), often with a momentum term for stability.

A logistic unit is a poor representation for spectrograms (a binary one is certainly so). It is more fitting to replace the binary visible units by linear units with independent Gaussian noise. The energy function then becomes:

$$E(\mathbf{x},\mathbf{h}) = ||(\mathbf{x} - \mathbf{a}) \oslash \sigma||^2 - \mathbf{b}^T\mathbf{h} - (\mathbf{x} \oslash \sigma)^T W\mathbf{h}$$

Where σ is a vector of standard deviations of the independent noise, $||...||$ is the euclidian norm and $\oslash$ is elementwise division. Although it is possible to learn the variance of the noise per input dimension it is more practical to normalise each component to have zero mean and unit variance. The energy function and its gradient then become simpler.

$$E(\mathbf{x},\mathbf{h}) = ||\mathbf{x}-\mathbf{a}||^2 - \mathbf{b}^T\mathbf{h} - \mathbf{x}^T W\mathbf{h}$$
$$\frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \mathbf{a}} = \mathbf{a}-\mathbf{x}\ ;\ \frac{\partial E(\mathbf{x},\mathbf{h})}{\partial \mathbf{b}} = -\mathbf{h}\ ;\ \frac{\partial E(\mathbf{x},\mathbf{h})}{\partial W} = -\mathbf{x}\mathbf{h}^T$$

Gaussian-Bernoulli RBMs typically require a lower learning rate to show stable learning than their standard Bernoulli-Bernoulli counterpart and consequently train more slowly.

To check for overfitting, one would like to keep track of the loglikelihood of data in the training set and the validation set. If the model is overfitting, the loglikelihood of the training set is much higher. However, computing the loglikelihood (Eq 3.1) requires computing the partition function (Eq 3.2), which is intractible. If we keep track of their ratio, however, the partition function is cancelled out. To check for overfitting, we keep track of the free energy ratio:

$$r = \frac{E(\mathbf{x}_{val}, p(\mathbf{h}|\mathbf{x}_{val}))}{E(\mathbf{x}_{train}, p(\mathbf{h}|\mathbf{x}_{train}))}$$

The model is overfitting if $r \gg 1$.

W2S-R. In the first training stage of W2S-R, we train two separate RBMs — one trained on whispered and one on spoken data. Spectrograms are normalised such that each feature is distributed according to $\mathcal{N}(0,1)$.

In the second training stage, we train an LSTM to match the sequence of learned features $[\mathbf{h}_{x1}, ..., \mathbf{h}_{xT}]$ to $[\mathbf{h}_{y1}, ..., \mathbf{h}_{yT}]$ where $\mathbf{h}_x, \mathbf{h}_y \in \{0,1\}\mathbb{R}^{128}$ (which are thus **binary** vectors). The loss function of the LSTM is Binary Cross-Entropy (BCE) as opposed to MSE:

$$\mathcal{L} = BCE(H_{\widehat{Y}}, H_Y) = BCE([\widehat{\mathbf{h}}_{y1}, ..., \widehat{\mathbf{h}}_{yT}], [\mathbf{h}_{y1}, ..., \mathbf{h}_{yT}])$$
$$BCE(X,Y) = -\frac{1}{D}\sum_i^D y_i^* \log x_i^* + (1-y_i^*)\log(1-x_i^*)$$

Where $\mathbf{x}^* \in [0,1], \mathbb{R}^D$ and $\mathbf{y}^* \in \{0,1\}, \mathbb{R}^D$ are vectorised forms of multidimensional vectors, matrices or tensors X and Y.

The bidirectional nature of RBMs means that at test time the conversion $X \rightarrow H_X \rightarrow H_{\widehat{Y}} \rightarrow \widehat{Y}$ can be made. A training schematic is shown in figure 3.5. Because the targets are binary we can also keep track of accuracy during training to check for convergence and overfitting.

RBMs for Crisper Spectrograms. The update rule for RBMs promotes the joint logprobability of $(\mathbf{x}_{data}, \mathbf{h}_{data})$ pairs where $\mathbf{h}_{data}$ is binary, while suppressing the joint logprobability of $(\mathbf{x}_{recon}, \mathbf{h}_{recon})$ pairs where $\mathbf{h}_{recon}$ is non-binary. This means that the RBM is trained to have vectors $\mathbf{x}$ associated with binary hidden vectors $\mathbf{h}$ deemed as realistic and vectors $\mathbf{x}$ with continuous hidden vectors $\mathbf{h}$ as unrealistic. Discretizing a hidden vector $\mathbf{h}_{continuous} \rightarrow \mathbf{h}_{binary}$ and computing $p(\mathbf{x}|\mathbf{h}_{binary})$ thus constitutes the closest realistic output for the predicted $\mathbf{h}_{continuous}$, whereas computing

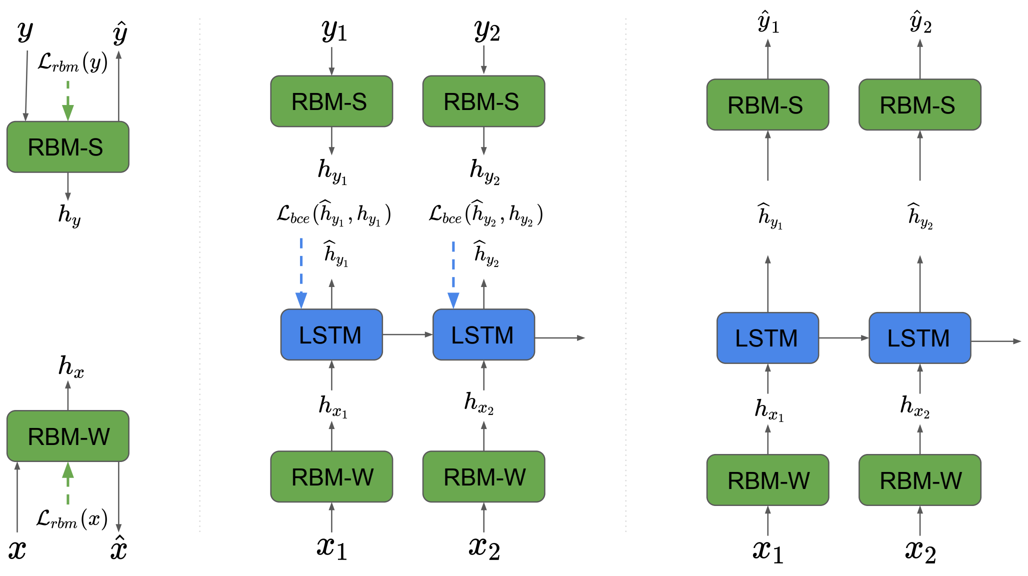


FIGURE 3.5: Training Schematic for W2S-R. During the first training stage (left) the RBMs are trained in an unsupervised manner. The second training stage (center) shows training of the feature matching LSTM. At test time (right) the spoken RBM is used in reverse to obtain spectrograms from the predicted hidden features.

$p(\mathbf{x}|\mathbf{h}_{continuous})$ would constitue a mixture of all likely realistic outputs, which — as we've seen for the case of MSE — is itself not a realistic output. We hope discritizing the hidden vector this will help prevent smooth spectrograms.

Related Work. RBM-pretraining is also used by McLoughlin *et al.* [31], except in their work the spectral envelope is matched (not the spectrogram), an additional model is employed to estimate pitch and synthesis is done using the STRAIGHT vocoder [22]. Their feature matching network is a fully connected feed-forward neural network, which processes each frame independently from its neighbours. In order to give the network some sense of context, they employ temporal delta-features.

Network Architecture. We set the dimensionality of the hidden feature vectors $\mathbf{h} \in \{0,1\}, \mathbb{R}^{128}$. Though this may not seem like much of an information bottleneck, the binarity of the hidden values restricts the model's expressiveness enough for it to have to learn meaningful features in order to achieve its objective.

The feature matching network of W2S-R has the same architecture as W2S-L, except that the final activation function is *tanh* (as opposed to softplus). Before input to the model and output to the loss function the domain is moved from $[0,1]$ to $[-1,1]$ and back.

Training Procedure. In the first training stage of W2S-R we train the RBMs with SGD (learning rate 10^{-4}) for a total of 400 training epochs (each a quarter of a true epoch). In the first 20 training epochs we employ a momentum of .5, and in the remaining 380 epochs we employ a momentum of .9. Convergence of the RBM pretraining phase is shown in figure 3.6. The free energy ratios are .99 throughout training, which confirms that the models are not overfitting.

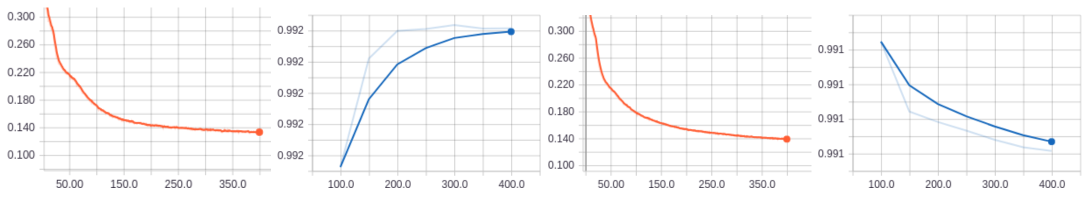


FIGURE 3.6: Convergence of the RBM reconstruction loss (orange) and the free energy ratio (blue) for the whispered domain (left) and the spoken domain (right).

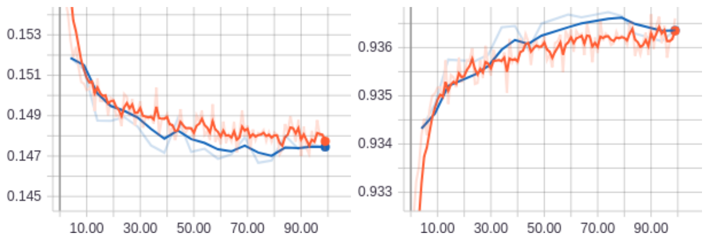


FIGURE 3.7: Convergence of the Binary Cross Entropy loss (left) and accuracy (right) of W2S-R.

During the second training stage of W2S-R, the RBM weights are frozen. The LSTM of W2S-R is trained under the same settings as W2S-L, except for the objective to minimize, which is Binary Cross Entropy. Its convergence is shown in figure 3.7.

Results and Discussion. Figure 3.8 shows that both the RBMs for both the whispered and the spoken domain reconstruct their input well, indicating that they have learned valuable features of their domains. In figure 3.9 we see a conversion of the full W2S-R model, showing that discretizing the RBM feature vector has a positive effect on crispness. However, the audio is not intelligible and the audio has a non-human character. The general performance of W2S-R improves slightly over W2S-L.

3.3 CNN with WGAN training (W2S-C)

As a third alternative we propose a Convolutional Neural Net (CNN) trained with an adversarial loss function — specifically, the loss function of the Wasserstein Generative Adversarial Network (WGAN) [3, 12].

Rather than computing a distance between a prediction and a supervised target, we train an additional, adversarial network to gauge how realistic the output. In doing so, we circumvent requiring DTW-aligned ground truth data[1], which figure 3.1 shows to be a source of data-contamination.

We first introduce the loss function(s) of the WGAN paradigm. We do this to familiarize the reader with the concept of a 'critic' network before we define its architecture.

[1]In fact, the WGAN model doesn't even require that a whispered or spoken item has a counterpart in the other domain. However, it does help to ensure that the distributions differ only in speech characteristics, not content

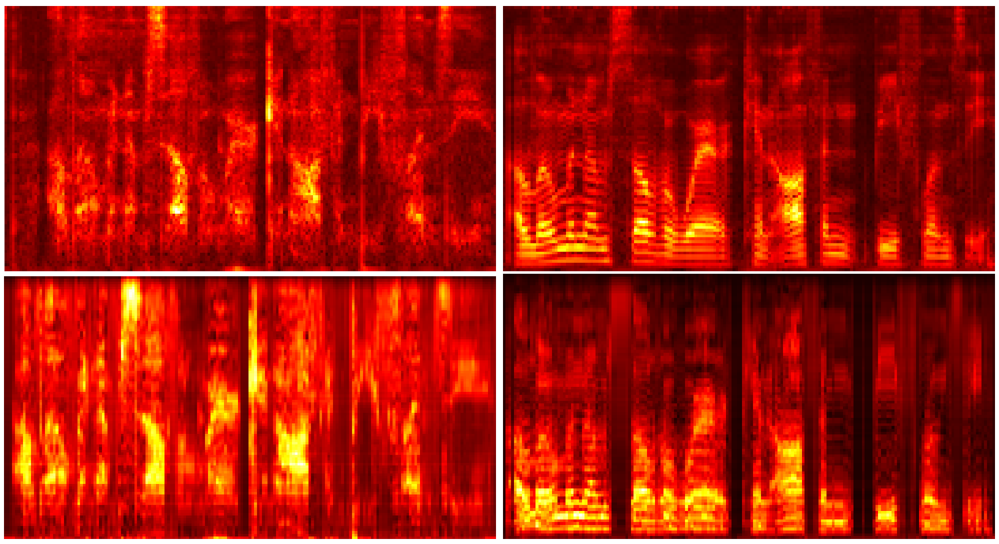


FIGURE 3.8: Original whisper (top left), original speech (top right), reconstructed whisper by RBM-W (bottom left) and reconstructed speech by RBM-S (bottom right). Audio can be found on 'drive/RBM-S/...' and 'drive/RBM-W/...'.

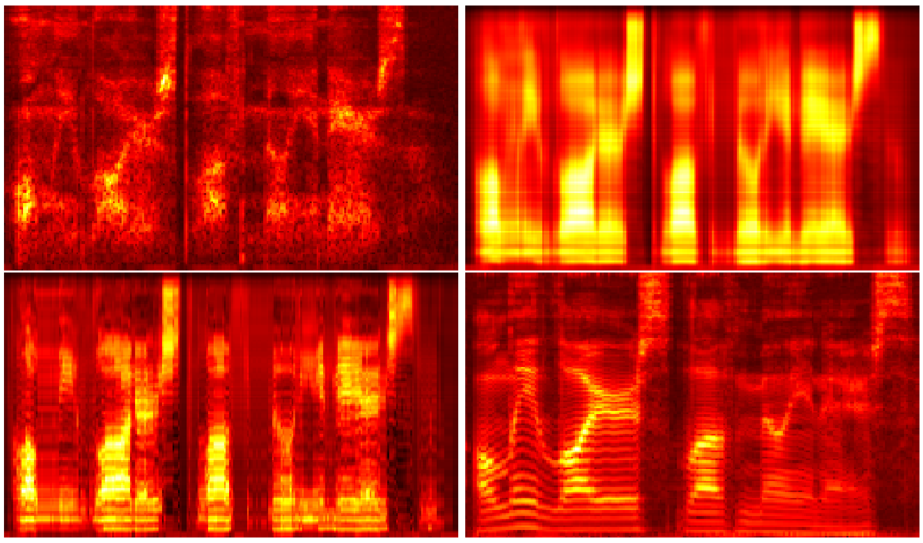


FIGURE 3.9: Whisper (top left), W2S-R with continuous hidden features (top right), W2S-R with discretized hidden features (bottom left) and ground-truth speech (bottom right). The excerpt tag in the wTIMIT dataset is 's107u025'. Audio can be found on 'drive/W2S-R/...'

The WGAN Paradigm. We train an adversarial network — referred to as the 'critic'[2] — to estimate a measure of distance — or similarity — between the distributions of real and converted speech. The converter[3] is trained to minimize this distance.

The critic estimates the Wasserstein distance — also known as the Earth Mover Distance (EMD).

$$W(p_a, p_b) = inf_{\gamma \in \pi(p_a, p_b)} \mathbb{E}_{(a,b)\sim\gamma} ||a - b|| \tag{3.3}$$

Where $\pi(p_a, p_b)$ is the set of all joint distributions $\gamma(a, b)$ whose marginals are p_a and p_b.

$$\int_b \gamma(a, b) db = p_a(a)$$

$$\int_a \gamma(a, b) da = p_b(b)$$

Intuitively, $\gamma(a, b)$ is the amount of probability mass that must be moved from a to b to transform p_a into p_b. The Wasserstein distance is the cost of the optimal $\gamma(a, b)$. Finding the Wasserstein distance is hard. However, by the Rubinstein-Kantorovich duality [58] we can instead solve an equivalent problem:

$$W(p_a, p_b) = sup_{||g||_L \leq k} [\mathbb{E}_{a\sim p_a} g(a) - \mathbb{E}_{b\sim p_b} g(b)] \tag{3.4}$$

The condition is that the function g is k-Lipschitz continuous. A function $g : A \to B$ is k-Lipschitz continuous if for distance functions d_A and d_B on spaces A and B it holds that:

$$d_B(g(a_1), g(a_2)) \leq K \cdot d_A(a_1, a_2) \; ; \; \forall a_1, a_2 \in X$$

Put differently, the slope of a k-Lipschitz continuous function never exceeds k.

We find the function g by implementing it as a neural network (the critic) parameterized by ϕ. Lipschitz-continuity can be achieved by adding a loss term that penalises the gradient norm for deviating from k [12]. We set $k = 1$ and weight the gradient penalty term by λ.

We draw a minibatch of whispered spectograms and convert it to speech using a converter network f parameterized by θ:

$$\widehat{Y} = f_\theta(X)$$

The objective of the critic is the supremum of equation 3.4 augmented with a loss term for Lipschitz continuity. The Lipschitz constraint is enforced at the location of an item Z, which is a linear interpolation of a 'fake' speech spectrogram $\widehat{Y}$ and a real speech spectrogram Y:

$$Z = interpolate(\widehat{Y}, Y)$$

$$\mathcal{L}_{cr} = g_\phi(\widehat{Y}) - g_\phi(Y) + \lambda(||\nabla_\phi g_\phi(Z)|| - 1)^2$$

$$\phi \leftarrow \nabla_\phi \mathcal{L}_{cr}$$

The objective of the converter f_θ is to minimize the estimated Wasserstein distance:

$$\mathcal{L}_{adv} = g_\phi(Y) - g_\phi(f_\theta(X))$$

[2]We use the term 'critic' instead of 'discriminator', because the network does not classify

[3]We use the term 'converter' instead of 'generator', because the input is not random noise, but has important meaning

$$\theta \leftarrow \nabla_\theta \mathcal{L}_{adv}$$

Note the sign flip between $\mathcal{L}_{cr}$ and $\mathcal{L}_{adv}$; the critic minimizes the negative estimated Wasserstein distance (because it wants to maximize it). The converter minimizes the positive estimated Wasserstein distance. Although not a requirement, we draw matching pairs of converted and real spectrograms, but they are not warped by DTW.

Whereas in classic GANs [10] the gradient becomes small if the discriminator is more powerful than the generator, the gradient of the critic is always one or near one. This helps the converter to keep learning. It also means that we do not need to find a delicate balance between the two networks, which is an oft-experienced nuisance in working with GANs. In the case of WGANs, the critic may — or even *should* — be stronger than the converter. A stronger critic (more parameters or trained for more steps) also implies that learning will be slower. However, it may be faster in the long run if we don't have to spend time tuning hyperparameters to craft the delicate balance between the adversarials.

Though it is theoretically only necessary to provide an upper bound for $||\nabla_\phi g(Z)||$, we follow the recommendation of [12] and define a two-sided constraint. Interestingly, [52] find that a one-sided constraint converges faster. Wei *et al.* [59] promote enforcing the Lipschitz constraint on an additional location Z^* that lies closer to Y by applying dropout to the critic.

For the optimal transport plan, the computed quantity of equations 3.3 and 3.4 is the same. If the infimum in equation 3.3 is not found, the transportation plan is suboptimal and the estimate will be higher than the true Wasserstein distance. If the supremum of 3.4 is not found, the estimate will be lower than the true Wasserstein distance.

Reconstruction Loss. Theoretically, a converter network could learn to completely ignore the input and 'hallucinate' spectrograms that the critic deems realistic. In order to teach the converter that the output should be related to the input, we introduce a reconstruction loss. The reconstruction loss is the MSE between original whisper X and $\widehat{X}$, obtained from converting fake speech $\widehat{Y} = f_\theta(X)$ back to whisper with a speech-to-whisper (S2W) network f^*_η. Both f_θ and f^*_η are trained to minimize the reconstruction loss. A training schematic is shown in figure 3.10.

$$\widehat{X} = f^*_\eta(\widehat{Y}) = f^*_\eta(f_\theta(X))$$

$$\mathcal{L}_{recon} = \alpha \cdot MSE(X, \widehat{X})$$

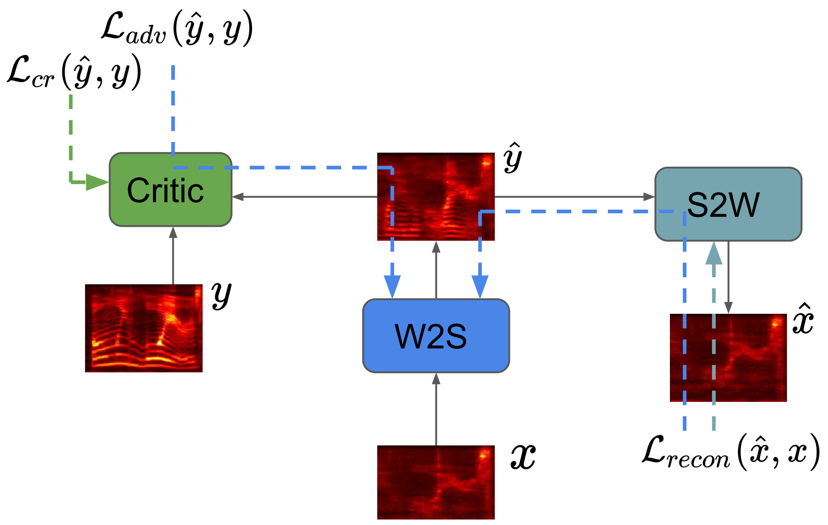
$$\theta \leftarrow \nabla_\theta \mathcal{L}_{recon}$$

$$\eta \leftarrow \nabla_\eta \mathcal{L}_{recon}$$

The reconstruction loss is weighed by a term α in such that:

$$||\nabla_\theta \mathcal{L}_{adv}|| \approx ||\nabla_\theta \mathcal{L}_{recon}||$$

Although gradient magnitude is a crude proxy for how much learning is taking place, it is an easy thing to monitor. It can help verify whether one loss term is not completely overpowering the other.



$\mathcal{L}_{adv}(\hat{y}, y)$
$\mathcal{L}_{cr}(\hat{y}, y)$
Critic
$\hat{y}$
S2W
y
W2S
$\hat{x}$
x
$\mathcal{L}_{recon}(\hat{x}, x)$

FIGURE 3.10: Training schematic for W2S-C. The dashed blue lines running through other networks indicate that the loss is backpropagated through those networks, but the updates are only applied to the parameters of the network with the same color.

Related Work. The image-to-image translation model pix2pix [18] has two loss terms; L1-distance between the pixels of a converted and a target image and a loss term computed by a discriminator that evaluates small patches of converted and real images. Our work is different in that we use a self-supervised loss rather than a supervised loss, and therefore don't need an aligned target or any paired data. Our critic evaluates the full output rather than patches of it.

Our model is most closely related to the Voice Conversion model of Fang *et al.* [8]. They also employ both adversarial and reconstruction loss terms, except they employ a discriminator per domain (for their task, a domain is a particular speaker, whereas for our task a domain is either the whispered or the voiced speech mode). They parameterize audio according to the WORLD vocoder [39], which deconstructs it into pitch, harmonic spectral envelope and aperiodic spectral envelope. For voice conversion, the aperiodic spectral envelope is copied, the harmonic spectral envelope is converted by feeding its lower MFCCs to a fully connected DNN and pitch is converted by a linear mapping. In our work, we use the WGAN paradigm rather than the traditional GAN paradigm. The most striking difference is that our source data does not have a pitch that can be mapped by a simple linear operation.

Network Architecture The U-net architecture of W2S-C converter is inspired by pix2pix [18] and the work of Janssen *et al.* [19], who employ a U-net for audio source separation.

We treat the frequency dimension as 'height', temporal dimension as 'width' and the feature channel dimension as 'depth'. In the downward phase the channel depth doubles with each layer and height is halved by max-pooling layers. After 6 layers, the process is reversed. The featuremaps are upsampled by a transposed convolution of height and stride 2 to mirror the max-pooling operation (preserving depth). After upsampling, the respective featuremap of the downward phase is cropped to match and concatenated in the depth dimension. This combined featuremap is convolved to obtain a new featuremap (decimating the depth of the total featuremap to a

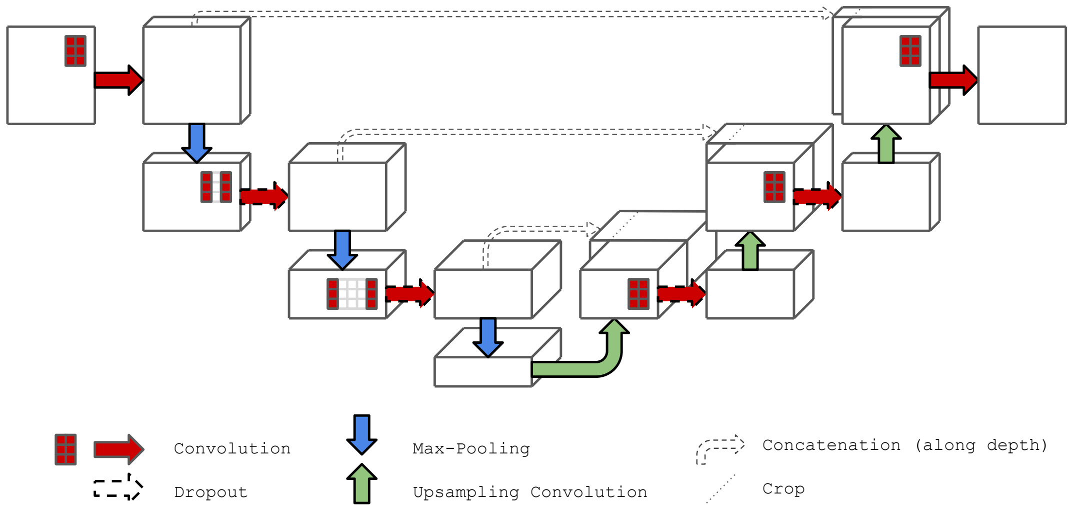


FIGURE 3.11: Architecture of the converter network (a convolutional U-net). The figure shows 3 downward layers, but in reality the network has 6.

quarter, or to half of the depth before concatenation of the downward featuremap). The height of all convolutional kernels is 3, whereas the width is 2. During the downward phase, the convolutions are dilated with an exponentially increasing factor. This idea has been borrowed from WaveNet [56], which employs increasingly dilated convolutions to ensure a large receptive field size. Convolutions in the upward phase are not dilated. All (non-upsampling) convolutions except the outermost layers employ dropout ($p = .25$). All convolutions (except the very last) are followed by an instance normalisation[4] layer and a leaky-ReLU activation function with leakiness .1. The final convolution has ReLU activation. We set the depth of the initial convolution to 10. The architecture is shown in figure 3.11.

There are two converter networks in our training paradigm; W2S and S2W. For the former it is necessary that the network does not use information 'from the future', because this would cause latency in a real-time setting. Hence, before an input is shown to the W2S network we pad it with zeros to the left (in the 'past') and crop feature maps from the left before concatenation. The S2W network does not have this constraint, because we will never require real-time speech to whisper conversion. Before an input is shown to the S2W network we pad it with zeros on both the left and the right and crop featuremaps from both sides before concatenation.

The critic network is a convolutional network consisting of 5 convolutional layers with 3×3 kernels and 2×2 stride that double channel depth per layer. A final convolution with kernel width 1 takes the height back to 1 (for our hyperparameters, this means the kernel height is 3) and again doubles the depth. The following 1×1 convolution halves the depth, and the final convolution takes depth to 1. Because the input length of the critic is variable it may happen that the output is not a single scalar. We make sure to output a single scalar by taking the mean of the final outputs. All convolutions before height becomes 1 use instance normalisation. All layers but the last use regular ReLU activation. The output has linear activation. No dropout is applied. We set the depth of the initial convolution to 25. The architecture

[4]Instance normalization and batch normalization are equivalent if the batch size is 1.

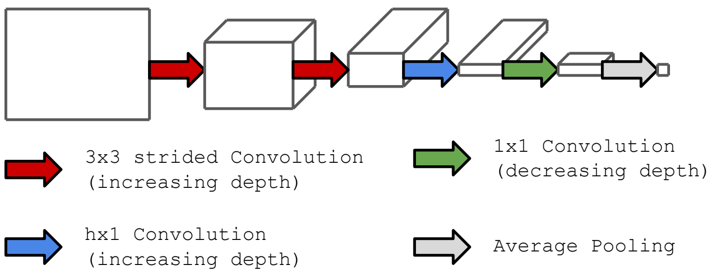

3x3 strided Convolution (increasing depth)
1x1 Convolution (decreasing depth)
hx1 Convolution (increasing depth)
Average Pooling

FIGURE 3.12: Architecture of the critic network. The figure shows 2 3x3-convolutions and a single convolution of decreasing depth, whereas in reality there are 5 3x3-convolutions and 2 decreasing depth convolutions.

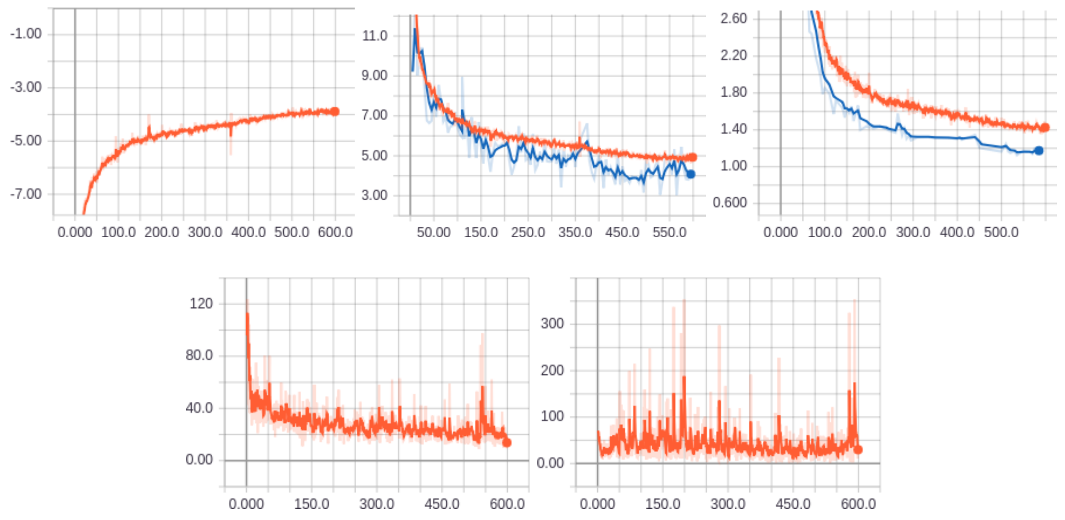


FIGURE 3.13: Convergence of the critic loss (top-left), adversarial loss (top-centre) and the reconstruction loss (top-right) of the Convolutional WGAN model. Validation losses in blue. The bottom row shows $||\nabla_\theta \mathcal{L}_{adv}||$ (bottom-left) and $||\nabla_\theta \mathcal{L}_{recon}||$ (bottom-right).

is shown in figure 3.12.

Training procedure. The weight of the gradient penalty is set to $\lambda = 10$ and the weight of the reconstruction loss is set to $\alpha = 1000$. For these settings, the gradients of the converter network due to the two loss metrics are of about the same magnitude. The critic is trained for 4 steps per training step of the converter network. All networks are optimized with the ADAM optimizer with learning rate 10^{-4} for 600 training epochs of 600 items (counting the amount of items that the converter network sees). Batch size is set to 1.

Figure 3.13 shows the close relation between the critic loss — the loss the critic aims to minimize — and the adversarial loss — which the converter aims to minimize. The critic loss is the negative estimated Wasserstein distance and the gradient penalty. The adversarial loss is the positive estimated Wasserstein distance.

Results and Discussion. Figure 3.14 shows that W2S-C improves over W2S-L and W2S-R. In samples by W2S-C wee can make out definite pitch and voices sound more natural. Though intelligibility is far better than for W2S-L and W2S-R it is

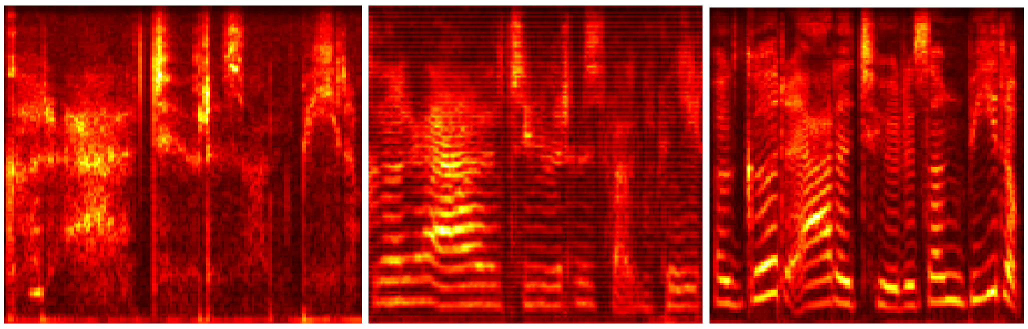


FIGURE 3.14: Whisper (left), converted by W2S-C (centre) and ground-truth speech (right). The excerpt tag in the wTIMIT dataset is ′s123u031′. Audio can be found on ′drive/W2S-C/...′

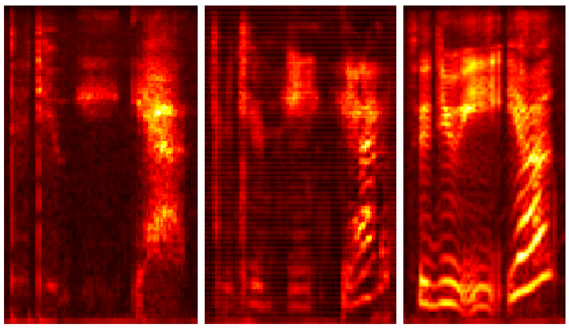


FIGURE 3.15: Whisper (left), converted by W2S-C (centre) and ground-truth speech (right). Questions are often characterised by rising pitch in the last syllable. W2S-C picks up on this prosodic profile. The excerpt tag in the wTIMIT dataset is ′s123u070′. Audio can be found on ′drive/W2S-C/...′

challenging to make out the words.

We notice that W2S-C can sometimes confuse the identity of the speaker. One example is shown in figure 3.16. W2S-C is trained to have realistic output (with the critic loss) and represent the input (with the reconstruction loss). The voice of another person still being a realistic spectrogram gives the converter free reign in modifying the personality of the speaker. Chapter 4 is dedicated to resolving that issue.

All converter networks in W2S-L, W2S-R and W2S-C have about 1.5M parameters. Although it is true that the WGAN paradigm employs additional networks — which would increase the parameter count — we do not consider these to be part of the conversion network, because they are only employed to *teach* the conversion network. At test time, we require only the converter network itself. This observation further strengthens our conclusion that among these alternatives W2S-C is the best paradigm for whisper-to-speech conversion.

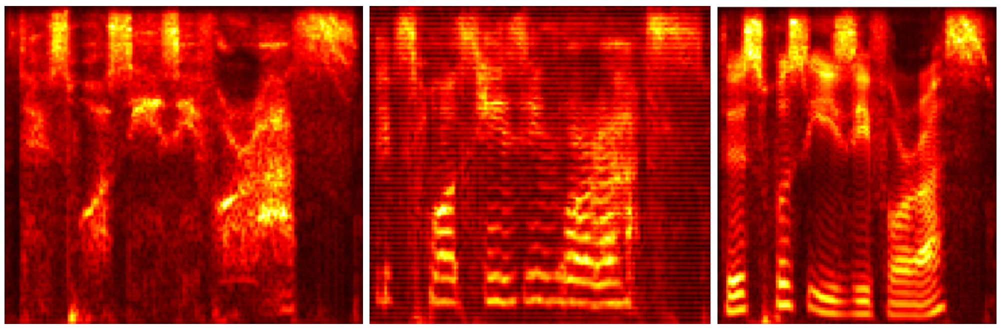


FIGURE 3.16: Whisper (left), W2S-C output (center) and ground truth speech (right). The pitch of W2S-C is higher than is characteristic for this speaker, implying that the model has trouble with speaker identity. The excerpt tag in the wTIMIT dataset is 's000u003'. Audio can be found on 'drive/W2S-C/...'

Chapter 4

Speaker Identity

W2S-C can sometimes output a voice that does not resemble the real spoken voice of the whisperer. Figure 3.16 shows an extreme example of this. This chapter is dedicated to resolving that issue.

Section 4.1 describes how we incentivize the converter network to maintain speaker identity with an additional loss term. Challengingly, all the model has to infer spoken voice characteristics from is the whispered input. In section 4.2 we propose to train speaker embeddings that can be used as additional input.

4.1 Speaker Identification as Auxiliary Task (W2S-aux)

It should be possible to recognize the speaker from the output of the W2S model. Therefore, we propose to train a W2S model to minimize an identification loss. In the first stage of training we train a speaker identity classifier network (the identifier) on spoken data and freeze its weights upon convergence. The second training stage is similar to the procedure of W2S-C, but we add the identifier loss to the objective and backpropagate its gradient to the W2S network. A training schematic is shown in figure 4.1. We call the model W2S-aux.

The identifier loss is the Cross Entropy Loss. The output of the identifier i_v is a softmax-normalised prediction vector $\mathbf{p} \in \mathbb{R}^D$ and the target is a one-hot vector $\mathbf{t} \in \mathbb{R}^D$ of the a-th speaker. As we held out 4 of 48 speakers of the wTIMIT dataset, we set $D = 44$.

$$\mathbf{p} = i_v(Y)$$

$$t_{i=a} = 1 \; ; \; t_{i \neq a} = 0$$

$$\mathcal{L}_{CE}(\mathbf{p}, a) = -\sum_{i}^{D} t_i \log p_i + (1 - t_i) \log(1 - p_i)$$

$$\mathcal{L}_{idtf}(Y) = \beta \cdot \mathcal{L}_{CE}(i_v(Y), a)$$

In the first stage of training, we train i_v to classify real spoken spectrograms y:

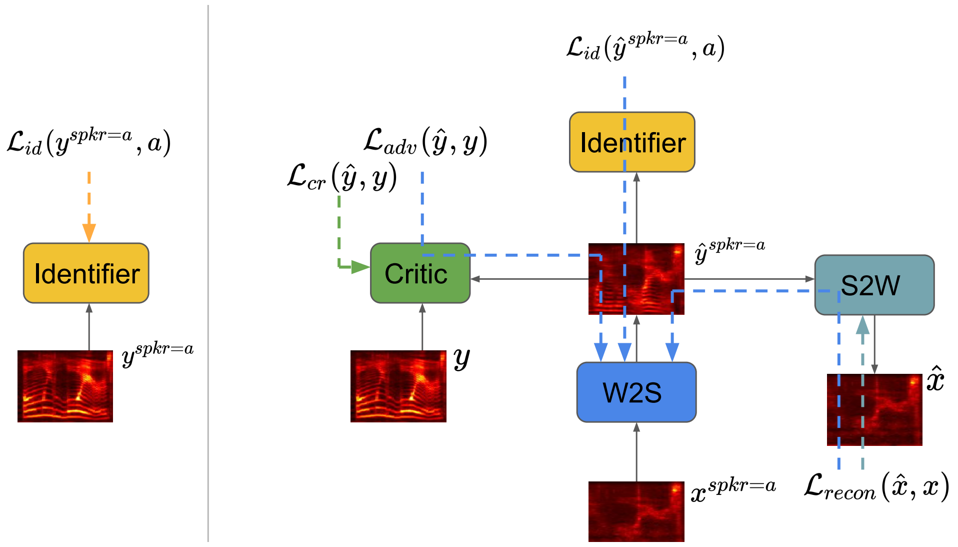
$$v \leftarrow \nabla_v \mathcal{L}_{idtf}(Y)$$

In the second stage of training, the weights v are frozen and the W2S network minimizes the following loss, including a term obtained by showing fake speech $\widehat{Y}$ to the identifier:

$$\widehat{Y} = f_\theta(X)$$

$$\mathcal{L}_{w2s} = \mathcal{L}_{adv} + \mathcal{L}_{recon} + \mathcal{L}_{idtf}(\widehat{Y})$$

$$\theta \leftarrow \nabla_\theta \mathcal{L}_{w2s}$$



$\mathcal{L}_{id}(y^{spkr=a}, a)$
Identifier
$y^{spkr=a}$
$\mathcal{L}_{id}(\hat{y}^{spkr=a}, a)$
$\mathcal{L}_{adv}(\hat{y}, y)$
$\mathcal{L}_{cr}(\hat{y}, y)$
Identifier
Critic
$\hat{y}^{spkr=a}$
S2W
y
W2S
$\hat{x}$
$x^{spkr=a}$
$\mathcal{L}_{recon}(\hat{x}, x)$

FIGURE 4.1: Training schematic for the Convolution WGAN paradigm with additional identifier loss. In the first training stage (left) the identifier is trained to classify speakers based on real spoken excerpts. In the second training phase (right) we keep the identifier network fixed, but use it to backpropagate the classification loss to the converter network.

Weighting by β ensures that:

$$||\nabla_\theta \mathcal{L}_{idtf}(\widehat{Y})|| \approx ||\nabla_\theta \mathcal{L}_{adv}|| \approx ||\nabla_\theta \mathcal{L}_{recon}||$$

Model Architecture. The architecture of the identifier network is the same as the critic network as described in section 3.3 and figure 3.12, except that its initial depth is 8 and that its final convolutional layer has a depth of 44. It has about 200k parameters. The other architectures are the same as for W2S-C.

Training Procedure. In the first training stage of W2S-aux, we train the identifier for 30 true epochs with the ADAM optimizer with learning rate 10^{-4} and weight decay 10^{-4}.

Aside from training W2S-aux, we also train an identifier on only whispered audio so as to get a feeling for how much of a person's identity can be derived from whisper. We train the whisper-identifier under the same conditions, but on the whispered part of WTIMIT.

During the second stage of training W2S-aux, we freeze the weights of the speech-identifier and set $\beta = .75$. All other settings are the same as for W2S-C.

Results. The convergence of the speech identifier is shown in figure 4.2. It converges at 98% validation accuracy.

The convergence of the whisper identifier is shown in figure 4.3 and converges at 96% validation accuracy. This means that whispers contain enough information to predict identity from. However, this score does not grant insight into how well someone's spoken voice characteristics can be derived from their whispers.

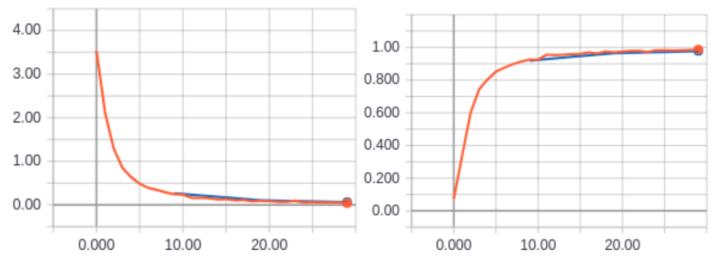


FIGURE 4.2: Loss (left) and accuracy (right) of the identifier network per epoch in the first training stage. The validation score is shown in blue and validation accuracy converges at 98%.

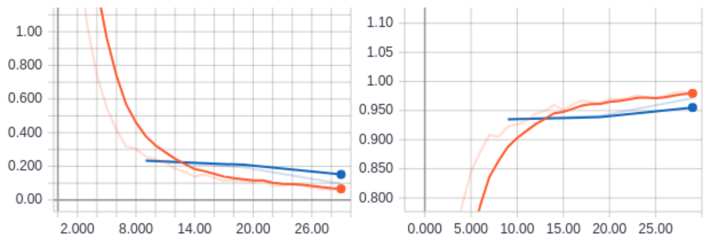


FIGURE 4.3: Loss (left) and accuracy (right) of the whisper identifier. The validation accuracy converges at 96%.

During training of W2S-C we kept track of the identification score on the validation set (evaluated by the identifier model of this chapter). The loss was not backpropagated; W2S-C was not optimized for speaker identification. The identification loss and accuracy of the validation set is shown in figure 4.4 and reaches at 55%.

W2S-aux *has* been optimized for speaker identification. Its convergence is shown in figure 4.5. Identification accuracy on the validation set converges at 95%.

Discussion. Figure 4.6 shows that the model can still mistake characteristics of the speaker. This is likely due to the fact that it is hard to infer spoken voice characteristics from whispers alone. In general, the performance of W2S-aux is equal to that of W2S-C.

Because the training and validation set contain the same speakers, it may be the case that the network is by-hearting their characteristics and essentially overfitting to speaker identity (rather than to utterance).

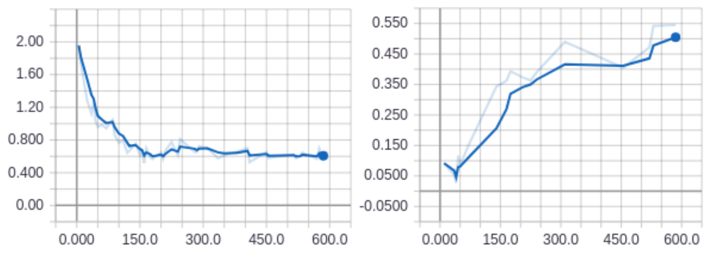


FIGURE 4.4: Identification loss (left) and accuracy (right) of W2S-C on the validation set, as evaluated by the speech identifier of this chapter (with frozen weights).

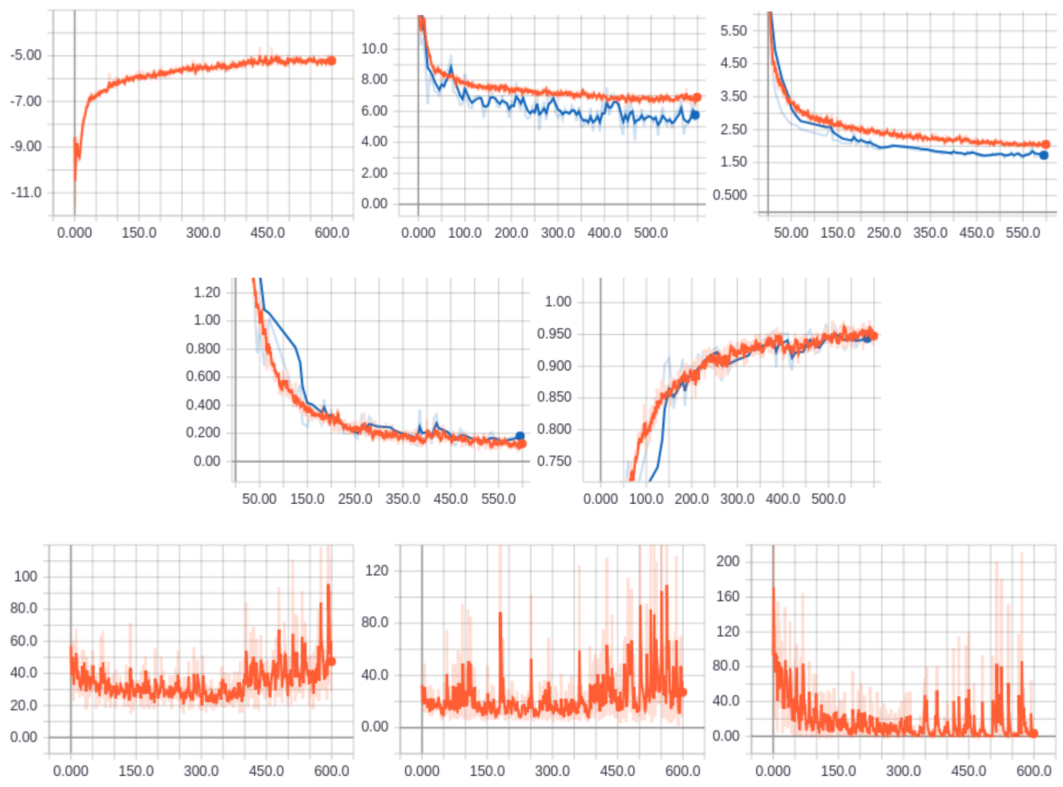


FIGURE 4.5: Convergence of $\mathcal{L}_{cr}$ (top left), $\mathcal{L}_{adv}$ (top centre),$\mathcal{L}_{recon}$ (top right), $\mathcal{L}_{idtf}$ (centre left) and identifier accuracy (centre right) for W2S-aux. The bottom row shows $||\nabla_\theta \mathcal{L}_{adv}||$ (bottom-left), $||\nabla_\theta \mathcal{L}_{recon}||$ (bottom centre) and $||\nabla_\theta \mathcal{L}_{idtf}||$ (bottom right).

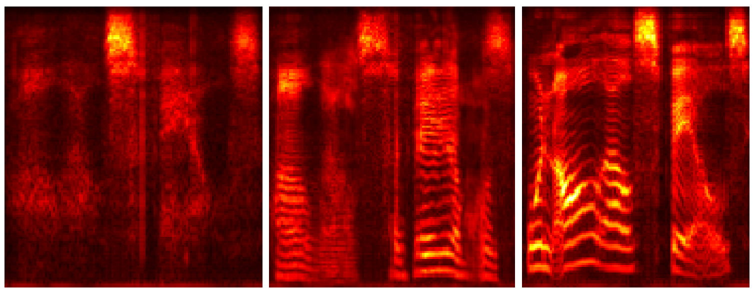


FIGURE 4.6: Whisper (left), W2S-aux output (centre) and ground truth speech (right). The pitch of W2S-aux is lower than is characteristic for this speaker, implying that W2S-aux still has trouble with speaker identity. The excerpt tag in the wTIMIT dataset is 's123u022'.

4.2 Speaker Embeddings (W2S-embed)

In this section we propose to provide the W2S converter with additional input to help it synthesize speech with the correct voice. We extract speaker embeddings; vectorial representations of a person's spoken voice — and condition the W2S model on them. We call the model W2S-embed.

Arik *et al.* [1] extract speaker embeddings by backpropagating the regression loss (over spectrograms) of a TTS-task to a trainable vector. Speaker embeddings can also be obtained by training a network to predict them directly (if some pre-computed embeddings are already available). Other approaches are based on *i*-vectors [34] or on a bottleneck layer of a network with a classification loss [28, 21]. We follow the latter approach, making use of the identifier/critic architecture we have used in the previous section.

Jia *et al.* [21] note that training good speaker embeddings requires a dataset of many speakers, though typically as little as 10 minutes per speaker will suffice.

In the first training stage of W2S-embed, we train an identifier network on the LibriSpeech [43] dataset (augmented with the spoken part of wTIMIT). We use the training set containing 360 hours of audiobooks by 921 different speakers (about 20 minutes of audio per speaker). LibriSpeech contains only spoken voice.

We regard the activations in penultimate layer of the identifier network as an embedding of the voice of the speaker of the excerpt. To obtain a speaker embedding, we take the mean of the embeddings of all her utterances. The final set of speaker embeddings is normalised such that every dimension is distributed according to $\mathcal{N}(0, 1)$.

Model Architecture. The identifier network has the same architecture as in sections 3.3 and 4.1 (shown in figure 3.12), except that the dimensionality of the prediction vector is 956 and the initial depth is set to 16, which brings the parameter count up to about 1M. The penultimate layer — which determines the speaker embedding — has a dimensionality of 256.

We modify the W2S network such that we can condition its prediction on the speaker embedding. Its architecture is shown in figure 4.7 (original non-conditioned architecture is shown in figure 3.11). For each of the layers in the upward phase, we define a 1×1 convolution over the speaker embedding with equal output dimensionality of the layer of interest. The $1 \times 1 \times depth$ output of the convolution is broadcast to $height \times width \times depth$ and added (addition, not concatenation) to the featuremap before the activation function.

Training Procedure. The identifier network was trained under the same conditions as in 4.1, but for 50 true epochs, after which we use it to determine speaker embeddings.

During the second training stage of W2S-embed we randomly select one of three scenario's with equal probability for each batch:

- Condition the W2S network on the speaker embedding of the whisperer and backpropagate identification loss on the identity of the actual whisperer. This pass ensures that W2S learns to interpret the speaker embedding.
- Condition the W2S network on the zero-vector and backpropagate identification loss on the identity of the actual whisperer (effectively the situation of

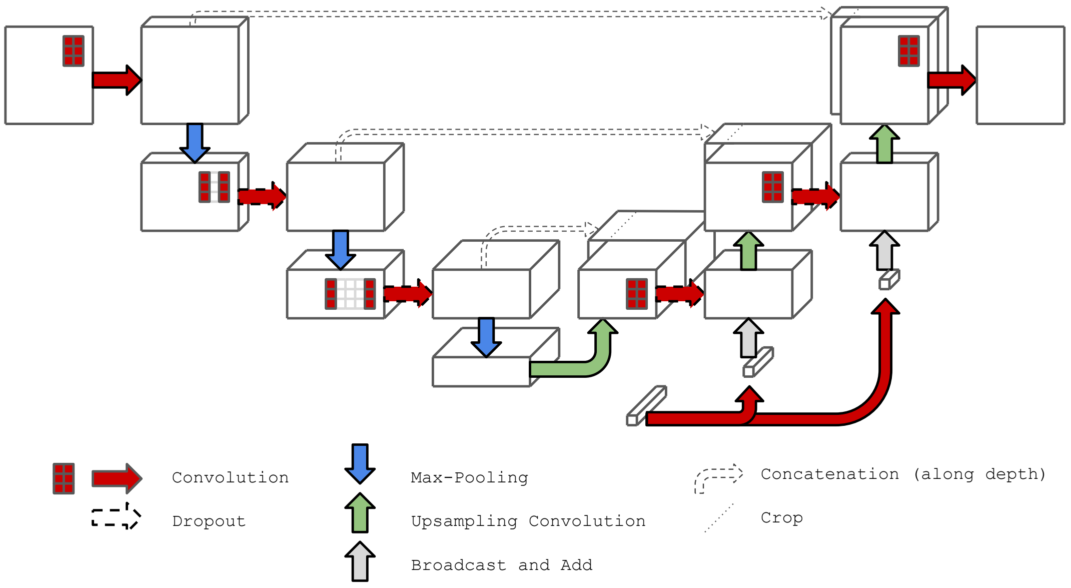


FIGURE 4.7: Architecture of the converter network W2S-embed. The speaker embedding is shown on the bottom. The feature map is put through the activation function (not shown in the figure) after addition with the broadcast featuremap of the conditioning module. The figure shows 3 downward layers, but in reality the network has 6.

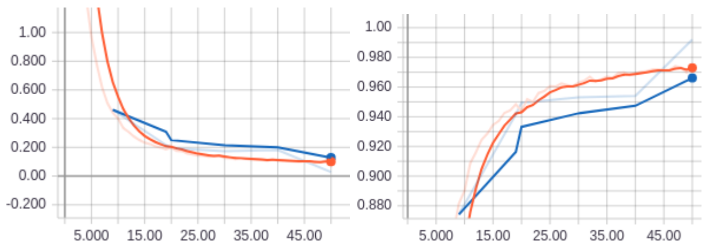


FIGURE 4.8: Loss (left) and accuracy (right) of the identifier network of W2S-embed in the first training stage. The validation accuracy converges at 99%.

W2S-aux in section 4.1). This pass makes sure the network cannot get lazy and should still pay attention to cues in the whisper.

- Condition the W2S model on a random speaker embedding and backpropagate the identification loss on the identity of the drawn speaker embedding. This also ensures that W2S learns to interpret speaker embeddings and it allows us to leverage the 921 speaker embeddings of the LibriSpeech speakers.

Otherwise the training procedure is exactly that of W2S-aux (section 4.1).

Results. The convergence of the identifier network of W2S-embed is shown in figure 4.8 and reaches 99% validation accuracy.

Convergence of the second training stage is shown in figure 4.9. Identification accuracy on the validation set reaches 94%.

User Survey. To better quantify the benefit of conditioning on speaker embeddings we perform a user study. We compare intelligibility and overall quality of speech, whisper, W2S-C and W2S-embed. We ask participants to transcribe 16 sentences and rate each clip on general audio quality on a scale of 1 to 5. Participants are asked to

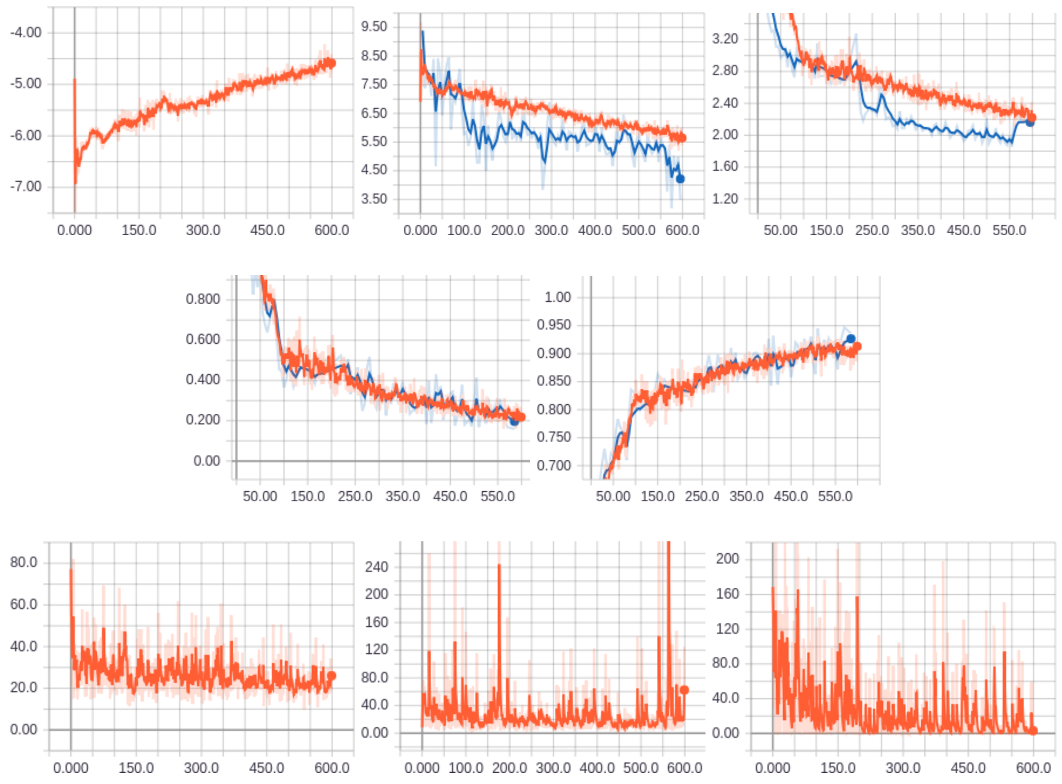


FIGURE 4.9: Convergence of $\mathcal{L}_{cr}$ (top left), $\mathcal{L}_{adv}$ (top centre), $\mathcal{L}_{recon}$ (top right), $\mathcal{L}_{idtf}$ (centre left) and identifier accuracy (centre right) for W2S-embed. The bottom row shows $||\nabla_\theta \mathcal{L}_{adv}||$ (bottom-left), $||\nabla_\theta \mathcal{L}_{recon}||$ (bottom centre) and $||\nabla_\theta \mathcal{L}_{idtf}||$ (bottom right).

	μ	σ	n
Speech	4.2	0.8	64
Whisper	3.5	1.1	50
W2S-C	1.8	0.8	47
W2S-embed	2.0	0.7	43

TABLE 4.1: User Survey Results - Perceived Audio Quality on a Scale of 1 to 5. For these statistics, the hypothesis W2S-embed > W2S-C has a p-value of .10, which is not statistically significant.

	% Correct	% Semi-Correct	% Incorrect	n
Speech	91	8	1	65
Whisper	74	18	8	51
W2S-C	11	20	69	55
W2S-embed	29	27	43	51

TABLE 4.2: User Survey Results - Intelligibility by Percentage Correctly Transcribed Samples. For these statistics, the hypothesis W2S-embed > W2S-C has a p-value of $6 \cdot 10^{-11}$, which is statistically significant (counting correct=1, semi-correct=.5 and incorrect=0).

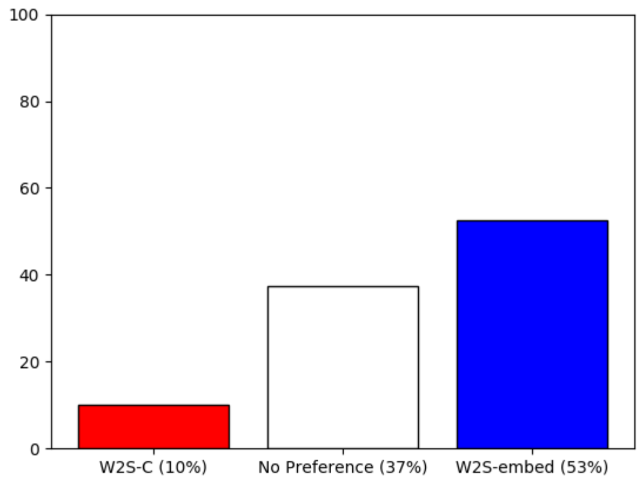


FIGURE 4.10: User preference for W2S-C or W2S-embed in a direct comparison ($n = 51$).

only listen to the excerpt once, so as to have a fair evaluation of intelligibility[1]. The set of 16 sentences shown to a participant is balanced in that it contains 4 excerpts of each condition (speech, whisper, W2S-C and W2S-embed) where the 4 speakers are an SG and US man and woman. 4 such balanced sets are prepared (the same 16 sentences). A participant is shown one of these sets in a different random order for each participant.

Additionally, we ask participants which excerpt they prefer in a direct comparison between W2S-C and W2S-embed (same sentence, same speaker). For this test they are instructed to replay the clips as often as they need.

There are 22 participants. Per result we report n as the amount of statements gathered.

Table 4.2 indicates that W2S-embed is more intelligible than W2S-C, although both are drastically worse than whisper and speech. Ideally, the intelligibility of a W2S model is equal or higher than that of whisper. Table 4.1 shows that perceived quality of W2S-embed is not better than W2S-C in a statistically significant sense. However, figure 4.10 shows that people have a preference for W2S-embed in a direct comparison.

Discussion. Figure 4.11 shows that W2S-embed model outputs audio with a voice that is close to the speaker's true voice. The W2S-C and W2S-aux models only confuse speaker identity in about 5% of cases whereas the W2S-embed model doesn't make such mistakes anymore. Although untested by the user study, this may be the biggest benefit of W2S-embed; speech is synthesized with the true voice of the user.

Figure 4.12 shows that conditioning on a different speaker embedding changes the characteristics of the output speech, meaning that the W2S-embed model has learned to pay attention to the speaker embedding. Best quality speech is obtained by conditioning on the speaker embedding of the actual whisperer, as the model

[1]There would have been more experimental control if the participants were completely unable to replay the clip. In this survey they did have the option to.

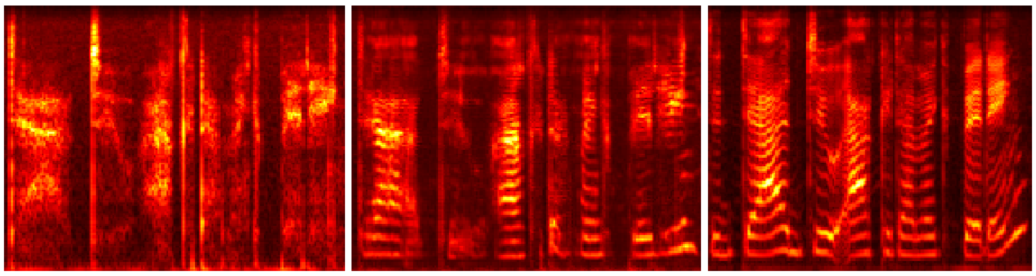


FIGURE 4.11: Whisper (left), W2S-embed output (centre) and ground truth speech (right). The excerpt tag in the wTIMIT dataset is 's123u027'.

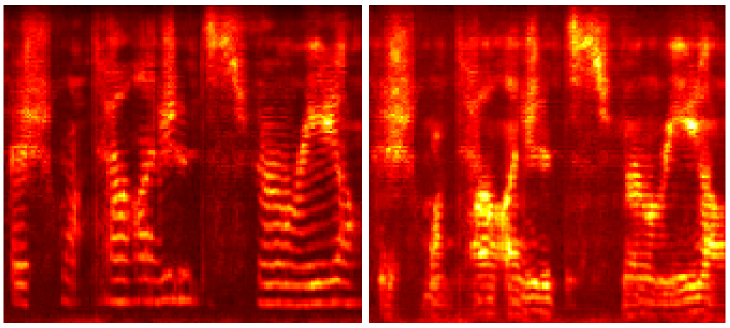


FIGURE 4.12: W2S-embed conditioned on the speaker embedding of speaker 123 (left) and of speaker 107 (right). The excerpt tag in the wTIMIT dataset is 's123u126'.

still does pay attention to the input whisper when inferring speech characteristics and the two sources of information shouldn't conflict.

Intelligibility and quality is not yet that of whisper or speech. Ideally, we'd like intelligibility and quality to be higher than that of whisper (prioritising intelligibility). This model is not yet ready for a real-world application. However, the emergence of pitch and prosody from completely unvoiced input by W2S-C was already a promising result. It is reassuring that W2S-embed can maintain speaker identity.

Meaningful Embeddings. By projecting the embeddings on a 2D subspace we can visually inspect similarities in speaker identity. Figures 4.13 and 4.14 show that the embeddings encode high-level meaning such as speaker gender and native tongue.

We inspect excerpts of female speakers in the 'male domain'. The striking characteristic is not a particularly masculine voice, but a low-frequency buzz in the background. It appears that the first principal components primarily register the presence/absence of low frequencies. This may be unsurprising and even desired, but it is not desired that background noise can influence the speaker embedding. Additional focus on data pre-processing (e.g. denoising) could greatly benefit the models of this thesis.

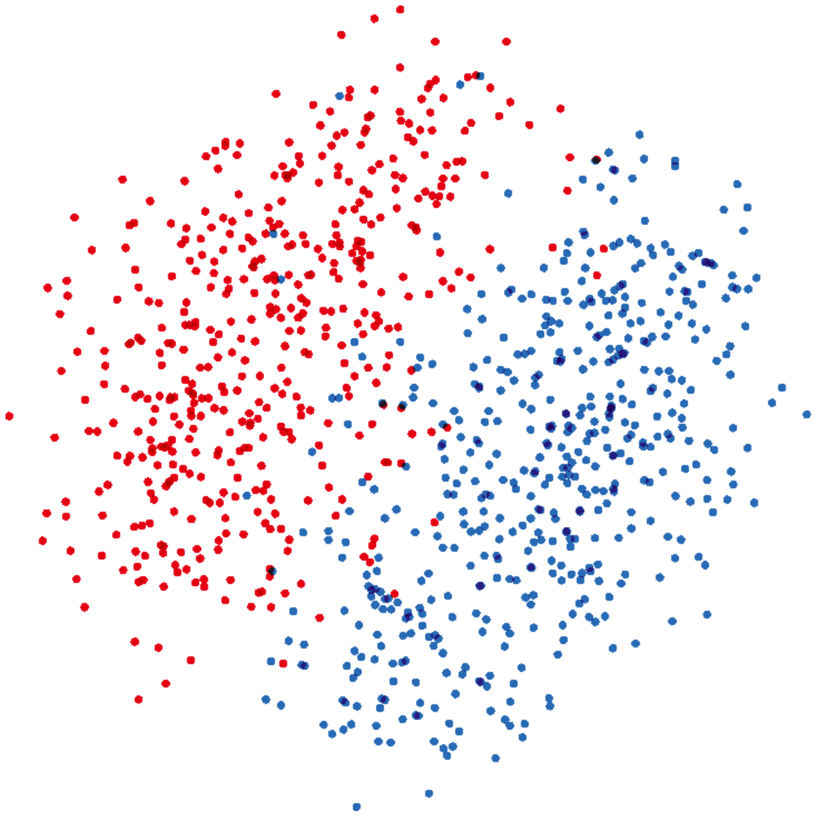


FIGURE 4.13: 2D subspace of speaker embeddings found by PCA. Male speakers in blue, female speakers in red.

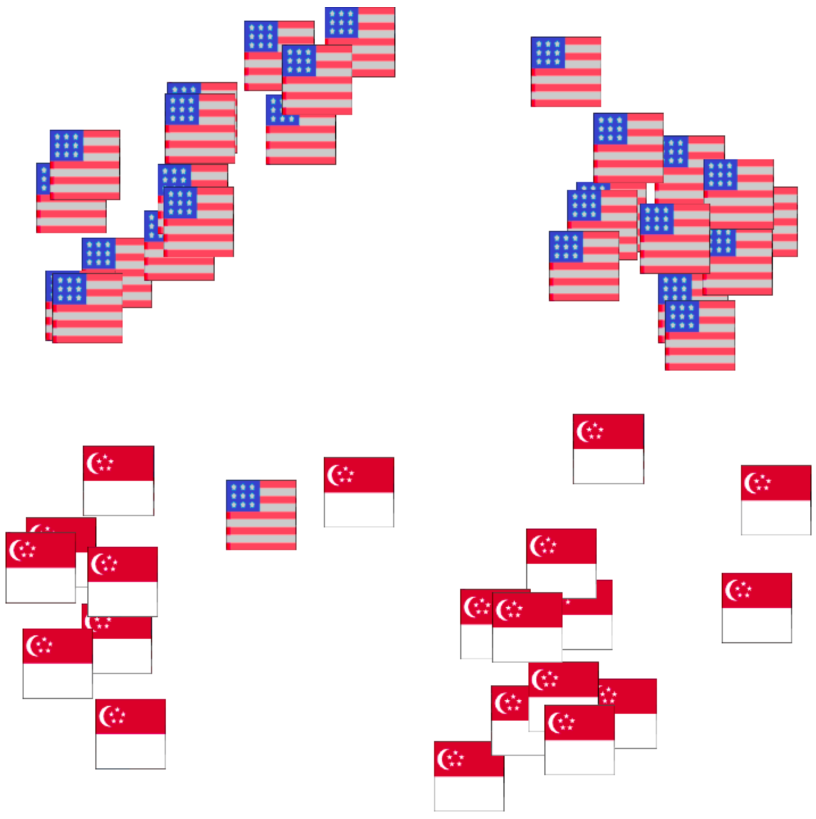


FIGURE 4.14: 2D subspace of speaker nationality found by PCA (only speakers in the wTIMIT dataset).

Chapter 5

Beyond the Spectrogram Level

All models discussed in chapters 3 and 4 output magnitude spectrograms. However, the spectrogram is not yet the waveform, and we must perform some steps to obtain it. The standard approach is to perform the iterative Griffin-Lim algorithm. In this chapter we explore alternatives, but will not find an improvement.

We require STFT and iSTFT to be differentiable. Hence, we implement them in terms of convolutions. An additional benefit is that we can control the evaluated frequencies, which allows us to draw complex-valued mel-scaled spectrograms. Section 5.1 discusses this implementation.

Section 5.2 discusses the importance and the difficulty of working with phases in more detail.

Section 5.3 discusses the current literature on spectrogram inversion.

In section 5.4 we propose PhaseNet, a simple recurrent neural network that outputs phase values for a given magnitude spectrogram. PhaseNet is trained by spectrogram continuity only, which means there is no need for a ground-truth waveform.

In section 5.5 we propose to modify W2S-C to output multichannel spectrograms. This is similar to modifying an image based network to output multiple color channels rather than grey-scale images. We call the model W2S-M.

In section 5.6 we infer a source-filter model that is implicit in the spectrogram and synthesize audio according to it.

For the models in this chapter we employ a data augmentation. At training time we cut n samples from the start of the excerpt before computing the spectrogram, where $n \sim U(0, hop)$. This constitutes large changes the phase values of the spectrogram, whereas the magnitude will change little.

5.1 Mel-scaled STFT (STmFT)

We implement $STFT$ as a convolutional operation in PyTorch, which gives us freedom to specify which frequencies we evaluate and the possibility of backpropagating the gradient through the operation. It should be noted that the $\mathcal{O}(n \log n)$ FFT-trick of Cooley and Tukey [5] cannot be applied if evaluated frequencies are not linearly scaled. Hence, the complexity is $\mathcal{O}(n^2)$. We did not find this to be a bottleneck.

Forward STmFT. We repeat the math of the $STFT$ from section 2.2:

$$\phi = \frac{2\pi f}{N}$$

$$X_{fm} = \sum_{n=m}^{m+N} x_n w_n e^{i\phi n}$$

$$X_{fm} = \sum_{n=m}^{m+N} x_n w_n (cos(\phi n) + i\ sin(\phi n))$$

We can see that the real and imaginary values of the $STFT$ describe how much the signal window **x** agrees with a cosine and (negative) sine of the frequency f (windowed by **w**, typically the Hamming window). This is a convolution where the filters are cosines and sines of the frequencies of interest. The window size is the length of the filter and the hop size is the stride. We multiply the windowing function with the filters. We rearrange the output to be a 3D tensor: time, frequency and complex component.

$$A_{ft0} = \Re(X_{ft})\ ;\ A_{ft1} = \Im(X_{ft})$$

Inverse STmFT. We implement $iSTmFT$ as a transposed convolution with the same filters as the forward operation (divided by the amount of frequencies that are evaluated). To properly account for the windowing operation, we apply the following formula [11][1]:

$$x[n] \leftarrow \frac{\sum_t x_t[n]w[n-tH]}{\sum_t w^2[n-tH]}$$

where H is the hop size.

The term in the numerator has been accounted for by multiplying the filters by the window. The term in the denominator is obtained by convolving over a sequence of ones with the same length of the signal **x**. The convolutional filter is $\mathbf{w}^2$ and the operation has the same filter length and stride.

One improvement over the current implementation would be to store the denominator. As it does not depend on the input, re-computing it is redundant.

Mel-scaled convolutions. Convolutional $STFT$ gives us the freedom to pick the frequencies of interest. We propose to use frequencies derived from the mel-scale (see section 2.4). We notice that the mel-scale makes $iSTmFT(STmFT(x))$ emphasize lower frequencies. To counter-balance this, we propose to multiply the filters of higher frequencies with a weight proportional to the ratio between mel- and linear-scale.

$$x_m = \begin{cases} x_l & \text{if } x_l < 1000 \\ 1000\log(\frac{x_l}{1000}) + 1000 & \text{if } x_l \geq 1000 \end{cases}$$

$$w = \frac{x_l}{x_m}$$

Current practice is to derive the mel-spectrogram from the linear magnitude spectrogram, which implies that the mel-spectrogram is always a magnitude spectrogram. $STmFT$ computes phase values as well. Hence the mel-spectrogram can readily be inverted by the $iSTmFT$ without a mel-inversionbank or Griffin-Lim.

We've found that $iSTmFT(STmFT(x))$ has low quality with 128 frequency bins. We suspect that using fewer bins — fewer frequencies that can be imposed on the wave — is the cause of quality loss. It is customary to perform $iSTFT$ or Griffin-Lim on

[1]https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.istft.html

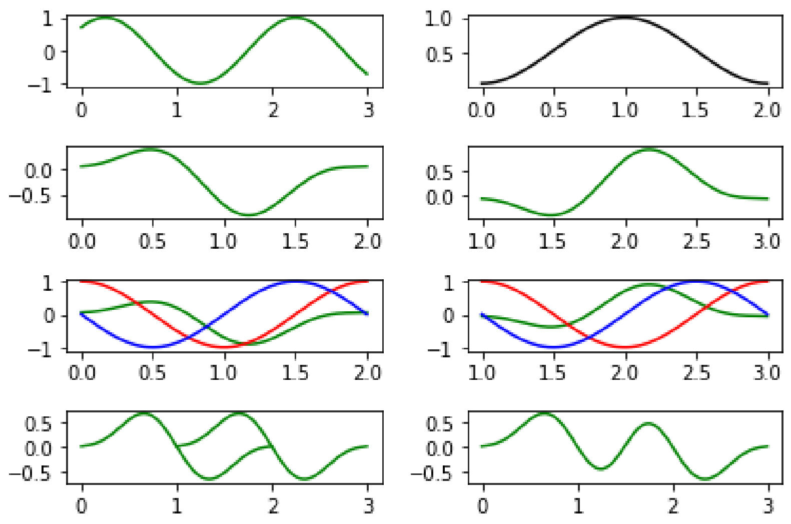


FIGURE 5.1: An example of how spectrogram inversion with unsuitable phase values corrupts the reconstructed waveform. The original signal is shown in the top left. The second row shows two overlapping windows of the signal multiplied by the Hamming window (top right). The third row shows the in-phase and quadrature-phase convolutional filter. Inverting the magnitude spectrogram with an unsuitable phase (here 0 for both windows) amounts to summing the windows shown in the bottom left. The final signal is shown in the bottom right. Even though the reconstructed signal is derived from the spectrogram of the original waveform it is very different from the original wave. A higher-frequency artifact has been introduced.

the linear spectrogram, which typically has more bins (at least 271 in our previous experiments). We recommend to use at least 256 mel bins for $iSTmFT$.

5.2 Importance and Difficulty of Phase

When recording audio with two microphones a few centimeters apart, the phase values of the two signals' $STFT$ will be radically different[2]. However, the difference in audio content is negligible, if perceivable at all. This is the reason that for many recognition tasks a magnitude spectrogram suffices; the phases don't encode high-level semantics. In synthesis, however, we need some estimate of the phase, as the input to $iSTFT$ demands a complex-valued spectrogram.

The spectrograms we obtain from the models in chapter 3 are magnitude spectrograms. Hence, we have no estimate of the phases. What we are really after are waveforms *whose spectrograms* are the spectrograms obtained by the model. As we can see in the simple example of 5.1, inverting a spectrogram with inappropriate phase values renders a signal whose spectrogram is not the spectrogram that the signal was derived of.

[2]The phase values of the two recordings will be correlated by a linear shift proportional to the wavelength of the frequencies whose phase-offset they encode. The changes in their phases will thus not be completely random. However, section 5.5 will show that neural nets find it hard to learn invariance to this type of change.

Intuition says we should include the phase values in the spectrogram and predict them as well. However, predicting raw phase values in a supervised manner would be a bad idea. As mentioned before, the same audio recorded a few centimeters away can have very different values. We'd like a model to be invariant to such numerically large but perceivably insignificant changes. Predicting raw values would incur a large loss if all phase values are shifted, though it would not be perceivable. Additionally, phase values of frequencies that are not strongly present become increasingly random.

5.3 Literature Review on Spectrogram Inversion

Takamichi *et al.* [55] propose to train a Fully Connected DNN to predict a Von Mises distribution over the phase values (the Von Mises distribution is a distribution on a circle). They draw time-derivatives of raw phase values in order to use them as supervised targets.

The Deep Griffin-Lim paradigm of Masuyama *et al.* [30] formulates the task of spectrogram inversion as a denoising task. They modify their initial complex spectrogram $X^* \in \mathbb{C}^{F \times T}$ by adding complex Gaussian noise.

$$N_{ft} = \mathcal{N}(0, \sigma) + i \cdot \mathcal{N}(0, \sigma)$$

$$X_0 = X^* + N$$

Next, they perform a regular Griffin-Lim iteration, but use its artifacts as inputs to a neural network $\mathcal{F}_\theta$, whose task is to predict the residual between Z^* and X^*:

$$Y_t = STFT(iSTFT(X_t))$$

$$Z_t = A \odot Y_t \oslash |Y_t|$$

$$X_{t+1} \approx Z_0 - \mathcal{F}_\theta(X_t, Y_t, Z_t)$$

Where

$$Z_0 = STFT(iSTFT(X_0))$$

They then train F_θ to have X_t match X^* by minimizing MSE. At test time we can impose random phases to a spectrogram and regard this as a noisy spectrogram to be denoised by the Deep Griffin-Lim stack. Deep Griffin-Lim iterations — like traditional Griffin-Lim iterations — can be performed as often as necessary. It is manually adjustable for a quality/time tradeoff. Their benefit over traditional Griffin-Lim is that of a higher quality upper-bound.

MelNet [57] is a generative model based on the mel-spectrogram. It inverts spectrograms with the gradient-based approach of Decorsière *et al.* [7]. An initial signal estimate is updated along the gradient of the MSE between its spectrogram and the target spectrogram. By considering the STFT to be a multiplication with a filterbank they derive an analytical solution for the gradient.

Tacotron 2 [50] is amongst the state of the art in Text-to-Speech conversion (TTS). It predicts spectrograms and inverts them by conditioning a WaveNet [56] on them. WaveNet is high-quality but slow due to its autoregressive nature. This objection is resolved by developping flow-based generative models. Parallel WaveNet [40] and WaveGlow [45] are considered the current state of the art in raw audio waveform modelling. Chapter 6 discusses their like further.

The Multi-Headed Convolutional Neural Network (MCNN) [2] proposes to invert a spectrogram in a parallel manner. They perform upsampling in the temporal dimension using transposed convolutions with stride. It is trained by spectrogram-continuity; its loss is the MSE between the input spectrogram and the spectrogram derived from its output. Additional loss terms are taken over time derivatives of the phase values and over raw phase values weighted by respective frequency magnitude.

5.4 PhaseNet

The methods of section 5.3 assume that ground-truth waveforms are available. Although it is not hard to obtain waveform-spectrogram pairs, we want to train on the outputs of another network — which are just magnitude spectrograms. Although WaveNet-based models are the current state of the art in audio quality they are generally quite heavy models (as is MCNN). Considering a real-time application of which spectrogram inversion is only the second stage, we prefer a lightweight solution to spectrogram inversion. We propose to predict phase values that match the spectrogram with a simple RNN.

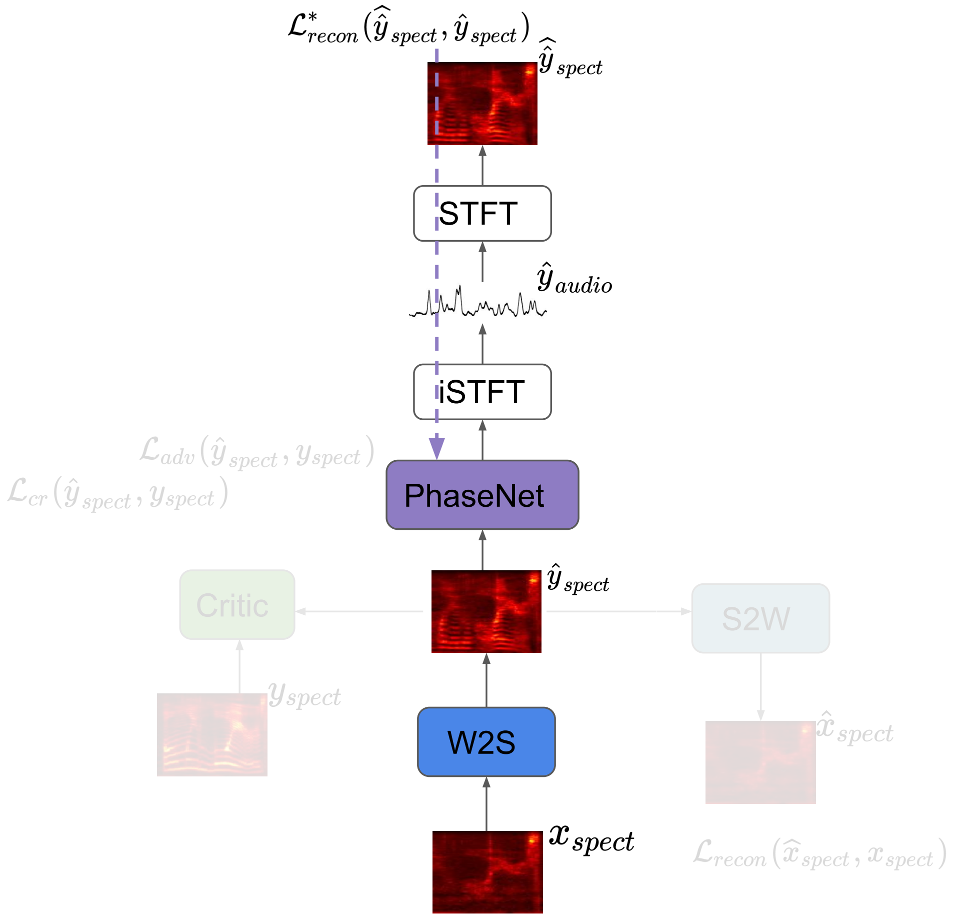
We propose to train a neural net j_ψ to predict phase values Φ that match a given spectrogram $\widehat{Y}$ such that the waveform derived from them has approximately $\widehat{Y}$ as its spectrogram (this is the same spectrogram continuity used to train MCNN). We implement j_ψ as a simple RNN that processes the spectrogram column by column and outputs angular values. We use $STmFT$ and its inverse to compute 256-dimensional complex-valued mel-spectrograms; $\widehat{\mathbf{y}}_t \in \mathbb{R}^{256}$; $\widehat{\mathbf{y}}^*_t \in \mathbb{C}^{256}$; $\boldsymbol{\phi}_t \in [-\pi, \pi), \mathbb{R}^{256}$:

$$\begin{gathered}
\boldsymbol{\phi}_t = j_\psi(\widehat{\mathbf{y}}_t, \widehat{Y}_{<t}) \\
\widehat{\mathbf{y}}^*_t = \widehat{\mathbf{y}}_t \cdot \cos(\boldsymbol{\phi}_t) - i \cdot \widehat{\mathbf{y}}_t \cdot \sin(\boldsymbol{\phi}_t) \\
\widehat{\widehat{Y}} = |STmFT(iSTmFT(\widehat{Y}^*))| \approx \widehat{Y} \\
\mathcal{L}_{continuity} = MSE(\widehat{\widehat{Y}}, \widehat{Y}) \\
\psi \leftarrow \nabla_\psi \mathcal{L}_{continuity}
\end{gathered}$$

It is optional to backpropagate this loss all the way to the network that generated $\widehat{Y}$, making the pipeline fully end-to-end. In our experiments, we only backpropagate the loss to PhaseNet. A training schematic is shown in figure 5.2.

Model Architecture. The architecture we propose is simple. PhaseNet consists of a 6-layer RNN[3] with input-, output- and hidden-dimensionality of 256. The final layer maps the 256 dimensional vector to two 256 dimensional vectors of which we take $atan2$ to output 256 angular values (which means values near $-\pi$ and near π

[3]Initial experiments were run on a 3-layer RNN, in order to keep the model lightweight. We found that neither 3-layer nor 6-layer PhaseNets are succesful.



$\mathcal{L}^*_{recon}(\widehat{y}_{spect}, \hat{y}_{spect})$
$\widehat{y}_{spect}$
STFT
$\hat{y}_{audio}$
iSTFT
$\mathcal{L}_{adv}(\hat{y}_{spect}, y_{spect})$
$\mathcal{L}_{cr}(\hat{y}_{spect}, y_{spect})$
PhaseNet
$\hat{y}_{spect}$
Critic
S2W
y_{spect}
W2S
$\widehat{x}_{spect}$
x_{spect}
$\mathcal{L}_{recon}(\widehat{x}_{spect}, x_{spect})$

FIGURE 5.2: Training schematic for PhaseNet.

are considered close together). PhaseNet has about 1M parameters.

$$atan2(y,x) = \begin{cases} arctan(\frac{y}{x}) & \text{if } x > 0 \\ arctan(\frac{y}{x}) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ arctan(\frac{y}{x}) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0 \end{cases}$$

Training Procedure. We train PhaseNet on the outputs of the CNN model of chapter 3. We denormalize the spectrograms before feeding them to PhaseNet, but we take the MSE loss over normalized spectrograms (normalized by $\sqrt[3]{10 \cdot \log(x+1)}$). We augment the data by removing $n \sim U(0, hop)$ samples from the start of the waveform before taking the spectrogram. The 128-dimensional output of W2S-C is upsampled to 256-dimensions by linear interpolation before it is shown to PhaseNet. We invert mel-scaled radial spectrograms with the $iSTmFT$ of section 5.1. The input (a single sequence) is batched into sub-sequences of length 32. We use the ADAM optimizer with learning rate 10^{-4}. We train for 400 training epochs each a quarter of a full epoch.

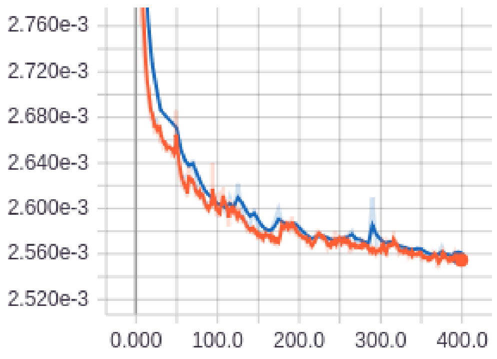


FIGURE 5.3: Reconstruction loss of PhaseNet

Results and Discussion. Figure 5.3 shows the reconstruction loss of PhaseNet. The decrease in loss steadily but very little. Perceived quality of the audio does not improve. Figure 5.4 shows that the phase-images of PhaseNet are of a different nature than phase-images of natural speech. This indicates that PhaseNet has trouble learning appropriate phases. In the audio obtained by PhaseNet has artifacts reminiscent of constant phases and actually sounds worse than random phases.

The failure of PhaseNet can mean multiple things. It may mean that the loss landscape of the spectrogram continuity loss function is highly non-convex and prone to local minima. This might be the reason that the authors of MCNN chose to augment the objective with more loss terms; so as to regularize learning.

It can also mean that $STmFT$ and/or $iSTmFT$ corrupt the gradient. In MCNN, the gradient is only backpropagated through the $STFT$ operation, not through its inverse.

The most interesting thing about PhaseNet is its loss function. Spectrogram continuity allows us to formulate a loss function *from nothing* (no data is required). However, it appears that spectrogram continuity by itself is not a strong enough teacher.

5.5 Multichannel Spectrograms (W2S-M)

In this section we propose to modify W2S-C to output spectrograms with three channels; one for magnitude, one for $sin(\phi)$ and one for $cos(\phi)$ where ϕ is the phase. This is similar to modifying image-based networks to output images with multiple color channels instead of greyscale images. By explicitly predicting phase, we don't have to estimate it from magnitude.

Model Architecture. We constrain the last two channels of each multichannel spectrogram (representing $sin(\phi)$ and $cos(\phi)$) to form a vector of length 1. We increase the dimensionality of the spectrograms from 128 to 256. (section 5.1 describes that $iSTmFT$ renders low quality for fewer spectrogram bins). To handle the larger-size input we add a layer to each of the networks. To maintain approximately the same parameter count, we halve the initial depth of each model. The converters have 7 layers (as opposed to 6) with initial depth of 5 (as opposed to 10). This increases the receptive field size to .7 seconds and the parameter count by less than 1%. The critic has 6 layers (as opposed to 5) and initial depth of 13 (as opposed to 25), which increases its parameter count from 2.2M to 2.4M.

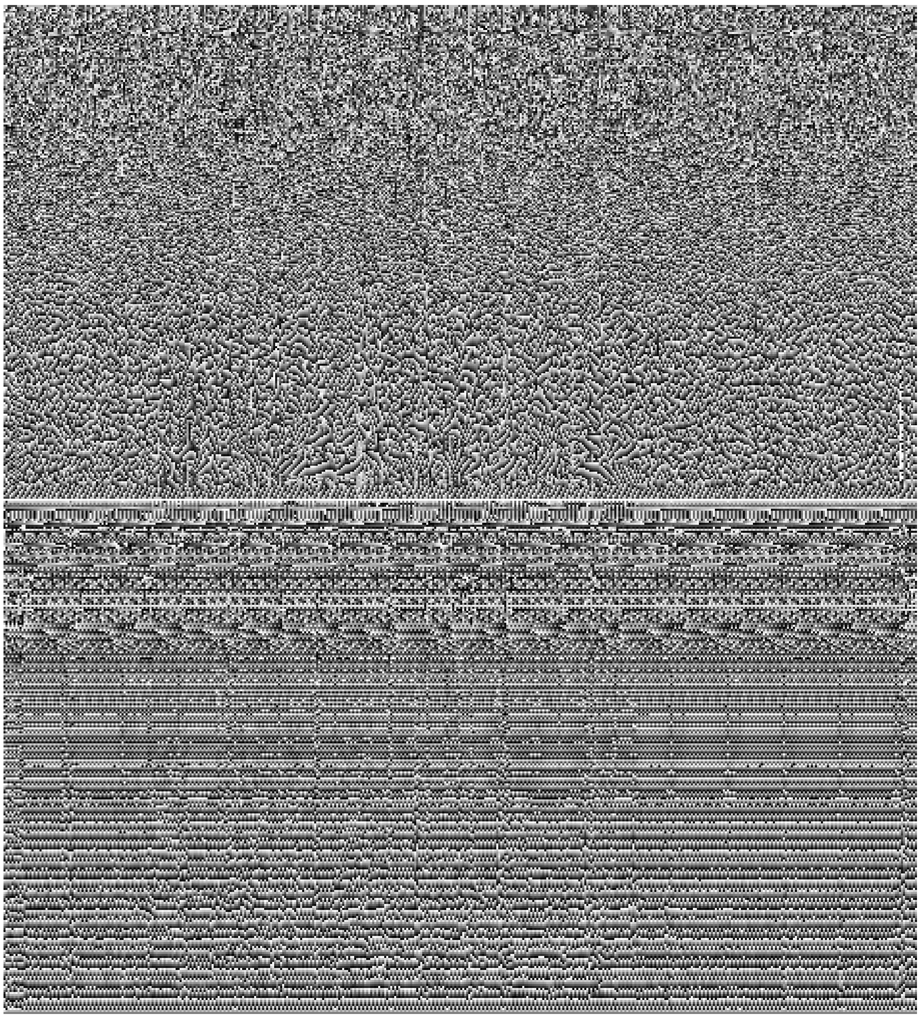


FIGURE 5.4: Phase images of natural speech (top) and of PhaseNet (bot). Best viewed electronically and zoomed in. Though at first glance the top image seems like random noise, closer inspection reveals that the scale of the artifacts is related to the frequency. The two images have very different characteristics. Note that black is $-\pi$ and white is π, which in angular values are close together.

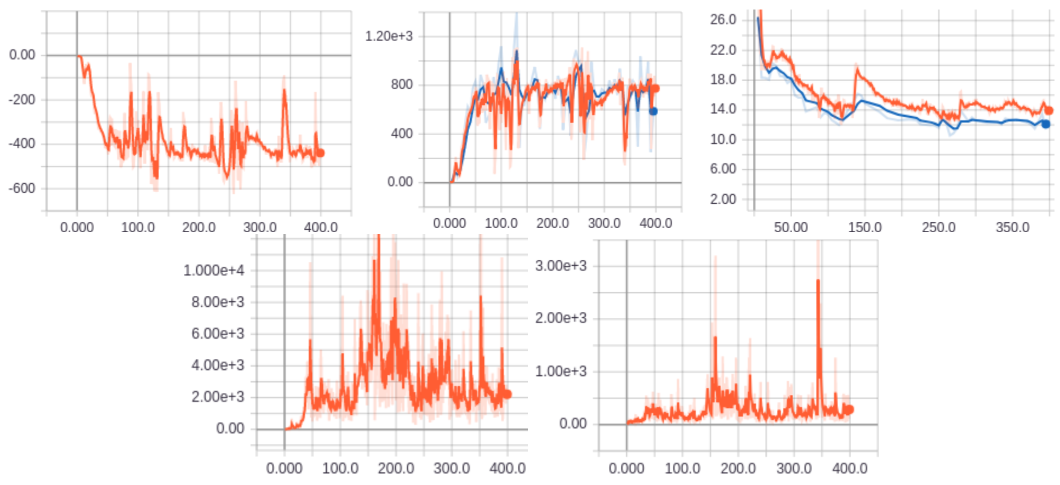


FIGURE 5.5: Convergence of the critic loss (top-left), adversarial loss (top-centre) and the reconstruction loss (top-right) of the Multichannel Convolutional WGAN model. Validation losses in blue. The bottom row shows $||\nabla_\theta \mathcal{L}_{adv}||$(bottom-left) and $||\nabla_\theta \mathcal{L}_{recon}||$ (bottom-right).

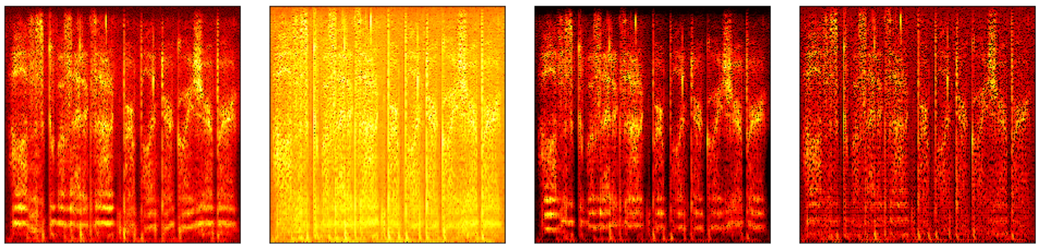


FIGURE 5.6: The output of the multichannel W2S model after 180, 260, 310 and 390 training epochs.

The model handles three-channel spectrograms everywhere except at the input of the W2S model and the output of the S2W model.These represent real and fake whisper and remain single channel spectrograms. As discussed in section 5.2, we refrain from taking (self-)supervised losses between raw phase values for the reconstruction loss term. We don't expect the phase information of the whispered signal to be informative, because it is of an unperiodic signal.

Training Procedure. The training procedure is exactly that of the single-channel W2S-C, except that we stop training after 400 training epochs.

Results and Discussion. Figure 5.5 shows that the convergence is noisy. More strikingly, figure 5.6 shows that quality of the converted items does not change steadily and that the model even unlearns desirable behaviour. Because figure 5.6 only shows the magnitude spectrogram, and not the phases, it may mean that the model has learned something valuable about phases instead, for which it has sacrificed performance on the magnitudes. Listening to the excerpts, however, tells us that quality does degrade. The model spends a lot of resources on getting phases right, which harms the performance on the magnitudes and — most importantly — on perceived quality.

We conclude that the original single-channel model W2S-C is preferred over the multi-channel counterpart W2S-M.

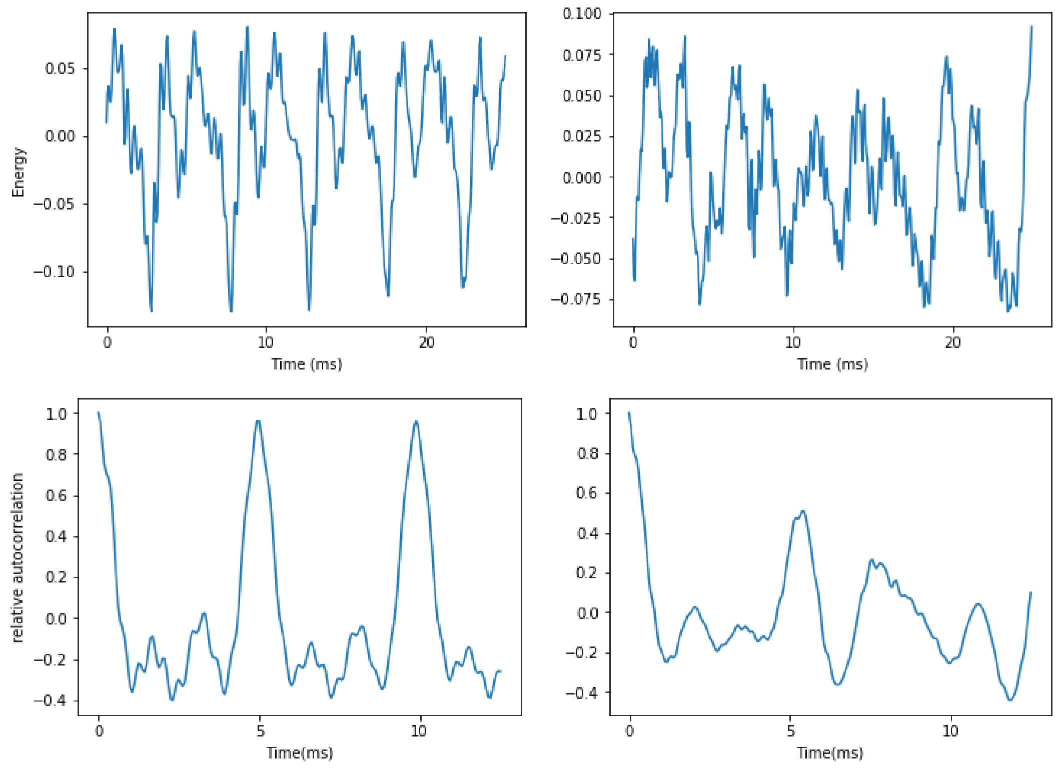


FIGURE 5.7: Audio wave of a recorded spoken item (top left) and an item converted from whispers (top right) and their autocorrelation plots (bottom).

5.6 Pitch Estimation and the Spectrogram as Filter

Periodicity of a wave can be measured by computing its autocorrelation. For voiced speech it is typical to have an autocorrelation above 90% a full period from the start of the period (where 100% is the autocorrelation at position 0). Whispered speech as converted in this work typically scores 60%. In figure 5.7 we can see changes in pattern per period, and we can also hear that the converted speech doesn't sound completely 'voiced'. A way to improve on the model is to promote periodicity during voiced phonemes.

A pulse train is a continuous signal with dirac-delta spikes at regular intervals. We will be picking the frequencies of pulse trains such that we can represent them as discrete signals that are 1 at the first sample of the period and 0 for the rest. Figure 5.8 shows that — like any periodic signal — a pulse train can be approximated as a weighted sum of sines and cosines. The phases of these sines and cosines make all frequencies interfere in such a way that they form the peaks of the pulse train. This is a very interesting property. If we apply these phases to a magnitude spectrogram, we would expect its frequencies to interfere in such a way that they peak at regular intervals like the pulse train. This would promote periodicity. Before we can do this, we need to extract a pitch contour from the spectrogram that we can construct a pulse train from.

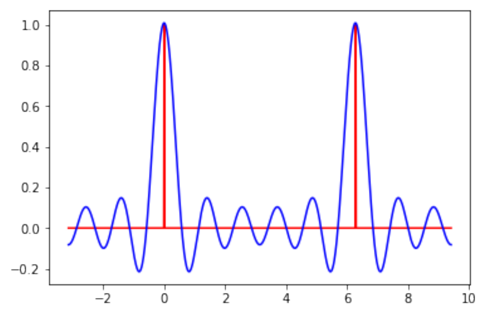


FIGURE 5.8: A pulse train can be approximated as a weighted sum of sinoids

Proposed Algorithm. We estimate the pitch contour from the magnitude spectrogram with the method of the librosa function piptrack[4]. It finds the peaks in the spectrum by fitting a quadratic function and evaluating the continuous-valued argmax. The estimated pitch is found if there is a regular interval (in the spectral dimension) between these peaks. If the confidence score of the obtained pitch is below a threshold, the frame is regarded as unpitched/unvoiced, which is denoted with a pitch estimate of 0 Hz.

We reconstruct once from a magnitude spectrogram of a real spoken example and once from a magnitude spectrogram obtained by W2S-C (converted to linear by multiplying with the mel-inversionbank). We smooth the pitch contour with a Gaussian blur over the non-zero entries and compare the reconstructions with reconstruction of the Griffin-Lim algorithm. The magnitude spectrogram and pitch contour are shown in figure 5.9. Audio can be found in the google drive folder with audio results or be obtained by running the notebook.

Discussion. The proposed algorithm has small (negligible) improvement over using purely random phases, but is no improvement over the Griffin-Lim algorithm. Neither smoothing the pitch contour nor initialising the Griffin-Lim algorithm with the phases of the pulse train seems to have any effect. Particularly in the W2S reconstruction we hear the unpleasantness of the pulse train in the reconstruction (a harsh kind of buzz). We can soften that effect somewhat by perturbing the phase values of the pulse train with Gaussian noise drawn from $\mathcal{N}(0, \frac{\pi}{4})$, but this is in effect restoring the randomness we meant to address and hurts periodicity.

[4]https://librosa.github.io/librosa/generated/librosa.core.piptrack.html

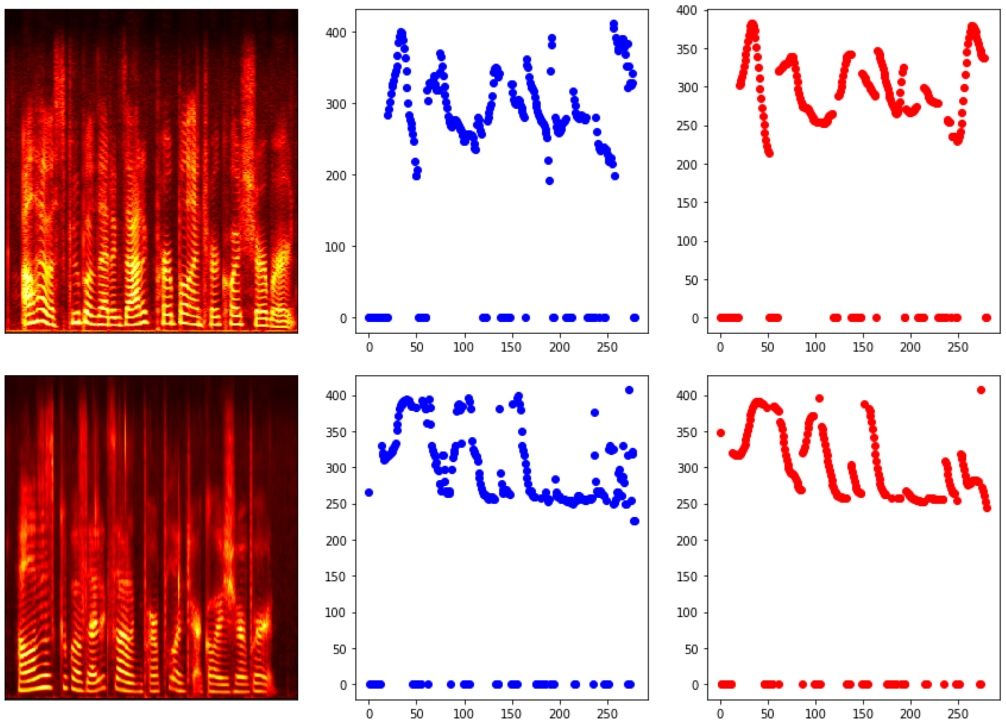


FIGURE 5.9: Spectrogram (left), pitch contour (centered) and smoothed pitch contour (right) for a spectrogram of a recorded spoken item (top) and a spectrogram obtained by W2S-C (bottom).

Chapter 6

Future Research

Chapter 5 spends much attention on spectrogram inversion, without improving over the Griffin-Lim algorithm. However, it may be that spectrogram inversion is not the quality bottleneck. Moreover, if we were to use a different representation of audio, spectrogram inversion would not be required at all.

6.1 WORLD vocoder

Spectrograms parameterize audio as the frequencies present in overlapping windows. The WORLD vocoder [39] parameterizes audio differently. It is specifically designed for real-time synthesis, and is used for Voice Conversion by Fang *et al.* [8].

WORLD parameterizes audio as pitch, harmonic spectral envelope and the aperiodic spectral envelope (relative to the harmonic spectral envelope). It exctracts pitch from the raw audio wave with the DIO algorithm [38]. The spectral envelope is estimated using CheapTrick[35], which uses both the estimated pitch and the raw audio wave. Though the original paper mentions using PLATINUM [37] to extract an excitation signal, the official WORLD vocoder website[1] and popular implementations[2] now promote the use of D4C [36] to extract the aperiodic spectral envelope. WORLD can also synthesize audio from these parameters. It is an interesting option to modify W2S-C to output these parameters rather than spectrograms.

The overlap of spectrogram windows can be a cause of latency. Before the current window can be converted, the model needs to wait for all the windows that overlap with it to have completed. Also, the uncertainty principle (section 2.2) dictates that taking too small windows hurts resolution in the frequency dimension. In this work we take spectrogram windows of 36ms with 12ms overlap. WORLD customarily takes non-overlapping 5ms windows, which incurs less latency.

For speech, the WORLD vocoder looks like a promising alternative to spectrograms in both audio quality and latency.

6.2 Flow-based Generative Modelling of the Waveform

Representing audio as a spectrogram or any vocoder's parameters means we are essentially hand-crafting features. Opposingly, feature learning or end-to-end learning dictates that we should feed the network raw data and leave it to the model to *learn* valuable features by itself. This way, the model can potentially leverage information

[1]http://www.kisc.meiji.ac.jp/ mmorise/world/english/introductions.html
[2]https://github.com/JeremyCCHsu/Python-Wrapper-for-World-Vocoder

that would otherwise have been lost in the feature extraction step (such as phase) and automatically tune the features to be most beneficial for the task at hand.

The current state of the art in audio synthesis models the raw audio wave. WaveNet [56] is an auto-regressive generative model. Though training can be done in parallel, the network must compute sequentially at test time, which makes it unable to leverage the power of parallel compute. Because many samples are needed per second (typically over 16k) and computation is sequential, WaveNet synthesis cannot be done in real-time.

Parallel WaveNet [40] and WaveGlow [45] are flow-based generative models descended from WaveNet, and allow for parallel compute and real-time performance at test time.

WaveNet. Some WaveNets' first step is to discretize the [-1, 1] range into a bins according to μ-law encoding [46] and train an embedding $x_{k=0} \in \mathbb{R}^d$ for each of the 'classes'. Another option for the first step is to perform a 1D convolution with d channels. The following 1D-convolutions have an exponentially increasing dilation factor (doubling until dilation reaches 512, after which the next dilation cycle starts). Typically WaveNet has 30 layers (3 dilation cycles/blocks of 10 layers each). Each layer has a gated activation function, a 'conditioning convolution' and a residual- and skip-connection. Specifically:

$$\mathbf{z}_k = \tanh(W_{k,f} * \mathbf{x}_k + V_{k,f} * \mathbf{y}) \odot \sigma(W_{k,g} * \mathbf{x}_k + V_{k,g} * \mathbf{y})$$

$$\mathbf{s}_k = W_{k,s} * \mathbf{z}_k$$

$$\mathbf{r}_k = W_{k,r} * \mathbf{z}_k$$

$$\mathbf{x}_{k+1} = \mathbf{r}_k + \mathbf{x}_k$$

Where $\sigma(\cdot)$ is the logistic sigmoid function, W_k are the weights of the k-th layer, $\mathbf{x}_k$ are the features of the k-th layer and $\mathbf{y}$ the sequence the network is conditioned on. Figure 6.1 shows a schematic. If the model is conditioned on a single vector $\mathbf{h}$ (global conditioning), the function becomes:

$$\mathbf{z}_k = \tanh(W_{k,f} * \mathbf{x}_k + V_{k,f}^T\mathbf{h}) \odot \sigma(W_{k,g} * \mathbf{x}_k + V_{k,g}^T\mathbf{h})$$

The final layers gather the skip-connections through summation and map them through two fully connected layers with ReLU activation. The output is an a-dimensional softmax vector, 'classifying' which discretized value is next in the sequence. Another option is to output the parameters of a mixture of logistics (MoL) distribution, which is then discretized and bounded to [-1, 1]. The loss is the negative loglikelihood of the next sample.

Flow-based Generative Models. A normalizing flow [47, 54] is an inverible flavour of neural network. On the forward pass it takes in random noise $\mathbf{z}$ drawn from a known and simple probability distribution — for instance an isotropic Gaussian — and outputs a structured sample $\mathbf{x}$. When inverted, it takes in $\mathbf{x}$ (a datapoint) and outputs $\mathbf{z}$ whose probability we can easily evaluate under the known simple distribution.

$$\mathbf{z} \sim \mathcal{N}(0, \sigma^2 \cdot I)$$

$$f(\mathbf{z}) = \mathbf{x}$$

$$f^{-1}(\mathbf{x}) = \mathbf{z}$$

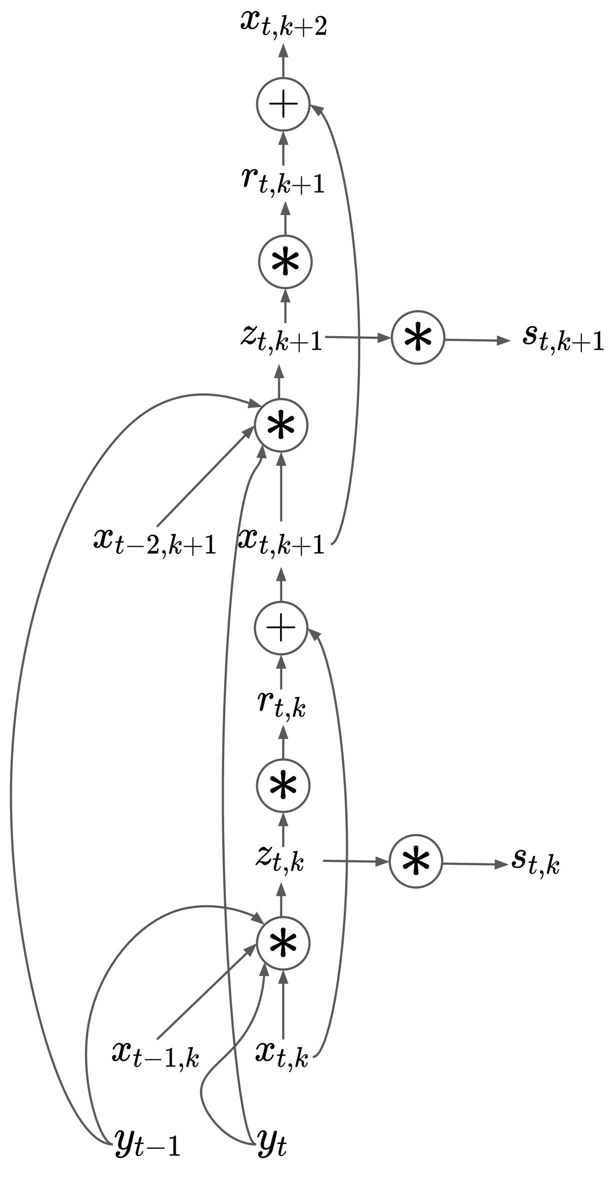


FIGURE 6.1: Two layers of WaveNet. Showing dilated convolution with conditioning, skip-connections and residual connections. Not showing the initial convolution, gated activation functions and how the skip-connections are gathered before prediction of the next sample.

The logprobability of the output $\mathbf{x}$ depends on the logprobability of the sample $\mathbf{z}$ and on the log-determinant of the Jacobian (logdet-Jacobian) of the transformation:

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) - \log |\frac{\partial \mathbf{x}}{\partial \mathbf{z}}|$$

During training, we use the network 'the other way around'. We input a sample $\mathbf{x}$ and maximise its logprobability. Because we chose p_z to be a simple distribution, $p_z(\mathbf{z})$ can readily be evaluated. The trick is to describe transformations who are easy to invert and whose logdet-Jacobian is easy to evaluate.

$$\mathbf{z} = f^{-1}(\mathbf{x})$$

$$\mathcal{L}(\mathbf{x}) = -\log p_x(\mathbf{x}) = -\log p_z(\mathbf{z}) + \log |\frac{\partial \mathbf{x}}{\partial \mathbf{z}}| = -\frac{\mathbf{z}^T\mathbf{z}}{2\sigma^2} + \log |\frac{\partial \mathbf{x}}{\partial \mathbf{z}}|$$

Many such transformations can be stacked, which allows for a very flexible probability distribution p_x. We use the shorthand $f_K \circ f_{K-1}(\mathbf{z})$ for $f_K(f_{K-1}(\mathbf{z}))$:

$$\mathbf{x} = f_K \circ f_{K-1} \circ ... \circ f_1(\mathbf{z})$$

$$\log p_x(\mathbf{x}) = \log p_z(\mathbf{z}) - \sum_{k=1}^{K} \log |\frac{\partial f_k}{\partial z_k}|$$

An example of a transformation with a tractable logdet-Jacobian is a linear transformation [47]:

$$\mathbf{x} = f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

$$|\frac{\partial \mathbf{x}}{\partial \mathbf{z}}| = |1 + \mathbf{u}^T h'(\mathbf{w}^T\mathbf{z} + b)\mathbf{w}|$$

Where h is a smooth element-wise non-linearity with derivative h'.

Parallel WaveNet. Inverse Autoregressive Flow (IAF) [25] is a type of normalizing flow that can compute conditional probability distributions in parallel. Autoregressive operations only depend on previous values, which make their determinant lower-triangular (and their inverse upper-triangular). The logdet-Jacobian of a triangular matrix is the product of its diagonal elements.

$$x_t = f(z_t, \mathbf{z}_{<t}) = z_t \cdot s(\mathbf{z}_{<t}) + \mu(\mathbf{z}_{<t})$$

$$\log |\frac{\partial \mathbf{x}}{\partial \mathbf{z}}| = \sum_t \log \frac{\partial f(\mathbf{z}_{\leq \mathbf{t}})}{\partial z_t}$$

For training, however, z_t must be computed sequentially[3]:

$$z_t = f^{-1}(x_t, \mathbf{z}_{<t}) = \frac{\mu(\mathbf{z}_{<t}) - x_t}{s(\mathbf{z}_{<t})}$$

Instead, the authors of Parallel WaveNet [40] propose Probability Density Distillation. The 'student' is an IAF network with WaveNet architecture that takes as input unconditional logistic noise $\mathbf{z}$ and outputs a structured sample $\mathbf{x}$. It also outputs the parameters s_t and μ_t that define the student's probability distribution $p_S(x_t)$ for each

[3]This is the opposite problem as traditional WaveNet. Traditional WaveNet can train in parallel but must synthesize sequentially, IAF can synthesize in parallel, but must train sequentially.

element of **x**. Because the architecture is that of a traditional WaveNet, it can be conditioned on things such as speaker identity, phonemes and/or spectrograms. The sample **x** is shown to a pre-trained traditional WaveNet (the teacher), which outputs the target probability distribution $p_T(x_t)$. The student is trained to minimise the KL-divergence between $p_S(x_t)$ and $p_T(x_t)$. The KL-divergence is split up into its Entropy and Cross-Entropy parts:

$$D_{KL}(p_S||p_T) = H(p_S, p_T) - H(p_S)$$

The Entropy term can be evaluated with the parameter s_t alone. For the Cross-Entropy term, many samples are drawn from $x_t \sim p_S(x_t|\mathbf{x}_{<t})$ (keeping $\mathbf{x}_{<t}$ fixed) and evaluated under $p_T(x_t|\mathbf{x}_{<t})$.

The power loss is an additional loss function that drives the output **x** to have the same spectral content as a sample **y**. Where **x** must have been generated under the same conditions as **y** and $\phi(\cdot)$ is averaged over time.

$$\mathcal{L}_{power} = \phi(f(x)) + \phi(y)$$

$$\phi(x) = |STFT(x)|^2$$

By leveraging parallel compute, Parallel WaveNet achieves real-time performance at test time, with virtually no loss in perceived quality as opposed to traditional WaveNet.

WaveGlow. Prenger *et al.* propose WaveGlow [45] as a real-time spectrogram inversion network. It uses invertible 1×1 convolutions and affine normalizing flows. The parameters of the affine layers are estimated by WaveNet layers.

A 1×1 invertible convolution can be regarded as a matrix multiplication. The weight matrix W must be initialized as orthonormal (and hence invertible). The addition of its logdet-Jacobian to the loss function serves to keep W invertible.

$$f_k^{-1}(\mathbf{x}_{k+1}) = \mathbf{x}_k = W\mathbf{x}_{k+1}$$

The logdet-Jacobian of f_k is the log-determinant of W itself and cannot be further simplified.

The affine layer splits the depth of the incoming vector x in two, and uses one half to compute the parameters that modify the other half.

$$\mathbf{x}_{k+1,a}, \mathbf{x}_{k+1,b} = split(\mathbf{x}_{k+1})$$

$$\mathbf{s}, \mathbf{t} = WN(\mathbf{x}_{k+1,a})$$

$$\mathbf{x}'_{k+1,b} = \mathbf{s} \odot \mathbf{x}_{k+1,b} + \mathbf{t}$$

$$\mathbf{x}_k = f_k^{-1}(\mathbf{x}_{k+1}) = concat(\mathbf{x}_{k+1,a}, \mathbf{x}'_{k+1,b})$$

$$\log|\frac{\partial f^{-1}(\mathbf{x}_{k+1})}{\partial \mathbf{x}_k}| = \log|S_t|$$

Where S is a diagonal matrix whose entries are $\mathbf{s}_t$. Its determinant is the product of the diagonal elements. For brevity, we don't show temporal indices, but the input of WaveNet is $\mathbf{x}_{<t}$, whereas all other vectors only consider the t-th timepoint.

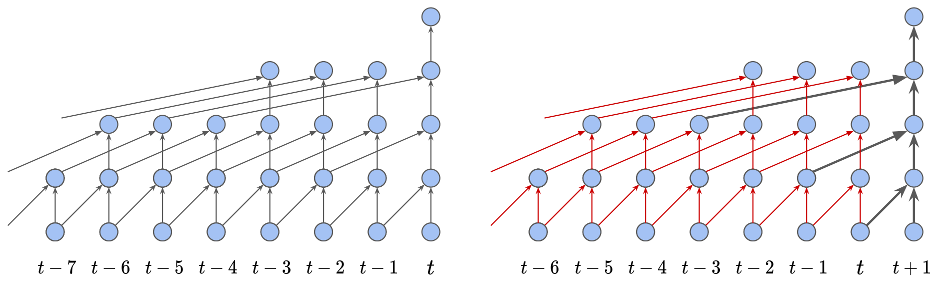


FIGURE 6.2: WaveNet computes features for all timepoints and all layers. During the next forward pass, most operations are redundant, which are here shown in red.

Invertibility of the network is guaranteed, because $\mathbf{x}_a$ is unchanged through the affine transformation. At test time, we use $\mathbf{s}$ and $\mathbf{t}$ to perform the inverse of the affine transform, but we predict them using the same (non-invertible) WaveNet.

$$\mathbf{x}_{k+1,a}, \mathbf{x}'_{k+1,b} = split(\mathbf{x}_k)$$

$$\mathbf{s}, \mathbf{t} = WN(\mathbf{x}_{k+1,a})$$

$$\mathbf{x}_{k+1,b} = \frac{\mathbf{x}'_{k+1,b} - \mathbf{t}}{\mathbf{s}}$$

$$\mathbf{x}_{k+1} = f_k(\mathbf{x_k}) = concat(\mathbf{x}_{k+1,a}, \mathbf{x}_{k+1,b})$$

WaveGlow is made up of 12 invertible convolution layers and 12 affine layers, where each affine layer contains an 8-layer WaveNet with 512 channels used for the residual connection and 256 channels for the skip connection ($\mathbf{x}, \mathbf{r}, \mathbf{z} \in \mathbb{R}^{512}$ and $\mathbf{s} \in \mathbb{R}^{256}$).

6.3 Preventing Redundant Computations

W2S-C uses the same dilated convolutional structure as WaveNet. At test time, they process spectrograms column-by-column, similar to how WaveNet processes sample by sample (the difference is that W2S-C is not autoregressive). Figure 6.2 shows that when processing input per timestep, this convolutional structure performs redundant computations [42]. By storing the values instead of recomputing them, we trade storage for speed, which may help to achieve real-time performance.

Chapter 7

Discussion

The user study of chapter 4 shows that intelligibility and quality of the models in this thesis are not good enough for a production-level application. However, this does not imply a total failure. The emergence of pitch, periodicity and prosody from completely unvoiced input is remarkable.

One of the main benefits of the methods in this thesis is that the adversarial paradigm does not require paired data. Neither does training the audio-to-audio mapping require transcribed data. These factors make data acquisition a lot easier. An additional benefit of the adversarial paradigm is that there is no need for Dynamic Time Warping (DTW), which can be source of data contamination.

The W2S-R model is an LSTM model augmented with pre-trained RBMs at the outermost layers. It improves over W2S-L, which is just the LSTM without RBM pretraining. Figure 3.8 shows that reconstructing speech with just the speech-RBM degrades quality. The quality is similar to that of a converted whisper by W2S-R (shown in 3.9). This suggests that the speech-RBM is the limiting factor. Perhaps unsurprisingly; The vast majority of the parameters reside in the LSTM module. A balance favouring more parameters to the RBM layers may help increase quality of the W2S-R model.

Chapter 4 shows that it is beneficial to train speaker embeddings. This way, the model can synthesize a voice that the user may recognize as her own. The embeddings encode meaningful features with which characteristics such as speaker gender and nationality can be distinguished.

The models in this thesis use spectrograms to represent audio. Spectrogram inversion can be done in many ways, and is touched upon at length in chapter 5. However, it may be the case that spectrogram inversion is not the quality bottleneck. If a predicted spectrogram is low quality, a good spectrogram inversion will still render low quality audio. This fact is supported by high-quality Griffin-Lim reconstructions of magnitude spectrograms drawn from natural speech (section 5.6).

Perhaps focus can best be spent on predicting better spectrograms by data preprocessing and augmentation. By denoising spoken but not whispered data, W2S will be trained to perform speech denoising in addition to conversion.

By using either the WORLD vocoder or WaveNet-type models, spectrogram inversion will not be required at all. How to modify the pipeline for WaveNet-type models other than for spectrogram inversion is an open problem and an interesting area of research.

Appendix A

Exploratory Experiment with WaveNet

To get a feel for what it takes to train WaveNet, we train one to perform spectrogram inversion. The choices and circumstances of this experiment are not optimal and the experiment was not successful.

We show the network real speech waveforms (16 kHz) and predict the next value in the sequence, conditioned on the spectrogram.

Model Architecture. We do not discretize the input wave, but rather convolve with 128 channels. The dimensionality of the feature-, residual- and skip-connections is set to 128 ($\mathbf{x}, \mathbf{r}, \mathbf{z}, \mathbf{s} \in \mathbb{R}^{128}$). The WaveNet has 9 layers, which brings the parameter count to about 1.2M and the receptive field size to 1024. Although the receptive field size is small, it is larger than the size of a spectrogram window (540). We predict the parameters of a MoL distributions with 10 mixtures and discretize the [-1, 1] range into 256 equal bins.

Training Procedure. We train WaveNet with the ADAM optimizer with learning rate 10^{-4}. We use Exponential Moving Average (EMA) with decay .999 over the parameters[1]. We train for 2000 training epochs of 600 items. This takes about 12 days on a single K20 GPU-node. Convergence is shown in figure A.1.

Results and Discussion. We observe that the model preferably outputs pure sinusoids. This is probably the easiest signal to synthesize. Synthesizing can either be done by taking the argmax of the predicted MoL distribution or by sampling from it. The output is always sinusoidal when synthesizing by argmax. With sampling, the

[1]The teacher of Parallel WaveNet is trained with Polyak averaging [44]. EMA behaves like Polyak in the limit as decay approaches 1

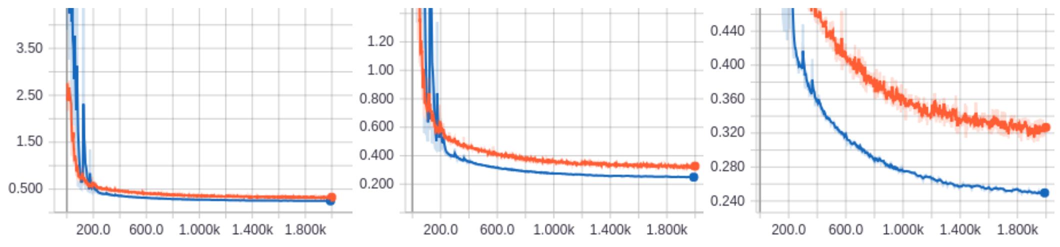


FIGURE A.1: Convergence of WaveNet loss. Though learning appears to stagnate, significantly zooming in reveals that the loss is still decreasing.

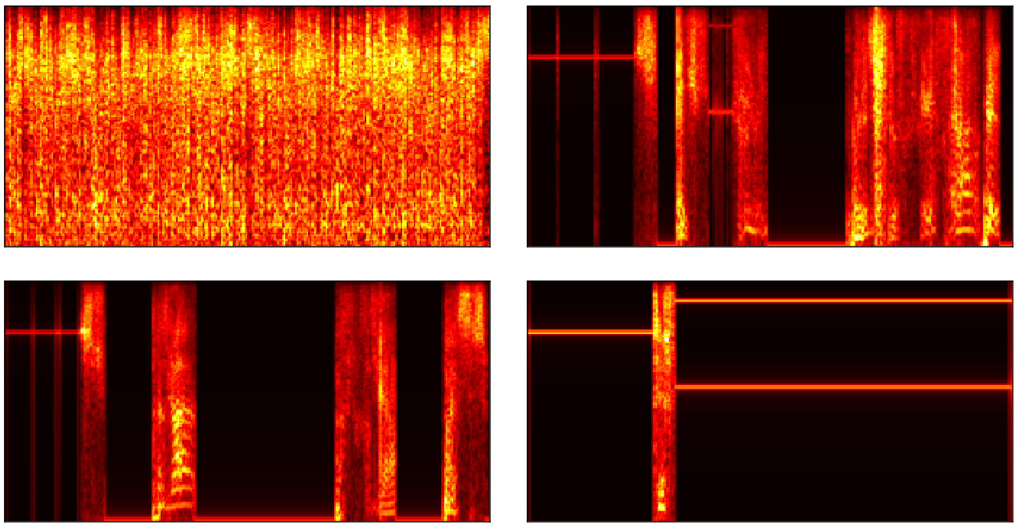


FIGURE A.2: Spectrograms of WaveNet output after 90 (top left), 600 (top right), 840 (bottom left) and 2000 (bottom right) epochs.

model sometimes outputs speech-like artifacts (shown in figure A.2), indicating that it is learning something about speech. However, as training continues, the model increasingly prefers sinusoids. The model seems to mostly ignore the spectrogram input. In general, this WaveNet is not a succes.

Our WaveNet has especially few parameters, which may explain its failure. A typical WaveNet has 512 or 256 channels where we use 128, and 30 layers where we use 9. Though originally WaveNets have a kernel size of 2, some more recent ones use a kernel size of 3. However, it should be noted that the authors of Tacotron 2 succesfully train a spectrogram inversion WaveNet with as few as 12 layers. They mention that the information-rich spectrogram conditioning helps synthesize quality speech.

Aside from parameter limitations, hardware limitations may have also played a role. The authors of TacoTron 2 mention that their WaveNet is trained across 32 GPUs. The authors of WaveNet and Parallel WaveNet are unclear about their hardware use.

We update the Exponential Moving Average every batch. Perhaps it is better to update every epoch, which would not favour more recent parameters as much.

Training a proper WaveNet for spectrogram inversion (likely involving more parameters and a WaveGlow or Parallel WaveNet descendant) remains an experiment outside the scope of this thesis. However, by the notion that spectrogram inversion may not be the quality bottleneck of whisper-to-speech conversion, leveraging raw audio waveform modelling for something else than spectrogram inversion is also an interesting vein of future research.

Appendix B

Additional runs for W2S-C

Section 3.3 discusses the W2S-C model. One could argue that figure 3.13 does not quite show the loss to have converged. Figure B.1 shows training for another 600 training epochs (totalling 1200 training epochs). Perceived quality does not improve over the 600-point.

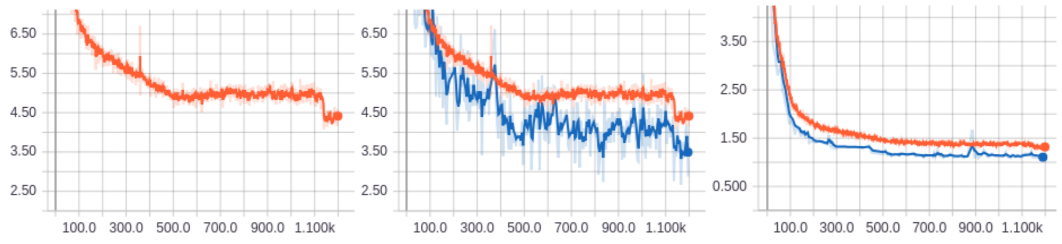


FIGURE B.1: Convergence of the critic loss (left), adversarial loss (centre) and recon loss (right) of W2S-C, continued for another 600 epochs, totalling 1200 epochs.

Appendix C

Additional runs for W2S-embed

Section 4.2 discusses the W2S-embed model. One could argue that figure 4.9 does not quite show that the network has converged. Figure C.1 shows that loss keeps decreasing after the 600-epoch point and up to the 1000-epoch point, where it still does not converge. Perhaps the loss may drop yet further. Difference between perceived quality of the 1000-epoch and 600-epoch versions is negligible. User-study results obtained from the 600-epoch version are still valid.

Another instance of the run (the same architecture under the same conditions, with different random initialisation) trained for 1000 training epochs shows a peculiar learning profile (figure C.2). The loss hits a strong local minimum around the 400th epoch and stops improving. Perceived quality does not improve over the W2S-embed model of section 4.2 or the W2S-C model of section 3.3. This highlights that training is not especially robust.

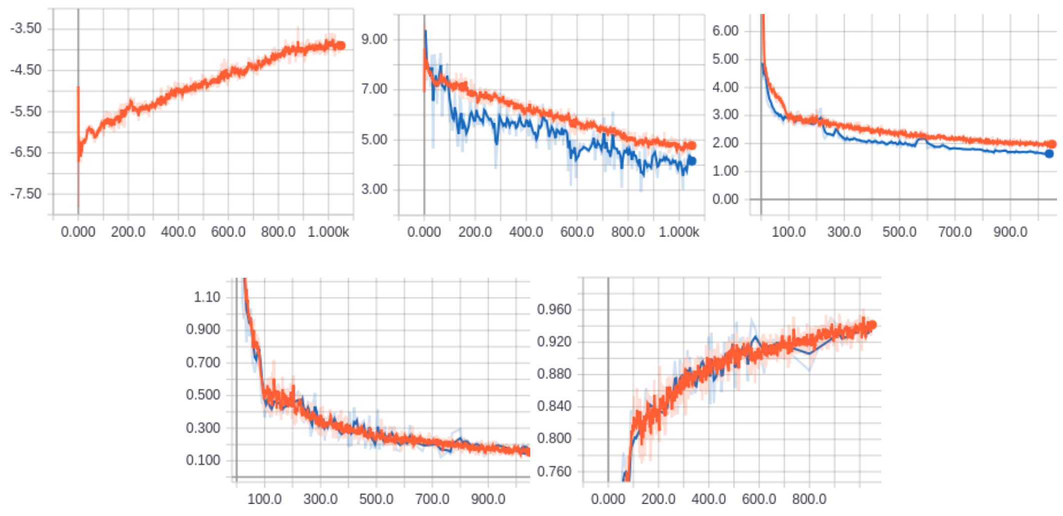


FIGURE C.1: Convergence of the critic loss (top left), adversarial loss (top centre), reconstruction loss (top right), identifier loss (bottom left) and identifier accuracy (bottom right) for the continued W2S-embed run.

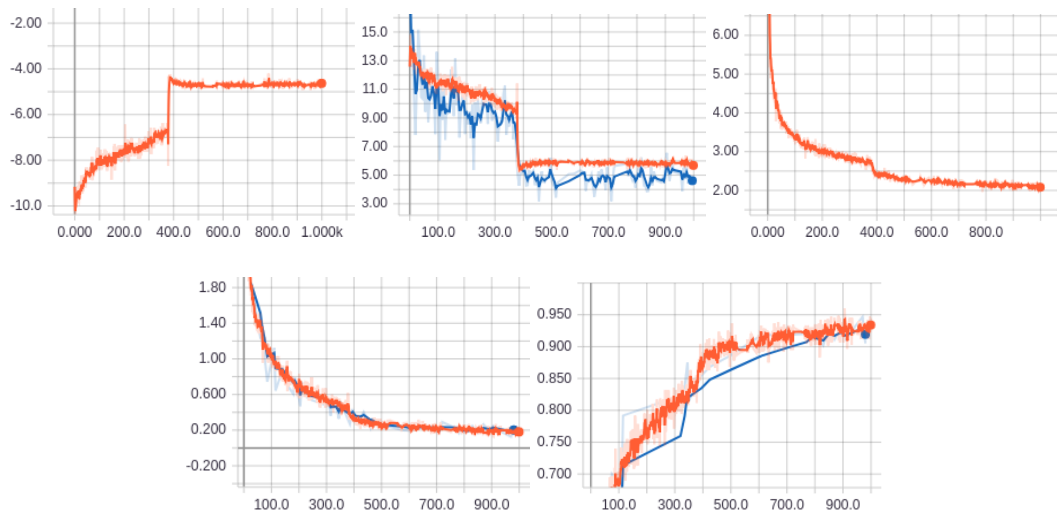


FIGURE C.2: Convergence of the critic loss (top left), adversarial loss (top centre), reconstruction loss (top right), identifier loss (bottom left) and identifier accuracy (bottom right) for the additional W2S-embed run with different random initialisation.

Bibliography

[1] Sercan Arik et al. “Neural voice cloning with a few samples”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 10019–10029.

[2] Sercan Ö Arık, Heewoo Jun, and Gregory Diamos. “Fast spectrogram inversion using multi-head convolutional neural networks”. In: *IEEE Signal Processing Letters* 26.1 (2019), pp. 94–98.

[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. “Wasserstein generative adversarial networks”. In: *International Conference on Machine Learning*. 2017, pp. 214–223.

[4] Donald J Berndt and James Clifford. “Using dynamic time warping to find patterns in time series.” In: *KDD workshop*. Vol. 10. 16. Seattle, WA. 1994, pp. 359–370.

[5] James W Cooley and John W Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of computation* 19.90 (1965), pp. 297–301.

[6] Marco Cuturi and Mathieu Blondel. “Soft-DTW: a differentiable loss function for time-series”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 894–903.

[7] Rémi Decorsière et al. “Inversion of auditory spectrograms, traditional spectrograms, and other envelope representations”. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 23.1 (2014), pp. 46–56.

[8] Fuming Fang et al. “High-quality nonparallel voice conversion based on cycle-consistent adversarial network”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 5279–5283.

[9] Gene A Frantz and Richard H Wiggins. “Design case history: Speak & Spell learns to talk”. In: *IEEE spectrum* 19.2 (1982), pp. 45–49.

[10] Ian Goodfellow et al. “Generative adversarial nets”. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680.

[11] Daniel Griffin and Jae Lim. “Signal estimation from modified short-time Fourier transform”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32.2 (1984), pp. 236–243.

[12] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5767–5777.

[13] Yanzhang He et al. “Streaming End-to-end Speech Recognition For Mobile Devices”. In: *arXiv preprint arXiv:1811.06621* (2018).

[14] Geoffrey E Hinton. “A practical guide to training restricted Boltzmann machines”. In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 599–619.

[15] Geoffrey E Hinton. “Training products of experts by minimizing contrastive divergence”. In: *Neural computation* 14.8 (2002), pp. 1771–1800.

[16] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[17] Roger J Ingham et al. "Measurement of speech effort during fluency-inducing conditions in adults who do and do not stutter". In: *Journal of Speech, Language, and Hearing Research* (2009).

[18] Phillip Isola et al. "Image-to-image translation with conditional adversarial networks". In: *arXiv preprint* (2017).

[19] Andreas Jansson et al. "Singing voice separation with deep U-Net convolutional networks". In: (2017).

[20] Ye Jia et al. "Direct speech-to-speech translation with a sequence-to-sequence model". In: *arXiv preprint arXiv:1904.06037* (2019).

[21] Ye Jia et al. "Transfer learning from speaker verification to multispeaker text-to-speech synthesis". In: *Advances in Neural Information Processing Systems*. 2018, pp. 4480–4490.

[22] Hideki Kawahara, Ikuyo Masuda-Katsuse, and Alain De Cheveigne. "Restructuring speech representations using a pitch-adaptive time–frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds". In: *Speech communication* 27.3-4 (1999), pp. 187–207.

[23] Hyunsoo Kim and Haesun Park. "Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method". In: *SIAM journal on matrix analysis and applications* 30.2 (2008), pp. 713–730.

[24] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[25] Durk P Kingma et al. "Improved variational inference with inverse autoregressive flow". In: *Advances in neural information processing systems*. 2016, pp. 4743–4751.

[26] Hideaki Konno et al. "Whisper to normal speech conversion using pitch estimated from spectrum". In: *Speech Communication* 83 (2016), pp. 10–20.

[27] Daniel D Lee and H Sebastian Seung. "Algorithms for non-negative matrix factorization". In: *Advances in neural information processing systems*. 2001, pp. 556–562.

[28] Xiangang Li and Xihong Wu. "Modeling speaker variability using long short-term memory networks for speech recognition". In: *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[29] Boon Pang Lim. "Computational differences between whispered and non-whispered speech". PhD thesis. University of Illinois at Urbana-Champaign, 2011.

[30] Yoshiki Masuyama et al. "Deep Griffin-Lim Iteration". In: *arXiv preprint arXiv:1903.03971* (2019).

[31] Ian McLoughlin et al. "Speech reconstruction using a deep partially supervised neural network". In: *Healthcare technology letters* 4.4 (2017), pp. 129–133.

[32] G Nisha Meenakshi and Prasanta Kumar Ghosh. "A robust Voiced/Unvoiced phoneme classification from whispered speech using the 'color'of whispered phonemes and Deep Neural Network". In: *Proc. Interspeech 2017* (2017), pp. 503–507.

[33] Paul Mermelstein. “Distance measures for speech recognition, psychological and instrumental”. In: *Pattern recognition and artificial intelligence* 116 (1976), pp. 374–388.

[34] Yajie Miao, Hao Zhang, and Florian Metze. “Speaker adaptive training of deep neural network acoustic models using i-vectors”. In: *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 23.11 (2015), pp. 1938–1949.

[35] Masanori Morise. “CheapTrick, a spectral envelope estimator for high-quality speech synthesis”. In: *Speech Communication* 67 (2015), pp. 1–7.

[36] Masanori Morise. “D4C, a band-aperiodicity estimator for high-quality speech synthesis”. In: *Speech Communication* 84 (2016), pp. 57–65.

[37] Masanori Morise. “Platinum: A method to extract excitation signals for voice synthesis system”. In: *Acoustical Science and Technology* 33.2 (2012), pp. 123–125.

[38] Masanori Morise, Hideki Kawahara, and Haruhiro Katayose. “Fast and reliable F0 estimation method based on the period extraction of vocal fold vibration of singing voice and speech”. In: *Audio Engineering Society Conference: 35th International Conference: Audio for Games*. Audio Engineering Society. 2009.

[39] Masanori Morise, Fumiya Yokomori, and Kenji Ozawa. “WORLD: a vocoder-based high-quality speech synthesis system for real-time applications”. In: *IEICE TRANSACTIONS on Information and Systems* 99.7 (2016), pp. 1877–1884.

[40] Aaron van den Oord et al. “Parallel WaveNet: Fast high-fidelity speech synthesis”. In: *arXiv preprint arXiv:1711.10433* (2017).

[41] Douglas O’Shaughnessy. “Linear predictive coding”. In: *IEEE potentials* 7.1 (1988), pp. 29–32.

[42] Tom Le Paine et al. “Fast wavenet generation algorithm”. In: *arXiv preprint arXiv:1611.09482* (2016).

[43] Vassil Panayotov et al. “Librispeech: an ASR corpus based on public domain audio books”. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2015, pp. 5206–5210.

[44] Boris T Polyak and Anatoli B Juditsky. “Acceleration of stochastic approximation by averaging”. In: *SIAM Journal on Control and Optimization* 30.4 (1992), pp. 838–855.

[45] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. “Waveglow: A flow-based generative network for speech synthesis”. In: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, pp. 3617–3621.

[46] CCITT Recommendation. “Pulse code modulation (PCM) of voice frequencies”. In: *ITU*. 1988.

[47] Danilo Jimenez Rezende and Shakir Mohamed. “Variational inference with normalizing flows”. In: *arXiv preprint arXiv:1505.05770* (2015).

[48] Hamid Sharifzadeh et al. “Formant smoothing for quality improvement of post-laryngectomised speech reconstruction”. In: *Orange Technologies (ICOT), 2017 International Conference on*. IEEE. 2017, pp. 11–14.

[49] Hamid R Sharifzadeh et al. “A training-based speech regeneration approach with cascading mapping models”. In: *Computers & Electrical Engineering* 62 (2017), pp. 601–611.

[50] Jonathan Shen et al. "Natural TTS synthesis by conditioning wavenet on mel spectrogram predictions". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 4779–4783.

[51] Malcolm Slaney. "Auditory toolbox". In: *Interval Research Corporation, Tech. Rep* 10.1998 (1998).

[52] Daniel Stoller, Sebastian Ewert, and Simon Dixon. "Adversarial semi-supervised audio source separation applied to singing voice extraction". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 2391–2395.

[53] Ilya Sutskever and Tijmen Tieleman. "On the convergence properties of contrastive divergence". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 789–795.

[54] EG Tabak and Cristina V Turner. "A family of nonparametric density estimation algorithms". In: *Communications on Pure and Applied Mathematics* 66.2 (2013), pp. 145–164.

[55] Shinnosuke Takamichi et al. "Phase reconstruction from amplitude spectrograms based on von-Mises-distribution deep neural network". In: *2018 16th International Workshop on Acoustic Signal Enhancement (IWAENC)*. IEEE. 2018, pp. 286–290.

[56] Aäron Van Den Oord et al. "Wavenet: A generative model for raw audio". In: *CoRR abs/1609.03499* (2016).

[57] Sean Vasquez and Mike Lewis. "MelNet: A Generative Model for Audio in the Frequency Domain". In: *arXiv preprint arXiv:1906.01083* (2019).

[58] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media, 2008.

[59] Xiang Wei et al. "Improving the improved training of wasserstein gans: A consistency term and its dual effect". In: *arXiv preprint arXiv:1803.01541* (2018).