Disentangled Speaker Representations in Neural Text-to-Speech Synthesis

Richard Sterry[1]

MSc Computational Statistics & Machine Learning

Academic Supervisor: Dr. Mark Herbster
Industry Supervisor: Jiameng Gao

Submission date: 7 September 2018

Abstract

Current production Text-to-Speech (TTS) synthesis technologies are based on heavily-engineered pipelines of components that incorporate huge amounts of domain knowledge from linguistics, phonetics and signal processing. In the past two years, there have been rapid advances in alternative TTS approaches based on neural sequence-to-sequence models, which are trained end-to-end on raw data with very little domain-specific feature engineering. A key advantage of neural TTS models is that a single model that can learn the voices of many speakers, by representing each speaker's voice as a vector in a fixed-dimensional embedding space.

In recent years, there has also been a large volume of research into using deep neural networks to learn disentangled representations of data, in which the dimensions of the representation space are encouraged to align to independent generative attributes of the data.

In my project, I investigate whether disentanglement techniques can be applied to a neural TTS system to force it to learn a disentangled speaker representation, in which dimensions of the speaker embedding space align to generative attributes such as the speaker's gender, accent and age. Using FAIR's VoiceLoop neural TTS system as my baseline, I investigate two disentanglement methods: Fader Networks, a supervised method employing an adversarial network with parallels to GANs; and β-VAE, an unsupervised method that uses a VAE with a variable bottleneck. I find that the Fader Network approach can disentangle the gender attribute, giving excellent control over the perceived gender of generated speech by varying a single factor in the speaker embedding. The β-VAE approach is targeting a much tougher unsupervised problem, but is still able to achieve a good degree of gender disentanglement.

A companion website to this report, including lots of audio samples, can be found here: `https://richardsterry.github.io/msc-project/`

Contents

Part I

Introduction

For my Masters project, I've been investigating how well the individual characteristics of different people's voices can be represented in Text-to-Speech (TTS) synthesis models based on sequence-to-sequence neural networks (as opposed to traditional feature-engineered systems). In particular, I've investigated whether we can force these models to learn speaker representations in which key interpretable attributes such as gender and accent are **disentangled** within the embedding space.

In chapter 1, I give a brief introduction to the rich and venerable field of speech synthesis: why it's interested researchers and engineers for centuries; the main approaches used in existing production speech technology; and how the field may be on the cusp of a revolution as a result of progress in sequence to sequence models.

In chapter 2, I define disentanglement and review recent advances in the literature.

In chapter 3, I introduce my project. I explain the guiding aim of my project, how I developed the aim into more concrete objectives, and then talk through the structure of the rest of my report.

Chapter 1

From Bellows to Batch Norm: a Brief History of Speech Synthesis

It's easy to take human communication for granted. We become so familiar with expressing our thoughts and intentions to other humans, and with understanding the thoughts and intentions of others, that communication can seem a mere tool that is a slave to the content it transmits. On reflection, though, communication is truly a marvel. That we can signal our desires and intentions across space and time with extremely high fidelity, with the ability to impart knowledge and arouse precise emotions, through such constrained channels as waves of air pressure or static marks on a page, is almost a miracle. The ability to communicate is essentially human, and is the wedge that humans have used to drive apart from the rest of the animal kingdom despite our genetic similarities and physical weaknesses.

This, I think, is why there's something visceral about communicating with a machine on human terms. We normally treat machines as tools rather than agents. But when we communicate directly with a machine, particularly through speech, we begin to treat the machine as an agent and interpret it in human terms of motives, goals and understanding. The study of affective computing and the interaction between humans and machines is, in my opinion, one of the most fascinating areas of computer science, and will grow increasingly important in coming years as technology becomes ever more embedded in the fabric of human life.

Humans have long been drawn to the challenge of building systems that can generate intelligible human speech: speech synthesis, or Text-to-Speech (TTS) synthesis. In section 1.1 we will encounter some fascinating examples of early speech synthesis machines.

Of course, as well as being a seductive challenge, speech synthesis has proven to be hugely useful and applicable to real-world problems:

- **Medical**: Giving a voice to those who no longer have the ability to speak, i.e. speaking *for*

people. Stephen Hawking was the most famous user, but many others have, and continue to, benefit from communication devices that convert their text into speech.

- **Accessibility**: Providing quick and pleasant access to written material for people who have sight or learning difficulties that impair their ability to read, i.e. speaking *to* people.
- **Lifestyle/Safety**: Satellite navigation systems; personal assistants (Alexa, Siri); automated telephone systems; audio books; and no doubt many, many more in coming years.

In section 1.2 I briefly discuss the traditional speech synthesis techniques that provide the basis for essentially all such production speech technology today.

In the past couple of years, there has been rapid progress in applying sequence-to-sequence neural networks to speech synthesis, resulting in simpler pipelines and hugely impressive results: I will introduce this literature in section 1.3.

Finally, in section 1.4 I will highlight one of the key aspects in which neural approaches may revolutionise speech technology: the ability to train a single model that can generate speech in many different speakers' voices.

1.1 The Early Days

Although not the earliest example of a machine designed to produce speech sounds, the 'acoustic-mechanical speech machine' built by **Wolfgang von Kempelen** between 1769 and 1790 is one of the most iconic[24]. The machine could produce consonant and vowel sounds by passing air flow controlled by bellows (to simulate the lungs) through mechanical models of the human vocal tracts, tongue and lips. By pulling levers and other devices, a skilled operator could generate recognisable simple sentences. This tremendous feat of engineering captured the imaginations of such luminaries as Sir Charles Wheatstone, who built an improved version in 1837, and Alexander Graham Bell, who was inspired to experiment with speaking machines after seeing Wheatstone's machine at an exhibition[1].

By the time of the New York World's Fair in 1939, the state-of-the-art speaking machine was Bell Lab's **Voder**[2]. Although it replaced von Kemplen's bellows with electricity, the Voder was still essentially an instrument, and required a highly-skilled human operator to coordinate multiple keys, bars and pedals to produce recognisable sounds. Fascinating archive footage of the 1939 demonstration shows the skill of the operator and the impressive quality of the speech.

As the era of modern computers dawned, there was renewed interest in synthesizing speech, with the huge advancement of being able to synthesize directly from text rather than via a human operator. I would recommend watching at least some of this 1984 documentary[3] on speech technology to appreciate how impressive TTS results already were almost a third of a century ago[4].

[1] `https://en.wikipedia.org/wiki/Wolfgang_von_Kempelen%27s_speaking_machine`
[2] `https://en.wikipedia.org/wiki/Voder`
[3] `https://www.youtube.com/watch?v=OXB9v3z22MI&index=12&list=RD5hyI_dM5cGo`
[4] also, to gape at the 1980s dress sense and computer hardware.

1.2 Traditional Speech Technology

Current production speech technology is typically based on highly-engineered pipelines of systems developed over many years by experts, using extensive domain knowledge in fields such as phonetics, linguistics and signal processing. These systems are designed by analysing the structure of human language (breaking speech down into component units), by understanding the physical mechanisms through which the human vocal system generates sound waves (vocal tract, vocal chords, mouth, lips etc.), modelling many components separately, and then building the parts back together (sythesising them) to generate sound waves. To work in traditional speech technology requires a deep understanding of speech, language and signal processing.

Almost all production TTS systems fall into one of two paradigms:

Concatenative Synthesis A carefully recorded and annotated speech corpus is 'chopped up' into units that correspond to phonemes, or short sequences of phonemes (diphones or triphones). To synthesize speech for a given piece of text, the system must select the best units from its database using complex rules based on the context, prosody etc., and stitch (concatenate) the units together with smoothing applied at the edges. Concatenative synthesis can achieve excellent results on large, high-quality datasets of individual professional vocal artists.

Statistical Parametric Speech Synthesis (SPSS) SPSS can be understood in reference to a 'vocoder': a system that can *analyse* the waveform of speech into a set of **acoustic features**, and can then then *synthesize* these acoustic features back into a waveform. By identifying a set of acoustic features that provides a good representation of speech data, a vocoder can therefore compress/encode a sample of speech to features, and then decompress/decode the features back to speech.

In SPSS, a speech corpus is first analysed into acoustic features (from the speech waveforms) and linguistic features (from the transcripts of the speech waveforms). Then, a complex model is built, using extensive domain knowledge, to learn how the linguistic features map into the corresponding acoustic features. Wheres the vocoder simply reconstructs known samples, the SPSS system learns a model to map text into acoustic features. To synthesize speech for a piece of text, the system maps text to acoustic features, and then synthesizes these into a speech waveform.

SPSS is a theoretically attractive approach because it genuinely learns a model of speech from data, rather than simply stitching samples together like a highly-engineered ransom note. This can give an SPSS model more generality and possibly control. However, although SPSS can outperform concatenative synthesis on small datasets, on large, professional single-speaker datasets, concatenative synthesis tends to produce the most realistic results.

These technologies are highly mature, and can achieve the excellent results we hear from production speech technology every day. However, some argue that the technologies appear to be saturating, in the sense of approaching the limits of the quality they can achieve. As they are based on such complex pipelines of independently-trained component, hard-won improvements in individual

components often don't translate into improvements at the overall system level. It can be difficult to pin down where weaknesses are coming from: solving one problem can often just create another one somewhere else in the system. They are hard to optimize at a global level.

Researchers in traditional speech technology have been applying neural network techniques to individual components of the pipelines for years (for example [53]). However, it is only in the past couple of years that researchers have been successful in applying neural networks to the *entire, end-to-end TTS pipeline*, which may potentially lead to systems that can compete with or even dominate the concatenative and statistical parametric approaches.

1.3 Neural Approaches to TTS

Sequence-to-sequence models encode an input sequence (e.g an English sentence) into a hidden state, and then decode this state into an output sequence (e.g. a German sentence.) Both input and output sequences can be of variable length. Sequence-to-sequence models based on neural networks have shown tremendous results in domains such as machine language translation, image captioning, text summarisation and, in particular, speech recognition, i.e. **speech-to-text**.

In the past couple of years, there has also been startling success at applying neural sequence-to-sequence models to the related *inverse* problem of **text-to-speech**. Although not the only, or necessarily the first, to apply neural sequence-to-sequence models to the full TTS pipeline (see Char2Wav[45] for example), three families of models dominate the recent literature:

- **DeepVoice**, from Baidu Research
- **Tacotron**, from Google Research/Brain
- **VoiceLoop**, from Facebook AI Research

New papers have been appearing at a furious rate, with numerous major contributions appearing during my project period. I will summarize the most important of them, organized by model family.

1.3.1 DeepVoice

Baidu published their first DeepVoice paper[3] in Feb-17, followed by DeepVoice2[15] in May-17. These early models stuck closely to a traditional TTS pipeline structure: DeepVoice1 is a single-speaker model that uses five separately-trained components for tasks such as modelling phoneme duration and predicting fundamental frequencies. However, by using a neural network for each component they removed the need for extensive manual feature engineering, and created a system that is simpler and more flexible than traditional TTS. DeepVoice2 extended the model to handle multiple speakers by introducing speaker embeddings.

DeepVoice3[41] (Oct-17) replaced this traditional architecture with an attention-based sequence-to-sequence model, which is a more general approach that encodes less domain knowledge and

can be applied to a range of input features (characters, phonemes) and output features (acoustic vocoder features, spectrograms.) They use a fully convolutional architecture rather than recurrent neural networks (RNN), which affords much faster training and parallel inference. In DeepVoiceCloning[2] (Feb-18), the DeepVoice3 system was extended to clone the voices of new speakers (not seen in the training set) given only small sets of their speech data.

1.3.2 Tacotron

Google's first Tacotron system[54] (Mar-17) is an end-to-end single-speaker TTS model, based on sequence-to-sequence networks with attention. It takes $< text, audio >$ inputs, and trains from a random initialization to predict spectograms, which can then be inverted directly into waveforms. Although the top-level architecture is straightforward and allows for end-to-end training, Tacotron uses complex components inside the encoder and decoder networks: in particular, 'CBHG' modules (composed of a 1-dimensional convolution bank, a highway network, and a bidirectional GRU) are used instead of vanilla RNNs in the encoder, and also on the output of the decoder.

Tacotron2[42] (Dec-17) uses simpler building blocks, replacing the CBHG modules and GRU layers with vanilla LSTM and convolutional layers. It also improves the quality of the spectrogram-to-waveform inversion by replacing the original Griffin-Lim algorithm with a WaveNet[50] vocoder. Tacotron2 set a new benchmark for speech quality (measure by Mean Opinion Score (MOS)), and for short samples is almost imperceptible from real human speech[5]. However, it is still a single-speaker model and other researchers were unable to recreate the same level of quality, suggesting that it is dependent on very high-quality training data and optimized hyperparameters.

In [22] (Jun-18), Tacotron2 was adapted into a multi-speaker model by adding an independently-trained speaker encoder network that outputs a fixed-dimensional embedding vector from a short speech sample. The speaker encoder was trained on a separate speaker verification task, using a separate, lower-quality dataset of speech from thousands of speakers. The model can synthesize natural speech in the voice of a target speaker, even for speakers not seen during training. It can also generate good-quality in the voices of 'fictitious' speech by randomly sampling from the speaker embedding space[6]. In this version of the model, the inputs are phonemes rather than raw text.

A number of papers have investigated learning **prosody** within the Tacotron architecture. Google's own work began with [55] (Nov-17), which aimed to learn independent prosodic styles in an unsupervised fashion (no prosodic annotations) by adding 'style tokens' to the original Tacotron. These tokens give some limited control over the prosody of the generated speech. This was extended in two papers from Mar-18: [57] introduced 'Global Style Tokens' (GST) for the Tacotron2 architecture, a similar but more general approach that was, for example, able to learn to factor out noise; [44] focused on explicit prosody transfer from reference to target utterances. Most recently, [46] (Jul-18) extended GSTs to 'Text-Predicted Global Style Tokens' (TP-GST), which learn to

[5]Samples are here (and they're awesome): `https://google.github.io/tacotron/publications/tacotron2/index.html`

[6]Samples are here: `https://google.github.io/tacotron/publications/speaker_adaptation/index.html`

predict prosodic (or other stylistic) variations in audio from text alone[7].

One of the early criticisms of the original Tacotron model was that it appeared to rely on large quantities of very high-quality training data. [11] (Aug-18) investigates a semi-supervised training approach based on large, public datasets of text and speech that are noisy and unpaired. The encoder is pre-trained using word embeddings from the text dataset. The decoder is pre-trained using the speech data. The model is fine-tuned using whatever paired $< text, audio >$ data is available. Tacotron's ability to learn from small datasets is much improved.

Other researchers have also published work using Tacotron-style systems[8]. [32] (Nov-17) adds emotion embeddings, and trains on a dataset of Korean speech with annotated emotions. [31] (Jun-18) investigated voice-cloning within a multi-speaker version of Tacotron2, with similarities to Google's own [22].

1.3.3 VoiceLoop

Facebook AI Research (FAIR) published their VoiceLoop paper[48] in Jul-17. VoiceLoop is a multi-speaker neural TTS system, in which each speaker is represented by a fixed-length vector in an embedding space. Rather than a complex recurrent architecture, it uses a much simpler shifting buffer working memory with Graves attention. It takes phonemes rather than raw text as input, and predicts WORLD acoustic vocoder features rather than spectograms. Most notably, it was the first paper to demonstrate the ability to add voices for new speakers to a previously-trained model, which it achieved by randomly initalizing an embedding for each new speaker and then training it on the new speaker's data with all other model parameters frozen.

I use VoiceLoop as the baseline implementation for my project, and so explain the architecture in much more detail in Chapter 4.

In [39] (Feb-18), the authors extend VoiceLoop by replacing the fixed speaker embeddings with utterance embeddings. This allows new speakers to be fitted given only a short sample (few seconds) of audio, and much more quickly than the technique used in the original paper[48]. The approach is similar to that used in [22], [2] and [31]. I follow a similar approach in chapter 6.

VAELoop[1] (Apr-18), a paper by a non-FAIR researcher, modifies VoiceLoop by replacing the speaker embedding lookup table with a variational autoencoder. The aim is to learn to generate more expressive speech, by capturing global characteristics of speech such as gender, emotion, and speaker identity. I follow a similar approach (although for different reasons) in chapter 8.

1.4 Multi-Speaker Models

One key respect in which neural TTS systems improve upon the traditional approaches is their ability to represent multiple speakers within a single model.

[7] Again, very impressive samples: `https://google.github.io/tacotron/publications/text_predicting_global_style_tokens/index.html`

[8] Google have not released a public implementation of Tacotron, although there are numerous open-source attempts on GitHub.

Concatenative synthesis systems are inherently single-speaker, as they can only draw upon the chopped-up speech samples of one speaker if the output is to sound natural. Statistical parametric synthesis systems have the potential to be multi-speaker, as they predict acoustic features from text and a single model could do this for multiple speakers with some shared model components[9]. Average-voice models with speaker-dependent fine-tuning using techniques such as MLLR have had some success[59]. However, in practice it is difficult to build a good multi-speaker SPSS system.

In contrast, in neural TTS, a single network can be trained on a multi-speaker dataset such that the bulk of the model's parameters are shared across all speakers, but each speaker's individual voice is represented by a low-dimensional vector in an embedding space. This allows the model to learn a decoder that is robust across a wide range of speakers, while still being able to capture the individuality of each speaker's voice. Excellent multi-speaker results have been reported across all the model families in section 1.3.

In addition, the flexibility of the model structure opens up exciting possibilities:

- **Fitting new (unseen) speakers** based on only a short sample of their speech, i.e. given a short audio sample of a new speaker, estimate that speaker's representation in the embedding space and then use it to generate speech in their voice[48][39][22][2]. The ability to recreate a voice would be invaluable to a person who has lost the ability to speak, and wants to communicate in a way that maintains their identity. There are, of course, also rather creepy applications such as generating speech in the voices of dead people, and downright malicious applications such as telephone impersonation[10] or fraud.
- **Manipulating the voices of known speakers** by transforming their representations in the embedding space, e.g. transforming the vocal characteristics of a known speaker, such as from male to female, or changing their accent from English to American.
- **Generating speech in 'brand new' voices**, by selecting a new point in the embedding space, either at random or by carefully choosing a point based on how the dimensions map to interpretable speaker attributes (gender, accent, age, speech rate etc.) This could be useful in applications where we want a unique voice for audio books, computer games or speech translation.

1.5 Summary

I began this chapter with some background on why speech synthesis is such a fascinating problem, and has been attracting engineers for centuries. I discussed the two main traditional TTS approaches, and reviewed the literature on exciting recent research into neural TTS architectures. One of the numerous advantages of Neural TTS systems is the ease with which they handle multiple speakers, and the possible applications this opens up to further research; hence my decision to research speaker representations in neural TTS for my project.

[9] In a sense, the neural TTS systems are statistical parameter systems, because they predict some kind of acoustic features (spectrograms or vocoder features) from text - they just do it in a very different, end-to-end manner.

[10] `https://www.bbc.co.uk/news/technology-39965545`

Chapter 2

Disentanglement

In recent years, there has been growing interest in methods for learning disentangled representations in the context of deep learning models. Disentanglement is not a new concept, however, and has always been desirable in representation learning[6].

In this chapter I discuss what disentanglement is and why it's desirable, and then review recent disentanglement research for generative models, covering VAE-based approaches in section 2.3 and GAN-based approaches in section 2.4.

2.1 What is Disentanglement?

Real-world data is typically complex. A key first stage in machine learning is to find a good way of *representing* the data, that can adequately express the variations in the data while providing enough clues to allow learning algorithms to operate. Traditionally, a feature representation is hand-crafted by humans, who often have significant domain knowledge that allows them to understand the structure of the data and the processes that generate it, and can design features that give the machine learning algorithms a big 'head start', or even make learning possible at all. Recent advances, particularly in deep learning, have in some cases reduced the requirement for complex, human-designed feature representations: the networks can often learn directly from (relatively) raw data. However, this is not to say that the architectures don't use feature representations; rather, they *learn* good internal feature representations as part of the training process. Representations are still an essential part of the process.

Disentanglement is a property of a data representation. It is an intuitive property and is not typically presented in formal terms. Different authors cast it in widely-varying language; however, all agree that disentanglement is important.

Consider a dataset $\mathcal{D}$ that has been generated by a complex real-world process, through the interaction of multiple factors. These factors are called variously 'generative attributes', 'generative factors', 'explanatory factors', 'attributes', 'semantic units' and so on. Whatever the nomenclature, the important things are that the factors are interpretable by humans; are associated with

significant, perceptable changes in the data examples; tend to cause *independent* changes in the outputs[1]; tend to change independently of each other; and tend to change one (or a few) at a time when considering sequences of events. Let there be a set $\mathcal{A}$[2] of generative attributes, with members represented by a^i.

Now consider a representation of this dataset, $\mathcal{R}$. This representation may be a hand-crafted set of features, the internal layers of a deep neural network trained on a supervised learning task, or the representation learned by an unsupervised learning task that simply aims to capture variations in the data. Let the set of features in the representation be $\mathcal{F}$, with its member features f^i. Again, nomenclature varies here: the features are often referred to as 'latent units', 'factors', 'dimensions' etc. depending on the context in which the research is being done. The representation will also include additional features that don't correspond to any attributes, that capture other random sources of variation in the data.

$\mathcal{R}$ is a *disentangled* representation of $\mathcal{D}$ if single features f^i are sensitive to changes in single generative attributes a^i, while being relatively invariant to changes in other generative attributes $a^j, j \neq i$[20][6].

The concept is best understood through examples. Consider the following datasets:

Images of Faces Generative attributes include: eye colour; hair colour; hairstyle; presence/absence of glasses, smile, hat, beard; age; gender; and of course the person's identity. These are human-interpretable. Changes in these attributes cause significant, perceptible changes in the image, while keeping all other attributes unchanged[3]. We can imagine changing the attributes individually, and can imagine sequence of images as attributes are varied in which the images seem progressively distant from the original image.

A disentangled representation for this dataset would learn separate features f^i for each attribute a^i, and would not learn features in which the attributes are mixed. So, there would be separate features for eye colour, hairstyle, gender and so on. In the ideal case there would be a one-to-one mapping from attributes to features (plus additional features for random variation), but given the complexity of real data generating processes and the vagueness in the specification of the attribute set $\mathcal{A}$, in practice far less than this would suffice.

MNIST Digits Generative attributes include: the identity of the digit; thickness of the pen stroke; rotation of the digit; size of the digit; curviness of the handwriting style (note how vague these are becoming, although they are still interpretable and potentially useful.) A disentangled represented would learn separate features for each of these, and would not mix any of the attributes into the remaining features.

Images of Chairs Attributes would include: type of chair; rotation; azimuth; size; leg style; colour.

[1] Note the lack of formality here. [20] says that "Some of which are conditionally independent." [6] says "assumed existence of multiple underlying factors of variation, whose variations are in some sense orthogonal to each other."

[2] It seems to me that there is no guarantee of a unique $\mathcal{A}$ for most data generation processes...

[3] although this is rather informal: if you add a beard to a female image, at what point is the gender no longer female?

Speech This is my problem in this project. Generative attributes could include: the content of what is being said (i.e. the words, or the thoughts behind them); the speaker identity, gender, age, accent; the speaker's prosodic variation (emotion, intonation); channel effects (recording equipment, sampling rate); background noise.

2.1.1 Why is Disentanglement Desirable?

A disentangled representation provides **interpretability**: we can understand examples by looking at their representations. It can provide **control**: in a generative model, we can modify the generated outputs in intentional ways by manipulating the representation (e.g. adding a hat to a portrait by altering the 'hat' feature.) A disentangled representation can make downstream machine learning easier and more robust: learning a good feature representation is (more than) half the battle in many machine learning tasks. For example, classification tasks would be trivial if the representation already includes features that represent the attributes to be classified.

Disentanglement is particularly desirable in unsupervised approaches that learn feature representations to be used in downstream tasks, which may not be known at the time. The best representation will depend on the downstream task. However, a disentangled representation is likely to be a useful starting point in many tasks of interests to humans. Real progress in artificial intelligence (AI) requires AI to understand the world around us. Identifying and disentangling the underlying generative attributes hidden behind messy real data, without explicit human guidance through feature engineering, is an important first step[6].

2.1.2 How is this Related to PCA?

An obvious question is: why not just use Principal Component Analysis (PCA) or Independent Component Analysis (ICA) to learn independent, orthogonal factors from the data?

The issue is that a representation constituted of independent factors is not necessarily disentangled. The independent factors are chosen to explain as much variance in the dataset as possible, but don't generally align with the generative attributes of the data. The independent factors may not be interpretable. In addition, it's unlikely that the relationship between the underlying generative attributes and the data distribution can be captured by a linear transformation.

2.1.3 Disentanglement Metrics

As stated, disentanglement is a rather vague, intuitive notion. There are currently no standard metrics for measuring how well a given representation disentangles a dataset. Metrics have been proposed, for example in [20] and [26], but at present most researchers develop a specific metric that is convenient for the task at hand. In this spirit, in Part III I will also use ad hoc metrics depending on the task.

2.2 Approaches to Learning Disentangled Representations

Approaches can be grouped by:

- **Supervision**: Whether they rely on labelled data (supervised, semi-supervised, unsupervised)
- **Underlying Generative Model**: VAE, GAN, or flow-based generative model

I will structure my review by the type of generative model, dealing with VAE-based approaches and then GAN-based approaches.

I will not discuss the flow-based generative models such as Glow[27][12], although these have reported some very exciting results in recent months. If I were starting my project now, I would certainly want to look at these techniques.

2.3 VAE-Based Approaches

2.3.1 The Variational Autoencoder

Kingma & Welling introduced the Variational Autoencoder (VAE) in 2014[28], and the technique has become a tremendously important approach to unsupervised learning and generative modelling. VAEs can be approached from a number of different directions, with varying degrees of formality.

They can be viewed as a general, practical tool to perform efficient variational inference in directed probabilistic models with intractable posterior distributions and complex data (i.e. a fallback for graphical models when neither expectation maximisation, mean-field Variational Bayes, nor Monte Carlo EM algorithms are tractable[28]).

They can also be viewed as a modification of a standard deep auto-encoder in which the encoder network is switched from being deterministic to stochastic, such that the encoder maps an example x to a distribution of z over the latent space, rather than mapping deterministically to a point z.

In this report I will give only a very brief outline of how the VAE works, helped considerably by[13], and coming from the direction of the VAE being a modification of the standard autoencoder that yields a generative model.

Let $P_{data}(X)$ represent the true distribution of datapoints X in some high-dimensional space $\mathcal{X}$. For example, each datapoint could represent a possible image in a large pixel-space. We are interested in modelling $P_{data}(X)$ with an approximate distribution $P(X)$, which is as similar as possible to $P_{data}(X)$ and from which we can generate sample images. But learning a distribution $P(x) \quad \forall(X)$ is hard.

Instead, we introduce a latent variable z in an high-dimensional latent space $\mathcal{Z}$ that we can sample from easily using a probability density function prior $P(z)$ on $\mathcal{Z}$. We typically assume a centered,

isotropic multivariate unit Gaussian, i.e. $P(z) = \mathcal{N}(z; 0, I)$. We then assume the existence of a function from this sampled z to datapoint x using a conditional distribution $P_{\theta^*}(x|z)$. θ^* are the unknown true parameters; we assume that P_{θ^*} comes from a parametric family of distributions $P_\theta(x|z)$, typically multivariate Gaussian. We call $p(x|z)$ the 'decoder'.

We now consider the probabilistic 'encoder' model $q_\phi(z|x)$, which is an approximation to the intractable true posterior distribution $P(z|x)$. For any datapoint x, this gives a distribution of latent variables over the latent space from which x might have been generated.

The decoder $p(x|z)$ and encoder $q(z|x)$ are both implemented using deep neural networks. To maximise the likelihood, we train by optimizing the evidence lower bound (ELBO), averaged over the empirical distribution (i.e. our dataset):

$$\mathcal{L}^{avg}_{ELBO} = \frac{1}{N}\sum_{i=1}^{N}\Big(\mathbb{E}_q[\log p(x^{(i)}|z)] - D_{KL}\big(q(z|x^{(i)})||p(z)\big)\Big) \tag{2.1}$$

If the distributions are specified to be multivariate Gaussians, the parameter gradients for mini-batch training can be estimated using the reparamaterization trick[28] and the KL-divergence between the posterior and prior can be calculated analytically.

The first term on the right-hand side of equation 2.1 can be seen as a reconstruction loss, measuring the likelihood under the decoder of recovering the original datapoint x from its code z. The second term is a regularization penalty according to how far the posterior distribution deviates from the prior. When maximizing the ELBO, there is a trade-off between obtaining good reconstructions and not allowing the posterior to stray too far from the prior. The KL-divergence penalty therefore results in a worse reconstruction loss than would otherwise be the case: the images generated by VAEs can be somewhat 'blurry'.

2.3.2 Disentanglement in VAEs

[28] show that in the right circumstances, a VAE can achieve some degree of disentanglement. For example, dimensions representing emotion and pose can be disentangled for a dataset of simple facial images. However, this disentangling performance does not scale to more difficult problems[20].

β-VAE [20] enhances the ability of the VAE to learn disentangled representations by adding a hyperparameter β to control the relative importance of the reconstruction and regularization components of the training objective:

$$\mathcal{L}_\beta = \frac{1}{N}\sum_{i=1}^{N}\Big(\mathbb{E}_q[\log p(x^{(i)}|z)] - \beta D_{KL}\big(q(z|x^{(i)})||p(z)\big)\Big) \tag{2.2}$$

When $\beta = 1$, the objective equation 2.2 reverts to the standard ELBO of equation 2.1. However, for $\beta > 1$ and if the prior $p(z)$ is a factorized distribution (here, the unit isotropic Gaussian), then the dimensions of the latent representation are more strongly encouraged to align to the

independent factors. $\beta > 1$ forces a stronger constraint on the latent bottleneck, and limiting the capacity of z. As the KL penalty increases, the model must learn a more efficient latent representation of the data: if the true data generation process has some independent generative attributes, then these are pushed into independent latent variables.

By optimizing the β hyperparameter, a trade-off can be made been reconstruction quality and a disentangled representation. [20] reports good disentanglement results on image datasets such as CelebA.

In [8], the authors give more details on the reasons why β-VAE achieves disentangling.

The β Total Correlation VAE (β-**TCVAE**) approach[9] extends β-VAE by decomposing the variational lower bound, discovering a term that measures the total correlation between latent variables. This gives insight into how BetaVAE works and also a means to improve performance further by assigning different penalty weights to the individual terms in the decomposition of the lower bound.

FactorVAE[26] also extends the β-VAE approach to give a better trade-off between disentanglement and reconstruction loss.

2.3.3 Disentangled Representations for Sequences

Numerous papers in the past year have addressed the problem of learning disentangled representations for sequences of data, such as videos (i.e. sequence of images), or audio data such as speech, as opposed to 'static' data such as images. This type of data can be naturally structured or multi-scaled, e.g. latent factors can include static components (speaker identity) and dynamic components (the words they're saying.)

The Factorized Hierarchical Variational Autoencoder (**FHVAE**)[21] learns disentangled and interpretable representations for sequence- and segment-level attributes without any supervision. It uses a factorized hierarchical graphical model that assigns sequence-dependent priors to one set of latent variables, and sequence-independent priors to another set. When trained on speech datasets, it can manipulate the appropriate latent variables to transform speaker identity (i.e. sequence-independent attribute), or linguistic content (i.e. sequence-dependent.)

Disentangled-VAE[33] is a "minimalist" VAE architecture for high-dimensional time-series that learns a latent representation disentangled into two parts: static (global/context) and dynamic (time-varying). This gives control over generation and reconstruction, including style transfer. Although mostly focussed on video, it also demonstrates some potential on Male-to-Female conversion using TIMIT speech data.

2.4 GAN-Based Approaches

Whereas VAE-based approaches are based on solid fundamental principles of maximum likelihood and variational inference, the GAN-based approaches to generative modelling are more practical,

'engineering' solutions.

2.4.1 The GAN

The Generative Adversarial Network (GAN) approach was introduced by Goodfellow et al.[16] in 2014, and has quickly become a hugely popular approach for designing generative models. Rather than aiming to maximise likelihood of the posterior distribution when reconstructing real examples, GANs are explicitly optimized for realistic output generation (and in fact don't reconstruct training examples at all.)

Assume we have a distribution over 'real' data x given by $P_{data}(x)$. For example, the domain of x could be fixed-size images of individual human faces. The goal of the GAN is to learn a *generator distribution* $P_G(x)$ that matches $P_{data}(x)$, such that sampling from $P_G x$ always generates realistic outputs in the same proportion as the real distribution.

However, rather than explicitly learning the distribution $P_G(x) \quad \forall(x)$, the GAN trains a *generator network* G to generate samples from $P_G(x)$. To generate a sample from $P_G x$, we sample a noise variable $z \sim P_{noise}(z)$ (e.g. a diagonal multivariate Gaussian), pass it as input to the generator network G, and receive an output $G(z)$. The trick is then to pass $G(z)$ through an adversarial discriminator network D that is trained to distinguish between 'real' samples from the true distribution P_{data} and 'fake' samples from our generator distribution P_G. If the generator is good then the discriminator should struggle to distinguish between 'real' and 'fake'. The overall network is trained as an adversarial game in which, at each step, the generator G is trained to produce 'fake' outputs that the discriminator D cannot distinguish from 'real' ones, while the discriminator D is trained to get better at distinguishing between 'real' and 'fake' data. As training proceeds, G produces better outputs, D is then trained on higher-quality 'fakes' and can learn to be a more powerful discriminator, which forces G to produce even better outputs, and so on.

This training procedure can be expressed formally as a two-player mini-max objective with value function $\mathcal{V}(D, G)$:

$$\min_G \max_D \mathcal{V}(D, G) \tag{2.3}$$

where:

$$\mathcal{V}(D, G) = \mathbb{E}_{x \sim P_{data}}\big[\log D(x)\big] + \mathbb{E}_{z \sim P_{noise}(z)}\big[\log(1 - D(G(z)))\big] \tag{2.4}$$

Although the original GAN approach is attractive and capable of producing good results, the early versions were subject to a number of problems: GANs are hard to train; they're fragile with respect to changes in network architecture; and as they have no encoder, we can't map a particular example x onto its latent representation z and therefore can't reproduce or modify known examples.

Since the original paper, there has been much work on improving GANs, both theoretically and practically, and many applications to different domains.

As an example of a theoretical improvement, the Wasserstein GAN (WGAN)[4], improved in [18], is an alternative training approach that makes the networks easier and more stable to train by

replacing the KL divergence metric for the distance between distributions with the Earth-Mover metric, which is a more interpretable loss metric that correlates with the quality of generated outputs, gives smoother gradients, and is more forgiving of the balance between the complexities of the generator and discriminator.

2.4.2 Disentanglement using GANs

The GAN approach does not enforce any restrictions on how the generator G can map the input noise z to an output $G(z)$. In general, we would expect the generator to use the dimensions of z in any way it can to 'beat' the discriminator: a simple entropy argument[4] supports the idea that the generator is far more likely to learn an entangled representation, with the separate dimensions of z not encoding the generative attributes of the underlying data. Whether or not the person is wearing a hat, for example, is more likely to be expressed across many dimensions of z rather than just one (and with that dimension not having an impact on any other generative attributes.)

Conditional GANs (**cGAN**)[36] are a straightforward extension to the GAN framework that uses a supervised approach to encourage a disentangled representation. Both the discriminator and the generator are conditioned on class or attribute labels y, and the mini-max game becomes:

$$\min_G \max_D \mathcal{V}(D,G) = \mathbb{E}_{x\sim P_{data}}\big[\log D(x|y)\big] + \mathbb{E}_{z\sim P_{noise}(z)}\big[\log(1-D(G(z|y)))\big] \tag{2.5}$$

By training a cGAN on MNIST data images conditioned on their class labels, the original cGAN paper was able to learn a network in which the digit class identity is disentangled from other factors of variation (digit size, width, rotation etc.)

The Invertible Conditional GAN (**IcGAN**)[40] approach combines the cGAN[36] with an encoder, to address the limitation that GANs lack an inference mechanism and hence can't map a real example x to its latent representation z. After training the cGAN, two sub-encoders are trained: E_y maps an input example x to its labelled attributes y, while E_z maps to the latent representation z. The idea is that even though the cGAN was trained without any notion of reconstructing training examples, we can nonetheless learn a way to map a training example to its representation in the latent space learned by the cGAN. The two sub-encoders are trained in slightly different ways:

- **Latent encoder** $E_z(x)$: The cGAN generator is used to create a synthetic dataset (z, y', x') of sampled noise vectors z and labels y', together with the corresponding generated outputs $x' = G(z, y')$. We can now use E_z to encode these outputs x' and see how well they approximate the true z values that generated the x', using the squared reconstruction loss:

$$\mathcal{L}_z = \mathbb{E}_{z\sim p_z, y'\sim p_y}\|z - E_z(G(z,y'))\|^2$$

- **Attribute Encoder** $E_y(x)$: This is trained directly on the real training dataset, again using

[4] i.e. there are only a few ways to learn a disentangled representation, many ways to learn an entangled one, and no preference in the system to prefer the former

a squared reconstruction loss:

$$\mathcal{L}_y = \mathbb{E}_{x,y\sim p_{data}} \|y - E_y(x)\|^2$$

The advantage of the IcGAN approach is that we can now take an input example x, encode it to $z = E_z(x)$ and $y = E_y(x)$, modify the attributes y to y', and then pass (z, y') to the cGAN generator to generate a realistic output of x with modified attributes. For example, take an image x of a woman with blonde hair and encode it into an attribute y that represents the blonde hair, and a latent code z that represents all other information required to reconstruct the original image. Then modify the attribute y to the value y' that represents black hair. Now pass (z, y') to the generator to generate a plausible image of the original woman with black rather than blonde hair.

The IcGAN paper reports good results in controlling the attributes of generated images on MNIST and CelebA. On MNIST, IcGAN can for example take an image of a thickly-drawn digit and, by varying the digit identity attribute, generate realistic images of other digits in a similar thick style. On CelebA, IcGAN can generate images of an example person with different colour hair, with or without glasses, with changed gender and so on.

However, IcGAN is a supervised approach that requires extensively labelled data to train the cGAN and the encoders.

In contrast, the 'Information Maximizing GAN' (**InfoGAN**) approach[10] introduces an unsupervised way to learn interpretable and disentangled representations. InfoGAN decomposes the unstructured GAN noise vector z into two components:

(i) c: the latent code, reserved for learning the generative attributes of the data

(ii) z: residual, incompressible noise

It then adds an information-theoretic regularization to force the network to learn the generative attributes as components of c. InfoGAN aims for high mutual information between the latent codes c and generator distribution $G(z, c)$, i.e. high $I(c; G(z, c)) := H(c) - H(c|G(z, c))$, where H stands for entropy. Intuitively, $I(X; Y)$ measures the reduction in uncertainty in X given Y. Therefore, high $I(c; G(z, c))$ means that an output generated using latent code c gives us a lot of information about what c was. So, for example, if c were the latent code for 'wearing a hat', then it should be pretty obvious from the generated image $G(z, c)$ that the subject is wearing a hat.

The InfoGAN paper[10] successfully disentangled the attributes of digit identity[5], style, rotation and width on MNIST data. The approach also found success in disentangling attributes such as rotation, width, azimuth, elevation, hairstyle and wearing glasses on other image datasets (faces, chairs, CelebA.)

However, although InfoGAN training is unsupervised, the model specification does require some prior knowledge of the number and types of attributes that are present in the data distribution.

[5] the identity is a categorical code, i.e. 1, 2, 3...; the other codes are continuous

For the MNIST case, for example, we must specify one categorical variable with ten classes, and two continuous codes distributed uniformly on $[-1, 1]$.

The IcGAN has the huge practical advantage that it can take a known input (e.g. face) and generate outputs with realistic changes to interpretable attributes such as hair colour and gender: IcGAN gives *control*. However, it is a supervised approach that requires labelled data, which may not be available or reliable in all contexts.

In contrast, InfoGAN is an unsupervised approach that can learn a disentangled latent representation without being supplied with labels (although some prior knowledge of the number and type of generate attributes is required). However, it lacks an inference network: we can't map a known input (e.g. face) to its latent representation, so we have limited control over the images we generate. In addition, the quality of generated images is lower than for IcGAN.

2.4.3 Fader Networks: Adversarial Training on the Latent Space

An interesting practical variation on the adversarial approach is proposed in the Fader Networks paper[30]. This is a supervised approach applied to the image domain that has similarities to IcGAN: it allows us to map a known example to its latent representation, and by manipulating independent dimensions of the representation gives control over the perceived attributes of the generated images. However, whereas IcGAN uses a cGAN in *pixel space* (i.e. uses a GAN discriminator to judge whether images are real or fake), Fader Networks use a deterministic encoder-decoder architecture and then apply adversarial training directly on the *latent space* to force the network to learn representations in which labelled attributes are disentangled.

The Fader Networks approach is able to generate very high quality images while achieving disentanglement and control.

In chapter 7 I will apply the Fader Networks approach to the problem of disentangling speaker representations in VoiceLoop: I will therefore defer a more detailed explanation of the approach to section 7.1.

2.5 Summary

In this chapter, I explained the concept of disentanglement and why it's desirable. I then briefly reviewed recent literature, focusing on VAE and GAN-based approaches, including a mix of supervised and unsupervised techniques.

Chapter 3

My Project

In chapter 1 we learned that single neural TTS systems can learn the voices of multiple speakers.

In chapter 2, we saw that there has been much work on learning disentangled representations, particularly in the image domain.

In my project, I aim to bring these two strands together by investigating whether applying recent disentanglement techniques to a multi-speaker neural TTS system could allow us to disentangle the speaker representations, affording both interpretability and control over the generated speech.

3.1 Why this Project is Exciting

In section 2.1.1 I discussed why disentanglement is desirable: amongst other things, it provides interpretability and control. In section 1.4 I discussed the exciting possibilities of multi-speaker TTS systems. Clearly, a *disentangled* speaker representation would be beneficial, giving the ability to interpret and control speaker characteristics. It could be possible to make a speaker's gender more or less pronounced by varying the value of a single factor in the speaker embedding, exaggerate or soften an accent using another factor, and so on.

In addition to points discussed elsewhere, interesting applications of a high-quality system include:

- **Gender-Neutral Voices**: By controlling the value of a gender factor, the system could produce 'gender-neutral' voices. This could be useful in avoiding the reinforcement of existing gender stereotypes: it has been reported that female voices are often used for personal assistants such as Alexa because they are perceived as servile; in contrast, a male voice is used for Watson because it is perceived as authoritative[1].

[1] Interesting article

- **Voice Privacy**: An area of the embedding space around each real speaker could be 'blocked off' to prevent the generation of speech that is too similar to any known human. Individuals would know that the system is not allowed to produce speech that sounds too much like them.
- **Speech Translation**: When translating a person's speech into a different language, we could develop a system for mapping their speaker embedding across languages (or build a common multi-speaker, multi-language system that uses a single embedding per speaker). Then, the translated speech would sound *like the original speaker*. Communicating in a voice that sounds like the original speaker is much more pleasing to the ear, and facilitates closer social interaction.

3.2 Architectures Used in my Project

I use four variants of the VoiceLoop architecture in my project: the first is the public implementation, and the remaining three are my own work. To make it clear what I worked on in my project, I list the techniques below.

3.2.1 Baseline Model: VoiceLoop

Architecture 1) I use FAIR's **VoiceLoop**[48] neural TTS system as the baseline for my project. It uses a relatively straightforward architecture, is well-documented, and has an official public implementation.

Architecture 2) I also implemented a modified VoiceLoop architecture based on **utterance embeddings** rather than speaker embeddings: this is similar to what FAIR did in[39].

3.2.2 Disentanglement Techniques

I selected two techniques from those discussed in chapter 2 to give an interesting mix of supervised, unsupervised, GAN-like and VAE-like approaches:

- **Architecture 3) Fader Networks**: Supervised approach using an adversarial technique (GAN-like)
- **Architecture 4) β-VAE**: Unsupervised approach using a VAE-based approach

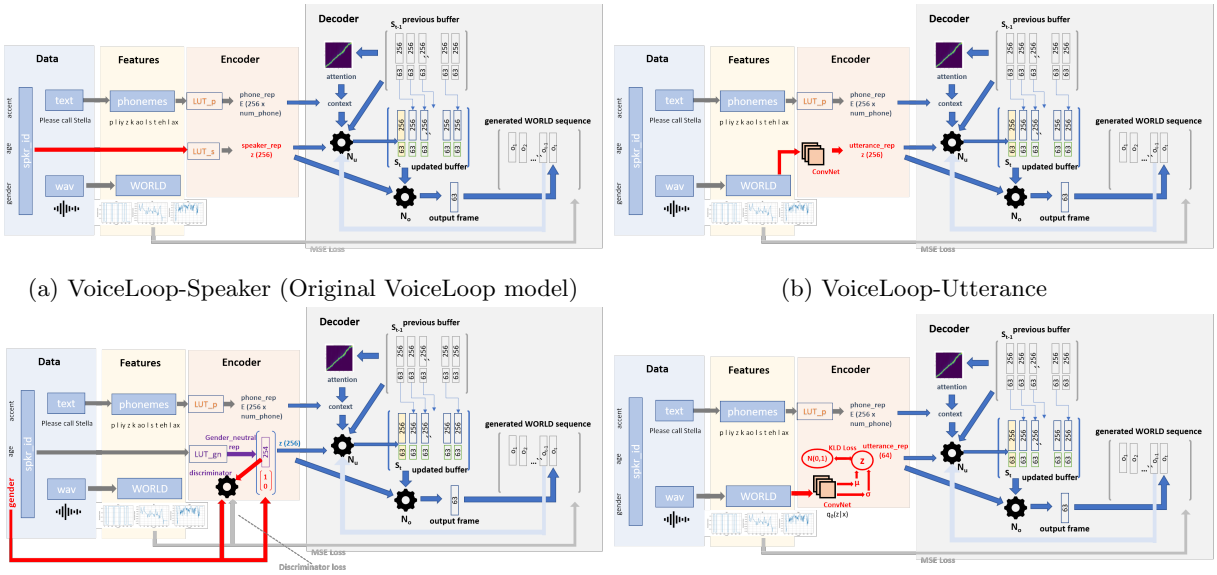
3.2.3 Comparison of Architectures

Figure 3.1 compares the four architectures I use. These will make more sense when they are introduced at the relevant points in the report, but it's useful to collect them here for reference.

3.2.4 Project Website

At various points in my report I refer to audio samples generated from my systems. These are all available at the project website: `https://richardsterry.github.io/msc-project/`

My code is also available on GitHub: `https://github.com/RichardSterry/msc-project`



(a) VoiceLoop-Speaker (Original VoiceLoop model)

(b) VoiceLoop-Utterance

(c) VoiceLoop-Fader

(d) VoiceLoop-BetaVAE

Figure 3.1: The four architectures used in my project.

3.3 Structure of this Report

The remainder of my report is organised into three parts.

Part II Speaker Representations in VoiceLoop:

- Chapter 4 describes the VoiceLoop architecture in detail.
- Chapter 5 illustrates how VoiceLoop represents speakers using a fixed-length vector in an embedding space, and examines how these vectors can be interpreted and used to control the generated speech.
- Chapter 6 discusses my extension to use utterance embeddings rather than speaker embeddings.

Part III Learning Disentangled Speaker Representations:

- Chapter 7 explains how I modified VoiceLoop to use a supervised Fader Network-style approach to encourage a disentangled speaker representation, and reports good results on disentangling the gender attribute.

- Chapter 8 discusses how I added an unsupervised β-VAE module to VoiceLoop to encourage a disentangled speaker representation, and reports encouraging results on disentangling gender.

In **Part IV Discussion & Conclusions**, I summarise my results, reflect on what went well and poorly, and make suggestions for future research.

Part II

Speaker Representations in VoiceLoop

In this part of my report I introduce the baseline VoiceLoop neural TTS model, and investigate how it represents speakers. This provides a solid base to build on in Part III.

Chapter 4 provides a detailed description of the VoiceLoop architecture, the dataset I use in my project and the two baseline models I trained that will be referred to throughout my report.

Chapter 5 establishes subjectively and objectively that VoiceLoop is a good multi-speaker TTS system that can recreate the voices of known speakers. It illustrates the way that VoiceLoop represents speakers in the embedding space, demonstrating that the embeddings are interpretable and can be manipulated to generate interpretable changes in the output speech.

Chapter 6 explains the motivation for modifying the VoiceLoop architecture to use *utterance* embeddings rather than speaker embeddings. It presents the modified VoiceLoop-Utterance architecture, and illustrates the interpretability of its embedding space.

Chapter 4

The VoiceLoop Model

I use the VoiceLoop model proposed by Facebook AI Research (FAIR), introduced in Taigman et al. [48] and extended in [39], as the benchmark neural TTS system for my project. FAIR has released the source code from [48] on GitHub: I use this as the starting point for my research.

VoiceLoop is a sequence-to-sequence model with attention that predicts sequences of vocoder features given a free text string and a one-hot speaker ID for the desired voice. Figure 4.1 shows the full pipeline, which is made up of four blocks: **data**, **features**, **encoder**, and **decoder**. In this chapter I will go through the architecture block by block.

Section 4.1 introduces the VCTK **data** corpus.

Section 4.2 explains how this raw data is pre-processed into linguistic and acoustic **features**.

Section 4.3 discusses the two datasets built from subsets of processed VCTK on which I trained my baseline VoiceLoop models: VCTK-US-21 and VCTK-All-107.

Section 4.4 lays out VoiceLoop's architecture in detail, explaining the **encoder** and **decoder** networks, the training process and evaluation of synthesized speech.

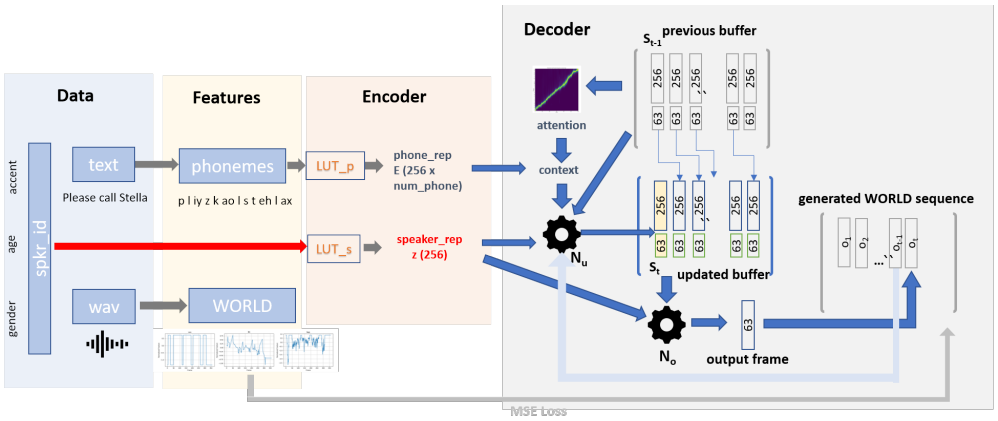


Figure 4.1: Architecture of the VoiceLoop model

4.1 Data

The VoiceLoop approach is not tied to any particular dataset. The authors have applied it to single-speaker datasets (LJ, Blizzard Challenge 2011 & 2013), multi-speaker datasets (VCTK, LibriSpeech, VoxCeleb), and even a set of speeches downloaded from YouTube that contain significant noise in both the audio and the auto-generated transcripts.

In my project I use the **VCTK** dataset for all experiments.

VCTK (Voice Cloning ToolKit)[1] is a multi-speaker, English language dataset consisting of pairs of short utterances (typically a few seconds long) in waveform format together with their text transcripts. The data was collected by The Centre for Speech Technology Research (CSTR) at the University of Edinburgh between around 2012-7, mainly for the purpose of building HMM-based TTS systems. VCTK was designed to help build a voice synthesis system that would allow people suffering from motor neurone disease to communicate using a synthetic voice that sounds like them[2]. However, they have kindly made a subset available to the community under an open source licence (ODC-By), and VCTK has become a standard resource for speech research.

Number of Speakers: 108

	id	age	gender	accents	region
236	236	23	F	English	Manchester
237	237	22	M	Scottish	Fife
238	238	22	F	NorthernIrish	Belfast
239	239	22	F	English	SW England
240	240	21	F	English	Southern England
241	241	21	M	Scottish	Perth
243	243	22	M	English	London
244	244	22	F	English	Manchester
245	245	25	M	Irish	Dublin
246	246	22	M	Scottish	Selkirk
247	247	22	M	Scottish	Argyll
248	248	23	F	Indian	
249	249	22	F	Scottish	Aberdeen
250	250	22	F	English	SE England
251	251	26	M	Indian	

Figure 4.2: Samples of VCTK speaker metadata

The full VCTK dataset includes around 2000 speakers, of which 109 are included in the public dataset. Two of these speakers are dropped during my initial pre-processing stage: one is missing the text transcripts, while the other is missing from the metadata file. We therefore have 107 speakers available for training and validation of the main model. The dataset provides metadata for each speaker covering age, gender, accent and the region they come from, as illustrated in figure 4.2. The speakers were recruited from across the student population at the University of Edinburgh, and therefore the age profile is young and rather uniform: as shown in figure 4.3, all speakers are at least 18, most are in their early 20s, and there is nobody older than 38. Speakers from England account for around a third of the set, followed by Americans making up a fifth. As you'd expect, there are plenty of Scots. Almost all appear to be native English speakers, and

[1] `https://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html`
[2] `https://www.youtube.com/watch?v=xzL-pxcpo-E`

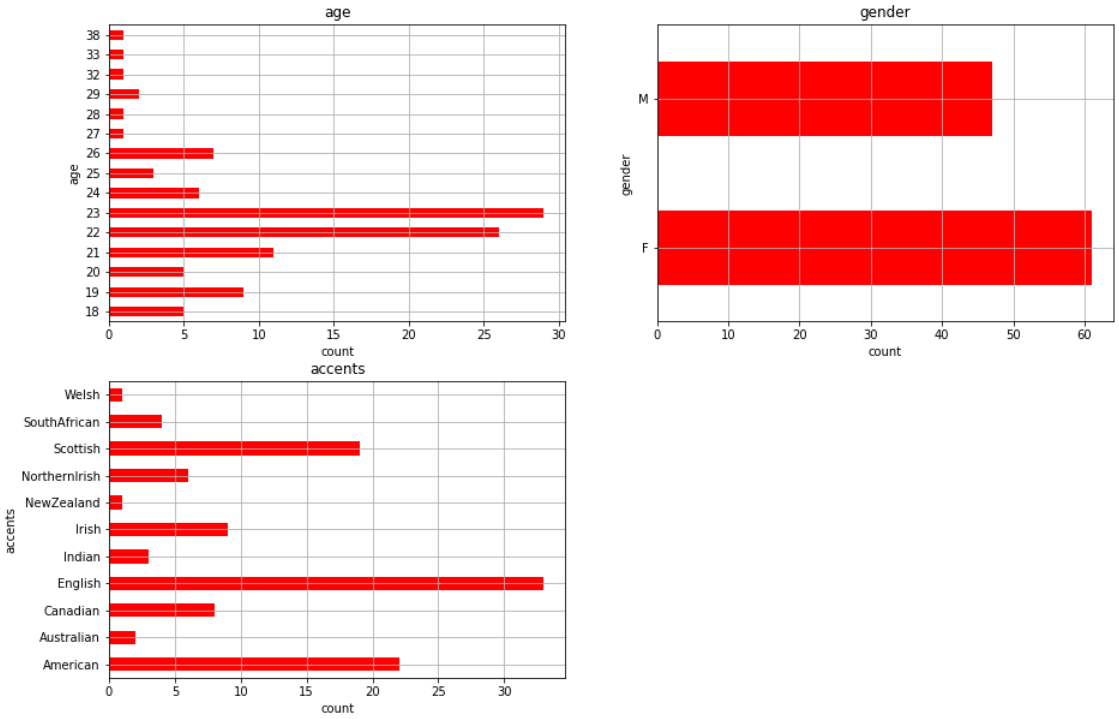


Figure 4.3: VCTK dataset descriptive statistics

as they're part of an international student body based in one location their accents are relatively mild[3]. There are slightly more females than males.

This dataset is not a representative cross-section of all English speakers, and isn't large enough to come close to covering the full range of voices. However, there is a reasonable degree of diversity and it's more than sufficient for my research project.

Each speaker was given a set of around 400 sentences to read out loud. Some of these sentences were chosen for their ability to discriminate accent and to cover a range of phones, diphones etc.[4], and were therefore given to all or most speakers. Other sentences were given to only one or a few speakers. So, the sentence lists have some overlap across speakers, but are mostly individual to the speaker. Most sentences were taken from a newspaper: The Herald (Glasgow). As the speakers are amateurs (as opposed to professional voice artists), were reading single sentences from newspapers, and were told to speak naturally, there is very little prosodic variation in the dataset and it is rather 'flat'. In contrast, a dataset taken from an audiobook read by a professional voice actor would contain a lot of prosody, i.e. dynamic, dramatic speech.

All speakers were recorded using the same, high-quality equipment in a single recording studio, at

[3]Compared to, say, the accent you might expect from older Scottish speakers based in the Highlands
[4]This is a common requirement in traditional TTS systems

96kHz sampling frequency and 24 bits. These were converted and downsampled to 48 kHz, 16 bits. I will often refer to a single audio example of a speaker reading a sentence as an **utterance**.

The project website `https://richardsterry.github.io/msc-project/vctk.html` includes audio samples, transcripts and a speaker metadata table: it is worth listening to a few sample utterances to gain some familiarity with the dataset.

4.2 Features

Each VCTK example utterance is of the form $\{\text{text}, \text{waveform}, \text{speaker\_id}\}$. However, VoiceLoop does not operate directly on these raw representations: the components are first transformed into more convenient, domain-relevant features:

- text $\rightarrow$ phonemes
- waveform $\rightarrow$ WORLD vocoder features

Each of these transformations encodes a significant amount of domain expertise developed over decades of traditional speech research. In sections 4.2.1 and 4.2.2 I will describe each of these pre-processing steps in more detail.

Note that these are not the only possible choices. Some neural TTS approaches use different feature representations, or work directly on the lower-level data and effectively require the model to learn how to handle these transformations themselves. For example:

- char2wav[45] and original Tacotron[54] operate directly on the **text transcripts** rather than on the phoneme sequences. Although it is an interesting research question to see whether the models can learn all the way back to characters, in practice the phoneme transformation is very powerful and robust, and makes it possible for the model to pronounce unseen or rare words such as proper names. The more recent Tacotron work on prosody[55][44][46] and multi-speaker representation[22] has shifted to using phonemes, which highlights the helpfulness of this transformation.
- Tacotron and DeepVoice[41] operate on mel-frequency **spectrogram** representations of the waveforms rather than the WORLD features.

4.2.1 Text $\rightarrow$ Phones Using a Pronouncing Dictionary

VCTK provides us with a text transcript for each example, i.e. a sequence $\mathbf{c}$ of characters, for example:

Please call Stella

In many languages, the mapping from text to pronunciation is non-trivial: the sounds of individual characters can vary significantly depending on the context provided by neighbouring characters. As a simple example, in most English words the character 'k' maps to a clear plosive sound, but in certain specific cases such as in the word 'knight' or 'knife' it is silent.

Linguists have studied the rules of pronunciation (and their exceptions) for decades, and have developed numerous systems of **phonemes** to describe the basic units of sound expressible in a given language. English is typically analysed as having around 40 phonemes. Some examples of phonetic transcriptions:

Text	Phone Sequence
TRANSLATION	t r ae n z l ey sh ax n
NEIGHBOUR	n ey b r
TUESDAY	t uw z d iy

The task of mapping words into phonemes is accomplished using a pronouncing dictionary. VoiceLoop uses the open-source CMU pronouncing dictionary `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`, which provides phonetic transcriptions for over 134,000 words of North American English based on a set of 39 phonemes derived from the ARPAbet[5] symbology. There are additional levels of differentiation within each phoneme depending on context and stress, but this course categorization is a good start.

In the pre-processing pipeline, text transcripts are converted to phoneme sequences using the CMU dictionary via Festival[6], based on the Phonemizer[7] toolkit.

4.2.2 waveform → WORLD Vocoder Features

A **waveform** simply represents the amplitude of a longitudinal pressure wave over time. This is the low-level sense experience our ears are presented with and is in some sense the lowest-level representation of the audio data.

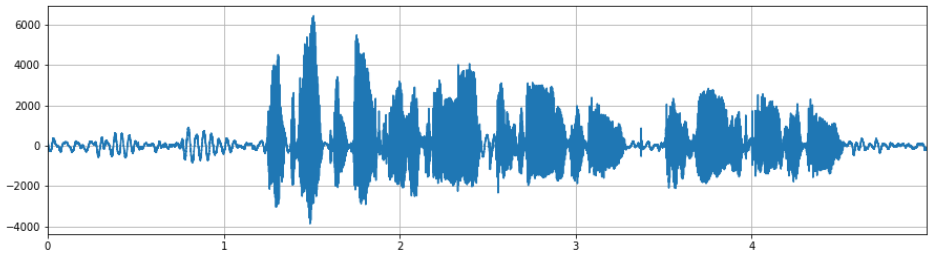
But sound is best understood as superpositions of multiple frequencies. Part of the genius of the interaction between the human ear and brain is its ability to analyse a pressure sound wave into frequencies and segment it into sources coming from many directions[8].

The standard approach to moving from the amplitude to the frequency domain is to apply a Fourier transform to convert the waveform into a **spectrogram**, a representation of the energy of the signal at different frequencies over time. Figure 4.4a shows the wavefore of an example utterance from VCTK; figure 4.4b shows its representation as a spectrogram.

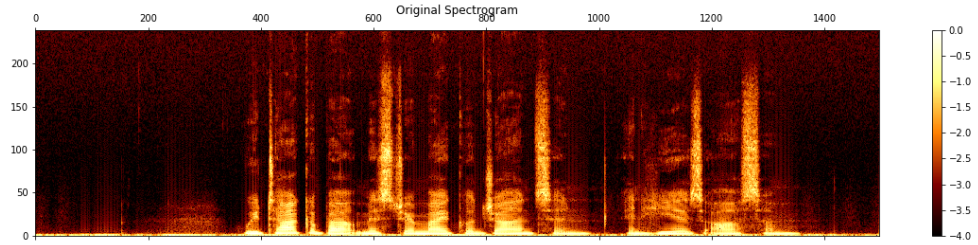
Tacotron and others use a spectrogram representation of the speech audio, or the closely related **mel spectrogram**, which applies a filter bank designed to reflect the fact that human hearing is not equally sensitive to all frequency bands. The mel filter bank is spaced uniformly below 1 kHz, and logarithmically after 1 kHz.

Another option is to convert the waveform into a set of **acoustic features** that has been carefully designed to be relevant for representing human speech. This has the advantages of:

[5] `https://en.wikipedia.org/wiki/ARPABET`
[6] `http://www.cstr.ed.ac.uk/projects/festival/`
[7] `https://github.com/bootphon/phonemizer`
[8] cocktail party effect



(a) Waveform



(b) Spectrogram

Figure 4.4: Analysing the audio from a VCTK sample (speaker 330, sentence 230)

- **Interpretability**: We can understand features through their relevance to human speech production, e.g. pitch, voiced/unvoiced.
- **Robustness**: We would expect that small perturbations in features from correct values would still produce approximate speech, whereas in another representation could be very fragile.

As discussed back in section 1.2, these acoustic features are part of a vocoder, which can analyse speech into features and then synthesise the features back into speech. VoiceLoop uses features from the **WORLD vocoder**[38][9], via the Merlin[10] interface.

The analysis part of the WORLD vocoder takes in a waveform and converts it to sequences of 63[11] WORLD features per time step. Table 4.1 shows the features. Note that VoiceLoop uses **normalised** versions of these features, using a simple z transformation based on the mean and standard deviation of all samples in the training set. Also, a pre-processing step is required to clip out silences at the start and end of the raw VCTK utterance samples.

In figure 4.5 I show three of the WORLD feature sequences taken from the analysis of a VCTK example. I will typically consider these three features (vuv, lf0 and bap) in my report because they are the easiest to interpret. vuv is a binary feature, and is 'on' when the vocal chords are vibrating. lf0 is a measure of pitch: when it's high (and this data has been normalised, so zero is 'average' pitch), the corresponding sound would have a higher pitch. The example in the figure shows the pitch falling over the utterance.

[9] `http://www.kki.yamanashi.ac.jp/~mmorise/world/english/`
[10] `https://cstr-edinburgh.github.io/merlin/`
[11] This depends on the sampling frequency of the raw audio, but for 16kHz audio it outputs 63 features.

Feature	Index	Name	Description
mgc	0-59	Mel-Generated Cepstral Coefficients	Frequency-based metrics from spectral envelope
vuv	60	Voiced/unvoiced	Related to a binary feature of vocal production (whether the vocal chords vibrate when making a sound. Compare 'p' and 'b')
lf0	61	Log F0	F0 (f-zero) is the fundamental frequency, informally referred to as 'pitch'. Here we have the log of f0.
bap	62	Band Aperiodicities	

Table 4.1: WORLD acoustic features

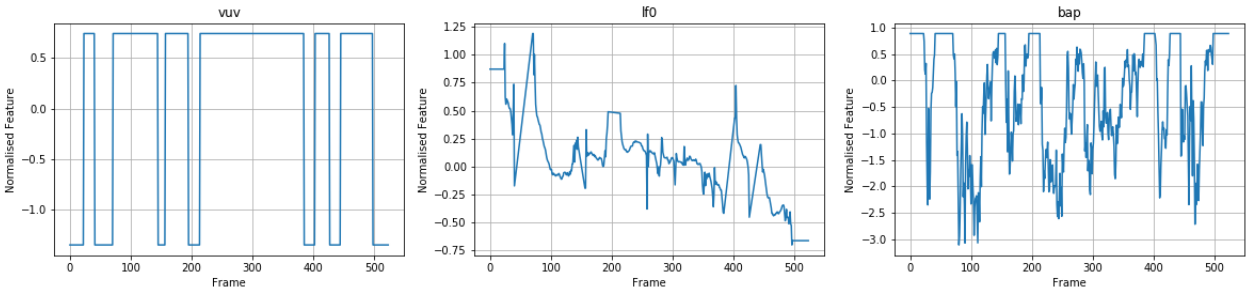


Figure 4.5: Example normalised WORLD features for speaker 301, sentence 117

A vocoder encodes and then decodes speech: in general, we would expect there to be a loss of quality as a result. The WORLD **vocoder reconstructions** of the VCTK data are pretty good, but you can definitely hear the loss of quality. The WORLD representation of the audio serves as ground truth for VoiceLoop, and gives an upper bound on the achievable quality of the model. On the project website, unless otherwise stated, 'ground truth' samples refer to those reconstructed from the WORLD features. Other neural TTS systems achieve higher quality by using neural vocoders such as WaveNet[50], SampleRNN[34] or WaveRNN[25].

Interpreting the LF0 Feature

To help build some intuition for the subjective interpretability of these features, I took the set of features corresponding to a particular VCTK example and replaced the LF0 feature sequence with a synthetically-generated sequence that followed a parabolic trajectory starting at a high pitch, falling to a low at the mid-point, and the rising into the close: see figure 4.6. All other features were left untouched. I then passed these features into the WORLD synthesizer and listened to the results: the sample[12] exhibits a clear drop and then rise in pitch, in contrast to the flat pitch of the baseline. Figure 4.7 compares the spectrograms corresponding to the baseline and to my generated sample: the eagle-eyed can spot the indications of falling pitch if the left side of the middle plot, and rising pitch on the right hand side.

[12] `https://richardsterry.github.io/msc-project/world_features.html`

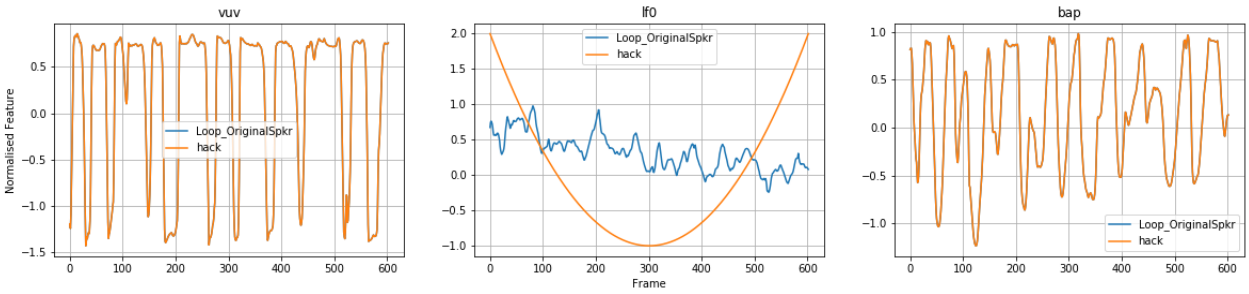


Figure 4.6: Example of manually overriding the LF0 WORLD feature

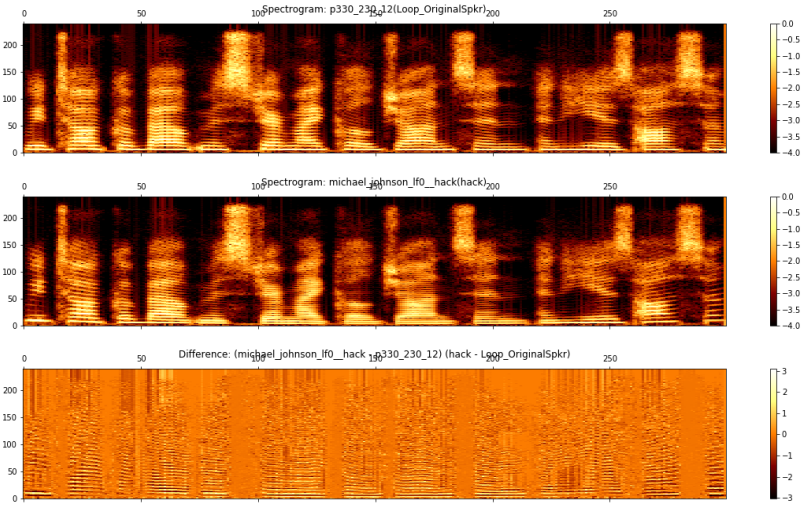


Figure 4.7: Spectrogram comparison with manually-overriden LF0 WORLD feature

4.3 Baseline Models

I trained VoiceLoop on two subsets of VCTK data: these provide the baseline models for all my later work.

VCTK-US-21

The first model uses the pre-computed dataset provided by FAIR for the subset of 21 VCTK speakers with American accents. Other than the technical overheads of setting up AWS, CUDA, PyTorch etc. this was a reasonably straightforward process and essentially involved following the instructions on the GitHub page.

The dataset is already separated into training and validation sets, with around 10% of data reserved for validation. All speakers are included in both training and validation: the validation set includes unseen *sentences* for each speaker, but all the speakers have been seen before. One weakness of the approach is that VCTK includes some common or repeated sentences that were given to more than one speaker. It's possible that a given sentence will appear in the training set for one speaker, and the validation set for another. This may give the model some useful clues

when it generates a previously-seen sentence for a different speaker. Note that there is no separate test set. The evaluation of generated speech is difficult, as we will see in section 4.4.5, and simply measuring the loss on a test set would not be especially meaningful. To test a model, we can always use it to generate speech for arbitrary text scraped from any available source and then evaluate the results.

VoiceLoop-VCTK-US-21 is relatively quick to train: although it takes a day or so to train the model fully, it's often possible to get an indication of whether a code change is having an impact more quickly than that. This makes the development cycle tolerable. I will generally present results using VCTK-US-21 where necessary, and save the full VCTK-All-107 for 'final' results.

VCTK-All-107

The two differences are that VCTK-All-107 uses:

- Full public VCTK dataset (107 speakers)
- Our own pre-processed features

As this version includes more speakers and is trained on a wider variety of accents, it gives a much richer experimental ground for understanding speaker representations. However, it also takes much longer to train fully: around four days to go to completion.

To create the VCTK-All-107 dataset, we had to pre-process the raw VCTK data into WORLD features ourselves. This requires silence trimming, use of the Merlin toolbox to run the WORLD analysis on the waveforms, and finally normalisation. Many thanks to Jiameng Gao, who kindly explained the process and provided me with the processed features. I split the data into training and validation folds using the same approach FAIR used for VCTK-US-21. This gives 39,313 training utterances and 4,316 validation utterances, each around a few seconds in length.

In preliminary work, I first built a model for the 21 US speaker subset using our own pre-processed features to check that we get a similar result to using the dataset supplied by FAIR. Results were comparable. I then built a model on only the 19 Scottish speakers from our dataset to check that the model would train on a similar-sized, disjoint subset. Again, results were comparable. This gave confidence that everything was technically in order before I began training the full model.

4.4 The VoiceLoop Model

I now move to describe the VoiceLoop model in detail. In section 4.4.1 I lay out the formalism for the problem. In section 4.4.2 I outline the architectural components. Both sections closely follow the original VoiceLoop paper[48].

4.4.1 Formalism

Consider the i^{th} example in the dataset: a $\{\text{text}, \text{wav}\}$ pair for speaker s^i. After pre-processing into features, we present the model with the triplet $\{\mathbf{p}^i, s^i, \mathbf{x}^i\}$, where:

- $\mathbf{p}^i = \{p_1^i, p_2^i, ..., p_{l^i}^i\}$ is the sequence of l^i phonemes extracted from the text transcript. Each phoneme p_j is an integer ID that represents a single phoneme in the dictionary, i.e. $p_j \in 1, 2, ..., 42$
- $s^i \in \{1, 2, ..., 107\}$ is the speaker ID
- $\mathbf{x}^i = \{\mathbf{x}_1^i, \mathbf{x}_2^i, ..., \mathbf{x}_{T^i}^i\}$ is the sequence of WORLD feature values $\mathbf{x}_j \in \mathbb{R}^{63}$ at each of the T^i steps

The learning task is to generate a predicted sequence $\{\hat{\mathbf{x}}^i\}$ of WORLD features given the inputs $\{\mathbf{p}^i, \mathrm{s}^i\}$, i.e. to generate a sequence of acoustic features (which can be deterministically converted to a waveform) for a given piece of text and a speaker ID, i.e. text-to-speech. During training we aim for the model to reconstruct the training $\mathbf{x}^i$ as a regression task, viewing the ground-truth feature sequence to be the only correct target and measuring all predictions through their deviation from this target.

In general, we would view the prediction of speech for a given text/speaker pair to be a generative model. The task is underdetermined in the sense that there are very many possible ways of a given speaker speaking a given sentence. The speech could be influenced by other unknown factors such as the sentences that came before or after, who the speaker is talking to, the level of formality and other contextual details, their emotional state, how tired they are, and so on. In addition, there will be random fluctuations in pitch, utterance length and so on.

We can therefore view the problem of TTS as modelling a conditional distribution over all possible feature sequences in an autoregressive manner, where at each step t the model (parametrized by ξ) gives a probability distribution over features given the sequence up to that point, the phonetic transcript and the speaker ID:

$$p_\xi(\mathbf{x}|\mathbf{p}, s) = \prod_{t=1}^{T} p_\xi(\mathbf{x}_t|\mathbf{x}_{1:t-1}, \mathbf{p}, s)$$

Our task would then be to build a model that generates plausible (high likelihood) sequences.

In fact, VoiceLoop is a fully deterministic autoregressive model. At each step it outputs a single most likely set of features. It would be possible to alter the structure to introduce randomness in the generated speech (e.g. in chapter 8 I experiment with a VAE in the encoder).

4.4.2 Model Architecture

In this section, I explain in detail VoiceLoop's **encoder** and **decoder** networks, as shown in the architecture diagram of figure 4.8. For more detail, Table 1 of the VoiceLoop paper[48] gives a very useful summary of the components and their dimensionality.

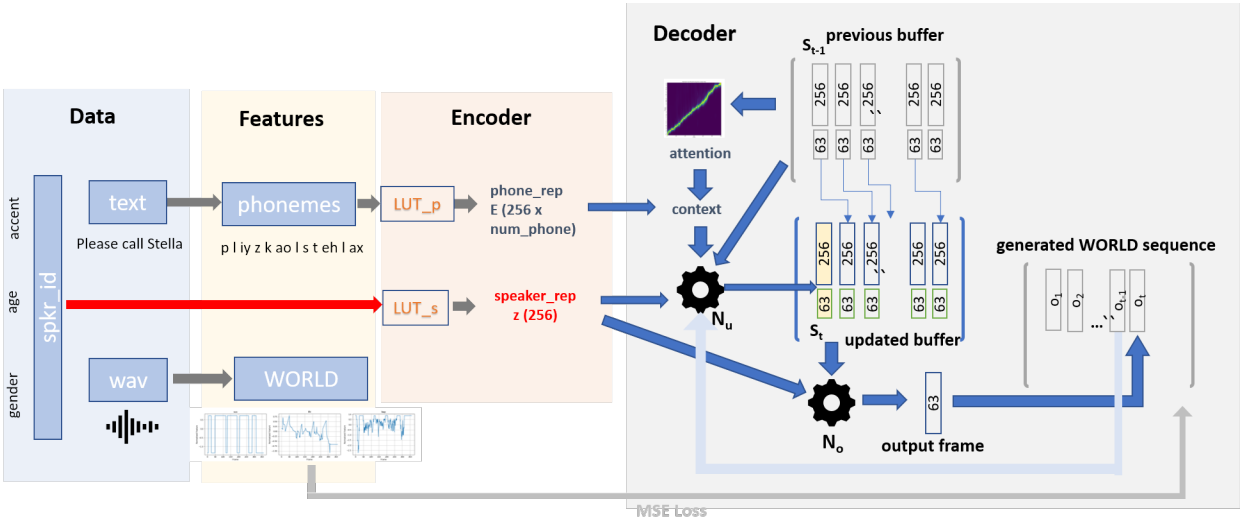


Figure 4.8: Architecture of the VoiceLoop model (repeat)

A. The Encoder: Encoding Input Features

Speaker Encoder Each speaker ID $s \in 1, 2, ..., 107$ is represented by a single vector z in a 256-dimensional embedding space, $z_s \in \mathbb{R}^{256}$. The mappings from speaker ID to z are stored in a lookup table LUT_s that is learned during training. In chapter 5 I discuss in detail the properties of the speaker vectors that VoiceLoop learns during training, and in the rest of my report I investigate various methods for augmenting VoiceLoop with mechanisms to force it to learn the speaker vectors in an interpretable way.

Phoneme Encoder Each phoneme p^i_j in the sequence $\mathbf{p}^i$ is independently encoded into a 256-dimensional vector representation using a learned lookup table LUT_p. This yields a $256 \times l^i$ matrix E.

B. The Decoder: Generating Sequence of WORLD Features

The shifting 'buffer' is the core component of the VoiceLoop decoder[13]. Rather than using a standard RNN variant to model the autoregressive generation of output steps as is done in various other neural TTS architectures, VoiceLoop uses a structure with an explicit short term memory. The buffer is stored as a matrix $S \in \mathbb{R}^{d \times k}$, where the buffer memory size $k = 20$ and the buffer dimensionality $d = 256 + 63 = 319$ comprises an upper block of 256 and a lower block of 63. The k columns $S[1], ..., S[k]$ each represent the 'memory' output of a previous step in the sequence where column 1 was generated by the current step t, column 2 was generated by the previous step $t - 1$, and so on. At each step, we discard the oldest memory column $S[k]$, shift the other columns one to the right ($S[i + 1] = s[i], i = 1, 2, ..., 19$), and add into the leading column $S[1]$ a new representation vector u, which is a function of four factors including the prior state of S (we will see shortly.) The buffer is therefore able to represent short-term dependencies by retaining

[13]It was inspired by Baddeley's 'phonological loop' working-memory model, whence the name 'VoiceLoop'.

most of the information in the buffer from recent steps; longer-term dependencies through each new vector u depending on the full buffer; but also update with new information.

Initializing the Buffer S_0

Prior to the first decoding step, each column of buffer S_0 is initialized identically, with the upper block of 256 elements set to match the speaker vector z, and all 63 elements in the lower block set to zero. Initializing the buffer with speaker information helps the model to produce speech in the speaker's distinctive voice from the first step of the output sequence.

Generating Output Step $\hat{\mathbf{x}}_t$

For each step in the output sequence, we generate the next set of output features $\hat{\mathbf{x}}_t$ autoregressively as follows:

i) Attention Over Phonemes (Computing the Context)

The attention mechanism tells the decoder which phonemes in the sequence $\mathbf{p}$ are most significant for generating the output frame at time t. At each point in the output sequence we might expect to be most focused on a single phoneme $\mathbf{p}$, but also to factor in surrounding phonemes that may influence how the central phoneme is pronounced. A single phoneme may be cut short or partially elided depending on what comes next, for example. Traditional phonetic systems would consider diphone or triphone groups rather than isolated phones when determining the sound to make; here, the attention mechanism can achieve a similar result but learned from the data rather than engineered.

VoiceLoop uses the Graves attention mechanism[17], which is based around a Gaussian Mixture Model (GMM). The attention uses ten components, each of which is a Gaussian distribution over the phonemes p_j in $\mathbf{p}$, where $j = 1, ..., l$ but is treated as a continuous variable for the purposes of the distribution. Each Gaussian component $i = 1, ..., 10$ is specified at time t by a mean $\mu_t[i]$ and variance $\sigma_t[i]$.

At each time step, the attention network N_a takes in the buffer S_{t-1} from the previous time step and generates three quantities for each component: the GMM prior $\gamma_t[i]$, the mean shift $\kappa_t[i]$, and the log-variance $\beta_t[i]$. N_a is a simple fully-connected neural network with one hidden layer of 638 nodes followed by a ReLU activation function, and then a linear projection layer. The attention weights are then calculated over the following steps:

- **GMM component weights** $\gamma_t^{'}$ are calculated by applying a softmax to the priors $\gamma_t[i]$. The weight given to the i^{th} GMM component is:

$$\gamma_t^{'}[i] = \frac{\exp \gamma_t[i]}{\sum_{j=1}^{10} \exp \gamma_t[j]} \tag{4.1}$$

- **GMM means are shifted** from their values in the previous time step according to the mean shift outputs $\kappa_t[i]$. Note that because of the exponential, the means can never decrease from step to step: this makes the attention monotonic, which is sensible given that we would always expect to move forwards through the phoneme sequence.

$$\mu_t[i] = \mu_{t-1}[i] + \exp \kappa_t[i] \tag{4.2}$$

- **GMM variance** $\sigma_t[i]$ is computed straightforwardly from the log-variance (the latter is more robustly generated by the neural network):

$$\sigma_t[i]^2 = \exp \beta_t[i] \tag{4.3}$$

- **Weighted GMM component densities to each phoneme** are calculated by finding the density of the i^{th} Gaussian at phoneme j and multiplying by the weight to the i^{th} GMM component:

$$\phi[i, j] = \frac{\gamma_t^{'}[i]}{\sqrt{2\pi\sigma_t^2[i]}} \exp - \frac{(j - \mu_t[i])^2}{2\sigma_t^2[i]} \tag{4.4}$$

- **Attention weights** $\alpha_t[j]$ are calculated for each location j in the phoneme sequence by summing over all 10 GMM components:

$$\alpha_t[j] = \sum_{i=1}^{c} \phi[i, j] \tag{4.5}$$

Finally, the attention weights are applied to the phoneme sequence embedding matrix E to give a **context vector** c_t for decoding at time step t.

$$\underset{256\times 1}{c_t} = \underset{256\times l_l\times 1}{E}\ \alpha_t \tag{4.6}$$

In the matrix multiplication, each 256-dimensional phoneme embedding is multiplied by the attention weight for that phoneme's position in the sequence and then the results are summed across all the phonemes.

ii) Updating the Buffer

First, we calculate a 319-dimensional **representation vector** u_t[14], which uses a neural network N_u to summarize the relevant information from four key sources:

- Previous buffer S_{t-1}: allows the network to learn dependencies on previous states
- Context vector c_t: embedding in phoneme space that the decoder should be generating

[14] $319 = 256 + 63$, i.e. the dimensionality of the phoneme/speaker embedding spaces plus the number of output features.

- Previous output features $\hat{\mathbf{x}}_{t-1}$: explicit dependency on the previous step
- Speaker vector z: allows the network to factor in speaker dependence

In order to pass these four inputs to the network, we first make the context vector c_t speaker-dependent by adding a tanh-squashed projection of the speaker vector z; then, we append the previous output feature vector $\hat{\mathbf{x}}_{t-1}$ to the end to give an augmented context vector C_t:

$$\underset{319\times1}{C_t} = \begin{bmatrix} \underset{256\times1}{c_t + \tanh(F_u z)} \\ \underset{63\times1}{\hat{\mathbf{x}}_{t-1}} \end{bmatrix}$$

Then we concatenate C_t with the previous buffer S_{t-1}, and pass the resulting matrix into the network N_u to yield a 319-dimensional representation vector u_t. N_u is another simple fully-connected network with a single hidden layer of size one tenth of the number of input elements, ReLU activations, and a final linear projection.

$$\underset{319\times1}{u_t} = N_u\left(\begin{bmatrix} \underset{319\times20}{S_{t-1}} & \underset{319\times1}{C_t} \end{bmatrix}\right)$$

Finally, we generate the new state of the buffer S_t by inserting the representation vector u_t into the first buffer column $S_t[1]$, and copying across the first 19 columns from the previous buffer state into columns 2-20, i.e. all shifted one position to the right. The final column from the previous buffer, $S_{t-1}[20]$, is discarded.

iii) Generating the Output Features

The WORLD feature outputs at time t are computed by applying a further simple network N_o to the buffer S_t, adding a linear projection of the speaker vector (using F_o) to give yet another site for learning speaker-dependence, and then projecting the resulting matrix from 256-dimensions down to the 63-dimensions required for the WORLD vector using linear projection F_w:

$$\underset{63\times1}{\hat{\mathbf{x}}_t} = \underset{256\to63}{F_w}\big[N_o(S_t) + F_o z\big] \tag{4.7}$$

$\hat{\mathbf{x}}_t$ is appended to the existing output sequence $\hat{\mathbf{x}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_{t-1}\}$, and then the decoder moves on to the next step $t+1$.

4.4.3 Training

MSE Loss & Teacher Forcing

During training, we compare the feature outputs $\hat{\mathbf{x}}$ to the ground-truth feature sequence $\mathbf{x}$ to measure how well the network has reconstructed the example. The loss is measured using a mean-

square error (MSE) on a frame-by-frame basis, with all time steps t and vocoder features being equally weighted. The loss on example i is given by:

$$\mathcal{L}^i = \frac{1}{T^i} \sum_{t=1}^{T^i} ||\mathbf{x}_t^i - \hat{\mathbf{x}}_t^i||^2 \tag{4.8}$$

A problem with this measure is that it requires $\hat{\mathbf{x}}^i$ and $\mathbf{x}^i$ to be temporally aligned. Otherwise, barely perceptible differences in $\hat{\mathbf{x}}^i$ such as an extra few frames of silence at the start, or a fractionally slower speech rate, could lead to to huge divergences when comparing features on a frame-by-frame basis, and hence large measured losses for a speech sample that a human would judge as almost identical. We would prefer to judge a sample by its likelihood under a model rather than a simple reconstruction error, but that is a far more difficult problem. During training, we therefore force alignment between $\hat{\mathbf{x}}^i$ and $\mathbf{x}^i$ by using a teacher-forcing technique.

The VoiceLoop codebase uses a simple version of **teacher-forcing**. At each decoder step, rather than using the predicted output features from the previous step $\hat{\mathbf{x}}_{t-1}$ as an input into the buffer update, we instead pass the ground truth features plus noise: $\mathbf{x}_{t-1} + \eta$, where $\eta \sim \mathcal{N}(0, \sigma\mathbb{I})$ and σ is a scalar hyperparameter. Using $\mathbf{x}_{t-1}$ prevents the generated sequence from straying out of alignment, while the random noise forces the model to learn robustness to imperfect inputs.

The VoiceLoop paper[48] states that they trained their system using the modified technique of **semi-teacher forcing**, which addresses a key weakness in simple teacher-forcing: the system only needs to learn to predict one sample ahead, with no constraints on how far it can diverge over longer periods, and may therefore generalize poorly to test examples where no ground-truth is available. The input at step t is now an average of the previous output and ground truth, which penalizes systematic drift while always pulling back towards correct alignment:

$$\frac{\mathbf{x}_{t-1} + \hat{\mathbf{x}}_{t-1}}{2} + \eta$$

We could extend this further using a **scheduled sampling** technique[5] in which at each step, either $\mathbf{x}_{t-1}$ or $\hat{\mathbf{x}}_{t-1}$ is used according to some probability p. Lower values of p mean that the model generates autoregressively without teacher-forcing over multiple steps, which allows significant drifts from ground truth to be penalised; but periodically the sequence reverts to ground truth. This encourages the model to learn to generate coherent sequences. Reducing p according to a schedule could allow the model to learn to generate only the next step or two at first, but over time encourages it to learn over longer horizons.

Although the VoiceLoop paper states that they used semi teacher-forcing in their research, the published codebase uses simple teacher-forcing. For consistency with the baseline I also used simple teacher-forcing in my project; however, this would be an interesting avenue to pursue in further work.

Training Details

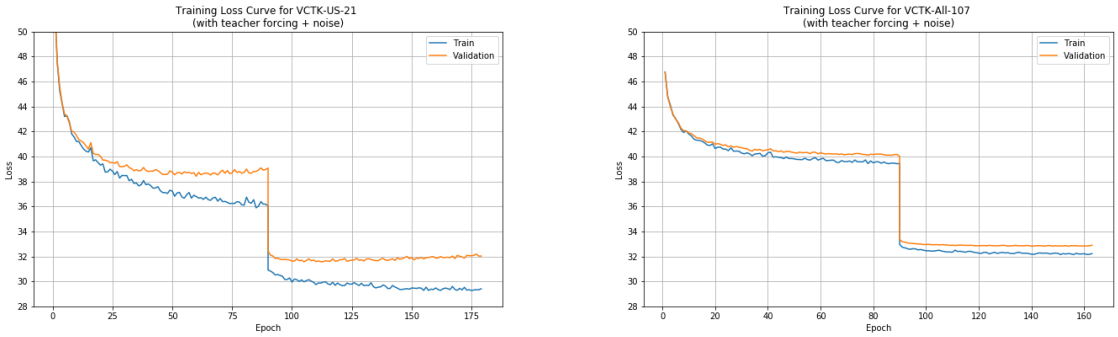
The network parameters are trained by sampling mini-batches of 64 training examples, running forward passes of sub-sequences through the network, calculating the loss metric, and then propagating gradients back through the network using truncated back-propagation through time (TBPTT). The parameters are updated using the Adam optimizer with a fixed learning rate of 10^{-4}, and using gradient clipping and checking to handle exploding gradients.

I trained the models in two stages essentially as recommended in the VoiceLoop paper[15]:

- Initial phase: 90 epochs with noise $\eta = 4$ and TBPTT sequence length of 100 steps
- Second phase: 90 epochs with noise $\eta = 2$ and TBPTT sequence length of 300 steps

The codebase uses PyTorch 2.7. I trained using a AWS p3.2xlarge instance, which includes one NVIDIA Tesla V100 GPU.

Figure 4.9 shows the training curves for VCTK-US-21 and VCTK-All-107. For both models, training loss continues to fall over the 180 epochs, but validation loss eventually begins to rise as over-training kicks in. The VCTK-US-21 model achieves a better validation loss of 31.6 versus VCTK-All-107's 32.8 - its problem is slightly easier, with less variation across speakers. The step down in MSE after epoch 90 is caused by the teacher-forcing noise dropping from 4 to 2.



(a) VCTK-US-21 (min valid loss = 31.6)

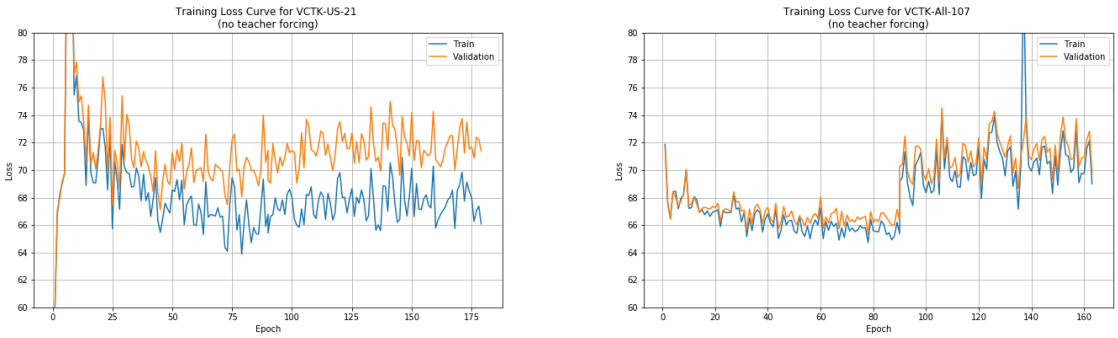
(b) VCTK-All-107 (min valid loss = 32.8)

Figure 4.9: Training loss curves: teacher forcing used during evaluation

Teacher forcing must still be used when evaluating the models on the validation sets: otherwise, the generated speech could move slightly out of alignment and be very heavily penalised, even though it might sound almost no different to the human ear. Humans don't care about a slight shift or warp in time - that's all within the normal range of vocal variation. But the MSE method would punish it severely. For comparison, figure 4.10 shows comparable curves on the same sequences of models but where the generation is done **without** teacher forcing. The losses remain very high throughout, and actually get worse when we enter the second phase of training - although the model is learning to generate more robust longer sequences, it appears that they're

[15] although my machine had insufficient GPU memory to use 1000 step sequences on the second phase, so I had to use 300

moving slightly further away from the ground truth samples, which doesn't necessarily matter at all.



(a) VCTK-US-21

(b) VCTK-All-107

Figure 4.10: Training loss curves: without teacher forcing during evaluation

This puts us in a slightly difficult position, in that when we compute our evaluations we're still 'helping' the model along by feeding it the ground-truth features from the previous steps. The validation MSE therefore doesn't really measure the model's ability to generate speech unaided. We will return to this shortly when we discuss evaluation.

C. Synthesizing the Speech Waveform

When testing, or when using a trained model to generate actual speech waveforms, there is no teacher forcing or added noise, and the decoder generates each new step based on the previously-generated output $\hat{\mathbf{x}}_{t-1}$. The final sequence of WORLD features $\hat{\mathbf{x}}$ is then passed into the synthesis component of the WORLD vocoder to generate the output waveform (i.e. .wav file.). This is done using the Merlin toolbox.

4.4.4 An Example

I'll now work through an example to illustrate the components in action. I've picked sentence 230 for subject 330. You can listen to the original sample on the website here[16].

- First, we take the transcript:

  We talk about Mr Michael Johnson, and he is awesome.

- Translate it into a sequence of 36 phonemes:

  w iy t ao k ax b aw t m ih s t er m ay k ax l jh aa n s ax n ae n d hh iy ih z aa s ax m

- And then look up the phonemes in the phoneme embedding table to give a 256x36 phoneme representation.

[16] `https://richardsterry.github.io/msc-project/vctk_us_22_samples.html`

- In this case we are trying to reconstruct the training example, so want to generate a speech sample in the same voice 330. We therefore look up speaker 330 in the embedding LUT to find the 256x1 representation for speaker 330.
- The buffer is initialized with this speaker representation in the top parts of all 20 buffer positions, and zeros in the bottom parts.
- The decoder now iterates over the output steps.
- In each step, it first calculates the attention distribution over the phonemes, as shown in figure 4.11. Note that the attention is monotonic: as we increase through the frames, it always moves forwards through the phoneme sequence. You can see how it spends longer attending to some phonemes than others, e.g. the 'ao' phoneme in position 4 (long vowel sound in 'talk') is a long sound spanning many frames; in contrast, the 'm' in 'mister' is a short sound that receives significant weight in only a few frames around the 150 mark.

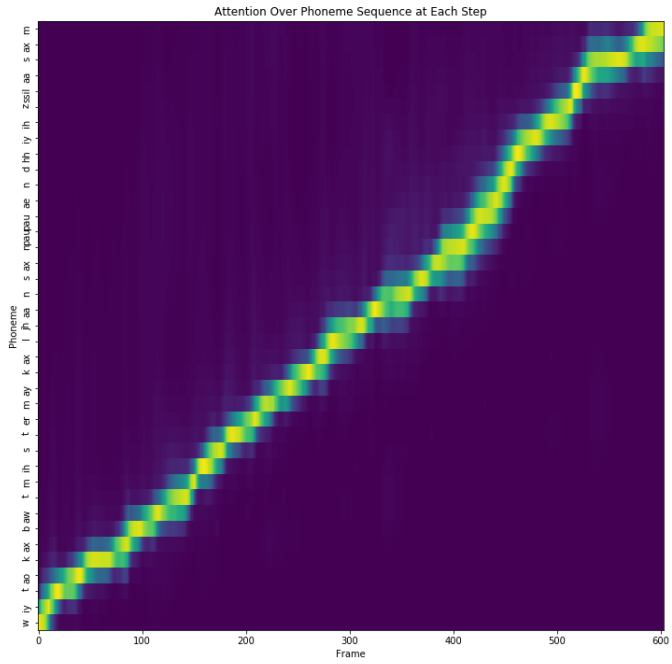


Figure 4.11: Examples of attention over phonemes

- As the decoder progresses, it generates sequences of WORLD features one frame at a time. Figure 4.12 shows how the VoiceLoop generated output sequences compare to the ground truth sequences for the VUV, LF0 and BAP features. Figure 4.14 shows the same thing for the bank of 60 MGC features. The model does a pretty good job of recreating these features. Note that in this example I turned teacher forcing *off*, so the model is generating speech freely with no guidance frame-to-frame. Therefore, some temporal warping is expected.

- Finally, the WORLD features are passed to the synthesis component of the WORLD vocoder to convert them into a waveform. You can listen to the sample here to judge the results yourself.

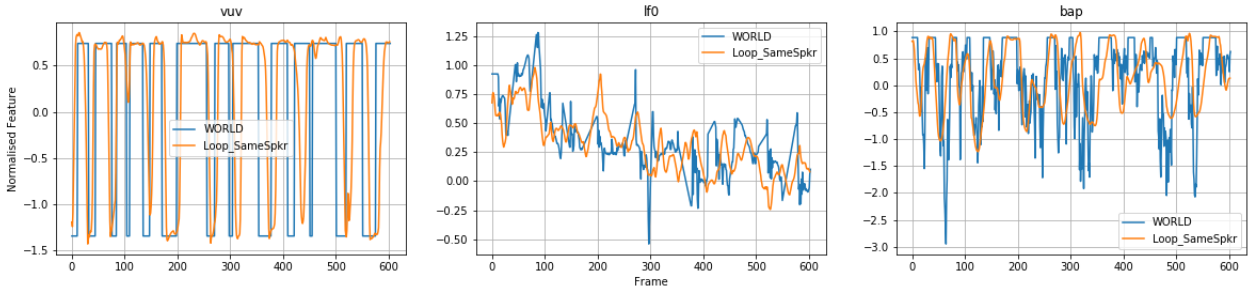


Figure 4.12: Comparison of WORLD features synthesised by VoiceLoop to the ground truth features: VUV, LFO, BAP

Aside: Loss by Feature

Something that concerned me early in the project was whether an *equally-weighted* mean square error loss was appropriate. VoiceLoop treats all 63 normalised WORLD features equally, even though it might be the case that certain features are more perceptually important than others. For example, lf0 (pitch) and vuv (voiced/unvoiced) would appear to matter a huge amount to our perception of speech. It's not obvious that they should be weighted the same as the much noisier higher-order mgc terms. I put quite a lot of effort into decomposing the MSE by feature, speaker and so on looking for any interesting patterns. The most interesting of these is shown in figure 4.13: loss by feature. The blue crosses show each feature's contribution to MSE loss in the model as at epoch 10. The orange crosses show the same thing in the trained model at epoch 160. When the model is randomly initialised before any training, all the contributions would be about 1. It's clear that VoiceLoop quickly learns the lower-order mgcs, lf0, vuv and bap. Over training, it continues to improve all these a little more. In contrast, it never manages to learn the higher-order mgcs well. It seems that these are just a good deal noisier and very hard to learn. What's not clear is whether this matters: are the higher-order mgcs pure noise that we shouldn't be concerned about reconstructing, or are they where the real fine details of speech and speaker identity are to be found?

I think this is an area of the model that would benefit from further attention in future work.

4.4.5 Evaluation

We've seen that during training, the model is evaluated by generating validation set samples using teacher-forcing, and then measuring the MSE loss. When it comes to testing, the model's ability to generate high-quality speech from completely unseen text without teacher forcing is what we really care about. Here, MSE cannot be used. We have to come up with something that better captures our objectives.

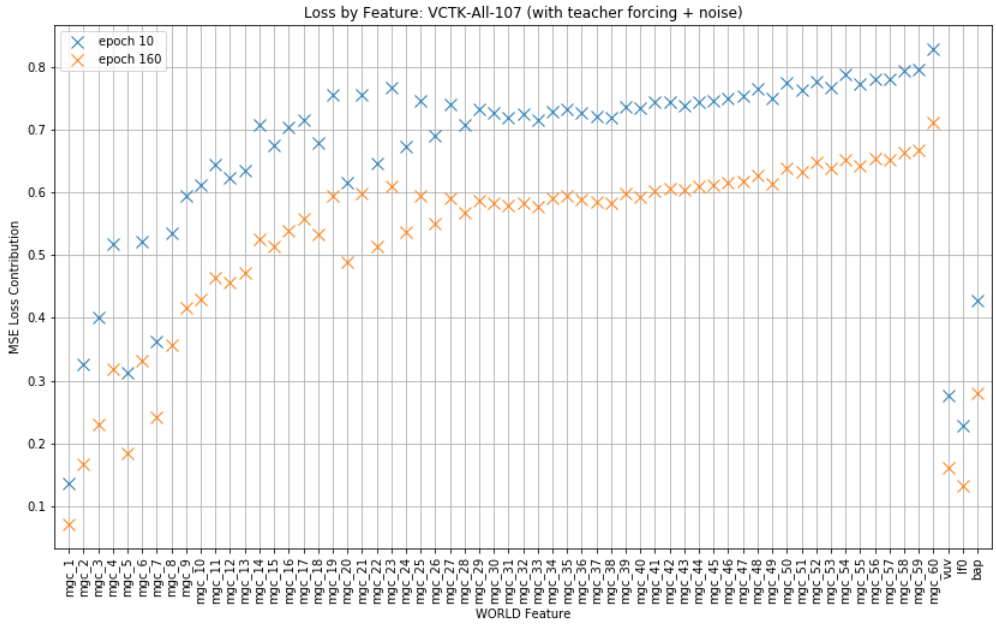


Figure 4.13: Loss by WORLD feature at two stages of training

Evaluation of generative models is notoriously difficult.

There is not one single correct way for a given speaker to say a given sentence. Plenty of variability is permitted in real life: a bit faster, a bit slower, a bit higher, slightly different emphasis or prosody. Sometimes these variations will be explained by other contextual factors that the model doesn't take account for, e.g. what the previous sentence was; the broader context; who the speaker is speaking to; how they're feeling (emotional state); if they're ill, tired, whether their lungs are full of air or they need to breathe. In other cases, there will just be random variation - that's how we talk. In fact, some variability is desirable: human listeners apparently expect variation, and speech will be judged to sound less natural (and be harder to listen to) if it comes out exactly the same way every time.

VoiceLoop learns to recreate the exact particular version of the training sample - exactly how the speaker said it on that occasion. It doesn't learn a distribution over all the plausible ways the speaker could have said it, on a different day, in a different context, or if they'd simply had a little longer since their last breath. So while the MSE loss is a valid way of training a system to reconstruct training examples, it isn't necessarily a good evaluation metric for generated speech for the validation set; and it's no use as a metric for evaluating speech synthesised from free text.

Speech evaluation is often addressed along dimensions such as:

- **Intelligibility**: Can we understand the sentence being said?
- **Naturalness**: Does it sound like real human speech, rather than tinny or like a robot?

- **Adaptation**: Does it sound like the person it was supposed to be?

The gold standard tests for these all require carefully-managed **subjective tests** in which real humans are asked to listen to and rate speech samples. There is currently no good substitute for human listening tests, and in industry listener tests are run all the time to evaluate whether changes to a model have indeed made an improvement. **Mean Opinion Score (MOS)** is generally the standard approach. Listeners are recruited anonymously, typically using a crowd-sourcing platform such as Mechanical Turk, to listen to sets of samples on a blind basis and rate them on a scale of 1 to 5 in 0.5 point increments. By collecting large numbers of ratings, a TTS system can be given a MOS rating (with a confidence interval) for intelligibility, naturalness etc., and systems can be compared. For reference, single-speaker Tacotron2[42] achieved a phenomenal overall MOS score of 4.5, almost indistinguishable from the 4.6 given to ground truth human speech. In contrast, [42] quotes MOS scores for its baseline concatenative and statistical parameteric systems of 4.2 and 3.5 respectively. VoiceLoop[48] reported around 3.6 on VCTK, which should be compared to multi-speaker Tacotron's[22] 4.1.

Other systems such as MUSHRA, which asks listeners to compare samples rather than give absolute ratings, are arguably better, but MOS is still typically the default approach.

Obviously it wasn't practical to run listener tests as part of my project. As a substitute I listened to lots of samples myself, and make plenty available on the website in case the reader wants to do the same. My project is largely concerned with speaker identity and changes in perceived speaker attributes such as gender and accent, rather than the overall quality of speech as such, so the lack of an overall quality evaluation framework didn't cause too may problems. To help judge the quality of identity and gender, I developed speaker and gender recognition classifiers, to be discussed in section 5.1.2. In future work, it would be necessary to use a subjective listener framework such as MOS or MUSHRA to obtain a better handle on the results.

4.5 Summary

In this chapter, I have worked through the VoiceLoop pipeline. I started by describing the VCTK dataset, and then discussed the pre-processing steps that convert raw data examples into phoneme sequences and WORLD features. I then explained VoiceLoop's encoder and decoder networks, and presented my baseline models on VCTK-US-21 and VCTK-All-107. Finally, I discussed the difficulties in evaluating the quality of synthesized speech, a problem that is found with many generative models.

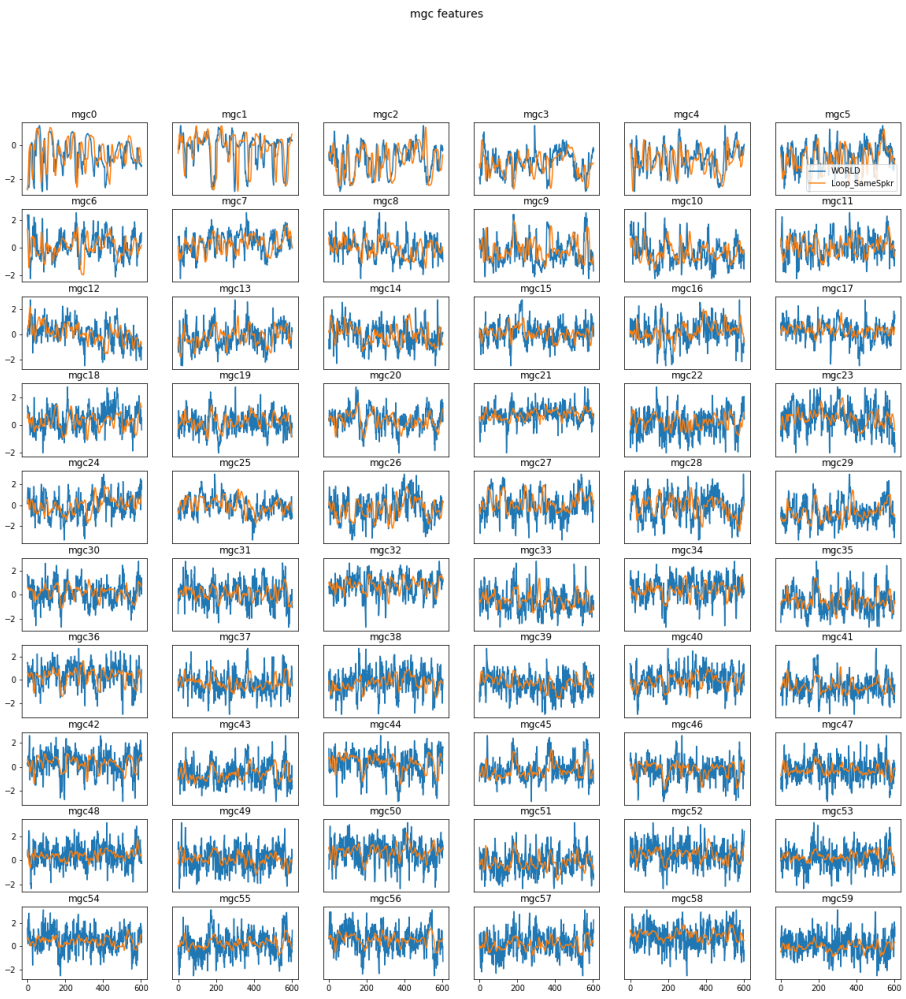


Figure 4.14: Comparison of WORLD features synthesised by VoiceLoop to the ground truth features: MGC

Chapter 5

Speaker Embeddings

In this chapter I take a closer look at how VoiceLoop represents the voices of different speakers.

Section 5.1 establishes using both subjective and objective methods that VoiceLoop can successfully synthesize speech in the voices of speakers from the training set, and is therefore a good *multi-speaker* neural TTS model.

Section 5.2 shows that the speaker space is *interpretable*, with similar voices being found close to each other in the space, and different voices further away.

Section 5.2.3 demonstrates that we can generate meaningful changes in speech by manipulating the speaker vector, giving us some degree of control.

Section 5.3 concludes by noting that although the speaker space is interpretable and manipulable, it is heavily entangled, which limits what we can achieve.

5.1 VoiceLoop Can Represent Known Speakers

VoiceLoop represents each of the speakers (21 in VCTK-US-21; 107 in VCTK-All-107) with a single, fixed vector in a 256-dimensional embedding space. The only constraint imposed is that the embedding vector has a maximum norm of 1: if higher, the vector is renormalised. This is the speaker's 'average' embedding that gives the minimum MSE across all that speaker's utterances, even if there is significant variation in speaking style (i.e. prosody) across the utterance set.

Can VoiceLoop represent speakers accurately? This isn't guaranteed by the results we've seen so far. MSE is a rough measure, and even if the model trains well it's quite possible that the model could have learned a single average voice and ignored the individual speaker embeddings entirely; or perhaps learned gross differences between speakers but not enough for the voices to be identifiable. Or perhaps it can represent the voices of some speakers and not others.

In this section, I will establish first qualitatively and then quantitatively that VoiceLoop can indeed represent the voices of speakers on which it has been trained.

5.1.1 Qualitative: Listening to Samples

VCTK-US-21 Samples

Please begin by listening to the samples on VCTK-US-21.

Each row in the table corresponds to the voice of one of the 21 speakers in the VCTK-US-21 dataset.

The first two columns of the table give examples of ground-truth audio (synthesized from WORLD features) for each of the US speakers: the first column is the standard sentence "Please call Stella" sentence, while the second is a sentence picked at random from the training set. The final two columns give examples of speech generated by passing the VCTK-US-21 model the text string and a speaker identity: the first column is the 'Stella' sentence and the final column gives the 'Michael Johnson' sentence.

You can hear that the generated samples sound *different* as the speaker id is varied, and are generally intelligible and roughly natural-sounding. In most cases, they sound at least *similar* to ground-truth examples of the corresponding speaker. Therefore, at a subjective level we can see that VoiceLoop is indeed able to represent the voices of known speakers.

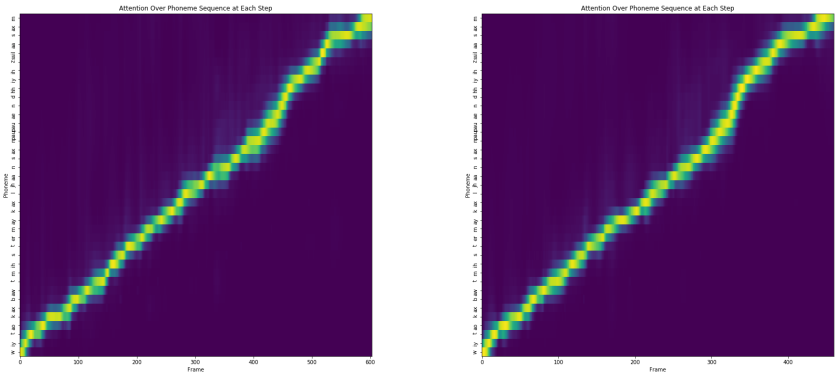
It's instructive to look at which internal model components change when the speaker ID changes. The **encoder** uses the same embedding matrix for the phoneme sequence regardless of speaker ID: it's only the speaker embedding vector that changes. As a result, the **decoder** generates a different attention sequence for each speaker. Figure 5.1 shows the attention profiles for the 'Michael Johnson' sentence for two speakers: the original female speaker (for whom this sentence is in the training set) and another, male speaker. Although the profiles are similar, there are clear and interpretable differences that correspond to minor perceivable differences in the audio samples. For example, the male speaker's sequence is shorter (fewer frames on the x-axis), moves a fraction slower through the 'ay' of 'Michael', and is a little steeper (i.e. faster) through the 'and'.

Figure 5.2 shows how some of the output features differ between the original speaker (female) and a different speaker (male). The different (male) speaker speaks more quickly (the sequence is shorter), and has a clearly lower pitch (LF0).

Figure 5.3 shows the resulting differences in spectrograms computed from the synthesized waveforms. Speaker 14 speaks more quickly, has more energy at lower frequencies, and less distinct boundaries between phonemes.

VCTK-All-107 Samples

VCTK-All-107 presents a similar table of audio samples for the much more diverse VCTK-All-107



(a) Original speaker (female) (b) Different speaker (male)

Figure 5.1: Comparison of attention over phonemes for two speakers, using the 'Michael Johnson' sentence.

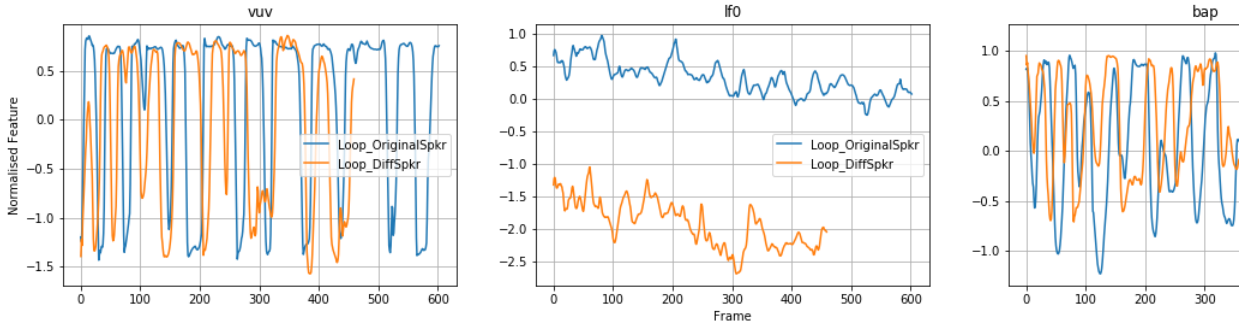


Figure 5.2: Comparison of three WORLD features for two speakers, using the 'Michael Johnson' sentence.

model. For this model, the first generated sample of speech is the 'Michael Johnson' sentence, while the second is based on a longer fragment of text I took from a New Yorker article.

This model is not as robust as VCTK-US-21. Many of the generated samples are not intelligible, with the attention mechanism sometimes failing to articulate all the phonemes in the sentence, or spending far too long on some phonemes, or even failing to speak at all (sometimes generating a 'breathy' background noise.) I believe it would be possible to improve results through additional training, including with longer sequence lengths inside the TBPTT and with more careful mixed teacher forcing or scheduled sampling. The model could also be improved to enhance robustness: the follow-up to the VoiceLoop paper [39] included some ideas on modifying the attention mechanism, for example. However, to some extent the VCTK-All-107 dataset is just a more difficult problem: there is much greater variety across speakers, relative to quite a small amount of data per speaker and accent.

In my project I'm focusing on the ability of the model to *represent* speakers, rather than the robustness of the model per se, so I did not pursue these ideas and remained close to the benchmark

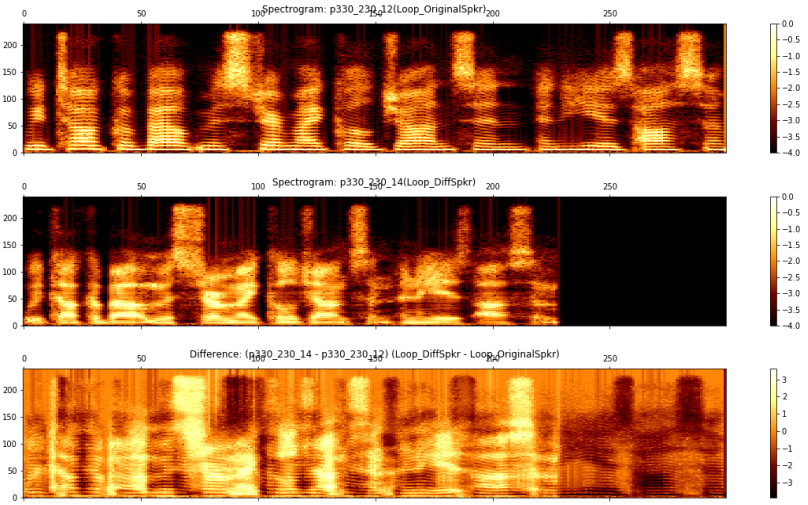


Figure 5.3: Comparison of spectrograms of the synthesized speech for two speakers, using the 'Michael Johnson' sentence.

model. In the rest of my report I will generally use successful examples to illustrate the ability of the model to represent speakers, but it should be borne in mind throughout that the model does struggle or even fail in many cases. However, even when the model struggles with an example, it's often still possible to recognise the speaker's distinct voice, and at least their gender and accent.

The first few rows of the table in VCTK-All-107 are actually quite poor quality, but there are many better examples further down. For example: Speaker 231 (English female); 232 (English male); 248 (Indian female) struggles but the voice is quite good; 260 (Scottish male); 266 (Irish female); 285 (Scottish male); 304 (Northern Irish male); 326 (Australian male); 302 (Canadian male, fast speech rate); 285 (Scottish male, fast speaker); 278 (English male); 273 (English male).

5.1.2 Quantitative: Speaker Classification Network

In the previous section we found that, subjectively, the models do a pretty good job of learning the voices of individual speakers in the training set and can generate samples that sound similar to a chosen speaker. We now move on to develop an objective measure to capture this: we build a simple machine learning model that, for a given sample of speech, predicts which of a known set of speakers spoke it.

Speech researchers have long worked on the problem of how to recognise a speaker based on a short sample of speech - the problem is much older than the recent neural TTS techniques. There are important applications in for example security systems (e.g. telephone voice recognition), and segmenting audio across separate speakers during voice recognition. Researchers differentiate between two related tasks:

- **Speaker Classification**: Train a model on a large set of known, labelled speakers. Receive a test sample of speech. Which of the known speakers is most likely to have spoken the test

sample?

- **Speaker Verification**: Train a model on a large set of speakers. Receive two samples of speech, not necessarily from speakers in the training set: one 'enrollment' and one 'test'. How likely is it that the test sample was spoken by the same person as the enrollment sample?

Currently, the best speaker recognition models based on deep learning models don't outperform more traditional feature-engineered systems such as 'I-vectors'. However, they are getting very close and are more than adequate for our needs here. We don't need perfect speaker recognition for our problem (this is not a bank telephone security system!) It's a useful metric but beyond a point the utility tails off.

Speaker Classification Architecture

The classification problem is as follows. Given a set of $\{\text{wav}, \text{speaker\_id}\}$ examples from the VCTK training set, build a model that predicts the speaker ID from the WORLD features corresponding to the waveform. The model is then evaluated on $\{\text{wav}, \text{speaker\_id}\}$ examples from the VCTK validation set. Note that this model is not specific to VoiceLoop or any other speech synthesis model: it knows nothing about VoiceLoop, and is trained completely separately.

I use a neural network based on applying two-dimensional convolutional layers to short speech samples, each represented as a sequence of 63 WORLD features. The network predicts the speaker label and is then trained using a standard classification cross-entropy loss. This is similar to what was done in [39], [1], plus various other papers.

The architecture is:

- 5 2D convolutional layers, each using 3x3 filters, 32 output channels, batch normalisation and ReLU activations
- Max pooling over time
- 2 fully-connected layers with 256 nodes and ReLU activations
- Linear projection layer with number of nodes matching the number of speakers in the dataset
- Softmax output to give speaker probabilities

Training

The model was trained using a standard cross-entropy loss and the adam optimizer with a learning rate of 10^{-3}, and random 300-step sub-sequences from each training example to help build robustness. I used the same training/validation split as in the VoiceLoop model. Note that in this case it would have been helpful to have a separate held-out test set on which to quote final performance: unlike in the VoiceLoop task, there is a clear success metric. However, given that I'm only using the speaker classification network for evaluation and that we can also test on unlimited

quantities of speech synthesized using VoiceLoop on unseen text, I didn't go back and re-split the datset.

As show in figure 5.4, the speaker classification network achieves almost perfect validation-set accuracy on VCTK-US-21 after 15 epochs of training. In fact it reaches more than 98% accuracy after the first epoch. It appears relatively straightforward to classify speakers based on acoustic features: presumably, even simple metrics such as average pitch would suffice to build a pretty good model. It would be interesting to run an ablation analysis in which some of the most important features (e.g. LF0 are removed) to see whether the classifier is still able to separate speakers using the remaining features.

Performance is not quite as good on VCTK-All-107, with validation set accuracy peaking at about 98%. However, this is still more than adequate for our requirements in this project.

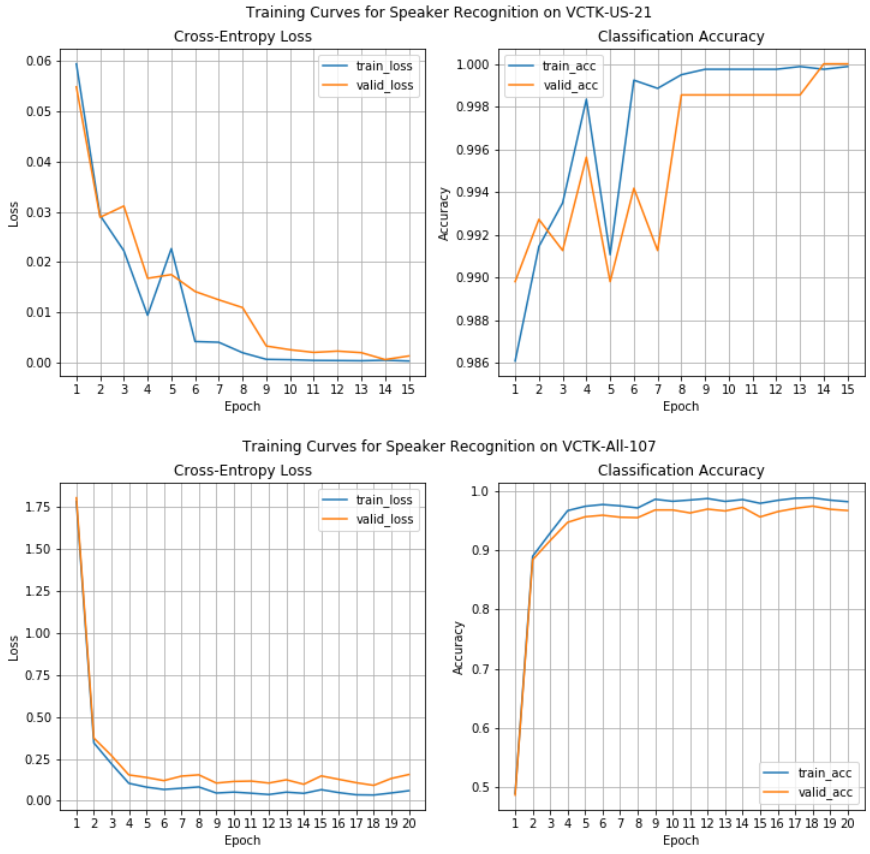


Figure 5.4: Training curves for the speaker classifier network

Performance at Classifying Samples Generated by VoiceLoop

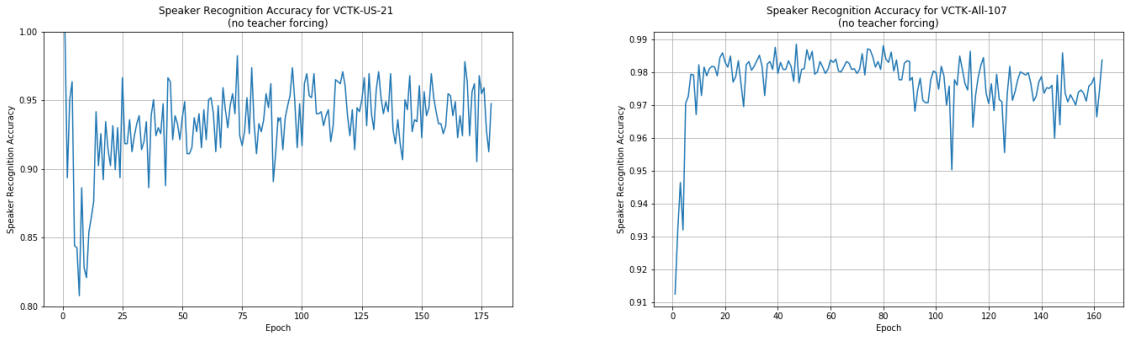
In the previous section, the speaker classifier was trained on examples from the VCTK training set and evaluated on examples from the **VCTK validation set** - this is a general model that has nothing to do with VoiceLoop. In this section, I move on to using this trained speaker classifier to evaluate how well VoiceLoop can express the voices of known speakers - are the **VoiceLoop**

samples good enough for the speaker classifier (trained only on original VCTK data) to be able to correctly predict the speaker?

Figure 5.5 shows the classification accuracy of the best trained speaker recognition models on synthesized VoiceLoop samples (sentences from validation set, samples generated with no teacher forcing) as VoiceLoop training progresses.

After one epoch of training, the samples generated by VCTK-All-107 are good enough for the classifier to guess the correct speaker 91% of the time. This rises to around 98% over the course of training. It's interesting that the accuracy falls slightly during the second phase of training when the sequence lengths are longer: I would guess that VoiceLoop is giving up a little of its ability to differentiate speaker voices to compensate for the more difficult task of working over longer sequences. If we use teacher forcing during evaluation, this rises to more than 99.5% - constraining the speech rate in this way makes the generated samples easier to identify.

Without teacher forcing during evaluation, the model trained on VCTK-US-21 is able to infer the correct speaker around 95% of the time (some of the American females sound very similar.) With teacher forcing, this reaches 100% after the first couple of epochs of training.



(a) VCTK-US-21

(b) VCTK-All-107

Figure 5.5: Training loss curves: without teacher forcing during evaluation

These results are consistent with the literature, which finds that automated speaker recognition systems are often better than humans. In fact, research shows that human aren't very good at judging *unknown* voices: automatic speaker verification systems are much better than humans. This stands in clear contrast to the related problem of judging naturalness of speech, where human judgement is far superior to automated techniques. However, automated speaker recognition systems don't necessarily pick up on the same details as a human ear (fine details of human-interpretable attributes such as accent or vocal habits, for example), but instead are more consistent in the application of acoustic features such as average pitch, range, harmonics etc.

In summary, our objective speaker classification network shows that VoiceLoop can successfully generate samples of speech in the voice of a selected speaker - or, at least, that it can do this well enough to fool a speaker classifier network, which may not pick up on the same acoustic features as the human ear when judging speaker identity.

Observations

During my research I experimented with various neural network architectures for the speaker classifier, including varying the number of convolutional layers (more depth), channels and filter sizes; dropout; mean-pooling rather than max-pooling; different sequence lengths for the samples; weight normalization; using a RNN after the convolutional layers. While this was instructive, I ultimately decided to keep the architecture simple and similar to what was reported in other papers.

One architectural decision is worth further comment. Following other papers, I used two-dimensional convolutional filters. Convolutional layers are useful where the problem is translationally invariant: in an image, for example, it doesn't matter whether an eye is in the top left of the image or the bottom right, it should still be detected as an eye feature. In audio, we clearly have time invariance: for the purposes of identifying a speaker, it doesn't matter much whether a sound comes at the start of the sequence or in the middle. However, it's less clear that we have invariance across the second dimension, which is the index of the WORLD feature. It doesn't seem correct to say that a pattern occurring across the first three WORLD features (MGC 1-3) should be treated the same as a pattern across the last three WORLD features (VUV, LF0, BAP). While there may be a meaningful order through the MGC features, the final three are clearly disjoint features; therefore, a convolution along the feature dimension seems inappropriate. Some other papers that use mel-spectrograms rather than WORLD features may be able to get away with 2D convolutions because the mel buckets are at least ordered in terms of increasing frequency, even if the spacing is not even, but this does not apply to the WORLD representation.

It would be useful to repeat the experiment using one-dimensional convolutions that rely only on temporal invariance and don't work across the feature dimension.

5.2 The Speaker Embedding Space is Interpretable

Having shown that VoiceLoop can successfully reproduce the individual voices of speakers in the training set, in this section I investigate how it *represents* the speakers. In section 5.2.1 I use word embeddings in NLP to introduce the idea of an interpretable embedding space; in section 5.2.2, I apply these ideas to VoiceLoop's 256-dimensional speaker embeddings.

5.2.1 word2vec

A naive language system might model words in a language as discrete atomic units with no structured relationships between each other: each word in the dictionary might be represented by its own one-hot vector. When the system learns about how the word 'apple' is used, it will simply file this under 'apple' and not be able to relate it to any other words such as 'banana' (fruit), 'grass' (green) or 'tennis ball' (small round object that can be thrown). Unable to share knowledge across words, the system will need huge amounts of data to learn what all the words

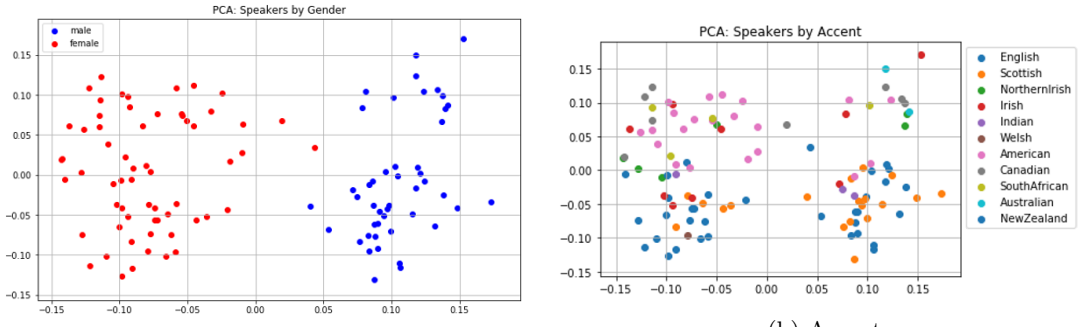
mean (sparsity), and it will lose out on many insights that might arise from understanding the ways in which words stand for similar things.

Models such as Mikolov et al's **word2vec**[35] learn to embed words in a continous vector space, rather than repesenting them as one-hot vectors. As they train, similar words become mapped to close-by points in the embedding space: words corresponding to fruits ('apple', 'orange', 'banana') tend to be close together. A well-trained model will also encode interesting semantic relationships about the relationships between words. As Mikolov et al showed, it is possible to find vector transformations in the space that correspond to relationships such as country-to-capital and male-to-female. word2vec can learn a meaningful, interpretable embedding space. Might the same be true of the speaker embedding space in VoiceLoop?

5.2.2 speaker2vec

Each speaker is represented by a single 256-dimensional vector, which the model learns during the training process to minimise the loss across that speaker's set of approximately 400 training utterances. The only constraint is that the maximum L2 norm of each vector is one: if greater than one, the vector is rescaled back to norm one. It's certainly not obvious that the speaker embeddings should be interpretable, with similar speakers having similar vectors. It's possible that the model would learn to push the vectors as far apart as possible, for example forcing speakers to maximally distant points on the unit hypershell, and memorizing each speaker's characteristics independently.

In figure 5.6, I apply a standard principal component analysis (PCA) to the 107 speaker vectors of the VCTK-All-107 model, plot each speaker against the values of the first two principal components, and then label with known attributes from the VCTK metadata. The plot 5.6a shows a very clear separation between male and female speakers. The plot 5.6b, with each speaker marked by accent, is harder to interpret. However, this is partly down to the number of accents, many of which have only a few speakers in the dataset.



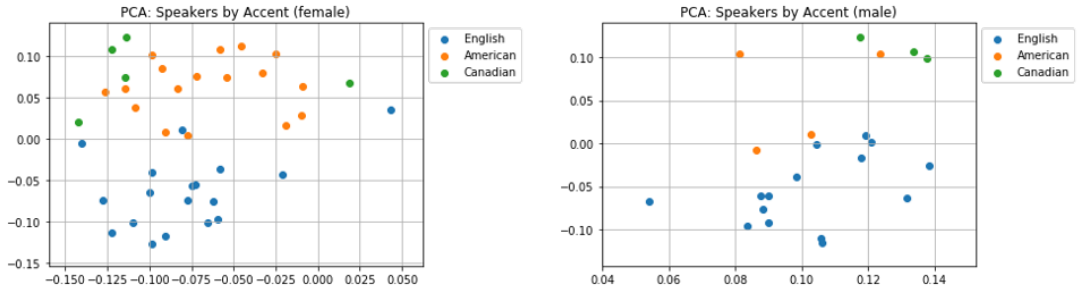
(a) Gender

(b) Accent

Figure 5.6: Speaker embeddings for VCTK-All-107 plotted by the values of their first two principal components, and labelled by gender/accent

In figure 5.7 I restrict the accent plot to English and North American speakers, with the plot

5.7a showing females and plot 5.7b showing males[1]. It is now clear than English/NorthAmerican speakers separate reasonably well, with the English speakers in the lower half and the North American speakers in the upper half. The vectors joining the average English point to the average North American point would be similar in the male and female plots.



(a) Female (b) Male

Figure 5.7: Speaker embeddings for only English and American speakers in VCTK-All-107

As a further illustration of the interpretability of the space, I checked whether nearby speakers in the space tended have similar metadata attributes. I used **cosine similarity** as the distance metric. Cosine similarity measures the angle between vectors, and is defined by $\frac{\mathbf{A}\cdot\mathbf{B}}{|\mathbf{A}||\mathbf{B}|}$ where $\mathbf{A}$ and $\mathbf{B}$ are vectors in an inner-product space. A similarity of 1 indicates that the vectors have the same orientation, whereas 0 indicates orthogonality. Figure 5.8 shows the nearest ten speakers to speaker 283, an Irish female. The most similar speaker is 266, another Irish female. A number of other close speakers are also Irish or Northern Irish. All the closest speakers are female. So this analysis also suggests that the speaker embedding space is interpretable.

	id	age	gender	accents	region	cosine_sim
55	283	24	F	Irish	Cork	1.000000
39	266	22	F	Irish	Athlone	0.241125
75	307	23	F	Canadian	Ontario	0.167515
60	288	22	F	Irish	Dublin	0.162790
50	277	23	F	English	NE England	0.162063
99	351	21	F	NorthernIrish	Derry	0.155133
12	238	22	F	NorthernIrish	Belfast	0.132402
14	240	21	F	English	Southern England	0.131049
13	239	22	F	English	SW England	0.129371
49	276	24	F	English	Oxford	0.119836

Figure 5.8: Ten closest speakers to speaker 283, based on cosine similarity of their speaker embedding vectors

5.2.3 The Speaker Embedding Space is Manipulable

Having shown that the space is interpretable, the next stage is to investigate whether we can manipulate vectors in the space. First, can we generate speech using speaker vectors that don't

[1]All plots are still using the same PCA decomposition run on the full 107 speaker dataset

correspond to any voices in the training set? Second, can we alter the perceived characteristics of generated speech by transforming the speaker vectors?

I experimented with generating speech using slightly perturbed speaker vectors, and found that the resulting samples sounded very similar but not identical. This isn't too surprising. During the training process, the speaker embeddings are constantly shifting around as network parameter weights are updated to reflect loss gradients. If the changes in vocoder feature output didn't vary relatively smoothly with changes in speaker vectors, then it would be tough to train the network at all (perhaps beyond a single, average voice.) So, it is possible to generate speech for vectors that don't correspond exactly to the current values for the known speakers. But what about if we make larger transformations in the space?

In figure 5.9, the large red and blue dots indicate the average vectors for all female and male voices. Define the **gender transformation** vector to be the vector difference between these average points. I found that transforming the vector of a known speaker by adding this gender transformation vector could (in the clearest cases) generate speech as if the original speaker had swapped gender. You can listen to an example here: a test sentence is first generated in voice 39 (an Irish female), then in the voice of speaker 39 + female-male transformation. The result is a reasonably convincing Irish male. I also tried applying double the transformation: the result was a deeper, more extreme male voice, but one that still bears perceived characteristics of an Irish accent.

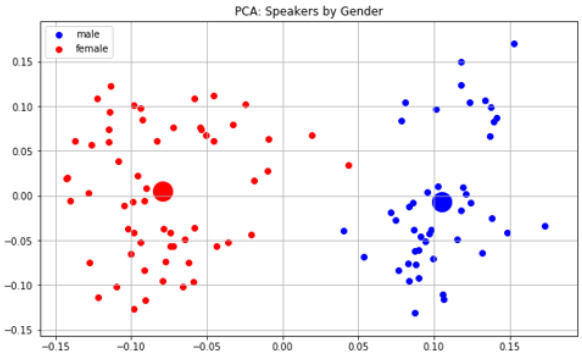


Figure 5.9: Gender transformation vector, defined as the difference between average female and average male embeddings.

I ran similar experiments for English → American **accent transformations**, again defining the transformation vector as the difference between the average embeddings of English and American speakers in the training set. The sample on the website, of speaker 34 transformed from English to American, is again quite convincing.

5.3 But... Generative Attributes are Entangled

Unfortunately, the transformation vectors we've found for gender and accent are not aligned to any of the dimensions of the embedding space, and have significant components across all features.

Nor are they orthogonal. Therefore, the generative attributes of the speech dataset are heavily entangled in this speaker representation.

We had to rely on the labels in VCTK to be able to identify the interpretability of the space and discover the transformation vectors - otherwise, this structure would have been very difficult to uncover. Also, we were only able to learn very crude averages. This limits what we can achieve with speaker vectors unless we can encourage the model to learn a disentangled representation.

Chapter 6

VoiceLoop with Utterance Embeddings

Although the speaker embeddings learned by VoiceLoop are impressive, two difficult questions quickly arise:

- **New Speakers**: The model has been trained on a set of known speakers. But what if we want to generate speech in the voice of a new, unseen speaker? Do we have to start over? Or is there a way to map the speaker to an embedding in the existing space?
- **Speaker Variability**: VoiceLoop has learned a single embedding vector for the average voice of each speaker. But in real life, we don't always say a sentence the same way: there is prosodic variation. How can we reflect the variation in a given speaker's voice?

Section 6.1 briefly reviews the literature on fitting voices for new speakers in neural TTS models.

Section 6.2 proposes an altered architecture, VoiceLoop-Utterance, which is based on utterance embeddings rather than speaker embeddings.

Section 6.3 investigates how this relates to the second point above: speaker variability.

6.1 Fitting Voices for a New Speaker

From the start of their research, the VoiceLoop researchers identified the problem of generating voices for speakers outside the training set as of key importance.

In the original paper[48], they were the first to demonstrate the possibility of fitting a new speaker into an existing, trained model. Their method requires a full set of examples of speech audios and text transcripts for the new speaker. The new speaker is added to the speaker LUT with a randomly initialised vector. Then, this vector is updated using the existing stochastic gradient descent method by training the model on the new speaker's data with all other parameters frozen.

In this way, the rest of the model remains unchanged while the speaker vector is gradually updated to get the best fit for the new speaker in the existing structure. 24 minutes of data per new speaker gave good results, which is much less than would be required to build a single-speaker model on that speaker's data alone.

However, there were a number of weaknesses with the approach:

- **Large Data Requirement**: Although 24 minutes is less than needed for a single-speaker model, it's still a very demanding requirement in practice.
- **Labelled Data**: The data for the new speaker must be carefully transcribed. Again, this is quite a demanding requirement.
- **Slow to Train**: The process is a slow and cumbersome way to fit a single 256-dimensional vector into an existing space - we have to run forward and backward passes through the whole model.
- **Lost Opportunity for the Model to Learn**: If we do have 24 minutes of high-quality transcribed data for a new speaker, it's a shame not to be able to use this to improve all the model parameters: instead, they're frozen during the updates.

Numerous researchers followed up with models that place a much less onerous data requirement on new speakers.

VoiceLoop's own version[39] learns the voice of a new speaker from a short sample without a transcription. Even a sample a few seconds long is sufficient to generate good-quality speech on which a speaker classifier network achieves top-1 classification accuracy of around 80%. The model architecture is altered by replacing the speaker embedding lookup table with an utterance embedding encoder, which gives one embedding per utterance rather than per speaker. I will explain how this works in the next section.

Similar approaches were reported to be successful by DeepVoice[2], Tacotron[22] and [31].

6.2 VoiceLoop with Utterance Embeddings

VoiceLoop uses **speaker embeddings**: one vector per speaker, that gives the best loss across all that speaker's training utterances.

An alternative approach is to use **utterance embeddings**: one vector per utterance, without relying on speaker labels at all.

The speaker lookup table is replaced with an encoder that maps a short speech sample (typically a few seconds in duration) into a vector in a 256-dimensional embedding space. This vector is then passed into the decoder in place of the original speaker vector. During training, the network learns to generate speech in a voice similar to the voice of the reference utterance. We therefore move from having a single vector per speaker, to having a separate vector for every utterance in the training set.

The obvious disadvantage of this approach is robustness: with only a short reference utterance to go on, it could be difficult for the network to extract a good enough representation. There is no opportunity to 'borrow strength' from other utterances from the same speaker.

However, the key advantage is that we now need only a single, short sample of speech with no text transcript to fit a voice for a new speaker.

6.2.1 Utterance Encoder Architecture

To encode a short sample of speech into an utterance embedding, I used a similar convolutional network architecture as in the speaker classification network, but with mean pooling rather than max pooling. This is very similar to the structure used in [39], and also [31][22][2].

- 5 2D convolutional layers, each using 3x3 filters, 32 output channels, batch normalisation and ReLU activations
- Mean pooling over time
- Linear projection to 256 dimensions followed by weight normalisation
- tanh activation and renormalisation

Figure 6.1 illustrates the changes to the overall architecture. A 256-dimensional embedding vector is produced from the WORLD feature representation of an utterance, and this is then passed to the decoder in place of the original speaker embedding.

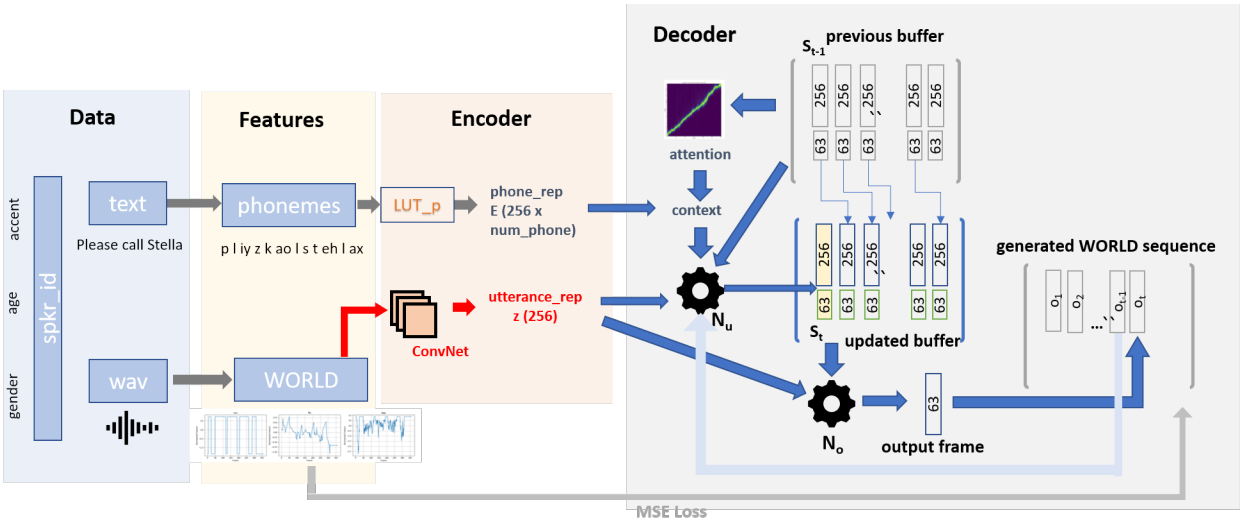


Figure 6.1: Architecture of VoiceLoop-Utterance

6.2.2 Training VoiceLoop + Utterance Embeddings

I trained the US-21 utterance embedding model according to the same protocol as the baseline. However, for the much larger All-107 model I compared two alternative training methods:

From Scratch As for the baseline, I trained the model in two phases with different noise values and sequence lengths. This look a long time to train (best part of a week), which makes the development cycle rather difficult.

From a Pre-Trained Model In this approach, I took the baseline VCTK-All-107 model as a starting point for training. I copied all the model parameters from the baseline (except speaker embeddings), randomly initialized the utterance embedding network, and then trained from then onwards. This should be a much quicker way to train the model. It may also get better results, by starting in a better place.

In [39], the authors added two additional terms to the loss function in addition to the MSE reconstruction loss:

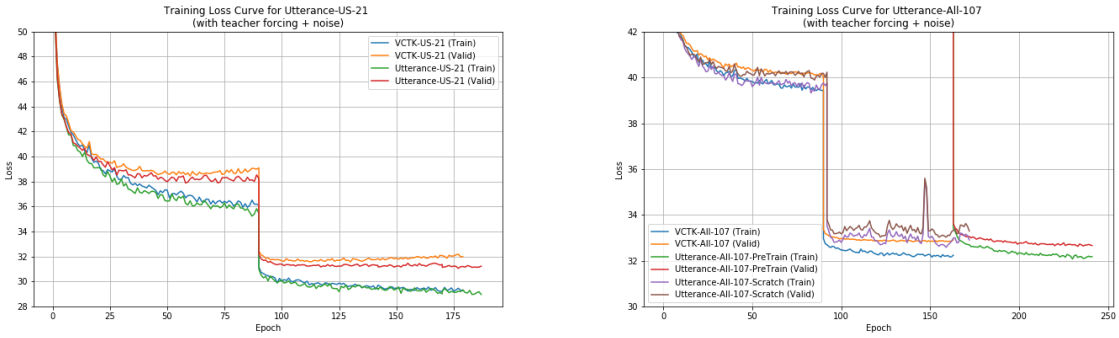
- **Contrastive Loss**: Take three samples, y_1 and y_2 from one speaker, and y_3 from another. The contrastive loss term forces the embeddings of y_1 and y_2 to be close to one another, and both to be further away from the embedding of y_3. That is, embeddings for utterances of a speaker should be close to each other, and far from the utterances of other speakers.
- **Speaker Constancy Loss**: Embed an utterance, generate speech using it, and then pass the output speech back through the embedding encoder. The resulting embedding should be close to the original embedding. This loss term penalises the model according to the distance between the embeddings.

In my version, to keep things simple I used only the original MSE loss. I suspect this is why the 'from scratch' version did not train as well as the 'pre-trained' version, which began from a better starting point.

6.2.3 Training Curves

Figure 6.2 compares the training curves for utterance embeddings on the US dataset with the baseline simulation using speaker embeddings. Utterance-US-21 achieves a better validation loss, with the best model achieving 31.1 versus 31.6 for speaker embeddings. It appears that allowing the model to encode specific details of how a speaker speaks in a particular utterance gives it additional information that allows a more faithful recreation of the audio. For example, consider a particular example in which the speaker talks with a higher pitch than normal. A sample generated using their single speaker embedding will reflect only their typical pitch. In contrast, a sample generated using an embedding encoded from utterance itself might be better able to reflect the higher pitch, and so achieve a lower reconstruction loss. Of course, this may come at the cost of poorer generalisation: using an utterance embedding constructed from one utterance to generate speech for a different piece of text might be less robust than using a single speaker embedding. From listening to many samples I suspect that this may be the case; however, I would need to run a systematic test using human listeners to be sure.

The utterance embedding models are generally a little worse than the baseline speaker embedding versions at reproducing voices, as show by the speaker recognition figures in table 6.1, which also includes figures for the VCTK-All-107 version. The utterance embedding model will generate



(a) VCTK-US-21

(b) VCTK-All-107

Figure 6.2: Training loss curves for utterance embeddings vs. speaker embeddings (baseline)

Sim	MSE Loss Training	MSE Loss Validation	Speaker Recognition Teacher Forcing	Speaker Recognition Evaluation
Utterance-US-21	29.2 (*29.7*)	31.1 (*31.6*)	99.4 (*99.9*)	97.1 (*95.0*)
Utterance-All-107-PreTrained	32.1 (*32.2*)	32.6 (*32.8*)	94.7 (*99.5*)	85.2 (*98.0*)
Utterance-All-107-Scratch	32.6 (*32.2*)	33.0 (*32.8*)	95.9 (*99.5*)	88.2 (*98.0*)

Table 6.1: Losses and speaker classifier accuracies for the utterance embedding models. Corresponding figures for the baseline speaker embedding models are given in brackets.

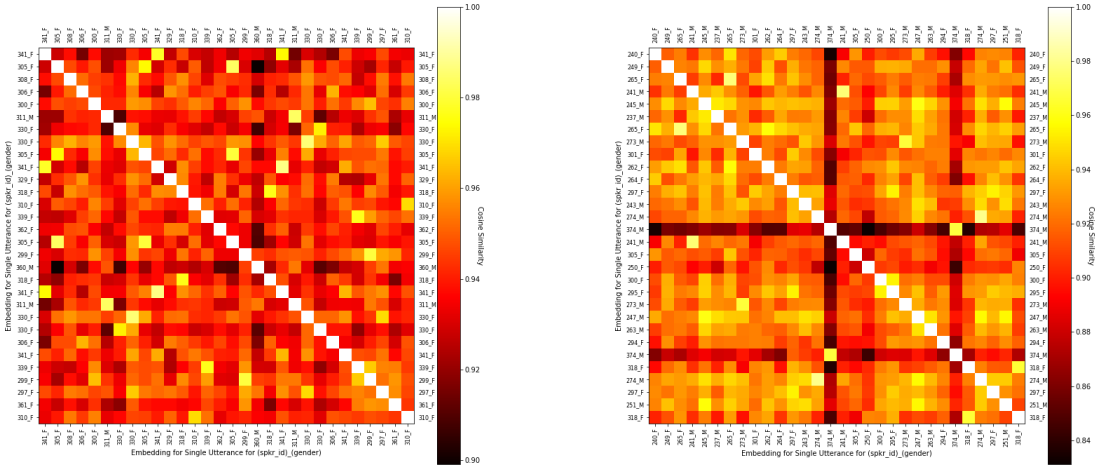
speech to match the reference sample, and if that particular sample is far from the average of the source speaker then the speaker classifier will get it wrong. In contrast, the speaker embedding model reliably generates speech in the voice of the source speaker, even if that particular utterance was not characteristic.

6.2.4 Testing on Unseen Speakers

Unfortunately, due to time pressures I didn't investigate how well my models could fit the voices of unseen speakers. I had planned to use the audio samples from the two speakers I had to drop from the VCTK dataset (one was missing transcripts, the other metadata), and evaluate on a speaker classifier build on all 109 speakers (which doesn't require transcripts or metadata, so is unaffected by the issues that caused me to drop the speakers.) However, as this wasn't the main focus of my project I had to move on. I would like to look at this in future work.

6.2.5 Looking at the Embeddings

Figure 6.3 shows the cosine similarities between utterance embeddings for 60 examples in the validation set. The first observation is that the average cosine similarity is high: in fact, the mean is 0.94 across the validation set. This suggests that the model has learned to embed the utterances in a small sector of the unit hypersphere (recall that the norms are constrained to be max unity) rather than uniformly across the space.



(a) VCTK-US-21 (b) VCTK-All-107

Figure 6.3: Cosine similarities between utterance embeddings for 60 examples in the validation set of Utterance-US-21. The label shows the speaker ID for the utterance, plus that speaker's gender. A speaker's utterances tend to be more similar to each other than to the utterances of other speakers.

We see that a speaker's utterances tend to be more similar to each other than to the utterances of other speakers.

The points in the heatmap with the highest cosine similarity (the yellow squares) are generally between two utterances of a given speaker. For example, the top row shows similarities between each sample and one particular utterance from speaker 341. The only two yellow squares in the top row correspond to other utterances from speaker 341: speaker 341 tends to sound similar across utterance, and different to other speakers[1].

Also, note that the points with the lowest similarity (black squares) are generally between male and female speakers.

Table 6.2 shows the closest 20 utterances (out of a set of 342) to one particular utterance for speaker 339. The closest five utterances are all from the same speaker. Speaker 339 has eight utterances in this set, and all appear in the top 12. Also note that I haven't picked an 'easy' example: in many other cases, all that speaker's utterances would have stacked at the top of the list.

I investigated the distribution of cosine similarities between pairs of utterances depending on whether the utterances were from the same speaker, or from different speakers. The same-speaker distribution in figure 6.4 (in blue) has a mode of 0.98, compared to 0.94 for the different-speaker

[1]although note there is a fourth utterance from speaker 341 in this set 5th from the right on the top row: this actually has a lower cosine similarity, closer to the overall average. There is something different about this utterance: either she spoke differently, or there is some other characteristic of the sentence that causes it to be separated out.

	cosine_sim	spkr_id	gender
0	1.000000	**339**	F
1	0.986227	**339**	F
2	0.985143	**339**	F
3	0.984908	**339**	F
4	0.979222	**339**	F
5	0.976928	**339**	F
6	0.966691	299	F
7	0.966179	300	F
8	0.965626	300	F
9	0.964847	299	F
10	0.963975	299	F
11	0.963739	**339**	F
12	0.962752	**339**	F
13	0.961560	300	F
14	0.961236	297	F
15	0.960338	299	F
16	0.960053	300	F
17	0.959985	299	F
18	0.959922	300	F
19	0.959374	333	F

Table 6.2: Closest 20 utterance embeddings in the dataset to an utterance from speaker 339

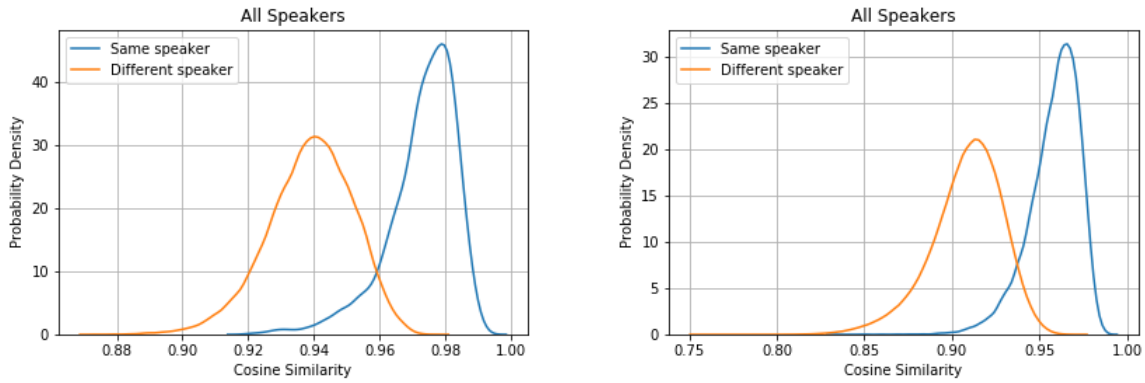
distribution (in orange). The distributions are not completely separated (i.e. some different-speaker pairs are closer than some same-speaker pairs), but there is strong evidence that same-speaker pairs cluster together reasonably well. I also produced these plots separately for each speaker, and found that some (e.g. speaker 360) have very clear separation, whereas for others (e.g. speaker 297) the distributions are much more overlapping.

Further confirmation that the model learns the utterance embeddings in an interpretable way can be found by clustering using t-SNE[2]. Figure 6.5 shows the utterance embeddings marked with the speaker's gender: there's clear separation between males and females in this space. The embeddings cluster into clear blobs.

Figures 6.6 and 6.7 shows the same plots but with the points marked with the ID of the speaker the utterance came from, rather than the gender. Each blob is dominated by the utterances of a single speaker, and although there are occasional discrepancies, there is little mixing between speakers.

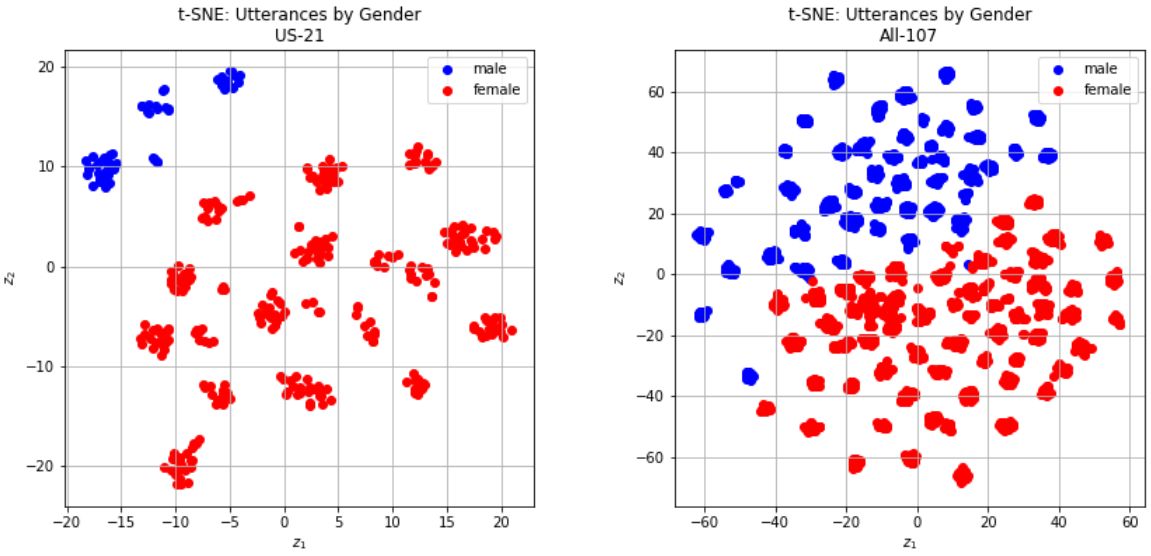
So we can conclude that the utterance embedding model learns a meaningful, interpretable embedding space in which utterances separate by gender and speaker identify.

[2]I ran the same analysis with PCA, but t-SNE gives clearer visualisations here.



(a) VCTK-US-21 (b) VCTK-All-107

Figure 6.4: Distributions of pairwise utterance embedding similarities in the US-21 validation set. Pairs of utterances from the same speaker (blue) are higher than pairs of utterances from different speakers (orange).



(a) VCTK-US-21 (b) VCTK-All-107

Figure 6.5: t-SNE visualisation of utterance embeddings in the validation set of VCTK-US-21, labelled with speaker gender.

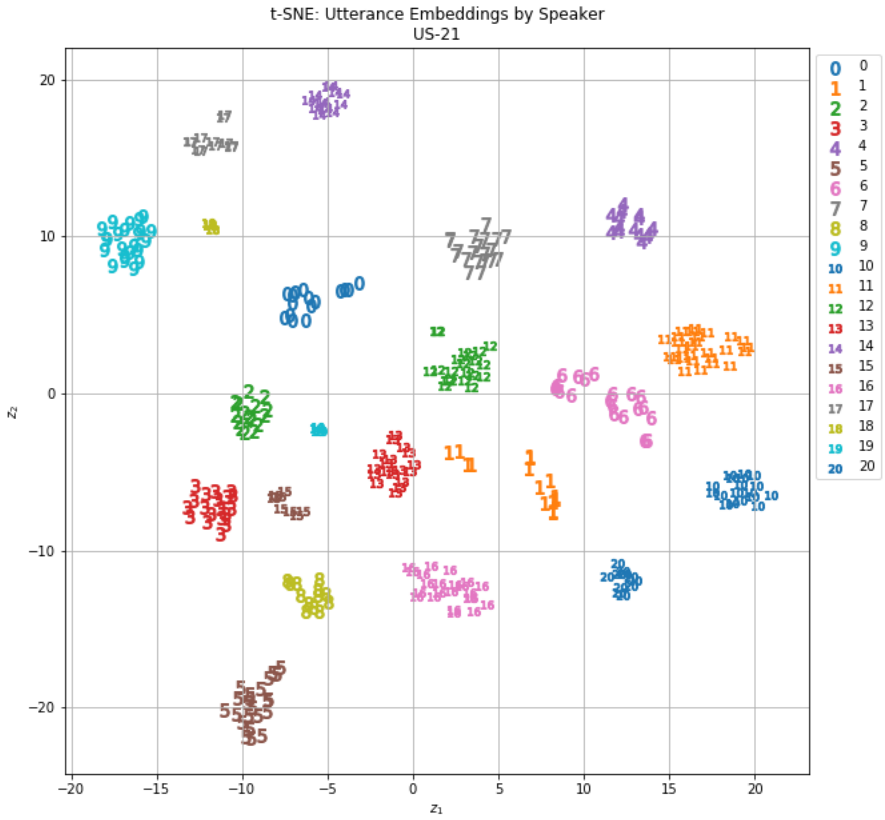


Figure 6.6: t-SNE visualisation of utterance embeddings in the validation set of VCTK-US-21, labelled with speaker ID.

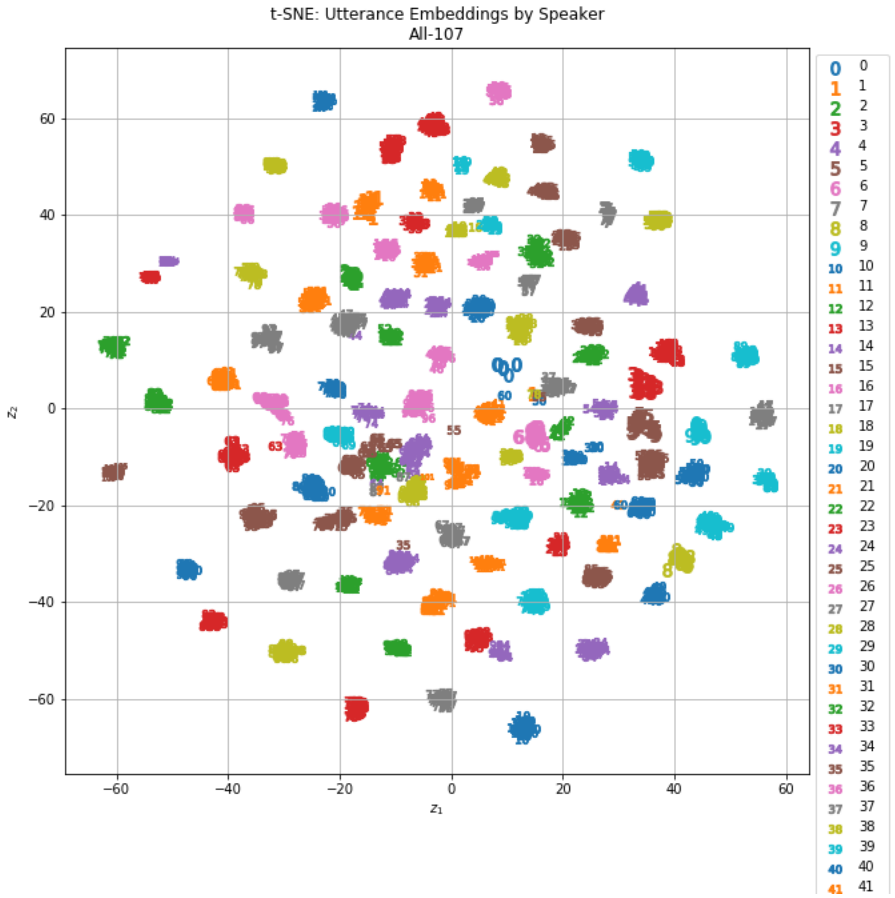


Figure 6.7: t-SNE visualisation of utterance embeddings in the validation set of VCTK-All-107, labelled with speaker ID.

6.3 Introducing Variation by Sampling from a Speaker's Utterances

As we discussed earlier, the baseline VoiceLoop model based on speaker embeddings is completely deterministic, and will always generate the same output features for a given $\{\text{text}, \text{speaker}\}$ input. Using utterance embeddings gives us the possibility of introducing a degree of variation in the generated speech for a given speaker, by generating with embeddings selected from within the region of space spanned by that speakers training utterances.

When using the model in **inference** mode for new strings of text, we pick a reference utterance that we want the generated speech to sound like. The encoder maps this utterance to an embedding, which is then used in the decoder. Note that in the standard VoiceLoop-Speaker model, during inference you pick a *speaker* that you want the speech to sound like. In VoiceLoop-Utterance, you pick a *reference utterance* that you want the speech to sound like. The instruction is: "make the speech sound like *this*", rather than "make the speech sound like e.g. speaker 66's average voice".

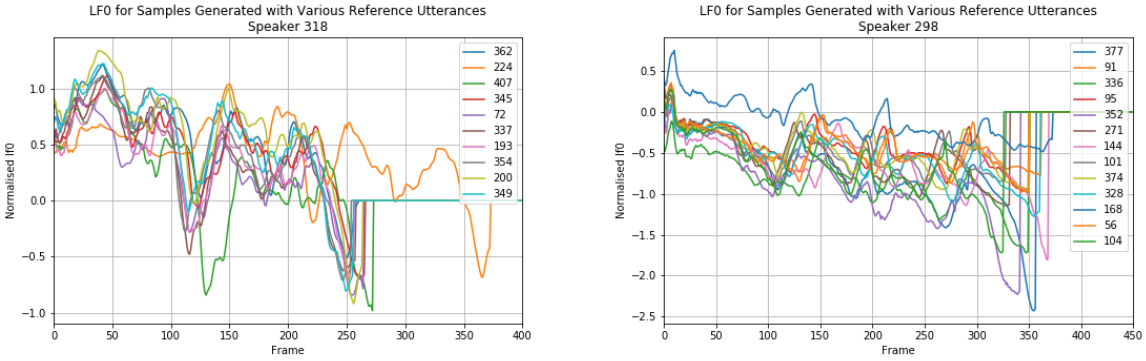
As a test, I generated a set of samples for speaker 318 in the US-21 model using the text:

We have great team spirit.

but passing in different reference utterances from that speaker's set to calculate each embedding. You can listen to the samples here.

I ran a similar test for speaker 298 in the All-107 model: you can listen to the samples here.

The generated samples all sound very similar, and in the voice of the selected speaker, but it's possible to hear minor variations. Figure 6.8 shows the different LF0 feature sequences for ten examples. The sample generated using the embedding of sentence 224 is much longer - if you listen to the sample, you'll hear that the speech has a longer, breathy pause at the start. The sample generated with sentence 407 has more stress on 'have'. Others have minor differences in speech rate, pitch and stress.



(a) VCTK-US-21

(b) VCTK-All-107

Figure 6.8: LF0 trajectories

6.3.1 Can we Find Any Structure Inside a Speaker's Set of Utterances?

So we have seen that the set of embeddings for all the utterances of a selected speaker, can generate variation in the output speech. Is there any interesting structure inside this set, perhaps based on the type of speech act (question, statement, command), or differences in the way the speaker vocalises in each case (pitch, speed, intonation changes)? We might hope to find that the questions cluster in one part of the space, for example, or that utterances in which the speaker talks more quickly than they normally do can be separated from the utterances in which they speak more slowly. Any patterns such as these would make the space even more interpretable, and could give us more control over the speech generated by the model: if we wanted to make speech sound a little more urgent, perhaps we could shift the embedding slightly in one direction.

I looked for patterns across a range of weakly-labelled attributes such as:

- Whether the utterance was a question
- Sentence length (long vs. short)
- Average pitch high/medium/low (mean lf0)
- Pitch range (standard deviation of lf0)

Figure 6.9 shows the training-set utterances of speaker 315, labelled by whether or not the utterance was a question. I identified questions by the presence of a question mark in the transcript, which is only a rough guide. Reading a question aloud from a newspaper isn't the same as asking a genuine question to another person. Also, some questions may have been rhetorical. In any case, the utterance embeddings showed no clear pattern by whether or not they were questions.

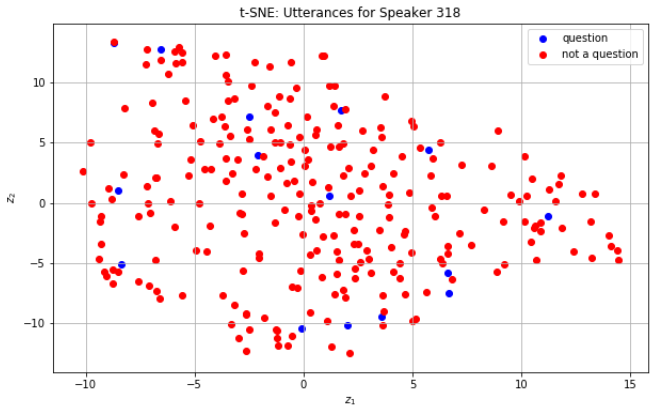


Figure 6.9: t-SNE visualisation of utterance embeddings for speaker 318, with questions marked as blue.

Unfortunately I didn't find any clear patterns to exploit, which was disappointing. I think this is partly because the VCTK dataset has little prosodic variation. The speakers are not professional voice actors, and were not asked to speak dramatically or in an exaggerated fashion. They were

reading out short sentences from a newspaper, which were generally simple statements with limited emotional value. As each sentence was largely isolated, there was no opportunity to build up prosodic range over an extended piece. In future research I would see if any patterns can be identified in a more dramatic dataset, such as one of the audiobook datasets from the Blizzard Challenge read by professional voice actors. The results achieved by the Tacotron prosody papers[57][44] were based on audiobook data.

6.4 Summary

In this chapter, I motivated the advantages of switching from average speaker embeddings to utterance embeddings. I proposed changes to the VoiceLoop architecture, in line with others in the literature, and described the results of the VoiceLoop-Utterance model. I then showed that this approach allows us to recover some degree of speaker variability, although unfortunately I was not able to find any interpretability in the structure of a single speaker's embedding distribution.

Part III

Learning Disentangled Speaker Representations

This part of my report contains my original research contributions. I investigate whether we can modify the VoiceLoop architecture to encourage the model to learn a disentangled speaker representation.

In Chapter 7, I use a Fader Networks[30]-style supervised, adversarial approach that uses gender labels as part of the training process to learn a speaker embedding space in which gender is disentangled.

In Chapter 8, I use a β-VAE unsupervised approach that encourages the model to learn a disentangled speaker representation purely from the data.

Chapter 7

Disentangling Using Labels: Fader Networks

In this chapter I explain the technique used by the Fader Networks paper[30] to disentangle **labelled** attributes of image data, and then investigate whether a similar technique can be used in the context of the speaker-embedding VoiceLoop model to disentangle the speaker attributes for which VCTK provides labels (gender, accent, age).

In section 7.1 I discuss the Fader Networks paper in detail, including examples of the results I was able to recreate using the codebase the authors have kindly published on GitHub.

In section 7.2 I explain the modifications I made to the VoiceLoop architecture to build and train a Fader-style model on speech data.

In section 7.3 I present the results of my Gender Fader model on the All-107 data.

In section 7.4, I conclude this chapter by discussing weaknesses in my model, and the further work I would do in future.

7.1 Fader Networks Paper

7.1.1 Motivation: Controlling the Attributes of Generated Images

Consider the canonical example of generating images of faces. A model that could generate a realistic face at random would be impressive, but not especially useful: we would not have any control over the attributes of the face. If we need an image of, say, an older, female face with long hair and glasses that doesn't correspond to any actual person, we would have to sample a large number of images and filter them manually. And if the best image we could find was good but not perfect, we would have no easy way to 'tweak' the image to requirements.

A much more useful model would use an internal representation that allows direct *control* over the

perceived attributes of the generated image. There could be independent features in the latent space that correspond to attributes such as age, gender, hair length, wearing glasses and so on. For a particular image, the values of these features would directly correspond to the attributes we can perceive in the image; and by changing the feature values and generating a new sample, we can effect a clear change in the image. By changing the gender feature, for example, we could transform an image of a man into an image of what they might look like as a woman, holding all other attributes fixed. Or by varying the age feature, we could choose how young or old we want to face to appear. A generative model like this could be used either to encode a known test image and generate an output image with controlled changes to its attributes; or to generate a random image where some of the attributes have been pre-selected, and where they can be tweaked if needed.

The Fader Networks paper[30] presents a technique for learning such a model from a training set of images labelled with attribute values, with examples of the form:

$$\{\text{image}, \text{age}, \text{glasses/no-glasses}, \text{male/female}\}$$

Note that the dataset doesn't include explicit attribute *transformations*: there aren't versions of a person with and without glasses, for instance, from which the model can learn an explict transformation. The dataset examples are *not* of the form:

$$\{\text{image}, \text{transformation\_name}, \text{transformed\_image}\}$$

Instead, the model learns the transformation implicitly from the set of images and attributes. During inference, the user can use the latent factors to 'fade' attributes in and out of the generated image as if using the slides on an audio mixing desk.

7.1.2 The Approach

The Fader Network approach starts with a standard **autoencoder architecture**. A deterministic encoder maps an example input image x with known attributes y to a latent representation z. A deterministic decoder generates an image $\hat{x}$ given (z, y), and is trained to minimise the reconstruction error between $\hat{x}$ and x.

The key ingredient of the Fader approach is that the **latent space is constrained to be invariant to the attributes**. The idea is that the latent representation z should not contain any information about the attributes y of the particular example x. So if our input example is an image of a man with glasses, it should not be possible to determine from the representation whether the original image was of a man or a woman, or whether or not they were wearing glasses. Alternatively put, imagine we have two images of a particular person, one in which they're wearing glasses and one in which they aren't. Both images should map to the **same** latent representation. When it comes to decoding this single, common representation, we can set the glasses feature to '1' to reconstruct the first image, and to '0' to reconstruct the second.

This attribute-invariance is enforced using a form of **adversarial training**. A classifier is first

trained on the latent space to predict the attributes y for a latent representation z. So, in our example above, the classifier would take in the latent representation and try to predict whether the original image had glasses or not. This would typically be a straightforward task in an unconstrained autoencoder. However, the Fader approach then adds a discriminator loss to the training objective of the autoencoder, that penalises the network if it's possible to predict the attributes y from the representation z. During training, the encoder and decoder learn to work together to remove all traces of the attributes from z, such that the discriminator can't guess y from z, but the decoder can still generate good reconstructions of x given (z, y). The network learns to encode 'attribute-neutral' representations, such that all information about the desired attributes is fed into the decoder directly, with no residuals leaking in through z.

7.1.3 The Architecture

The model is an: "encoder-decoder architecture with domain-adversarial training on the latent space."[30]. Full details of the architecture and training technique (plus useful diagrams) can be found in [30]; the key points are :

Notation Consider a training set $\mathcal{D} = \{(x^1, y^1), ..., (x^m, y^m)\}$, of m (image, attribute) examples, where $x^i \in \mathcal{X}$, the domain of images, and $y^i \in \mathcal{Y}$, the domain of attributes associated with these images, e.g. *male/female*, *glasses/no glasses*. If there are n binary attributes, $\mathcal{Y} = \{0, 1\}^n$. For simplicity we set the problem up with binary attributes, although the approach can be extended. Note also that although we *train* on binary values, during inference the attributes are treated as continuous variables.

Learning Problem Learn a model that, for any example (x, y), and any arbitrary set of attributes y', generates an output x' that can be identified as a version of x but with attributes y'.

Encoder-Decoder x

- Encoder $E_{\theta_{enc}} : \mathcal{X} \to \mathbb{R}^N$: The encoder is a convolutional neural network mapping the image x to its latent representation, $E_{\theta_{enc}}(x)$. The network has seven layers of 4x4 kernels, with batch norm and leaky ReLU activations; the network parameters are represented by θ_{enc}. The latent space has N dimensions.
- Decoder $D_{\theta_{dec}} : (\mathbb{R}^N, \mathcal{Y}) \to \mathcal{X}$: The decoder is a deconvolutional neural network that generates a variant image x' for an arbitrary set of attributes y'. The decoder architecture is essentially symmetric to the encoder, but with the structure augmented to factor in the attribute information. The attributes y' are first expressed as one-hot vectors (e.g. $[1, 0]$ for glasses, $[0, 1]$ for no glasses), concatenated, then appended as constant input channels to every stage of the decoder. The network parameters are represented by θ_{dec}.

Discriminator Network A separate discriminator network is trained to identify the ground-truth attributes y of training example (x, y), given the latent representation $E_{\theta_{enc}}(x)$ of the input image. The network is formed of a single transposed convolutional layer followed by two

fully-connected layers, with dropout. For a given state of the discriminator parameters θ_{dis}, the network outputs the probabilities of the attribute vector y given a latent representation: $P_{\theta_{dis}}(y|E_{\theta_{enc}}(x))$. The discriminator is designed to predict the ground-truth attributes y, so we train it with a cross-entropy loss for a fixed state of the encoder θ_{enc}:

$$\mathcal{L}_{dis}(\theta_{dis}|\theta_{enc}) = -\frac{1}{m}\sum_{(x,y)\in\mathcal{D}} \log P_{\theta_{dis}}\big(y|E_{\theta_{enc}}(x)\big)$$

where, using k to represent the $k-th$ attribute out of n, the log joint probability of a particular attribute state is given by:

$$\log P_{\theta_{dis}}(y|E_{\theta_{enc}}(x)) = \sum_{k=1}^{n} \log P_{\theta_{dis},k}\big(y_k|E_{\theta_{enc}}(x)\big)$$

Loss Function The objective of the encoder-decoder network is to encode x into $E_{\theta_{enc}}(x)$ in such a way that the decoder can successfully reconstruct x given $(E_{\theta_{enc}}, y)$, while preventing the discriminator network from being able to identify y given $E_{\theta_{enc}}(x)$. This gives rise to two components in the loss function:

- **Reconstruction Loss**: The loss associated with the auto-encoding aspect of the network is measured using the usual mean square error (MSE), which captures how closely the network can reconstruct the original input image when the decoder is supplied with the ground-truth attributes y:

$$\mathcal{L}_{rec}(\theta_{enc}, \theta_{dec}) = \frac{1}{m}\sum_{(x,y)\in(D)} \|D_{\theta_{dec}}(E_{\theta_{enc}}(x), y) - x\|_2^2$$

- **Adversarial Loss**: The discriminator makes an error for attribute k when it predicts value $1 - y_k$. The paper considers the objective to be to maximise the probability of the discriminator being wrong, i.e. for a given fixed state θ_{dis} of the discriminator, to minimize:

$$\mathcal{L}_{adv}(\theta_{enc}|\theta_{dis}) = -\log P_{\theta_{dis}}(1 - y|E_{\theta_{enc}}(x))$$

The complete loss combines these with a parameter $\lambda > 0$ that tunes the relative importance of reconstruction quality and integrity of the attribute-invariant representation:

$$\mathcal{L}(\theta_{enc}, \theta_{dec}|\theta_{dis}) = \frac{1}{m}\sum_{(x,y)\in(D)} \|D_{\theta_{dec}}(E_{\theta_{enc}}(x), y) - x\|_2^2 - \lambda \log P_{\theta_{dis}}(1 - y|E_{\theta_{enc}}(x))$$

Training Each round of training progresses in two stages:

- Train discriminator, with encoder parameters θ_{enc} fixed
- Train encoder-decoder, with discriminator parameters θ_{dis} fixed

This sets up a two-player adversarial game, in which first the discriminator takes a turn at improving itself to classify the embeddings, then the encoder-decoder has a change to update

its parameters to flummox the discriminator, and so on.

As in many adversarial architectures, finding an effective training regime in which the training of the two components is balanced, with both learning while not dominating the other, is a delicate matter. The paper uses **discriminator cost scheduling**: λ is initially set to zero, such that the model is trained as a standard autoencoder, and the discriminator loss is introduced gently during training. This allows the model to first learn to reconstruct without worrying about the latent space, and then to push the latent representations towards neutrality later on.

Model Selection The aim is to find a final model that, in addition to good reconstruction loss and an attribute-invariant latent space, performs well at attribute-flipping when generating new images. Therefore, at the end of every epoch, a set of images is generated by swapping attributes of all examples in the validation set. These images are passed through a pre-trained classifier to evaluate whether the images have the correct attributes. The final model is selected by human evaluation, using these results as a guide.

7.1.4 Results

The paper's main results are on CelebA, a dataset of 200,000 facial images labelled with 40 attributes. All evaluations are based on subjective judgement of whether the generated images are natural (are they plausible images of faces?), and accurate (do they reflect the selected attributes for the target person?). Initial evaluation is based on studying sets of images produced by sweeping the latent features; this is then formalised by using impartial raters on Mechanical Turk to evaluate naturalness and accuracy against baselines of ground truth and IcGAN under a blind-testing protocol. This is a very similar setup to the subjective evaluations used to evaluate speech synthesis systems using MoS scores.

The authors report strong results. The Fader Network reconstructs input images to a very high quality, achieving naturalness and accuracy score far in excess of the IcGAN benchmark and not far behind the ground truth evaluations. When used to generate new images with altered attributes, Fader Networks achieves impressive results, with more than half of generated images being judged as natural by the human raters, and around three-quarters being judged to successfully represent the selected attributes.

Using the code the authors released on GitHub, I was able to build a version of the model and generate samples by sweeping the latent factors for a given example: see figure 7.1 for samples using the 'glasses' attribute. For more, and higher-quality samples, see [30] or GitHub.

7.1.5 Why This Might Work in VoiceLoop

There are a number of parallels between the problem in the Fader Networks paper and my problem in this project.

In both cases, we are working with an encoder-decoder model, which uses a MSE reconstruction

Figure 7.1: Sample images generated using the Fader Networks code. The first column shows the original images. The second column shows the reconstructed images using glasses=False. The images in the remaining columns were generated by sweeping the 'glasses' variable: the images in the right hand column are fairly convincing versions of the original subject wearing glasses.

loss to train the model but for which we are really interested in its ability to generate unseen samples (images: altered attributes; speech: different text, or altered speaker attributes).

In both cases, we have labels for known attributes that can be perceived in the outputs, but do not have any explicit examples of attribute transformations (i.e. no pairs of examples of a person with and without glasses). Many of the attributes can be labelled as binary, although it makes sense for them to be treated as continuous during inference to allow a 'slider' approach to tweaking the generated output.

In both cases, we would like to be able to control the attributes in the generated attribute.

In both cases, subjective evaluation by humans is the ultimate test of success. Although we can build speaker classification models or models to classify whether an image has glasses in it, the ultimate tests of accuracy and naturalness are the eye (or ear.)

With these considerations in mind, I move on to describing how I altered the VoiceLoop architecture to include ideas from the Fader Networks paper.

7.2 Applying the Fader Approach to VoiceLoop

I had to make a number of decisions early on in my project about how to adapt VoiceLoop to accommodate the Fader approach. In particular, I focused my research in the following ways:

- **Speaker Embeddings**: All work in this section applies the Fader approach to the baseline *speaker embedding* version of VoiceLoop, rather than the utterance embedding version. This was partly accidental, in that I began the Fader work before the utterance embedding work. However, one can also argue that the speaker embedding approach should learn a more robust representation of a speaker based on a wide range of their utterances, which may give a stronger base from which to modify attributes.
- **All-107 Dataset**: The US-21 dataset has too few speakers, and in particular too few males, to be practical for the Fader approach on speaker embeddings. Unfortunately, using All-107 made the development cycle much slower and held back the amount of research I could complete.
- **Gender Attribute Only**: Although the Fader approach can handle multiple attributes, and the attributes can be multinomial as well as binomial, in this work I considered only the single binary attribute of gender. This was purely for practical reasons: I had intended to work on accent as well, but there wasn't time to complete this work, in part because of the slow development cycle mentioned above.

Figure 7.2 illustrates the architectural changes I made. In section 7.2.1 I explain the changes to the speaker embedding space. In section 7.2.2 I discuss the discriminator network. In section 7.2.3, I explain the training procedure.

7.2.1 Imposing Gender Variables on the Embedding Space

I initially investigated two options for adapting the speaker embeddings to incorporate independent representations for gender and gender-neutral speaker identity:

(a) **Additive**: In addition to the existing speaker embedding lookup-table (LUT_S), introduce a *gender LUT* that learns separate embeddings in the 256-dimensional speaker space for male and female attributes. The network is trained such that the impact of each speaker's gender is accounted for by the gender embedding, forcing the speaker's *individual* embedding to be gender-neutral.

During training, the 256-dimensional embedding z^i passed into the decoder for speaker i is formed as the vector sum of the relevant gender embedding g^j (where speaker i has gender j) and the speaker's individual gender-neutral embedding s^i:

$$\underset{256\times1}{z^i} = \underset{256\times1}{\begin{bmatrix} g_1^i \\ g_2^i \\ \vdots \\ g_{256}^i \end{bmatrix}} + \underset{256\times1}{\begin{bmatrix} s_1^i \\ s_2^i \\ \vdots \\ s_{256}^i \end{bmatrix}}$$

When generating new samples for a given speaker, we can choose to produce their voice is either male or female versions by adding in the required gender embedding.

I used this approach in the early stages of research: it works, and in practice it learns something like the gender transformation vector we identified in section 5.2.3. However, the approach isn't ideal. We haven't really disentangled the latent factors, as gender is still smeared across all 256-dimensions. but just learned the entangled representation for male and female. In addition, as the gender LUT is binary we can't directly apply a continuous modification to the gender variable, which means that we can't easily 'fader' the gender attribute.

For the rest of this report, I will be using the following method:

(b) **Concatenative**: Carve out two of the 256 dimensions and reserve them for one-hot gender factors, i.e. a binary male factor and a binary female factor. So the gender of a male speaker is represented by [1, 0] and a female is represented by [0, 1]. We then force the model to learn a gender-neutral speaker representation across the remaining 254-dimensions.

During training, the 256-dimensional embedding z^i passed into the decoder for speaker i is formed as the concatenation of the speaker's individual gender-neutral embedding s^i and the one-hot vector that represents the speaker's gender. If speaker i is female:

$$\underset{256\times 1}{z^i} = \begin{bmatrix} \begin{bmatrix} s_1^i \\ s_2^i \\ \vdots \\ s_{254}^i \end{bmatrix} \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

When generating new samples for a given speaker, we can choose to produce their voice is either male or female versions by concatenating $[1, 0]$ or $[0, 1]$.

In contrast to the additive approach, the concatenative method imposes structure on the embedding space by forcing it to assign gender to individual factors. It also allows us to generate speech with values of the gender vector unseen in training, including linear mixtures such as $[0.5, 0.5]$ (halfway between male and female), and extreme values such as $[2, 0]$ (extreme male voice.)

Of course, to force the model to learn gender-neutral speaker embeddings we need to train adversarially against a gender discriminator network that tries to infer the gender of the source speaker from their gender-neutral embedding vector.

7.2.2 The Discriminator Network

The main challenge in designing the discriminator is that we have only 107 speakers, but 254 dimensions in the speaker space. This brings two problems:

- **Discriminator Complexity**: It obviously won't be difficult to find a powerful non-linear classifier that can learn to separate speakers perfectly. On the other hand, if we use a

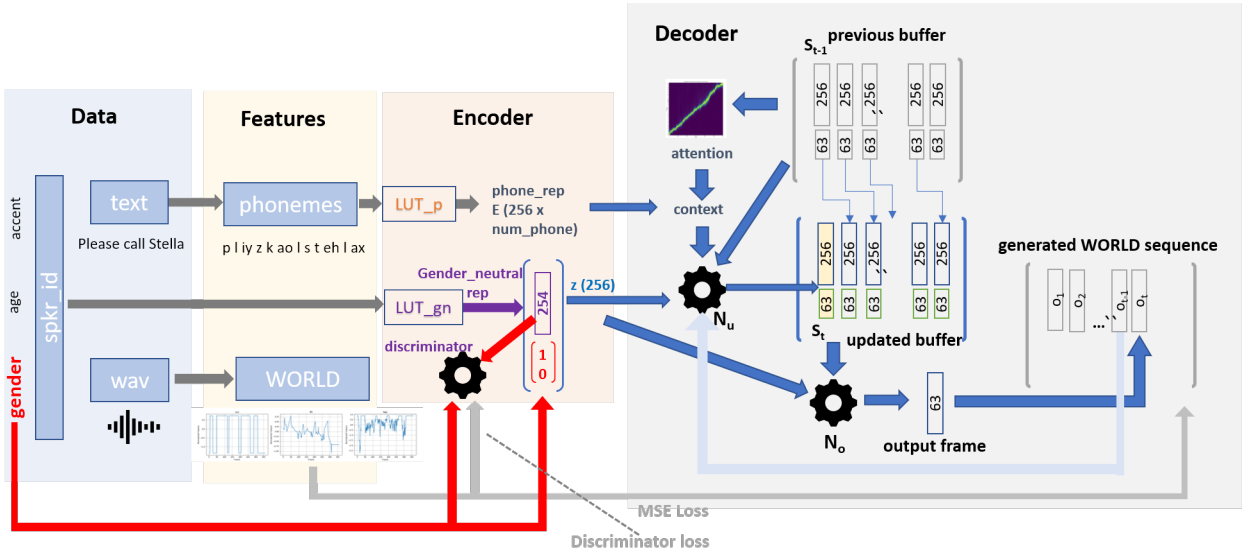


Figure 7.2: Architecture of VoiceLoop with a concatenative gender fader

simple linear classifier then it would be straightforward for the encoder to learn to 'hide' gender information in the embeddings, such that it can recover gender information while the discriminator can't. As is often the case in adversarial architectures such as GANs, choosing a discriminator architecture of appropriate complexity is a difficult art, and has to be done through experimentation.

- **Training/Validation Test Split**: Ideally, I would split the speakers into disjoint training and validation sets, use the embeddings from the training set to train the discriminator, and then evaluate it on the validation set embeddings to yield the adversarial loss term to add into the main network's loss function. This would give a measure of how well the discriminator generalises to unseen speakers, rather than just measuring how well it can classify a training set, on which a complex discriminator could heavily overfit and reduce in-sample error to zero. However, with so few speakers we would either have to make the training too small to learn with, or the validation too small to give a useful measure of generalisation. Perhaps I could have used leave-one-speaker-out cross-validation and taken a model average, but this would have added a lot of complexity to the training (if indeed it is possible.)

Given these points, after some experimentation I decided to keep the discriminator complexity low, and used a simple two-layer fully-connected neural network. The first layer has input size 254 (for the concatenative method's 254-dimensional gender-neutral speaker representation), 32 nodes and ReLU activations. The second layer has 16 nodes with ReLU activation. Finally, there is a linear projection to an output size of two, from which a softmax gives the two class probabilities. The hope was that this network would be complex enough to prevent the encoder from representing gender in any straightforward way, while not being so complex that it could always find a way to separate the speakers in the set. So we're not really testing whether the embeddings are 'absolutely' gender-neutral, but whether they're gender-neutral given the resolving power of a relatively simple non-linear classifier.

The discriminator's loss is computed in much the same way as for the original Fader Networks on images, using a cross-entropy loss for a fixed state of the encoder θ_{enc}:

$$\mathcal{L}_{dis}(\theta_{dis}|\theta_{enc}) = -\frac{1}{m}\sum_{(x,y)\in\mathcal{D}} \log P_{\theta_{dis}}\big(y|E_{\theta_{enc}}(x)\big)$$

The discriminator is trained using the Adam optimizer with a learning rate of 0.001.

7.2.3 Training the VoiceLoop-Fader

As with the original Fader Networks, the discriminator and encoder-decoder are trained together in an adversarial game. In each round of training, first the discriminator parameters are updated while holding the encoder-decoder fixed, and then the encoder-decoder parameters are updated while holding the discriminator fixed.

Loss Function For training the VoiceLoop-Fader encoder-decoder we use a loss function that starts with the standard VoiceLoop MSE loss (reconstruction quality) and adds in an adversarial term that penalises the loss if the discriminator can detect gender in the speaker embeddings (i.e. whether the speaker space is disentangled):

$$\mathcal{L}(\theta_{enc},\theta_{dec}|\theta_{dis}) = \frac{1}{m}\sum_{(x,y)\in(D)} \|D_{\theta_{dec}}(E_{\theta_{enc}}(x),y) - x\|_2^2 - \lambda \log P_{\theta_{dis}}(1-y|E_{\theta_{enc}}(x))$$

To measure the quality of disentanglement, we are using the *cross-entropy* between the discriminator's output gender probability given the speaker embedding, and the *incorrect* gender label $(1-y)$. That is, we consider the discriminator to have made a mistake when it predicts $1-y$ for the gender. This doesn't seem quite right, in that it would give the lowest loss for a discriminator that always make a mistake, which is only a sign change away from always being correct - effectively, the discriminator would be perfect up to a sign change.

A better objective would be for the discriminator to be perfectly *uncertain*, rather than *wrong*, as this better captures the intuition that the speaker embeddings should provide no basis from which to determine gender, as opposed to actively fooling the discriminator. This would be captured by the *entropy* of the discriminator distribution rather than the cross-entropy between the predicted probability and the incorrect class:

$$\mathcal{L}(\theta_{enc},\theta_{dec}|\theta_{dis}) = -\frac{1}{m}\sum_{(x,y)\in(D)} P_{\theta_{dis}}(y|E_{\theta_{enc}}(x)) \log P_{\theta_{dis}}(y|E_{\theta_{enc}}(x)) + (1-P_{\theta_{dis}}(y|E_{\theta_{enc}}(x))) \log(1-P_{\theta_{dis}}(y|E_{\theta_{enc}}(x)))$$

I experimented with using entropy rather than cross-entropy in the loss, but found it much harder to train the network this way. In practice, I found that using the cross-entropy loss doesn't lead to the discriminator learning to be maximally wrong - presumably, if it could do this it would simply learn to flip the sign by itself. And training with cross-entropy did result in the discriminator

being forced into uncertainty, with accuracy of around 50% and entropy reaching the maximum of $-\log \frac{1}{2} = 0.693$. Therefore I am satisfied that using cross-entropy is sensible, as per the original paper[30].

Training It took some experimentation to find a harmonious balance between the encoder-decoder and the discriminator:

Discriminator Training On each 'round' of the training, for how many epochs should the discriminator be trained? Recall that we only have 107 speakers, so one epoch of training represent a relatively small opportunity for the network parameters to update. If the discriminator isn't trained enough on each round, then the encoder-decoder will easily be able to beat it; however, if it is trained too much, it may overtrain and learn to discriminate regardless of how well the gender is disentangled (recall that we have to evaluate performance on the training set.) I found that training for 50 epochs in each round gave good results.

λ in Loss What is an appropriate value of λ, which dictates the balance between reconstruction and adversarial losses? Too high a value will sacrifice reconstruction quality and lead to lower-quality speech samples. Too low, and the speaker embeddings might remain entangled. I found that a value of 0.1 worked well.

λ Schedule In practice, I found it beneficial to increase the value of λ according to a schedule, allowing the network initially to learn to reconstruct examples without the disentanglement constraint, and then gradually increasing the value over training epochs to encourage disentanglement.

Pre-Trained vs. From Scratch Training the model from scratch was difficult, and it took me multiple iterations before I found hyperparameters that led to stable results. I therefore also tried to train the model starting from a pre-trained baseline model, with the idea being that the pre-trained model would already have learned to reconstruct, and could now be encouraged to disentangle. This is not too different to using a λ schedule that uses $\lambda = 0$ for an initial set of epochs.

I used the VCTK-All-107 speaker embedding VoiceLoop model from chapter 4 as the baseline. I initialized each speaker's 254-dimensional gender-neutral embedding by copying the first 254 dimensions from the the 256-dimensional representation in the baseline model. All other model parameters were copied directly from the baseline.

7.3 Fader Network Results

In section 7.3.1 I show that the VoiceLoop-Fader model trained successfully, as measured by the loss curves. In section 7.3.2 I show that the enforced gender factors in the speaker representation give us control over the perceived gender of the synthesized speech, and that the results of swapping and fading gender are convincing. In section 7.3.3 I investigate whether the speaker embedding is really disentangled: is there really no trace of gender in the gender-neutral part of the speaker representation? In section 7.4 I conclude with some observations on the success of this part of the project.

7.3.1 Training Curves

Although I was able to train VoiceLoop-Fader both from scratch and from the pre-trained baseline, the pre-trained version gave slightly better results. It also eases the comparison to the baseline VCTK-All-107. Therefore I will present only the results of the pre-trained version here.

Figure 7.3a shows the loss curves of the baseline model (blue/orange lines) from the second phase of training (epoch 90 to epoch 164), followed by the loss curves for the VoiceLoop-Fader (green/red) trained from epoch 164 onwards. The VoiceLoop-Fader loss is high initially because the adversarial loss punishes the heavily entangled speaker representation (plus the integrity of the existing speaker representation was harmed when I copied over only 254 of the original 256 dimensions and replaced the final two with the enforced gender variables.) However, after around only 30 epochs the VoiceLoop-Fader validation loss had reached the value of the best baseline model. After this the training loss continued to fall (noisily), but validation loss remained around the same level.

Figure 7.3b shows the evolution of speaker recognition accuracy for samples generated from the baseline (blue) and VoiceLoop-Fader (orange) models. VoiceLoop-Fader's samples achieve a marginally lower accuracy than the baseline, but it doesn't appear to be too significant given the variation in the plots.

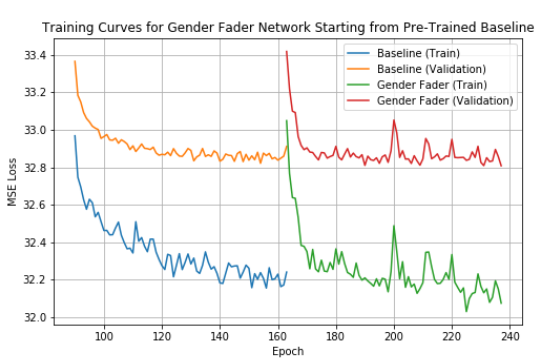
Figure 7.4 shows how the discriminator loss terms evolve during training[1]. The adversarial cross-entropy term quickly falls towards random levels, although the spikes such as those around epoch 200 and 220 illustrate that the training is not stable and occasionally the network parameters jump away from the optimum values. The discriminator entropy is generally around the random level (0.69), indicating maximum uncertainty as discussed in 7.2.3. The discriminator accuracy also hits random levels of around 50% (VCTK-107 is not completely gender-balanced, though.)

So, the VoiceLoop-Fader model appears to train well, achieving a similar reconstruction loss to the baseline while also learning gender-neutral speaker embeddings. When supplied with the ground-truth gender, VoiceLoop-Fader can reconstruct the original example. But what happens if we *change* the values of the gender factors?

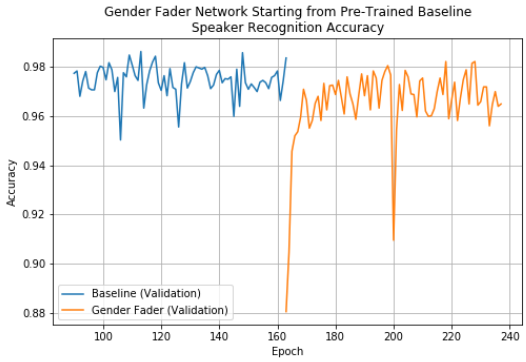
7.3.2 Manipulating the Gender Variables

In 7.3.1 we saw that the gender fader network can *reconstruct* the original training examples when supplied with the ground-truth gender. We now consider in this section whether the gender variables also provide *control* over the characteristics of the generated speech: if we *flip* the gender variable, does the generated speech sound like a plausible version of the original speaker but with swapped gender? And do we get meaningful results if we *sweep* the gender variables outside the binary values the model has seen in training, such as interpolating between pure male and female attribute, and even using extreme values such as 2xmale or both genders set to 'off'?

[1]For comparison, I trained a discriminator on the baseline model's embeddings and found that it almost immediately learned to classify with 100% accuracy, with entropy approaching zero.



(a) Reconstruction loss



(b) Speaker recognition accuracy

Figure 7.3: Reconstruction loss and speaker recognition accuracy for the gender fader approach starting from the pre-trained baseline model.

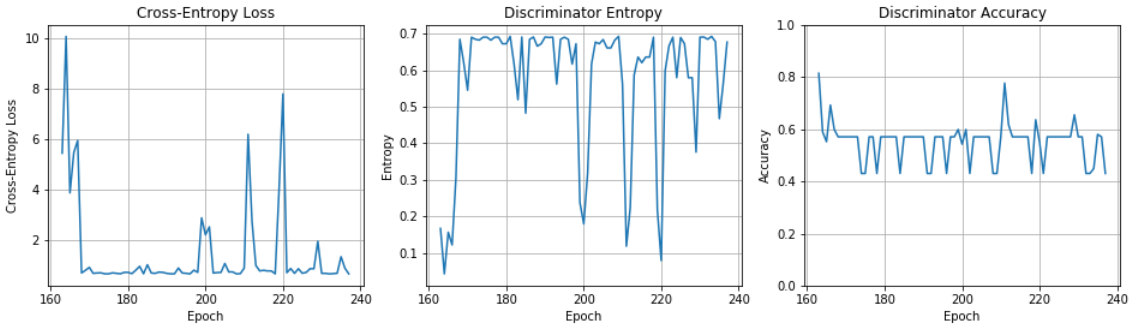


Figure 7.4: Discriminator loss for the gender fader approach starting from the pre-trained baseline model.

Qualitative Test: Listening to Samples

The first step is to listen to samples with manipulated genders and subjectively evaluate whether the perceived gender has changed while other attributes such as accent have been preserved.

I have uploaded samples for six speakers here: `https://richardsterry.github.io/msc-project/fader_networks.html`. Consider the results for the Scottish male 272. The table in the **'Fading Between Genders'** section displays samples generated by manipulating the gender variable for three sentences[2]. The first row contains the original VCTK sample (synthesized from the ground-truth WORLD features.) The next row $[1, 0]$ contains the sample generated with the male factor set to 1 and the female set to 0, i.e. the VoiceLoop-Fader reconstruction of the original, as this speaker is male. These reconstructions sound pretty good. The final row $[0, 1]$ contains the samples of speech generated in a female voice. I think they sound good and can be perceived as the original speaker as a female, preserving non-gender attributes of the speaker, especially the accent.

[2]These sentences come from the training set, but as I'm focusing on manipulating speaker representation rather than reconstruction quality this is not too severe an issue.

The remaining rows contain samples in which the gender variables are 'faded' from pure male to pure female. During training the network only saw examples with pure male or pure female gender factors, so these interpolations test the robustness of the representation. Again, I think these are reasonably convincing interpolations between pure male and pure female voices, and that the voice changes smoothly with the factors with other attributes preserved.

The next table on the webpage, **'Extreme Values of Gender Factors'**, contains samples with more extreme settings such as hyper-male $[2, 0]$, hyper-female $[0, 2]$, and no gender $[0, 0]$. Again, these settings are well outside what the network saw in training and give a good test of the robustness of the speaker representations. Although the results are less robust than the earlier samples, they are still generally good and are interpretable: $[2, 0]$ is an extreme, deep male voice while $[0, 2]$ is an extreme female.

So subjective evaluation of samples supports the assertion that VoiceLoop-Fader has learned a representation in which gender can be controlled without affecting other perceived attributes.

Is VoiceLoop-Fader Simply Learning to Change Pitch?

One immediate concern is that the network is simply learning to alter the pitch of the voice, and that we perceive this as a change in gender. Although this would still useful, it would be much less impressive.

Figure 7.5 shows an analysis of the normalised lf0 acoustic feature (i.e. pitch) sequences for the male-to-female sweep for the Irish male speaker 298. It is clear that as the factors move from pure male to pure female, the *average* pitch does increase monotonically (middle plot). However, the lf0 sequences are not simply parallel shifted: the left plot shows a much more complex relationship as the gender factors change. The right hand plot shows that the sequence length changes, but is not monotonically increasing. And examination of the other acoustic features (not shown) shows that there are many other differences in feature sequences as the gender factors change[3].

Therefore, I don't think that VoiceLoop-Fader is simply learning to change pitch as a proxy for manipulating gender. Although pitch is an important aspect, the changes are far more nuanced than that.

Are These Really New Voices?

Another concern is that the model isn't really generating new voices corresponding to the new speaker embeddings - instead, it could simply be finding the 'nearest' speaker of the opposite gender out of the original speakers, and using that voice. For example, rather than turning speaker 272 from a Scottish male into a female while retaining all other attributes, it could just be finding the best Scottish female speaker in the training set and using that voice. This could still *sound* quite convincing, although would be much less of a technical achievement.

[3]In addition, I believe that simply shifting lf0 and leaving all other acoustic features unchanged would not sound as natural as these samples.

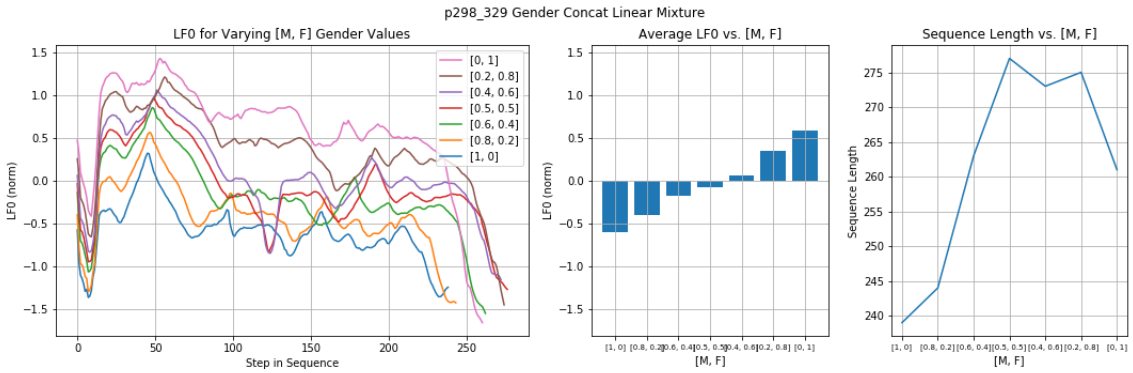


Figure 7.5: LF0 features for a sweep of the gender variables from male to female for speaker 298, sentence 329.

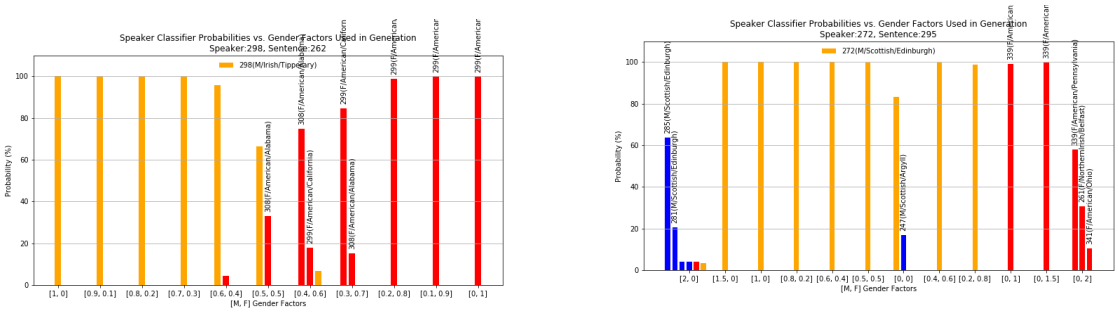
It's interesting to see what the speaker classifier we built back in section 5.1.2 makes of these new voices. Recall that the speaker classifier evaluates which speaker's voice is being used in a given sample of audio, outputting probabilities for each of the 107 speakers in the dataset. In figure 7.6 I show how the speaker classifier's output probabilities change as the gender variables are altered.

Figure 7.6a shows an example of a linear gender sweep for speaker 298, a Scottish male. When the decoder used the ground-truth gender $[1, 0]$, the speaker classifier outputs the correct speaker identity with almost certainty (the classifier's output probability for the correct speaker identity is indicated with an orange bar). As the gender sweep moves to 50:50 it still ranks the correct speaker as most likely, but is much less sure. By the time the gender is above $[0.3, 0.7]$ it no longer assigns a material probability to the original speaker, and now thinks the sample is probably from the American female speaker 299. So, the most likely known speaker is *American* rather than Scottish, even though the generated speech seems subjectively to be a reasonable approximation of a Scottish accent. As another test, I generated the same sample using speaker ID 299 and found that it didn't sound the same as speaker 298 with gender flipped to $[0, 1]$ (you can judge for yourself by listening to the sample on the website). Therefore, we can be reasonably confident that the gender fader model is to some extent producing new voices, rather than simply recycling known speakers.

Figure 7.6b shows a similar analysis for more extreme gender values. The speaker classifier again identifies the original speaker for values around $[1, 0]$. However, when we use the extreme male $[2, 0]$ it gives only 5% probability to the original speaker, and considers other Scottish males more likely, with no strong conviction overall. When we use the extreme female $[0, 2]$, again there is no strong conviction as to which female voice it sounds like. Again, there is reasonable evidence here that the model is producing 'new' voices, rather than simply recycling known speakers.

Quantitative Tests: Gender Classifier

As a final quantitative test, I ran a gender classifier on generated samples to see whether this agrees with the gender factor values used to generate them.



(a) Linear sweep from male to female

(b) Extreme value sweep from hyper-male to hyper-female

Figure 7.6: Most probable speakers as judged by the speaker recognition model, for sweeps of the gender variables for two example utterances. Orange bar is the probability the speaker recognition model assigns to the correct speaker. Red bars are incorrect females; blue bars are incorrect males. Bars are shown for all speakers receiving at least 5% probability.

First, I built a gender classification network that predicts the speaker's gender from an audio sample (sequence of WORLD features.) This is essentially the same as the speaker classifier network discussed in section 5.1.2, except that the final output layer now maps to two output classes rather than 107, and the network is trained on ground-truth gender labels rather than speaker ID labels. Unsurprisingly, as this is a much easier problem than speaker classification, the network trains quickly to give an accuracy of 99.5% on the validation set. So, given an unseen audio sample (with no labels), the gender classifier infers the correct speaker gender with almost perfect accuracy.

Then, for a random set of examples drawn from the VCTK validation set, I generated one speech sample using VoiceLoop-Fader with the gender factors set to $[1, 0]$ and another with the factors set to $[0, 1]$. I passed these samples through the gender classifier and noted the predicted probability that the speaker of the sample was male. If the gender classifier classifies the $[1, 0]$ samples as male with high probability, and the $[0, 1]$ samples as female with high probability, then we have objective evidence that VoiceLoop-Fader's gender factors do indeed control perceived gender.

Figure 7.7 shows the distributions (over my random set of example) of the probability the classifier assigns to the samples being male. The orange distribution is for samples generated with $[0, 1]$, i.e. female; the blue distribution is for $[1, 0]$, i.e. male. The classifier typically has very high confidence in the correct gender, and makes hardly any errors at all (i.e. assigning less than 50% probability to the correct class). It is very clear that the VoiceLoop-Fader gender factors give control over perceived gender.

7.3.3 Is Gender Disentangled?

We've seen in 7.3.2 that the two reserved variables give good control over the gender attribute. But to achieve disentanglement, we also need require that none of the other variables in the speaker representation are related to gender.

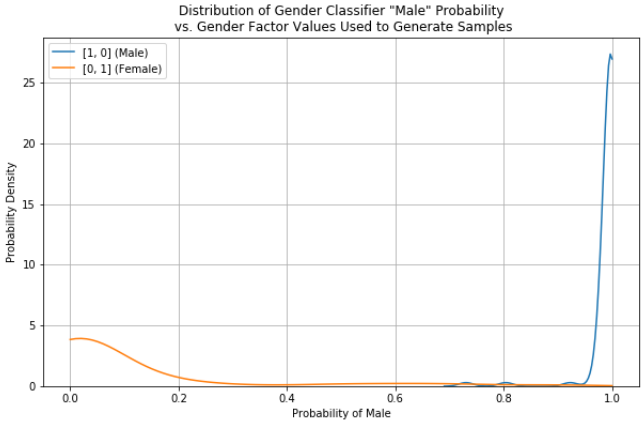
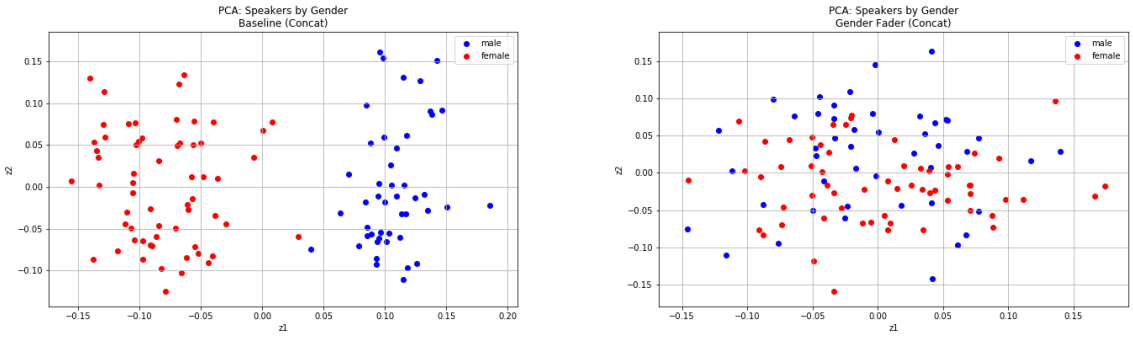


Figure 7.7: The probability of the speaker's voice being male, as judged by a speaker classifier trained on VCTK, for synthesized samples generated by VoiceLoop-Fader with manipulated values of the gender factors.

Are the Embeddings Gender-Neutral?

We saw in section 5.2.2 that a PCA analysis of the speaker embedding space in baseline VoiceLoop shows a very clear separation by gender: figure 7.8a is a reminder of the plot for the baseline VCTK-All-107 model. In contrast, the same analysis on the 254-dimensional embeddings for the gender-fader-all-107 model shows no evidence of separation by gender (figure 7.8b). Although this is a simplistic linear analysis, and doesn't imply 'complete' disentanglement of gender from the gender-neutral speaker representation, it illustrates that VoiceLoop-Fader has certainly removed simple entanglement.



(a) Baseline model speaker-vctk-all-107

(b) Gender fader model gender-fader-all-107

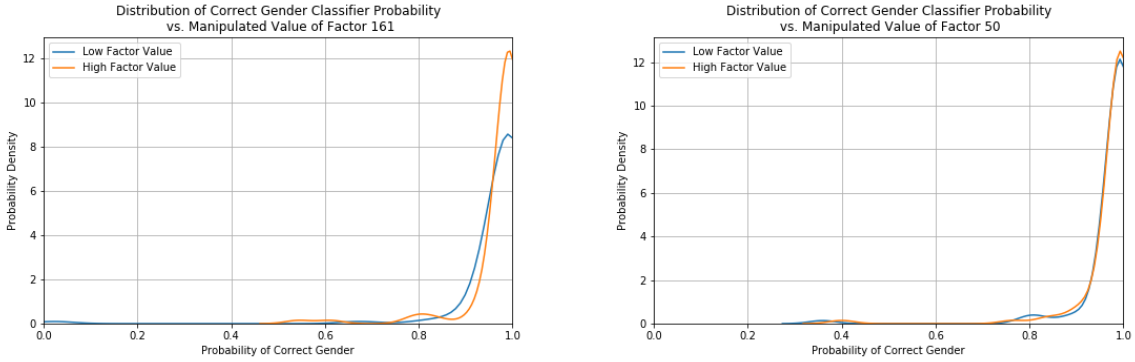
Figure 7.8: First two principal components of speaker embeddings labelled by speaker gender

Sweeping Other Latent Variables

Finally, I performed an analysis similar to section 7.3.2 to investigate whether any of the factors in the gender-neutral part of the embedding space have an impact on perceived gender.

Consider an arbitrary factor in the gender-neutral part of the space: index 99, for example. For a random set of examples drawn from the VCTK validation set, I generated one speech sample using VoiceLoop-Fader with factor 99 set to a low value (based on the empirical distribution) and another with the factor set to a high value. In both cases, the gender factors were set to the ground-truth value. I passed these samples through the gender classifier and noted the predicted probability for the correct, ground-truth gender. I then investigated whether there was any difference in the distribution of these probabilities for low vs. high values of factor 99. A significant difference would imply that factor 99 is entangled with the gender attribute, such that varying factor 99 causes changes in perceived gender.

Figure 7.9a illustrates the results of flipping factor 161 between low and high values. The blue plot is the distribution of probabilities assigned to the correct gender when factor 161 is set to be low. The orange plot is the distribution when factor 161 is set to be high. There is no clear difference between these distributions: in both cases the gender classifier normally assigns a very high probability to the correct class, and only very rarely does it assign less than 50%. There is therefore no evidence that factor 161 is entangled with the gender attribute.



(a) Factor 161

(b) Factor 50

Figure 7.9: Probability the gender classifier assigns to the correct gender, for samples generated with manipulated values of factors from the gender-neutral part of the embedding space.

Similar results hold for the other factors: see figure 7.9b for another example. Although it's possible that some combination of the factors has an impact on gender, it seems reasonable to conclude that the representation is disentangled to a good degree.

7.4 Discussion

The results of section 7.3 show that the VoiceLoop-Fader approach succesfully learns a speaker representation in which gender can be controlled, and is to some extent disentangled from other generative attributes.

Despite the good results, I have a concern that the discriminator network may be less important than it first appears. I ran some rough experiments in which I kept the one-hot gender factors and the 254-dimensional learned speaker embedding, but ignored the adversarial discriminator loss in

the objective function (i.e. set $\lambda = 0$). There was evidence that the gender factors could still control the perceived gender of the output samples reasonably well, and that the 254-dimensional speaker embeddings were less entangled than in the baseline VoiceLoop model. This implies that most of the work is being done by enforcing gender factors in the speaker representation, rather than by the discriminator. Supplying the gender factors may be enough of a hint for the model to work out that gender is an important variable. This effect is likely increased by the fact that the gender factors are set to 1, whereas the 254-dimensional learned embedding has a max norm constraint of 1, so the scale of the other factors is much lower than the gender factors. This was unintentional, but may have had a significant effect.

A major limitation of my research is that I based it on the baseline *speaker embedding* VoiceLoop architecture: with such a small number of speakers, the discriminator network is not a robust approach. In future work, I would apply the fader approach to the *utterance embedding* architecture of chapter 6, which gives around 40,000 embeddings on which to build a discriminator rather than the 107 speakers. Of course, the 40,000 utterance embeddings cluster by speaker so the increase is not as dramatic as it first seems, but I would expect it to help.

This would make it possible to carve out additional factors from the embedding space to represent other labelled attributes such as accent or age. Given the initial success of the accent transformation in section 5.2.3, it would be interesting to see whether a fader approach can learn to disentangle accent.

Chapter 8

Disentangling Without Labels: BetaVAE

In chapter 7 I used known, labelled attributes to encourage the model to learn a disentangled latent space.

In this chapter, I investigate β-VAE, an unsupervised technique that encourages VoiceLoop to disentangle generative attributes across the latent factors *without* using labelled attributes for each speaker or utterance. The model is encouraged to learn the attributes itself from patterns in the data alone.

This is much more ambitious, but also more powerful: as well as allowing us to build interpretable models for datasets that don't have labels, it could potentially identify addditional unlabelled attributes in partially-labelled datasets. For example, as well as disentangling factors for gender and accent in VCTK, an unsupervised approach might be able to disentangle factors for average speaking rate, dynamic range, average intonation patterns, fundamental frequency (orthogonal to that captured by gender) and so on. Unsupervised approaches are also more robust to labelling errors, missing labels, and unclear binary labels (e.g. speakers are binary male/female, but their voices are clearly on a spectrum.)

8.1 β-VAE

I first introduced β-VAE[20] in section 2.3, as an extension of the vanilla VAE framework with an additional hyperparameter β that controls the relative importance of the reconstruction and regularization components of the training objective:

$$\mathcal{L}_\beta = \frac{1}{N}\sum_{i=1}^{N}\Big(\mathbb{E}_q[\log p(x^{(i)}|z)] - \beta D_{KL}\big(q(z|x^{(i)})||p(z)\big)\Big) \tag{8.1}$$

A larger value of $\beta > 1$ imposes a stronger constraint on the latent bottleneck, and limits the

capacity of z. As the KL penalty increases, the model must learn a more efficient latent representation of the data. If the true data generation process has some independent generative attributes, then these are pushed into independent latent variables and we achieve some degree of disentanglement.

However, a higher value of β will tend to increase the reconstruction loss and result in lower-quality outputs. β must therefore be optimized as a hyperparameter to find the optimum trade-off between reconstruction quality and disentanglement.

8.2 Related Work

As discussed in section 1.3.3, VAELoop[1] modifies VoiceLoop by replacing the speaker embedding lookup table with a variational autoencoder. The paper aims to learn to generate more expressive speech without relying on expression labels, which it refers to as 'unsupervised expressive speech synthesis'. The VAE is intended to capture global characteristics of speech such as gender, emotion, and speaker identity and encode them in a tractable probability distribution. My approach is similar, but I focus more explicitly on disentanglement, and whereas [1] uses a standard $\beta = 1$ VAE, I will investigate the impact of varying β on disentanglement and reconstruction quality. In addition, [1] use only the VCTK-US-21 dataset: I find that the speaker variation in this dataset is too low to give clear disentanglement, but get stronger results by running on the larger and more diverse VCTK-All-107 dataset.

8.3 Architecture: VoiceLoop-VAE

My VoiceLoop-VAE architecture is essentially an extension to the utterance embedding model I discussed in chapter 6: see figure 8.1. Now, rather than mapping each utterance to a single 256-dimensional embedding, the utterance encoder network produces two vector outputs: the mean and log-variance of the distribution of z given x, assuming a multidimensional Gaussian with diagonal covariance. So each individual utterance x now maps to a *distribution* of embeddings z over the latent space, $p(z|x)$, rather than deterministically mapping to a single embedding $x \rightarrow z$. Each example maps to a region of embedding space, rather than a fixed point.

The network begins with the same structure as in chapter 6: five convolutional layers followed by mean pooling. From this point, there are two separate fully connected layers, one that maps this internal representation to μ and the other to logvar $= \log \sigma^2$. So the mean and standard deviation are derived from a common internal representation.

Although we have a distribution for $z|x$, the decoder requires a single embedding vector z. Selecting z is handled differently depending on whether we are training the network, or using a pre-trained network for generating samples (inference mode).

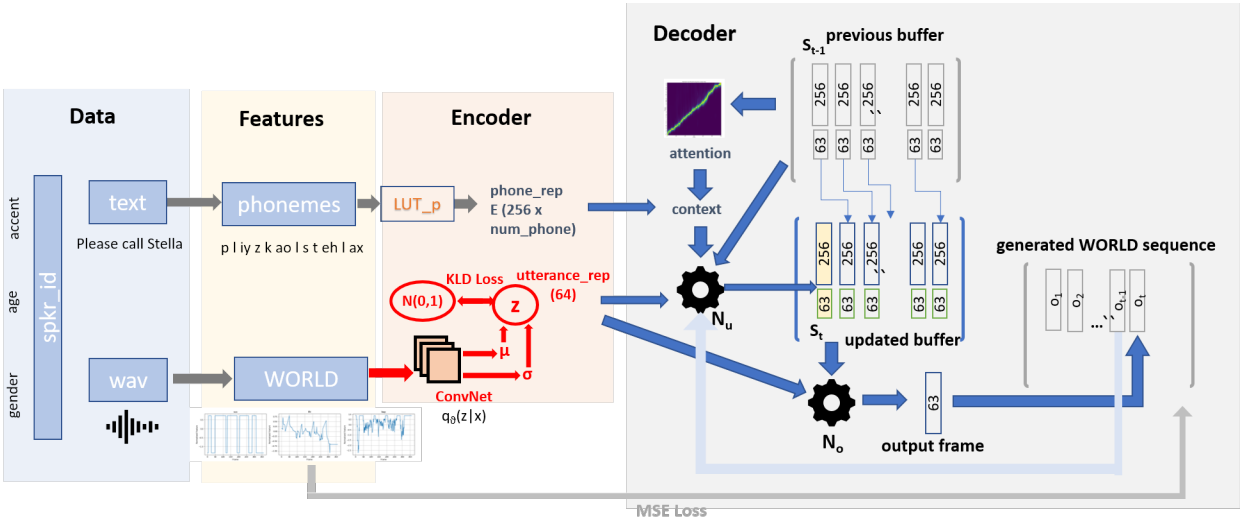


Figure 8.1: Example WORLD features for speaker 301, sentence 117

Training Embeddings

During **training**, we use the reparamaterisation trick[28] to sample a value of z from $p(z|x)$ while still enabling loss gradients to flow back through the network. We sample a n-dimensional vector ϵ, with each component sampled independently from the standard normal distribution, and calculate z as follows:

$$z = \mu + \epsilon e \sigma \qquad \text{where } \sigma = e^{0.5\text{logvar}} \tag{8.2}$$

The model therefore learns to generate samples with low reconstruction loss for points throughout the region of $z|x$.

At the same time, the model also aims to minimize the KL divergence between the empirical distribution $p(z|x)$ of the latent variable and the multivariate normal distribution $\mathcal{N}(0, \mathcal{I})$. In practice, this is done by calculating the KLD over each mini-batch and adding that to the batch's MSE loss. The KLD for a mini-batch of examples is given by[28]:

$$-\frac{1}{2n}\sum_{i=1}^{n}\left(1 + \text{logvar} - \mu^2 - e^{\text{logvar}}\right) \tag{8.3}$$

Inference Embeddings

When using the model in **inference** mode for new strings of text, we pick a reference utterance that we want the generated speech to sound like, just as in the VoiceLoop-Utterance approach. The encoder maps this utterance to a distribution of $z|x$ parameterised by μ and σ. However, in inference mode we simply set $z = \mu$ to use the MAP point in the embedding space.

8.3.1 Dimensionality of the Embedding Space

During the initial research phase I experimented with various architectures. For example, I added atanh activation and max norm constraints on the outputs of the fully-connected layers for μ and logvar in an attempt to enforce ranges on the outputs; I also tried max-pooling across time rather than mean-pooling. However, the most important hyperparameter was the dimensionality of the embedding space.

During my initial experimentation I decided to reduce the dimensionality of the embedding space from 256 to 64. The main motivation was to improve the chances of forcing disentanglement, giving the model fewer degrees of freedom with which to represent the utterances and perhaps forcing it to isolate the most important attributes in individual factors. The choice of 64 also matched [1].

This requires a very small architectural change: an additional linear projection layer was needed to map the 64-dimensional embedding into the 256 dimensions required for the decoder.

In further research I would investigate reducing the dimensionality even further, to 32 or even 16 dimensions, to see whether this encourages learning disentangled representation and understand how the reduced capacity affects the reconstruction error.

8.4 Results

The β-VAE results split into two parts. As the full All-107 model takes a number of days to converge, my initial experimentation and β sweep experiments were done on the smaller US-21 dataset. In section 8.4.1 I show that, as expected, higher beta values force the posterior distribution closer to the prior, at the cost of worse reconstruction loss and (subjectively) lower-quality samples with less speaker variation. However, I didn't find any clear evidence of disentanglement, which I attribute to the limited diversity of speakers (all American accents, only four males).

I had time to run one full simulation on the full All-107 dataset: I picked the default $\beta = 1$. In section 8.5 I show that although the reconstruction quality is reduced, β-VAE has learned to disentangle gender despite not having access to any gender labels.

8.4.1 VoiceLoop-VAE-US-21

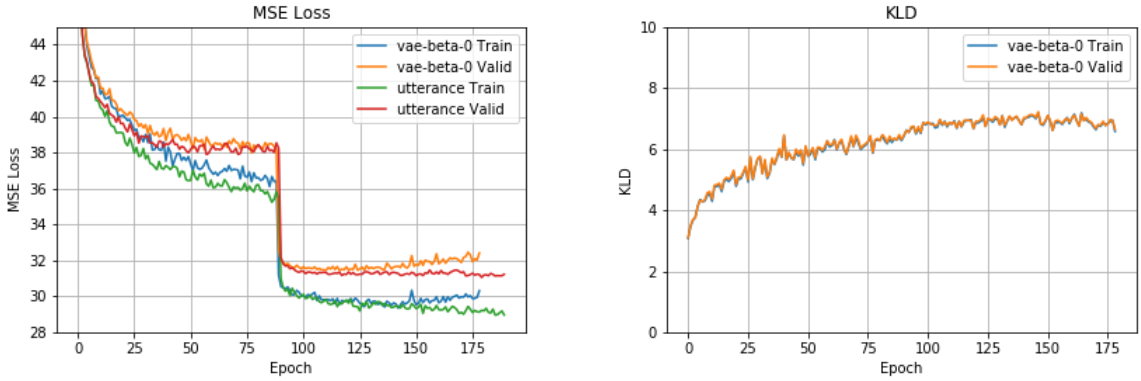
As with the baseline models, I trained in two phases of 90 epochs each. I ran simulations for five values of the β hyperparameter:

- $\beta = 0$: No constraint on KLD
- $\beta = 0.5$: Weak KLD constraint
- $\beta = 1$: Standard VAE
- $\beta = 2$: Stronger encouragement to disentagle

- $\beta = 5$: Very strong encouragement

$\beta = 0$ Case

As a preliminary, we consider the case where $\beta = 0$, i.e. the KLD term is completely ignored and the distribution $p(z|x)$ is unconstrained. As shown in figure 8.2, the loss curves take longer to converge than in the utterance embedding model, and VAE-Beta-0 model shows evidence of over-training in later epochs. This makes sense, as the model now has to learn to output σ as well as μ: this extra complexity is likely to require more training time, and to give more potential for over-fit. The validation MSE loss of the best model is 31.4, which is a little worse than for utterance-US-21 (31.1) but still reasonably good. Figure 8.2b shows that in the absence of a β constraint, the KLD value grows to a large value as the model can learn whatever distribution it likes for $z|x$.



(a) MSE

(b) KLD

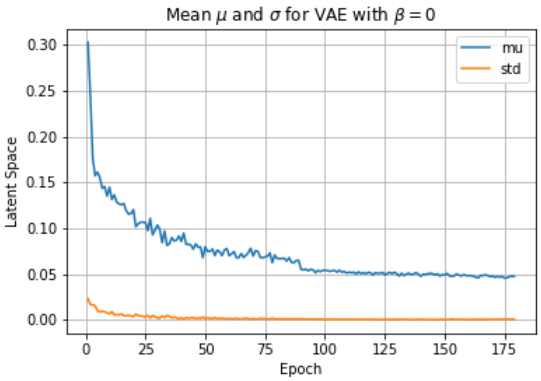
Figure 8.2: Training curves for BetaVAE with $\beta = 0$ on US-21

Figure 8.3a shows how the average μ[1] and σ values (computed over all the utterances in the validation set) evolve during training. Average σ drops quickly, and by about the $50th$ epoch is very close to zero against a μ of around 0.05. So in the absence of a KLD constraint, the model has learned to map each utterance to a fixed point given by μ rather than a distribution over the space - it has effectively reverted to the utterance embedding model of chapter 6. Figure 8.3b shows the accuracy of the speaker recognition model on samples generated from the validation set. Initial accuracy is low, and it takes until around epoch 25 to reach 80%, once σ has been pushed down close to zero and the generated voices become deterministic. The accuracy of the best model is 87.5%, which although reasonably good is far short of the 97.1% achieved by the utterance embedding model.

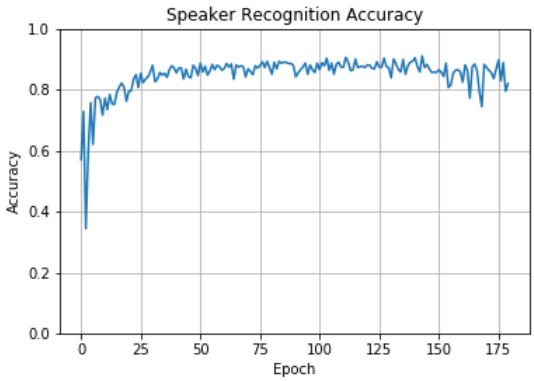
Non-Zero β

Figure 8.4 shows how the training curves change for different values of β. As β increases from zero, the bottleneck tightens and the KLD penalty has an increasing impact on the model. The

[1]Measure as root mean square, so this is an average magnitude.



(a) Mean μ and σ



(b) Speaker recognition accuracy

Figure 8.3: BetaVAE with $\beta = 0$ on US-21

	Validation Loss			$p(z\|x)$		
Sim	Total Loss	MSE	KLD	μ	σ	SpkrRecog(%)
Speaker-US-21	*31.6*	*31.6*	-	-	-	*95.0*
Utterance-US-21	*31.1*	*31.1*	-	-	-	*97.1*
VAE-β-0	31.4	31.4	6.940	0.052	0.001	87.5
VAE-β-0.5	31.7	31.4	0.616	0.629	0.443	86.5
VAE-β-1	31.8	31.5	0.241	0.563	0.726	71.6
VAE-β-2	31.8	31.7	0.072	0.371	0.885	39.3
VAE-β-5	31.7	31.7	0.000	0.004	0.992	4.1

Table 8.1: Losses and speaker classifier accuracies for the utterance embedding models. Corresponding figures for the baseline speaker embedding models are given in brackets.

plots in the right-hand column show that for $\beta = 1$, for example, KLD falls quickly over the first 25 epochs before reducing gradually. For $\beta = 5$ KLD falls rapidly, and is pushed almost to zero i.e. the distribution of $z|x$ reverts to the unit Gaussian prior, with information in the source utterance x being ignored when calculating z. The bottom-centre plot shows that validation MSE (reconstruction) loss is worse for higher values of β.

Figure 8.5 shows how average μ and σ across the set of validation utterances depend on the value of β. For $\beta = 1$, average μ is around 0.5 and relatively stable, while σ rises towards 0.8. For the unit Gaussian prior the corresponding values would be 0 and 1, which implies that while constraining the distribution, $\beta = 1$ still allows the model to store a reasonable about of information in the $p(z|x)$. In contrast, for $\beta = 5$, the model learns to set $\mu = 0$ and $\sigma = 1$, reverting to the prior as we saw in the KLD plot.

Table 8.1 summarises these results for the best models of each β value. As β increases, the distribution is more constrained with lower KLD, higher MSE (reconstruction loss), and μ and σ values that approach the prior. Unsurprisingly, the speaker recognition accuracy also worses with β, and by $\beta = 5$ it's at completely random level. The results are all worse than the VoiceLoop-Utterance-US-21 benchmark.

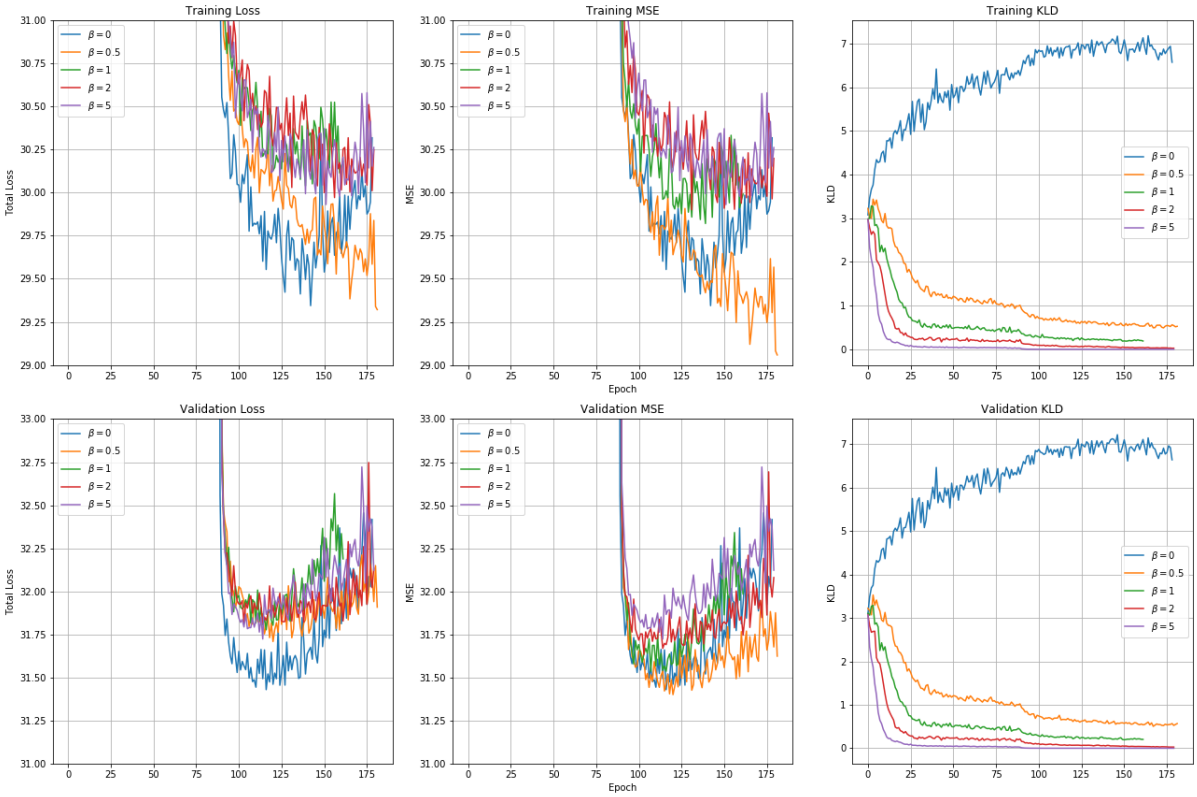
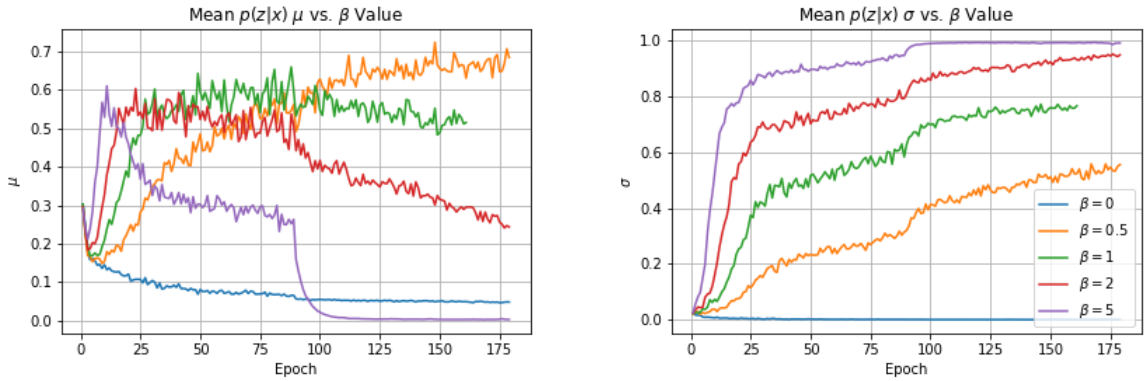


Figure 8.4: Training curves for BetaVAE with $\beta > 0$ on US-21

Generating Samples

I investigated the impact on generated speech of sampling z in the following ways; you can listen to examples and see the lf0 feature comparisons on the website:

- **Sample from the Distribution of a Reference Utterance** $z \sim p(z|x)$: I selected a reference utterance x, drew samples from $p(z|x)$, and compared speech generated using different z. Anecdotally, it seems that for low β most examples sound close to the speaker of the reference utterance, as $p(z|x)$ is concentrated on a small region of embedding space. For mid-range values of β there is more variability because $p(z|x)$ is less concentrated. For $\beta = 5$, there was less variability but now samples don't sound much like the speaker of the reference utterance: the model appears to have learned some sort of average voice, and $p(z|x) \approx p(z)$. These results are only subjective, though, and like the eye, the ear can be deceptive when looking for patterns.
- **Samples from the Prior** $z \sim p(z)$: I sampled embeddings at random from the prior $p(z)$, and compare the range of speech generated from them. Anecdotally, for low β we get a lot of variability in the speech as the model has learned to encode a range of voices across the speaker space in the region covered by the prior. For higher β the variability reduces, and by $\beta = 5$ most of the speech samples sound similar, as variations on an average voice. Again, this is very subjective, but it's consistent with the other results.



(a) μ (b) σ

Figure 8.5: Average μ and σ for BetaVAE with $\beta > 0$ on US-21

What about Disentanglement?

So far this is all what we'd expect to see if the architecture is working correctly. The real question is whether the models have learned to disentangle any generative attributes. Unfortunately, I could not find any clear evidence of disentanglement in the speaker space. I suspect this is because there is limited speaker diversity in the US-21 dataset, with only 21 speakers, all of them having American accents, and only four being male.

8.5 VoiceLoop-VAE-All-107

The All-107 dataset has much more diversity, but is too large for development work. I had time for a single simulation, and chose to use $\beta = 1$ as a reasonable default value. This model trained reasonably well, and although the reconstruction quality is not great, there is good evidence that the model has learned to disentangle the gender attribute.

8.5.1 The Model Learns to Ignore Some Factors

Figure 8.6 shows the average μ and σ in each of the 64 embedding space dimensions/factors. Each average is computed over the embeddings of all utterances in the validation set[2]. For around a third of the factors, the model has learned to set the average magnitude of μ close to zero and the average σ close to one: for these factors, the distribution has reverted to the prior, and the model is essentially making the factors unconditional on x. Instead, it appears that information is being forced into other factors with higher μ and lower σ, e.g. factor 36 and factor 58.

[2]The 'average' for μ is a root mean square value, so measures the average magnitude

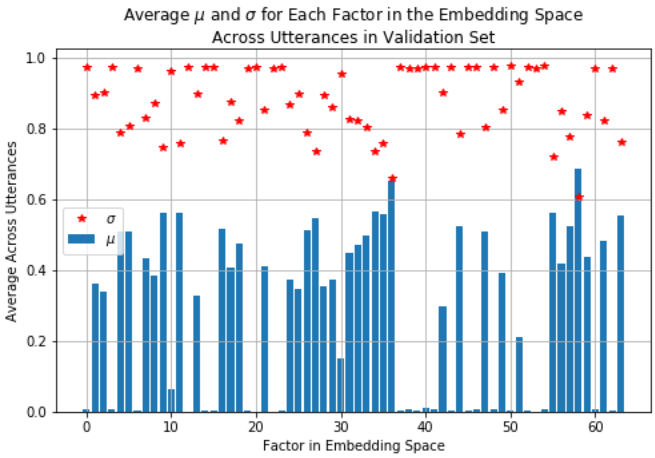


Figure 8.6: Average μ and *sigma* for all embeddings in the validation set

8.5.2 Has Gender Been Disentangled into Factor 58?

In figure 8.7, I consider the distributions of μ values across all validation set utterance embeddings. For each of the 64 factors, I plot the distribution of μ values for utterances coming from male speakers (blue) and female speakers (orange). For all except one factor, the male and female distributions look the same (for the 'ignored' factors, the distributions are just vertical lines because all μ values are essentially zero.) However, for factor 58 there is a clear separation between the distributions of male and female speakers. Male speakers almost always have a positive value of factor 58, while female speakers have a negative value. This suggests that the model has learned to disentangle the gender attribute into factor 58, without ever having seen male or female labels.

8.5.3 Factor 58 Controls Gender

To test whether factor 58 does indeed represent gender, I generated pairs of audio examples, one with factor 58 set to -1 and the other to +1, and then evaluated whether the -1 example sounds like a female voice and the +1 example sounds like a male voice. That is, whether varying factor 58 while holding all other factors equal is sufficient to flip the gender while holding other attributes constant.

Subjective Tests

For the subjective evaluation, I listened to pairs of samples and judged the gender[3]. I found that the gender was reliably flipped when changing from -1 to +1: factor 58 does appear to control gender. However, it wasn't the case that all *other* attributes were unchanged. The alternative voices didn't always sound like pairs up to a gender change - often the accents weren't too similar.

[3]Samples are on the website here: `https://richardsterry.github.io/msc-project/betavae.html`

The results were definitely not as convincing as for the Fader Networks. However, given that β-VAE is solving a much harder unsupervised approach, and that this was the first value of β I tried, the results are encouraging.

Objective Tests

For an objective evaluation, I produced a large number of pairs based on taking utterances from the validation set, finding the embedding $z = \mu|x$, and then setting factor 58 to -1 and +1. I passed the results through the gender classification network, as in section 7.3.2. Figure 8.8 shows the distributions of probabilities assigned to male for samples generated with factor 58=-1 (blue) and +1 (orange). It's clear that for -1, the gender discriminator is very confident that the speaker is female; for +1, it's confident that the speaker is male. This therefore confirms the conclusion of the subjective test: factor 58 does reliably control perceived gender.

8.5.4 Is Gender Entangled With Any Other Factors?

Finally, we need to check whether any other factors play a role in controlling perceived gender. This time I generated pairs of samples in which each factor was flipped between -1 and +1, holding all other factor values unchanged. Again, these were passed through the gender classifier. Figure 8.9 shows, for each factor, the distributions of the probability assigned to the correct gender (as measured by the gender of the speaker who spoke the reference utterance). For all factors except factor 58, The distributions for test factor=-1 are not significantly different to those for +1 (in both cases, they assign the vast majority of the probability to the correct gender). This means that changing the factor value from one end of its distribution to the other does not have an impact on perceived gender. In other words, none of the other factors plays a role in controlling gender.

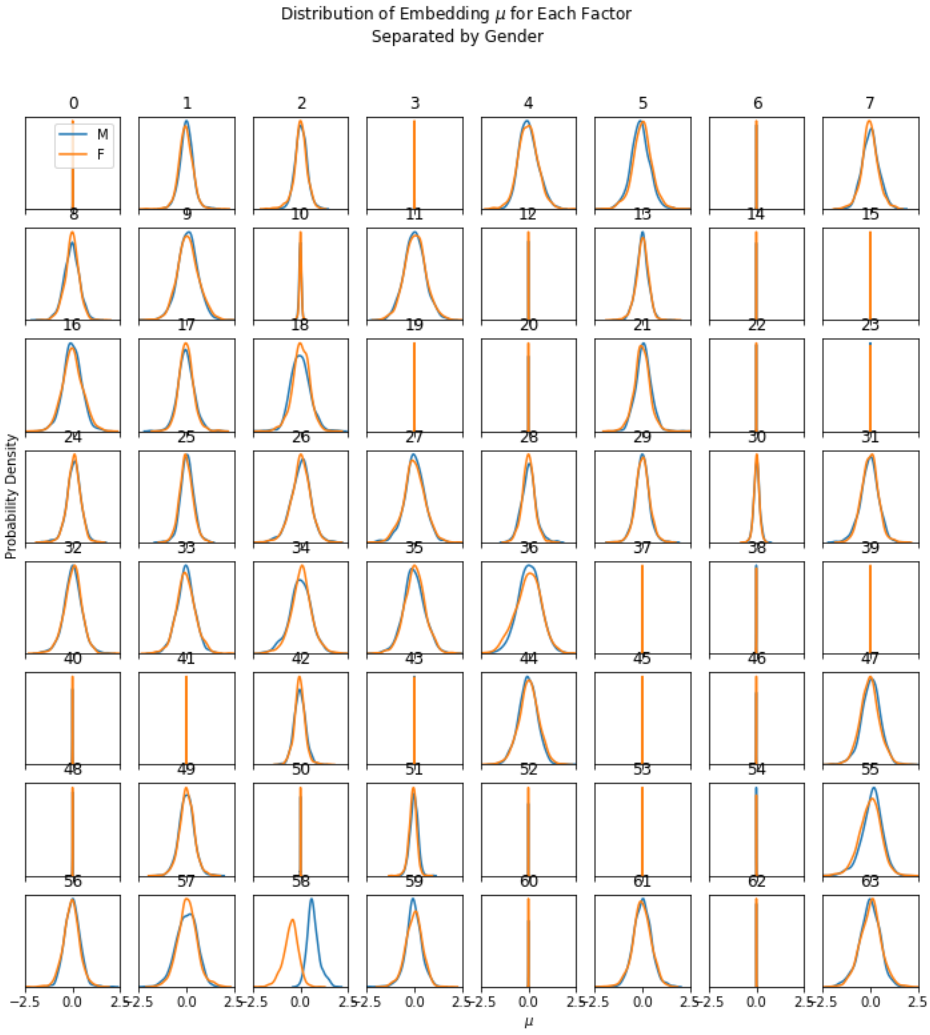


Figure 8.7: Distribution of μ by gender, for each latent factor

Figure 8.8: Probability assigned to 'male' by a gender classifier, for speech generated generated with factor 58 = -1 (blue) and factor 58 = +1 (orange)

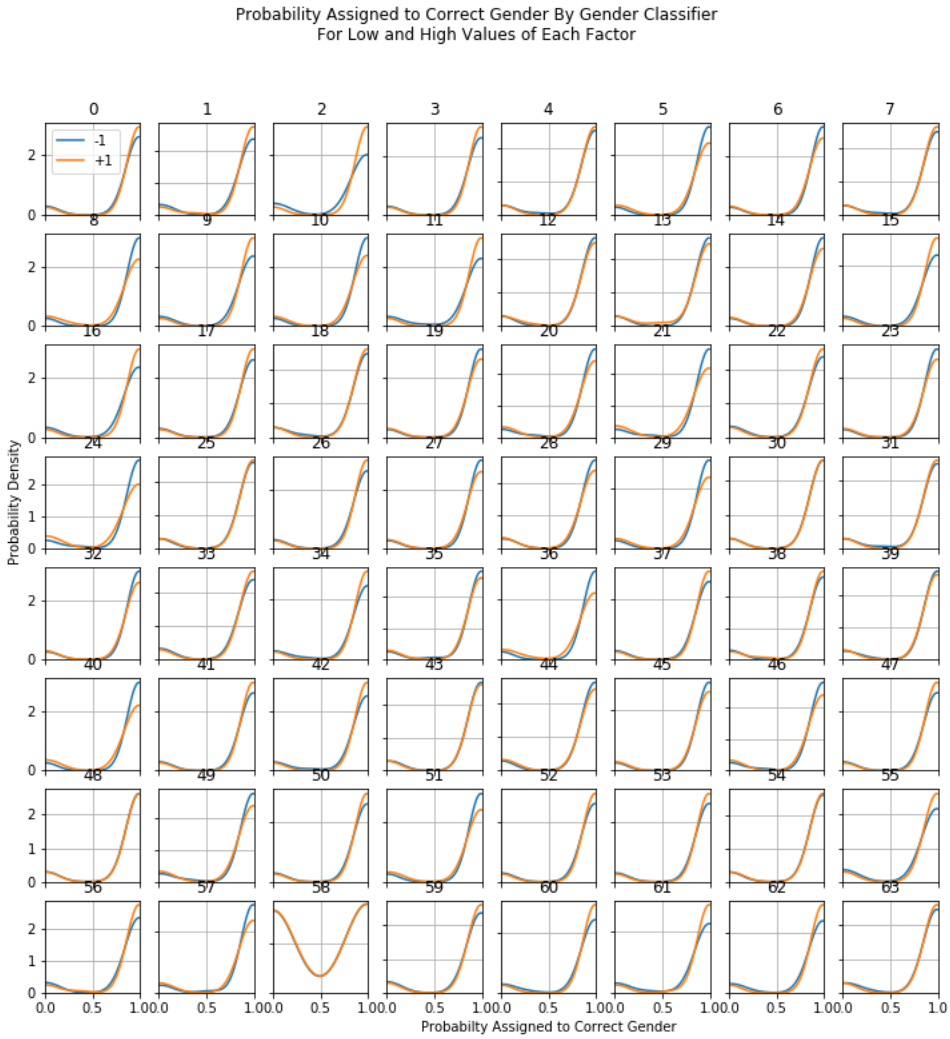


Figure 8.9: Probability assigned to correct by a gender classifier, for speech generated generated with low (blue) and high (orange) values of each factor

8.6 Summary

In this chapter, I applied the β-VAE approach to VoiceLoop in an attempt to force it to learn a disentangled representation without attribute labels. Experiments on the smaller US-21 dataset showed that the model was working as expected, giving up some reconstruction quality in return for a constrained posterior distribution. However, there was no evidence of disentanglement on US-21. My one-shot experiment on the All-107 dataset did show evidence of disentanglement, with the gender attribute disentangled into factor 58 and no other factors having a clear impact on perceived gender. However, the quality of samples was lower, and it didn't seem like all other speaker attributes were being preserved when flipping gender using factor 58.

It was disappointing that I didn't find any clear evidence of disentanglement of accent. In future research I would sweep the β value to see how well this controls the level of disentanglement. I would investigate reducing the dimensionality of the embedding space to 32 or even 16. I would also investigate more systematically the effects of varying other factors to see if any have learned meaningful generative attributes such as pitch: factor 36, for example appears to control the speaker's initial pitch at the start of the utterance, although this remains anecdotal at this stage.

Finally, I would like to explore the ideas in BetaTCVAE[9] and FactorVAE[26] that extend the original β-VAE loss decomposition to improve the trade-off between reconstruction and latent channel control.

Part IV
Discussion & Conclusions

In this final part, I draw my report to a close by summarising the results, reflecting on what worked well and what did not, and discussing extensions.

Chapter 9

Discussion & Conclusions

9.1 Report Summary

In my project, I aimed to investigate whether state-of-the-art disentanglement techniques could be applied to a neural TTS system to encourage it to learn a disentangled speaker representation, which would provide interpretability and control over characteristics of the generated speech.

I used FAIR's VoiceLoop as my baseline model, trained on two subsets of the VCTK dataset: US-21 and All-107. I showed that VoiceLoop can learn to generate speech in the voices of the training set speakers, and that the speaker embedding space it learns is interpretable and affords some degree of control over the generate speech, but that the generative attributes are heavily entangled across the embedding dimensions.

I extended the baseline VoiceLoop model to use utterance embeddings rather than speaker embeddings, which allows the model to quickly learn embeddings for new speakers not seen in the training set, and also allows us to investigate basic prosodic variation of an individual speaker's voice, rather than simply learning a single average voice for each speaker.

In Chapter 7 I aimed to learn a disentangled speaker representation by supplying the VoiceLoop-Speaker model with gender labels, using a supervised Fader Networks-style adversarial approach. The speaker representation is formed by concatenating one-hot gender labels onto a learned gender-neutral speaker embedding. The network is then trained in a two-player minimax game, with the encoder-decoder aiming to reconstruct training examples while fooling an adversarial discriminator network that tries to determine a speaker's gender from the gender-neutral part of their speaker embedding. The results were encouraging, with the gender factors giving a good level of control over the perceived gender of the generated speech while retaining other speaker characteristics such as accent, and allowing us to fade between genders despite the model only seeing binary genders during training. My investigations confirmed that gender had been disentangled from the gender-neutral representation to a good degree.

In Chapter 8 I aimed to learn a disentangled representation in the VoiceLoop-Utterance version of

the model without supplying the model with labels, using an unsupervised β-VAE approach. The deterministic mapping from utterance to embedding is replaced with a probabilistic interpretation, with a neural network learning a mapping from utterance to a distribution over embedding space. The network is trained to reconstruct training examples with an additional KL-Divergence regularisation term that penalises posterior distributions that deviate strongly from a unit Gaussian prior. The balance between reconstruction and KL-Divergence terms is set by a hyperparameter β, which controls the capacity of the latent bottleneck and in theory can be tuned to encourage the model to learn a disentangled representation. Results on the US-21 dataset confirmed that the model internals were working as expected, with higher values of β resulting in posterior distributions that collapse towards the prior at the cost of worse reconstructions. Results on the All-107 dataset showed good evidence that β-VAE had learned to disentangle gender, although reconstruction quality was low and changing the gender variable did not leave all other attributes unchanged to the extent that the Fader Networks approach did. However, given that this is a much harder unsupervised task, the results were very encouraging.

In conclusion, I have found that a degree of speaker representation disentanglement can be achieved using both supervised and unsupervised approaches. The supervised approach achieved higher quality results, but the unsupervised approach was able to recover a known attribute purely from the data, which is a much more difficult problem.

9.2 Future Research

I've made various suggestions for future research throughout this report. To recap some of the most interesting:

- Apply Fader Networks to utterance embeddings, and attempt to disentangle other known attributes such as accent or age.
- β-VAE: sweep β for the larger All-107 dataset, and investigate the modified approaches in TCBVAE etc. Consider running on a more prosodic dataset.
- Further work on trying to understand structure in the distribution of a single speaker's utterance embeddings. Use a dataset with more prosodic structure e.g. an audiobook, which may be likely to include interesting features.
- Generative flow models such as Glow[27]

9.3 Closing Remarks

This has been a fascinating problem to work on for my Masters project. I feel that I've learned a new domain, done some interesting work, and have ideas for further research that could improve our ability to interpret and control speaker characteristics.

This is an exciting period in TTS technology. It will be interesting to see how research progresses in coming months and years, and whether a neural TTS approach will be able to supplant tra-

ditional, engineered TTS pipelines in production speech technology, opening up a world of new applications.

Bibliography

[1] K. Akuzawa, Y. Iwasawa, and Y. Matsuo. Expressive speech synthesis via modeling expressions with variational autoencoder. *CoRR*, abs/1804.02135, 2018.

[2] S. Ö. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou. Neural voice cloning with a few samples. *CoRR*, abs/1802.06006, 2018.

[3] S. Ö. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, J. Raiman, S. Sengupta, and M. Shoeybi. Deep voice: Real-time neural text-to-speech. *CoRR*, abs/1702.07825, 2017.

[4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pages 214–223, 2017.

[5] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1171–1179, Cambridge, MA, USA, 2015. MIT Press.

[6] Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning Long-Term Dependencies with Gradient Descent is Difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[8] C. P. Burgess, I. Higgins, A. Pal, L. Matthey, N. Watters, G. Desjardins, and A. Lerchner. Understanding disentangling in $beta$-VAE. (Nips), 2018.

[9] T. Q. Chen, X. Li, R. Grosse, and D. Duvenaud. Isolating Sources of Disentanglement in Variational Autoencoders. 2018.

[10] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. 2016.

[11] Y.-A. Chung, Y. Wang, W.-N. Hsu, Y. Zhang, and R. J. Skerry-Ryan. Semi-supervised training for improving data efficiency in end-to-end speech synthesis. *CoRR*, abs/1808.10128, 2018.

[12] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *CoRR*, abs/1605.08803, 2016.

[13] C. Doersch. Tutorial on Variational Autoencoders. pages 1–23, 2016.

[14] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, V. Lempitsky, U. Dogan, M. Kloft, F. Orabona, and T. Tommasi. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research*, 17:1–35, 2016.

[15] A. Gibiansky, S. Arik, G. Diamos, J. Miller, K. Peng, W. Ping, J. Raiman, and Y. Zhou. Deep voice 2: Multi-speaker neural text-to-speech. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2962–2970. Curran Associates, Inc., 2017.

[16] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[17] A. Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[18] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs. 2017.

[19] N. Hadad and L. Wolf. A Two-Step Disentanglement Method. pages 772–780.

[20] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, A. Lerchner, and G. Deepmind. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *Iclr*, (July):1–13, 2017.

[21] W.-N. Hsu, Y. Zhang, and J. Glass. Unsupervised Learning of Disentangled and Interpretable Representations from Sequential Data. (Nips), 2017.

[22] Y. Jia, Y. Zhang, R. J. Weiss, Q. Wang, J. Shen, F. Ren, Z. Chen, P. Nguyen, R. Pang, I. L. Moreno, and Y. Wu. Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis. 2018.

[23] M. J. Johnson, D. Duvenaud, A. B. Wiltschko, S. R. Datta, and R. P. Adams. Composing graphical models with neural networks for structured representations and fast inference. (Nips), 2016.

[24] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[25] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu. Efficient neural audio synthesis. *CoRR*, abs/1802.08435, 2018.

[26] H. Kim and A. Mnih. Disentangling by factorising. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of*

Machine Learning Research, pages 2649–2658, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[27] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *CoRR*, abs/1807.03039, 2018.

[28] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. (Ml):1–14, 2013.

[29] R. F. Kubichek. Mel-cepstral Distance Measure for Objective Speech Quality Assessment. *Proc. of IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, 1:125–128, 1993.

[30] G. Lample, N. Zeghidour, N. Usunier, A. Bordes, L. Denoyer, et al. Fader networks: Manipulating images by sliding attributes. In *Advances in Neural Information Processing Systems*, pages 5963–5972, 2017.

[31] Y. Lee, T. Kim, and S.-Y. Lee. Voice Imitating Text-to-Speech Neural Networks. 2018.

[32] Y. Lee, A. Rabiee, and S.-Y. Lee. Emotional End-to-End Neural Speech Synthesizer. (Nips):1–5, 2017.

[33] Y. Li and S. Mandt. A Deep Generative Model for Disentangled Representations of Sequential Data. 2018.

[34] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. Courville, and Y. Bengio. Samplernn: An unconditional end-to-end neural audio generation model, 2016. cite arxiv:1612.07837.

[35] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[36] M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. pages 1–7, 2014.

[37] N. Mor, L. Wolf, A. Polyak, and Y. Taigman. A Universal Music Translation Network. pages 1–10.

[38] M. Morise, F. Yokomori, and K. Ozawa. WORLD: a vocoder-based high-quality speech synthesis system for real-time applications. *IEICE TRANSACTIONS on Information and Systems*, 99(7):1877–1884, 2016.

[39] E. Nachmani, A. Polyak, Y. Taigman, and L. Wolf. Fitting New Speakers Based on a Short Untranscribed Sample. 2018.

[40] G. Perarnau, J. van de Weijer, B. Raducanu, and J. M. Álvarez. Invertible Conditional GANs for image editing. (Figure 1):1–9, 2016.

[41] W. Ping, K. Peng, A. Gibiansky, S. O. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller. Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning. 2017.

[42] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu. Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions. 2017.

[43] R. Skerry-Ryan, E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark, and R. A. Saurous. Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron. 2018.

[44] R. J. Skerry-Ryan, E. Battenberg, Y. Xiao, Y. Wang, D. Stanton, J. Shor, R. J. Weiss, R. Clark, and R. A. Saurous. Towards end-to-end prosody transfer for expressive speech synthesis with tacotron. *CoRR*, abs/1803.09047, 2018.

[45] J. Sotelo, S. Mehri, K. Kumar, J. F. Santos, K. Kastner, and A. Courville. Char2wav: End-to-end speech synthesis. 2017.

[46] D. Stanton, Y. Wang, and R. J. Skerry-Ryan. Predicting expressive speaking style from text in end-to-end speech synthesis. *CoRR*, abs/1808.01410, 2018.

[47] T.-t.-s. Synthesis, J. Yamagishi, T. Nose, H. Zen, Z.-h. Ling, and T. Toda. Robust Speaker-Adaptive HMM-Based. 17(6):1208–1230, 2009.

[48] Y. Taigman, L. Wolf, A. Polyak, and E. Nachmani. VoiceLoop: Voice Fitting and Synthesis via a Phonological Loop. pages 1–12, 2017.

[49] A. Tamamori, T. Hayashi, K. Kobayashi, K. Takeda, and T. Toda. Speaker-dependent WaveNet vocoder. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2017-Augus:1118–1122, 2017.

[50] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. pages 1–15, 2016.

[51] A. van den Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. van den Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, N. Casagrande, D. Grewe, S. Noury, S. Dieleman, E. Elsen, N. Kalchbrenner, H. Zen, A. Graves, H. King, T. Walters, D. Belov, and D. Hassabis. Parallel WaveNet: Fast High-Fidelity Speech Synthesis. 2017.

[52] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural Discrete Representation Learning. (Nips), 2017.

[53] V. Wan, Y. Agiomyrgiannakis, H. Silen, and J. Vit. Google's next-generation real-time unit-selection synthesizer using sequence-to-sequence LSTM-based autoencoders. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2017-Augus:1143–1147, 2017.

[54] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, and Q. Le. Tacotron: Towards end-To-end speech synthesis. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2017-Augus:4006–4010, 2017.

[55] Y. Wang, R. J. Skerry-Ryan, Y. Xiao, D. Stanton, J. Shor, E. Battenberg, R. Clark, and R. A. Saurous. Uncovering latent style factors for expressive speech synthesis. *CoRR*, abs/1711.00520, 2017.

[56] Y. Wang, D. Stanton, Y. Zhang, R. Skerry-Ryan, E. Battenberg, J. Shor, Y. Xiao, F. Ren, Y. Jia, and R. A. Saurous. Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis. 1, 2018.

[57] Y. Wang, D. Stanton, Y. Zhang, R. J. Skerry-Ryan, E. Battenberg, J. Shor, Y. Xiao, F. Ren, Y. Jia, and R. A. Saurous. Style tokens: Unsupervised style modeling, control and transfer in end-to-end speech synthesis. *CoRR*, abs/1803.09017, 2018.

[58] Z. Wu, O. Watts, and S. King. Merlin : An Open Source Neural Network Speech Synthesis System. *9th ISCA Speech Synthesis Workshop (SSW9)*, pages 218–223, 2016.

[59] J. Yamagishi, T. Kobayashi, Y. Nakano, K. Ogata, and J. Isogai. Analysis of speaker adaptation algorithms for HMM-based speech synthesis and a constrained SMAPLR adaptation algorithm. *IEEE Transactions on Audio, Speech and Language Processing*, 17(1):66–83, 2009.

[60] Y. Zhang, W. Chan, and N. Jaitly. Very Deep Convolutional Networks for End-to-End Speech Recognition. 2016.