University of London
Goldsmiths College
Department of Computing

One-Shot Neural Audio Synthesis

Leon Fedden

Submitted in part fulfilment of the requirements for the degree of
Masters in Creative Computing of the University of London and
the Diploma of Goldsmiths College, 2018

Abstract

Audio synthesis is a low-level engineering challenge that spans many domains and has many real-world uses. Successfully generating audio using neural networks at a high fidelity in real-time is a non-trivial problem, and is the one this thesis sets out to tackle. Because we work with data in a one-shot manner, the iterative development loop is shortened, enabling easier experimentation. Six neural network architectures are comparatively evaluated against one another, empirically assessing the efficacy of learning using different audio representations and the pitfalls of each approach. We also explore audio style transfer and semantically visualising generative audio. The sequence to sequence model with Luong attention received the highest rating for its capacity to synthesise sound by some margin from over three-hundred and fifty volunteers, and we successfully model a complex valued signal using this architecture. Both of these approaches generated approximately 23 seconds of audio in 4 seconds, with a sample rate of 44100 Hz and bit depth of 64 bits. If we discard phases, and just model magnitudes, this is quicker still; here, Local Weighted Sums is the optimal algorithm given requirements of real-time synthesis. Multiple musical artists have expressed interest in using the techniques laid out in this thesis as an approach in their practice. These techniques also have conceivably broader implications for the domain of speech synthesis.

Acknowledgements

What an adventure these four years at Goldsmiths have been. London has truly been more than I ever thought it would be, and I feel I am a much happier person than I would have been, had I had not come here. I would like to express my gratitude to the following people for enabling my journey:

- Professor Mick Grierson, who has been instrumental in pushing me in the right directions and keeping me on track for a few years now, which is honestly no mean feat. He has been compassionate and understanding when I've been stressed and surely difficult to deal with, and has helped me improve in both programmatic and academic ability. He also sent me to South Korea which was the first time I left the European Union, something I'll never forget. Thank you Mick.

- The entirety of the Goldsmiths computing faculty, who as a tight knit community helped a bored boy who many would have said shouldn't be a programmer be exactly that. Thank you for helping me to enjoy learning again!

- My local support network, my girlfriend Ella who sacrificed her time in my final year to give me the time I needed to meet my deadlines, and didn't let me procrastinate too much by watching back to back episodes of Have I Got News For You. Also to my close friend Becky who also strived to care for me even when I was a difficult housemate. To all of my friends who let me put myself and my degree before them.

- Sorcha, Matthew, Rachel - the three whom I see so little yet receive so much support from. Thank you to all of you for getting me this far, I love you all.

Dedication

For my parents.

'I've seen the Pokemon movie, which is probably the worst movie ever made on any subject ever.'

Ian Hislop, from Have I Got News For You.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation and Objectives

The principal problem tackled in this thesis is the topic of audio synthesis. In more specific terms, can complicated audio signals be modelled with professional studio-quality fidelity, in real-time capable speeds? Each of these requirements are easy to do by themselves, but the combination can join to become an insurmountable task. Next, consider the expert domain knowledge required to model such sounds at such fidelity and speed. It is immediately apparent that a considerable body of knowledge must be drawn from to develop sounds by hand for any particular discrete class of sound while satisfying the requirements as mentioned earlier, let alone for a wide array of sounds. Thus, another problem to contemplate is the issue of creating complex sounds with as little domain expertise required as possible. Finally, consider that there is little guarantee that the vast majority of potential users will have access to vast stores of high-quality datasets, whilst also having the computational resources to run large programs on these datasets for long periods of time. Can compelling sounds be made at a hobbyist level? Satisfying all of these requirements has been the focal and driving point of this thesis.

Sound synthesis is highly interesting because it is a fundamentally low-level engineering challenge that spans many industries and mediums. Of course, the entirety of the genre of electronic music has elements of sound design and synthesis built into its roots, but it is important to

note other mediums such as cinema, games and other multimedia all use sound as an integral component in the creation of new media. The rise of assistive and communicative technologies also make use of sound design; we not only can talk to our computers, but they can talk back to us. Speech synthesis shares a vast number of the issues previously mentioned, and progress for musical modelling can well inform those of text to speech systems.

What are the current approaches for sound synthesis applied in the field, concerning the previously laid out requirements of speech, fidelity, complexity, computability and low barrier to entry? In short, none. Handcrafted approaches are too complex and often are not able to model a diverse range of sounds outside their target range of noises. Phoneme-based concatenative speech synthesis has shown promise, as well as Statistical Parametric Synthesis methods based on hidden Markov models, though these techniques have not been applied in particularly compelling ways directly to signals outside of speech. In general, neural networks have shown the most promise: they have a consistent inference time and can model complex and high dimensional signals; computability can be achieved for some models provided the user has a consumer-level graphics processing unit; and the barrier to entry is low once the model is defined as the user can simply provide audio examples of how they want their synthesis to sound, and the neural network will learn the rest. Now is an exciting time; we reside in an avant-garde period of neural sound synthesis as researchers strive to bring the technology to a point that can be widely deployed and used in varying audio areas.

One clear issue for investigation is the fidelity of the sound produced by the neural networks. To date, there are no successfully acclaimed neural networks attempting to produce studio quality audio with a sample rate of 44100 Hz and a bit depth of at least 16 bits. This thesis aims to explore the possibility of neurally synthesising audio at this quality and examines the difficulties of doing so. Another key point of inquiry focuses on finding the best neural architectures for the synthesis of audio. There are a dizzying array of neural components one can choose from, and we cover a small subsection of these in the models used in this thesis. Finally, can one user train and perform inference in a reasonable timeframe? The inference must be real-time and the training must not take weeks on a small supercomputer.

1.2 Contributions

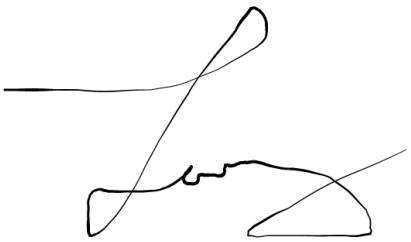
The contributions this thesis makes to the field are as following. A series six of neural architectures are described, the code made available, and the architectures empirically reviewed by both the author and a few hundred volunteers on their capacity to model the original sounds. The networks are real-time and run at a sample rate of 44100 Hz with a bit depth of 64 bits. The sequence to sequence recurrent neural network with attention is found to be the most performant model.

A comparison of phase estimation methods is carried out, whereby Local Weighted Sums is offered as the only viable solution concerning quality and computation time. The sequence to sequence model is also extended to directly model phases, which negates the need for a phase estimation method. The code and details are provided, and the efficacy of such an approach is reviewed.

Audio style transfer is implemented and reviewed, with essential modifications made to focus on only audio texture synthesis and determinism and control. The suitability of this approach is subsequently explored. Finally, a dimensionality reduction on the generated audio is performed, and an interactive web application is built and detailed to help visualise and highlight the preferential hyperparameters needed for optimal audio synthesis.

1.3 Statement of Originality

I, Leon Fedden of Goldsmiths College, University of London being a candidate for the Creative Computing MSci, hereby declare that this thesis and the work described in it are my work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.



Leon Fedden

18th May 2018

Chapter 2

Background

This chapter reviews the building blocks for neural audio synthesis; types of neural networks; types of synthesis; and audio representations; all of which are reviewed within the maximum possible scope afforded to this thesis. A brief history of synthesis and computer music is offered, after which the basic theory of neural networks, audio on computers and other ancillary topics for the main body of this thesis is covered.

2.1 The Problems Of Synthesis

The definition of synthesis is the combination of elements that form a connected whole, and, in the context of audio, a synthesiser is a musical instrument that is capable of generating a wide array of sounds by combining signals of various frequencies. On the topic of computer music, early pioneer Max Mathews wrote that there are two principal problems that one must consider when tackling sound synthesis [2].

The first of these problems is the magnitude of data required to specify a pressure function for the speakers; the volume of data needed necessitates a fast program to supply the data. This issue has been solved in principle thanks to Moore's law [3]; we now have high-speed processors that are capable of computing traditional digital synthesis programs at very high

sample rates. However, the thrust of this thesis is on the topic of neural audio synthesis, which differs from traditional digital synthesis. In the reviewed literature, there are various neural architectures, some of which are fast enough for real-time inference that meets the demands of the high sample rate pressure function, and some that are not. Whether or not the neural network is quick enough is determined from the computational operations that comprise it, and the hardware that it is run on.

The second problem that Mathews raises regarding tackling sound synthesis is the need for a simple language that can describe complex sequences of sounds. In computers, audio is represented as numbers that measure a given soundwave's pressure at a given moment in time. This is an ideal representation; because sounds in the audible hearing range are bounded both in bandwidth and dynamic range, the representation of a sound as a stream of numbers is a completely generalised mode of audio synthesis; any sound can be created given enough time to infer the correct values [4]. Whilst any sound can be represented on a computer in this format, manually tuning each value to produce a target sound is infeasable to do in real-time and thus meet the requirements of the first problem Mathews posed.

However, practical solutions that solve the first problem of supplying enough data quickly enough whilst representing a range of complex sounds do exist. Max Mathew's solution for a language to model complex sounds was a set of building blocks that modularised and optimised music on computers called MUSIC [5]. What if one could specify what they want in a data driven manner, to both simplify the role of the sound designer, and to discover new sounds that would be hard to specify in mathematical terms or to sample in real life? The objective of this thesis is to steer away from the traditional handcrafted approach and to instead study methods in which humans instruct their machines to imitate what they like, using machine learning. This thesis will explore these learning systems that can be used to model complex sounds with varying degrees of quality, speed and complexity.

2.2 Audio From Electricity

In this section, a summary of significant and pioneering efforts are documented that describe the act of synthesising audio, through electronic and electromechanical means.

It is challenging to identify the earliest synthesiser; early synthesisers and electronic musical instruments are difficult to differentiate. In 1876, invented by Elisha Gray, one of the earliest synthesisers was the "Musical Telegraph", a picture of which can be seen in figure 2.1. This was during a period where only one message could be transferred over a wire at one time via the telegraph. Western Union were offering up to $1,000,000 (around $22,000,000 at the time of writing when adjusted for inflation [6]) to the inventor that could make their wires carry multiple messages simultaneously. Considering the "Musical Telegraph" as a stepping stone towards multiple telegraphy, Gray tuned some single-tone transmitters to a different note on a musical scale, and thereby invented the oscillator [7].

Later in 1897, Thaddeus Cahill invented the Telharmonium, which created mixtures of frequencies by summing dozens of tone generators to produce additive tone complexes. Frequency mixtures are the essence of additive synthesis, and the power of additive synthesis lies in the fact that it is theoretically possible to closely reproduce any complex waveform with a set of elementary waveforms [5].

Figure 2.1: The "Musical Telegraph". Invented by Elisha Gray to transmit musical tones through telegraph wire.

Another important mode of synthesis is subtractive synthesis. While there is not a shred of evidence to support this, there is a quote commonly attributed to Michelangelo; upon being asked how he carved David out of a block of stone, Michelangelo said: "I just cut away everything that doesn't look like David". Michelangelo's misattributed quote captures the essence of what subtractive synthesis is; frequently used in conjunction with ad-

ditive synthesis, subtractive synthesis revolves around the attenuation of a signal, often rich with harmonics, to alter the timbre of the sound [8]. The attenuation of the sound is customarily achieved with filters, although other methods are available through careful modelling [9]. In Germany, a very early subtractive synthesiser was developed in 1936 and named the Warbo Formant Organ. It featured two formant filters and an envelope generator [10], and subsequently, nearly all analog synthesisers were subtractive in design.

Twelve years later, in 1948, Alan Turing would be the first documented person to generate musical notes using a computer, at the Computing Laboratory of the University of Manchester [11]. Later, in 1957, at the revered Bell Telephone Laboratories, Max Mathews explored synthesising sound from first principles using simple waveforms generated on a computer. These experiments created MUSIC-I. Although not the first computer music program, MUSIC-I was the first program widely accepted by the musical research community, and the first program to generate sound from standardised PCM samples [12]. The subsequent series of MUSIC-N computer music software is a precursor to common computer music frameworks that modularise the synthesis components, such as, but not limited to; Csound [13]; SuperColider [14]; Pure Data [15]; Maximilian [16].

In the late 1960s, John Chowning developed the digital implementation of the frequency modulation algorithm [17]. While frequency modulation was feasible on analog synths that used analog oscillators, analog frequency modulation was rarely implemented due to pitch instability. After a near decade of prototypical and experimental digital frequency modulation synthesisers, the Yamaha DX7 was released in 1983 and subsequently became ubiquitous over the next decade [18].

As the decade progressed, there was a strong trend towards digital synthesis [19], and synthesisers began making use of techniques such as sample-based synthesis and physical modelling. Sample-based synthesis describes a form of audio synthesis where the waveforms are constructed from sampled sounds or instruments [20]. As memory costs decreased over time, it became more feasible to multisample; to make more recordings of the target instrument at different pitches or velocities, rather than artificially modifying the single sampled wave. Physical modelling is

a form of audio synthesis that realistically approximates, through careful use of equations and algorithms, the physical source of a sound [21]. With physical modelling synthesis, often there will be a minimal number of parameters to provide the user control over the synthesised sound, for example, a string's base-material or pluck strength [22].

In 1982, the United Kingdom Music Union negatively received a tour by Barry Manilow, which featured synthesiser players instead of a orchestra like that of his previous tour. Feeling that this precedent raised the issue of the replacement of musicians, the Music Union sought to ban synthesisers on the merit they regarded them as a threat to the session musicians that they represented [23]. A neural network with a single hidden layer of a finite size, as a universal function approximator, is capable of approximating any continuous function including the pressure function of a stringed instrument [24]. Furthermore, a more complex network is capable of modelling most types of instruments simultaneously in the same model instance. This is why neural networks are so exciting, with little expert knowledge there is the potential to create complex synthetic sounds that far outstrip those that shocked the Music Union in the 1980s.

2.3 Representations Of Audio

When neural networks are trained on audio, there are many ways that an continuous analog audio signal can be digitally represented on a computer. Various different types of features can be derived from these representations that have been useful in statistically modelling acoustic signals.

A careful choice of representation for audio can have a massive impact on the ability of a generative network to learn to produce perceptually appropriate sound; data representations can make a problem much easier or much harder to solve [25]. It is imperative for the practitioner to pick a suitable feature that articulates what it is that matters in the problem at hand. Using features that are perceptually relevant can help guide the model to produce what is expected. Taking cues from our physiology, or more precisely, our auditory system, might help to make the generative problem easier for the deep learning model.

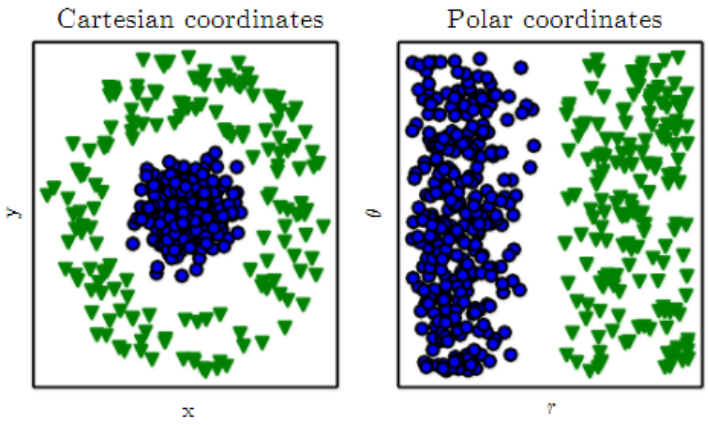


Figure 2.2: Cartesian and polar coordinates of the same data points. Whilst both graphs use the same data, the classification problem becomes linearly separable by changing the basis. Reproduced from Goodfellow et al, Deep Learning book, 2016.

2.3.1 Time Representation With Pulse-Code Modulation

Pulse code modulation (PCM) is a method of digitally representing analog signals, and is the manner in which audio is stored and streamed on computers today. PCM describes the reduction of a continuous-time signal to a discrete-time signal and represents a stream of quantised amplitude values stored at a uniform rate, sampled over a period of time [26]. Typical PCM file formats are wave, aiff or pcm. The bit-depth and the sampling rate determine the quality of a PCM representation of the original signal. The bit depth is the number of bits afforded to each sample and denotes the reconstructed amplitudes resolution [27].

The sampling rate, sometimes expressed in kiloHertz (kHz), describes the number of values sampled per second and determines the maximum frequency that the audio signal can represent through the Nyquist theorem. The Nyquist theorem states when converting an analog signal to a digital signal, half the value of the sampling rate is the maximum frequency that the audio signal can reproduce [28]. The Nyquist theorem is of particular importance when reviewing a generative neural network's synthesised output. Many popular neural networks have been optimised to be performant on PCM files with a 16kHz sampling rate, meaning that the maximum frequency the generative material can incorporate is 8kHz, well below the standard of professional CD-quality audio, which is sampled at 44.1kHz and has a Nyquist limit of 22.05kHz [29].

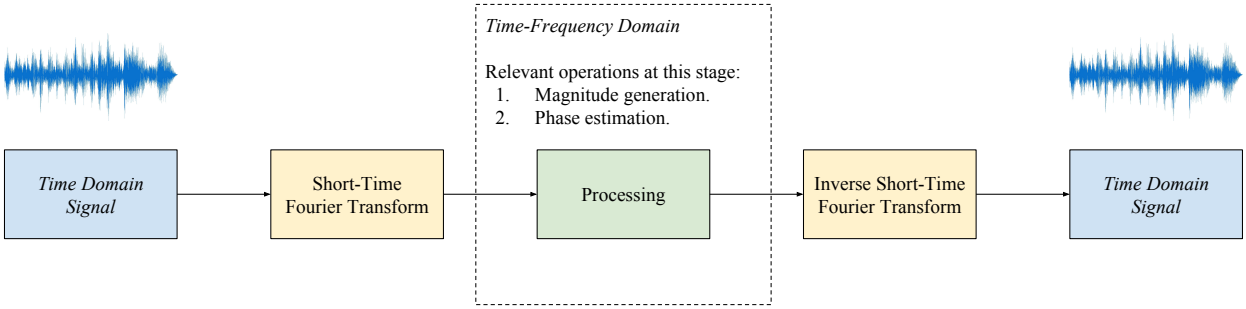


Figure 2.3: Converting a time-domain signal to the time-frequency domain and back again.

2.3.2 Time-Frequency Representation With The Fourier Transform

PCM signal streams operate in the time domain, which means that the samples represent amplitudes over time. Alternatively, a PCM signal can be viewed over both the time and frequency domains by making use of the Fourier transform. The Fourier transform takes a time domain signal that measures the change of signal over time, and, moves it into the frequency domain that subsequently measures the amount of signal that resides in every frequency band over the range of frequencies afforded by the Nyquist theorem [30]. Repeatedly using the Fourier transform on windows of the time domain signal results in the time-frequency representation, and is called the short-time Fourier transform (STFT). This sequence of Fourier frames is a complex valued signal, and the absolute value of the complex signal denotes the magnitude of frequency in each respective frequency bin. The complex argument indicates the amount of phase offset of a sinusoidal wave in that frequency bin [31].

The human ear has other uses than a canvas for jewellery; chiefly the ear is in effect, a biological hardware for a Fourier decomposition. The cochlea hosts the basilar membrane, and upon receiving vibrations from the inner ear instigated by a sound, the hair cells on this membrane vibrate at different frequencies in a non-linear fashion. This is illustrated in figure 2.4. In effect, the human ear sorts the incoming sound's frequency energy as a function of frequency over surface, from high frequencies at the entrance to low frequencies at the apex. Each fibre in the auditory nerve carries information about a narrow range of frequencies, to be further processed eventually at a higher level in the brain [32, 33]. Here, there is biological inspiration to provide computational neural networks with an audio representation of Fourier frames, or

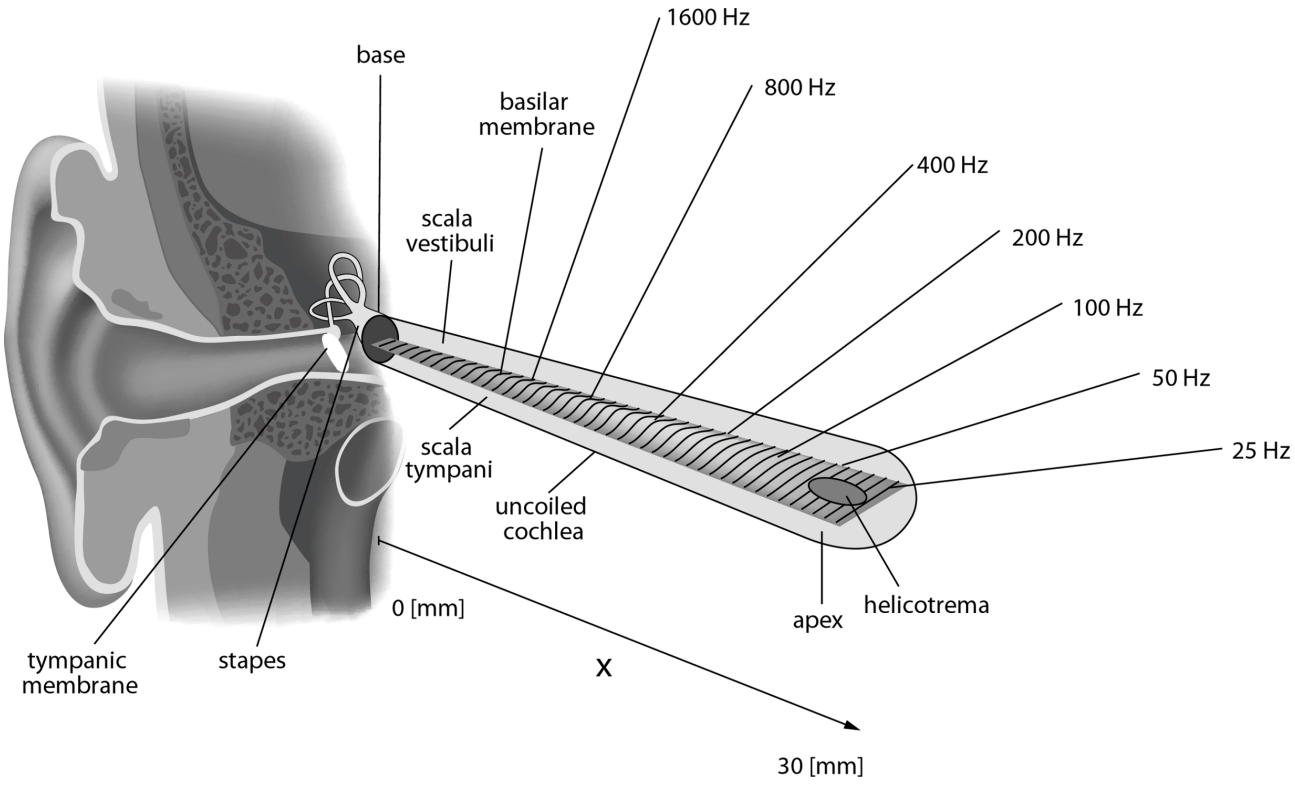


Figure 2.4: A diagram of the cochlea and the basilar membrane. The basilar membrane essentially performs a Fourier analysis over its surface, breaking down complex sounds into their component frequencies and thus serves as biological inspiration as the preferred representation for neural networks. Reproduced from Wikimedia.

frequencies over time, rather than a sound pressure wave, or amplitude over time.

2.4 Phase Estimation Methods

Assuming the representation of audio to model is in the time-frequency domain, the signal in question will be a complex valued one. This means both magnitudes and phases will need to be modelled to operate in this domain, which effectively requires two neural networks or some other trick because no popular deep learning library supports complex gradients at the time of writing.

Alternatively, and potentially more effectively, the phases can be estimated using an algorithm, and thus only the magnitudes need be modelled by the neural network. Several such phase estimation algorithms were used in this thesis, and are briefly described in this section.

2.4.1 Griffin-Lim

Griffin-Lim is an iterative and non-realtime algorithm to estimate a set of phases from a set of magnitudes [34].

The goal of the Griffin-Lim algorithm is to estimate some phases for a set of magnitudes. The phases are initialised to be completely random. The magnitudes are combined with the phases to become a complex signal, on which the Inverse Short-Time Fourier Transform is computed, creating a time-domain signal. Next, we compute the Short-Time Fourier Transform and discard the magnitudes. The new phases are applied to the target magnitudes, and the process is repeated for the number of specified iterations.

Because the process is non-convex, the Griffin-Lim is highly dependant on the initialisation and will produce different results on different runs depending on the initialisation [35]. As mentioned, this is almost always a non-realtime process due to the high amounts of iteration required to generate audio of better quality.

2.4.2 Local Weighted Sums

Local Weighted Sums (LWS) is a fast phase estimation algorithm. It works differently to Griffin-Lim; LWS works by replacing the costly iterative STFTs to ISTFTs, which is the bottleneck, by using some light computations instead on a select few bins in the time-frequency domain. The authors made use of the sparseness of the magnitudes bins, and limited the updates to phase bins that have a significant corresponding amount of magnitude; the other lower magnitude bins do not perceptually add artifacts when combined with incorrect phase values. The bins are updated one by one, which then informs the subsequent bin's updates [36].

2.5 Deep Learning For The Audio Domain

In this section, we review the use and progression of more complex generative neural networks' architectures, and list their uses in the audio domain. Optimisation of neural networks is not covered in this background; for a comprehensive tutorial see [37], however, adversarial learning will briefly be covered.

2.5.1 Multi-Layer Perceptron Networks

Multi-layer perceptron networks (MLPs), also called feedforward neural networks, are standard deep learning models, see Figure 2.5. The goal of a multi-layer perceptron network is to approximate a function, $f*$. In the case of audio, a network might do a regression on some previous frames of audio, using a mapping function f, that takes an input $\boldsymbol{x}$ and maps it to the next frame of audio $\boldsymbol{y}$. We would denote this as below, where $\boldsymbol{\theta}$ are parameters (network weights) that are optimised for the best function approximation [37].

$$\boldsymbol{y} = f(\boldsymbol{x}; \boldsymbol{\theta}) \tag{2.1}$$

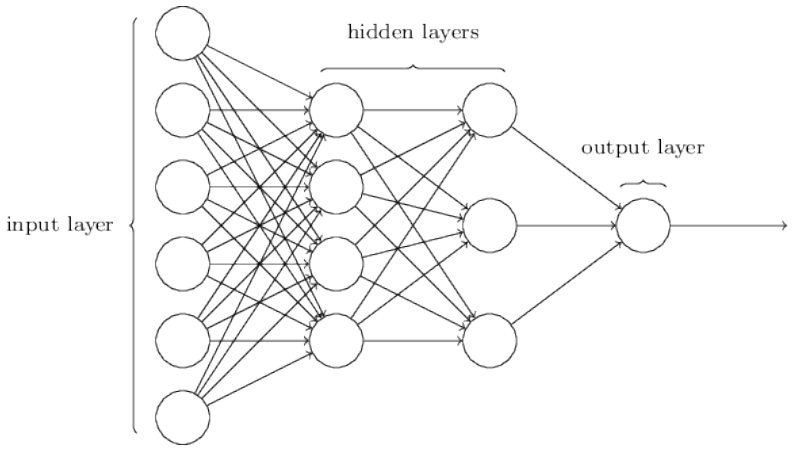


Figure 2.5: A diagram of a Multilayer Perceptron Network. Note the two hidden layers, and the input and output layers. The dimensions of all of the layers can be of arbitrary sizes. Figure courtesy of Hastie and Tibshirani, Statistical Learning and Data Mining IV, 2017.

Composition

Feedforward neural networks are composed of fully connected layers. A MLP has an input layer, an arbitrary depth of hidden layers, and an output layer. The input and output layers should be determined from the dimensionality of the problem [38]. For example, a famous dataset is the MNIST handwriting dataset, which is comprised of images that have a width and height of 28 pixels, and one colour channel [39]. This means a MLP must have 786 input neurons (28^2 flattened) to process this dataset. The hidden layers can have an arbitrary width of neurons, and the output layer must have a number of neurons that matches the learning task that it must be optimised for.

Every neuron in the given hidden layer is connected to each successive layer's neurons [40], and performs a multiplication of the neuron's weights $\boldsymbol{W}$ on the neuron's input $\boldsymbol{x}$ before adding the bias $\boldsymbol{b}$, such that the neuron's output is: $\boldsymbol{y} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$.

Non-Linearities

An integral component of MLPs, and neural networks in general, is the activation function, or non-linearity. *The purpose of the activation function is to introduce non-linearity into the neural network.* A neural network without activation functions can only ever be a linear function, which

would make a lot of functions impossible to approximate to any degree of success; see the logical exclusive-OR (XOR) function for a toy example of a function that cannot be approximated with a linear model [41]. Indeed, a lot of the interesting functions that one might want to approximate are non-linear in nature.

Given a simple MLP with two input neurons, one hidden layer with two neurons, and a single output neuron; with no activation functions on any neuron we can denote it as the following equations. In equation 2.2, the output $\boldsymbol{y}$ equals some MLP approximating a function f, after being passed $\boldsymbol{x}$ as an input. In the subsequent equations it is analysed whether adding multiple linear layers produces a non-linear composition.

$$\boldsymbol{y} = f(\boldsymbol{x}) \tag{2.2}$$

$$f(\boldsymbol{x}) = \boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2} \tag{2.3}$$

$$f(\boldsymbol{x}) = \boldsymbol{W}_2\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{W}_2\boldsymbol{b_1} + \boldsymbol{b_2} \tag{2.4}$$

We can see that the resultant equation 2.4 is still a linear equation thanks to the commutativity of the matrix multiplications in equation 2.3, and thus would not be possible to solve the XOR problem.

There are however a number of important non-linear functions that enable the composition of layers in a neural network to become a non-linear process, and thus model and approximate much harder functions than a linear model could. Unfortunately, due to the number of possible activation functions, only the ones most pertinent to the neural architectures used in this thesis can be covered. The common activation functions that are described are also graphed in figure 2.6.

The sigmoidal function, sometimes known as the logistic function, is a well known activation function, and produces its characteristic curve within the range from zero to one.

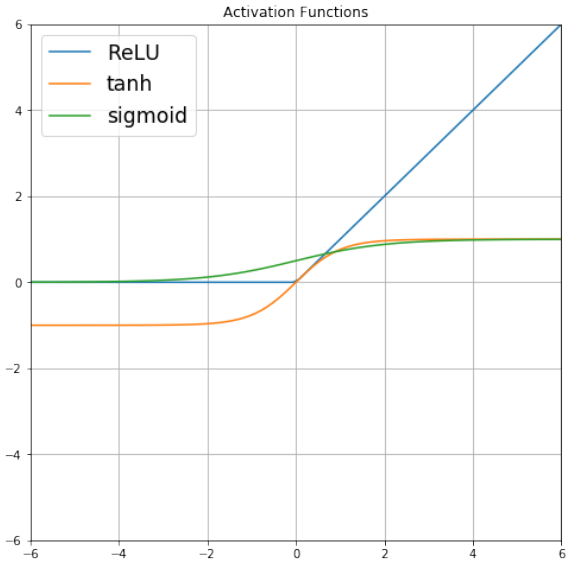


Figure 2.6: A diagram of common activation functions used in neural networks.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

The hyperbolic tangent function, or tanh, has a similar curve to the characteristic one seen in the sigmoidal function, but is instead zero centred; the tanh has a range of minus one to one.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.6}$$

The rectified linear unit (ReLU) function simply returns whatever is greater, the value passed to the ReLU function or zero. ReLU is comprised of two linear functions that combine to create a non-linear effect, the zeroing of any less than zero value; and the linear returning of any greater than zero value.

$$f(x) = \max\{0, x\} \tag{2.7}$$

The sigmoidal-based activation functions have lost traction in the literature. This is due to any function with a sigmoidal curve's tendency to have its gradients diminish as the absolute value of the input increases, see figure 2.7. This diminishing of gradients is slightly less present

in tanh functions due to the steepness of their curve, and another benefit of tanh in many applications of neural networks is that they are zero centred, which means the gradient can be positive or negative, unlike a sigmoid [42].

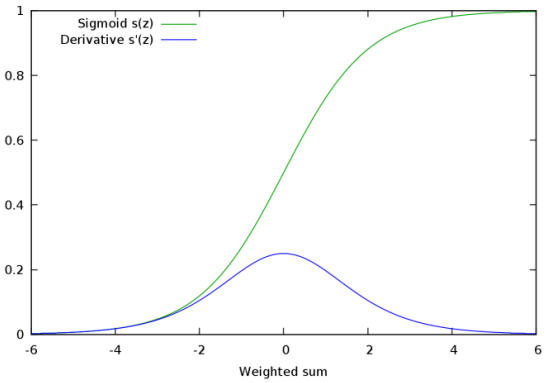


Figure 2.7: The sigmoid curve and it's derivative (gradient). Note the gradient vanishes as the absolute value of the sigmoid function increases.

ReLUs are not party to this vanishing of gradients, and also provide a benefit of sparsity which arises when the input to the ReLU is less than zero. The more units that exist like this in the neural network, the more sparse the neural network is [43]. Sparsity is a desirable feature of a neural network; sparsity induces information disentangling, which means the network's outputs will be invariant to small perturbations to the input [43]. However, sigmoids are still in wide use in gated recurrent neural networks such as long short term memory cells to control the gating connections [44], for more, see the section on recurrent neural networks.

Applications

MLPs are not ideal architectural choices for the problem of generative audio, and as such, the literature is not rife with examples of generative audio feedforward neural networks. That being said, the dense layers that comprise an MLP which otherwise are known as fully connected layers due to every neuron in the current layer connecting to every neuron in the next layer, are often featured in the more advanced neural networks that feature later in this section. Whilst not directly synthesising audio, MLPs also have been used to map control parameters to audio synthesis parameters in real time [45, 46]. On the topic of synthesis, a Mixture Density

Network (an MLP with an output layer double the size to parameterise a gaussian for each output neuron) was used for speech synthesis [47]. Sarroff and Casey created an autoencoder, a neural network with a diabolo shaped topology, that was trained to reconstruct musical magnitude frames. This can be understood as an early musical neural synthesiser [48].

Whilst there are not a lot of examples of direct synthesis with the simple feedforward neural network, within the literature there is a strong narrative of modelling acoustic signals. Examples include: an acoustic mapping with MLPs between two voice signals, transforming frames of frequency content to the target voice signal [49, 50, 51]; gender classification that used Mel Frequency Cepstral Coefficient (MFCC) frames as features [52]; automatically determining the environment through auditory features such as MFCCs [53]; audio event classification [54, 55]; speech enhancement and noise cancellation [56, 57]; and musical onset detection [58].

2.5.2 Recurrent Neural Networks

In this section we review the key aspects of the recurrent neural network (RNN), including some of the architectural choices that have helped projects become state of the art applications for their domain. First a motivation is given as to why RNNs are useful, next, various architectures are explored and motivated, and we end by reviewing popular applications of the RNN within the context of audio synthesis.

Sequence Modelling

All of the neural networks listed from here on out in this chapter implement some form of weight sharing. *Recurrent neural networks have loops in them, and allow information to persist, see figure 2.8.* Often we need to model sequences $\boldsymbol{X}$ of varying sizes, such that there is a sequence of values $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \ldots, \boldsymbol{x}^{(n)}$.

Weight sharing within neural networks can be highly beneficial, and aside from being a useful form of regularisation, it can be an advantageous prior for a model depending on the type of problem. Let us assume we are modelling a sequence of words. For example, there are sentences:

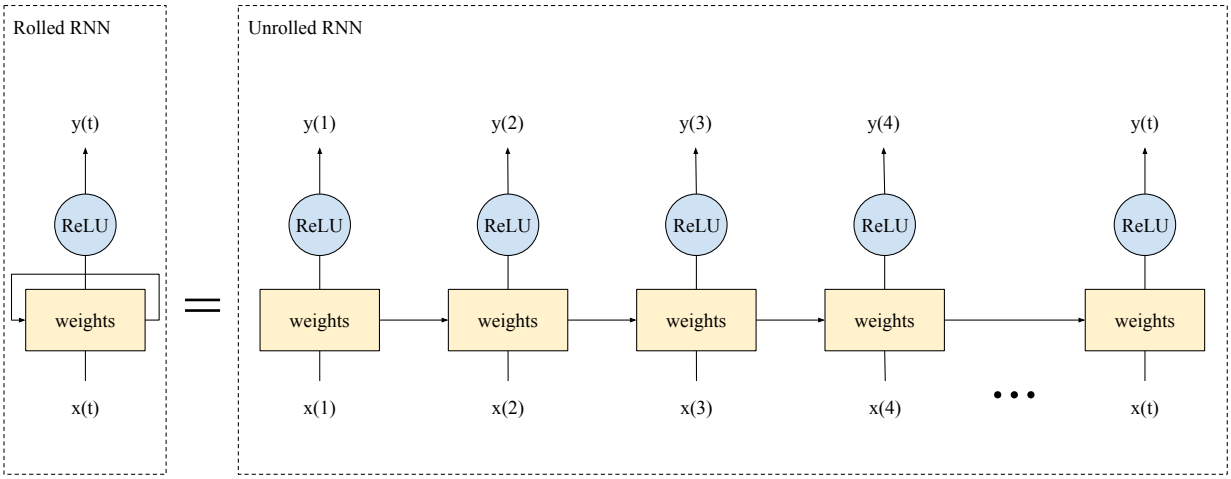


Figure 2.8: A figure of a recurrent neural network. The loop can be unrolled to reveal a sequence of compuations that both takes sequences of information and outputs sequences of information.

(1) "I got my degree at Goldsmiths." (2) "At Goldsmiths, I got my degree." Now assume that the model must infer where the author obtained their degree, and of course, the objective is to recognise that it was obtained at Goldsmiths. Given the two sentences, the important information is either the second or sixth word. Feeding this information to a traditional MLP network would firstly require a fixed sized input of six for the MLP. The MLP would also need to learn the rules to extract the correct information for each word position in the fixed size sentence. In contrast to the MLP, a recurrent neural network can share parameters for each timestep in the sequence, or in this example, each word position [37].

$$\boldsymbol{y}^{(t)} = f(\boldsymbol{x}, \boldsymbol{y}^{(t-1)}; \boldsymbol{\theta}) \tag{2.8}$$

Consider the previous equation for the MLP network, in equation 2.1. In the equation for the RNN, equation 2.8, there is now a crucial difference that at each timestep t, the RNN will feed the previous timestep's output $\boldsymbol{y}^{(t-1)}$ as an input into the current timestep $\boldsymbol{y}^{(t)}$, whilst sharing the parameters $\boldsymbol{\theta}$ across each timestep.

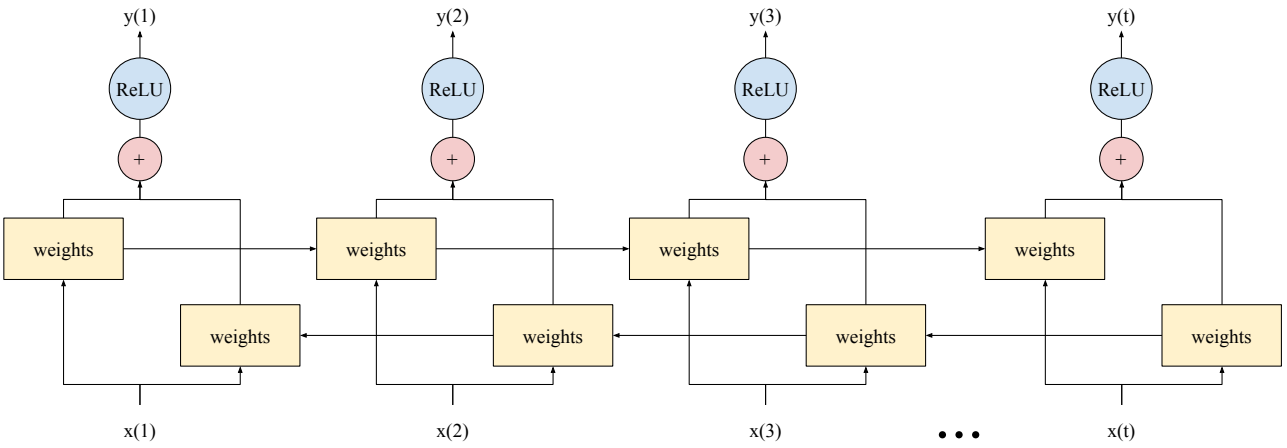


Figure 2.9: A bidirectional recurrent neural network that processes the sequence in both directions before summing the two representations and applies the non-linearity.

Bidirectionality

With the RNNs that have previously been covered, the RNNs process the input sequence in a unidirectional fashion. When producing musical sounds with neural networks, it can potentially be the case that the current note or chord depends on either the past or future notes or chords in some manner. A unidirectional RNN will incorporate the information from any previous frames of sound but discard the data from the next sounds in the sequence. By processing the sequence of audio frames in a bi-directional manner, one can double the information in the series of audio frames [59, 37]. See figure 2.9 for a simple computational graph that would do this.

$$\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \ldots, \boldsymbol{x}^{(t-2)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t)}) \tag{2.9}$$

$$\boldsymbol{y}_{\text{forward}} = f(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \ldots, \boldsymbol{x}^{(t-2)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t)}) \tag{2.10}$$

$$\boldsymbol{y}_{\text{backward}} = f(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \tag{2.11}$$

$$\boldsymbol{y} = \boldsymbol{y}_{\text{forward}} + \boldsymbol{y}_{\text{backward}} \tag{2.12}$$

Above, f is the RNN, and whilst sharing parameters it processes the sequence $\boldsymbol{X}$ in two directions, backwards and forwards, in equations following from 2.9. Subsequently, the outputs

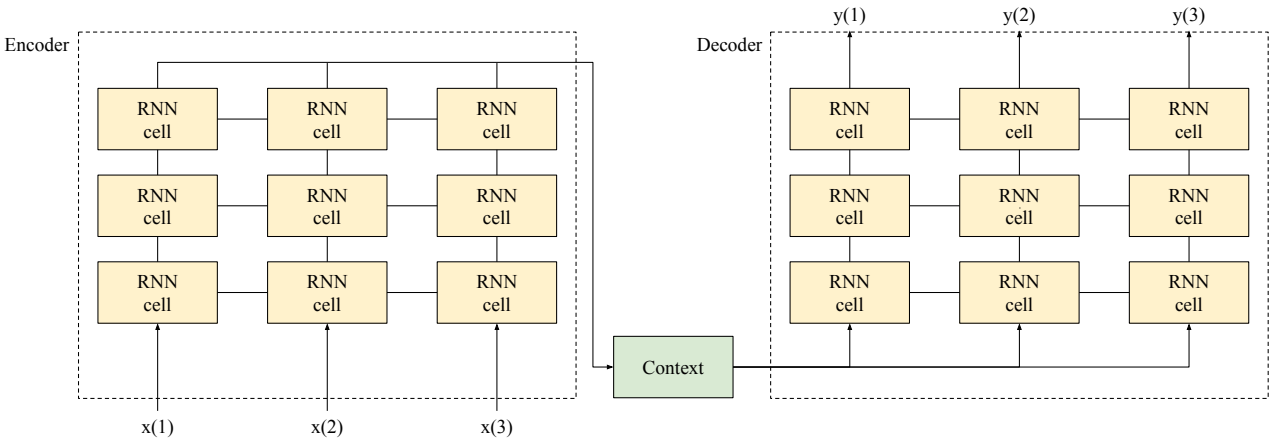


Figure 2.10: A diagram of a three-layer deep sequence to sequence recurrent neural network architecture. The encoder reads some input sequence and this representation is stored as some fixed size memory tensor called the context C. This context is then processed by the decoder and the output sequence is computed.

for both directions are linearly summed into the RNN output $\boldsymbol{y}$. Generally, in practice making a recurrent bidirectional improves results for lots of types of problems and is worth trying if one can spare the extra computations [60].

Sequence-to-Sequence Models

One can train RNNs in either a many to one or many to many fashion. A many to one representation means that one output will be produced for the sequence fed into the RNN, and a many to many representation means that an output sequence will be produced for the input sequence processed by the RNN. When mapping an input sequence to an output sequence, a sequence-to-sequence model may be useful [61]. In particular, the encoder-decoder architectures enable the input and output sequences to differ in length. Encoder-decoder models were first proposed by Cho et al. [62].

Encoder-decoder models are split into two components, the encoder and the decoder. The encoder's role is to learn a representation of the input to the RNN, known as the context C. The context C may be a vector or sequence of vectors that represents the input $\boldsymbol{X}$, and care must be taken to create a context sequence long enough to properly summarise a long sequence fed into the RNN [63]. The decoder is subsequently fed the fixed-size context sequence C and

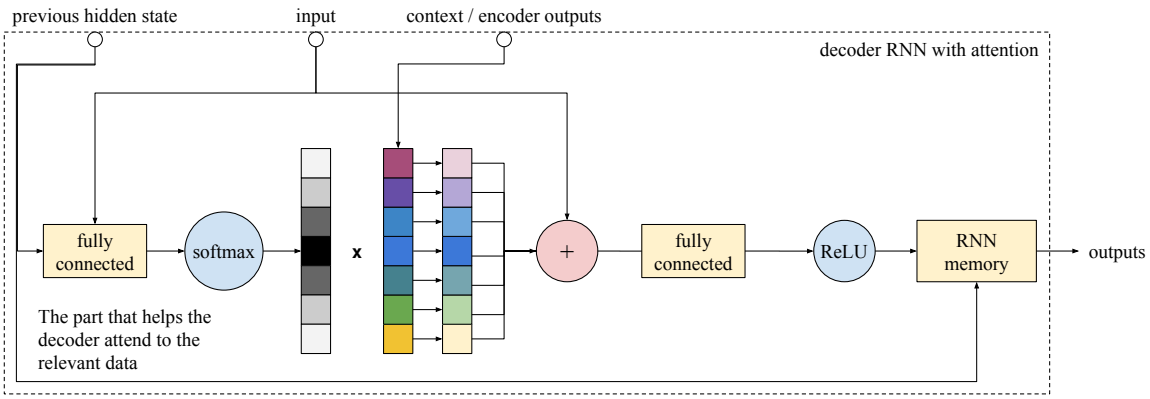


Figure 2.11: The computational graph illustrating the application of attention to the context vector, which is then passed to the RNN memory.

produces the output sequence $\boldsymbol{y}$ [37]. This encoder-decoder architecture is illustrated in figure 2.10.

Attention

One way to prevent a context vector C having too high a size that would be necessary to effectively model a long sequence of inputs to the RNN, is to learn to selectively see and attend to elements of C when generating the output sequence.

In the case of this thesis, the attention used was originally derived from Luong et al. [64], although a few changes are made. Firstly, the decoder RNN creates outputs from the raw input sequence, to help provide information about the whole input sequence at each step of the context sequence C. Attention energy is calculated from a fully-connected linear layer that takes the encoder context sequence C as inputs. This attention energy is then applied to the decoder RNN's outputs, through a matrix-matrix product, creating cues for what to attend to and what to discard. Next, the attention weighting has a softmax non-linearity applied to the attention energy's sequence.

A new context is created from a matrix product of the attention weighting and the original encoder context C. The decoder RNN's output and the new context vector are concatenated to combine the information from both vectors. The attentional concatenated vector is then used for either classification or regression by being passed through an output fully-connected layer

with the appropriate non-linearity.

Gated RNNs

Because vanilla RNNs apply the same weight matrix $\boldsymbol{W}$ at each step in the sequence, the simpler RNNs are susceptible to a difficulty that can arise in training called Long-Term Dependencies. The problem is that the gradients propagated backwards over many recurrent stages tend to vanish or occasionally explode, which respectively hinders by making it uncertain which parameters should be optimised, or damages, via unstable gradients, the optimisation. The resultant method, assuming a recurrent neural network with no input nor non-linearity, is close to an eigenvalue method called the power iteration. The following recurrence relation can describe the neural network at timestep t:

$$\boldsymbol{h}^{(t)} = (\boldsymbol{W}^t)^\top \boldsymbol{h}^{(0)} \tag{2.13}$$

Now we decompose the weight matrix $\boldsymbol{W}$ into an expression featuring only eigenvectors, $\boldsymbol{Q}$, and eigenvalues $\boldsymbol{\Lambda}$.

$$\boldsymbol{W} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top \tag{2.14}$$

Substituting the eigendecomposition into the recurrence expression in equation 2.13 then shows us that each of the eigenvalues are raised to the order of t, meaning that any eigenvalues greater than one will explode in magnitude, and any eigenvalues less than one will shrink to obscurity.

$$\boldsymbol{h}^{(t)} = \boldsymbol{Q}^\top \boldsymbol{\Lambda}^t \boldsymbol{Q} \boldsymbol{h}^{(0)} \tag{2.15}$$

A carefully designed defence against exploding or vanishing gradients comes in the form of the family of gated recurrent neural networks. The gated RNN introduces new edges into the computational graph designed to propagate gradients that neither explode or vanish. Information

is accumulated by the RNN while processing a sequence. This information is the aggregated internal state from previous timesteps of the RNN. Assuming the RNN has used the aggregated data for some purpose, and has not reached the end of the sequence, it could feasibly be prudent to discard and forget the aggregated information to more effectively process new subsequences without diluting them.

The canonical gated recurrent neural network is the Long Short-Term Memory (LSTM) network. The LSTM does not differ from the vanilla RNN concerning the dimensions of the inputs and outputs, but varies instead in its internal computational graph. LSTMs feature a forget gate that moderates its self-recurrence, which runs on an inner loop inside the LSTM cell [44]. This self-recurrence loop is separate to the outer recurrence that is already present because the network is a recurrent neural network. The forget gate determines how much of the previous internal state should be present in the current state based on the current data being passed in at the timestep in the input sequence. This feature was added to help a network process a continuous sequence that is comprised of many subsequences; allowing the LSTM to forget previous subsequences no longer relevant to the current sequence [65].

Because the forget gate is passed through a sigmoidal non-linearity and is a learnable parameter, the forget gate learns to output zero when it needs to ignore the previously accumulated information, and output one to pass prior knowledge to the current time step. It is important to note that the LSTM's default behaviour is to automatically forget, and it must learn to retain information. This is in stark contrast to a vanilla RNN, which always updates the state no matter what the input data.

Another critical gate is the output gate. Similar to the forget gate, it is a learnable sigmoidal parameter. In the case of the forget gate, the gate determines what of the previous state will be fed into the current state. The output gate acts as a scalar, and determines what the LSTM will output. The Gated Recurrent Unit (GRU) was developed to simplify the LSTM; GRUs trim the computational fat by merging the input and forget gates into a single update gate. The GRU also integrates a new gate called the reset gate which determines what of the previous timestep's state goes into the current state [66, 37].

Clipping Gradients

An initiative that tackles exploding gradients is to clip the gradients by some value should the magnitude be too high. Anything but applying an unstable parameter update is a good idea here; even taking an utterly random gradient step when the gradient explodes prevents a numerically unstable parameter configuration.

Algorithm 1 Basic pseudo-code to clip gradients by their norm

if gradient norm $\|\boldsymbol{g}\|$ is greater than theshold v **then**
    clip gradients $\boldsymbol{g} = \boldsymbol{g}\frac{v}{\|\boldsymbol{g}\|}$
end if

Pascanu et al. demonstrated that clipping the gradient by its norm before the parameter update keeps the gradient in the same direction as the original unstable one, but at a safe step size [67]. As such, this is the dominant method used in this thesis.

RNNs For Audio Synthesis

A typical application for recurrent neural networks within the audio domain is speech synthesis. Speech synthesis is particularly relevant to this thesis as it shares a majority of techniques required to model audio signals.

A four-layer recurrent neural network was found to be capable of synthesising Mandarin Chinese [68]. In 1997, a survey of wearable audio computing noted concatenative synthesis with RNNs was more successful than the tested alternatives [69]. Zen showed that a simple LSTM is more effective than traditional techniques such as hidden Markov models for acoustic modelling [70], and this sentiment was echoed in later papers that explored similar themes plus optimisations for mobile usage [71, 72]. Similar works extended these RNN architectures to facilitate multi-language and multi-speaker acoustic modelling, whilst demonstrating transfer learning to generalise to new languages with small amounts of data [73]. Similar efforts were made for Bangla, the primary language of Bangladesh [74]. Unsurprisingly, a bidirectional LSTM proved the most capable out of the combinations of unidirectional LSTM, bidirectional LSTM, unidirectional GRU and unidirectional vanilla RNN when building a speech synthesiser [75].

Turning to more recent and sophisticated solutions, sequence-to-sequence GRUs were used to transform textual characters to raw spectrogram magnitudes which were converted to PCM audio with the Griffin-Lim phase estimation algorithm [76]. The 'Tacotron' network in question was an architecture that incorporated many neural blocks; Conv1D layers; highway layers; fully-connected layers; GRUs; attention; bidirectional RNNs; and a post-processing network to improve the predicted magnitude frames. A similar sequence-to-sequence architecture reproduced some of the results for speech synthesis using a 'Tacotron' architecture [77]. Researchers extended the results from the 'Tacotron' paper by conditioning the model on emotion labels and created a parameterised emotive speech synthesiser [78].

As an extension of the popular SampleRNN that directly produces PCM sample audio, the authors modified the SampleRNN to take characters by producing a sequence-to-sequence model that converted characters into vocoder features fed into SampleRNN [79].

On the central theme of this thesis, RNNs have also frequently been used for musical audio generation. Text to speech systems usually needs some conditioning vector or label to provide the context to produce the correct phonemes or audio. These contextual values can be discarded in the case of music audio neural synthesis; only the future frames need to be predicted given the previous frames of music. More complicated systems could reincorporate this conditioning as a sophisticated form of control, but this is rarely seen in the literature.

An important recurrent neural network for audio generation was SampleRNN [80]. It features a novel architecture that digests the input PCM samples at different clock rates, and each layer conditions the successive layer. This technique massively widens the receptive field of the neural network and enables the SampleRNN to model long dependencies of audio. Because the audio samples are quantised, the model performs a classification rather than a regression with a MSE loss. Learning a representation for PCM sound is difficult, and the authors of SampleRNN are commended for sharing their code and full details of their experiments, where other authors of similar papers seemed to omit essential information on their models and sampling methods. However, the model is tested up to a 16kHz sample rate, which massively limits the high-frequency content available in the generated sound. Perhaps because of its open source nature,

SampleRNN proved a popular model in the literature; it was used to model trombones [81]; it was also used to create a math-rock album [82]. Another important RNN for audio generation in the PCM domain would be WaveRNN [83]. Key innovations include a dual softmax layer that allows the search space for the classification layer to be massively reduced, as well as a sampling method based on subscaling that enables efficient sampling at generation time.

2.5.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are neural networks that replace matrix multiplication with a mathematical convolution somewhere in the neural network.

Convolutional Versus Fully Connected Layers

In equation 2.16 the convolution operation is detailed. The idea is that we take a structured piece of data like a greyscale image $\boldsymbol{I}$, and convolve it with the kernel $\boldsymbol{K}$. Kernels are also often referred to as a feature map [37].

$$c(i,j) = (\boldsymbol{I} * \boldsymbol{K})(i,j) = \sum_m \sum_n \boldsymbol{I}_{i+m,j+n} \boldsymbol{K}_{m,n} \tag{2.16}$$

Convolutions, like recurrent neural networks, allow for arbitrarily sized input data. CNNs also bear other similarities with RNNs as they promote the sharing of parameters. This is in stark contrast to fully connected layers, where every parameter is used exactly once on a single pass, and the input size is fixed and cannot be changed mid-operation.

Sparse Connections

With a standard feedforward neural network, the idea is that each layer is connected to the successive layer; for each layer, the preceding layer's parameters multiply with each of the succeeding layer's parameters. Aside from being wasteful, this amount of parameters can

lead to overfitting. Convolutional neural networks are not fully connected in the same manner; convolutional neural networks induce sparse connectivity by making the set of parameters called the kernel smaller than the input, and this can improve the efficacy of the neural network [84].

Parameter Sharing

Because the CNN's kernels are nearly always smaller than the input, this means that the parameters in the kernel are used multiple times across the image [85]. Therefore the memory requirements for the layer are substantially lower than those of a dense layer, as is the amount of operations required to compute the layer. Applying the kernel at different locations across the width and height of the input enables the kernel to learn statistically efficient feature maps.

Equivariant Representations

To say a function is equivariant is to say that if you modify the inputs in a direction, the outputs will reflect this change delta in the same direction [86]. In this sense, CNNs are equivariant to translation [37]. This means that if the translation transformation is T, and the convolution operation is f, then f is equivariant concerning T because $f(T(x)) = T(f(x))$; this means that applying the translation T directly to the input x is comparable to applying the translation to the output of the convolution $f(x)$.

Pooling

While the convolutional operation is equivariant to translations, the pooling operation hard-codes an invariance to small amounts of translation of the input [87, 37, 88]. The pooling layer typically follows the non-linearity applied to the convolutional layer. Because of the pooling operation's invariance to translation of the input, if the input is moved a small amount, the pooling layer should return a similar value to that of if the input had not been translated.

Including pooling operations into the model increases the priority of a feature being present over a feature being in the correct place [37]. Within the context of audio, one must consider

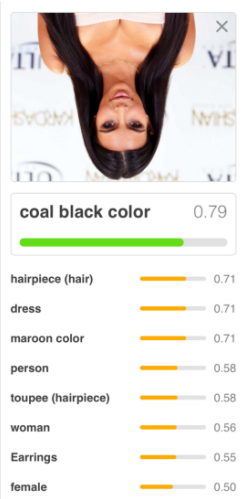


Figure 2.12: Just because convolutional neural networks are invariant to translation does not mean they are invariant to other transforms, such as rotation [1].

whether this infinitely strong prior of invariance to small translations is a useful benefit; do we need to know when a note appears in a spectrogram, or if it just appears at all?

Another useful benefit of pooling is that it enables downsampling of the parameters [89]. By making the pooling layer have fewer units than the convolutional units, the pooling layer can act as a downsampler. The downsampling will outline the responses from the neighbourhood of convolutional units and can improve the statistical, performance, and memory efficiency of the model.

Types Of Pooling Operations

There are two types of pooling operations worth noting that are present in the neural networks used in this thesis. Max pooling, and, average pooling, which are illustrated in figure 2.14. The former will extract the maximum value in each feature map, and the latter will obtain the average feature.

On Dilated Convolutions

A term thrown around frequently when discussing models such as WaveNet or its successors is a type of convolution operation called a dilated convolution. This dilated operation facilitates

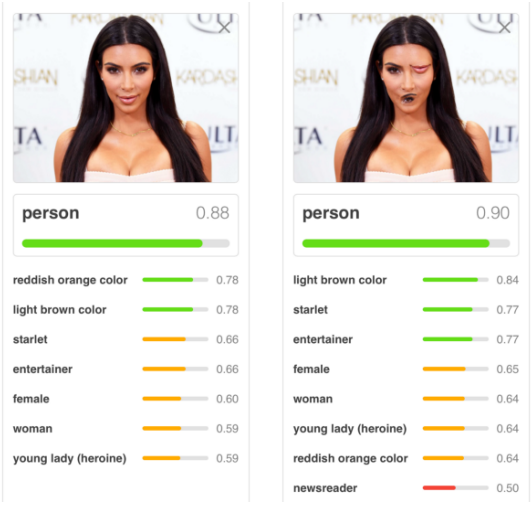


Figure 2.13: By encoding invariance to local translation, the model learns that to prioritise the feature being present rather than if it is exactly where it should be. This can lead to unexpected results [1].

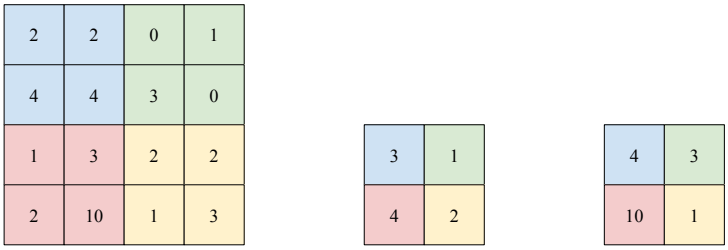


Figure 2.14: Different types of the pooling operation. Left most image are the two times two feature maps with stride of two. Middle image is average pooling. Right image is max pooling.

vast receptive fields while growing the number of parameters logarithmically [90]. It means the convolution kernels are dilated, which means to grow in size before a higher stride is applied. The growing of the kernel and the size of the stride is proportional to the dilation coefficient.

CNN Audio Applications

CNNs are a common neural architecture for modelling and synthesising audio. A popular method of applying CNNs to audio data is as a learned feature extractor or dimensionality reduction on audio. Data augmentation strategies revolving around pitch and time shifting and additive synthesis were devised to train a CNN to acoustically model sound and classify birds from audio samples. The authors also found CNNs to be the more performant architecture than bidirectional LSTMs [91]. Data augmentation is rarely documented in audio modelling papers so it was nice to see the details listed here, as it can be an essential way to increase the performance of the model and grow the dataset. Along similar lines of modelling and dimensionality reduction, an interesting audio application of CNNs was to learn a representation on PCM samples from unlabeled video to the associated audio in a multimodal, unsupervised manner. This clustering learned to minimise the KullbackLeibler divergence (how much one probability distribution differs from the other, otherwise known as the relative entropy) between CNN features from image frames from video, and CNN features learned from raw audio samples [92].

But what of generative audio representations? An interesting form of generative modelling of audio signals is the learned upscaling of PCM data; the task for the neural network is to reconstruct high-quality audio from downsampled lower-quality audio. The model in question makes use of an auto-encoder shape with stacked residual connections from the encoder to the decoder [93].

A well known CNN for audio generation is WaveNet. It popularised the approach for directly modelling on raw waveforms by making use of successive stacks of dilated convolutions. Due to the size of the model and the sampling strategy, it is non-real time and requires a lot of data to suitably model audio [94]. The non-real-time of this model was a disappointing factor given its

relatively low sample-rate. A follow-up paper from the same authors demonstrated they could train a similar WaveNet architecture which outputs multiple samples in parallel. The training was done by minimising the probability distribution from the new fast architecture against the slower original one by using the KullbackLeibler divergence. However, this then meant that the new model was limited to speech output only, as it required conditioning vectors to instruct the model in the voice sounds it should make [95]. The WaveNet architecture proved popular in the literature: Baidu built a smaller model with fewer parameters for speech synthesis [96]; the WaveNet architecture was extended to an autoencoder and introduced a vast new dataset of one-shot instruments in the neural audio synthesis paper by Google Magenta [97]; and a WaveNet architecture was also used to enhance the quality of degraded audio [98].

2.5.4 Generative Adversarial Networks

In the previous sections, we covered common neural network architectures. Generative Adversarial Networks (GANs) describes a setting that pits two neural networks against one another competitively, in a game theoretic manner.

A GAN is comprised of two neural networks, the generator g, and the discriminator d. The generator's job is to produce generative samples $\boldsymbol{x}$ from some noise vector $\boldsymbol{z}$.

$$\boldsymbol{x} = g(\boldsymbol{z}) \tag{2.17}$$

The discriminator's role is to review both real examples from the training set, and fake examples produced by the generator, and correctly classify all of the provided samples on whether they are fake or real pieces of data. This is formulated as a zero-sum game, which means the generator's success is directly balanced by the failure of the discriminator, and vice versa. The models will ideally converge, and the discriminator will output 0.5 for every example it sees; it cannot distinguish the difference between the real data and the fake data synthesised by the generator. At this stage, one can discard the discriminator and solely use the generator.

A set of improvements, with theoretical groundings and empirical evidence to back it up on vanilla GAN training, comprises the Wasserstein GAN. One change is the employment of a critic rather than a discriminator. The critic is trained to converge every time the generator should be updated and doesn't classify fake examples, but now reviews as a critic and returns quality of data scores. Another key difference is the loss function, which is now derived from the Wasserstein distance. The critic does not classify fake from real but instead estimates the Wasserstein metric between the generated data's probability distribution and the actual data's probability distribution. The critic's weights are also clamped to a small fixed range after each gradient update [99].

2.6 Other

There have been a number of other ancillary and arbitrary topics explored in the duration of working on this thesis. The notable ones will be covered in this section.

2.6.1 Style Transfer

Style transfer is about separating content and style. The content of some media can then be recombined with a new style, breeding media in the style of some arbitrary creator. Style transfer began with the work by Gatys et al. which sought to impose famous painting styles on arbitrary photos.

Let us assume the media for style transfer is images. Two key images are required for style transfer, the content image and the style image. The content image will determine the focal points and general content of the generated image. The style image will provide the textural qualities of the image, such as the impasto painting technique found in Van Gogh images such 'The Starry Night'. See figure 2.15 for some examples of the algorithm's output.

The general gist of style transfer is as follows. The first step is to pass both images through a CNN. A loss is constructed from representations of both the content and style images. The

Figure 2.15: Style transfer, where the content and style images are combined using the style transfer technique to produce novel images. Reproduced from Google style transfer tutorial.

content loss is the Euclidean distance between a layer's representation of both the input image and the content image. In this case, the input image may be either the content image or random noise. The style loss is a bit more complicated. For the feature map in the CNN, a gram matrix is constructed from the convolutional kernel, denoting the covariance of the feature map and which features tend to activate with one another while remaining apathetic to the position of such features. By computing the Gram matrix of both the input image and the style image, and subsequently using these in a Euclidean loss, this provides the style loss. The two losses are minimised by modifying the input image, which creates the typical artistic style transfer photo. This concept can be extended to audio, both in the frequency and time domain, and is explored later in this thesis.

2.6.2 Generative Material Dimensionality Reduction

The training of many generative models produces a lot of generative material. The various combinations of hyperparameters multiply the amount of content generated during hyperparameter searches. The simplest way to review the content is to listen to it all, and this was, of course, carried out. However, it can be difficult to plot this data and see the hyperparameters

responsible for the samples; to some degree, this problem of understanding some generated content's causality was explored through dimensionality reduction.

One such dimensionality method is t-SNE, standing for t-distributed stochastic neighbour embedding. The algorithm takes a set of high dimensional data points, such as audio, and creates a similar set of lower dimensional data points with the desired target number of dimensions. Next, t-SNE coverts the two sets of data points to joint probabilities and minimises the Kullback-Leibler distance metric between the two sets of data points [**?**]. Because t-SNE is both non-convex and stochastic, results can vary with different runs, and also Euclidean distance can be unreliable; data from a single Gaussian distribution can appear clustered in a t-SNE dimensionality reduction [100].

This algorithm, necessary methodology, and a resultant web application for generative audio material will be reviewed later over the next chapters.

2.6.3 The One-Shot Learning Problem

Depending on the sources that are read, the reader will get different definitions of one-shot learning. The phrase one-shot learning has been used to describe our ability as humans to understand information from few training examples [101]. In statistics, it is common knowledge that estimating a given number of parameters requires a many-fold increase in training examples [102]. Learning information from a handful of examples is a useful trait, and in this thesis, we build neural networks that train in one sound, often short in length. This speeds up training of the networks, often meaning the network takes approximately only an hour to converge, and enables a much tighter feedback and iterative development loop. The size of the neural network, as well as the audio representation, will have large impacts on the learnability of the scarce data.

Chapter 3

Implementation

In this chapter the neural network architectures used in this thesis are detailed, drawing on the prerequisite knowledge explained in the previous chapter to describe the operations and functionality. Any neural networks detailed, except anything concerning style transfer, model the probability of a sequence of audio signal, in either a PCM, complex or magnitude representation. If we generalise the audio signal to a sequence $\mathbf{x} = \{x_1, x_2, x_3, \ldots, x_{T-2}, x_{T-1}, x_T\}$ comprised of T elements, we can say that the neural networks model the probabilities of each element conditioned on all previous elements from the signal.

$$p(\mathbf{x}) = \prod_{t=0}^{T} p(x_t | x_1, x_2, x_3, \ldots, x_{t-1}) \tag{3.1}$$

Each signal step x_t is conditioned on all previous timesteps.

3.1 PCM Neural Networks

3.1.1 WaveNet

WaveNet is the one PCM based neural network obtained as a reference architecture. Due to the original network not being published, a community-driven derivation of WaveNet was used

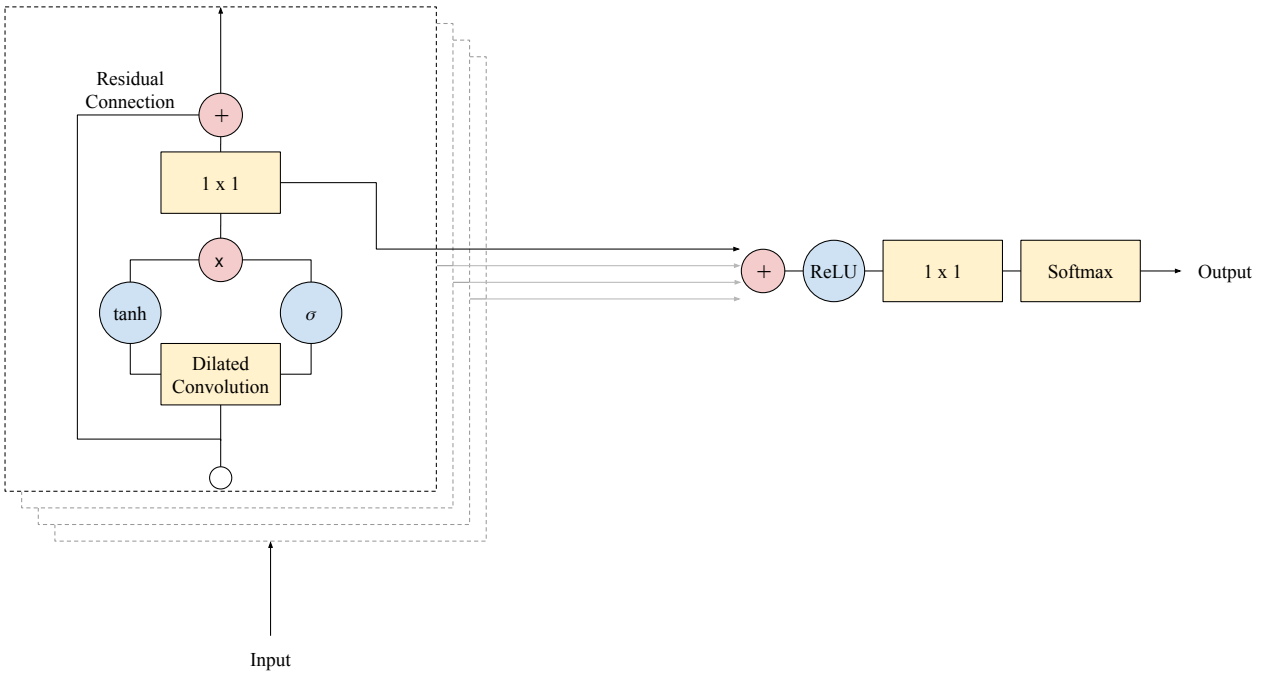


Figure 3.1: The WaveNet architecture. Note the residual connection between the stacks of convolutional layers and the skip connections that sum the outputs together to produce the final output.

[103]. The network directly takes quantised PCM audio and predicts the next quantised audio sample conditionally given the previous audio samples. In the task of speech synthesis, the model would also be conditioned on additional contextual parameters to control the speech generated.

The network is comprised of stacks of dilated convolutions, of which the operations are further described in the background chapter. The outputs from each of these dilated convolutional layers are joined and processed by some fully connected layers, before being passed to a softmax non-linearity function help to classify the output as a categorical distribution. There wasn't a lot of architectural optimisation to be made due to the long time and significant amount of computational resources required to optimise a single WaveNet model. Otherwise, more time would have been devoted to improving the output. The model was optimised with the Adam first order optimisation method [104].

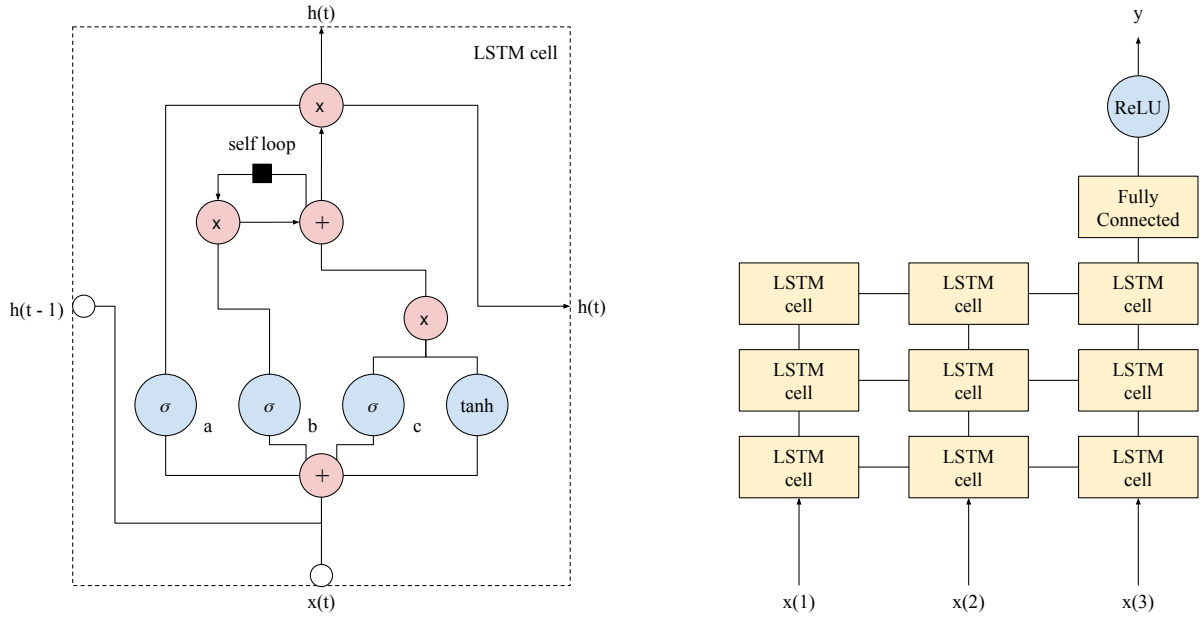


Figure 3.2: The many to one RNN architecture. Note LSTM gates a, b and c. These are respectively the output, forget and input gate. The black square is a self loop that takes a time step to compute. This shows the previous time steps state being added to the current state assuming the forget gate tends towards one.

3.2 Magnitude Neural Networks

A much larger range of neural network architectures was developed over the scope of this Thesis. The evolution of the models was gradual, and essentially a human performed a guided random search to find better neural network generative audio. Thus, the neural networks that learned a magnitude representation are shown in chronological order. For inference, all of the magnitude only networks used either random initialisation, Griffin-Lim, or Local Weighted Sums after the magnitudes had been generated to estimate the phases.

3.2.1 Many To One Recurrent Neural Network

A range of permutations across various recurrent neural network hyper-parameters was tried in the effort of generating compelling audio. The recurring theme was a multilayer gated recurrent neural network with a series of fully connected layers to resize the output to the correct size. The fully connected layers featured L2 regularisation which penalises the layer on the magnitude of

its weights through an additional term in the neural network's loss function. L2 regularisation was not however present in the RNN layers; regularisation in recurrent networks can cause information to die exponentially fast and wipes out long-term memory traces [105].

Variational recurrent dropout was used in the Keras RNN models, which has both theoretically and empirically been demonstrated to be more performant than naive dropout which varies at each timestep [106]. Many aspects of the neural network were modified in the pursuit of improved generative audio: the amount of successive recurrent layers; the size of the memory in the recurrent layers; the amount and size of fully connected layers; the non-linearities used within each neural layer; the recurrent cell used (LSTM or GRU); the amount of directions in which the sequence was processed (bi-directionality versus uni-directionality).

RMSProp, Adam and Stochastic Gradient Descent were all tried to optimise the networks. In general Adam and RMSProp enabled quicker learning, and the latter was settled for all recurrent networks; it empirically provided better results faster.

3.2.2 Convolution To Many To One Recurrent Neural Network

The previous many to one recurrent architectures were extended by prepending a convolutional feature extractor to the recurrent neural network. Each convolutional layer was followed by a non-linearity such the Rectified Linear Unit (ReLU), and subsequently, it was followed by max pooling. The depth of layers, kernel size, and amount of filters were all manually tuned. The learned feature's maps were reduced from a tensor of rank four to rank three, by joining the width and height channels.

3.2.3 Many To Many Recurrent Neural Network

The many to many recurrent neural network (sequence to sequence or seq2seq) is a multilayer neural network whereby two recurrent neural networks work in tandem to learn a representation between an input sequence and an output sequence [107]. The idea is to encode an input

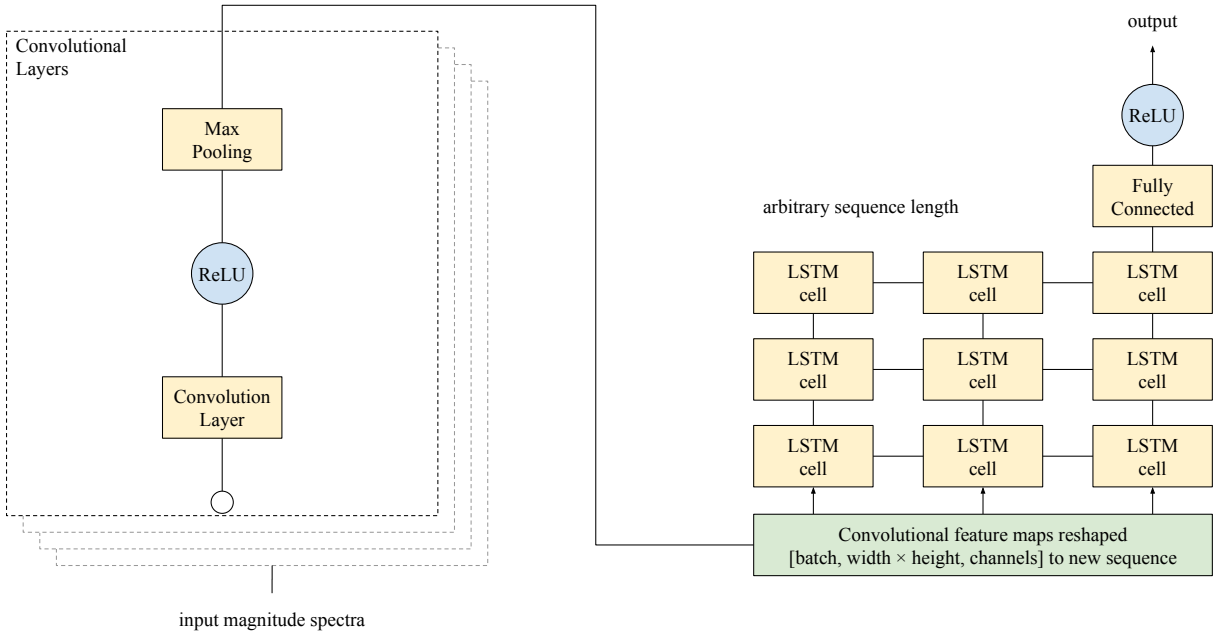


Figure 3.3: The many to one CNN to RNN architecture. A convolutional feature extractor is reshaped to be processed by a many to one relationship RNN.

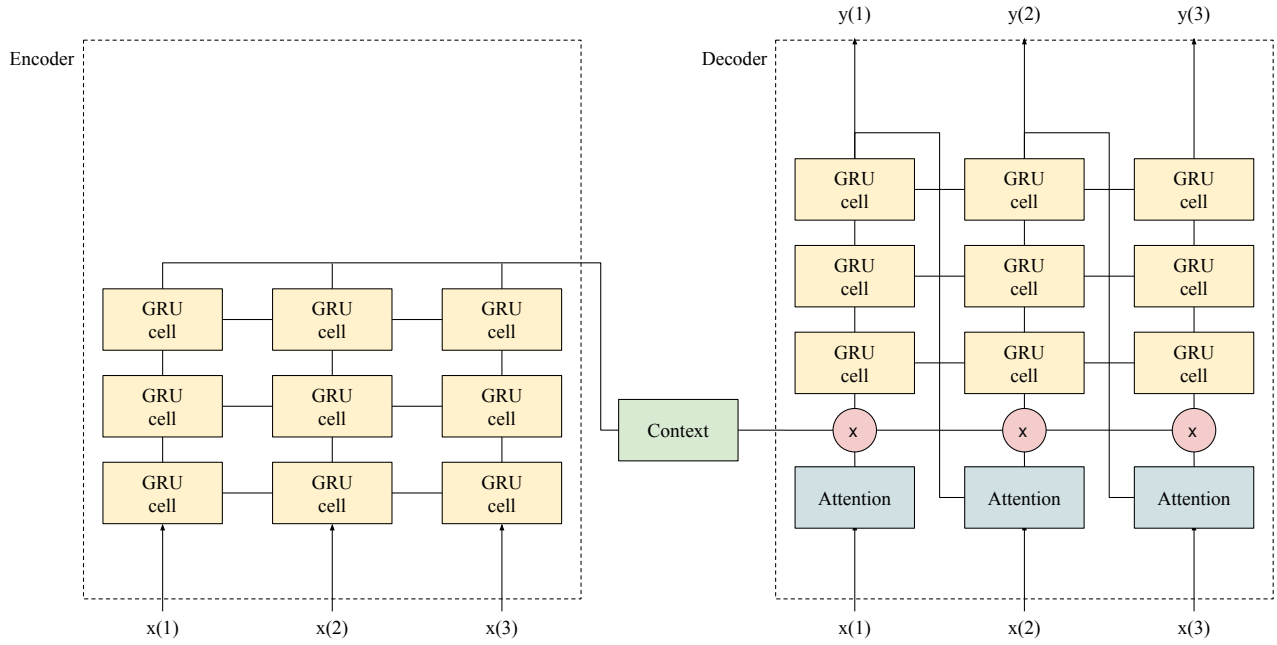


Figure 3.4: The many to may RNN architecture, otherwise known as a sequence to sequence RNN or seq2seq.

sequence to a fixed size representation using an RNN called the encoder. Another RNN, the decoder, takes this fixed sized representation and generates the new sequence. Luong attention [64] was used to enable the decoder to concentrate on the most relevant parts of the output of the encoder.

The encoder and decoder both are GRUs, and tended to be two or greater layers in depth each. Bidirectionality was applied where the graphics memory was available, as it empirically improved results. The input sequence and the output sequence were always the same lengths; typical sequences had a length of 100, 150 and 200. In other words, the neural network's task was simply to take an input sequence of magnitude frames and predict the subsequent amount of magnitude frames, and in this was trained in a supervised manner by using MSE as a loss. Teacher forcing didn't help the network learn better representations than without, so was dropped early on in the project [108].

The inference was relatively simple; a random sequence was selected to start the neural network off, and subsequently, the outputs of the network were fed in as inputs iteratively to produce the generated audio.

3.2.4 Many To Many Recurrent Neural Network GAN

Adversarial training was introduced to try to improve the representation that the neural network learned. From now on, the encoder, attention and decoder are conceptually wrapped up together and referred to as the generator component of the Generative Adversarial Network. The discriminator component of the GAN was more simple; it was a many to one recurrent network that outputted a singular value constrained by a sigmoidal activation function; the output would be one if it thought the input sequence was real, or zero if it thought the input sequence was fake.

Taking cues from the successes of the well known conditional GAN pix2pix, the model combines a mean squared error (L2) loss with binary cross entropy (BCE) [109]. The binary cross entropy rewards the discriminator on how many it fake generated sequences it correctly classified, and

how many fake sequences were incorrectly classified for the discriminator. Even though pix2pix made use of an L1 distance metric in the original paper, L2 was used in place of L1 as this produced better results; L1 merely returned noise. The generators loss l, for the nth term in the input batch, can be described as the following, as the combination of BCE and L2 losses:

$$l_n = (-w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right]) \cdot \alpha + (x_n - y_n)^2 \tag{3.2}$$

Here α is simply a scalar to control the influence of the BCE loss term. The discriminator's loss is similar but removes the L2 term and the α scalar value. Upon inference, the discriminator is not required, and the generator used in much the same way as the normal sequence to sequence model.

3.2.5 Many To Many Recurrent Neural Network Wasserstein GAN

Wasserstein adversarial training makes a few crucial differences to the RNN GAN architecture as mentioned in the previous subsection. Firstly, the discriminator becomes a critic. More concretely, instead of outputting a value squashed between zero and one using a sigmoid activation function, the output is a scalar whose value directly corresponds to the quality of the fake material. The Wasserstein changes made to the vanilla GAN algorithm were empirically shown to improve the stability of learning and remove problems such as mode collapse, as well as providing meaningful learning losses [110].

The training procedure is modified too; prior to training each batch on the generator, the critic will be trained to convergence to highly score real data and offer a low score for fake data. During this stage, the generator's learnable parameters are all locked and cannot be updated. The critic will next have its weights clipped between a small range, such as $[-0.01, 0.01]$. The next stage of the training procedure is to optimise the generator, and as such, the generator's learnable parameters are unlocked, but the critic's parameters are now locked. The L2 distance loss is again applied like last time, as well as the learned score made by the critic when processing the sampled magnitude frames from the generator.

Algorithm 2 Wasserstein GAN RNN training. Here α_c is the critics learning rate, α_g is the generators learning rate, and α_{mse} is the weight for the MSE loss.

while θ_g has not converged **do**

  for $t = 0, \ldots, N$ **do**

    $\boldsymbol{x}_{\text{real}} \leftarrow$ sample m sized batch of target sequences from dataset

    $\boldsymbol{x}_{\text{fake}} \leftarrow$ sample m sized batch of input sequences from dataset for generator f_g

    $l_{\text{real}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} f_c(x_{\text{real}}^{(i)})$ compute the loss for the real magnitudes

    $l_{\text{fake}} \leftarrow -\frac{1}{m} \sum_{i=1}^{m} f_c(f_g(x_{\text{fake}}^{(i)}))$ compute the loss for the synthesised magnitudes

    $g_c \leftarrow \nabla(l_{\text{real}} + l_{\text{fake}})$ compute the gradient

    $\theta_c \leftarrow \theta_c + \alpha_c \cdot \text{RMSProp}(\theta_c, g_c)$ update the critic

    $\theta_c \leftarrow \text{clip}(\theta_c, -0.01, 0.01)$ clip the critic's parameters to range

  end for

  $\boldsymbol{x} \leftarrow$ sample m sized batch of input sequences from dataset for generator f_g

  $\boldsymbol{y} \leftarrow$ sample m sized batch of target sequences from dataset corresponding to $\boldsymbol{x}$

  $\boldsymbol{y}' \leftarrow f_g(\boldsymbol{x})$ predict future sequences from inputs

  $l_{\text{em}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} f_c(y'^{(i)})$ compute the critic score for generators synthesised sequences

  $l_{\text{mse}} \leftarrow \alpha_{\text{mse}} \cdot \frac{1}{m} \sum_{i=1}^{m} (y'^{(i)} - y^{(i)})$ compute target sequences MSE loss

  $g_g \leftarrow \nabla(l_{\text{em}} + l_{\text{mse}})$ compute the generators gradient

  $\theta_g \leftarrow \theta_g + \alpha_g \cdot \text{RMSProp}(\theta_g, g_g)$ update the generator

end while

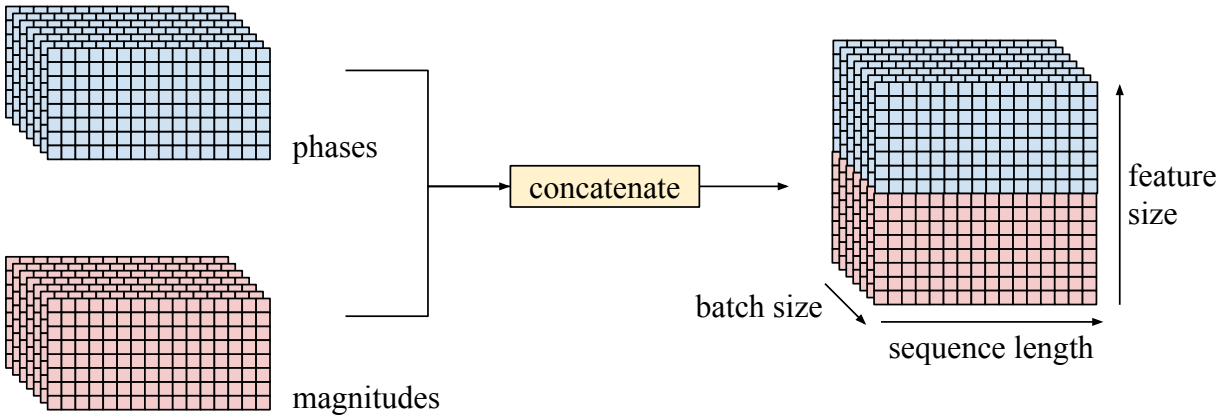


Figure 3.5: How phases and magnitudes are concatenated togther. Note that the phases are converted to real values and normalised to the range minus one to one.

3.3 Complex Audio Neural Networks

3.3.1 Many To Many Recurrent Neural Network

The complex sequence to sequence model is incredibly similar to the first many to many RNN mentioned in the previous subsection, which featured no adversarial training. A crucial difference is that the network must both take and produce sequences of double the size; the model now intakes and produces both magnitudes and phases. Complex gradients and operations are not widely supported in deep learning libraries at the time of writing, although this is a changing landscape. Because of the lack of complex automatic differentiation support, the network itself is more a collection of hacks than a legitimate, complex neural network.

The network simply concatenates the phases to the magnitudes in the third axis of the tensor, where the first axis is the batch, the second is the sequence length, and the third is the feature depth. This is iilustrated in figure 3.5. This makes the representation harder to learn than a magnitude only representation, additionally so because upon plotting the phase spectra, they are considerably more noisy and chaotic than the generally structured magnitude spectra. The memory requirements also rise, so in certain cases, the batch size must be reduced to fit bidirectional many layers RNNs on one GPU. Additionally, the normalisation of the phase spectra is covered in the next chapter in experiment three.

3.4 Audio Style Transfer

Audio style transfer moves the technique of imagery style transfer [111] to the medium of sound. The content and style audio samples are selected, and a noisy signal is optimised so that its hierarchical features obtained from layers of a convolutional neural network match that of the content and style audio features derived from the same hierarchical neural network.

Parag's method of using a discrete Fourier transform as the first step in the computational graph comparing the hierarchical convolutional neural network is used in this thesis [112]. Phases and magnitudes are derived from the real and imaginary components and passed through convolutional layers to get the style gram matrices $\boldsymbol{G}$ required for the loss. The Gramian matrices are essentially the covariances of the convolutional feature maps $\boldsymbol{F}$, and are computed as:

$$G_{ij} = \frac{1}{M} \sum_{m=1}^{M} F_{im} F_{jm} \tag{3.3}$$

A problem with previous audio style transfer works is the domination of either the style audio or content audio in the resultant sound. Because the output can sound quite confusing, in the style of Ustyuzhaninov et al. the content sound is dropped, and noise is purely optimised to produce similar Gram matrices to the style sound. The noise that is optimised is deterministically created from simplex noise.

3.5 Dimension Reduction On Generative Outputs

A dimensionality reduction process is applied to the audio generated by various neural networks to help visualise the efficacy of each combination of hyperparameters used.

The entire dataset of synthesised audio and associated hyperparameters are firstly loaded into memory. Next, a preliminary dimensionality reduction is performed by computing Mel Frequency Cepstral Coefficients (MFCCs) from the audio dataset. MFCCs are generally considered

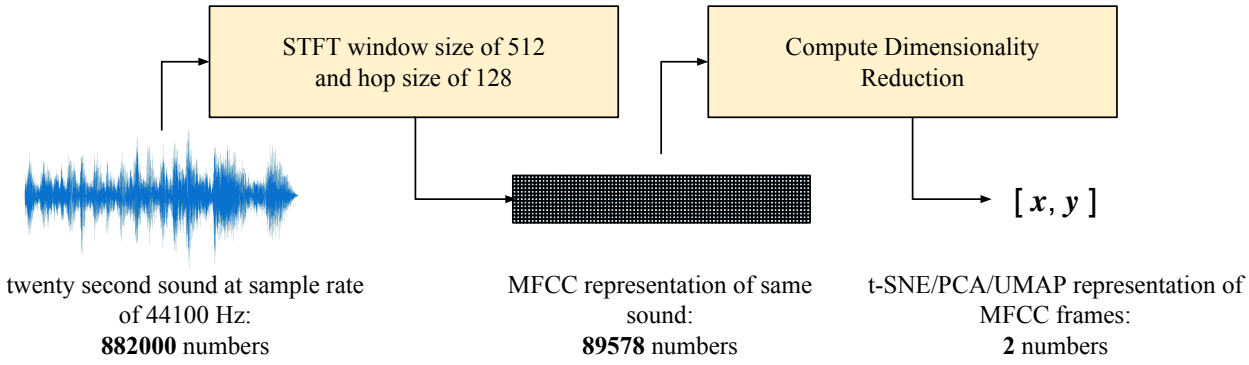


Figure 3.6: A simplified dimensionality reduction process for one twenty second audio file.

good pitch-invariant timbral features [113] and this is essentially the application of an early dimensionality reduction. A reduction in size can be observed for the following reason; one would pass a power of two audio samples, such as 2048, to the Fourier transform that precedes the MFCC computation. The computation would subsequently return a smaller fixed size set of cepstral coefficients, such as 13 values, to describe the sound, which is a notable reduction in the quantity of numbers.

Once sequences of MFCCs have been produced for each datapoint, each datapoint is further reduced to a concatenative fixed-size feature vector that enables different lengthed synthesised sounds to be further processed by dimensionality reduction algorithms. This concatenative feature vector is the combination of the mean, standard deviation, and the mean first-order difference of the MFCC features. The mean MFCC provides us with the average MFCC frame, the deviation will tell us how large the spread of the distribution of MFCC frames is, and the mean first-order difference tells us how much the features tend to change over time. Once these feature vectors are computed, they are further reduced to two-dimensional coordinates by the dimensionality reduction algorithms UMAP, PCA and t-SNE, before being graphed in an interactive web application created with three.js [114]. We divulge more about this application in experiment four in the next chapter.

3.6 Deep Learning Libraries

This project was carried out over an extended period, in some cases leveraging code that was from multiple authors and many years old. Thus, whilst the predominant base language was frequently Python for most projects, a vast array of Deep Learning libraries were used, and this project wouldn't have been possible without them. TensorFlow is a deep learning package developed and maintained by Google, and WaveNet and the many to one recurrent neural networks made use of it [115]. The many to one recurrent neural networks also made use of higher-level APIs such as Keras [116] and TFLearn [117]. The many to many recurrent neural networks made use of PyTorch [118], a framework with a focus on imperative programming that is maintained by Facebook.

Chapter 4

Evaluation

4.1 Experiment 1: Generative Audio Comparisons

The various neural architectures described in the implementation chapter were trained on the same sounds to test the capacity for the neural networks to model small amounts of data in a one-shot manner.

4.1.1 Experiment Conditions

To ascertain the capacity to model percussive sounds, as well as more complex vocal pitching sounds, the two audio sources that were used were a drum beat and a woman singing. The same sounds were used for all the neural architectures, and the neural networks were subsequently trained to convergence to model these sounds. A web application was constructed, and a group of over three hundred and fifty volunteers were presented with a series of pairs of audio clips. Each pair of sounds was comprised of the original input audio to the given neural network and the resultant generative material after the network was trained to convergence. The users were asked for each pair to determine on a five-point scale how well the generative sound captured the original audio's characteristics and content.

4.1.2 Results

Firstly, in this subsection, we discuss the results from the volunteers tasked with determining the generative networks' capacities to capture the characteristics of the sound. We follow by reviewing the sounds generated by the architectures, and the author's thoughts are articulated regarding why these sounds are as they are.

Volunteer Reviews

More than three hundred and fifty volunteers were asked to rate the synthesised sounds on how well they captured the original sounds' characteristics. The clear winner was the sequence to sequence RNN without adversarial training of any sort, scoring predominantly fives and fours, and the only model to recieve a high proportion of fives. The next best model was the Wasserstein seq2seq RNN and then the GAN seq2seq RNN. These both scored around two to three as a median rating, with the GAN's mean a little lower than the Wasserstein RNN. At the lowest end of the rating scale was the simple RNNs and Wavenet.

Architecture	drums reconstruction mean rating	vocals reconstruction mean rating
many to one RNN	2.35	1.66
many to one CNN to RNN	2.25	1.22
seq2seq	**4.34**	**4.33**
seq2seq GAN	3.25	2.12
seq2seq Wasserstein GAN	3.47	2.97
WaveNet	1.05	1.07

Table 4.1: The mean scores on how well each method modelled the characteristics of the original sound.

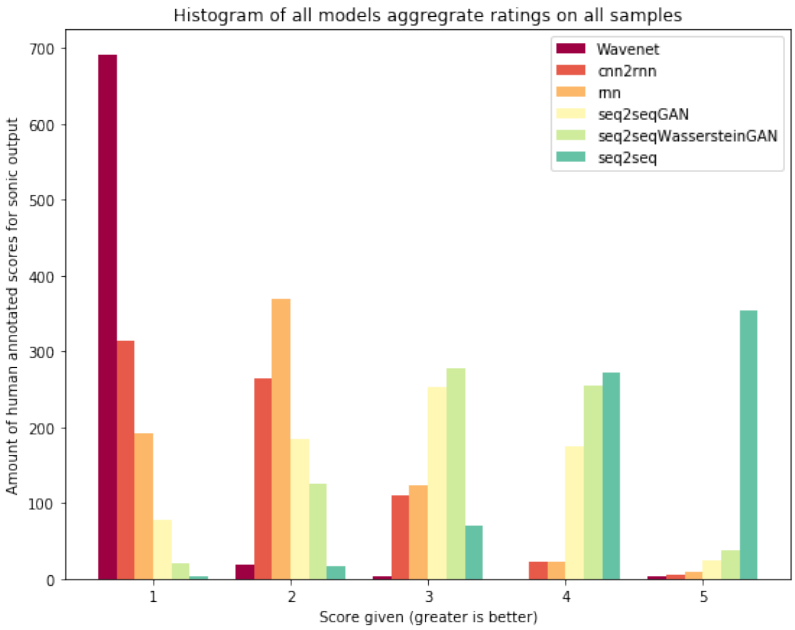


Figure 4.1: A multi-histogram of the various neural networks' scores on the samples, from a group of volunteers. Results show a clear winner, the sequence to sequence RNN. There were over three-hundred five star votes made for a sequence to sequence synthesised sample, and just under three-hundred votes gave a sequence to sequence sample a four star review.

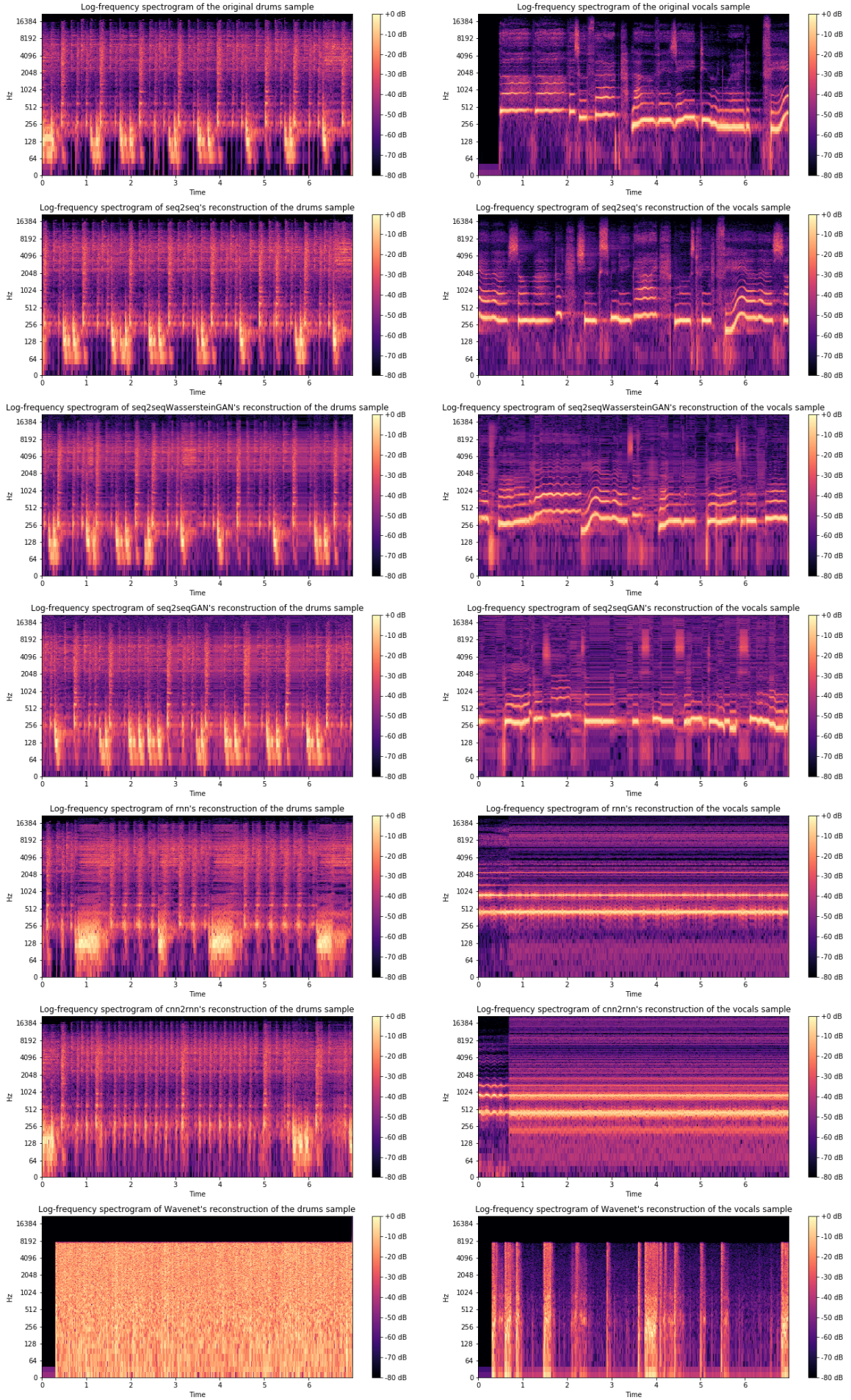


Figure 4.2: All the synthesised magnitude spectra. The left column is the drums, the right is the vocals. The first row is comprised of the original samples, and is followed by the synthesised sounds ordered from best to worst as voted by the few hundred volunteers.

Original Samples Description

The two original samples consist of a drum break and a female voice singing, which is chopped into stuttered arrangements of fake words. Both audio clips are sampled at 44100 Hertz. The drum break is considerably shorter than the singing; the drum loop is around seven seconds long, and the vocals are just longer than a minute and a half. All of the silence was trimmed from the vocals.

Description Of Seq2Seq's Output

The sequence to sequence RNN with attention as a model fitted both the samples the best, by a considerable margin. The generated vocals had a small amount of noise in random magnitude bins, but this was the least noticeable from all the tested methods. The characteristics of the vocals, the random stutters and shifts in pitch were very well captured.

The synthesised drums sounded incredibly close to the original sample and retrained relatively long-term rhythmic dependancies, albeit with a few stutters. The generated drum samples were slightly less bright than the original samples, but this was not immediately apparent upon a casual listen. One of the successes of this model was that it managed to not add high frequencies in the drums where they shouldn't occur. Other models have added noise in the higher frequency bins, which is particularly noticeable upon listening.

Description Of Seq2Seq GAN's Output

In short, introducing adversarial training into the sequence to sequence model with attention added a few side effects. Firstly, it introduced noise from around 9kHz upwards which significantly degraded the generated sounds quality. This is particularly evident upon inspection of the provided spectrograms. It should also be noted that there was noise introduced at all frequencies, although this was less perceptually relevant than the high-frequency noise because that the high frequencies shouldn't be present in the first place. Another notable effect was the poor capture of the rhythmic qualities present in the previous sounds; short stuttering sounds

were introduced, and are notably pronounced in the drum sample. This, like the high-frequency noise, is down to an inferior representation learned through the adversarial training.

Description Of Seq2Seq Wasserstein GAN's Output

Wasserstein adversarial training did admittedly improve the generative outputs of the RNN while increasing the time to convergence many-fold. There is less noise in all frequency bins, especially the higher ones, and less rhythmic stutters like those produced by the vanilla GAN seq2seq model. However, there is still a light amount of noise generated which causes phase estimation efforts issues when creating the phases, giving the neural network a tinny sound when converted back to samples.

Description Of Wavenets's Output

The Wavenet's performance was deficient for this specific task. Wavenet managed to generate little more than noise for the two samples provided, although it proved able to learn pulses of noise for the vocal sample.

Firstly and foremostly, it is much harder to learn a PCM representation than a magnitude spectrum. There is also not enough information present in a single sample for a decent PCM representation to be learned; a minor deviation from an expected PCM frame could quickly lead to a sequence never encountered before, and noise can suddenly accumulate. For this model, better results are achieved when more data is supplied to the network.

Another small thing to note is the frequency bandwidth; the Nyquist rate was at 8kHz due to the 16kHz sample rate imposed by the Wavenet architecture. This is not present in the RNN based models.

Description Of LSTM Output

The LSTM model managed to learn the drum break but overfit the vocals. The drum sounds were quite distorted but had a pleasant crash sound that many users liked aesthetically, while counterintuitively giving it a low score on quality. These crash sounds can be seen in the stretching artifacts in figure 4.2 in the drum LSTM sub graph. With respect to the vocal sample, the model failed to successfully learn a generative representation. Instead, the LSTM model output energy in the same frequency bins that were input as a sequence to the model.

Description Of CNN to LSTM Output

The CNN to LSTM model produced more noisy representations than the RNN and was generally perceived worse than all architectures but Wavenet. Previous experiments building up to this did not indicate this result, but the hyperparameters for the model have since been lost and cannot be replicated.

On Why Some Sounds Get Stuck And Loop

The minimisation of a mean-squared error loss function means the neural network's outputs will learn to approximate the conditional averages of the target data. In regression tasks, conditional averages are a limited description of the target data properties [119]. Consider problems where the representation to be approximated is multi-valued. In audio signals, it is reasonable to expect such a situation could occur. In the case of the magnitude spectra of a sine tone playing the C major scale up one octave, and then back down again, there will be multiple ambiguities concerning which note's magnitude spectra follow each note. Consider the note F. Because the note is preceded by G if the scale is descending, and preceded by E if the scale is ascending, the model will learn a conditional average of the two, and the conditional average given the input data of the note F will reside in the spectra of F, which is incorrect.

The usage of mean-squared error in the recurrent neural networks is potentially problematic when modelling audio signals. Overfitting a sound with a many to one ratio recurrent neural

network can lead to a repeating set of magnitudes, which is more than likely the conditional average of the target data. This tendency was observed many times in the development in this thesis, and is particularly evident in figure 4.2; see the vocal samples for the RNN and CNN to RNN. A many to many relationship recurrent neural network fares a little better; the sequences it must learn to output still can vary, forcing it not to produce a stuck repeating looping noise, but can still lead to choppy, junglist sounds. It is potentially the case that a mixture density network is an answer, although this hypothesis remains to be tested.

4.2 Experiment 2: Comparing Phase-Estimation Methods

4.2.1 Motivation

The majority of the neural networks presented in the first experiment generated frames of magnitudes' values. In a time-frequency representation of audio, a series of phases must be provided to perform the inverse Fourier transformation to produce PCM audio. None of the neural networks estimated the phases in experiment one, although we explore this in the next experiment. The phases must be determined in some manner, and in this experiment we evaluate empirically the available handcrafted methods to produce such phase spectra, and the common pitfalls of such methods in the context of real-time synthesis.

4.2.2 Experiment Details

Three methods were pitted against one another, and the results are empirically evaluated. The three methods included Griffin-Lim, an inherently random initialisation and Local Weighted Sums.

4.2.3 Results

Plotting the most significant phase spectra and the histogram comprising the distribution of phases doesn't reveal much on a given phase estimation's capacity to synthesise convincing sounds, as noted in figure 4.3. However, we can observe the sound's magnitudes post application of a random initialisation of phases and inverse Fourier transform and see degradation in the resultant audio. Upon listening to the output, this is sonically confirmed; a random initialisation of phases produces a spectral smoothing effect that destroys the clarity of the sound.

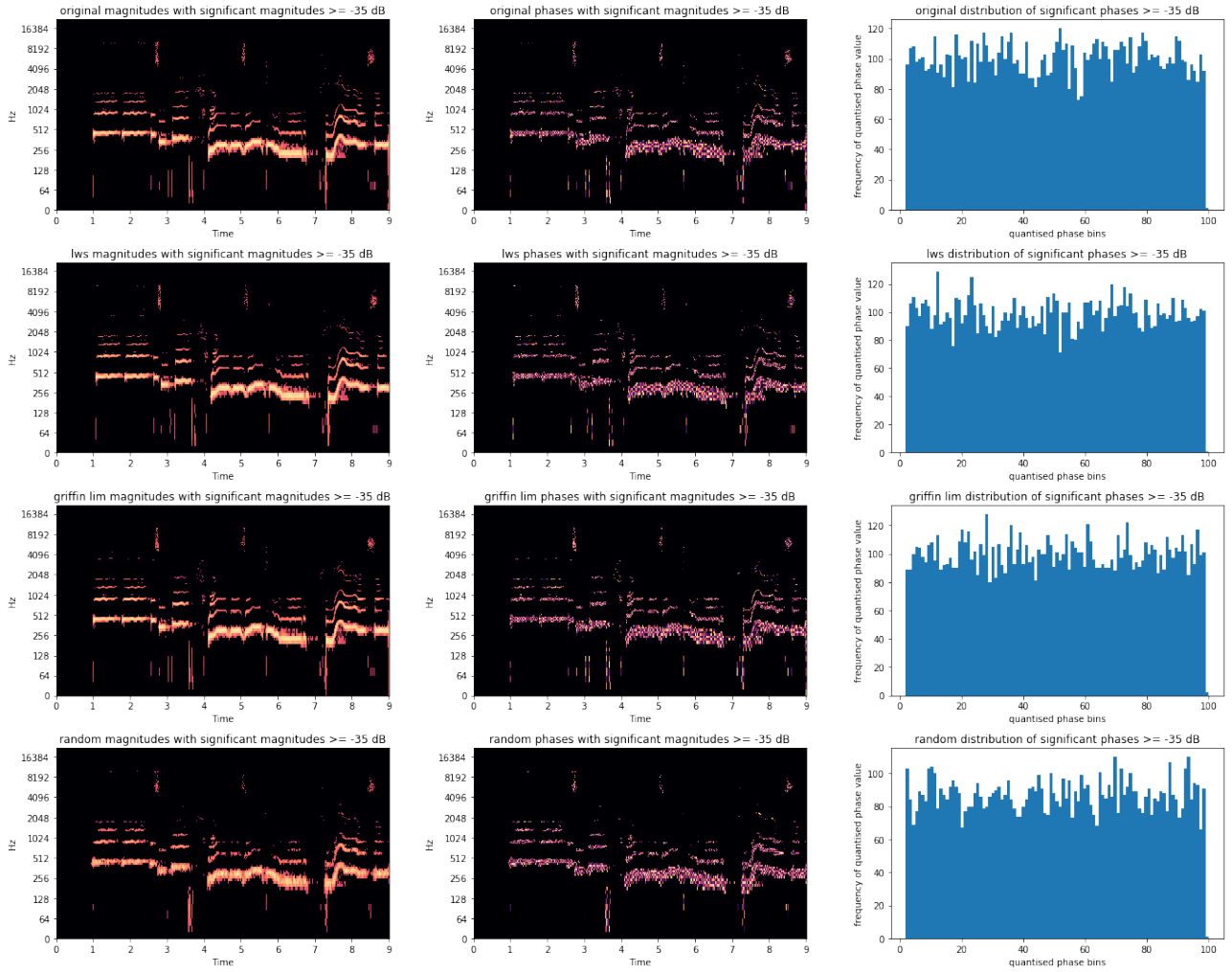


Figure 4.3: The significant phases estimated by each of the used algorithms for the same magnitudes and the histograms of these significant phases.

Comparing Griffin Lim verses Local Weighted Sums on the ability to produce compelling generative audio is a more difficult task; both algorithms produce similar output. There is a clear winner on the grounds of computational efficiency, however, as LWS performs the computation

in a fraction of the requisite time of Griffin Lim.

4.3 Experiment 3: Complex Audio Modelling

4.3.1 Motivation

Experiment one highlighted the aptest neural network for unconditional audio synthesis available in the author's arsenal for the given task of learning from a limited dataset of audio quickly and efficiently. Because the approach generated magnitude values only, the network was reliant on a handcrafted system of some kind detailed further in experiment two.

However, these handcrafted approaches are fundamentally flawed each in at least one manner. Griffin-Lim suffers due to the high iteration amount required for a quality phase estimation, thus rendering the approach unusable for real-time applications. Griffin-Lim is also a non-convex algorithm [35], meaning that there is no guarantee of a decent solution each time it is used and local minima will plague it. The randomised phases method produces poor sonic output due to the random phase offsets causing the Fourier series to create a vastly different time-domain waveform than the target sound. Finally, the Local Weighted Sums method produces phase information quickly, however, in practice, it must be applied to all generated magnitudes in one go. LWS requires a look ahead; meaning while it is fast enough for real-time usage, it is not feasible as all the seconds or minutes of magnitudes must be available before it can be applied. LWS can in effect be buffered, but this will cause a degradation in the quality of the generated time domain signal.

These kinds of problems motivated Oord, Aaron van den, et al. to build WaveNet [94]; an entirely end to end neural synthesis system can sidestep these kinds of problems. However, end to end neural networks are not limited to a PCM audio representation, by training the neural network to produce phases, a fully end to end neural system can be optimised in the time-frequency domain, and the resultant spectrum can be converted to time-domain audio through the inverse Fourier transform. A magnitude and phase representation network subsequently

sidesteps all the real-time issues priorly listed.

4.3.2 Neural And Dataset Modifications

The neural network with the highest rating from the public votes from experiment one was used as the foundational neural architecture for this experiment. This meant that the sequence to sequence with Luong attention, and no adversarial training was used. The amount of layers were reduced for both the encoder and decoder for this experiment to reduce the graphical memory requirements during training.

In experiment one, the dataset was comprised of sequences of magnitude frames; the sequence to sequence model would take a sequence of magnitude frames and predict the following sequence. At the time of writing, there are no deep learning frameworks that support complex gradients to the level that supports a recurrent neural network with attention, so a workaround was introduced. The workaround was that the phase information was normalised, and concatenated to the sequences of magnitudes.

The normalisation of the phases was relatively simple; the imaginary component of the time-frequency spectrum is converted to a phase angle specified in radians. This angle is then divided by two times Pi to produce a value between zero and one. Finally, $\frac{1}{2}$ is subtracted from the normalised phase values to zero centre them. The same procedure run in reverse was used to de-normalise the predicted phases generated by the neural network.

4.3.3 Results

Initial experiments with a short drum break proved successful, shown in figure 4.4; the model quickly produced convincingly similar sounds in a short period of training and well within the speed required for real-time usage; around thirty seconds of audio was generated in about five seconds of computation.

The same model next optimised a larger and more difficult sample. The network found this

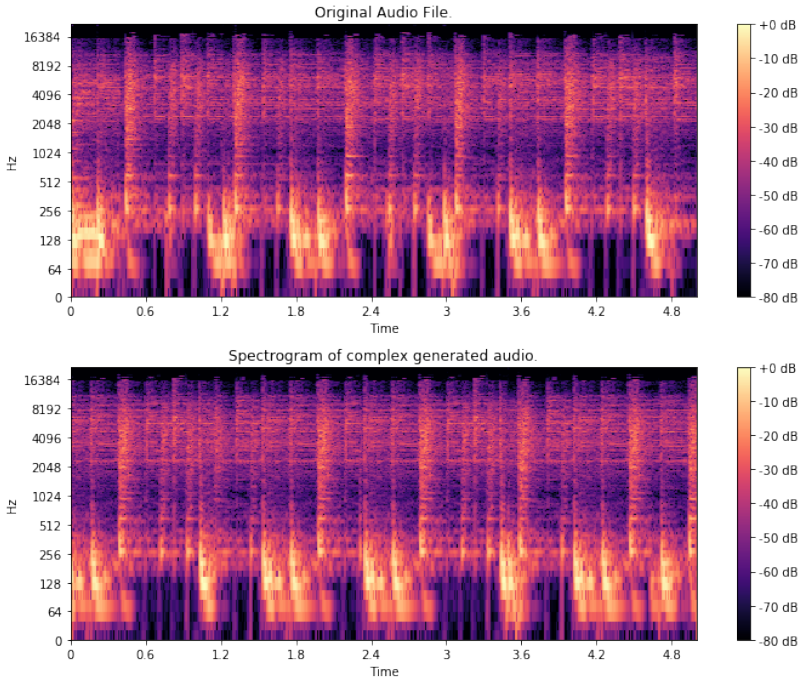


Figure 4.4: The original sample, and the spectrogram derived from magnitudes and phases dually generated during inference.

sample harder to model; phase smearing made the generated output sound tinny and metallic. This effect is visualised in figure 4.5. Further work needs to be done on this model and approach; clearly complex signals can be modelled with the sequence to sequence architecture with the few modifications made. Therefore, with more care and thought, it should be possible to improve the representations learned and the resulting generative audio. One avenue worth trying is conditioning the phases on the magnitudes. This would not be a dissimilar approach to WaveRNN [120] and could prove a fruitful endevour.

4.4 Experiment 4: Hyperparameter Search Visualisation

4.4.1 Motivation

Another active interest was visualising the distribution of generative material produced during a neural network's hyperparameter search. Vast datasets of generated audio can easily be

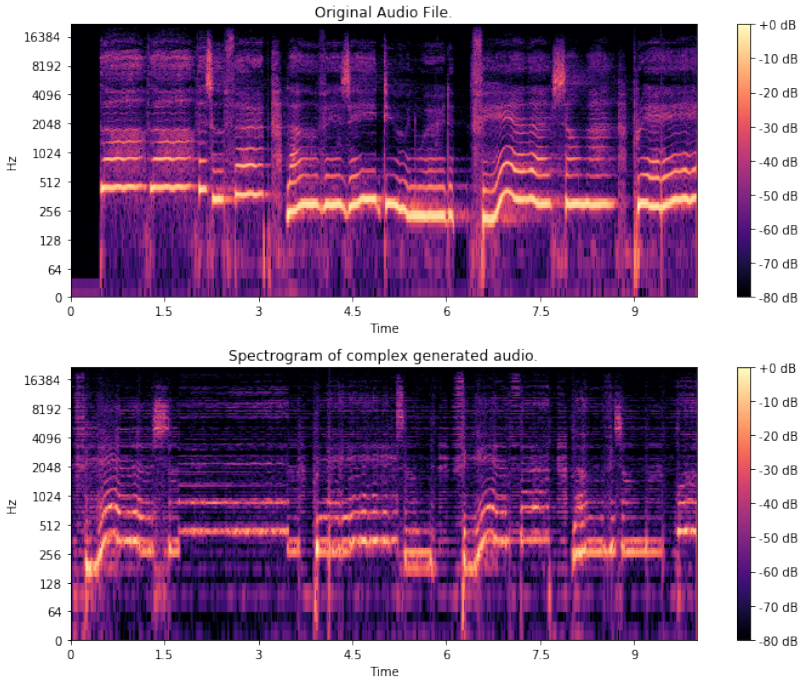


Figure 4.5: The original vocal's sample, and a noisy spectrogram featuring phase smearing derived from magnitudes and phases dually generated during inference.

produced during the training of many models over many epochs, and it can be difficult for a single programmer to go through this data and extrapolate the preferred hyperparameters.

4.4.2 Experiment Process

A random search was performed over around twenty hyperparameters, each combination trained to a fixed size amount of epochs. The resulting audio was processed into MFCC frames and converted into fixed-size feature vectors derived from the mean, standard deviation, and first order difference between the MFCC frames. These feature vectors are further processed by a range of dimensionality reduction algorithms to two dimensions, and plotted in a web application as an interactive two-dimensional scatter-plot. The dimensionality reduction algorithms used were t-SNE, UMAP and PCA, excluding the matter that MFCCs are essentially a form of dimensionality reduction when representing audio. When the user hovers the mouse over a data-point, which is a coordinate representation of some generated material by a set of hyperparameters, the hyperparameters used to make the sound are shown in the sidebar [1].

[1]This web application used with the audio generated from a style transfer random search and the associated parameters is hosted at the following url `http://doc.gold.ac.uk/~lfedd001/web_visualiser/demo.html`

4.4.3 Results

This process allows the user to cluster sets of hyperparameters on sound similarity and intuitively see which settings frequently contribute to their preferred sounds. Of course, the process of converting hundreds of thousands of audio samples into just two coordinates is incredibly lossy and is not semantically perfect, but it provides a viable process to inspect the material in a far more intuitive manner than manually navigating filesystems. The author will be fully automating this process within a Python module and applying it to future generative works.

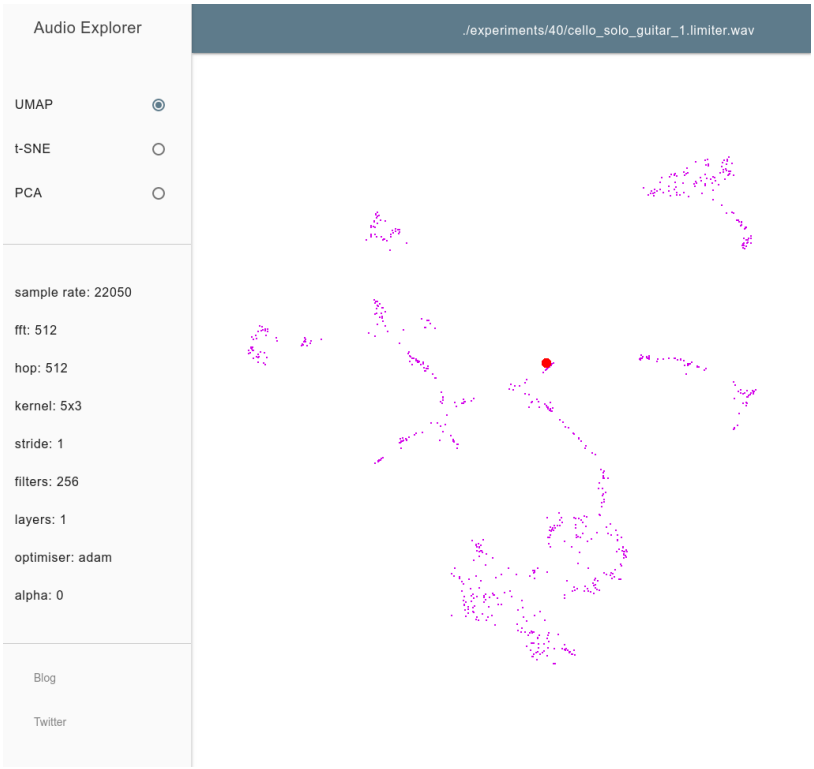


Figure 4.6: The web application that plots the generative audio as a two dimensional scatter plot, clustered by sound similarity and showing hyperparameters in the left hand side tool bar.

4.5 Experiment 5: Deterministic Audio Style Transfer

4.5.1 Experiment Conditions

This experiment builds on the work of Mital [112] by experimenting how to deterministically parameterise and control audio style transfer. A crucial difference to traditional style transfer

is that the content part of the loss is discarded.

The algorithm must begin by being seeded, to enable its determinism for predictable results. Next, simplex noise is constructed, from a fixed amount of parameters [121]. The noise and a concatenative feature vector of real, imaginary, and magnitudes are derived from the style audio sample, and are both passed through a randomised convolutional neural network. The idea is to compute style features $\boldsymbol{F}$, from successive layers of the convolutional neural network as a gram matrix $\boldsymbol{G}$, such that $\boldsymbol{G} = \boldsymbol{F}^T\boldsymbol{F}$. This matrix $\boldsymbol{G}$ captures a measure of which features fire together.

The loss is the sum of the Euclidean distance between the successive gram matrices at each convolutional layer created from both the audio and noise inputs. The noise is optimised by the L-BFGS algorithm [122] to minimise the loss.

4.5.2 Results

The synthesised results capture the timbral and pitch content quite well of the sample used for the style features. The reconstruction is a little noisy, and the transients have mostly been lost.

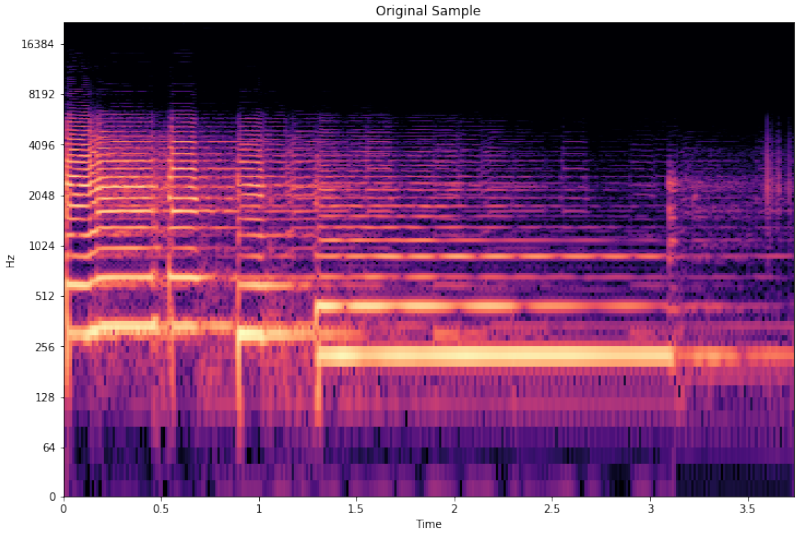


Figure 4.7: A sample of a guitar used as the style input for the audio style transfer algorithm.

Because the network and noise is seeded, this process is deterministic and controllable through its intialisation, which in this case, is the two parameters determining the simplex noise. This is

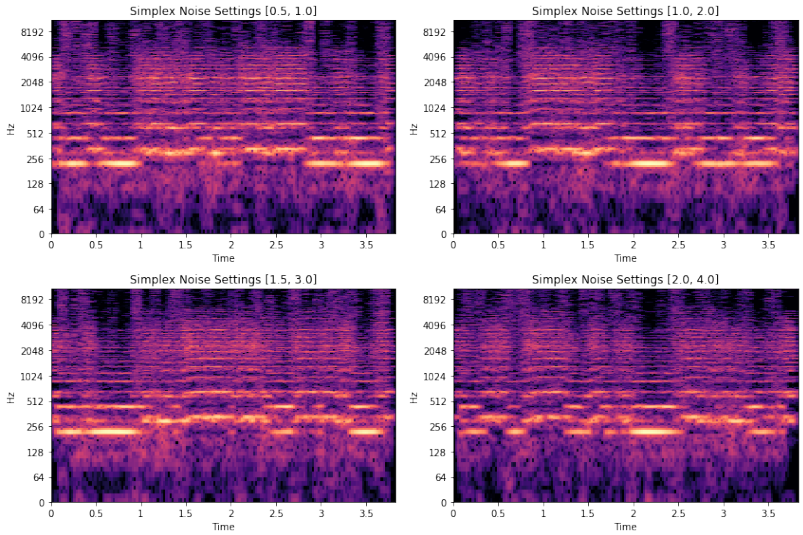


Figure 4.8: Samples produced from the sample show in figure 4.7 using the style transfer method.

not intuitive to control, and whilst the sonic differences across different intialisations exist, they are not particularly varied. It could however be argued that unconditional generative neural networks are not particularly intuitive to control either. There is a downside to this method, in that each new sample takes a long time to optimise and create, whereas once a neural network is trained, inference can be performed relatively cheaply.

Chapter 5

Conclusion

5.1 Summary of Thesis Achievements

This thesis has met the aims set out in the introduction chapter, of producing compelling audio synthesis using neural networks. The best network synthesises high-quality audio at real-time capable speeds; the network produces 23 seconds of audio in only 4, at a sample rate of 44100 and a bit depth of 64 bits. Media artists on an economic budget can achieve usable results with little computation power; a single consumer-level Graphical Processing Unit can train a network in an hour or less to synthesise audio that mimics the desired sound in new ways, and this process has no more domain knowledge required than running a script. In time, this could easily be packaged into a more intuitive user interface such as a VSTi or web application.

The achievements of this thesis are listed as follows: a recurrent neural network architecture has been empirically voted as the best against five other architectures, and the details of all network implementations have been set out; a method and web application of visualising the efficacy of such architectures has been implemented and offered; textural audio synthesis, an extension of style transfer, has been reviewed; early experiments into modelling complex signals have been conducted; and phase estimation methods have been reviewed for the task of neural synthesis.

5.2 Applications

Since the time of writing and the preliminary results having been published online, several artists and musicians have been in touch to find out more about the process. One popular group has had a more sustained interest in this and preceding projects for some time, so it is likely that this kind of technology will garner wider interest on the introduction of better user interfaces that make the technology more accessible.

One way to deploy the leading model is to embed the model in a VSTi using a framework such as JUCE. Parameters could be trained separately and loaded as patches. Alternatively, deploying as a javascript framework packaged as a web application would make the network much more accessible to users through the internet. The network could directly be used to produce new fixed size samples; percussion and producer packs are popular with those who make music electronically. Naturally, the real-time, high fidelity, neural networks could also be applied to create higher quality voice samples that would contribute to the field of text to speech synthesis.

5.3 Future Work

There are a number of interesting directions to extend the neural network architectures detailed in this thesis for the problem of musical audio synthesis. Firstly, extending the approach of modelling complex signals could lead to quality synthesis results, and this could be explored in a number of avenues. One approach could be to directly build the operations to enable recurrent or convolutional neural networks that are capable of propagating complex based signals and gradients. Another approach is to look at WaveRNN, where a series of predicted fine amplitude values are conditioned on course amplitude values [83]; this approach could be extrapolated to magnitudes and phases, where the phases containing the finer grained audio information are conditioned directly on the magnitude values.

Another point of interest is reducing the search space of the recurrent neural networks. Cur-

rently, the model performs a regression, and should there be a manner to quantise each magnitude bin at each timestep, this should be tested. Transforming the regression problem to a classification problem, and reducing the search space could make the representation a lot easier to learn whilst returning perceptually comparable results much quicker.

There is also the issue of controlling the recurrent neural networks listed in this thesis. Currently, the networks generate new audio in an unconditional manner. Perhaps providing some global context in the style of popular speech to text systems [79] would be worth pursuing, although this would raise the problem of creating such a dataset of sound and context vectors. Or, another avenue would be to explore variational RNNs and walk through the latent space [123].

Bibliography

[1] N. Bourdakos, "Capsule Networks Are Shaking up AI - Here's How to Use Them," 2017.

[2] M. V. Mathews, J. E. Miller, F. R. Moore, J. R. Pierce, and J. C. Risset, *The Technology of Computer Music*. The MIT Press, 1969.

[3] G. E. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[4] J. O. Smith, "Viewpoints on the history of digital synthesis," in *Proceedings of the International Computer Music Conference*, pp. 1–1, INTERNATIONAL COMPUTER MUSIC ACCOCIATION, 1991.

[5] C. Roads and J. Strawn, *The Computer Music Tutorial*. The Computer Music Tutorial, MIT Press, 1996.

[6] Alioth Finance, "Financeref inflation calculator." `http://www.in2013dollars.com/1876-dollars-in-2018?amount=1000000`, 2011. [Online; accessed 23-April-2018].

[7] D. A. Hounshell, "Elisha gray and the telephone: On the disadvantages of being an expert," *Technology and Culture*, vol. 16, no. 2, pp. 133–161, 1975.

[8] K. Collins and C. Collins, *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design*. MIT Press, 2008.

[9] J. Lazzaro and J. Wawrzynek, "Subtractive synthesis without filters," *Audio Anecdotes IITools, Tips, and Techniques for Digital Audio*, pp. 55–64, 2004.

[10] D. Bush and R. Kassel, *The Organ: An Encyclopedia*. Encyclopedia of Keyboard Instruments: The Organ : an Encyclopedia, Routledge, 2006.

[11] B. J. Copeland and J. Long, *Turing and the History of Computer Music*. Cham: Springer International Publishing, 2017.

[12] P. Manning and O. U. Press, *Electronic and Computer Music*. Oxford University Press, 2004.

[13] V. Lazzarini, S. Yi, J. Ffitch, J. Heintz, O. Brandtsegg, and I. McCurdy, *Csound: A Sound and Music Computing System*. Springer Publishing Company, Incorporated, 1st ed., 2016.

[14] J. McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002.

[15] M. Puckette, "Pure data: Recent progress," pp. 1–4, 1997.

[16] M. Grierson and C. Kiefer, "Maximillian: An easy to use, cross platform c++ toolkit for interactive audio and synthesis applications," pp. 276–279, 01 2011.

[17] J. M. Chowning, "The synthesis of complex audio spectra by means of frequency modulation," *Journal of the audio engineering society*, vol. 21, no. 7, pp. 526–534, 1973.

[18] P. Elsea, *The Art and Technique of Electroacoustic Music:*. Computer Music and Digital Audio Series, A-R Editions, 2013.

[19] Yamaha, "The dawn of home music production." `https://usa.yamaha.com/products/contents/music_production/synth_40th/history/chapter02/`, 2014. [Online; accessed 24-April-2018].

[20] S. Cann, *How to Make a Noise: Sample-Based Synthesis*. BookBaby, 2011.

[21] R. Rabenstein and L. Trautmann, "Digital sound synthesis by physical modelling," in *Image and Signal Processing and Analysis, 2001. ISPA 2001. Proceedings of the 2nd International Symposium on*, pp. 12–23, IEEE, 2001.

[22] A. Hugill, *The Digital Musician*. Taylor & Francis, 2012.

[23] MU History, "The musicians' union: A history (1893-2013)." `https://www.muhistory.com/contact-us/1971-1980/`, 2013. [Online; accessed 25-April-2018].

[24] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[25] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[26] B. Waggener, W. Waggener, and W. Waggener, *Pulse Code Modulation Techniques*. A Solomon press book, Springer US, 1995.

[27] R. Freeman, *Fundamentals of Telecommunications*. Wiley Series in Telecommunications and Signal Processing, Wiley, 2005.

[28] D. Benson, *Music: A Mathematical Offering*. Cambridge University Press, 2007.

[29] A. Nisbett, *Sound Studio: Audio Techniques for Radio, Television, Film and Recording*. Taylor & Francis, 2003.

[30] C. Chen, *Signal Processing Handbook*. Electrical and Computer Engineering, Taylor & Francis, 1988.

[31] L. Stankovic, M. Dakovic, and T. Thayaparan, *Time-Frequency Signal Analysis with Applications*. Artech House Radar, Artech House, Incorporated, 2014.

[32] N. Council, D. Education, C. Board on Behavioral, C. Impairments, S. Van Hemel, and R. Dobie, *Hearing Loss: Determining Eligibility for Social Security Benefits*. National Academies Press, 2004.

[33] F. Camastra and A. Vinciarelli, "Audio acquisition, representation and storage," in *Machine Learning for Audio, Image and Video Analysis*, pp. 13–55, Springer, 2015.

[34] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[35] D. L. Sun and J. O. Smith III, "Estimating a signal from a magnitude spectrogram via convex optimization," *arXiv preprint arXiv:1209.2076*, 2012.

[36] J. Le Roux, H. Kameoka, N. Ono, and S. Sagayama, "Fast signal reconstruction from magnitude STFT spectrogram based on spectrogram consistency," in *Proc. International Conference on Digital Audio Effects (DAFx)*, pp. 397–403, Sept. 2010.

[37] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[38] G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[39] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[40] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[41] D. Elizondo, "The linear separability problem: Some testing methods," *IEEE Transactions on neural networks*, vol. 17, no. 2, pp. 330–344, 2006.

[42] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *CoRR*, vol. abs/1511.07289, 2015.

[43] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323, 2011.

[44] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[45] R. Hoskinson, K. van den Doel, and S. Fels, "Real-time adaptive control of modal synthesis," in *Proceedings of the 2003 Conference on New Interfaces for Musical Expression*, NIME '03, (Singapore, Singapore), pp. 99–103, National University of Singapore, 2003.

[46] G. Costantini, M. Todisco, and M. Carota, "A neural network based interface to real time control musical synthesis processes," in *Proceedings of the 11th WSEAS International Conference on CIRCUITS, Agios Nikolaos, Crete Island, Greece*, pp. 41–45, 2007.

[47] H. Zen and A. Senior, "Deep mixture density networks for acoustic modeling in statistical parametric speech synthesis," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 3844–3848, IEEE, 2014.

[48] A. M. Sarroff and M. A. Casey, "Musical audio synthesis using autoencoding neural nets," in *ICMC*, 2014.

[49] S. Desai, A. W. Black, B. Yegnanarayana, and K. Prahallad, "Spectral mapping using artificial neural networks for voice conversion," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, pp. 954–964, July 2010.

[50] L.-H. Chen, Z.-H. Ling, L.-J. Liu, and L.-R. Dai, "Voice conversion using deep neural networks with layer-wise generative training," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, pp. 1859–1872, Dec. 2014.

[51] A. Shahina and B. Yegnanarayana, "Mapping speech spectra from throat microphone to close-speaking microphone: A neural network approach," *EURASIP Journal on Advances in Signal Processing*, vol. 2007, no. 1, p. 087219, 2007.

[52] S. Khanum and M. Sora, "Speech based gender identification using feed forward neural networks," in *International Journal of Computer Applications (0975–8887): National Conference on Recent Trends in Information Technology*, Citeseer.

[53] Y. Petetin, C. Laroche, and A. Mayoue, "Deep neural networks for audio scene recognition," in *Signal Processing Conference (EUSIPCO), 2015 23rd European*, pp. 125–129, IEEE, 2015.

[54] M. Lim, D. Lee, H. Park, U. Park, J.-H. Kim, J.-S. Park, and G.-J. Jang, "Audio event classification using deep neural networks," *Phonetics and Speech Sciences*, vol. 7, no. 4, pp. 27–33, 2015.

[55] E. Cakir, "Multilabel sound event classification with neural networks," 2014.

[56] V. Pulkki, S. Delikaris-Manias, and A. Politis, *Parametric Time-Frequency Domain Spatial Audio*. Wiley - IEEE, Wiley, 2017.

[57] L. B. Fah, A. Hussain, and S. A. Samad, "Speech enhancement by noise cancellation using neural network," in *TENCON 2000. Proceedings*, vol. 1, pp. 39–42, IEEE, 2000.

[58] M. Marolt, A. Kavcic, and M. Privosnik, "Neural networks for note onset detection in piano music," in *Proceedings of the 2002 International Computer Music Conference*, 2002.

[59] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[60] M. Berglund, T. Raiko, M. Honkala, L. Kärkkäinen, A. Vetek, and J. T. Karhunen, "Bidirectional recurrent neural networks as generative models," in *Advances in Neural Information Processing Systems*, pp. 856–864, 2015.

[61] A. Karpathy, "The Unreasonable Effectiveness of Recurrent Neural Networks," May 2015.

[62] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[63] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[64] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[65] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Fifteenth annual conference of the international speech communication association*, 2014.

[66] C. Olah, "Understanding lstm networks," *GITHUB blog, posted on August*, vol. 27, p. 2015, 2015.

[67] R. Pascanu, T. Mikolov, and Y. Bengio, "Understanding the exploding gradient problem," *CoRR*, vol. abs/1211.5063, 2012.

[68] S.-H. Chen, S.-H. Hwang, and Y.-R. Wang, "An rnn-based prosodic information synthesizer for mandarin text-to-speech," *IEEE transactions on speech and audio processing*, vol. 6, no. 3, pp. 226–239, 1998.

[69] D. K. Roy, N. Sawhney, C. Schmandt, and A. Pentland, "Wearable audio computing: A survey of interaction techniques," *http://www. media. mit. edu/˜ nitin/NormadicRadio/AudioWearables. html*, 1997.

[70] H. Zen, "Acoustic modeling in statistical parametric speech synthesis–from hmm to lstm-rnn," *Proc. MLSLP*, 2015.

[71] H. Zen and H. Sak, "Unidirectional long short-term memory recurrent neural network with recurrent output layer for low-latency speech synthesis," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 4470–4474, IEEE, 2015.

[72] H. Zen, Y. Agiomyrgiannakis, N. Egberts, F. Henderson, and P. Szczepaniak, "Fast, compact, and high quality LSTM-RNN based statistical parametric speech synthesizers for mobile devices," *CoRR*, vol. abs/1606.06061, 2016.

[73] B. Li and H. Zen, "Multi-language multi-speaker acoustic modeling for lstm-rnn based statistical parametric speech synthesis.," in *INTERSPEECH*, pp. 2468–2472, 2016.

[74] A. Gutkin, L. Ha, M. Jansche, O. Kjartansson, K. Pipatsrisawat, and R. Sproat, "Building statistical parametric multi-speaker synthesis for bangladeshi bangla," *Procedia Computer Science*, vol. 81, pp. 194–200, 2016.

[75] M. S. Al-Radhi, "High quality continuous residual-based vocoder for statistical parametric speech synthesis,"

[76] Y. Wang, R. J. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio, Q. V. Le, Y. Agiomyrgiannakis, R. Clark, and

R. A. Saurous, "Tacotron: A fully end-to-end text-to-speech synthesis model," *CoRR*, vol. abs/1703.10135, 2017.

[77] A. Barron, "End-to-end neural speech synthesis,"

[78] Y. Lee, A. Rabiee, and S. Lee, "Emotional end-to-end neural speech synthesizer," *CoRR*, vol. abs/1711.05447, 2017.

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

[80] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. C. Courville, and Y. Bengio, "Samplernn: An unconditional end-to-end neural audio generation model," *CoRR*, vol. abs/1612.07837, 2016.

[81] V. Badenas Crespo, "Trombone synthesis using deep learning," B.S. thesis, Universitat Politècnica de Catalunya, 2017.

[82] Z. Zukowski and C. Carr, "Generating black metal and math rock: Beyond bach, beethoven, and beatles," *NIPS*, 2017.

[83] 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*, vol. abs/1802.08435, 2018.

[84] M. Thom and G. Palm, "Sparse activity and sparse connectivity in supervised learning," *Journal of Machine Learning Research*, vol. 14, no. Apr, pp. 1091–1143, 2013.

[85] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[86] K. Lenc and A. Vedaldi, "Understanding image representations by measuring their equivariance and equivalence," in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pp. 991–999, IEEE, 2015.

[87] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *Advances in neural information processing systems*, pp. 2017–2025, 2015.

[88] M. Jaderberg, K. Simonyan, A. Zisserman, and k. kavukcuoglu, "Spatial transformer networks," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2017–2025, Curran Associates, Inc., 2015.

[89] F. Tschopp, J. N. Martel, S. C. Turaga, M. Cook, and J. Funke, "Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems," in *Biomedical Imaging (ISBI), 2016 IEEE 13th International Symposium on*, pp. 1225–1228, IEEE, 2016.

[90] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[91] E. Sprengel, M. Jaggi, Y. Kilcher, and T. Hofmann, "Audio based bird species identification using deep learning techniques," in *LifeCLEF 2016*, no. EPFL-CONF-229232, pp. 547–559, 2016.

[92] Y. Aytar, C. Vondrick, and A. Torralba, "Soundnet: Learning sound representations from unlabeled video," in *Advances in Neural Information Processing Systems*, pp. 892–900, 2016.

[93] V. Kuleshov, S. Z. Enam, and S. Ermon, "Audio super resolution using neural networks," *arXiv preprint arXiv:1708.00853*, 2017.

[94] A. v. d. 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," *arXiv preprint arXiv:1609.03499*, 2016.

[95] A. v. d. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. v. d. Driessche, E. Lockhart, L. C. Cobo, F. Stimberg, *et al.*, "Parallel wavenet: Fast high-fidelity speech synthesis," *arXiv preprint arXiv:1711.10433*, 2017.

[96] S. O. Arik, M. Chrzanowski, A. Coates, G. Diamos, A. Gibiansky, Y. Kang, X. Li, J. Miller, A. Ng, J. Raiman, *et al.*, "Deep voice: Real-time neural text-to-speech," *arXiv preprint arXiv:1702.07825*, 2017.

[97] J. Engel, C. Resnick, A. Roberts, S. Dieleman, D. Eck, K. Simonyan, and M. Norouzi, "Neural audio synthesis of musical notes with wavenet autoencoders," *CoRR*, vol. abs/1704.01279, 2017.

[98] K. Fisher and A. Scherlis, "Wavemedic: Convolutional neural networks for speech audio enhancement,"

[99] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[100] M. Wattenberg, F. Vigas, and I. Johnson, "How to use t-sne effectively," *Distill*, 2016.

[101] L. H. Dicker and D. P. Foster, "One-shot learning and big data with n= 2," in *Advances in Neural Information Processing Systems*, pp. 270–278, 2013.

[102] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[103] Igor Babuschkin, "Community implementation of wavenet." `https://github.com/ibab/tensorflow-wavenet`, 2015. [Online; accessed 24-April-2018].

[104] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[105] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International Conference on Machine Learning*, pp. 1310–1318, 2013.

[106] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, pp. 1019–1027, 2016.

[107] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[108] A. M. Lamb, A. G. A. P. GOYAL, Y. Zhang, S. Zhang, A. C. Courville, and Y. Bengio, "Professor forcing: A new algorithm for training recurrent networks," in *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

[109] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," *arXiv preprint*, 2017.

[110] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," in *Advances in Neural Information Processing Systems*, pp. 5769–5779, 2017.

[111] L. A. Gatys, A. S. Ecker, and M. Bethge, "A neural algorithm of artistic style," *arXiv preprint arXiv:1508.06576*, 2015.

[112] P. K. Mital, "Time domain neural audio style transfer," *arXiv preprint arXiv:1711.11160*, 2017.

[113] P. Hamel and D. Eck, "Learning features from music audio with deep belief networks.," in *ISMIR*, pp. 339–344, Utrecht, The Netherlands, 2010.

[114] three.js, "three.js / editor," 2015.

[115] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[116] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[117] A. Damien *et al.*, "Tflearn." `https://github.com/tflearn/tflearn`, 2016.

[118] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[119] C. M. Bishop, "Mixture density networks," 1994.

[120] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. v. d. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," *arXiv preprint arXiv:1802.08435*, 2018.

[121] K. Perlin, "Improving noise," in *ACM Transactions on Graphics (TOG)*, vol. 21, pp. 681–682, ACM, 2002.

[122] D. C. Liu and J. Nocedal, "On the limited memory bfgs method for large scale optimization," *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.

[123] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio, "A recurrent latent variable model for sequential data," in *Advances in neural information processing systems*, pp. 2980–2988, 2015.