Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

Barnabás Suciu

EXPLORING EFFICIENT NEURAL ARCHITECTURES FOR TEXT-TO-SPEECH SYNTHESIS

SUPERVISOR

Dr. Mohammed Salah Al-Radhi

BUDAPEST, 2021

Contents

STUDENT DECLARATION

I, **Suciu Barnabás**, the undersigned, hereby declare that the present BSc thesis work has been prepared by myself and without any unauthorized help or assistance. Only the specified sources (references, tools, etc.) were used. All parts taken from other sources word by word, or after rephrasing but with identical meaning, were unambiguously identified with explicit reference to the sources utilized.

I authorize the Faculty of Electrical Engineering and Informatics of the Budapest University of Technology and Economics to publish the principal data of the thesis work (author's name, title, abstracts in English and in a second language, year of preparation, supervisor's name, etc.) in a searchable, public, electronic and online database and to publish the full text of the thesis work on the internal network of the university (this may include access by authenticated outside users). I declare that the submitted hardcopy of the thesis work and its electronic version are identical.

Full text of thesis works classified upon the decision of the Dean will be published after a period of three years.

Budapest, 14 May 2021

Suciu Barnabás

.....

Suciu Barnabás

Summary

The current thesis focuses on the exploration of efficient neural architectures for vocoder-based text-to-speech (TTS) systems. As opposed to end-to-end solutions, the use of a vocoder and separate steps for creating a duration and an acoustic model with which the speech is synthesized, a much shorter training time frame can be achieved, the size requirements of the training dataset are also quite modest, and inference can be solved in real-time. Furthermore, I explore the recently proposed statistical parametric continuous vocoder in a sequence-to-sequence recurrent neural network based on TTS. Experimentation is conducted with the Merlin open-source text-to-speech toolkit. The WORLD vocoder was chosen for comparison with the optimized vocoder. The testing includes both traditional sequential neural architectures and recurrent models, such as LSTM (long short-term memory) and GRU (gated recurrent unit). I use objective evaluation as well as subjective metrics to evaluate and compare my results. Experimental results have shown that the Seq2Seq model has higher naturalness than the one based on the WORLD framework while being simpler in terms of the number of excitation parameters.

Összefoglaló

Jelen dolgozat fókusza a hatékony neurális háló architektúrákkal történő kísérletezés vokóder alapú szövegfelolvasó (*text-to-speech, TTS*) rendszerekkel kapcsolatban. *End-to-end* megoldásokkal ellentétben, a vokóder használata, valamint a szintézis felbontása külön akusztikus és időtartam-modellek segítségével lehetővé teszi, hogy a tanítási időtartam lerövidüljön, a tanító adatbázis méretét nagyban lecsökkentsük, valamint a végső, szintézis lépés kivitelezhető lesz valós időben. Továbbá megvizsgáljuk a nemrég kifejlesztett statisztikai parametrikus folytonos idejű vokóder lehetőségeit egy szekvencia-szekvencia mély neurális architektúrával. A tesztelés során a Merlin nyílt forráskódú TTS eszközt használjuk. A folytonos vokóderrel történő összehasonlítás alapjául a WORLD vokódert választottuk, melyet alapértelmezetten tartalmaz a Merlin rendszer. A tesztelés során mind hagyományos előrecsatolt (*feed-forward*) mély neurális architektúrákkal, mind visszacsatolt (*recurrent*) hálózatokkal foglalkozunk, úgymint LSTM (hosszú rövid-távú memória) és GRU (*gated recurrent unit*). Objektív és szubjektív metrikákat is alkalmazunk az eredmények kiértékelésére. A kísérletek eredményei megmutatták, hogy a szekvencia-szekvencia modellünknek jobb természetessége van, mint a WORLD vokóderen alapulónak, valamint paraméterek tekintetében is kisebb komplexitást értünk el.

1 Introduction

Text to speech (TTS for short) or speech synthesis refers to the transformation of written text to an audible speech waveform. This technology is similar to natural language processing, except that the goal is not to understand speech, but to generate it. Two of the main evaluation points for any synthesizer are naturalness and intelligibility, with solutions often trying to maximize both. Although most of the energy in speech is located in the lower frequencies, the human ears are sensitive to the high frequency components, meaning an ideal text-to-speech system has to accurately model both short-and long-term variations of the speech waveform.

The applications of TTS technology are numerous. One of the most notable areas is assistive technology, enabling people with sight impairment and dyslexia to interpret written text more effectively, as well as helping people with speech impairment vocalize their thoughts. Widespread availability, ease of use and real-time synthesis are often crucial in applications like this. Another area where speech synthesis sees frequent use is entertainment, mostly in video games and movies, as well as content creation on video hosting and streaming sites. In education it is often used as a method for teaching second languages, allowing applications to circumvent having to rely on prerecorded segments of text, enabling a more personalized experience at a lower cost.

State-of-the-art text-to-speech (TTS) synthesis is either based on unit selection [1] or statistical parametric methods [2]. In the last two decades, particular attention has been paid to hidden Markov model (HMM) [3], which has gained much popularity due to its advantages in flexibility, smoothness, and small footprint.

The most recent development in speech synthesis research is the use of neural networks. These systems mimic the way the human mind works by having the ability of finding and learning correlations in the input data, and then using this knowledge to produce a new output. Deep neural networks have become the most common models used in speech synthesis, achieving significant improvements in quality [4] over previous solutions.

Although intelligibility has been steadily improving over the last decades, a constant challenge of speech synthesis is the “robotic” and unnatural sounding voice, especially when it comes to parametric methods. Another concern when it comes to these systems is training and inference time frames, with the best sounding solutions (WaveNet, for example) often requiring days to train and speech cannot be generated in real-time. Therefore, there is still room for improvement both in terms of quality and speed of synthesis.

1.1 History of text-to-speech

Ever since the late 1950s, electronic speech synthesis has been a constantly evolving field of study. The two main types of early speech synthesizers are concatenative and formant systems. The former relies on stringing together pre-recorded segments of speech (often called unit selection), and allows for high naturalness, although mismatches between samples and glitches can occur. In contrast, formant synthesis relies on generating the speech waveform itself based on parameters like fundamental frequency and voicing. This provided the possibility of real-time synthesis with high intelligibility, although with much less naturalness. In 1961, an IBM 704 computer was used to synthesize speech at Bell Labs, recreating the song "Daisy Bell". This is what gave inspiration to Arthur C. Clarke for his artificial intelligence, HAL 9000 in his screenplay for his novel, 2001: A Space Odyssey. In 1975, MUSA (Multichannel Speaking Automaton) was released as one of the first stand-alone speech synthesis systems. Still in the 70's, portable TTS systems started emerging, mainly for educational purposes and to be used in calculators for enabling accessibility features. As new software and signal processing techniques were developed, the quality of synthesized speech steadily improved throughout the 80's and 90's. From there, the field evolved into many different branches, for example the utilization of hidden Markov models, and, as is the focus of this paper, deep neural networks. In deep learning, the two most prominent approaches are end-to-end models and vocoder-based solutions. End-to-end systems utilize a large training dataset and require a large timeframe (often days) for the learning process, as well as having long inference times. They provide excellent results, though, as demonstrated by DeepMind's WaveNet in 2016 [5]. The second approach, sometimes referred to as statistical parametric speech synthesis, makes use of a vocoder and parameter extraction to speed up the process, often at the cost of some speech quality. On Figure 1.1, the block diagram of one of the first vocoders, the VODER (Voice Operating Demonstrator) is visualized, introduced by Homer Dudley at the New York World's Fair in 1939. It required an operator who controlled the selection of the noise source and operated a foot pedal to manipulate the fundamental frequency.

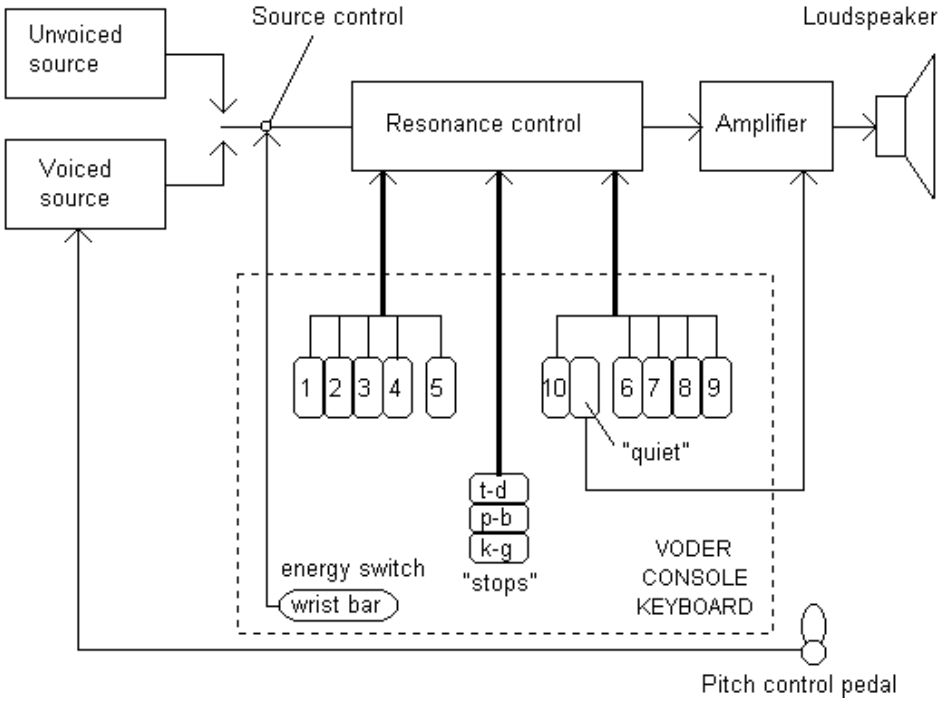

Figure 1.1: The VODER speech synthesizer

Since then, electronic vocoders have made great advances, with the advent of general-purpose computers and improvements in signal processing techniques, today's vocoders are capable of approximating the characteristics of human speech effectively. The parameters extracted by a vocoder often include fundamental and maximum voiced frequency, aperiodicity and spectral envelope. In this thesis I propose an efficient architecture for a continuous vocoder-based system, which relies on the open-source Merlin text-to-speech library.

1.2 Related work

In this section I'll discuss a few of the existing solutions for both end-to-end synthesis as well as statistical parametric speech synthesis systems.

Starting with end-to-end methods, one of the most notable ones is DeepMind's WaveNet [5] from 2016. It uses a feed-forward convolutional architecture, however, a technique called dilation (similar to strided convolution or pooling, but unlike those, the output size stays the same) allows the network to take into account many of the previous samples, without increasing computational complexity like a recurrent network would. WaveNet provided excellent results in both intelligibility and naturalness, however, the training time frame can be on the order of days, and inference is not real-time, which is especially a problem in embedded environments, like mobile phones. It is also important to mention that WaveNet required some preprocessing of the training data, therefore it is not truly end-to-end.

In 2017, Deep Voice [6] was proposed. It comprises 5 building blocks, each with a separate function like fundamental frequency estimation or phoneme duration prediction, with the final block combining results from previous ones to generate the audio waveform. Each block consists of a neural network, and these networks are individually trained. It was an important step towards end-to-end speech synthesis, the relatively simple neural networks at each step could be optimized separately with relative ease. Inference was also faster than real-time, which is an important milestone, optimization for GPU environments allowed a 400x speedup over traditional processors.

Still in the same year, researchers at Google released a paper on Tacotron [7], which uses an attention-based sequence to sequence model for speech generation. The model includes uni-and bidirectional GRU layers, as well as complex subnetworks for the encoder and post-processing functions, which are made up of ReLU, GRU and convolutional layers. This system is truly end-to-end, as it generates a speech waveform directly from text, the training input data consisting of text-audio pairs only. It made effective use of sequence-to-sequence recurrent neural networks, providing good quality speech compared to statistical parametric models while keeping computation cost down thanks to frame, rather than sample-level synthesis.

In 2018, NVIDIA Corporation developed the WaveGlow [8] system, which allowed for great parallelization on GPU-s, while being less complex to implement than previously mentioned solutions. It makes use of a mel-spectrogram to model the distribution of audio samples, which is then fed through the neural network to arrive at the desired distribution. The network itself is simpler than previously mentioned examples, providing quality matching the best WaveNet implementations at the time, while being able to produce speech at a rate of 500 kHz on a GPU, which is 25 times faster than real-time.

The last end-to-end synthesizer I am mentioning is WaveFlow [9] from 2020, which used dilated convolution to great effect, keeping computation cost down without sacrificing quality compared to WaveNet. It has a small footprint, having about 15 times less parameters than WaveGlow, as well as being able to generate high-quality audio about 40 times faster than real-time. Its flow-based generative synthesis process uses invertible transformations to convert a simple initial density into the complex distribution necessary for speech. The 1-D waveform is transformed into a 2-D matrix during training, then a series of 2-D convolutions are applied on this matrix.

The next major point of discussion is statistical parametric speech synthesis, which is the focus of this thesis. These systems rely on the use of a vocoder to model the characteristics of human vocal cords. Statistical parametric synthesis might be most simply described as generating the average of some set of similarly sounding speech segments. This contrasts directly with the desire in unit

selection to keep the natural, unmodified speech units, but using parametric models offers other benefits [10].

Some of the most prominent vocoders are discussed in [11]. The popular STRAIGHT (Speech Transformation and Representation using Adaptive Interpolation of Weight Spectrum) [12] vocoder was developed to better remove the periodicity effects of the fundamental frequency (F0) on extracting the vocal tract spectral shape. Although it succeeded in solving the "buzzy" problem often found in earlier systems, the number of parameters for both the spectrum and aperiodicity components is the same size as the FFT length used, which is not suitable for statistical modelling.

The Glottal vocoder (Glot) [13] proposed a method to represent the glottal pulse signals instead of using a pulse-train excitation to represent the voiced excitation. Parameters such as energy and harmonic-to-noise ratio (HNR) are calculated to bias the noise component of the source.

While end-to-end methods have shown that they can be the state-of the art technology when it comes to speech synthesis, there are still problems with their widespread applicability. To achieve good quality, training times are often on the order of days, while the database requirements are prohibitively large for some applications. As opposed to the few hours of samples required by most modern statistical parametric synthesis systems, the number of utterances for an end-to-end system is often on the order of 20 thousand, meaning dozens of hours of recorded speech. Therefore, there is an area where the improvement of the existing statistical parametric systems would prove beneficial for applications where time and data are constrained, particularly when it comes to the widespread use of mobile phones in the last decade. Combining neural networks with the efficiency of vocoders, it is possible to meet these demands and expand the possible use cases of speech synthesis. In this work I focus on the exploration of neural architectures for the improvement of statistical parametric speech synthesis systems, with a focus on the possibilities provided by recurrent neural networks.

2 Vocoders

As opposed to end-to-end TTS solutions, statistical parametric speech synthesis takes a different approach by utilizing a vocoder to model certain aspects of the speech generation. Although there are several different types of vocoders that use analysis/synthesis, they follow the same main strategy. The analysis stage is used to convert the speech waveform into a set of parameters which represent separately the vocal-folds excitation signal (sound is voiced or unvoiced) and vocal-tract filter transfer function to filter the excitation signal (vocal-folds movements), whereas in the synthesis stage, the entire parameter set is used to reconstruct the original speech signal. In this section, I discuss two vocoder systems, starting with the open-source WORLD vocoder [14] included in the Merlin TTS toolkit, then the continuous vocoder proposed in [15].

2.1 WORLD vocoder

The Merlin toolkit comes equipped with the WORLD vocoder by default, which is capable of high-quality real-time speech synthesis. It extracts 3 parameters from the analyzed waveform: fundamental frequency (F0), spectral envelope, and aperiodic parameters.

Fundamental frequency is defined as the inverse of the smallest period of a signal. For its estimation in the WORLD vocoder, the DIO [16] algorithm is employed. First, the signal is low pass filtered using different cutoff frequencies. In these filtered signals the candidates for the fundamental frequency are calculated, then a reliability score is associated with them based on their deviation from pure sine waves. Lastly, the candidate with the highest reliability is selected as the F0.

The spectral envelope of the waveform is estimated with CheapTrick [17]. It uses a Hanning-windowed portion of the signal to calculate the power spectrum, from which the Cepstrum is produced. Liftering is applied here to obtain a better estimation, which makes use of the previously obtained F0 from the DIO algorithm.

In the third step, the aperiodic parameters are extracted from the signal. The algorithm used for this is PLATINUM [18]. The signal is first windowed with a window length of two times the fundamental period obtained with the DIO algorithm, then the temporal positions associated with each vocal cord are determined using the fundamental frequency waveform and its contour.

Finally, the speech is synthesized by convolving the minimum phase response with the extracted excitation signal, making use of fewer convolutions than other popular vocoders like STRAIGHT, resulting in better performance with similar end results.

An overview of these steps is displayed on Figure 2.1.

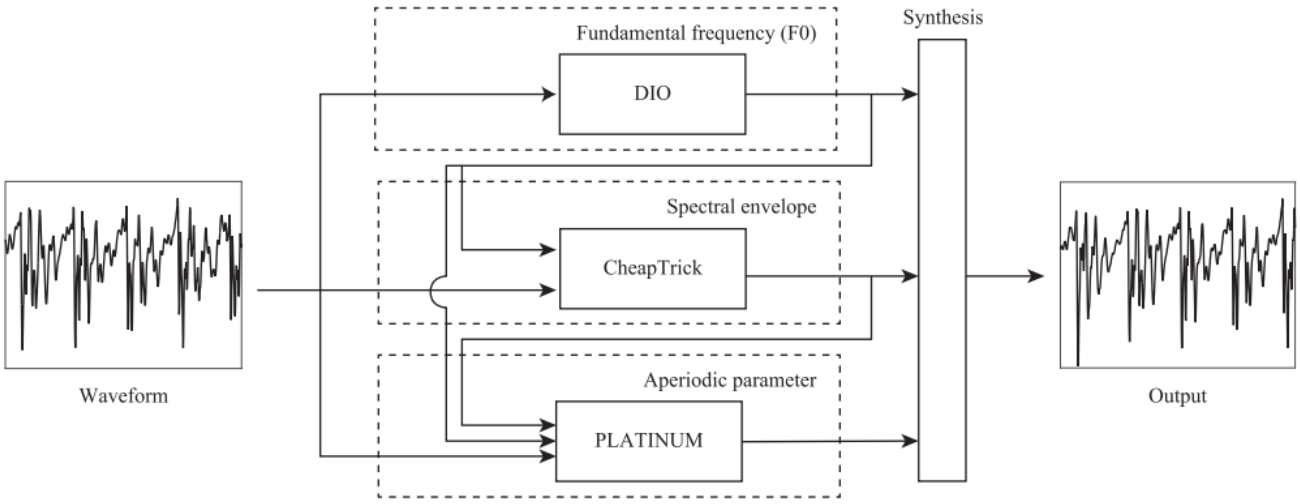

Figure 2.1: An overview of the three analysis steps in the WORLD vocoder system with their connections to each other and the final synthesis step.

2.2 Continuous vocoder

The continuous vocoder proposed at my supervisors' laboratory [15] utilizes a continuous fundamental frequency measurement (contF0), maximum voiced frequency (MVF), and 24-order Mel-generalized cepstral analysis. The implementation of the vocoder is compatible with the Merlin toolkit.

For a better understanding of what is next, the analysis and synthesis phases of continuous vocoder are shown in Fig. 2.2.

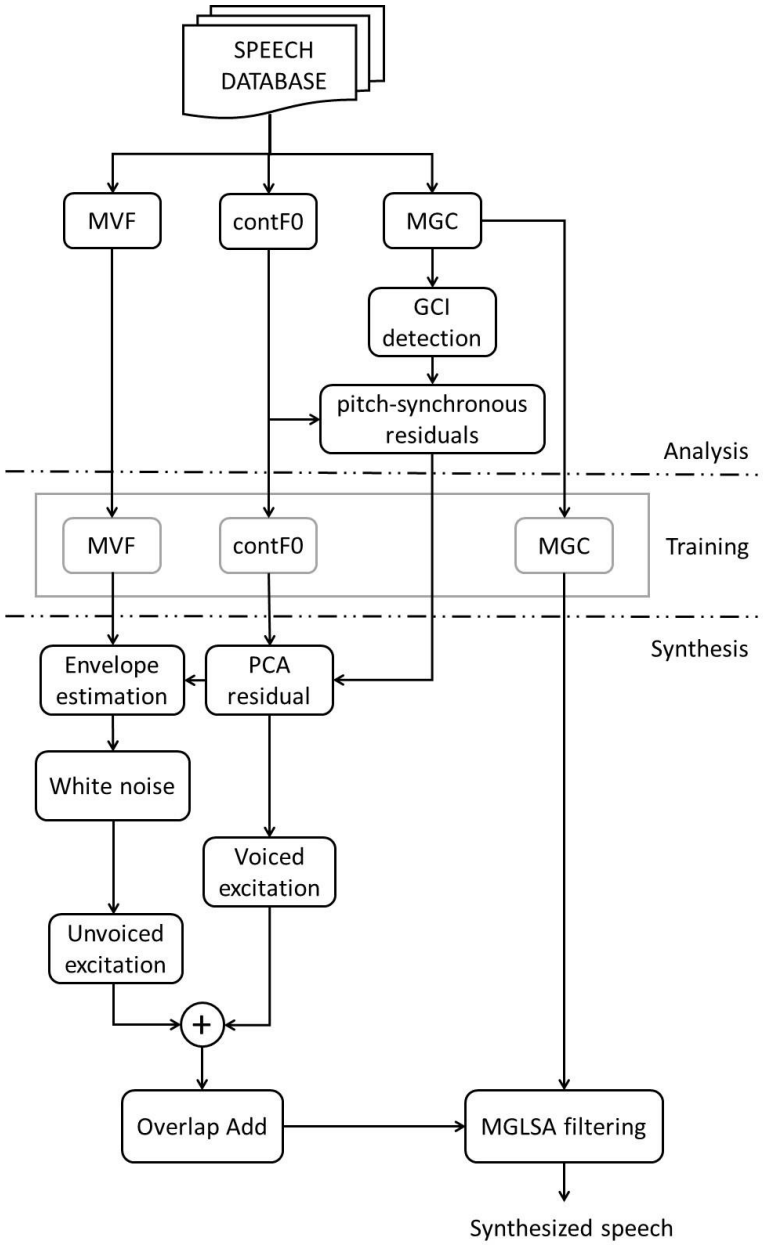

Figure 2.2: Schematic diagram of the analysis and synthesis phase using continuous vocoder.

During the analysis phase, continuous fundamental frequency (contF0) is calculated on the input waveforms using a simple continuous pitch tracker [19]. In areas of creaky voice, and in the event of unvoiced sounds or silences, this pitch tracker interpolates F0 based on a linear dynamic system and Kalman smoothing. Another excitation parameter is the maximum voiced frequency (MVF) which exploits both amplitude and phase spectra that integrated into a maximum likelihood criterion to derive the MVF decisions [20]. Additionally, 24-order Mel-Generalized Cepstral analysis (MGC) [21] is performed on the speech signal with alpha = 0.58 and gamma = 0. The steps of creating the Cepstral plot of a signal are visualized on Figure 2.3, starting with the original waveform at the top, and arriving at its Cepstrum at the last step.

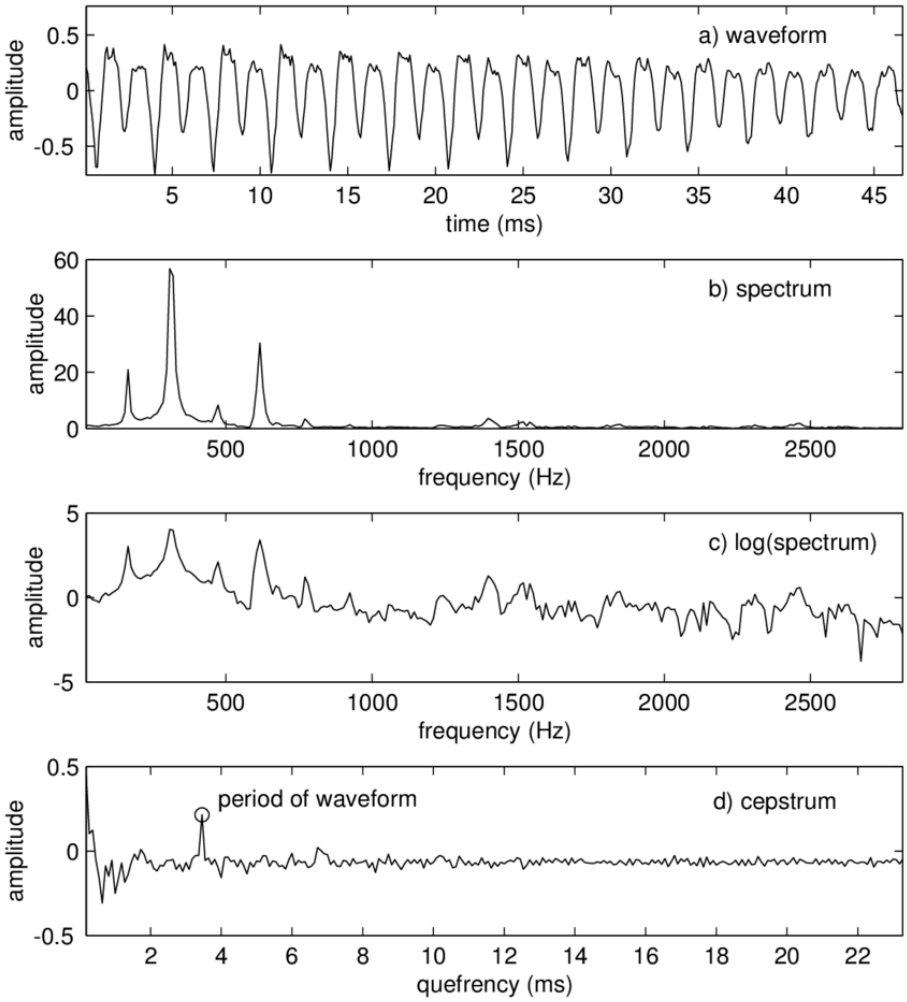

Figure 2.3: Cepstral analysis in four steps.

The frameshift is 5ms and the sampling frequency is 16kHz. The results are the contF0, MVF, and MGC parameter streams. The Glottal Closure Instant (GCI) algorithm [22] is used to find the glottal period boundaries of individual cycles in the voiced parts of the inverse filtered residual signal. From these pitch cycles, a principal component analysis (PCA) residual is finally built which will be used in the synthesis phase. During the synthesis phase, voiced excitation is made of PCA residuals overlap-added pitch synchronously. This voiced excitation is lowpass filtered frame by frame at the frequency given by the MVF parameter. In the frequencies higher than the actual value of MVF, white noise is applied. Voiced and unvoiced excitation is combined, and the MGLSA (Mel-Generalized Log Spectrum Approximation) filter is used to synthesize speech [23]. The continuous vocoder has the obvious advantage of avoiding voicing decision per frame that may be considered to reduce the perceptual degradation caused by voicing decision errors. Moreover, it uses only two one-dimensional parameters for modeling the excitation, which is computationally feasible in the deep neural network-based text-to-speech.

3 Deep Neural Networks

3.1 Feed-forward deep neural networks

In this section, I begin exploration of the different deep neural network (DNN) architectures with feed-forward deep neural networks (FF-DNN). The idea of neural networks stems from how the human brain is structured, with the neurons and connections between them being responsible for a similar function in both cases. The neural model consists of an input and output layer of neurons, and between them there are several hidden layers. Each layer houses neurons with an activation function, the connections between these neurons comprising the weights or parameters of the model. As data is fed through the model, the activations produce an output based on the weighted data on their inputs, and in the case of feed-forward networks, data only travels in one direction, from the input of the model towards the output. A simple example of this architecture is displayed on Figure 3.1.

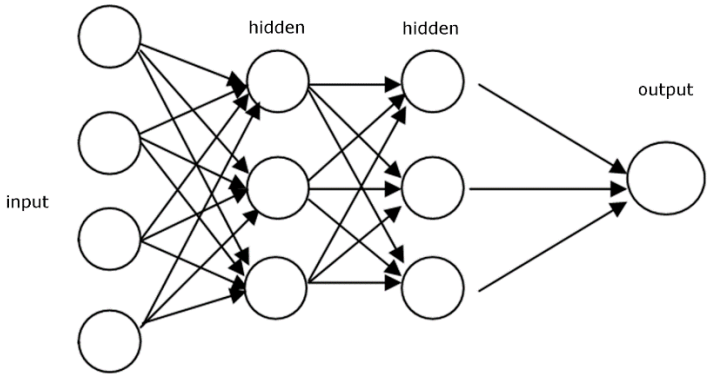

Figure 3.1: An illustration of a simple feed-forward deep neural architecture

The weights of this model can be trained by backpropagating the error (difference between the expected and actual output of the model) through the network, and gradually updating the weights throughout the training epochs to converge on a state in which the accuracy of the model is high. This method is called stochastic gradient descent (SGD), and is one of the many types of optimizers used in training neural networks. The types of optimizers supported by the Merlin toolkit are explained in more detail in Chapter 5.3.

Similar to how the human mind works, inside each neuron is an activation function. This takes the inputs of the neuron, the data weighted with the parameters of the network, sums them up, and produces a single output based on what type of function is used. On Figure 3.2 three commonly used activation functions can be observed.

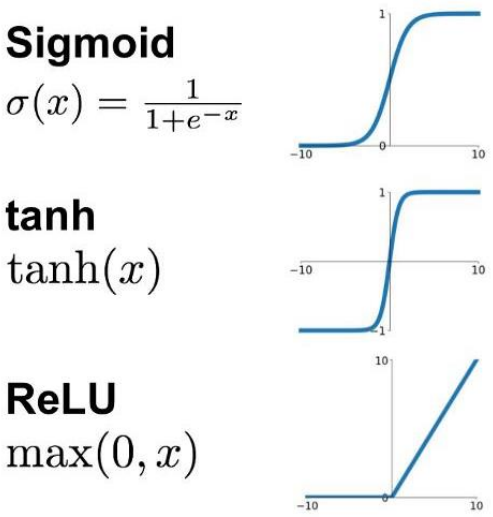

Figure 3.2: Three commonly used activation functions and their respective illustrations.

One important characteristic of these functions is that they can be nonlinear, allowing deep neural networks to approximate complex functions between input data and output, creating the possibility of solving difficult problems with a relatively small number of nodes.

3.2 Recurrent neural networks

While feed-forward neural networks are suited to perform many types of tasks efficiently, they have difficulty extracting longer-term information from the input data, therefore ignoring the sequential nature of speech in the case of spoken language processing. This problem can be solved with the introduction of recurrent neural networks (RNN).

RNNs are a more popular and effective acoustic model architecture which can process sequences of inputs and produce sequences of outputs. In particular, the RNN model is different from the FF-DNN the following way: RNN operates not only on inputs (like the FF-DNN) but also on network internal states that are updated as a function of the entire input history. In this case, the recurrent connections are able to map and remember information in the acoustic sequence, which is important for speech signal processing to enhance prediction outputs.

RNNs vary from feed-forward networks in their hidden layers. Every RNN hidden layer gets inputs not only from its previous layer but also from activations of itself for previous inputs. A basic version of this architecture is displayed in Figure 3.3, in which every node in the hidden layer is connected to the previous activation of every node in that layer.

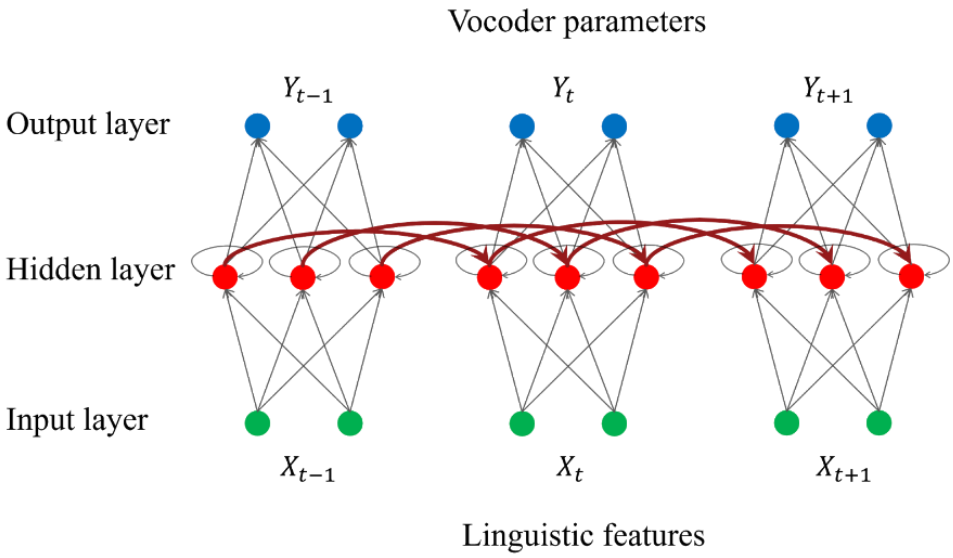

Figure 3.3: A basic version of RNN.

This basic version of recurrent networks is included in the Merlin toolkit, simply referred to as RNN in the configuration files.

There is an important issue with training this type of recurrent network, called the vanishing gradient problem. As we backpropagate the error through the network and update the weights, the recurrent connections between the nodes, more specifically the self-connections, often cause the weight to quickly decrease as we multiply by a small value multiple times. This can be solved for example by stopping the backpropagation early, but in the following section I discuss numerous architectures which are often better suited for extracting sequential information.

3.2.1 Long short-term memory

As originally proposed in [23] and recently used for speech synthesis [24], long short-term memory network (LSTM) is a class of recurrent networks composed of units with a particular structure to cope better with the vanishing gradient problems during training and maintain potential long-distance dependencies. This makes LSTM applicable to learn from history in order to classify, process and predict time series data. Unlike the conventional RNN unit which overwrites its content at each time step, LSTM has a special memory cell with self-connections in the recurrent hidden layer to maintain its states over time, and three gating units (input, forget, and output gates) which are used to control the information flows in and out of the layer as well as when to forget and recollect previous states. LSTM is formulated as follows:

$$i_t = \delta(W_i x_t + R_i h_{t-1} + p_i \odot c_{t-1} + b_i) \quad (1)$$

$$f_t = \delta(W_f x_t + R_f h_{t-1} + p_f \odot c_{t-1} + b_f) \quad (2)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c x_t + R_c h_{t-1} + b_c) \quad (3)$$

$$o_t = \delta(W_o x_t + R_o h_{t-1} + p_o \odot c_t + b_o) \quad (4)$$

$$h_t = o_t \odot tanh(c_t) \quad (5)$$

where i_t, f_t, and o_t are the input, forget, and output gates, respectively; c_t is the so-called memory cell; h_t is the hidden activation at time t; x_t is the input signal; W, and R are the weight matrices applied on input and recurrent hidden units, respectively; p and b are the peep-hole connections and biases, respectively; $\delta(\cdot)$ and $tanh$ are the sigmoid and hyperbolic tangent activation functions, respectively; Θ means element-wise product. A visual representation of this architecture is displayed of Figure 3.4.

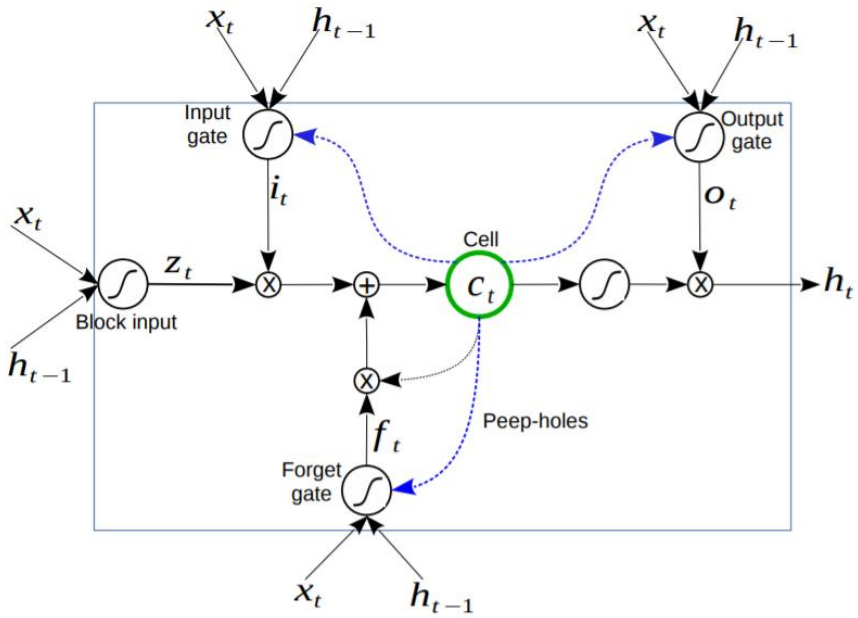

Figure 3.4: A single LSTM cell's architecture.

3.2.2 Bidirectional long short-term memory

While LSTM's ability to discover long-term patterns in the sequential data is beneficial in a lot of cases, it can only consider past data. As originally proposed in [25], adding bidirectionality to the network allows the model to consider not only past, but future data as well, usually leading to

faster training times and better convergence. On Figure 3.5 it can be observed that bidirectional LSTM (Bi-LSTM) makes use of a forward and backward LSTM sequence.

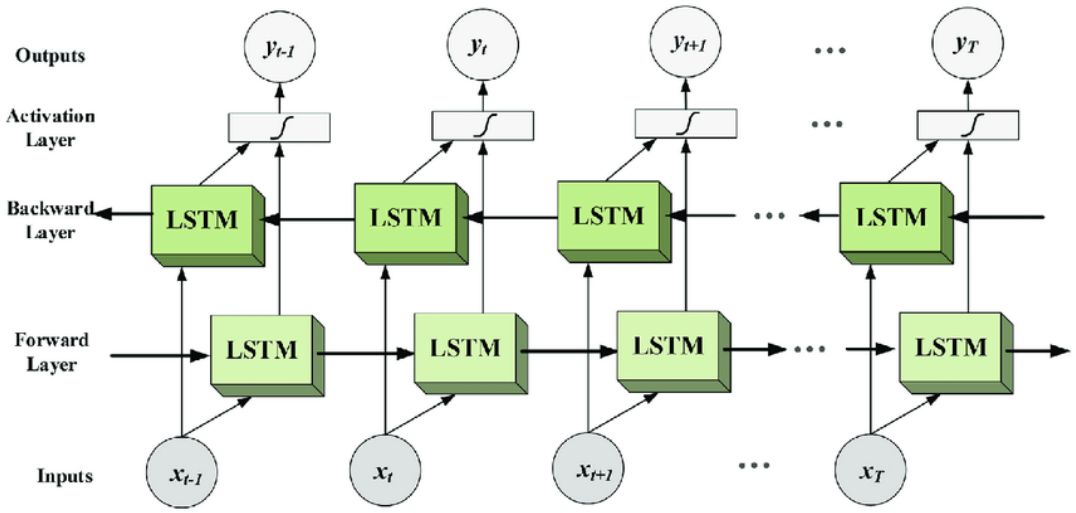

Figure 3.5: The structure of a BiLSTM network.

As is the case with FF-DNN models, we can add depth to the recurrent network by increasing its hidden layer count. LSTM and BLSTM are inherently deep in time, but it appears that adding more recurrent layers on top of each other allows the model to extract information across different time scales, similar to how adding depth in an FF-DNN allows for higher and higher levels of abstraction.

3.2.3 Gated recurrent unit

A slightly more simplified variation of the LSTM, the gated recurrent unit (GRU) architecture was recently defined and found to achieve a better performance than LSTM in some cases [26]. GRU has two gating units (update and reset gates) to modulate the flow of data inside the unit but without having separate memory cells. The update gate supports the GRU to capture long term dependencies like that of the forget gate in LSTM. Moreover, because an output gate is not used in GRU, the total size of GRU parameters is less than that of LSTM, which allow that GRU networks converge faster and avoid overfitting. A diagram of a GRU cell can be observed on Figure 3.6.

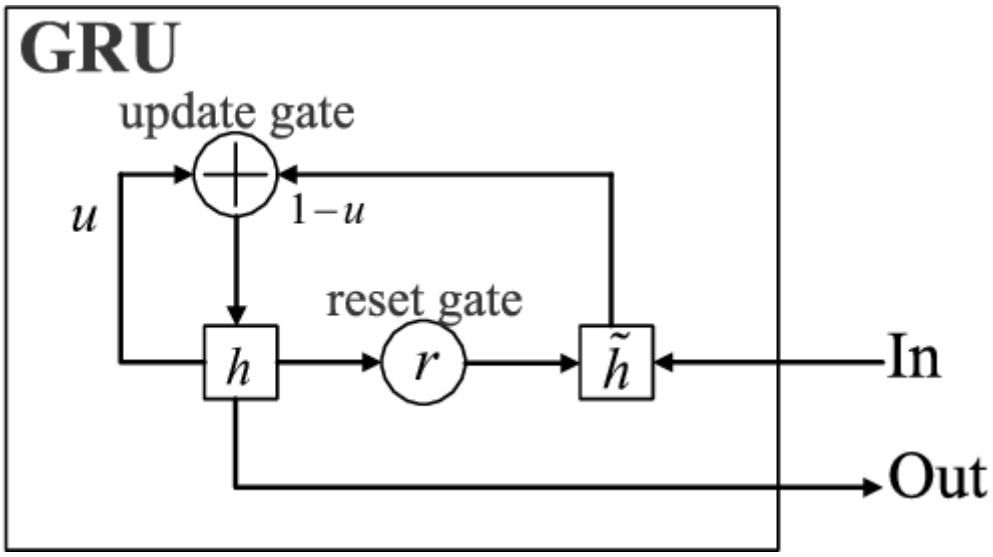

Figure 3.6: Topology of a single GRU cell

As opposed to standard RNN, the update and reset gates here can accurately control the information flow in the network, allowing it to keep long-term information without altering it. The weights of these gates are trainable, of course, and this allows the GRU network to match or even outperform LSTM in many cases while being simpler in terms of parameters and training time.

4 Methodology

In this section, I discuss the proposed approach based on the open-source Merlin TTS toolkit.

4.1 Overview of Merlin

The toolkit was originally proposed in [24] as an open-source benchmark TTS library to be used for testing different neural architectures and vocoders. It is highly extensible, providing easy access to settings thanks to the use of configuration files which store the configuration information in a modular way. The toolkit is written in Python and uses the Theano library as its basis for neural network implementations.

The toolkit uses an external front-end, in this case Festival, to generate the labels for the data, which is then used as the neural network input. Also, as a first step, as we saw in our discussion of vocoders in Chapter 2, parameters are extracted from the data based on the vocoder used.

Two separate neural networks are used in the Merlin toolkit. The first one models the duration characteristics of the speech, and the second one is responsible for the acoustic parameters. Both of these neural networks can be adjusted independently using the configuration files. After training the networks, a vocoder is used to synthesize the speech waveform, making use of the parameters extracted in the first step as well as the outputs of the neural networks. The default vocoder included with the toolkit is WORLD, I will be comparing it to the proposed continuous vocoder through evaluation of speech synthesized with different neural network architectures.

4.2 Database

For evaluation I will be using the CMU-Arctic database [27]. It includes labeled phonetic data for both female (denoted SLT) and male (denoted BDL) speakers. Each speaker produced one hour of speech, which was segmented into 1132 sentences, restricting their length from 5 to 15 words per sentence (a total of 10045 words with 39153 phones) with a sampling rate of 16 KHz. Moreover, CMU-ARCTIC databases contain phonetically balanced utterances with 100% phoneme, 79.6% diphone, and 13.7% triphone coverage, produced by professional speakers experienced in speech processing recordings. The waveform sampling rate of this database is 16 kHz. In the experimentation phase 1000 utterances are used for training, 66 for validation and 66 for testing.

4.3 Experimental conditions

After setting up the basic tools (Festival, necessary python libraries, etc.) Merlin is ready to begin the process of network training and speech synthesis. The first step is to download the database, which in my case was first the SLT, then the BDL speaker. This step also creates a global configuration file, in which basic information is stored relating the type of speaker, vocoder used, as well as the number of files used for training, validation, and for generating the final output. The second step generates the configuration files for the duration and acoustic models. In the third and fourth step, the duration and acoustic models are trained, respectively. Finally, the last step synthesizes the speech.

The default values for all the neural networks include 6 hyperbolic tangent hidden layers of size 1024, a learning rate of 0.02 with exponential decay, and 25 training epochs. Once the values in a configuration file are changed, the corresponding step reads the new values the next time it is run. Table 1 displays the default settings for the parameter extraction, neural architecture and learning process stored in the configuration file.

Table 1: Default neural network settings

Parameter	Value
LF0	1
MVF	1
MGC	60
LF0 dimension	3
MVF dimension	3
MGC dimension	180
Input nodes	425
Output nodes	186
Learning rate	0.002
Training epoch	25
Training utterances	1000
Hidden layer size	1024
Hidden layer type	TANH
Number of hidden layers	6
Output activation	Linear
Acoustic batch size	256
Duration batch size	64
Minimum phase order	511
Frequency warping coefficient	0.58

We can specify the model's architecture (what layers we would like to use and their sizes) in the beginning of the Architecture section in the configuration file. Here it's important to specify a size for each hidden layer, otherwise the program returns an error. After training, the model is stored with the name specified with the model_file_name variable, the synthesis step will use this to load the model from the disk.

The system supports dropout, a technique used to temporarily disable nodes in a layer, forcing the other nodes to take on more or less responsibility for the inputs, often making the network more robust by introducing noise into the training process. The model trains on batches of data, the batch_size variable controls how many samples the model passes through before updating the weights.

We can control the initial learning rate with the learning_rate variable, and a learning rate decay can also be specified here. Decay is often useful, because in the beginning large updates can be made to the weights to decrease the loss quickly, and as the training progresses and we converge on the global minimum, the learning rate can be decreased to fine-tune the weights in the last steps and to avoid "jumping out" of the area of the minimum value. This is illustrated on Figure 4.1, finding the correct value for the learning rate is crucial for optimal convergence.

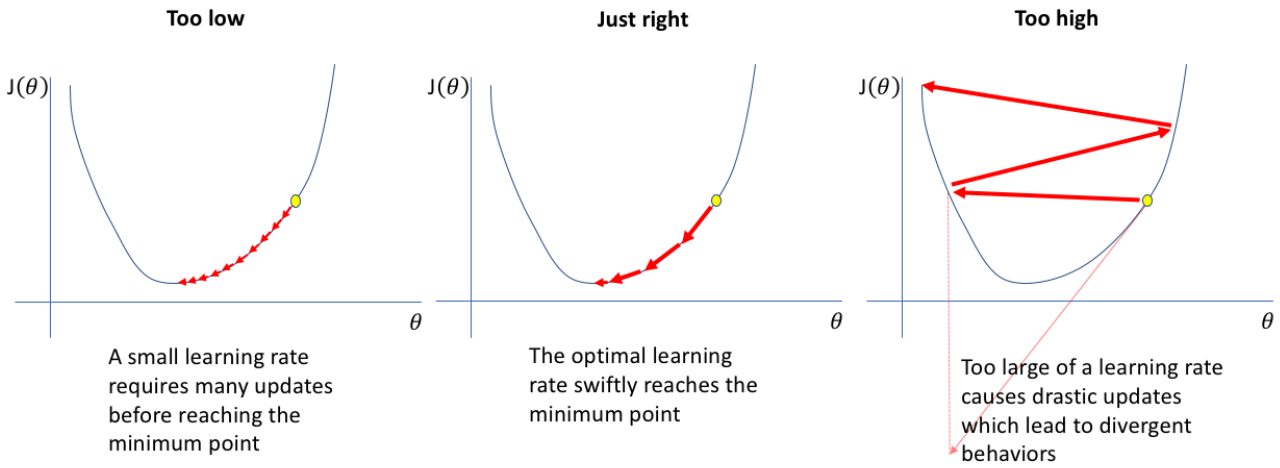

Figure 4.1: The effects of different learning rate values.

We have options for constant learning rate, and linear or exponential decay. The toolkit provides support for three optimizers: stochastic gradient descent (SGD), adam optimizer, and resilient backpropagation (Rprop). Each have their own strengths, but in this case, I concluded that the choice of optimizer has little effect on the outcome of the training, therefore I mostly resorted to experimenting with SGD. Figure 4.2 illustrates how SGD gradually finds the minimum value for the loss function, and how it can end up in different local minimums.

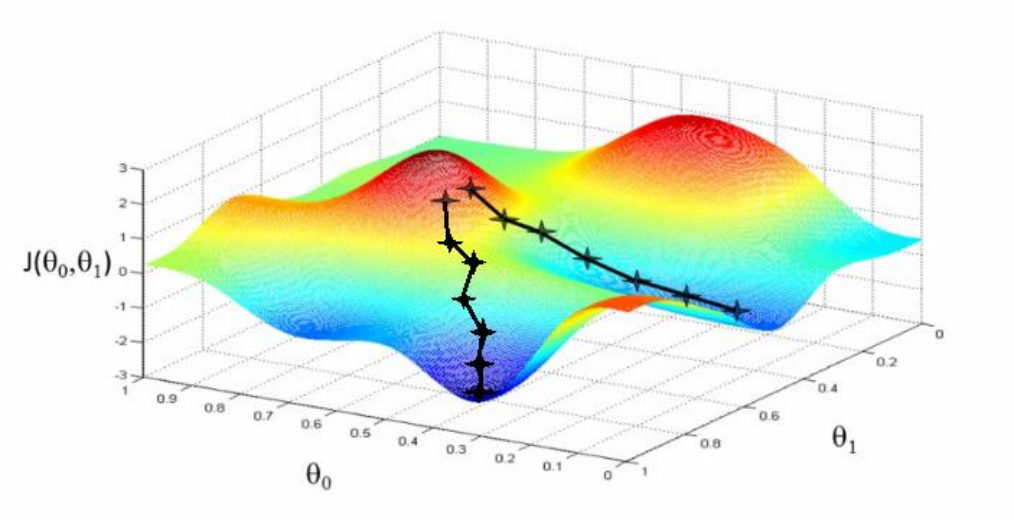

Figure 4.2: Stochastic gradient descent, visualized on 2 input parameters, with the Z axis being the loss function.

Finally, the number of training and warmup epochs can be specified. Warmup is a technique to minimize the effects of finding a set of closely related data in the training dataset, and then having to un-learn the correlations observed in that small subset later. In the warmup period, learning rate is reduced so as to minimize this effect in the beginning, reaching its intended value only at the end of the warmup. The default value for this variable is 10, in our case I found that using 5-8 warmup epochs provided satisfactory results.

After setting up the configuration files to the desired settings, the third, fourth and fifth steps execute the duration and acoustic model training, and the speech synthesis, respectively. During training, information is provided about the state of the execution on the console output.

5 Experimental evaluation

5.1 Experimental goals

In my experimentation I take a look at how the default FF-DNN architecture can be improved upon by the utilization of recurrent neural networks. I also aim to reduce the complexity of the neural networks, making use of fewer hidden layers containing less nodes. Evaluation of the results is based on both objective and subjective metrics, the latter focusing on two standard measurements, intelligibility, and naturalness.

Automated hyperparameter optimization (Bayesian optimization for example) is difficult here, unfortunately, as the results have to be evaluated subjectively first and foremost. Therefore, I conduct a crude grid search, relying on empirical observations about the effects of different settings.

5.2 Observations during experimentation

In this section I discuss the observations made during testing of the different architectures. Starting with feed-forward architectures, I can say that the default settings performed well in the case of the WORLD vocoder, and alterations like swapping the TANH activation to ReLU did not have a large impact on the result. Restricting the size of the hidden layers below 256 was detrimental to the end result, at that point the model was probably experiencing underfitting. When it came to FF-DNN, going below 4 hidden layers was also problematic. The choice of optimizer was of almost no consequence here, and dropout did not improve the results, either. The default 25 training epochs with 10 warmup epochs were satisfactory, with the last 5 epochs usually providing marginal improvement, and the validation error increasing slightly after the warmup period, as is correct. At this point it should also be mentioned that it was noticed that Merlin has support for early stopping, if the validation error starts increasing during consequent epochs (due to overfitting usually), the training will be stopped completely.

When it comes to continuous vocoder, the difference between feed-forward and recurrent architectures was pronounced. As expected, feed-forward networks had subpar performance, however, recurrent structures held more promise. Beginning with LSTM, it was obvious that the recurrent hidden layer size had to be decreased compared to feed-forward layers. Typical sizes were 128 and 256, with 512 being the upper limit that was worth testing. Too many recurrent nodes caused the training process to fail at certain times, with the validation error skyrocketing, or the GPU running out of memory. Adding multiple LSTM layers to the network did not improve results, it seemed best

to experiment with hybrid solutions containing both recurrent and feed-forward layers. Bidirectional-LSTM was an improvement over LSTM, providing slightly better results and faster convergence during the training process. Standard recurrent layers (denoted as RNN in the Merlin configuration file) made the end result sound muddy, it seems like they could not capture the necessary longer-term information like LSTM because of their lack of a memory cell. Gated recurrent units (GRU) were the last type of hidden recurrent layer I experimented with, and they provided excellent results. Thanks to their simpler structure compared to LSTM, the training process was also faster, due to the model having fewer parameters.

During the training of the recurrent networks, it was observed that the adam optimizer performed worse compared to SGD and rprop in many cases, while the difference between the latter two was often negligible. Decreasing the batch size below 128 was also unfavorable, we can assume that the recurrent networks relied on a larger batch of data to extract longer-term information from.

5.3 Developed neural architectures

Table 2 contains the default and the proposed neural architecture for the continuous vocoder. Due to the use of a recurrent hidden layer, the complexity of the network is decreased, with both the size and number of the layers reduced. The hidden layer types and sizes are listed in the order they appear in the neural network. For example, in the case of the default feed-forward network, there are six hyperbolic tangent (TANH) hidden layers of size 1024.

Table 2: The details of the default and proposed neural model architecture

Architecture	Default FF-DNN	Proposed hybrid RNN
Hidden layer sizes	1024, 1024, 1024, 1024, 1024, 1024	256, 512, 1024, 1024, 1024
Hidden layer types	TANH, TANH, TANH, TANH, TANH, TANH	GRU, RELU, TANH, TANH, RELU
Dropout	0.0	0.0
Batch size	256	256
Learning rate decay	-1 (exponential decay)	1 (linear decay)
Learning rate	0.002	0.01
Optimizer	Stochastic gradient descent (sgd)	Stochastic gradient descent (sgd)
Warmup epochs	10	8
Total number of epochs	25	30

5.4 Objective evaluation

5.4.1 Logarithmic spectral distance

For objective evaluation of the results, the spectral distortion is designed to compute the distance between two power spectra, the root mean square (RMS) log spectral distance (LSD) metric is suggested here to carry out the evaluation as seen on equation (6).

$$LSD_{RMS} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} mean\left(logP(f_k) - log\hat{P}(f_k)\right)^2} \quad (6)$$

P(f) is the spectral power magnitude of the natural speech, while $\hat{P}$(f) is the spectral power magnitudes of the synthesized speech. The optimal value of LSD_{RMS} is zero, which indicates a matching frequency content. I also used a set of example spectrograms for the synthesized voices, to see whether speech synthesis results captured the natural speaker. The results are shown in Figure 5.1, and I found that the proposed framework has a lower LSD_{RMS} equal to 1.8 dB that is closer to the original speech spectrogram than the WORLD model. Consequently, the proposed system introduces a smaller distortion to the sound quality and approaches a correct spectral criterion.

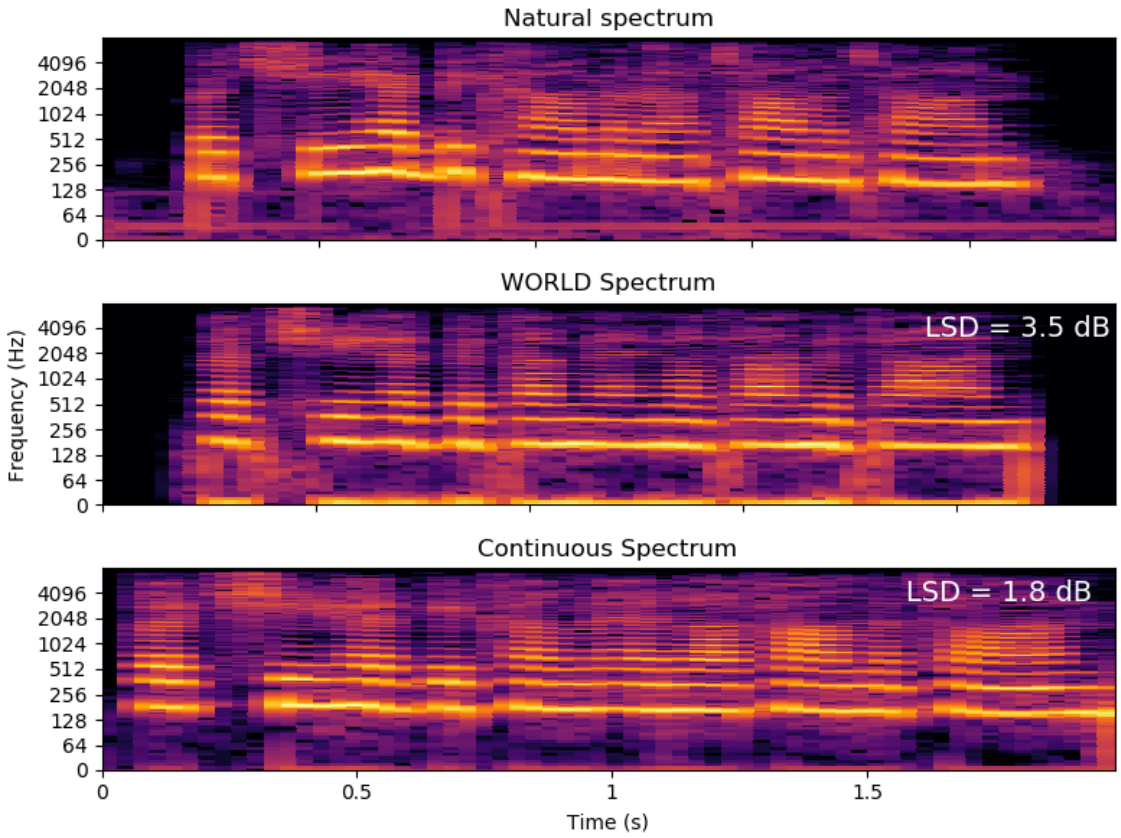

Figure 5.1: Example of the spectrogram based on RNN-TTS for natural speech, and synthesized speech from the continuous and WORLD vocoders.

Here it should be mentioned that during the continuous vocoder-based synthesis the silence from the beginning and end of the file is cut, hence the difference in the spectrum images in those areas.

5.4.2 Spectrograms

For additional evaluation, I present an overview of the spectrograms for the original and the synthesized speech for both the male (BDL) and female (SLT) speaker. For the male speaker, the spectrograms are based on the arctic_a0001.wav file from the database. On Figures 5.2 and 5.3, the waveforms as well as the spectrograms of the speech samples can be compared visually.

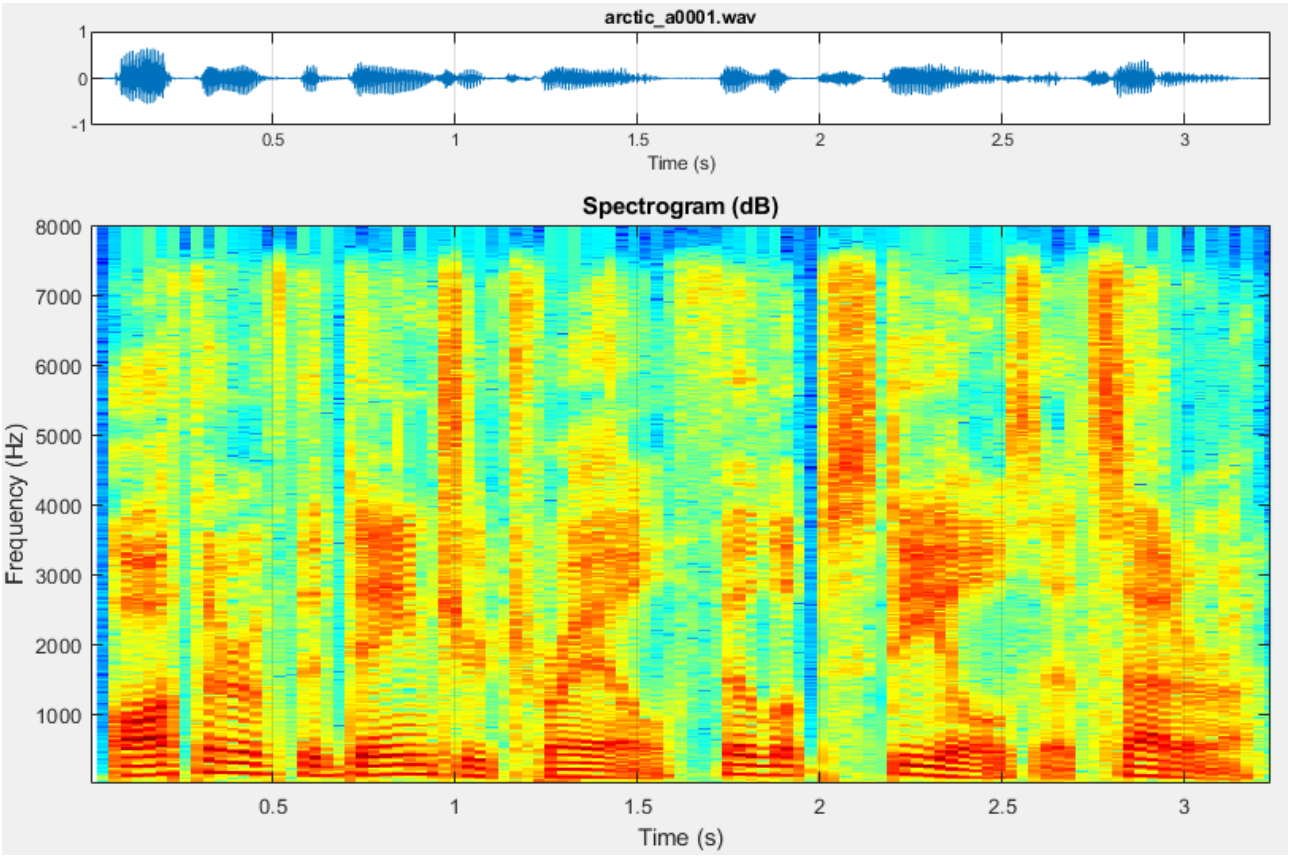

Figure 5.2: Spectrogram for the original speech waveform for the male speaker.

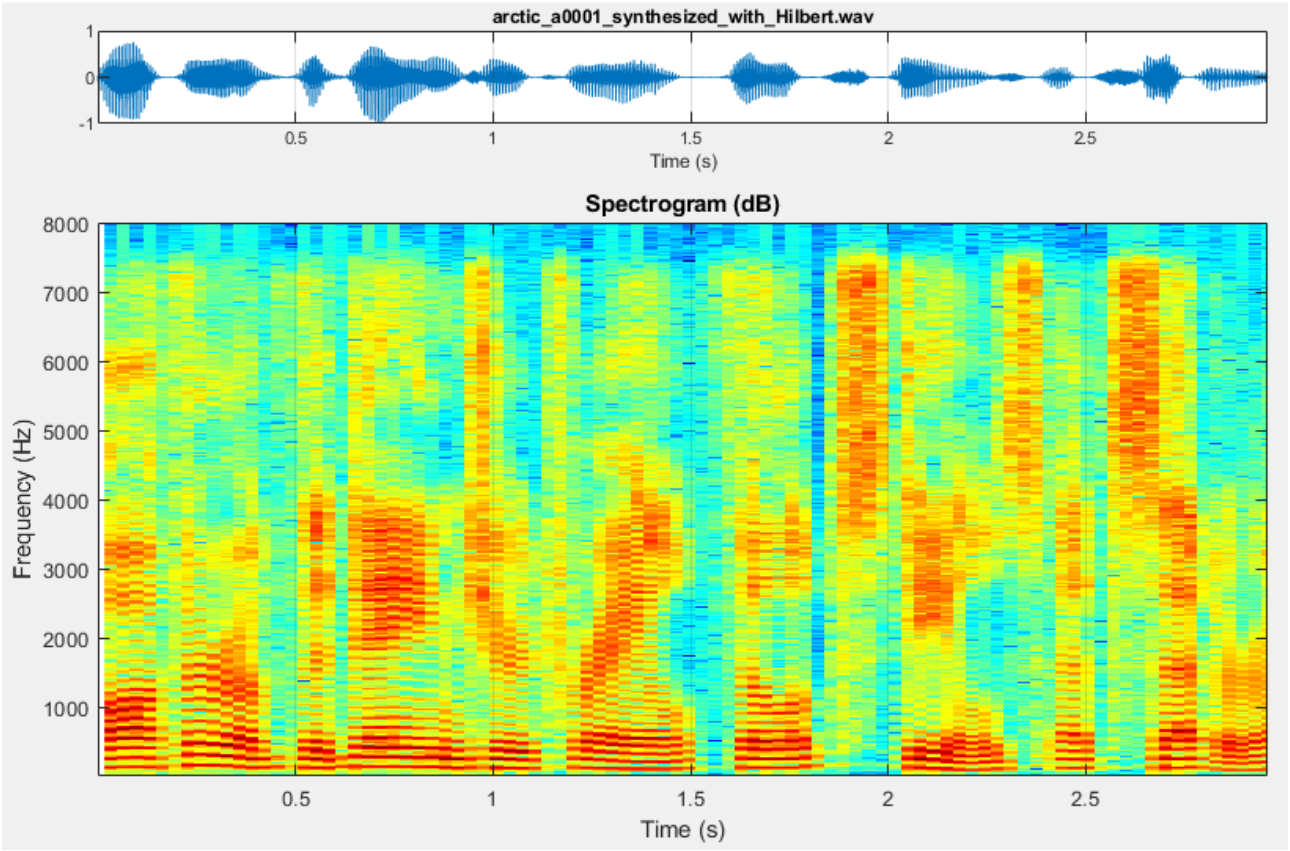

Figure 5.3: Spectrogram of speech waveform synthesized with continuous vocoder for male speaker.

For the female speaker, I choose the arctic_b0535.wav file, as this voice line originally contained a good example of different intonations in the same sentence. On figures 5.4 and 5.5, the waveforms and spectrograms for the original and synthesized speech are displayed.

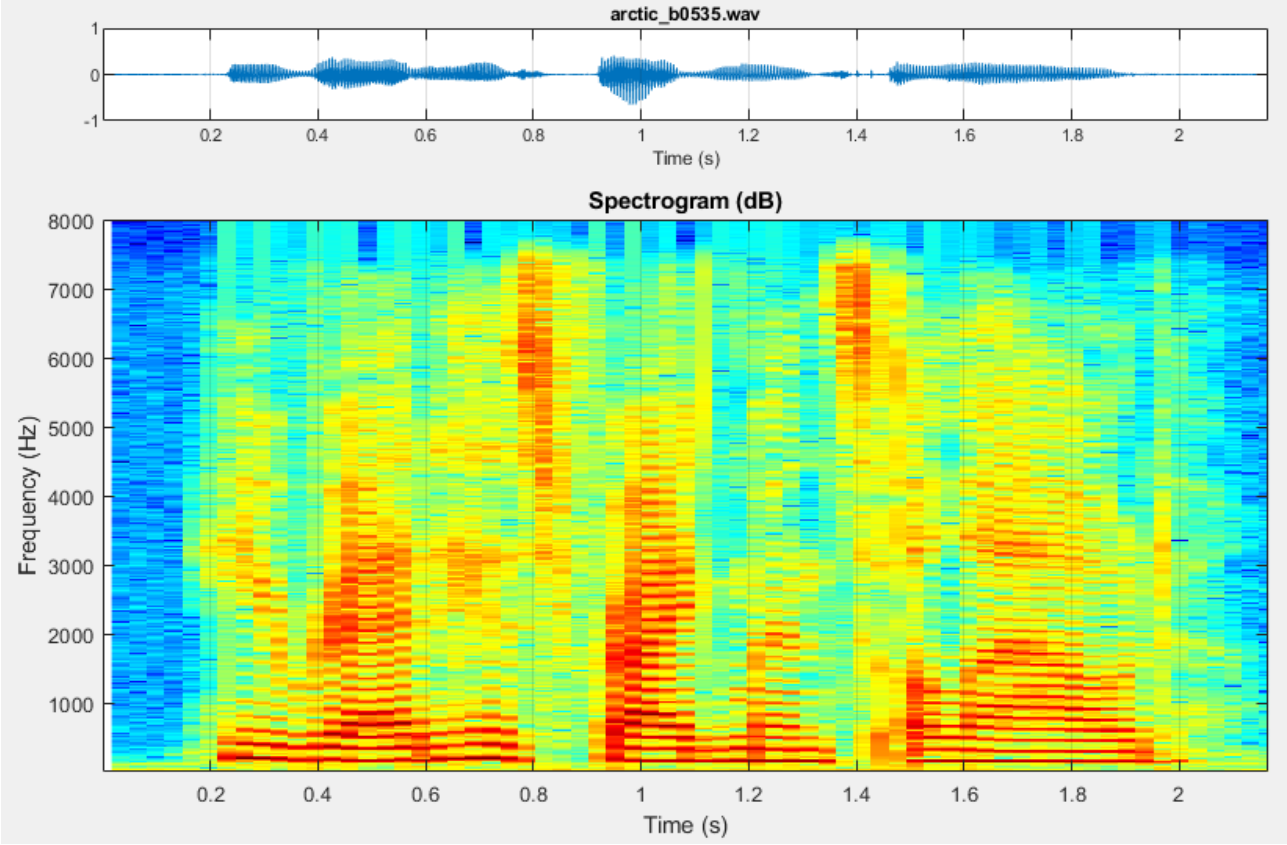

Figure 5.4: Waveform and spectrogram of the original speech sample for the female speaker.

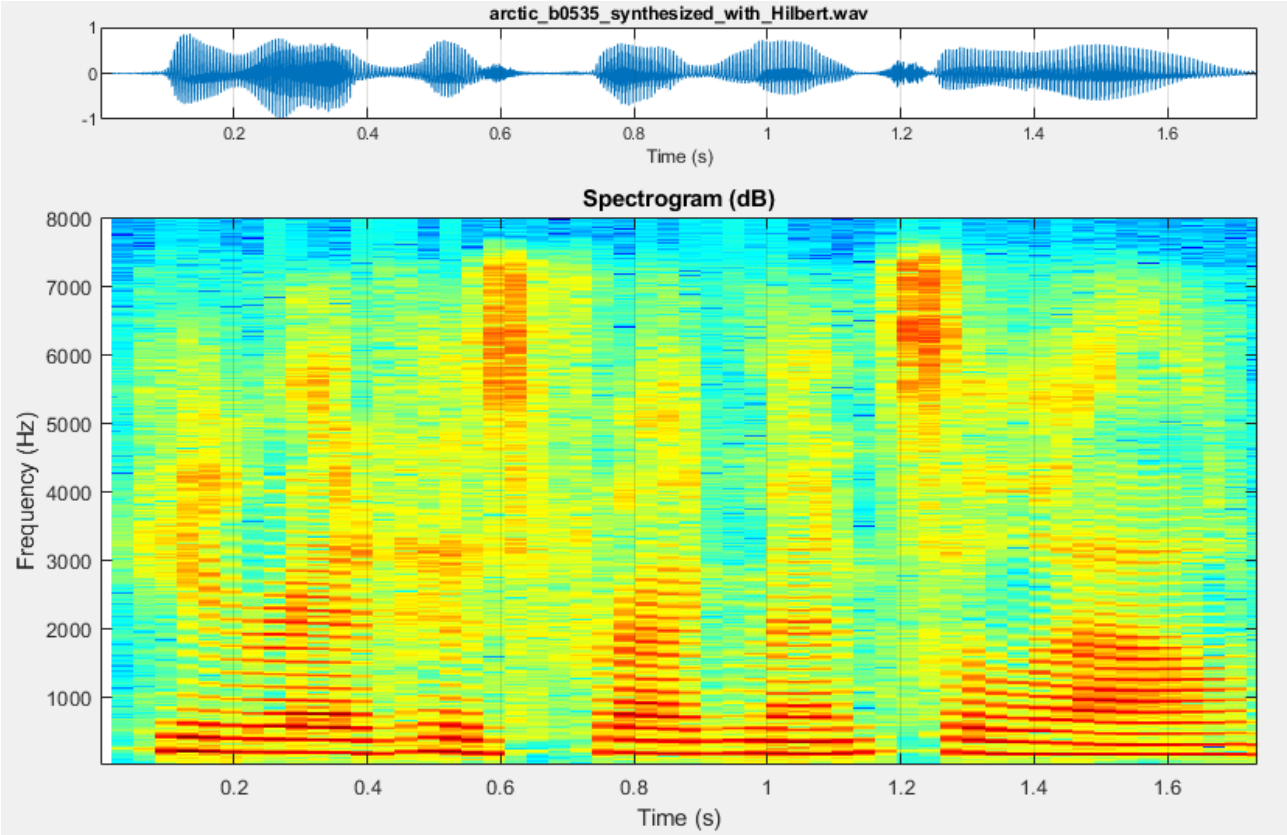

Figure 5.5: Waveform and spectrogram of the speech sample synthesized with continuous vocoder for the female speaker.

Looking at the waveforms first, it can be noted that much of the silence is removed from the beginning in the synthesized speech. In general, the amplitude is increased in the generated speech, this is consistent with the increase in volume that can be noted upon listening to the samples.

The spectrograms show that most of the energy is located in the lower frequencies of the speech waveforms, specifically, below 1000 Hz. In this region, the harmonics in the synthesized speech are more clearly defined, leading to the usual "robotic" sound that characterizes speech synthesis systems. That said, the higher frequency content is modeled well, with the most difference showing around 3-4000 Hz.

It should be noted that due to the removal of the silence from certain parts of the waveform, the temporal duration of the samples also shrunk, this should be taken into account when examining the spectrograms.

5.4.3 Built-in metrics

The Merlin toolkit provides built-in metrics to provide information during training of the neural networks. To get a general idea of how the neural network based on the continuous vocoder

performs against the WORLD-based FF-DNN and RNN, the following key metrics are extracted from the training process.

The Mel-Cepstral Distortion (MCD) is used to measure the difference between two cepstra, in this case, the cepstra of the original recording and the features at the end of the learning process. MCD is formulated as seen on equation 7.

$$MDC = \frac{1}{N}\sum_{j=1}^{N}\sqrt{\sum_{i=1}^{K}(x_{i,j} - y_{i,j})^2} \quad (7)$$

Here $x_{i,j}$ and $y_{i,j}$ are the $(i,j)^{th}$ cepstral coefficients of the natural and the generated speech, respectively. In this case, the proposed model matched the performance of the default feed-forward network based on the WORLD vocoder.

Correlation measures the degree to which the original and generated data are linearly related. It is formulated as follows:

$$CORR = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \quad (8)$$

Where $\bar{x}$ and $\bar{y}$ are the mean of the natural x_i and synthesized y_i speech frames; respectively. In this metric the proposed system almost matched the result of the WORLD FF-DNN, surpassing the WORLD RNN model.

On Table 3 I show these metrics as output by the fourth, acoustic training model step.

Table 3: Training metrics for the WORLD FF-DNN, WORLD RNN and Continuous RNN architectures.

Metric/architecture	WORLD FF-DNN	WORLD RNN	Continuous RNN
Mel-Cepstral Distortion (lower is better)	4.903 dB	7.128 dB	4.903 dB
Correlation (higher is better)	0.77	0.657	0.712

5.5 Subjective evaluation

Due to time constraints, subjective evaluation was carried out on a mean opinion score (MOS) basis on the female (SLT) speaker. 5 subjects were asked to rate the batches of samples (each containing the same 5 sentences synthesized with different settings) and judge their naturalness and intelligibility on a 0-100 scale, with 100 being the baseline of the original human speaker. As

expected, the recurrent network with the WORLD vocoder and the feed-forward network with the continuous vocoder showed poor results. The recurrent network for the continuous vocoder, however, received good scores, trading some intelligibility for slightly improved naturalness. When asked, most of the participants said that they preferred the intonation of the speech produced by the continuous vocoder, however, there appears to be some added noise or "buzzing" to the speech, which might be the cause of the lower intelligibility score. This is the typical robotic sound produced by many speech synthesis systems. The volume of the speech produced by the continuous vocoder was also higher than the one produced by WORLD, which could also contribute to the more pronounced noisiness. On Table 2 the results of the MOS evaluation are shown.

Table 4: Sound quality of synthesized speech based on MOS.

MOS	Natural	WORLD		Continuous	
	DNN/RNN	FF-DNN	RNN	FF-DNN	RNN
Naturalness	100	62	44	42	64
Intelligibility	100	70	52	44	60

6 Challenges, conclusions, and future research

One of the challenges faced during the evaluation of the proposed architectures was that some of the code in the Merlin toolkit was outdated and needed to be updated to modern standards (update to Python3, replace deprecated libraries). Most of the code was written in 2016, and while having support from python 2.6 to 3.6 means flexibility, it also introduced some problems on our virtual environment. With these issues solved, however, testing could be conducted relatively quickly thanks to the usage of a high-performance GPU, although the requirement to subjectively evaluate each result proved automation of the process difficult. Overall, I showed the possibility and advantage of using the proposed continuous vocoder as a viable alternative to WORLD through the example of a compact recurrent neural architecture.

In the future, there is possibility to improve on the automation process, with the use of the Bayesian optimizer to not only adjust the common hyperparameters of the learning process, but also to find the optimal architecture by including the order, size, and type of the hidden layers in the parameters of the optimizer. This would require setting up a robust and automated objective evaluation process, using an expanded version of the objective metrics showed in this paper.

Publications

[C1] Suciu Barnabás, Mohammed Salah Al-Radhi, Exploring Efficient Neural Architecture for Statistical Parametric Text-to-Speech Synthesis, Speech Synthesis Workshop (SSW11), 2021, **Submitted**.

Conference page: https://ssw11.hte.hu/en/

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Mohammed Salah Al-Radhi of the Faculty of Electrical Engineering and Informatics at the Budapest University of Technology and Economics. His help was invaluable in the completion of this thesis work, I could count on his advice any time I had stumbled upon an issue. Also, I thank Dr. Tamás Csapó Gábor for his help regarding the testing environment and the GPU settings there. The high-performance Titan X GPU used to conduct testing in this paper was provided by NVIDIA Corporation. Furthermore, I would like to thank the participants of the subjective listening test for their detailed feedback.

References

[1] Hunt A., Black A., “Unit selection in a concatenative speech synthesis system using a large speech database,” in Proceedings of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Atlanta, USA, pp. 373-376, 1996.

[2] Zen H., Tokuda K., Black A., “Statistical parametric speech synthesis,” Speech Communication, vol. 51, no. 3, pp. 1039-1064, 2009.

[3] Zen H., Toda T., Nakamura M., Tokuda K., “Details of the Nitech HMM-Based Speech Synthesis System for theBlizzard Challenge 2005,” IEICE Transactions on Information and Systems, vol. E90–D, no. 1, pp. 325-333, 2007.

[4] Zen H., Senior A., Schuster M., “Statistical parametric speech synthesis using deep neural networks” in Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Vancouver, Canada, p. 7962–7966, 2013.

[5] Oord A.V., Dieleman S., Zen H., Simonyan K., Vinyals O., Graves A., Kalchbrenner N., A. Senior, and Kavukcuoglu K., "WaveNet: A generative model for raw audio," arXiv preprint arXiv:1609.03499, 2016.

[6] Arik S. O., Chrzanowski M., Coates A., Diamos G., Gibiansky A., Kang Y., Li X., Miller J., Ng A., Raiman J., Sengupta S., Shoeybi M., “Deep Voice: Real-time Neural Text-to-Speech”, Proceedings of the 34th International Conference on Machine Learning (ICML), Stockholm, Sweden, pp. 195-204, 2017.

[7] Arik S. O., Wang Y., Skerry-Ryan R., Stanton D., Wu Y., Weiss R.J., Jaitly N., Yang Z., Xiao Y., Chen Z., Bengio S., Q. Le, Agiomyrgiannakis Y., Clark R., and Saurous R.A., "Tacotron: Towards End-to-End Speech Synthesis," in: Proceedings of the Interspeech, Stockholm, Sweeden, pp. 4006-4010, 2017.

[8] Prenger R., Valle R., Catanzaro B., “WaveGlow: A Flow-based Generative Network for Speech Synthesis” Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, pp. 3617-3621, 2019.

[9] Wei P., Kainan P., Kexin Z., Zhao S., “WaveFlow: A Compact Flow-based Model for Raw Audio”, arXiv:1912.01219v4 [cs.SD] 24 Jun 2020.

[10] Alan W. B., Heiga Z., Keiichi T., “Statistical Parametric Speech Synthesis”, Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA, Department of Computer Science and Engineering, Nagoya Institute of Technology, Nagoya, JAPAN, Speech Communication Volume 51, Issue 11, pp. 1039-1064, November 2009

[11] Qiong H., Korin R., Junichi Y., Javier L., “An experimental comparison of multiple vocoder types”, The Centre for Speech Technology Research,University of Edinburgh, U.K., Toshiba Research Europe Ltd, Cambridge, U.K., National Institute of Informatics, Tokyo, Japan, SSW8, 135-140.

[12] Kawahara H., Masuda-Katsuse I., de Cheveigne A., “Restructuring speech representations using a pitch-adaptive time–frequency smoothing and an instantaneous-frequency-based F0 extraction: Possible role of a repetitive structure in sounds,” Speech Communication, vol. 27, no. 3 , pp. 187-207, 1999.

[13] Raitio T., Suni A., Yamagishi J., Pulakka H., Nurminen J., Vainio M., and Alku P., “HMM-based speech synthesis utilizing glottal inverse filtering,” IEEE Trans. on Audio, Speech, and Lang. Proc., vol. 19, no. 1, pp. 153–165, Jan. 2011.

[14] Morise M., Yokomori F., and Ozawa K., "World: A vocoder based high-quality speech synthesis system for real-time applications," IEICE Transactions on Information and Systems, Vols. E99-D, no. 7, pp. 1877–1884, 2016.

[15] Al-Radhi M. S., Csapó T. G., Németh G., “High-Quality Vocoding Design with Signal Processing for Speech Synthesis and Voice Conversion”, Budapest, Hungary, 2020

[16] Morise M., Kawahara H., and Katayose H., “Fast and reliable f0 estimation method based on the period extraction of vocal fold vibra- 1884 IEICE TRANS. INF. & SYST., VOL.E99–D, NO.7 JULY 2016 tion of singing voice and speech,” in Proc. AES 35th International Conference, CD-ROM Proceedings, 2009.

[17] Morise M., “Cheaptrick, a spectral envelope estimator for high-quality speech synthesis,” Speech Communication, vol.67, pp.1–7, 2015.

[18] Morise M., “Platinum: A method to extract excitation signals for voice synthesis system,” Acoust. Sci. & Tech., vol.33, no.2, pp.123–125, 2012.

[19] Garner P.N., Cernak M., and Motlicek P., “A simple continuous pitch estimation algorithm,” IEEE Signal Processing Letters, vol.20, no.1, pp.102–105, 2013.

[20] Drugman T. and Stylianou Y., “Maximum voiced frequency estimation: exploiting amplitude and phase spectra,” IEEE Signal Process. Lett., vol.21, no.10, pp.1230–1234, 2014.

[21] Tokuda K., Kobayashi T., Masuko T., and Imai S., “Mel-generalized cepstral analysis: A unified approach to speech spectral estimation,” Proc. International Conference on Spoken Language Processing, Yokohama, Japan, pp.1043–1046, 1994.

[22] Drugman T., Thomas M., Gudnason J., Naylor P., and Dutoit T., “Detection of glottal closure instants from speech signals: A quantitative review,” IEEE Trans. Audio, Speech, Language Process., vol.20, no.3, pp.994–1006, 2012.

[23] Hochreiter S., Schmidhuber J., “Long short-term memory,” Neural computation, vol. 9, no. 8, pp. 1735-1780, 1997.

[24] Zhizheng W., Watts O., King S., “Merlin: An Open Source Neural Network Speech Synthesis System,” in Proceeding 9th ISCA Speech Synthesis Workshop (SSW9), California, USA, 2016.

[25] Schuster M., Paliwal K., “Bidirectional recurrent neural networks,” IEEE Transactions on Signal Processing, vol. 45, no. 11, pp. 2673-2681, 1997.

[26] Chung J., Gulcehre C., Cho K., Bengio Y., “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” arXiv preprint: 1412.3555, 2014.

[27] Kominek J., Black A.W., “CMU ARCTIC databases for speech synthesis,” Carnegie Mellon University, 2003.