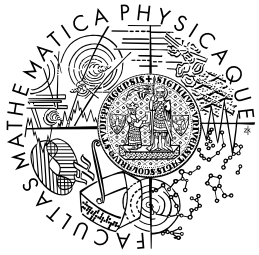

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Tomáš Nekvinda

Multilingual speech synthesis

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Mgr. et Mgr. Ondřej Dušek, Ph.D.
Study programme: Computer Science
Study branch: Artificial Intelligence

Prague 2020

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank my supervisor Ondřej for guidance, advice, patience, willingness, and all the helpful consultations, and my room-mate David for his great ideas and debugging sessions. Of course, I have to thank you all who provided me with food supplies and did not disturb me while I was pretending that I do something useful.

Title: Multilingual speech synthesis

Author: Bc. Tomáš Nekvinda

Institute: Institute of Formal and Applied Linguistics

Supervisor: Mgr. et Mgr. Ondřej Dušek, Ph.D., Institute of Formal and Applied Linguistics

Abstract: This work explores multilingual speech synthesis. We compare three models based on Tacotron that utilize various levels of parameter sharing. Two of them follow recent multilingual text-to-speech systems. The first one makes use of a fully-shared encoder and an adversarial classifier that removes speaker-dependent information from the encoder. The other uses language-specific encoders. We introduce a new approach that combines the best of both previous methods. It enables effective parameter sharing using a meta-learning technique, preserves encoder's flexibility, and actively removes speaker-specific information in the encoder. We compare the three models on two tasks. The first one aims at joint multilingual training on ten languages and reveals their knowledge-sharing abilities. The second concerns code-switching. We show that our model effectively shares information across languages, and according to a subjective evaluation test, it produces more natural and accurate code-switching speech.

Keywords: text-to-speech, speech synthesis, multilinguality, natural language processing, deep learning

Contents

Introduction

Deep learning has become a dominant field of artificial intelligence and machine learning. Contemporary models achieve state-of-the-art results in many areas of computer vision or natural language processing, and speech synthesis is not an exception. Current deep learning models can synthesize natural-sounding speech in real time [Kalchbrenner et al., 2018], and make possible an efficient end-to-end training that does not put high demands on quality, amount, and preprocessing of training data [Wang et al., 2017]. Based on these advances, speech processing researchers aim at, for example, expressiveness [Wang et al., 2018], controllability [Hsu et al., 2019], or few-shot voice cloning [Jia et al., 2018].

Many studies in the field of neural machine translation concern the possibilities of transfer learning or multilingual training, i.e., joint training of a single model using multiple language pairs. In general, multilingual training has some advantages over transfer learning. First, it can maintain performance while reducing the total number of models that need to be stored. Second, it can utilize data from multiple languages [Sachan and Neubig, 2018] and it can enable functionality such as one-to-many or many-to-many translation [Platanios et al., 2018].

As is the case with neural machine translation, current speech synthesis does not leave multilingual models aside. Chen et al. [2019] explore possibilities of adaptation of a pre-trained monolingual model to different languages using transfer learning, and Zhang et al. [2019] or Cao et al. [2019] successfully performed cross-lingual voice cloning and code-switching, respectively. To the best of our knowledge, the majority of current multilingual research evaluates simultaneous synthesis of just two or three languages.

Goal & Contribution

The goal of this work is to implement a multilingual system for speech synthesis based on neural networks, evaluate its performance, and compare it with monolingual or other multilingual approaches.

We examine the principle of multilingual training known from neural machine translation in the context of speech synthesis. At first, we introduce our reimplementation of a state-of-the-art text-to-speech model. Secondly, we compare three of our own models with various levels of cross-lingual parameter sharing on ten languages (namely on German, Greek, Spanish, Finnish, French, Hungarian, Japanese, Dutch, Russian and Chinese). Moreover, we mention multilingual training in low-resource situations. In the end, we experimentally compare the code-switching abilities of the three models on German, French, Dutch, Russian, and Chinese. For the training of these models, we created a new size-reduced

but cleaned multilingual and multi-speaker dataset. The source code, evaluation data, pre-trained models, and audio samples are available at the GitHub repository of this work.[1]

Outline

Chapter 1 provides signal processing and deep learning basics. In Chapter 2, we describe the historical development of speech synthesis, and we introduce contemporary influential text-to-speech models. We pay special attention to the architectures that we used in our experiments. Chapter 3 briefly mentions metrics and methodologies for speech synthesis evaluation. We discuss monolingual and multilingual datasets in Chapter 4. A summary of related work is provided in Chapter 5. The last part of this work, namely Chapter 6, describes our original experiments, their setup, and obtained results.

[1] https://github.com/Tomiinek/Multilingual_Text_to_Speech

Chapter 1

Preliminaries

In this chapter, we primarily describe some terms of signal processing which are a must to understand the further reading. The second section of this chapter groups deep learning terms and provides their brief explanation. Please note that a detailed description of these topics is not the real subject of this work. For more details, we refer the reader to Jurafsky and Martin [2014], who talk about various topics related to natural language processing, Li and Cox [2019], who describe acoustical engineering and low-level audio processing, and finally Goodfellow et al. [2016], who provide solid basics of deep learning.

1.1 Audio Signal Processing

Sound is a complex series of pressure changes propagated through a medium such as air, liquid, etc. We can view sound as mechanical waves. During the propagation of a wave, particles of the medium vibrate about their mean position and transfer their energy to neighboring particles. We can describe these sound waves using *frequency* f (speed of repetition), *amplitude* A, and *phase* ϕ. The following simple equation holds in the case of a sinusoidal wave:

$$x(t) = A \cdot \sin(2\pi f t + \phi)$$

where x is the distance from a mean position, and t is a particular time. When a pressure change or better said a wave reaches a sensor such as a microphone or an ear, the mechanical energy is converted to another form such as an electric signal. It is well known that only waves with frequencies between roughly 20Hz and 20kHz can elicit an auditory percept by humans.

Unfortunately, real-world sound waves are not as simple as sinusoids. However, we can measure the compression or the rarefaction of molecules of the medium to describe them. The number of measurements must be finite so we usually repeatedly and regularly sample the amplitude and thus convert the analog signal into a digital one. The number of samples we measure per second is called *sample rate*. Typically, a sample rate greater than 20 kHz is used in speech synthesis tasks because human speech is usually below 10 kHz and thus lower sample rates might not be sufficient for complete accuracy (according to the Shannon-Nyquist sampling theorem). A graph describing amplitude as a function in the *time domain* is called *waveform*, see Figure 1.1.

To reduce storage requirements, it can be beneficial to represent sampled am-

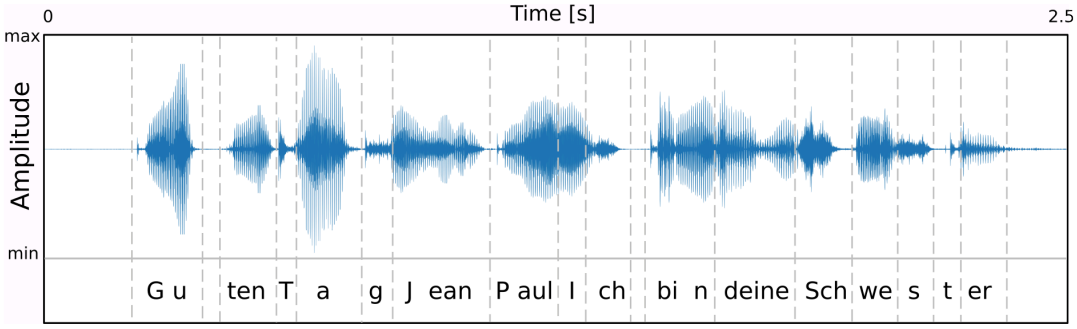

Figure 1.1: *Visualization of a speech waveform (blue color) together with an approximate alignment of characters being pronounced. The horizontal axis corresponds to the time domain and the vertical axis depicts the amplitude.*

plitudes as integers instead of real-valued numbers. This conversion is called *quantization* and it is in general lossy. Usually 8-bit or 9-bit integers are enough to represent a speech while still preserving good quality. *Pulse code modulation* (PCM) is a conversion method that simply groups samples into equally sized buckets based on their amplitude. However, PCM does not reflect that human hearing is more sensitive to lower intensities than large ones. An algorithm called *μ-law* companding transformation [ITU-T, 1988] takes advantage of this fact. This is an equation for the conversion of a PCM sample value x into μ-law:

$$\mu(x) = \operatorname{sgn}(x)\frac{log(1 + |x|\,\mu)}{log(1 + \mu)}$$

where $\mu = 2^b$ and b is the number of bits per sample.

1.1.1 Human Perception and Frequency

Human perception is logarithmic. It means that even though eyes or ears can catch an exponential range of input signals, an exponential change appears to us as linear. An ordinary characterization of sound that people use is by a comparison of *pitch* and *loudness*. These two terms correspond to human perception of something which is called *fundamental frequency* and *power*, respectively.

The power is simply the mean squared amplitude. However, it is more common to refer to *intensity* instead of power. The intensity is measured in decibels (dB) and is equal to:

$$I(x) = 10 \cdot log_{10}\frac{P_x}{P_0}$$

where P_x is the power of the sampled signal x and P_0 is a reference power value such as the auditory threshold pressure.

It is known from the musical theory that an octave (a linear change) corresponds to an interval defined by two pitches that are double or half the frequency of one another. However, the human voice is not a pure tone. It includes a fundamental frequency and a series of upper harmonic frequencies that are modulated by the vocal tract. The fundamental frequency (abbreviated as F_0) of a speech

signal at a particular time is the base frequency of vocal cords at that time. Various psychoacoustic models were proposed to describe the relation between F_0 and pitch for audible frequencies (for example, Zwicker [1961] or Stevens et al. [1937]). One of these models is called *mel scale* and defines a transformation m of a frequency f using the following equation:

$$m(f) = 2595 \cdot log_{10}\left(1 + \frac{f}{700}\right)$$

1.1.2 Frequency Domain

Even though waveforms allow us to easily describe a speech, other representations can be more suitable for some tasks. Remember the waveform from Figure 1.1. At the first glance, we can see where silent and loud segments are. However, it may be difficult for us to identify parts that contain a clean speech and parts that contain, for example, just noise or a piano playing. And it would be surely even more complicated to disentangle a noise and a clean speech given a raw waveform.

A representation that is more convenient for solving the task outlined in the previous paragraph makes use of the *frequency domain*. The Fourier transform enables us to decompose any signal into sinusoids with certain frequencies so that the sum of the sinus waves is equal to the original signal. In other words, it gives us a relation between amplitudes, phases, and component frequencies that can be turned into a graph called the *frequency spectrum*. As we usually work with discretized waveforms, we need to use the discrete Fourier transform:

$$FT(f) = \sum_{k=0}^{N-1} x_k e^{-2\pi i f t_k} = \sum_{k=0}^{N-1} x_k[\cos(2\pi f t_k) - i\sin(2\pi f t_k)]$$

where x_k denotes the k-th sample which appears at time t_k, N is the number of samples, and f is a frequency. Note that the transform can be computed efficiently using the Fast Fourier transform algorithm [Brigham and Morrow, 1967].

Analysis of arbitrary waveforms using discrete Fourier transform would not be of much use because waves change with time. We would lose detailed information about their local properties. To tackle this issue, we can use the discrete short-time Fourier transform (STFT):

$$STFT(f, k) = \sum_{l \in [0,L]} x_{k+l} e^{-2\pi i f t_{k+l}} \omega(l)$$

where f is a frequency, k is the index of the sample that corresponds to the timestamp t_k. The ω denotes a *window function* of length L and defines the relevant context for the particular sample. A typical window function used in speech processing is the *Hann window* [Harris, 1978].

The STFT can be used to compute spectra (also called *frames*) for some evenly spaced timestamps. Subsequently, we can calculate the power for each frequency bin and each frame. By applying this process, we obtain a two dimensional array

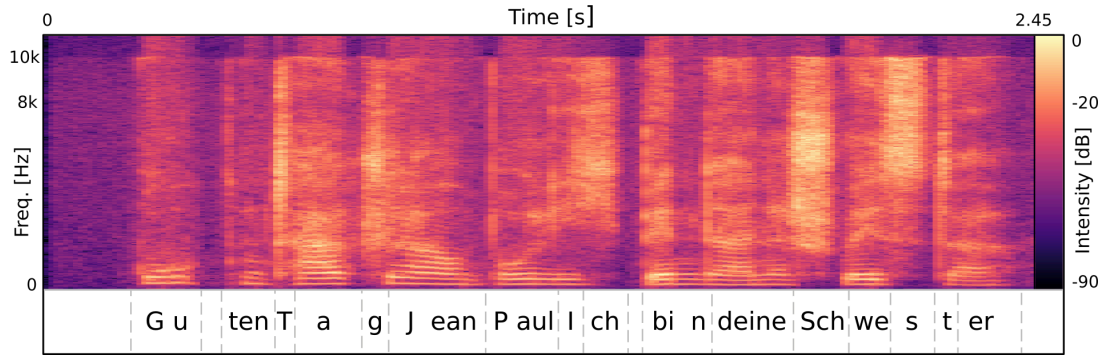

Figure 1.2: *Linear spectrogram of the waveform from Figure 1.1. Notice the linear scale of the frequency domain from 0 Hz to 10 kHz in 1102 frequency bands.*

called *spectrogram* (see Figure 1.2). Note that in the context of speech processing, the length of windows is usually chosen so that it spans tens or hundreds of milliseconds and the gap between succeeding frames usually takes around ten milliseconds. According to the Heisenberg-Gabor limit, there is a trade-off between the time domain and the frequency domain, i.e., wide windows imply more detailed information about frequencies, but a lower resolution in the time domain. That is why we should set the window size carefully.

We have described the construction of linear spectrograms. They contain values for each frequency bin and each frame, but as we have already said in Section 1.1.1, humans perceive frequencies in a logarithmic manner. This brings us to the idea of a compression of linear spectrograms that still preserves all information important for our hearing. To make this compression, we can use the mel scale mentioned earlier. The resulting representation obtained by this transform is called the *mel spectrogram* (see Figure 1.3).

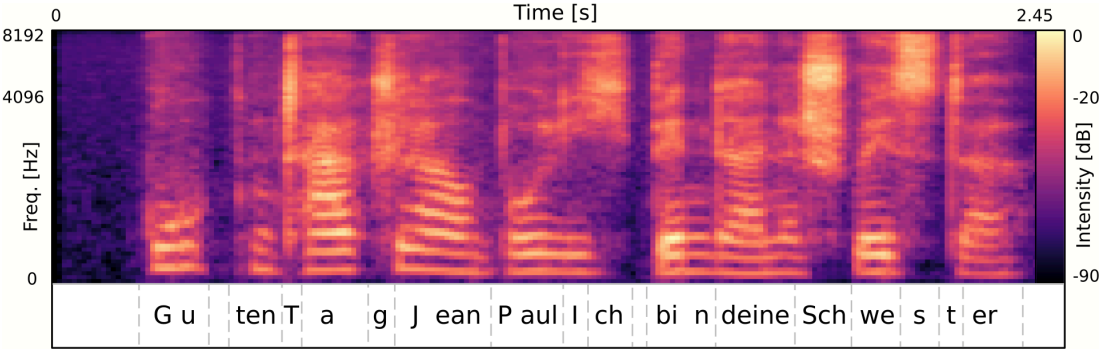

Figure 1.3: *Mel spectrogram of the linear spectrogram from Figure 1.2. The frequency domain is reduced to 80 mel bands.*

1.1.3 Quefrency Domain

We have seen time and frequency domain representations. However, we can go even further and represent audio signals in the *quefrency domain*. The conversion of a mel spectrogram to the quefrency domain is done by applying the *discrete cosine transform* (DCT) to the spectrogram along the frequency dimension. The

transform is closely related to the real part of the Fourier transform and it has various definitions; a commonly used one is the following:

$$DCT(k) = C \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi(2n+1)k}{2N}\right)$$

where C is a normalization constant, N is the number of samples (number of mel frequency bands), and x_n is the value of the n-th sample (it corresponds to the intensity in the input mel spectrogram at a particular frame and a particular frequency band). It gives us a series of *cepstra*, analogous to spectra in the frequency domain, with *mel frequency cepstral coefficients* (MFCC).

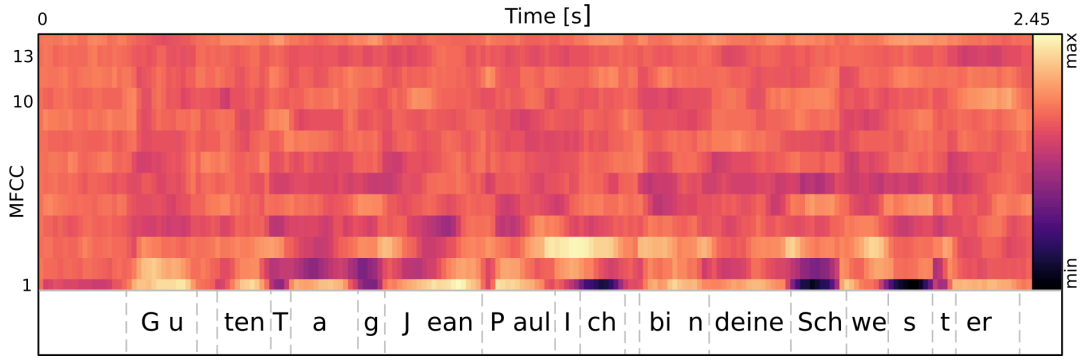

Figure 1.4: *Visualization of MFCCs obtained from the mel spectrogram in Fig. 1.3. Only coefficients 1–13 are shown. Low values (dark color) correspond to fricative sounds, high values (light color) to sonorant sounds.*

We should discuss what MFCCs express. The cepstral features describe changes in the frequency domain. The first coefficient carries information about the average power of the input signal and is usually ignored. Other coefficients simplify, in comparison with the frequency domain representation, the separation of effects caused by the vocal tract and vocal cords. Lower values correspond to fricative sounds and higher values to sonorant sounds (see Figure 1.4, note the fricatives such as "sch" or "s" and sonorants, i.e., vowels or "n"). It is usual to use just a few MFCCs because they can still provide a sufficient amount of information.

1.2 Deep learning

In further text, we refer to numerous deep learning terms and principles. Their explanation is out of the scope of this work. Instead, we provide a glossary that might be useful during reading. It contains brief descriptions of the terms together with links to relevant literature. However, basic knowledge of deep learning is still needed. We again refer the reader to [Goodfellow et al., 2016].

The rest of this chapter is a glossary with entries ordered in the lexicographical order. We often refer to the glossary further in this work.

Activation function – Suppose a neural network that consists of a few fully connected layers and suppose that each layer can be represented by a weight

matrix W. These layers can do a linear transformation: $y = W^\top x$. Thus the whole network can approximate linear functions. To describe more complex dependencies, neural networks usually use affine transformations followed by non-linearities: $y = f(W^\top x + b)$. The non-linear function f is called the activation function. Commonly used functions are *sigmoid*, *tanh*, *ReLU*, *softsign*, and their variants like *PReLU*, *LeakyReLU*, etc. See Hansen [2019].

Attention mechanism – In a general sense of the term, an attention mechanism is a trainable network that can focus on specific parts of inputs. In the context of sequence-to-sequence models (see the "Sequence-to-sequence" entry), the attention is a network that can be queried by the decoder and that extracts suitable information from the encoded input. There are many types of attention mechanisms; we are interested in content-based attention (also called Bahdanau attention [Bahdanau et al., 2015]). Let us consider the i-th decoder step. We denote the encoded input by $m \in \mathbb{R}^{L \times n}$ and the hidden state of the decoder by $h^i \in \mathbb{R}^n$. Then the Bahdanau attention can be described as follows:

$$e^i_j = w^\top \tanh(W h^{i-1} + V m_j + b)$$

$$\alpha^i_j = \exp(e^i_j) \Big/ \sum_{k=1}^{L} \exp(e^i_k)$$

$$c^i = \sum_{k=1}^{L} \alpha^i_k m_k$$

where $W \in \mathbb{R}^{A \times n}$ and $V \in \mathbb{R}^{A \times n}$ are weight matrices, $b \in \mathbb{R}^A$ is a bias and $w \in \mathbb{R}^A$ is a trainable vector. Items of the vector $e^i \in \mathbb{R}^L$ are usually called *energies*. The normalized vector α^i is a vector of *attention weights* and is also called *alignment*. The third equation describes how to combine computed weights with encoded inputs. The resulting vector $c^i \in \mathbb{R}^n$ is the so-called *context vector* and is passed into the decoder to generate the next item of the output sequence. Note that the term $Vm_j + b$, often referred to as *attention memory*, does not change with i, so it can be precomputed.

Batch normalization is a technique that normalizes layer inputs and thus speeds up convergence and allows to train deeper networks. It is very effective when used with convolutional layers. The layer first normalizes its input as follows: $\hat{x} = (x - \mu)/\sigma$ where μ and σ are vectors of means and standard deviations, respectively, computed along the batch dimension (for each input channel). Subsequently, the normalized input is transformed by learned parameters γ and β to compensate for a possible loss of representational ability: $y = \gamma\hat{x} + \beta$. See the work by Wu and He [2018] who compare batch normalization with other normalization layers.

Convolutional layer is a layer that consists of a set of trainable local convolutional *filters* (sometimes called *channels*). The number of filters is a hyperparameter. Every filter has the same size called *kernel size*, which is usually small, but filters are moved (convolved) and applied to the whole input. This implies that we need to specify also the *stride* with which we slide the filters over the input. Finally, the last hyperparameter of a convolutional layer, called *dilation*, controls the sparsity of filters. To be more specific, it inserts spaces between filter cells (so in the case of one-dimensional convolutional layers, for example, a filter of size

3 is stretched by a dilation 2 so that it spans 5 input cells and the output is connected with every other input cell). The calculations done by a one-dimensional convolutional layer can be described by the following equation:

$$\text{conv}(I)_{i,c} = b_c + \sum_{k,f} I_{i \cdot S + k, f} W_{k,f,c}$$

where $I \in \mathbb{R}^{L \times F}$ denotes an input of length L with F channels, $W \in \mathbb{R}^{K \times F \times C}$ is a weight matrix where K is the kernel size and C is the number of output channels. The stride is represented by S and b is a bias. Moreover, popular frameworks allow us to group multiple convolutional layers with the same hyperparameters into a single layer. This layer is called *grouped convolutional layer* and the included separate convolutions are referred to as *groups*.

Cosine similarity loss is a loss function that is based on the cosine similarity which is defined as: $\text{SIM}(a, b) = (a^\top \cdot b)/(\|a\| \, \|b\|)$ where a and b are input vectors.

Cross entropy loss is a loss function that is used for classification problems with C classes. It requires a predicted distribution o over C classes and a one-hot vector t with 1 in the position corresponding to the true class. The cross-entropy loss is then: $\text{CE}(t, o) = -\sum_{i=1}^{C} t_i log(o_i)$.

Dropout is a technique used to prevent overfitting of neural networks. During training, it randomly sets the outputs of a fraction (called *dropout rate*) of neurons to zero. It can be viewed as a random sampling from an exponential number of different networks. See Srivastava et al. [2014].

Embedding layer – Neural networks usually work with real numbers. Embedding layers are trainable lookup tables that are used to convert characters or words to numeric vectors. These layers can be trained in a way so that the similarity of the resulting vectors reflects the semantic or syntactic similarity of input words [Peters et al., 2018].

Fully connected layer is a basic type of layers used in neural networks. Neurons in a fully connected layer are connected to all outputs of the previous layer. It is an affine transformation, i.e., $y = W^\top x + b$ where W is a matrix of trainable parameters and b is a trainable vector called *bias*.

Gated layer – The architecture of gated layers enables a regulation of information flow. It is used in recurrent layers [Hochreiter and Schmidhuber, 1997] or highway networks [Srivastava et al., 2015]. The layer makes use of a gating vector g. Given an input x, the output vector y is computed as follows: $y = g \cdot x + (1 - g) \cdot x$. Note the pointwise multiplications.

Gradient – Training of neural networks is an optimization problem. It is usually done by gradient-based algorithms that use derivatives to update network parameters (see the "Optimizer" entry). Gradients are calculated using the backpropagation algorithm by applying the chain rule of differentiation starting from network outputs and propagating the gradients backward. That is why neural networks and loss functions (see the "Loss function" entry) have to be differentiable with respect to weights.

Gradient clipping is a method used to prevent exploding gradients (i.e., excessively large gradients resulting in unstable learning) in very deep or recurrent

neural networks. It can be implemented as a simple normalization of gradients. The normalization is done only when the L2 norm of a parameter vector exceeds a certain threshold. See Pascanu et al. [2013].

GRU is a simplification of the LSTM [Cho et al., 2014]. In comparison with the LSTM cell, it does not keep an internal state. At each step, it processes just a hidden state (also called output vector) from the previous step h_{t-1} and an input vector x_t. The internal architecture of the GRU cell includes a *reset gate* and an *update gate*. Updates done in a single step can be described as follows:

$$
\begin{aligned}
u_t &= \sigma(W^u x_t + U^u h_{t-1} + b^u) \\
r_t &= \sigma(W^r x_t + U^r h_{t-1} + b^r) \\
\tilde{h}_t &= \tanh(W^y x_t + U^y (r_t \cdot h_{t-1}) + b^y) \\
h_t &= u_t \cdot \tilde{h}_t + (1 - u_t) \cdot \tilde{h}_{t-1}
\end{aligned}
$$

where u_t, r_t denote outputs of update and reset gates, respectively. W and U are weight matrices and bias vectors are denoted by b.

Group normalization – See the "Batch normalization" entry. The group normalization layer uses means μ and standard deviations σ along the channel dimension, which is divided into a few groups of equal size. This can be helpful when a small batch size is used and the number of channels is greater than that because, in this scenario, the batch statistics do not have to be very reliable. Wu and He [2018] compare the group normalization with other normalization layers.

Highway network (layer) is an enhanced fully connected layer. It consists of two parallel fully connected layers that transform the input of the highway network. The first is followed by a sigmoid function (we denote its output by g) and the second layer precedes an arbitrary non-linearity (we denote its output by $\hat{x}$). The output of the highway layer is then computed as: $y = g \cdot x + (1 - g) \cdot \hat{x}$. Note the pointwise multiplications. See Srivastava et al. [2015].

Learning rate – Training of neural networks is an optimization problem. Algorithms used for it are called optimizers and they are usually controllable by a few parameters. One of these parameters is called learning rate. It controls the step size during the optimization. It is common to change the learning rate during training, a policy of these changes is called the *learning rate schedule*.

Learning rate schedule – See the "Learning rate" entry. Some frequently used schedules are, for example, the *stepped learning rate decay* (that decays the learning rate every n training steps), the *exponential learning rate decay* (that multiplies the learning rate by γ every training step), or the *cyclic learning rate policy* [Smith, 2017].

Loss function – The process of finding suitable parameters of a neural network is called training. It is reduced to an optimization problem that is solved by an algorithm called *optimizer*. During optimization, the optimizer tries to minimize a so-called *loss function*. In the context of neural networks, this function is a measure of the difference between expected outputs and actual outputs produced by the trained network and it is often defined in such a way that the optimization then corresponds to maximization of the likelihood of training data.

L1 loss is a loss function that measures the *mean absolute error*. For two input vectors y and $\hat{y}$, both of length N, the loss is: $\mathrm{L1}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} |y_i - \hat{y}_i|$.

LSTM is a type of a recurrent neural network introduced by Hochreiter and Schmidhuber [1997]. The LSTM at a particular time step t keeps an internal state c_t and processes an input vector x_t and a hidden state (also called output vector) from the previous step h_{t-1}. The internal architecture of the LSTM cell includes an *input gate*, an *output gate*, and a *forget gate*. Updates done in a single step can be described by the following equations:

$$
\begin{aligned}
i_t &= \sigma(W^i x_t + U^i h_{t-1} + b^i) \\
f_t &= \sigma(W^f x_t + U^f h_{t-1} + b^f) \\
o_t &= \sigma(W^o x_t + U^o h_{t-1} + b^o) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \tanh(W^y x_t + U^y h_{t-1} + b^y) \\
h_t &= o_t \cdot \tanh(c_t)
\end{aligned}
$$

where i_t, o_t, and f_t denote outputs of input, output, and forget gates, respectively. W and U are weight matrices and bias vectors are denoted by b.

MSE loss is a loss function that measures the *mean squared error*. For two input vectors y and $\hat{y}$, both of length N, the loss is: $\mathrm{MSE}(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2$.

Optimizer – See "Learning rate" and "Gradient" entries. Training of neural network's parameters is usually based on a gradient-based optimization. The optimizer is given gradients and updates parameters. However, updates based on whole training datasets are inefficient and intractable. Instead, so-called *mini-batches* are used. Training data are divided into random and relatively small groups. The updates and gradient computations are then done for each mini-batch in training data. Commonly used optimizers are, for example, *Stochastic Gradient Descent* (with momentum, see Sutskever et al. [2013]), *Adam* [Kingma and Ba, 2015], or more recent *AdamW* [Loshchilov and Hutter, 2019]. The vanilla gradient descent optimizer can be described by the following equation: $\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x_{i:i+B}; y_{i:i+B})$ where θ stands for a parameter of the network, η is the learning rate and $\nabla_\theta J$ denotes gradients (calculated from a mini-batch that contains examples with indices from i to $i+B$) with respect to the parameter θ and a loss function J.

Pooling layer is similar to a convolutional layer that implements a pooling technique, i.e., a method that performs a fixed pooling operation on a particular input. In comparison with an ordinary convolutional layer, it does not compute a weighted combination of its inputs, instead, it takes, for example, the maximal input value or the average of input values. These layers are usually used in image recognition tasks together with convolutional layers to support their translational invariance. The name *global* pooling layer refers to a layer that has a notion about the whole input (i.e., the kernel spans the whole input).

Recurrent neural network (RNN) is a type of a neural network used for processing of sequential data (possibly of variable length). A typical RNN processes input sequences sequentially. At each time step, it takes the current input and the previous hidden state and outputs a new hidden state. It is called recurrent because all the time steps apply the same network parameters (but to different in-

puts). Bidirectional RNNs contain two RNNs (forward and backward) processing the input in different directions. Outputs of the RNNs are then combined (concatenated). Note that RNNs usually suffer from the exploding gradient problem and thus are combined with teacher forcing (see the "Teacher forcing" entry) or gradient clipping (see the "Gradient clipping" entry). See Pascanu et al. [2013].

Residual connection is a link that bypasses a layer or a group of layers of a neural network. At the end of the bypass, the modified and the residual signals are usually summed. Residual connections allow the composition of very deep (convolutional) neural networks because they clear the way for the backpropagation of gradients and avoid their vanishing [He et al., 2016].

Sequence-to-sequence is a general term used for a family of methods that convert input sequences into other sequences [Sutskever et al., 2014]. A sequence-to-sequence network architecture consists of an encoder, a decoder, and an attention mechanism. The encoder turns input sequences into encoded sequences that are subsequently read by the decoder using the attention mechanism. The decoder generates output sequences. In a very basic setting, the attention mechanism is omitted and the last encoder output is used for the initialization of the decoder.

Skip connection – See the "Residual connection" entry. These terms are interchangeable.

Teacher forcing is a technique used for training of recurrent neural networks (see the "Recurrent neural network" entry). At each training step, the ground truth output from a prior time step is used as the current input instead of the output from the previous step. See Pascanu et al. [2013].

Variational autoencoder – An autoencoder is a neural network whose goal is to predict the input itself. The input is first encoded into a so-called *latent space* with a lower dimension. Afterward, the original input is reconstructed from a point in the latent space. The network is thus forced to learn a compressed representation of the input. Variational autoencoders predict a distribution (such as the Gaussian distribution parameterized by its mean and variance) in the latent space instead of a single point, then sample from the distribution and use the obtained point to generate the original input. See Kingma and Welling [2014].

Weight decay – It is a term in the weight update rule of optimizers that causes weights to exponentially decay (if no other update is done). For the stochastic gradient descent optimizer, the weight decay is equivalent to the L2 regularization which is a basic machine learning regularization technique that penalizes higher parameter values. See Loshchilov and Hutter [2019].

Zoneout – It is a regularization technique used for recurrent neural networks (see the "Recurrent neural network" and "Dropout" entries). The dropout regularization does not work well when applied to RNN states because it can lead to a loss of important information through time. The zoneout, on the other hand, preserves a fraction (called *zoneout rate*) of states from the previous step and updates the remaining states with new values [Krueger et al., 2018].

Chapter 2

Speech Synthesis

In a general sense of the term, speech synthesis is a discipline that aims at a controllable production of an artificial human voice. It includes various subfields, for example, speech reconstruction or more importantly *text-to-speech* (TTS). Text-to-speech, as the name suggests, concerns the automatic conversion of a written text to a speech audio signal. We use the terms speech synthesis and text-to-speech interchangeably further in this work.

A typical text-to-speech pipeline [Taylor, 2009] is divided into these steps:

- **Text analysis** – performing segmentation and text normalization, i.e., substitution of numbers, abbreviations, dates, etc. into spelled-out forms.
- **Linguistic analysis** – obtaining a linguistic specification from the input, i.e., examining the prosody pattern, converting input graphemes (basic units of writing systems such as letters, logograms, numbers, punctuation marks, etc.) into phonemes (fundamental units of sound that distinguish one word from another in a particular language; note that the relation between graphemes and phonemes varies greatly from language to language), and making decisions about intonation, stress, and phoneme durations.
- **Waveform generation** – producing a sound wave based on the output of linguistic analysis.

Text-to-speech has a one-to-many character because people speak different voices, accents, and an input text can have various realizations based on speaker intentions. That is why some TTS systems accept the characteristics of the target speaker or even a description of the desired prosody (for example, using the speech synthesis markup language).

Sometimes, speech synthesis systems omit the text analysis subtask completely. It is often rule-based and highly language-dependent. Instead, they reckon on a clean input with a suitable segmentation and without any unfavorable symbols.

The remaining part of this chapter summarizes the evolution of speech synthesis (Section 2.1), but more importantly, it discusses current influential end-to-end models based on neural networks (Section 2.2).

2.1 History

The first attempts to build a speech synthesis system worthy of the name were made by Wolfgang von Kempelen at the turn of the eighteen and the nineteen century [Dudley and Tarnoczy, 1950]. The structure of his man-powered speaking machine tried to imitate the human vocal tract – it contained holes acting as nostrils, a rubber cone representing the mouth, etc. Even though it is reasonable to think that speech synthesis systems should somehow follow the structure of vocal organs, the past decades brought to us more serious machines and algorithms for speech synthesis. Two main principles stood out – concatenative and parametric methods [King, 2011].

2.1.1 Concatenative Synthesis

Concatenative or *example-based* systems use a database of recorded speech units [Campbell, 1995], e.g., diphones spanning two halves of successive phonemes. During synthesis, these pieces are selected with respect to an evaluation function and concatenated to produce the desired speech. However, the process of retrieval or selection is complicated because the system has to search for the best units, possibly slightly mismatched in terms of pitch, speed, etc.

The advantage of the concatenative synthesis is that the pieces are natural-sounding. Unfortunately, the segmentation and concatenation are hard and often produce audible glitches even if adjustment and smoothing of the units are performed. Another drawback of this approach is that it requires a high-quality database of speech segments with suitable variations. This implies a limited scalability of this method. For example, a change of the speaker's voice or the source language would require a completely new database.

2.1.2 Statistical Parametric Speech Synthesis

Another method called statistical parametric speech synthesis (SPSS) is a *model-based* approach [Zen et al., 2009]. Instead of storing a database of speech, it stores a trained statistical parametric model. The main part of such models is an *acoustic model* that tries to predict or generate parameters of the output audio wave given some linguistic features or rather a specification. The generated parameters are subsequently used by a *vocoder* to reconstruct the speech signal (see Figure 2.1). However, the vocoder can be a non-statistical external module. The reason why the acoustic model usually predicts just features is that predicting a waveform directly at a high sample rate might be difficult. Typically used acoustic features are MFCCs with F_0 (with rates of change such as the first and second derivatives), or spectrograms.

The acoustic model can be represented by context-dependent Hidden Markov Models (the whole model is constructed by concatenating models corresponding to a particular context) or simple neural networks [Zen, 2015]. The model commonly used in SPSS also includes, besides the acoustic model, an explicit *duration model*. This predicts the number of frames to be generated by each state of the model.

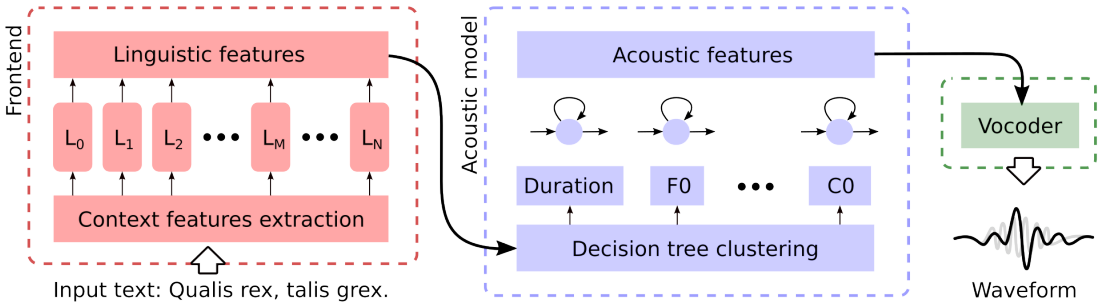

Figure 2.1: *Diagram of a typical SPSS model. The red rectangle depicts a frontend that takes the text input and extracts various linguistic features, which are then passed into the acoustic model (the blue rectangle). The acoustic model creates acoustic features suitable for the vocoder that synthesizes a final audible waveform.*

A decision tree context clustering is often used to control model complexity (which may quickly grow if separate models are used for different contexts).

In comparison with concatenative systems, statistical parametric speech synthesis is more flexible and does not require a huge amount of high-quality and phonetically balanced data. On the other hand, the SPSS often struggles with *over-smoothing*, i.e., audio signal details are removed or averaged.

2.2 Deep Neural Systems

The two solid methods outlined in the previous section can produce high-quality speech, make it possible to control some characteristics of the synthesized speech and offer guarantees which are necessary for production use. Recent neural models build on them and enable synthesizing more flexible and natural-sounding speech, even in an end-to-end manner. We are going to describe the most influential models and more importantly models related to our experiments in detail in Sections 2.2.1–2.2.6. At the end of this section, we briefly mention other interesting models in Section 2.2.7.

2.2.1 WaveNet

In 2016, van den Oord et al. [2016] introduced a neural generative text-to-speech model called WaveNet. It was a real breakthrough because WaveNet significantly outperformed previous state-of-the-art parametric and concatenative systems in terms of output quality. The model converts linguistic features into audio waveforms, so it replaces both the acoustic model and the vocoder known from the classical SPSS.

Let us consider a waveform $x = \{x_1, ..., x_T\}$. The joint probability of the waveform or, better said, of samples of the waveform can be factorized as follows:

$$p(x) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1})$$

WaveNet models the probabilities $p(x_t|x_1, ..., x_{t-1})$. The key idea behind the model is the usage of many convolutional layers.[1] Their advantage over recurrent layers is that they are more easily trainable and that they can be more easily parallelized (it implies faster training). Note that the first convolutional layer of WaveNet is *causal*, which means that its input is shifted by a frame so that the model cannot see into the future during training.

However, to make the whole convolutional setup working and to model the probability of x_t properly, the model needs to take into account a sufficiently large receptive field (i.e., hundreds of milliseconds). One way to enlarge the context is to use dilated convolutions.[1] Stacking of dilated layers causes an exponential grow of the receptive field (WaveNet doubles the dilation at each consecutive layer). To support and speed up convergence of the whole model, residual and skip connections are used.[2] See Figure 2.2, which symbolically describes the whole architecture.

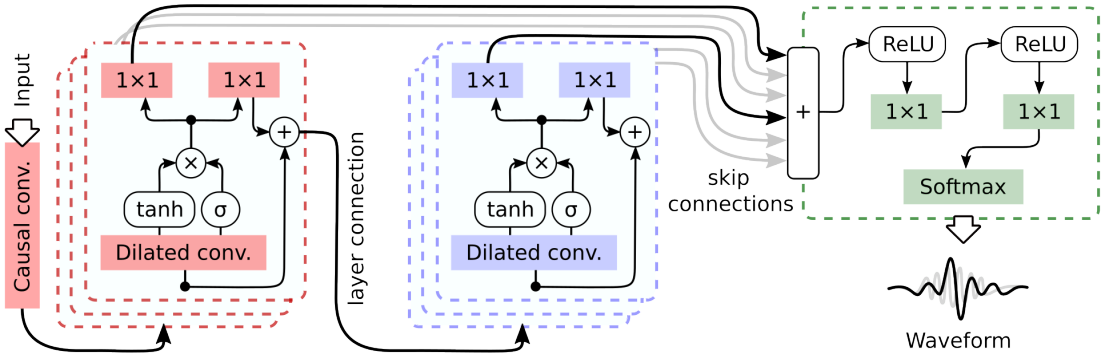

Figure 2.2: *Illustration of WaveNet with two stacks or dilation cycles (red and blue), each having three dilated convolutional layers where the k-th layer of a stack has dilation rate* 2^k*. The layers include two convolutions with the kernel size 1, a gating mechanism, a residual connection, and a skip connection. The output of each layer is the input to the next layer and the output of the last layer of a stack is the input to the first layer in the next stack. Skip connections are summed and the resulting value is used for predicting the next sample (green rectangle). Circles depict pointwise multiplications or summations and other colorless boxes visualize activation functions.*

Let us now describe what the final prediction of x_{t+1} is. The WaveNet model predicts a probability distribution over the 256 possible values of an 8-bit audio sample, quantized using the μ-law companding transformation (see Section 1.1). The concrete sample value is then obtained by sampling the distribution.

We have described the basic setting of WaveNet, but we would like to produce waveforms given linguistic features. To adapt WaveNet to this task, the authors upsample the input features to a higher sampling rate. This upsampled signal is then transformed in each convolutional layer using a fully connected layer[3] and summed to the output of the dilated convolution (before an activation function).

[1] See "Convolutional layer" in the Glossary.
[2] See "Skip connection" in the Glossary.
[3] See "Fully connected layer" in the Glossary.

The way in which WaveNet models the probability implies that all samples can be processed in parallel during training. During inference, the model needs to obtain all the samples $x_1, ..., x_t$ to generate the next sample x_{t+1}, so the model is sequential and autoregressive. This is the main drawback of the WaveNet architecture because the number of samples to be generated is relatively large (even for lower sampling rates). To sum it up, even though the training is fast and parallel, the inference is much slower than real-time (it takes minutes of GPU time to generate a few seconds of speech, so it is not very practical).

2.2.2 Deep Voice

Shortly after the unveiling of WaveNet, Arik et al. [2017] came up with a new model called Deep Voice. This was the first attempt at a fully end-to-end TTS system. Moreover, the authors managed to speed up the WaveNet model described in Section 2.2.1 and reached faster-than-real-time inference times. Deep Voice is inspired by traditional TTS pipelines and has a very similar structure, but it removes any handcrafted or hard-to-obtain features. During training, in addition to the input text transcriptions, it requires only phonemes with stress annotations, phoneme durations, and the fundamental frequency. Plain graphemes are the only input required during inference. Deep Voice consists of the following modules (see Figure 2.3):

- **Grapheme-to-phoneme** – this module predicts phonemes from input graphemes. It follows the encoder-decoder architecture[4] based on GRUs[5].
- **Segmentation** – this module is used to locate phoneme boundaries in training audio recordings and follows architectures used in speech recognition. The input audio is featurized by MFCCs (see Section 1.1.3).
- **Phoneme duration and fundamental frequency** – this module predicts duration and F_0 for each input phoneme. It consists of a few fully connected layers that are followed by a single unidirectional GRU and an output fully connected layer.
- **Audio synthesis** – to produce waveforms, Deep Voice uses the WaveNet model conditioned on phonemes with durations and F_0 profiles.

Please refer to the original paper [Arik et al., 2017] for more details about the architecture and parameters of the mentioned modules.

As mentioned above, Deep Voice uses an optimized version of WaveNet with faster-than-real-time inference times. This fast performance was achieved by parallelization, a careful choice of WaveNet's parameters, and subdivision of layers so that the whole model is cache-friendly and nothing has to be recalculated.

The second version of the model, called Deep Voice 2, extends the original version to enable multi-speaker speech synthesis [Gibiansky et al., 2017]. The third version of Deep Voice [Ping et al., 2018] follows Tacotron (which will be described in

[4]See "Sequence-to-sequence" in the Glossary.
[5]See "GRU" in the Glossary.

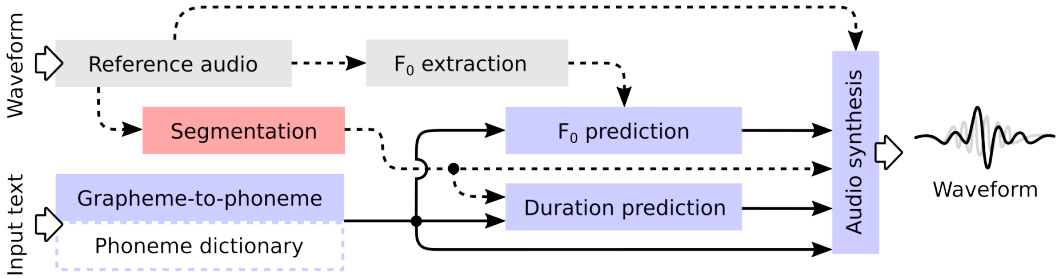

Figure 2.3: *Diagram showing interactions of modules of Deep Voice. Blue rectangles are trainable neural networks used during training and inference, the red rectangle depicts a trainable network used just during the training phase. Gray rectangles are non-trainable parts of the pipeline. Solid lines depict forward pass through the network and dashed lines mark supervision signals used by individual modules during training. Note that the duration and F_0 predictions are done by a shared network even though they are decoupled in this illustration.*

the next section) and adapts a fully convolutional attention-based[6] sequence-to-sequence architecture. However, the model does not contain any groundbreaking ideas and our experiments were focused on Tacotron.

2.2.3 Tacotron 1

Remember the diagram in Figure 2.1. WaveNet merged the acoustic model and the vocoder into a single neural network. Tacotron [Wang et al., 2017], on the other hand, substitutes both the frontend (which extracts linguistic and contextual features from the input text) and the acoustic model. Unlike WaveNet, the only input expected by Tacotron is a raw text. Tacotron does not need any feature engineering, domain expertise, or expensive extraction of features, and thus, together with a vocoder, it constitutes a fully end-to-end system.

Contrary to Deep Voice, Tacotron does not require phoneme dictionaries or phoneme-level alignments for training. It does not even need a complex vocoder because Tacotron can output detailed linear spectrograms. A popular vocoder that works with spectrograms is called the *Griffin-Lim algorithm* [Griffin and Lim, 1984]. It is a non-trainable algorithm that recovers complex-valued spectrograms using an iterative process that makes use of the pseudo-inverse STFT.

It is important to realize the advantages of integrated end-to-end systems without traditional components. According to Wang et al. [2017], the advantages over multi-stage systems are:

- **Fast and easy adaptation** to new data because no laborious feature engineering is needed.
- **Easier conditioning** on various attributes like speaker identity, language, or sentiment.
- **Robustness** because errors of components in multi-stage systems can be propagated and added.

[6] See "Attention mechanism" in the Glossary.

We also would like to note that we are putting less prior knowledge into an integrated system such as Tacotron, so the model can learn its own way of reasoning and is not restricted by human notions.

Wang et al. [2017] compared Tacotron to strong concatenative and parametric baselines in human evaluation, where it was able to outperform the parametric baseline. However, the concatenative system was rated better, probably because of the Griffin-Lim vocoder that often produces audible artifacts.

The architecture of Tacotron (see Figure 2.4) follows the general paradigm of sequence-to-sequence architecture with attention. This means that it can be decoupled into an encoder which transforms input texts of some length into another representation of the same length, and a decoder which follow particular positions in the encoder output using the attention mechanism and produces a final output sequence of a different length. The input sequence is expected to be a raw text and the output sequence consists of frames of a linear spectrogram with logarithmic magnitudes. Wang et al. [2017] use frames spanning 50 milliseconds with a frameshift of 12.5 milliseconds and with 2048 frequency bands.

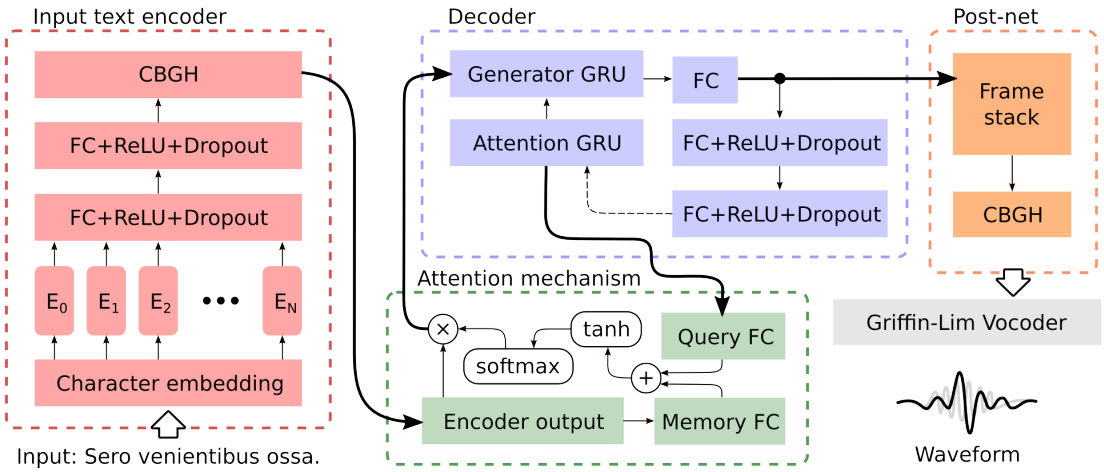

Figure 2.4: *Visualization of Tacotron. "FC" stands for a fully connected layer. The red dashed rectangle depicts the encoder with an embedding layer. The blue area describes a single step of the decoder (note the oriented cycle – the dashed line connects the result of the previous step with computations of the next step). The attention module is represented by the green rectangle (circles represent pointwise summation or multiplication). The query coming from the decoder has to be expanded to match the length of the input text. The post-net which transforms stacked frames produced by the decoder is shown in orange color. The final output spectrogram can be passed into a vocoder.*

In the following, we describe the model components in detail.

Throughout the model, a network module called *CBGH* is used repeatedly. An input sequence of the module is fed into a bank of one-dimensional convolutions. The bank of size k consists of k convolutional layers with kernels of size $1, \ldots, k$. These convolutional layers should model 1 to k input tokens. Their outputs are stacked and max-pooled[7] along the time dimension. Next, the sequence is passed

[7]See "Pooling layer" in the Glossary

through a few one-dimensional convolutions and summed with the original input sequence through a residual connection. The summation is passed into a stack of four highway layers[8] and then it is fed to a final bidirectional GRU layer[9] with 128 units. The convolutional layers are regularized using batch normalization[10]. It is not hard to see that this module is somewhat cumbersome and complex and that is why it was completely removed in Tacotron 2 (see Section 2.2.4).

The encoder consists of a character embedding[11] of size 256, a *pre-net* module and a final CBGH module with output size 128. The pre-net module includes just two simple fully connected layers with ReLU activations and dropout layers[12].

Tacotron uses content-based attention, i.e., the Bahdanau attention [Bahdanau et al., 2015]. However, this type of attention is not location-based and thus does not take into account the history of previously attended positions and also does not take advantage of the fact that reading has a monotonous character. That is why Tacotron 1 sometimes tends to jump to other positions in the input text. It results in skipping words or repetition of some parts of the text. These problems were successfully fixed in the second version (see Section 2.2.4).

The decoder has two GRU layers of size 256. The first layer produces queries to the attention mechanism and the second layer acts as a generator and accepts the output of the first layer together with a context vector returned by attention. The output of the generator layer is passed into a fully connected layer which predicts frames of a mel spectrogram with 80 frequency bins. Wang et al. [2017] report that it is crucial to emit two frames per decoder step because this speeds up convergence and results in a more stable attention.

The first decoder step is conditioned on a zero frame and subsequent steps are conditioned on the last predicted frame (or ground-truth frame during training, this method is called teacher forcing[13]. In fact, before feeding an output frame into the next decoder step, it is passed through a pre-net with two fully connected layers with ReLU activations and dropout layers. Wang et al. [2017] state that the pre-net acts as an information bottleneck and the dropout is extremely important for generalization.

The described sequence-to-sequence part of Tacotron outputs mel spectrograms. It is possible to invert them into waveforms using the Griffin-Lim algorithm, but this would produce salient artifacts. The purpose of the *post-net* is to turn mel spectrograms into a representation that works better with the Griffin-Lim vocoder. That is why Tacotron includes the so-called *post-net* that consists of a CBHG module that reconstructs linear spectrograms from the generated ones.

[8] See "Highway network" in the Glossary
[9] See "Recurrent neural network" in the Glossary
[10] See "Batch normalization" in the Glossary
[11] See "Embedding layer" in the Glossary
[12] See "Dropout" in the Glossary
[13] See "Teacher forcing" in the Glossary

2.2.4 Tacotron 2

The main drawback of Tacotron is the need for the Griffin-Lim vocoder. However, this imperfection was fixed in the second version [Shen et al., 2018]. It simplifies Tacotron 2 and modifies WaveNet so that it can synthesize waveforms from mel spectrograms. The spectrograms have the same properties as described in Section 2.2.3, i.e., the STFT uses the Hann window function (see Section 1.1.2), the frame size is 50 milliseconds, and the frame hop is equal to 12.5 milliseconds.

Simplifications

Tacotron 2 does not have any CBGH modules and the character embedding size and encoder output size were doubled to 512. The encoder includes three one-dimensional convolutional layers with kernel size 5. Each of these layers is followed by a batch normalization layer, a ReLU activation, and a dropout layer. Finally, a bidirectional LSTM layer[14] of size 512 comes after the final convolutional layer.

The original Bahdanau attention was enhanced using a so-called *cumulative location-sensitive attention* [Chorowski et al., 2015]. This type of attention mechanism uses cumulative attention weights as an additional feature and can be described using the following equations:

$$e_j^i = w^T \tanh(Wh^{i-1} + Vm_j + U[F * \sigma^{i-1}]_j + b)$$
$$\alpha^i = \text{softmax}(e^i)$$
$$\sigma^i = \sigma^{i-1} + \alpha^i$$

where $\alpha^i \in \mathbb{R}^n$ is a vector of attention weights at the i-th step of the decoder, $*$ is a convolution operator, $\sigma^i \in \mathbb{R}^n$ denotes the cumulative weights at the i-th step, and U, F are weight matrices (for a linear projection and a one-dimensional convolution, respectively). Other symbols follow the notation from the Glossary, i.e., $m \in \mathbb{R}^{L \times n}$ denotes the encoded input, $h^i \in \mathbb{R}^n$ is the hidden state of the decoder at the i-th step, $W \in \mathbb{R}^{A \times n}$ and $V \in \mathbb{R}^{A \times n}$ are weight matrices, $b \in \mathbb{R}^A$ is a bias, and $w \in \mathbb{R}^A$ is a trainable vector. This attention type encourages the model to move forward consistently and should mitigate certain shortcomings of the original Tacotron (word skipping and repetition). The convolution that extracts features for the attention mechanism has a kernel of size 31 and 32 filters.

GRU layers of the decoder are replaced with larger LSTM layers with 1024 units. The LSTM layers are regularized using zoneout layers[15]. The pre-net is unchanged. The only difference is that Shen et al. [2018] advise leaving the pre-net dropout enabled even during inference to bring more variation into outputs. However, the exact architecture of the decoder is not completely clear from the original description. It says:

> *The pre-net output and attention context vector are concatenated and passed through a stack of 2 uni-directional LSTM layers with 1024 units.*

[14] See "LSTM" in the Glossary
[15] See "Zoneout" in the Glossary

Unfortunately, Shen et al. [2018] do not mention which part of the decoder produces queries to the attention mechanism. We conclude that it is possible to use the same setup as in the first version of Tacotron (Section 2.2.3).

Contrary to the first version, Tacotron 2 generates just a single spectrogram frame per decoder step. It predicts the frame from a concatenation of the context vector and the output of the last LSTM layer. In parallel to this prediction, the concatenation is also projected into a scalar that is passed through a sigmoid function and decides when to stop the generation. This *stop token* is very useful during inference.

The post-net's CBGH module is also replaced with five one-dimensional convolutional layers with a kernel of size 5 followed by batch normalization layers and tanh activations (except for the last layer). The whole post-net is bypassed with a residual connection.

Since Tacotron 2 (without WaveNet) is directly used in our own experiments, we further specify the architecture hyperparameters, implementation details, and training procedure in Chapter 6. The WaveNet vocoder used by Shen et al. [2018] is described in the following.

WaveNet

The architecture of Tacotron 2's WaveNet vocoder follows the architecture described in Section 2.2.1. It contains 24 layers in four dilation cycles and two upsampling layers that enlarge the input spectrogram to a higher sampling rate. Instead of predicting a simple distribution (which is sampled during inference) of the quantized sample values, this version of WaveNet predicts a ten-component mixture of logistic distributions. So at the end of the forward pass, there are linear layers predicting a mean, a log scale, and a mixture weight for each component.

Tacotron 2 with a WaveNet vocoder is very slow during inference and even training does not seem to be trivial. The authors trained WaveNet using 32 GPUs with distributed mini-batches[16] of size 128.

2.2.5 WaveRNN

In our experiments (described in Chapter 6), we replaced the WaveNet vocoder with a model based on a single-layer recurrent neural network called WaveRNN [Kalchbrenner et al., 2018]. It still matches the quality of outputs produced by WaveNet while mitigating the problems associated with sequential models (such as slow inference).

WaveRNN uses a recurrent layer with a cell that is based on the architecture of GRU[17]. The original GRU was modified in a way that allows a fusion of certain matrix multiplications. Another change allows splitting the cell into two independent parts for predicting fine and coarse bits of samples. The WaveRNN cell

[16] See "Optimizer" in the Glossary
[17] See "GRU" in the Glossary

can be described as follows:

$$
\begin{aligned}
x_t &= [c_{t-1}, f_{t-1}, c_t] \\
u_t &= \sigma(W^u \odot x_t + U^u h_{t-1} + b^u) \\
r_t &= \sigma(W^r \odot x_t + U^r h_{t-1} + b^r) \\
\tilde{h}_t &= \tanh(r_t \cdot U^y h_{t-1} + W^y * x_t + b^y) \\
h_t &= u_t \cdot \tilde{h}_t + (1 - u_t) \cdot \tilde{h}_{t-1}
\end{aligned}
$$

where $\odot$ denotes a masked matrix multiplication that handles the coarse and fine parts. Compare the equation of the reset gate r_t with the definition in Glossary. Notice that U^y, U^z, and U^r can now be concatenated for a faster computation, contrary to the original GRU. To predict the resulting sample probability, the output vector h_t is split into two parts which are passed through a fully connected layer with a ReLU activation followed by one more fully connected layer and a softmax layer.

The tips and tricks suggested by Kalchbrenner et al. [2018] for reduction of sampling-time requirements include:

- **Reduction of the number of operations per sample** – sampling using a single-layer WaveRNN requires five sequential matrix-vector products, but WaveNet has to perform tens of matrix-vector products per sample.
- **Reduction of a launching overhead for individual operations** – this can be done by implementing custom GPU operations. Using this technique, authors of the WaveRNN model managed to sample high quality 24 kHz audio four times faster than real time. Their best WaveNet runs around three times slower than real time.
- **Reduction of the total number of parameters in the network** – Kalchbrenner et al. [2018] used a weight-pruning technique and sparsified the large RNN. The resulting network with a sparsity level greater than 96% significantly outperformed the original dense version.
- **Subscaling** – samples are split into B interleaving sub-tensors with a frequency or scale that is B times smaller. It means that all $(Bi+s)$-th samples for a particular $0 \leq s < B$ and for all $i \geq 0$ are in the same sub-tensor. The joint probability of samples is then reformulated as:

  $$p(x) = \prod_{s=0}^{B} \prod_{i=0}^{|x|/B} p(x_{Bi+s} | x_{Bj+s} \text{ for } j < i, x_{Bk+z} \text{ for } z < s, k \geq 0)$$

  It means that the generation of x can be done in parallel without breaking any *local* dependencies. Subscaling alone, even without any other speedups, allows sampling in real time.

Kalchbrenner et al. [2018] showed that they can sample in real time even on a mobile phone CPU using a combination of methods sketched above. Further details of these techniques are out of the scope of this work.

2.2.6 Deep Convolutional TTS

The previous section showed that the usage of recurrent neural networks for modeling of sequential data brings favorable results. On the other hand, Deep Convolutional Text-to-Speech (DCTTS) [Tachibana et al., 2017] offers a fully-convolutional alternative to the Tacotron model. The convolutional architecture results in faster training times and brings an ability to fully train a system producing intelligible speech overnight on an ordinary GPU.

We briefly describe interesting ideas introduced in DCTTS because we used some of them in our implementation of Tacotron (see Chapter 6).

The architecture of DCTTS consists of four components: text encoder, audio encoder, audio decoder (with attention) and super-resolution network. The composition of the network is similar to Tacotron (see Section 2.2.3). At first, the text encoder extracts key and value matrices K, V and the audio encoder converts the target spectrogram (or its already generated part during inference) using causal convolutions to a query matrix Q. The attention is then formulated as:

$$A = \text{softmax}\left(\frac{K^T Q}{\sqrt{d}}\right)$$

where d is the number of channels (in other words, the size of the last dimension, i.e., not the temporal dimension) of Q, K, and V. Subsequently, the matrix $R = VA$ is concatenated with Q and passed through the audio decoder. The decoder produces a coarse spectrogram (or a single coarse frame during inference). The training target of the coarse spectrogram is the spectrogram obtained by picking every fourth frame of the real ground-truth spectrogram. This intermediate representation is then upscaled into full temporal resolution using the super-resolution network.

All the modules use convolutional layers, possibly with ReLU activations, or convolutional highway layers with various parameters.

The *guided attention* proposed by Tachibana et al. [2017] incorporates more prior knowledge into the attention mechanism. There is a rough correspondence of the order of input text characters and output audio segments. It can be assumed that reading has a nearly linear tempo. That is why DCTTS uses an additional loss term called guided attention loss. It constrains the attention matrix to be roughly diagonal:

$$\mathcal{L}(A) = \mathbb{E}\left[AW\right]$$

where

$$W_{n,t} = 1 - exp\left(-\frac{\left(\frac{n}{N} - \frac{t}{T}\right)^2}{2g^2}\right)$$

and T is the total number of decoder time steps and N is the length of the input text; g controls strictness of the diagonal.

2.2.7 Others

There are a plenty of other interesting text-to-speech models. We do not use them in our experiments, but for the sake of completeness, we would like to mention some of them:

- **WaveGlow** [Prenger et al., 2019] – a non-autoregressive (i.e., fully parallelizable) alternative to WaveNet that makes use of normalizing flows and can produce high-quality audio at very high sampling rates. It has an official open-source implementation, but in our experiments, we preferred WaveRNN because because it has lower computational requirements.
- **GAN-TTS** [Bińkowski et al., 2020] – yet another non-autoregressive alternative to WaveNet. It is based on a generative adversarial network generating raw waveforms given a text with linguistic features.
- **Voice Loop** [Taigman et al., 2018] – an approach to text-to-speech completely different from other models described here. It is inspired by a concept of human memory called the phonological loop [Baddeley, 1984], i.e., it has a shifting buffer (its oldest item is at each step replaced by a new context vector) that is used for generating acoustic parameters.
- **MelNet** [Vasquez and Lewis, 2019] – a model generating spectrograms with an ability of modeling longer time dependencies. It is based on many recurrent layers that generate spectrograms in multiple tiers. The first one dictates high-level structure and subsequent tiers add fine-grained details. However, the results reported by Vasquez and Lewis [2019] have not been replicated yet.

Chapter 3

Evaluation Metrics

This chapter is devoted to common objective and subjective metrics and methodologies for performance evaluation of text-to-speech models. Direct comparison of text-to-speech models is difficult, similarly to a comparison of text or image generation models.

There is no objective metric that would measure TTS quality directly; instead, surrogate metric need to be used where correlation with TTS quality is not guaranteed. On the other hand, evaluation results with subjective metrics may highly depend on subjective preferences.

In the following, we list two objective metrics – mel cepstral distortion (see Section 3.1) and character error rate (see Section 3.2), then describe two methods for subjective evaluation – mean opinion score (see Section 3.3) and MUSHRA (see Section 3.4). We also mention how the individual metrics are relevant to our experiments in Chapter 6.

3.1 Mel Cepstral Distortion

The first objective metric we would like to mention is called *mel cepstral distortion* (MCD). It is based on a comparison of MFCCs obtained from reference and generated mel spectrograms [Kubichek, 1993]. It reflects the differences between prosody, pronunciation, voice similarity, etc., of speech recordings that correspond to the two spectrograms. It is defined as:

$$\mathrm{MCD}(c, c') = C \cdot \frac{1}{T} \sum_{t=0}^{T-1} \sqrt{\sum_{k=1}^{K} (c_{t,k} - c'_{t,k})^2}$$

where $c_{t,k}$, $c'_{t,k}$ denote the k-th mel frequency cepstral coefficient of the t-th frame of the predicted and reference recordings. C is a normalization constant and T is the total number of frames in the recording. The squared differences are summed over the first K coefficients except for the first one (see Section 1.1.2, the first coefficient carries just information about the average power of the input signal).

However, the above definition expects that the two spectrograms have the same number of frames, which is usually not true. This problem can be solved by padding the shorter spectrogram or cutting the longer one, but a more complex method utilizing *dynamic time warping* (DTW) will often provide more accurate comparison of different systems. Dynamic time warping is a pattern-matching

algorithm [Brown and Rabiner, 1982] that can stretch or shrink some parts of input series in order to optimize a metric (i.e., slowing down or speeding up certain parts of the audio to maximize MCD in our case).

Reported values of MCD vary in text-to-speech literature [Skerry-Ryan et al., 2018, Chen et al., 2019], i.e. even the same model with the same outputs could obtain different MCD values. This is because the computation of MCD relies on MFCCs, but the computation of MFCCs is implemented differently across popular libraries (e.g., Librosa [McFee et al., 2015] or Matlab). The implementations mainly differ in the type of normalization used.

In Chapter 6, we refer to MCD values obtained from MFCCs computed using default parameters of Librosa and with a unit normalization constant C. We also use DTW to overcome the problem with spectrogram lengths. We use this metric mainly to monitor the training progress because it can be computed easily and reflects the speech abilities more properly than typical loss functions[1] (such as the mean square error of generated spectrograms).

3.2 Character Error Rate

Some works make use of models for automatic speech recognition in order to objectively measure pronunciation accuracy or stability of text-to-speech systems (Battenberg et al. [2019], Lee et al. [2018]). Generated waveforms are sent to a speech recognition engine that outputs transcriptions. These are then compared to ground-truth texts that were originally passed into the TTS model. The comparison itself is done on the character level using the *character error rate* (CER) metric [Soukoreff and MacKenzie, 2001]. The CER of two strings is their *minimum string distance* (i.e., the minimal number of operations – substitutions, insertions, and deletions – needed to transform one string into the other) divided by the length of the longer one.

There might be difficulties with the calculation of the CER of logographic scripts, such as Chinese or Japanese. Text-to-speech models for these languages often use romanized forms of input texts, but ASR systems output native characters. It is possible to compare the native symbols directly, but we should not be surprised by higher CER values in comparison to sentences using alphabetic scripts. The alternative is to romanize the ASR output and compare the forms using Latin characters. However, conversion errors can compound.

Of course, the character error rate in the context of text-to-speech is not an ideal measure as it depends on the capabilities of the particular ASR module and the quality of synthesized waveforms. It does not depend just on correct pronunciation – artifacts produced by a lower-quality vocoder such as Griffin-Lim [Griffin and Lim, 1984] can cause ASR errors even though spectrograms produced by the system being evaluated are perfect. On the other hand, the ASR engine can guess some characters which are not pronounced at all.

Despite all these facts, we use this metric extensively in our experiments for

[1] See "Loss function" in the Glossary.

evaluation and comparison of our models as finding tens of native speakers for each of ten languages who are willing to rate tens of recordings is very difficult and expensive. In the experiments in Chapter 6, we use the ASR model available at Google Cloud Platform.[2] In the case of Chinese and Japanese, we compare their native characters.

3.3 Mean Opinion Score

In current text-to-speech literature, it is very common to report the so-called *mean opinion score* (MOS). It was originally used to evaluate phone call quality and it was defined as "*The mean of opinion scores*". Opinion score was further specified as "*The value on a predefined scale that a subject assigns to his opinion of the performance of the telephone transmission system used either for conversation or for listening to spoken material*" [ITU-T, 2016a]. Note that the mean is a simple arithmetic mean. The recommendation also precisely describes the methodology of data collection (room conditions, loudness, etc.). Another standard that describes quality assessment for multimedia applications introduces several scales compatible with the mean opinion score [ITU-T, 2008]:

- **Five-point scale** from 1 to 5 with labels *Bad*, *Poor*, *Fair*, *Good*, *Excellent*, respectively. The labels should indicate quality and should support agreement across users. However, it is questionable whether the labels are perceived equidistant.
- **Finer five-point scale** with the possibility of selecting an intermediate rating. The original five-point scale described above can be, for example, refined to a nine-grade or an eleven-grade scale.
- **Quasi-continuous scale** that only has labels for the beginning (*Bad*) and the end (*Excellent*). Other labels are omitted to reduce the bias caused by their interpretation.

In current text-to-speech literature, MOS is implemented loosely and does not precisely follow the standards. For example, Park and Mulc [2019] redefine the scale using different five labels, Shen et al. [2018] use a finer discrete scale from 1 to 5 with 0.5 increments, or Skerry-Ryan et al. [2018] use a seven-grade scale with custom labels and reference recordings. Evaluation conditions and data are not standardized and the result might depend on a wide range of factors. ITU-T [2016b] extensively describes how to report MOS results and states that MOSs collected in different experiments and contexts are not directly comparable.

In the experiments in Chapter 6, we use a subjective evaluation test to compare multiple models with different architectures. The evaluation method follows this metric and uses two finer five-point scales with the same labels as mentioned.

[2]Cloud Speech-to-Text, https://cloud.google.com/speech-to-text, accessed April 27, 2020

3.4 MUSHRA

Latorre et al. [2019] and Lee and Kim [2019] use an alternative methodology for subjective audio quality evaluation. It is called *Multiple Stimuli with Hidden Reference and Anchor* (MUSHRA) and it was defined by ITU-T [2015]. In comparison with MOS, samples of all compared systems are presented to the particular subject at the same time (MOS presents a pair or a single recording). The concurrent comparison aids more consistent ratings and enables usage of a paired t-test for later statistical analysis.

MUSHRA uses a discrete scale from 0 to 100 without any labels. The calibration of subject ratings is done via so-called *reference* and *anchor* recordings. The reference is usually a ground-truth recording and the anchor is a degraded version of the reference. These two recordings are rated together with the evaluated systems' outputs (it is not revealed to the evaluators which recordings are system predictions and which are references or anchors). They also provide a sanity-check for user ratings as we may expect ratings of the reference to be close to 100 and ratings of the anchor to be much lower.

Unfortunately, MUSHRA requires reference recordings which might not be available for some tasks. However, we adopted the idea of simultaneous rating of multiple systems and we used it in our subjective evaluation test (see Chapter 6).

Chapter 4

Datasets

It is not surprising that data-driven models such as neural networks strongly rely on training data. As described in Chapter 2, it is possible to automatically synthesize a near-human-like speech. However, we cannot expect that the quality or naturalness of generated speech will be better than the speech in the system's training data.

Making a high-quality text-to-speech dataset can be difficult and expensive. First, studio-quality audio has to be recorded. Then it has to be segmented and aligned with transcripts. Transcripts have to be cleaned and normalized. Thanks to the abilities of end-to-end systems, we do not need to extract any further features from transcripts and thus recent TTS datasets contain just audio-transcript pairs. Each of the mentioned steps has some difficulties:

- **Recording** – Studio-quality recordings require a studio with equipment and possibly a professional speaker. The speaker should read texts consistently and in a way that corresponds with the eventual TTS system's use – it can be more or less expressive, it can vary in tempo, etc. The production of recordings has also legal issues: in the end, a TTS system will allow us to synthesize an arbitrary utterance in the natural voice of the original speaker and it could be misused.
- **Segmentation** – The choice of suitable segmentation criteria is non-trivial. Most likely, we would like to synthesize shorter utterances in the order of sentences, so a sentence-based segmentation seems to be good. On the other hand, a balanced distribution of segment durations (such as a normal distribution) is desirable. However, splitting on the sentence level tends to produce imbalanced duration distributions. Another segmentation strategy splits recordings by silence. In contrast to automatic speech recognition, the silence-based segmentation might not be desired for the text-to-speech task (i.e., our training transcripts can start or end mid-sentence, which may hurt prosody produced by a system trained on this data). There are also other important segmentation criteria. For example, it is practical to have phonetically balanced data, i.e., a roughly linear dependence between segment durations and corresponding transcript lengths.
- **Alignment** – Another set of problems originates from aligning transcripts with recordings. It is costly or impossible to do that manually, and usually a forced aligner is used to do that (for example, Aeneas [Pettarin, 2017] or Montreal Forced Aligner [McAuliffe et al., 2017]). Automatic aligners have

a limited precision, which can result in slightly shifted alignments. This is a common problem of open-source datasets. In addition, there are often missing or extra words at the beginning of transcripts (in comparison with audio recordings), and models trained on these data tend to skip the first or last words of the input. Trailing or leading silence can also cause problems with stability.

- **Cleaning** – Finally, the cleaning of transcripts includes spelling out numbers, abbreviations, and other rare symbols. This brings about more problems, e.g., it can happen that the transcripts come from multiple different sources (e.g., books of diverse writers, publishers, and editors or books written in different languages) and then there are different writing styles with slightly different usage of punctuation, lengths of sentences, etc., which needs to be addressed.

Some of the problems mentioned above can be avoided by taking a different approach to data collection. Here, data collection participants read out presegmented and cleaned utterances (e.g., single sentences), which are recorded in isolation. This kind of data acquisition is common for massively multi-speaker datasets (such as VCTK described in Section 4.1). Even though it removes problems with segmentation, other problems may occur; for example, the readings are often very poor in terms of prosody (i.e., all utterances are read in the same monotonous way).

In comparison with ASR datasets (e.g., Ardila et al. [2019]), there are more demands on the quality of TTS data. Even smaller, properly built datasets with high-quality recordings can be more useful than huge and noisy datasets.

Unfortunately, a lot of recent TTS works mention proprietary datasets for training and evaluation (e.g., Zhang et al. [2019]). Their authors probably cannot publish the datasets because they are essential for successful commercial use.

We are now going to summarize the current publicly available TTS datasets with short characteristics and descriptions of their pros and cons, focusing especially on the datasets used in our experiments. First, we mention monolingual datasets in Section 4.1 and than multilingual in Section 4.2. We will describe our own data cleaning procedures in Chapter 6.

4.1 Monolingual Datasets

There are several high-quality and widely used English datasets (both single-speaker and multi-speaker) because current research mainly focuses on English. Beside these datasets, there is not much publicly available text-to-speech data. ASR datasets, such as the Free ST Chinese Mandarin Corpus [Surfingtech, 2017], are used instead.

Since we focus on multilingual text-to-speech, we mention them here only for completeness. We used LJ Speech for preliminary experiments and sanity checks.

LJ Speech

One of the recently popular English text-to-speech datasets is the LJ Speech dataset [Ito, 2017]. It is suitable for quick experimenting because it is relatively small and high-quality. Therefore, many open-source implementations of neural models described in Chapter 2 (e.g., [Park, 2018], [Ito, 2018], [Mama, 2018]) provide model weights trained on this dataset.

LJ Speech contains almost 24 hours of American English recordings read by Linda Johnson. It was built from seven non-fiction books from the LibriVox project.[1] LibriVox is maintained by volunteers and it contains a lot of audiobooks released into the public domain. Transcripts were obtained from Project Gutenberg.[2]

LJ Speech audio data are sampled at 22,050 Hz and saved in the 16-bit PCM format. The recordings contain subtle background noise and the speaker's voice has quite strong sibilants. The dataset includes 13,100 examples that consist of transcripts, normalized transcripts and audio clips with the mean duration of 6.57 seconds and the standard deviation of around 2.18 seconds. The range of durations spans an interval from 1.1 seconds to 10.1 seconds. Figure 4.1 shows some properties of audio duration and utterance length distributions. We can notice a nearly linear dependency between audio durations and utterance lengths.

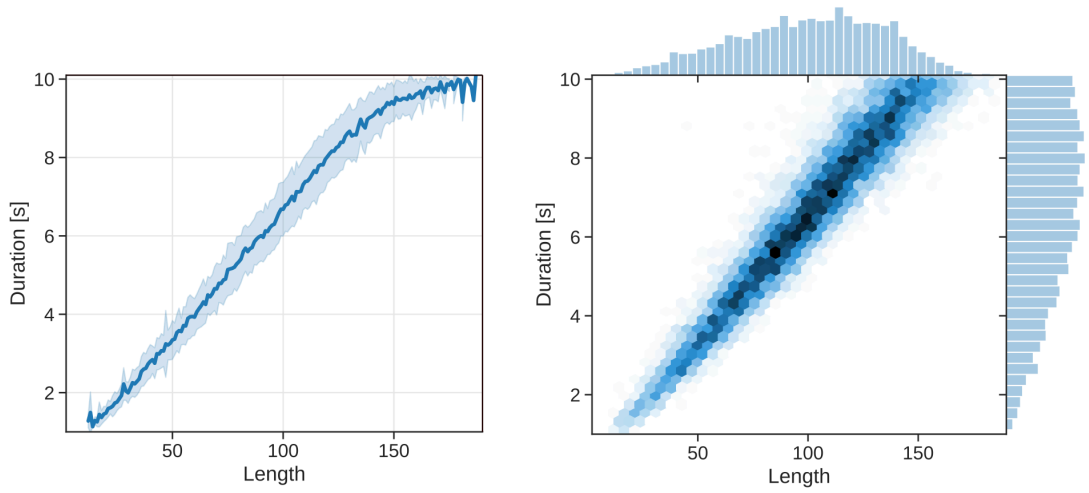

Figure 4.1: Left: *Dependency between lengths of transcripts and durations of corresponding recordings. The solid blue line depicts the mean of durations corresponding to the particular lengths. The light blue area shows the standard deviation of the durations.* Right: *Visualization of distributions of transcript lengths and audio durations (vertical axis and horizontal axis, respectively) together with their joint distribution. Darker color indicates higher relative frequencies.*

The segmentation was done by splitting the original text by sentence or clause boundaries. The alignment with audio recordings was created manually and thus it has a very good quality. This is why the LJ Speech dataset does not suffer from the segmentation and alignment problems described in the previous section.

[1] LibriVox, free public domain audiobooks, https://librivox.org, accessed February 29, 2020
[2] Project Gutenberg, https://www.gutenberg.org, accessed February 29, 2020

VCTK

LJ Speech contains recordings of a single speaker. On the other hand, the Centre for Speech Technology Voice Cloning Toolkit Corpus [Veaux et al., 2017], abbreviated VCTK, contains data from many different speakers. As the name suggests, it was built to study voice cloning abilities of TTS systems [Jia et al., 2018]. The VCTK data contain text-audio pairs for 108 speakers with different English accents. For each speaker, it encompasses more than four hundred recordings. All sentences used in this dataset were picked from newspapers by a greedy algorithm which tried to maximize the phonetic and contextual coverage.

LibriTTS

LibriTTS is a large multi-speaker dataset [Zen et al., 2019]. It was derived from an ASR dataset called LibriSpeech [Panayotov et al., 2015], and it contains around 585 hours of speech data from 2,456 speakers. Zen et al. [2019] took the original audiobooks from LibriSpeech and applied the following processing pipeline to them: they removed noisy audiobooks, resampled all recordings to 24 kHz, created a sentence-level segmentation using a proprietary sentence splitter, normalized all transcripts using the Kestrel system [Ebden and Sproat, 2015], and used the engine for YouTube's auto-sync feature to align recordings with texts.

4.2 Multilingual Datasets

Multilingual datasets are more important for this work. Some of them, namely CSS10 [Park and Mulc, 2019], M-AILABS [2019], or TUNDRA [Stan et al., 2013], consist partially or completely of LibriVox audiobooks (same as LJ Speech or LibriTTS, see Section 4.1). One may easily conceive creating a large and very diverse multilingual dataset using LibriVox because it contains audiobooks in nearly one hundred languages. However, there are not many usable LibriVox books for the vast majority of the languages. At the time of writing this work, there is, for example, only one book entirely in Czech. Furthermore, the book is read by a Slovak native speaker with a really strong accent. Most of the usable languages of LibriVox audiobooks (besides English) are already included in CSS10 or TUNDRA datasets.

We chose CSS10 and a subset of Common Voice [Ardila et al., 2019] for our experiments (see Chapter 6). In the following sections, we mention the aspects we considered for each dataset.

M-AILABS

The M-AILABS Speech Dataset [M-AILABS, 2019] is a very large collection of data from various sources. It is intended to be used for both ASR and text-to-speech research. It focuses on European languages and contains data for nine of them (German, English, Spanish, Italian, Ukrainian, Russian, French, and Polish). A lot of recordings are in German (around 237 hours) and French (190

hours). The data were collected from very different sources. That is why they have diverse sampling rates and quality. Some languages have just LibriVox audiobooks; German data, for example, contain many hours of Angela Merkel's speeches, etc. Ukrainian data come from commercial companies which provided them to M-AILABS for machine learning research purposes only. Spanish data contain speakers from both Spain and Latin America. An advantage of this dataset is that every language has several different speakers.

The enormous quantity of data implies that the costs of manually checking all the data for correctness would be excessive. The segmentation and alignments were created by an automatic tool. Missing or extra words at the beginning and end of transcripts are extremely frequent. Even though the dataset provides raw and normalized texts (except for French data), even the normalized variants require further extensive cleaning to make the whole dataset usable.

We tried to use this dataset for training our models, but the results were very poor due to noisy segmentation and alignments. It was also not very practical to work and experiment with such a huge amount of recordings.

CSS10

The CSS10 dataset [Park and Mulc, 2019] offers a smaller, but cleaner alternative to M-AILABS. It contains ten different languages including non-European languages (namely German, Greek, Spanish, Finnish, French, Hungarian, Japanese, Dutch, Russian, and Chinese). It is based on LibriVox audiobooks. Every language has up to 24 hours of single-speaker audio data (see Table 4.1), so it is not nearly as large as M-AILABS. However, the size is sufficient for text-to-speech experiments.

Table 4.1: *Summary of CSS10 data properties grouped by language. The second column provides information about the total length of all recordings. The third column shows mean durations of individual examples, with the corresponding standard deviations. The last column shows means and standard deviations of transcript lengths (number of characters).*

Language	**Recordings Total Duration [h]**	**Recordings Mean Duration [s]**	**Transcripts Mean Length**
German	16.13	7.82 ± 1.73	100.2 ± 25.9
Greek	4.14	8.08 ± 2.35	119.9 ± 40.9
Spanish	23.83	7.72 ± 2.10	112.6 ± 37.8
Finnish	10.53	7.83 ± 1.78	95.4 ± 26.8
French	19.15	7.97 ± 2.09	110.9 ± 36.5
Hungarian	10.01	7.98 ± 1.63	110.0 ± 28.6
Japanese	14.92	7.86 ± 1.45	118.8 ± 27.9
Dutch	14.11	7.82 ± 2.28	121.2 ± 41.2
Russian	21.37	8.01 ± 2.28	91.4 ± 34.9
Chinese	6.45	7.82 ± 2.05	118.3 ± 38.7

The segmentation was created automatically, and it follows silent pauses in original audiobooks. However, very short segments obtained by splitting recordings on silence were concatenated to reach a segment duration of approximately 10 seconds. So the segmentation does not follow sentence or clause boundaries, i.e., examples usually start in pauses in the middle of sentences. Recordings are sampled at 22,050 Hz. Figure 4.2 shows the distributions of text lengths and audio durations of German and French CSS10 data; these are very similar for all CSS10 languages. The alignment and the normalization of texts were done manually by experts. Chinese and Japanese were romanized and punctuation was preserved.

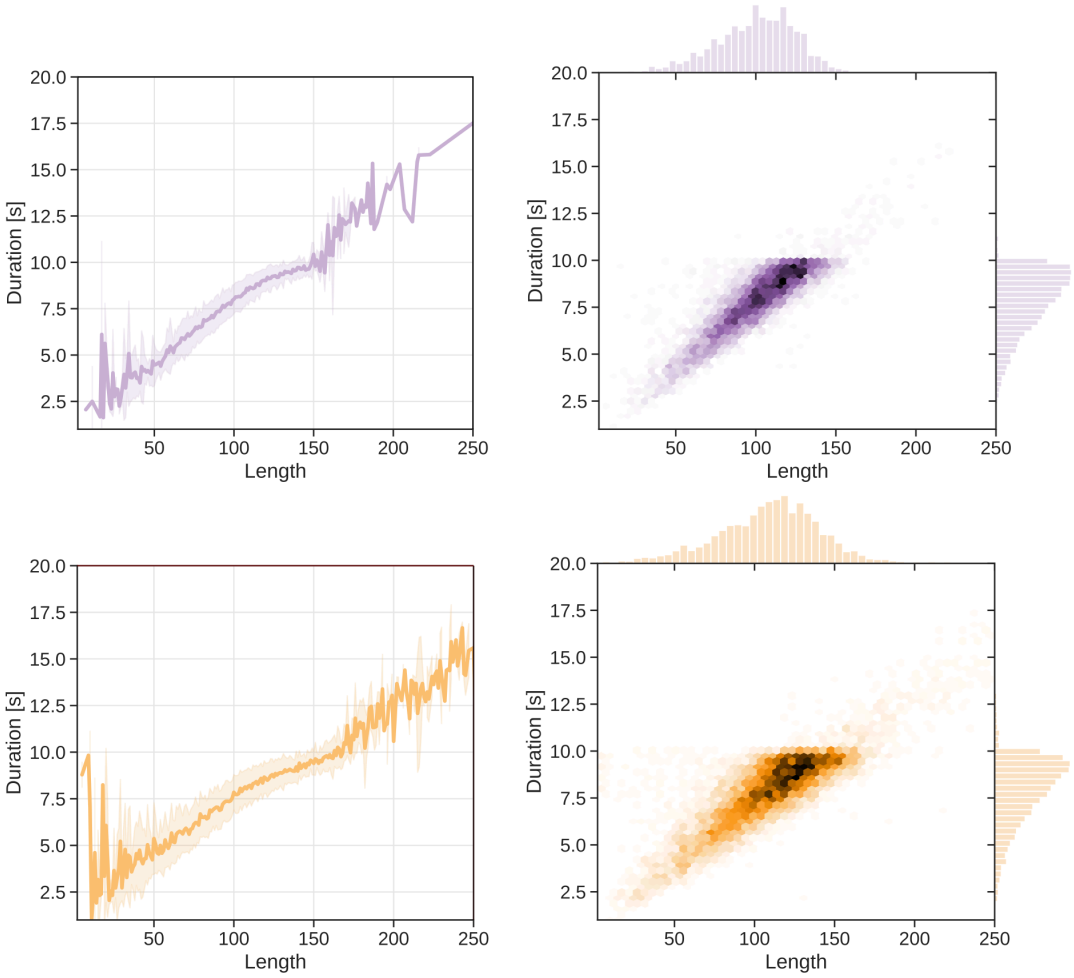

Figure 4.2: *Graphs of transcript length and recording duration distributions for German (top row) and French (bottom row) data in CSS10. The solid line in the left graph depicts the mean of durations corresponding to the particular lengths. The shaded area shows the standard deviation of the durations. The right graph visualizes distributions of audio durations (vertical axis) and transcript lengths (horizontal axis) together with their joint distribution. Darker color indicates higher relative frequencies.*

We chose CSS10 for our experiments because it represents a manageable size and sufficient quality. However, in initial experiments, we found out that some alignments are not precise and the normalization is sometimes imperfect. That is why we apply cleaning procedures to the dataset in our experiments (see Section 6.2).

TUNDRA

CSS10 provides more than ten hours of segmented and aligned audio data for eight of its languages. The dataset called TUNDRA [Stan et al., 2013] is smaller than that. It contains data in fourteen European languages; Table 4.2 summarizes the amounts of data available for each of them (Stan et al. [2013] declare slightly more aligned data than is, in fact, present on the download page of the dataset). Again, the recordings come from LibriVox audiobooks and the transcripts come from Project Gutenberg; there is one book for each language. The segmentation and alignments were done by a semi-supervised algorithm and only data segmented with high confidence (as predicted by the segmentation algorithm) were preserved. Consecutive segments with duration shorter than 5 seconds were concatenated.

We do not use this dataset in our experiments in Chapter 6 because it provides less data than is required by some TTS models.

Table 4.2: *Properties of the TUNDRA dataset. "*" denotes languages that are also included in the CSS10 dataset. The figures in parentheses are equal to durations declared by Stan et al. [2013], which slightly differ from the amounts available for downloading.*

Language	**Recordings Total duration [h]**	**Aligned recordings Total duration [h]**	**Number of Examples**	**Speaker Gender**
Bulgarian	6.1	4.1 (4.1)	3,139	Female
Danish	2.1	0.7 (1.1)	1,099	Male
*Dutch	6.5	4.5 (4.9)	3,844	Male
English	4.5	2.3 (2.4)	2,194	Female
*Finnish	3.1	2.5 (2.6)	1,357	Female
*French	4.0	2.1 (2.3)	1,890	Male
*German	9.5	7.9 (8.0)	4,865	Male
*Hungarian	8.5	5.0 (5.0)	4,510	Female
Italian	6.5	3.5 (5.0)	2,241	Male
Polish	3.1	2.6 (2.9)	2,078	Female
Portuguese	9.3	5.2 (5.2)	5,001	Female
Romanian	11.1	6.5 (7.0)	5,563	Female
*Russian	2.1	1.3 (1.6)	1,113	Male
*Spanish	12.1	8.0 (8.0)	7,902	Male

Common Voice

Multilingual datasets described so far were build on LibriVox audiobooks. Common Voice [Ardila et al., 2019], primarily aimed at ASR research, adopts another approach to building speech datasets. Anybody can visit the official Common Voice website[3] and create recordings. Participants are given short texts and are expected to read it aloud and record themselves. All recorded utterances are then

[3]Common Voice, https://voice.mozilla.org, accessed April 27, 2020

validated by other users. An advantage of this approach is that it enables creating a massively multi-speaker multilingual dataset. The Common Voice dataset quickly grows and at the time of writing this text, it has 40 languages from all around the world (e.g., Indonesian, Persian, Kinyarwanda, or Welsh) and, for example, English has 1,118 validated hours of audio collected from 51,072 speakers with various accents.

However, while this unmoderated way of collecting data is sufficient for ASR data collection, it has many drawbacks for TTS research, as we found upon manual inspection of selected examples. Some recordings are really noisy. The noise is caused by bad microphones, but also by room conditions (ambient noise such as traffic, playing children, animals, TV or radio, etc. in the background). The vast majority of recordings have a period of silence at the beginning and the end. Nevertheless, this is not a complete silence because speakers need to press a button to start and stop recording and this click is often present in the recordings. Another problem is that some users take an audible deep breath before speaking. All these imperfections imply that recordings cannot be trimmed easily. It is also not surprising that the recordings are rather short and their reading style is often rather monotonous.[4]

We use a cleaned subset of this dataset corresponding to the CSS10 languages. We will describe the subset we use for our experiments and the process of its preparation in Section 6.2.

CMU Wilderness

Finally, we would like to mention the CMU Wilderness dataset [Black, 2019]. Unlike CSS10, TUNDRA, or M-AILABS, it is not built on top of LibriVox audiobooks. It is based on the Bible.is project[5], which is, in our opinion, an excellent source of TTS data. Bible.is contains audio versions of various editions of the Bible in hundreds of languages. For example, the Czech audiobook is read by Alfréd Strejček and its quality is very high. A potential problem is that some recorded segments contain music and other sounds, as commercial audiobooks usually do. Therefore, a speech separation algorithm would be required to use these data. However, redistribution, modification, and adaptation of the content provided by Bible.is is prohibited.

The CMU Wilderness dataset contains information for the recreation of segments and alignments (which were created automatically) for 700 different languages. Most of them are from equatorial Africa, South America, and Indonesia. European and other more frequent languages are only included marginally. These data were, for example, used by Boito et al. [2019] for creating a parallel speech-to-speech corpus.

[4]On the other hand, we also found a participant who was singing all the interpreted texts.
[5]Bible.is | Read. Listen. See., http://www.bible.is, accessed February 29, 2020

Chapter 5

Multilingual Speech Synthesis

We are now going to discuss related work that has already been done in the field of multilingual end-to-end speech synthesis. First, we explain what the goals, applications, and benefits of multilingual speech synthesis are:

- It is possible to make use of *high-resource* languages for training text-to-speech systems for *low-resource* languages. This is usually done via transfer learning approaches. At first, models are pre-trained on a large monolingual dataset and then they are fine-tuned to a different language [Lee et al., 2018].
- One may think of using multilingual data for joint training of a single text-to-speech model. Intuitively, this sharing of cross-lingual knowledge can positively influence the performance and robustness of the resulting model. To the best of our knowledge, this usage of multilingual data in speech synthesis has not yet been explored. In a similar task, Latorre et al. [2019] examined the effects of using multi-speaker data in a single language.
- It is sometimes needed to synthesize speech in multiple languages, but preserve the same voice (e.g., in translational systems). This task is called *cross-lingual voice cloning*. However, there is not a lot of training data available where a single speaker speaks multiple languages. That is why systems for cross-lingual voice cloning need to be trainable using mixtures of monolingual data (with different speakers across languages).
- Closely related to cross-lingual voice cloning is the so-called *code-switching*. In this task, we would like to alternate languages within a single utterance, i.e., code-switching text-to-speech systems have to be able to read sentences that consist of words from different languages. This ability is extremely useful, e.g., for car navigational systems or station announcements, where a large number of foreign names is to be expected.

In the following, we describe the latest research works on neural multilingual TTS in detail. We focus on three areas sketched above in particular – low-resource languages (Section 5.1), cross-lingual voice cloning (Section 5.2), and code-switching (Section 5.3).

In our experiments in Chapter 6, we explore benefits of joint multilingual training (even in low-resource settings) and we compare code-switching and voice cloning abilities of models that closely follow the architectures by Zhang et al. [2019] and Cao et al. [2019], which are also described in the following sections.

5.1 Low-resource Languages

Lee et al. [2018] investigated pre-training and fine-tuning with multilingual data. For training, they used around 60 hours of phonemicized Korean audio-transcript pairs. Subsequently, they separately fine-tuned the model to the languages from the CSS10 dataset (see Section 4.2). During the fine-tuning steps, they used only two hours of phonemicized data for each target language.

Chung et al. [2019] showed that it is possible to produce intelligible speech with the Tacotron model (see Section 2.2.3) trained on only 24 minutes of English audio-transcript pairs. They first pre-trained the decoder with large-scale unpaired or unlabeled English speech data and initialized the encoder with previously pre-trained word embeddings. Then, they fine-tuned the model with the small amount of audio-transcript pairs. This smart initialization can be used in the case of languages with lack of transcribed or aligned audio data.

A recent work by Chen et al. [2019] aimed at adaptation of pre-trained Tacotron to low-resource languages with even less data. They used English data for pre-training, and they applied Tacotron that expects phonemicized texts on the input. They evaluated their technique on German, French, and Chinese and showed that they are able to produce a relatively good TTS system with as little as 15 minutes of paired low-resource training data.

Chen et al. [2019]'s training method has three stages. First, they jointly train an ASR model and Tacotron on high-resource data. The ASR module predicts phonemes that are passed into Tacotron. Subsequently, they make use of the low-resource data. They fix the ASR module and train another neural network called the phonetic transformation network (PTN) which learns a mapping between outputs of the ASR module and characters of the low-resource language. The target mapping required for synthesis in the other language is then the inverse of the mapping learned by PTN. This means that zero or more characters from the high-resource language can be mapped to a single characters of the low-resource language. If source characters are ambiguous, authors simply take the one with the highest probability, and characters that are not mapped by any source character are trained from scratch. So the model is able to obtain a mapping from characters of the low-resource language to characters of the language originally used for the pre-training. Finally, they fine-tune the pre-trained Tacotron model with the low-resource data with transcripts modified using the mapping.

Prakash et al. [2019] tried to overcome scalability issues of multilingual text-to-speech for Indian languages which are caused by the usage of many different scripts (they considered 13 languages with 8 different scripts). They propose a unified character representation called the multi-language character map (MLCM) and a unified phone representation called the common label set (CLS). The unified representation makes it possible to handle all the languages together by a single model.

5.2 Voice Cloning

Strong results in multilingual multi-speaker voice cloning were presented by Zhang et al. [2019]. The work builds on domain-adversarial training techniques proposed by Ganin et al. [2016] and it closely follows Hsu et al. [2019]. The proposed model is based on Tacotron 2 (see Section 2.2.4) conditioned on phonemes.[1] The model was trained on English, Spanish, and Chinese data. The authors used a proprietary dataset that comprises 385 hours of English from 84 professionals with various accents, 97 hours of Spanish from 3 speakers, and 68 hours of Chinese from 5 speakers. This huge amount of professional high-quality data, unfortunately, makes it very difficult to reproduce the results. Nonetheless, they describe conditions critical to achieving good performance:

- *Using a phonemic input representation to encourage sharing of model capacity across languages.* – This reduces demands on the complexity of the encoder, because it does not have to learn pronunciation anymore. However, such an approach requires an algorithm for converting graphemes to phonemes. Zhang et al. [2019] concatenated stress embeddings for Spanish and English and tone embeddings for Chinese to the input phoneme embeddings. This further reduces the information to be learned from raw text data by the encoder, but it also requires preprocessing and external tools.
- *Incorporating an adversarial loss term to encourage the model to disentangle its representation of speaker identity from the speech content.* – During training, Zhang et al. [2019] apply an adversarial speaker classifier to encoder outputs and revert gradients[2] flowing from this classifier. They are clipping the reverted gradients as some encoder outputs can be highly language-dependent and this could cause unstable training.[3] In this way, they try to obtain a language- and speaker-independent encoder and to move any language-dependent processing to the decoder.
- *Scaling up the model by training on data from multiple speakers of each language, and incorporating an autoencoding input to help stabilize attention during training.* – The decoder is conditioned on a language embedding and a speaker embedding. It means that multi-speaker multilingual data are needed for training. The conditioning is done by concatenating the embeddings to encoder outputs. Besides these embeddings, the decoder is conditioned also on an autoencoding input. This input is obtained by a separate variational autoencoder[4] with an architecture proposed by Hsu et al. [2019]. The autoencoder is conditioned on ground-truth spectrograms and outputs low-dimensional vectors in a latent space which should reflect prosodic variation in training examples. During inference, an all-zero vector is used instead of the autoencoding input. While the autoencoder improves performance, a very large batch size is required to stabilize its training.

[1] Zhang et al. [2019] also experimented with byte and character inputs, but they evaluated voice-cloning performance only on phoneme inputs; according to them, it guarantees correct pronunciation and more fluent speech.

[2] See "Gradient" in the Glossary.

[3] See "Gradient clipping" in the Glossary.

[4] See "Variational autoencoder" in the Glossary.

Figure 5.1 shows the architecture of the modified Tacotron 2 with speaker and language embeddings, together with the newly proposed modules which are used only during training, i.e., the adversarial classifier and the residual autoencoder.

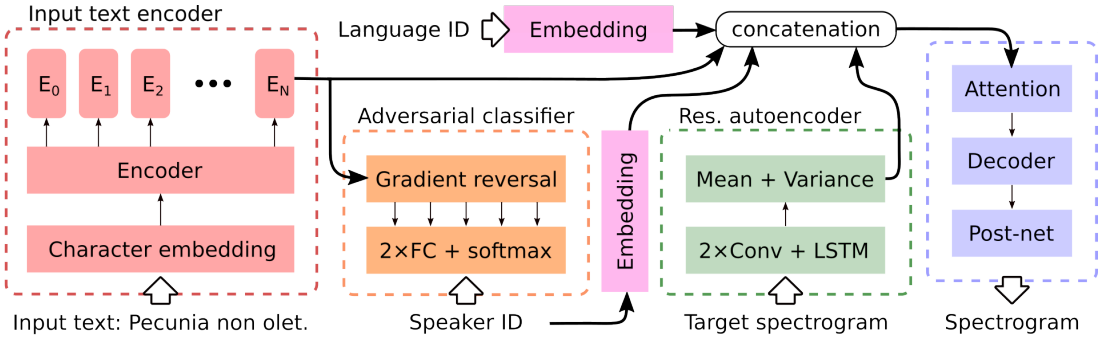

Figure 5.1: *Visualization of Zhang et al. [2019]'s modified Tacotron 2 that supports cross-lingual multi-speaker voice cloning. In comparison with the original Tacotron 2 model (see Section 2.2.4), the modified model contains an adversarial speaker classifier with a gradient reversal layer (orange), a residual encoder that encodes target spectrograms into a low-dimensional latent space (green), and language or speaker embeddings (pink) which are concatenated to outputs of the encoder. The adversarial classifier includes two fully connected layers with an output softmax layer. The autoencoder (green) consists of two one-dimensional convolutional layers that are followed by a single LSTM layer. Its final hidden state is passed into separate fully connected layers that predict vectors of means and variances of the residual autoencoding input.*

Nachmani and Wolf [2019] extended the Voice Loop architecture (mentioned in Section 2.2.7) to enable cross-lingual voice cloning. The authors showed conversion abilities in three languages – English, Spanish, and German. The model uses multiple text encoders, one per language, and a shared network that outputs a speaker embedding. To preserve speakers' identities during language conversions, a special loss term is applied to the space of speaker embeddings. For training, they used the English VCTK dataset (see Section 4.1) together with Spanish DIMEx100 [Pineda et al., 2009] and German PhonDat.[5]

5.3 Code-switching

Zhang et al. [2019]'s voice-cloning technique described in the previous section can also be used for reading texts that consist of words from multiple languages. It is enough to change character-level language embeddings which are concatenated to encoder outputs so that they correspond to the true language of the particular words. However, the primary subject of the work was to enable cross-lingual multi-speaker voice cloning.

On the other hand, Cao et al. [2019] focused directly on code-switching and for this purpose, they proposed some modifications of Tacotron 1 (see Section 2.2.3).

[5]Phonetik BAS, https://www.bas.uni-muenchen.de/forschung/Bas/BasKorporaeng.html, accessed April 28, 2020

They used a relatively small amount of data – 8 hours of Chinese and 7 hours of English single-speaker data. Both the English speaker and the Chinese speaker are females with very similar voices, so the requirements on voice cloning abilities to make the model sound convincing are not high. Unlike other mentioned multilingual approaches, this model expects grapheme inputs without any tone or stress labels. Note that the model uses romanized forms of Chinese texts.

Cao et al. [2019] suggest two simple, alternative modifications of Tacotron:

(a) *Adding a separate language-specific encoder for each language.* The code-switching itself is then done by masking and combining outputs of the particular encoders, i.e., input texts are passed through all encoders and the encoded sequences are mixed before decoding.

(b) *Having a language embedding that is incorporated into a single shared encoder in various places.* Code-switching can then be performed by changing character-level language embeddings.

The authors state that the approach with separate encoders outperformed the other approach on all evaluation settings.

Chapter 6

Experiments

We are now going to present our original experiments. We would like to answer a few questions that have not been investigated yet by current multilingual text-to-speech research.

There are multiple limitations to today's multilingual TTS models. First, a lot of multilingual models use phoneme inputs. This poses a problem because it can be very hard to obtain accurate phonemicized variants of texts for less frequent languages, even though there are tools like Epitran [Mortensen et al., 2018]. This motivates us to perform all our experiments directly on grapheme inputs. Second, the current multilingual works are multilingual in a limited way – they usually pay attention to two or three languages and do not discuss the scalability of their approaches. Third and finally, some works use proprietary or other not properly specified data, so reproducibility is complicated.

These limitations motivate us to search for an answer to the following main questions in this chapter:

1. Could it be beneficial to use *multilingual* instead of monolingual data?
2. Are currently proposed multilingual models *scalable* to more languages?
3. Are they also able to perform code-switching with *grapheme* inputs?

The first question touches the principle of multilingual training in the context of speech synthesis. Intuitively, cross-lingual parameter and knowledge sharing should be advantageous while training models for low-resource languages. This can make training on openly available data viable. The second question tries to reveal the performance of multilingual models that concern more than three languages. Finally, the third question aims at code-switching or voice cloning without expensive input preprocessing.

In Section 6.1, we describe our implementation of three text-to-speech systems. The first is a partial reimplementation of the model proposed by Hsu et al. [2019, see Section 5.2], the second is a modification of the model by Cao et al. [2019, see Section 5.3], and the third is our own new model that tries to combine advantages of the two previous models. We call these models *Shared*, *Separate*, and *Generated*, respectively. In Section 6.2 we discuss the preparation of our cleaned versions of CSS10 and Common Voice datasets. Finally, in Sections 6.3 and 6.4, we present the evaluation of our models and provide an answer the above questions, at least in the context of our three models.

6.1 Implementation

We decided to base our multilingual experiments on the Tacotron 2 architecture with the WaveRNN vocoder (described in Sections 2.2.3 and 2.2.5, respectively). There were multiple reasons for this choice: Tacotron 2 provides state-of-the-art results for end-to-end spectrogram generation. It is easily extensible and it is used by numerous recent works (see Section 5).

There are many open-source implementations of Tacotron available [Mama, 2018, Valle, 2020], but we decided to build our own implementation from scratch. The main reason for this was that we experimented with various architectures and the reimplementation was easier to work with for us than using and modifying existing code. Moreover, we gained awareness about all essential things that make the training of Tacotron 2 possible. The source code is available online at the GitHub repository of this work.[1]

In our initial experiments, we used Tacotron for generating linear spectrograms and we used the Griffin-Lim algorithm (cf. Section 2.2.3) for vocoding. That is why our implementation has an option for generation of linear spectrograms. To produce more natural-sounding speech, we later employed a public implementation of WaveRNN which is released under the MIT license [Fatchord, 2019]. It does not include all the features described by Kalchbrenner et al. [2018], but it can generate audio in real time on a regular hardware. Training of WaveGlow [Prenger et al., 2019] was not possible in our conditions because of hardware limitations. We would like to note that we have not experimented with other approaches to spectrogram inversion such as MelGAN [Kumar et al., 2019].

In the following, we describe the implemented models one-by-one and explain our design choices made based on preliminary experiments on the development data.

6.1.1 Tacotron 2

Shen et al. [2018] clearly describe the architectures of their Tacotron 2 encoder, prediction layers, the pre-net, and the post-net (see Section 2.2.4), but they do not mention some implementation details. It is not clear how exactly the decoder and the attention mechanism should look like. They do not tell us anything about the weighting of loss functions, nor do they tell us how to compute the loss function of stop tokens. The authors also does not mention whether or how to normalize input spectrograms. We are now going to describe the architecture details we decided to use in our own implementation.

Encoder: According to Shen et al. [2018], the vanilla encoder of Tacotron 2 starts with 512-dimensional character embeddings which are passed into three one-dimensional convolutional layers with kernel size 5 and with 512 filters. All the convolutional layers are followed by batch normalization layers, ReLU activations, and a dropout layers with rate 0.5. Finally, the output of the last

[1] https://github.com/Tomiinek/Multilingual_Text_to_Speech

convolutional layer is passed into a bidirectional LSTM layer with 256 units in each direction.

Pre-net, post-net: The pre-net includes two fully connected layers of size 256, each of these layers is followed by a ReLU activation and a dropout layer with rate 0.5 that is enabled also during inference. The post-net consists of five convolutional layers with 512 filters and with a kernel of size 5 followed by batch normalization layers and tanh activations (except for the last layer).

Decoder: Since Shen et al. [2018] do not give any details on the decoder, open-source implementations vary at this point. Multiple options are plausible, for example, using an extra RNN layer that produces attention queries or using the predicted frame as the attention query. We considered two possibilities. The first one is probably closer to the description mentioned by Shen et al. [2018] (see Figure 6.1 left). It uses two stacked LSTMs and the output of the second one is used for the stop token prediction, the frame prediction, and also as the attention query in the next decoder step. The second one (Figure 6.1 right) follows the architecture of the first version of Tacotron (cf. Section 2.2.3), i.e., the output of the first RNN layer is used as the attention query and the second RNN layer is used as a generator of frames and stop tokens. We decided to take the second option as it seems, in our opinion, to be more justifiable. In the first architecture, the final output of the LSTMs is overloaded and used for three different things. In contrast, the second option leaves the second, "generator" LSTM more freedom.

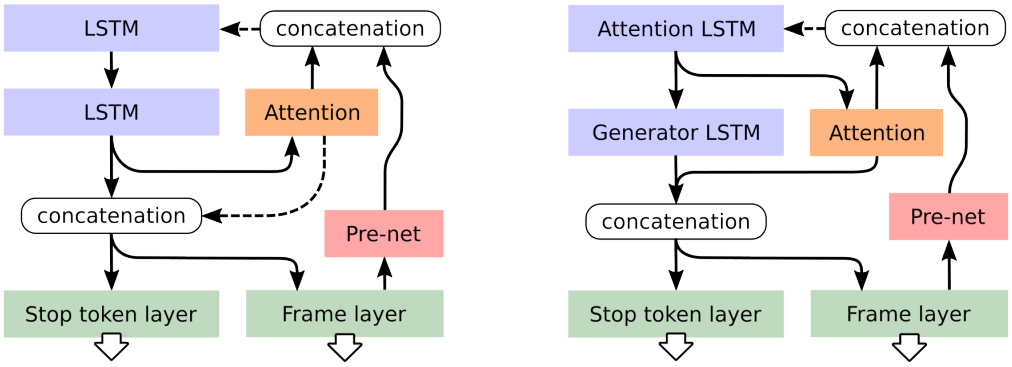

Figure 6.1: *Diagram showing two different architectures of Tacotron's decoder. Arrows depict information flow. Dashed lines represent dependency between two consecutive decoder steps.* Left: *An architecture that uses the final output of the second LSTM as the attention query.* Right: *Another Tacotron-1-style construction that uses the final output of the first LSTM as the attention query.*

Unlike Shen et al. [2018], we use a simple dropout on the decoder's LSTM outputs instead of the zoneout regularization. We experimented with both approaches and did not find any differences in the quality of resulting spectrograms. The dropout version is easier to implement and probably slightly faster.

We use full teacher forcing during training, i.e., at each decoder step, we feed the ground-truth frame from the previous step (passed through the pre-net) into the decoder. Any attempts to gradually use more predicted frames instead of ground-

truth ones were not successful. We experimented, for example, with linear and exponential decay of the teacher forcing ratio, but both cases lead to unstable training and problems with gradients.

Attention: Our implementation of the location-aware attention mechanism is straightforward and follows exactly the equations introduced in Section 2.2.4. While some public implementations of Tacotron (such as Valle [2020]) use attention weights computed in the previous decoder step as an additional feature, we do not use this extension. We did not implement any additional features such as *sharpening* and *smoothing* [Chorowski et al., 2015].[2]

As an alternative which could replace the location-sensitive attention, we additionally implemented the *forward attention* [Zhang et al., 2018]. This type of attention promotes monotonous alignments, i.e., once the attention mechanism focuses on a part of the input sequence, it most likely no longer allows to focus on any preceding part. It calculates the weight of the n-th sequence element at time t using a recursive formula: $\alpha^t(n) = (\alpha_n^{t-1} + \alpha_{n-1}^{t-1}) \cdot e_n^t$ where α and e follow the notation from the Glossary[3], i.e., α denotes attention weights and e denotes unnormalized energies returned by the attention layer. Based on preliminary experiments, we decided not to use the forward attention in our models. The forward attention provides faster convergence, but it may be too restrictive. We found out when using the original attention mechanism that some models attend to certain input tokens repeatedly once they reach punctuation marks, but this does not lead to word repetition on the output. With the guided attention loss (see Section 2.2.6), the location-sensitive attention can converge rapidly.

Loss functions and optimization: The authors of Tacotron 2 suggest optimizing the summed *mean squared error*[4] (MSE) from before and after the post-net to aid convergence. We experimented with optimization of the L1 loss function instead[5] (it was used in the first version of Tacotron), but we did not notice any substantial differences. During training, we mask predicted spectrograms to match the lengths of their targets.

We do the same with predictions of the stop token and we train the model to predict high stop probabilities for a few last frames. To be more specific, we use the cross-entropy loss function.[6] The target stop token class is negative for all frames except for $f = 5$ last, where it is positive. The loss function weights the positive class as $100\times$ more important to compensate for class imbalance. Because we use f last frames to decide on the end of synthesis, we stop the generation of new frames f frames after we encounter a positively classified stop token during inference.

We also incorporated the guided attention loss [Tachibana et al., 2017, see Sec-

[2]Sharpening uses a windowing mechanism so that the attention considers only a subsequence of the input sequence. Smoothing uses a different normalization of predicted attention weights.

[3]See "Attention mechanism" in the Glossary.

[4]See "MSE loss" in the Glossary.

[5]See "L1 loss" in the Glossary.

[6]See "Cross entropy loss" in the Glossary.

tion 2.2.6] into our implementation. We start with the g parameter (that controls the strictness of the diagonal) equal to 0.25. During training, we gradually increase the tolerance of the guided attention loss by increasing the g, so this loss term is only important at the beginning of training. We use an exponential schedule (we multiply g by a factor $\gamma = 1.00025$ every training step). This approach helps to establish the attention very well and we usually obtain a fully converged model after less than 50k training steps (which is the point where Shen et al. [2018] start with the exponential learning rate decay).

We found out that a proper weighting of loss terms is crucial for good performance and fast convergence. We divide the stop token loss and the guided attention loss by the number of mel bands of predicted spectrograms. Without this weighting, the stop token prediction was not very reliable and the model was too limited by the attention loss. We also weight the MSE before the post-net $2\times$ more than the MSE after the post-net.

For optimization of the loss function, we use the Adam optimizer with common settings ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-6}$, and weight decay of 10^{-6}) and with the initial learning rate 0.001.[7] A simple stepped learning rate decay worked better for us than the exponential schedule[8] that was suggested by Shen et al. [2018]. We thus halve the learning rate every few thousand steps (the exact number of steps varies for different models).

Proof-of-concept single-speaker model: As a proof of concept, we trained our Tacotron 2 implementation on the LJ Speech dataset [Ito, 2017, see Section 4.1]. See the GitHub repository of this work.[9] It contains a link to a few samples synthesized using the trained monolingual model. Note that the samples were vocoded by the Griffin-Lim algorithm, so the naturalness is not human-like.

Multi-speaker extensions: In order to use the architecture on our data, we finally needed to extend the Tacotron model to support multi-speaker or multilingual synthesis. A multi-speaker modification of the first version of Tacotron [Wang et al., 2017, see Section 2.2.3] was proposed by Gibiansky et al. [2017]. It transforms one-hot vectors corresponding to speaker identities using multiple site-specific embedding layers followed by soft-sign activations. The resulting speaker representations are then used for initialization of the initial hidden state of GRU layers (both, in the encoder and decoder).

In case of Tacotron 2 [Shen et al., 2018], the multi-speaker support is usually added by concatenating the same single speaker embedding with all encoder outputs (e.g., Jia et al. [2018]). We did not find any noticeable advantage in the speaker-dependent initialization of decoder's recurrent layers, so in order to make the model multi-speaker or multilingual, we simply use the concatenation. In our experiments, we choose speaker or language embedding size with respect to the number of voices or languages in the training data (i.e., the nearest power of two).

[7] See "Optimizer", "Weight decay", and "Learning rate" in the Glossary.
[8] See "Learning rate schedule" in the Glossary.
[9] https://github.com/Tomiinek/Multilingual_Text_to_Speech

6.1.2 Shared Encoder

We are now going to provide details about the *Shared model*. It follows the work by Zhang et al. [2019] and consists of Tacotron 2 (as described in Sections 6.1.1), an adversarial speaker classifier, and speaker or language embeddings. The original overall architecture of Zhang et al. [2019] was already described in Section 5.2.

A difference from Zhang et al. [2019] is that we do not include their residual autoencoder, which was originally used for stabilization of prosody. We implemented it but did not observe any performance improvements. Even without the residual autoencoder, we did not encounter any problems with very unnatural pauses as reported by the authors, and thus we conclude that the residual encoder is only needed when training on datasets with a high prosodic variation.

We implemented the adversarial speaker classifier as a neural network with a hidden fully connected layer of size 256 followed by another fully connected layer that predicts speaker identities. As described in Section 5.2, gradients from the classifier are reversed to enable voice cloning. We clip the reverted gradients to 0.25. Same as with the stop token (see Section 6.1.1), it was extremely important to properly weight the adversarial loss function (i.e., cross-entropy). We divide it by the number of predicted mel bands as in the case of the stop token loss. On top of that, we use an additional weighting parameter w. Zhang et al. [2019] use weights from 0.02 to 0.1 and state that it can be used to control accent level (lower values cause stronger accent and higher values correspond to lighter foreign accent). However, they used phoneme inputs and our approach – using the adversarial classifier together with grapheme inputs – seems to be more complicated. Higher values prevent the model from convergence and lower values result in poor voice-cloning abilities. We empirically found out that w in the range from 0.1 to 2.0 works well on our data.

We also experimented with an alternative approach to eliminating unwanted information via domain-adversarial training. The gradient reversal may lead to a perfect fooling of the speaker classifier, resulting in a case where the classifier has 0% accuracy (so the information is actually not removed). Heo et al. [2018] use the cosine similarity loss[10] over classifier weights and encoder outputs to make these two vector spaces orthogonal. They show that this kind of domain-adversarial process can reduce channel information on the speaker identification task and that it should not lead to 0% classifier accuracy. We implemented this method, but unfortunately the models did not converge on our data.

6.1.3 Separate Encoders

The second model that we would like to examine is called *Separate*, and follows Cao et al. [2019, see Section 5.3]. It builds on Tacotron 2 (Section 6.1.1), but it uses multiple distinct language-specific encoders. The decoder is conditioned on concatenations of encoder outputs and speaker embeddings. It enables code-switching by combining outputs of different encoders. The architecture was originally used just for two languages, but we would like to make it usable for more

[10] See "Cosine similarity loss" in the Glossary.

of them. Therefore, we adapted Cao et al. [2019]'s model by using convolutional encoders instead of recurrent.

Having a separate encoder for each language that is in our training data could be intractable. For example, the CSS10 dataset has 10 languages, so we would need 10 encoders. Since language-balanced batches are desired to promote learning all languages at an equal pace (see below), during training, we would have to pass the same input batch through all the encoders, which can be very slow. This problem can be overcome by parallelization of the encoders. Thus we replace original encoders that contain recurrent layers by fully convolutional encoders used in the DCTTS model [Tachibana et al., 2017, see Section 2.2.6]. This change allows us to implement all the encoders using grouped layers, and thus all encoder-related computations can be done effectively in a single pass.

We now provide a detailed description of the architecture and parameters of the encoders. They consist of two types of building blocks.

$C_{i,o}^{k,d}$ – This module consists of a one-dimensional convolutional layer, a batch normalization layer, and a dropout layer. The batch normalization can be followed by an activation function. For convenience, we will denote this block as $C_{i,o}^{k,d}$ where k is the size of the kernel of the convolutional layer, d stands for its dilation[11] and i or o denote input and output dimensions, respectively. We will omit o in the notation if $o = i$.

$H_{i,o}^{k,d}$ – The second module is very similar to the first one, but it has got a one-dimensional *highway convolutional layer* instead of a simple convolutional one.[12] The input to this layer is passed through two equal-sized convolutional layers. The output of the first one is followed by a sigmoid function that produces an output g which is used in a gating mechanism. The gating mechanism combines the input i of the module and the output o of the second convolutional layer as follows: $x = g \cdot i + (1 - g) \cdot o$, where x denotes the output of the convolutional highway layer. Same as for $C_{i,o}^{k,d}$, batch normalization and dropout follow. We will denote this module $H_{i,o}^{k,d}$ where all indices have the same meaning as for $C_{i,o}^{k,d}$.

We can now easily describe the architecture of the our convolutional encoder using the above notation: $C_{512,256}^{1,1}$ with ReLU $\to$ $C_{256}^{1,1} \to H_{256}^{3,1} \to H_{256}^{3,3} \to H_{256}^{3,9} \to H_{256}^{3,27} \to H_{256}^{3,1} \to H_{256}^{3,3} \to H_{256}^{3,9} \to H_{256}^{3,27} \to H_{256}^{3,1} \to H_{256}^{3,1} \to H_{256}^{1,1} \to H_{256}^{1,1}$.

In comparison with Tachibana et al. [2017]'s DCTTS encoder, we enhanced all the modules with batch normalization layers and with dropout layers with a dropout rate of 0.05. However, this regularization implies a slightly slower convergence. We also experimented with group normalization, but it did not work well.[13]

We implement all the convolutional layers and batch normalization layers using their grouped variants (they are directly supported by modern deep learning

[11] See "Convolutional network" in the Glossary

[12] Cf. "Highway network" in the Glossary.

[13] See "Group normalization" in the Glossary

frameworks such as Pytorch [Paszke et al., 2019]). However, to utilize the potential of this architecture during training, we need to construct training batches extraordinarily. We would like to have a batch of B examples that can be reshaped into a batch of size B/L where L is the number of encoder groups or languages. This new batch should have a new dimension that groups all examples with the same language, because then, we can process them in a single encoder pass. That is why we use a batch sampler that creates batches where for each $l < L$ and $i < B/L$, all $(l + iL)$-th examples in the batch (indexed from 0) are of the same language.

This per-language batch splitting results in a problem with an imbalanced batch size of the decoder and encoders. Encoders are trained with reshaped batches of size B/L, but the decoder uses the original batch of size B. So the effective size of the encoder's batches can be small and that is why they may require low learning rate values for stable training. However, the decoder with larger batch sizes requires a higher learning rate to properly converge. In our experiments, we attempted to solve this discrepancy by using two optimizers with different learning rates, but the model did not converge very well. We had problems with overfitting of the decoder, which resulted in a poor attention. In the end, a lower learning rate used for all parameters in the network works best. In our experiments, we use a lower initial learning rate values such as 10^{-4}.

6.1.4 Generated Encoder

At the first sight, the main advantage of the Shared model (Section 6.1.2) is that it shares encoder parameters between all languages. The sharing implicitly leads to a language-independent encoder and this effect is even more emphasized by the adversarial speaker classifier. All language-dependent processing happens in the decoder, and that is why the decoder needs the discriminative embedding explicitly factorized into language and speaker parts. However, the parameter sharing might be also a disadvantage because there are languages with very diverse pronunciation rules. For example, German has a nearly one-to-one correspondence between graphemes and phonemes, i.e., a sequence of graphemes corresponds to a sequence of phonemes with a similar length (e.g., fabelhaft results in [faːbəlhaft]), but French usually pronounces multiple graphemes as a single phoneme (e.g., merveilleux results in [mɛʀvɛjø]). These huge differences in pronunciation might cause problems, for example, with stability of the attention mechanism.

On the other hand, the Separate model (Section 6.1.3) is more flexible, and it can learn distinct pronunciation rules for each language. But having a separate encoder for each language may lead to worse voice cloning abilities as speaker-specific information (which correlates with language) can leak into language-specific encoders more easily. Obvious disadvantages of this approach are limited scalability and missing cross-lingual knowledge sharing.

These thoughts brought us to the *Generated model*. It combines parameter sharing and language-specific convolutional encoders. It is based on the idea of *contextual parameter generation* that was proposed by Platanios et al. [2018] for

neural machine translation and that was also successfully applied, for example, to the music separation task [Samuel et al., 2020]. This method makes use of multiple site-specific generator networks that produce parameters for layers of the backbone model (the convolutional encoder in our case) from an input language embedding. All the generator networks are implemented as fully connected layers. They can be viewed as bottlenecks that select features of the input language embedding that are relevant for the particular generated layer. That is why they should be smaller than the input language embedding.

This approach allows controllable parameter sharing (by changing the dimensionality of generator networks). However, the model can still decide about language similarities by itself. The model also does not suffer from the problem with imbalanced training like the Separate model and as we found out in preliminary experiments, it can be combined with the adversarial speaker classifier to further support voice-cloning abilities. Moreover, this model can control accent by a simple mixing of outputs of different encoders. To sum it up, the Generated model is the Tacotron 2 model (Section 6.1.1) with an adversarial classifier from the Shared model (Section 6.1.2) and with a convolutional encoder with generated parameters (see Figure 6.2). The decoder is conditioned on concatenations of encoder outputs and speaker embeddings.

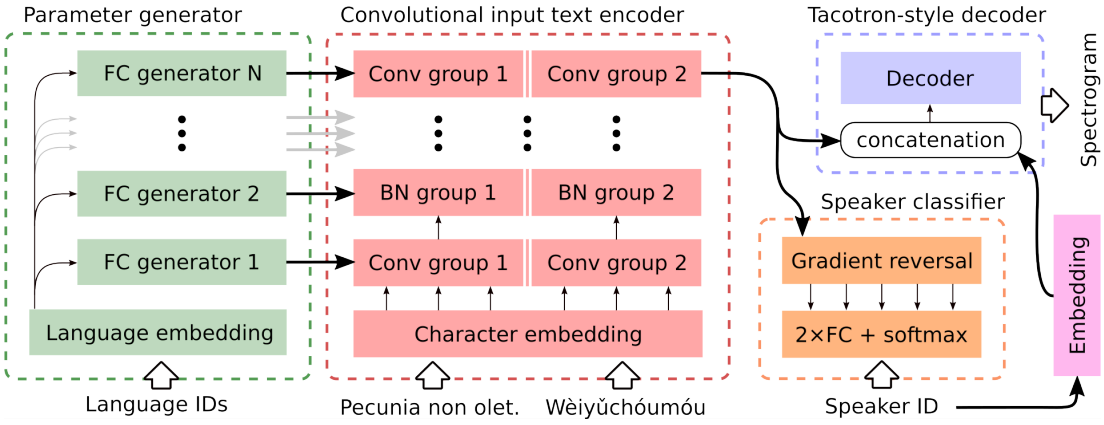

Figure 6.2: *Illustration of the Generated model. It includes a parameter generator network (green), a grouped convolutional encoder (red, each group has language-dependent parameters generated by the green module), the adversarial speaker classifier (orange), and finally the decoder (blue) that accepts a concatenation of encoder outputs and speaker embeddings (pink). The adversarial classifier is only used during training.*

During training, we use the same batch sampling method as in the case of the Separate model (Section 6.1.3). For each training batch, we generate parameters for all language-specific convolutional encoders. In our experiments, we usually set the size of the language embedding to double the number of languages. We choose the size of bottlenecks to be lower than the number of languages. While setting this number, we have to keep in mind our hardware capabilities because there is a linear dependence between this number and the number of encoder parameters. Note that a monolingual convolutional encoder has roughly 5× less parameters than the Tacotron 2 decoder.

6.1.5 WaveRNN Vocoder

As we said at the beginning of this section, we trained the WaveRNN model [Kalchbrenner et al., 2018, see Section 2.2.5] for the purpose of vocoding. We used an open-source implementation (the source code used for training is available online at GitHub).[14] The model consists of three modules. The first includes a stack of convolutional blocks with residual connections, the second is an upsample network that consists of two-dimensional convolutional layers (with just one filter). Finally, the third module modifies the output of the upsample network and produces the output waveform. It contains two GRU layers that are preceded and followed by a few fully connected layers. The output of the first module is split along the last dimension and parts of this split are fed into multiple different layers of the third module. See Figure 6.3 for a detailed description of the overall architecture. The last fully connected layer is followed by a softmax layer that produces a probability distribution over the current position of the audio wave. The distribution is sampled during inference to obtain the next audio sample.

We set the size of the two GRU layers and all fully connected layers to 512 (except for the last one that predicts outputs), we use ten convolutional blocks with 128 filters. The network generates μ-law quantized 10-bit audio sampled at 22,050 Hz. During training, we use mini-batches of size 64 balanced with respect to speaker identities. The network is optimized by the Adam optimizer [Kingma and Ba, 2015] with weight decay. The learning rate is initially set to 10^{-3} and is halved every 100k training steps.

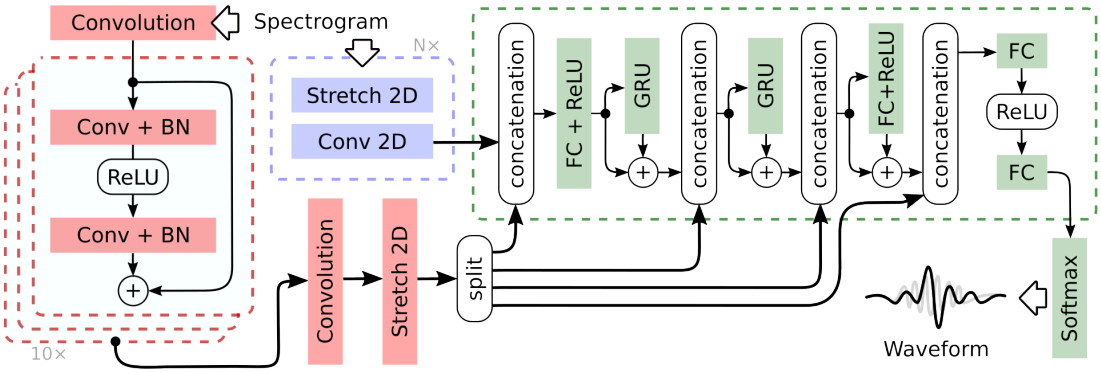

Figure 6.3: *Illustration of Fatchord [2019]'s implementation of the WaveRNN model. Red color groups a network made of stacked convolutional residual blocks. BN stands for batch normalization. Its output is at the end stretched to match the sampling rate of the target and split into four branches. Blue color groups an upsample network that gradually stretches the input spectrogram. The key part of the network is in green color. It consists of a few GRUs and fully connected layers (FC) with ReLU activations. Layers are interconnected with residual connections. In the end, an output distribution over audio bits is predicted.*

In our experiments, we trained the network for 725k training steps on ground-truth-aligned spectrograms (GTA). To obtain the GTA spectrograms, we trained a monolingual Tacotron 2 model (Section 6.1.1) for each language in our training

[14] https://github.com/Tomiinek/WaveRNN, which is based on Fatchord [2019].

data, and we synthesized new spectrograms in a supervised mode with enabled teacher forcing. The Tacotrons used for this purpose do not have to be even able to produce speech from text because we are only interested in their decoders.

Fatchord [2019]'s implementation can optionally do a batched inference that is quite fast on a GPU. However, CPU inference takes a lot of time. To enable real-time CPU inference, we experimented with weight pruning as described by Kalchbrenner et al. [2018], but the quality of outputs degraded, so we do not use it in the experiments described later in this chapter.

6.2 Data Preparation

This section is devoted to the description of data that we used for the training of our models. We base our dataset on CSS10 and Common Voice (see Chapter 4).

6.2.1 CSS10

We already discussed some imperfections of CSS10 in Section 4.2. We showed distributions of duration and length of CSS10 examples, and we compared the distributions to the statistics of the LJ Speech dataset (compare Figure 4.1 with Figure 4.2). Note that LJ Speech is a very popular dataset, partly because it is clean and suitable for TTS training. That is why we would like to accommodate CSS10 and make it more similar to LJ Speech.

Textual normalization: First, we decided to normalize all textual utterances across all languages. We computed frequencies of symbols in the dataset and we replaced some less frequent characters by their more frequent alternatives. For example, we replaced French œ with o and e, question marks of Eastern languages "？ " with European question marks "?", we unified quotation marks by replacement of Chinese 「」 , French «» or Hungarian „" by English "", etc. We also removed any examples with non-standard characters and numbers, and we obtained the following reduced character set in the end (for brevity, we do not show capital letters):[15]

> abcdefghijklmnopqrstuvwxyzçèéßäöōăïíõàáǔüèéìūòóùúāēěīâêôûñőű
> абвгдежзийклмнопрстуфхцчшщъыьэюяё
> άέήίαβγδεζηθιϰλμνξοπρςίστυφχψωόύώ
> ()¿?¡!、。, ,.:;-'"

CSS10 was created from different audiobooks and contains various punctuation styles. To tackle this issue, we removed punctuation from the beginning of transcripts, we removed spaces before full stops, exclamation marks, etc. We also merged multiple sentence endings, e.g., "?!?" was replaced by "?".

Very common inconsistency is caused by different usage of dashes. Some transcripts contain double hyphens to denote en dashes or em dashes, some do not

[15]Note that Chinese and Japanese were romanized (see below).

contain space before or after them, etc. At first, we replaced en dashes and em dashes (and Chinese dashes, horizontal bars, etc.) with double hyphens, then we replaced pairs of hyphens together with surrounding whitespaces by a single hyphen with a space before and after it. Secondly, we made sure that these characters are not preceded or succeeded by other punctuation marks (so, for example, we normalized "Yes! - Said Bob." into "Yes! Said Bob."). The cleaning was done using regular expressions. However, even the cleaned text might contain a small amount of odd punctuation combinations.

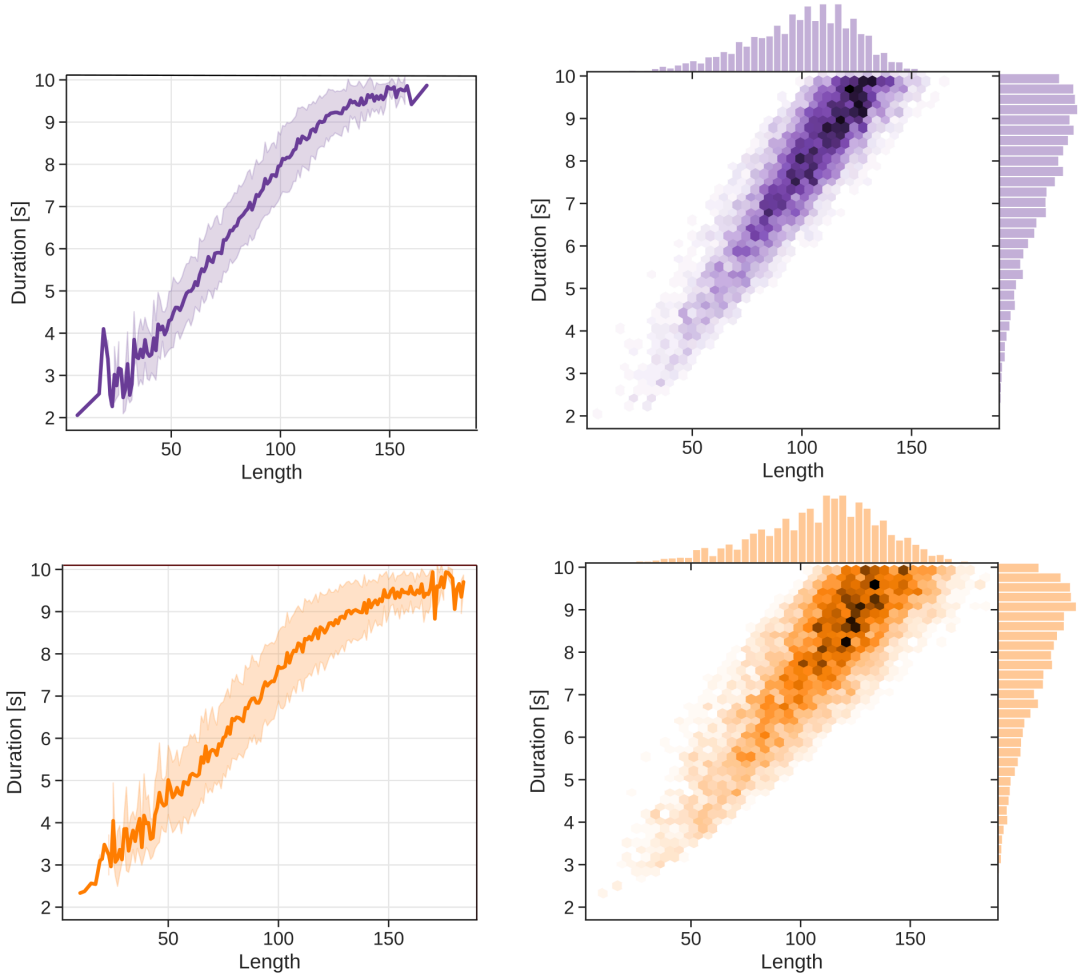

Figure 6.4: *Graphs describing dependencies between text lengths and audio durations of cleaned German (blue) and French (orange) data in the cleaned CSS10 dataset. Compare this figure with Figure 4.1 and Figure 4.2.* Left: *Dependency between lengths of transcripts and durations of their corresponding recordings. The solid line depicts mean duration and lighter colors visualize variance.* Right: *Distributions of transcript lengths (the upper histogram) and audio durations (rightmost histograms) together with a visualization of their joint distribution. Darker and lighter colors correspond to higher and lower relative frequencies, respectively.*

We needed to re-romanize Chinese characters in the CSS10 dataset to make the romanization consistent with Common Voice and our evaluation sets (see Sections 6.2.2, 6.3, and 6.4). Unfortunatelly, we were not able to use the original pipeline applied by CSS10 authors (Jieba segmenter [Sun, 2018] and CC-CEDICT

dictionary [Denisowski, 2020]), so we instead used a python package called pinyin [Yu, 2016]. The package uses a different segmenter or dictionaries and produces slightly different transcripts. That is why our romanized forms of CSS10 do not match the original variants. We did not have the same problem with Japanese because we were able to use the same tools as the authors of CSS10, i.e., the MeCab [Kudo, 2015] for segmentation and conversion into katakana and a python version of Romkan [Yao, 2013] for the romanization itself.

Audio data filtering: Recordings with durations longer than 10 seconds are not very common and those corresponding to transcripts with lengths shorter than 25 characters have a high variance of durations. In comparison with LJ Speech data, utterances with the same number of characters have generally higher variance of durations.

Therefore, we decided to remove a few examples from CSS10. We preserved only examples with audio durations between 0.5 seconds and 10.1 seconds, we excluded examples with transcripts longer than 190 characters and shorter than 3 characters, and we also removed examples with too deviated duration. More specifically, we first took audio durations and computed means μ and variances σ of all groups made of examples with the same transcript lengths. Then we removed examples with durations outside the interval $(\mu - 3\sigma, \mu + 3\sigma)$.

Resulting data statistics: See Figure 6.4 and Table 6.1 for the statistics of the resulting cleaned dataset. In comparison with original data (see Figure 4.2 and Table 4.1), the cleaned version has more similar duration and length distributions across languages. It has a slightly lower mean transcript length and a noticeably lower variance of audio durations (around 1.6 for all languages). It contains only 125.26 hours of recordings in total, i.e., we removed roughly 15 hours of data.

6.2.2 Common Voice

The CSS10 dataset provides a decent quantity of data. However, we realized that it is hard to train a model capable of voice-cloning and code-switching using a dataset that contains data from just a single speaker for each language. Language and speaker-specific information are too entangled and the situation becomes even worse when the languages use different scripts and when speakers have very different voices (which is the case of the CSS10 dataset). We enhanced CSS10 data with data from Common Voice and build a multi-speaker dataset that makes it possible to create multilingual systems with voice-cloning abilities.

The Common Voice dataset does not contain all languages that are included in CSS10. The intersection of common languages of those datasets covers seven languages, see Table 6.2 for details.

The vast majority of Common Voice recordings have very bad quality. Thus we filtered out recordings with a negative rating. We excluded speakers with less than 50 recordings. Then we went through a few samples from each speaker and we removed all data from the particular speaker if the samples had poor quality.

Table 6.1: *Summary of properties of cleaned CSS10 data grouped by language. The second column contains total durations of original uncleaned CSS10. The third column provides information about total lengths of all recordings. The fourth column shows mean durations of examples with corresponding standard deviations. The last column groups means and standard deviations of transcript lengths.*

Language	**Recordings Dur. Orig. Total [h]**	**Recordings Dur. Total [h]**	**Recordings Dur. Mean [s]**	**Transcripts Mean Length**
German	16.13	15.39	7.75 ± 1.64	99.7 ± 24.3
Greek	4.14	3.53	7.74 ± 1.66	115.7 ± 27.3
Spanish	23.83	20.85	7.44 ± 1.79	109.5 ± 30.7
Finnish	10.53	9.67	7.74 ± 1.58	94.7 ± 23.9
French	19.15	16.89	7.83 ± 1.59	108.9 ± 27.9
Hungarian	10.01	9.52	7.95 ± 1.49	110.0 ± 26.0
Japanese	14.92	14.33	7.88 ± 1.38	118.7 ± 23.8
Dutch	14.11	11.73	7.38 ± 1.79	114.7 ± 32.0
Russian	21.37	17.71	7.90 ± 1.48	90.8 ± 24.8
Chinese	6.45	5.64	7.62 ± 1.68	101.7 ± 27.3

Table 6.2: *Summary of the total audio duration and the number of speakers of the original Common Voice data in languages that are also included in CSS10.*

Language	**Recordings Total Dur. [h]**	**Number of speakers**
German	468	8460
Spanish	167	8252
French	350	8146
Japanese	3	52
Dutch	24	701
Russian	72	496
Chinese	26	963

We found out that recordings with the same speaker identification numbers do not have to be recorded in the same conditions and by the same speaker (it seems that the dataset contains a lot of data recorded in workshops which results in multi-speaker data labeled with the same IDs). We attempted at splitting these data into subsets with roughly equal conditions and speakers, and then we removed subsets with bad quality and with less than 50 recordings again.

We end up with a relatively small dataset. It contains 39 German speakers, 22 French speakers, 11 Dutch speakers, 6 Chinese speakers, and 6 Russian speakers. Japanese and Spanish data were removed completely. Transcripts were mostly consistent so we did not have to do an extensive cleaning like in the case of CSS10. We used the pinyin package for the romanization of Chinese.

On the other hand, recordings required much more processing. Many of them contained artifacts at the beginnings and ends (possible causes were described in

Chapter 4). To solve this issue, we conservatively removed (low volume thresholds in short windows) trailing and leading silence using Sound eXchange (SoX) open source audio editing software [Barras, 2012]. Afterward, we inspected a few recordings for each speaker and if we noticed that the automatic processing did not resolve the problems, we manually performed a cleaning of all recordings of the particular speaker. Table 6.3 summarizes the characteristics of our cleaned data. It has 13.71 hours of audio data in total. In comparison with the cleaned CSS10, it has approximately two or three times shorter recordings.

Table 6.3: *Summary of properties of cleaned Common Voice data grouped by language. The first column shows the total duration of all recordings of a particular language, the second column contains mean durations with the variance of contained recordings, and the third describes mean length of transcripts.*

Language	**Recordings Duration Total [h]**	**Recordings Duration Mean [s]**	**Transcripts Mean Length**
German	4.83	2.96 ± 1.20	50.2 ± 19.5
French	2.96	2.81 ± 1.27	48.5 ± 21.8
Dutch	1.30	2.79 ± 1.13	48.3 ± 18.9
Russian	3.37	3.70 ± 1.60	61.9 ± 28.3
Chinese	0.95	5.12 ± 2.08	73.8 ± 29.8

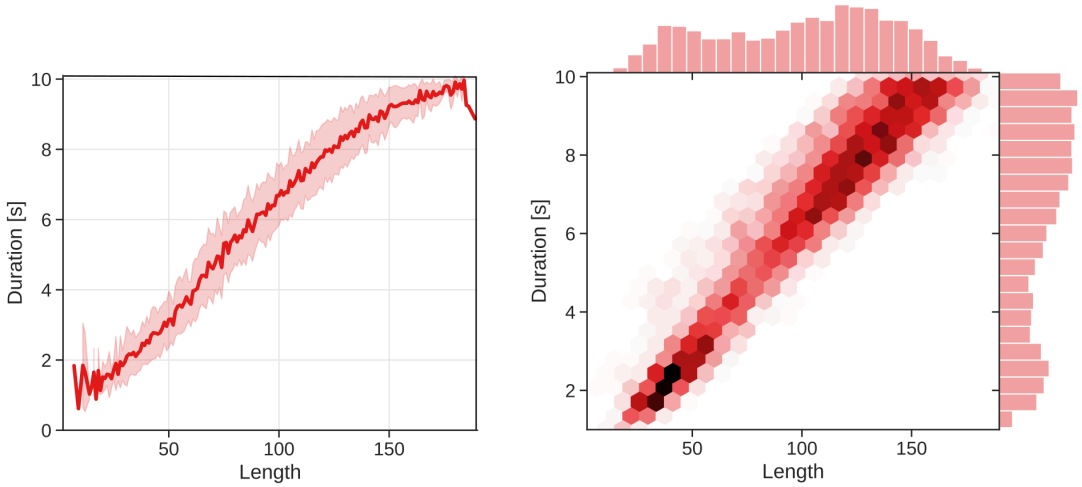

Figure 6.5: *Graphs describing dependencies between transcript lengths and audio durations of cleaned Dutch data from the union of Common Voice and CSS10 datasets.* Left: *Dependency between transcript lengths and recording durations, the solid line stands for the mean duration and the light color visualizes variance.* Right: *Illustration of length and duration distributions (top and right histograms, respectively) together with a visualization of their joint distribution. Darker and lighter colors correspond to higher and lower relative frequencies, respectively.*

We use this dataset together with the cleaned CSS10 data for the purpose of training of our models capable of voice cloning and code-switching. The cleaned CSS10 dataset has rather longer recordings and transcripts (see Figure 6.4) and

the cleaned Common Voice dataset has shorter recordings, so the durations of their recordings complement each other in this regard. Figure 6.5 visualizes distributions of Dutch data of the union of both datasets. The linear dependency between transcript lengths and audio durations is preserved. There are two noticeable peaks in the distributions. They correspond to the most frequent durations or lengths in CSS10 and Common Voice.

We follow Common Voice and release the cleaned dataset under the Creative Common CC0 license.[16] Cleaned transcripts and training or evaluation splits of all datasets (cleaned CSS10, cleaned Common Voice, and their combination) are available at the GitHub repository of this work.[17]

6.3 Multilingual Training

In this section, we would like to answer the first question that we asked at the beginning of this chapter, i.e., whether the usage of multilingual data for training of TTS systems can be beneficial. We are also going to touch the second question and discuss the scalability of compared models. See the GitHub repository of this work that contains links to synthesized samples and interactive demos.[18]

6.3.1 Experiment Setup

At first, we trained the four models described in Section 6.1 on a subset of the cleaned CSS10 dataset (Section 6.2), without an ambition to do voice cloning or code-switching. We trained the vanilla Tacotron 2 model for each language of the CSS10 separately. We refer to the set of these monolingual models as *Single* further in this text. The Single models were trained on particular language-specific subsets of the cleaned CSS10. We switched off the adversarial speaker classifier in the case of Shared and Generated models, i.e., the Shared model was reduced to the vanilla Tacotron 2 with a discriminative language embedding that is concatenated with encoder outputs. We used the same vocoder for all models, i.e., the WaveRNN model trained on the same subset of CSS10 with GTA spectrograms generated using the Single models (see Section 6.1.5).

Training setup: We trained the three models for 50k steps and we used the same hyperparameters as described previously in Section 6.1 except for the learning rate.[19] We decayed it every 10k training steps and in the case of the Separate model, we used an initial learning rate of 10^{-4} both for the encoder and the decoder. The number of training steps and the learning rate schedule were tuned individually in the case of Single models because the size of training data varies across languages. For example, we decayed every 2.5k steps in the case of the German model and we trained it just for 12.5k steps. Generally, we stopped

[16] https://www.dropbox.com/s/axoic9eoeii1zyd/clean_comvoi.tar.gz

[17] https://github.com/Tomiinek/Multilingual_Text_to_Speech/tree/master/data

[18] https://github.com/Tomiinek/Multilingual_Text_to_Speech

[19] See https://github.com/Tomiinek/Multilingual_Text_to_Speech/tree/master/params for all parameter configurations used in the experiments described here.

training when we noticed a stable and striking growth of the loss on validation data (64 random examples per language). For all models, we used mini-batches of size 60 which were balanced with respect to languages.

Evaluation: To evaluate all the trained models, we used two held-out test datasets. The first set is a random part of the cleaned CSS10 dataset. It contains 64 samples for each of the ten languages. The second set consists of sentences scraped from Wikipedia. We created this test set by random browsing of Wikipedia pages (at a particular language version) and by taking a single sentence for each visited page. We skipped pages that were most likely generated by bots or better said using templates. We selected sentences that do not contain foreign words and that do not need further normalization (so we did not pick sentences with dates, etc.). In total, the set contains 50 sentences for each language. These two evaluation datasets are available at the GitHub page of this work.[20]

We synthesized all evaluation data using all the models and computed the character error rate (CER) and in the case of in-domain data also the mel cepstral distortion (MCD) as defined in Chapter 3. We send the synthesized audios to Google Cloud Platform ASR[21] and acquired transcripts. Then we computed character error rate between the ASR transcripts and the original ones. In the case of Chinese and Japanese, we computed CERs using native characters.

Data-stress training experiment: We also wanted to test all the models in more data-stress situations, so we randomly chose 900 examples for each language from the training set. The amount of data roughly corresponds to 2 hours of recordings per language. We tried to go even further and we selected a random subset of 600 examples from the reduced dataset. We repeated the training and evaluation of all models for both new reduced datasets. In this setting, we accomplished training just for the Shared and Generated models and none of the models could be trained successfully with just 300 training examples per language. For both models, we decayed the learning rate every 7.5k and 5k training steps in the case of the bigger and the smaller dataset, respectively.

6.3.2 Discussion of Results

We first discuss obtained CERs and MCDs of models trained on the full training dataset. We support our findings by manual inspection. We also inspect the language embeddings learned by the Generated model. Finally, we summarize the CER and MCD results in the data-stress training and draw general conclusions from this experiment.

CER results in the default setting: The CER results are presented in Table 6.4. The second column contains error rates of the ground-truth recordings. This gives us a notion about the performance of the ASR engine. Japanese has

[20] https://github.com/Tomiinek/Multilingual_Text_to_Speech/tree/master/evaluation
[21] Cloud Speech-to-Text, https://cloud.google.com/speech-to-text, accessed May 24, 2020

high CERs in comparison with other languages. This may be an effect of the Japanese script, where one character represents a longer segment than in an alphabet. On the other hand, the best performance on Spanish is not surprising because the Google Platform ASR offers engines even for multiple Spanish accents. Note also the high variance of Russian CERs and worse results of Dutch and Chinese.

Table 6.4: *Summary of the CER evaluation of the Shared, Separate, Generated, and monolingual ("Single" column) models trained using the cleaned CSS10 dataset. The second column contains CERs obtained from original human recordings. The values in the table are sentence-level CER averages with their corresponding standard deviations. The top and bottom lines correspond to the evaluation on in-domain and out-of-domain data, respectively. Bold texts emphasize the model with the lowest average CER and "*" marks a statistically significant difference against the model with the second-lowest CER according to a one-sided paired t-test with 95% confidence level.*

Language	**Ground Truth**	**Single**	**Shared**	**Separate**	**Generated**
German	4.8 ± 4.6	7.3 ± 6.0	8.3 ± 6.0	15.3 ± 6.0	***5.8±5.3**
	N/A	6.2 ± 4.7	5.0 ± 5.6	8.0 ± 6.3	**3.8±4.0**
Greek	8.7 ± 6.9	*N/A*	**11.4±8.3**	22.2 ± 8.3	11.6 ± 7.1
	N/A	*N/A*	**6.9±8.0**	14.2 ± 12.5	8.1 ± 9.7
Spanish	3.9 ± 4.6	7.0 ± 10.8	7.2 ± 6.5	10.2 ± 8.1	**7.0±9.8**
	N/A	**2.8 ± 3.2**	3.7 ± 3.7	5.6 ± 5.1	3.0 ± 3.7
Finnish	6.9 ± 10.4	18.6 ± 12.6	**10.3±8.0**	18.1 ± 11.4	10.4 ± 7.0
	N/A	13.7 ± 11.5	9.1 ± 7.2	12.5 ± 10.7	***6.4±6.1**
French	11.2 ± 7.3	25.2 ± 12.6	30.0 ± 14.3	54.5 ± 21.9	***19.0±12.9**
	N/A	22.1 ± 11.6	24.3 ± 12.0	51.2 ± 23.3	***15.8±8.5**
Hungarian	6.3 ± 6.1	15.8 ± 9.5	15.9 ± 10.6	18.8 ± 9.9	***13.5±8.3**
	N/A	9.60 ± 7.9	8.1 ± 7.3	11.4 ± 8.6	**7.7±6.7**
Japanese	19.0 ± 9.3	28.8 ± 11.3	27.2 ± 11.8	33.7 ± 13.5	**25.1±12.2**
	N/A	31.5 ± 19.8	24.2 ± 22.5	36.4 ± 21.1	**23.6±21.3**
Dutch	14.5 ± 7.4	33.4 ± 13.8	31.6 ± 12.5	49.0 ± 17.4	***22.6±9.6**
	N/A	24.9 ± 8.4	19.6 ± 10.6	48.7 ± 5.2	***16.4±9.6**
Russian	12.3 ± 15.0	45.5 ± 24.1	44.4 ± 21.9	58.1 ± 24.7	***34.5±21.3**
	N/A	34.4 ± 21.7	23.3 ± 16.9	45.3 ± 20.4	**19.4±16.7**
Chinese	14.6 ± 11.8	62.8 ± 18.5	28.6 ± 15.9	27.3 ± 14.8	***20.5±13.6**
	N/A	74.9 ± 15.6	41.2 ± 19.1	42.1 ± 19.9	**36.1±19.7**

We were not able to train the Greek monolingual model, thus the "Single" column does not contain any CER values for this language. The decoder started to overfit soon before the attention could have been established. The CER performance of Single models seems to be dependent on the amount of available training

data. For example, Chinese with approximately 5.6 hours has a very high error rate in contrast to Spanish with 20.8 hours of training data. However, even a higher amount of data does not necessarily ensure a successful model, Dutch has 11.7 hours of data (more than Hungarian), but the CER is higher.

We had problems with the training of the Separate model. Most CERs of this model are not competitive with the results of other models. This is probably caused by the imbalance between the batch size of the encoder and the decoder. However, our attempts to compensate for this using different learning rates were not successful. Sharing of the data probably regularized the decoder, so the attention was established even in the case of Greek. Also, the model has a much better performance in comparison with the monolingual model on Chinese.

Either the Shared or the Generated model have a better CER than the corresponding monolingual model (with 95% confidence according to a t-test and in the case of both out-of-domain and in-domain data) except for Spanish, German, and Hungarian.[22] This suggests that multilingual training is really beneficial.

The Generated model is significantly better than the Shared model on Dutch and French. It fulfills our expectations, as the Generated model should be more flexible. We can see a lower mean CER of the Generated model for almost all languages for both test datasets (except for Greek and Finnish), but without a statistical significance (or statistically significant on only one of the datasets).

Table 6.5: *Summary of the MCD evaluation of Shared, Separate, Generated, and monolingual ("Single" column) models trained using the cleaned CSS10 dataset with ten languages (the first column). The values in the table are per-sentence MCD averages with their corresponding standard deviations. Bold texts highlight the model with the lowest average MCD and "*" marks a statistically significant difference against the model with the second-lowest MCD according to a one-sided paired t-test with 95% confidence level.*

Language	Single	Shared	Separate	Generated
German	***3.85±1.03**	4.54 ± 1.03	4.48 ± 1.01	4.42 ± 1.15
Greek	*N/A*	3.73 ± 0.63	**3.45±0.63**	3.68 ± 0.51
Spanish	3.99 ± 0.72	4.08 ± 0.72	4.38 ± 0.48	***3.77 ± 0.75**
Finnish	3.60 ± 0.83	3.64 ± 0.67	3.91 ± 0.80	**3.50 ± 0.79**
French	4.28 ± 0.51	4.39 ± 0.53	5.20 ± 0.76	***4.07 ± 0.54**
Hungarian	3.49 ± 0.61	3.37 ± 0.65	3.45 ± 0.47	***3.17 ± 0.47**
Japanese	3.21 ± 0.36	3.32 ± 0.33	3.42 ± 0.38	**3.19 ± 0.40**
Dutch	4.31 ± 0.43	4.42 ± 0.62	4.69 ± 0.63	***3.99 ± 0.37**
Russian	7.15 ± 1.75	7.07 ± 0.43	7.25 ± 1.94	**6.75 ± 1.71**
Chinese	4.11 ± 0.56	3.71 ± 0.73	3.44 ± 0.41	***3.22 ± 0.55**

MCD evaluation in the default setting: Table 6.5 shows the average MCD of all models. The MCD is not as descriptive and interpretable as the CER, but

[22]This does not correspond to the "*" markings in Table 6.4, where the comparisons are made against the second-best model, not the monolingual model.

we can notice that the Generated model has significantly lower values for five languages; the difference is most profound for Dutch and French. This supports our conclusion from CER results that the Generated model performs better.

Manual error analysis: We manually inspected and compared the outputs on German, French, and partially Russian and Spanish. In the case of Spanish, all the models (except for the Separate) work well and we noticed just differences in the treatment of punctuation. On German, outputs produced by the Generated model are clearly the best. Other models sometimes make unnatural pauses when reaching a punctuation mark. Right after the pauses, they sometimes skip parts of the sentences. French and Russian are noticeably better in the case of the Generated model. Other models obviously mispronounce some words.

Language embeddings learned by the Generated model: We now attempt to analyze the language embeddings learned by the Generated model. See Figure 6.6 that describes similarities between them. It is hard to interpret it (and it is a question if it is meaningful), for example, the similarity between Greek and Japanese or Chinese and Finnish is quite odd. On the other hand, we can understand the higher similarity between Dutch and German as these two languages have high lexical similarity (meaning that the words come from the same origin). The connection between French and Dutch or German and Hungarian is also surprising, but it could be caused by some sharing caused by close geographical coexistence of these languages. We would also expect a higher similarity between Spanish and Greek.[23]

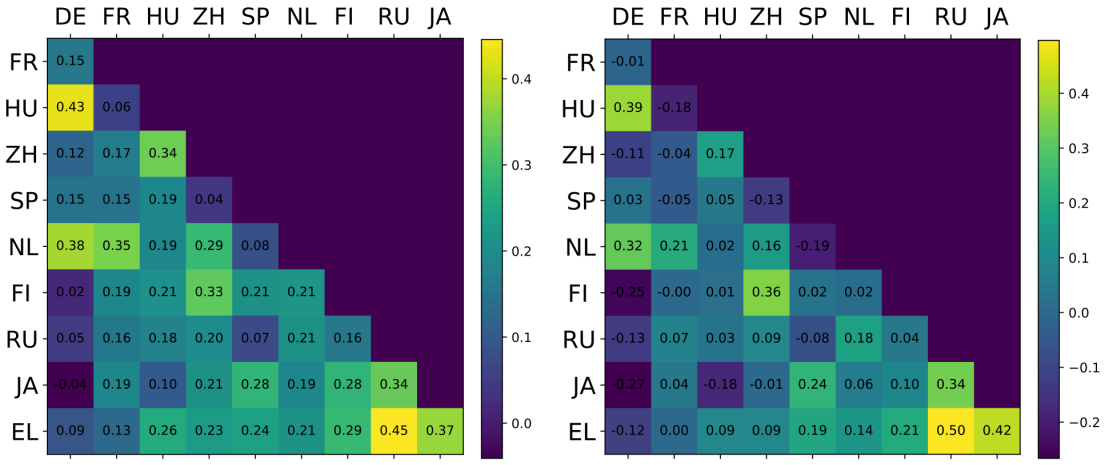

Figure 6.6: *Visualization of the cosine similarity of learned language embeddings of the Generated model.* Left: *Similarities of the model trained on the full training dataset.* Right: *Similarities of the model trained on the reduced training dataset with 900 sentences per language.*

Data-stress training: We are now going to discuss the effects of the training data reduction. Table 6.6 describes the performance of the Shared and the Generated models on both in-domain and out-of-domain test sets when trained on

[23]Note that Chinese and Japanese were romanized, while Greek and Russian were not.

reduced training sets.[24] It seems that the Generated model can work slightly better than the Shared model even in more data-stress situations. It has significantly lower CERs for both test sets in the case of French, Russian, and Finnish.

Table 6.6: *Summary of the CER evaluation of Shared and Generated models trained on reduced training datasets (with 900 examples and 600 examples per language). Upper and lower numbers correspond to the evaluation of in-domain and out-of-domain data, respectively. Bold texts emphasize models with the lowest average CER and "*" marks a statistically significant difference against another model (i.e., the better Generated model is compared with the better Shared model) according to a one-sided paired t-test with 95% confidence level.*

Language	**Shared 600 examples**	**Shared 900 examples**	**Generated 600 examples**	**Generated 900 examples**
German	13.2 ± 8.9 11.2 ± 7.7	$\mathbf{12.4 \pm 8.0}$ 8.9 ± 7.0	15.6 ± 9.4 12.4 ± 8.8	12.5 ± 9.3 $\mathbf{7.8 \pm 6.2}$
Greek	16.8 ± 9.7 11.3 ± 10.1	16.0 ± 10.2 $\mathbf{8.3 \pm 8.3}$	$\mathbf{14.2 \pm 8.7}$ 9.0 ± 8.9	14.7 ± 9.8 8.7 ± 10.5
Spanish	9.8 ± 7.5 6.6 ± 6.7	9.9 ± 8.4 5.2 ± 7.0	8.1 ± 6.0 $\mathbf{5.2 \pm 4.2}$	$^*\mathbf{7.6 \pm 5.9}$ 5.7 ± 6.8
Finnish	18.2 ± 12.2 14.8 ± 10.7	18.4 ± 13.2 16.0 ± 14.9	$^*\mathbf{13.2 \pm 10.9}$ $^*\mathbf{8.4 \pm 10.0}$	14.0 ± 10.6 9.8 ± 7.8
French	40.2 ± 15.8 39.2 ± 18.5	37.6 ± 16.2 34.5 ± 14.6	32.9 ± 13.2 31.4 ± 13.3	$^*\mathbf{27.2 \pm 12.2}$ $^*\mathbf{24.7 \pm 11.4}$
Hungarian	21.4 ± 10.4 13.4 ± 9.4	21.3 ± 13.0 12.2 ± 9.3	$^*\mathbf{16.5 \pm 10.4}$ 13.5 ± 10.3	18.0 ± 10.4 $\mathbf{11.4 \pm 7.7}$
Japanese	32.5 ± 12.8 36.4 ± 22.3	32.2 ± 15.0 34.0 ± 20.0	$\mathbf{29.9 \pm 13.0}$ 33.4 ± 29.2	30.9 ± 13.5 $\mathbf{28.8 \pm 17.1}$
Dutch	37.8 ± 13.5 34.6 ± 15.7	30.4 ± 10.2 27.4 ± 11.2	32.8 ± 12.3 28.6 ± 12.0	$\mathbf{28.3 \pm 9.80}$ $^*\mathbf{23.5 \pm 10.7}$
Russian	60.4 ± 18.6 51.2 ± 24.6	47.0 ± 20.5 36.4 ± 20.7	38.5 ± 20.1 34.8 ± 23.4	$^*\mathbf{34.4 \pm 17.9}$ $^*\mathbf{25.6 \pm 21.7}$
Chinese	40.2 ± 15.2 51.0 ± 20.4	39.8 ± 18.8 50.5 ± 21.4	33.0 ± 15.5 47.1 ± 20.3	$^*\mathbf{28.4 \pm 15.6}$ $\mathbf{41.6 \pm 20.9}$

In comparison with the models trained on the whole dataset (see Table 6.4), these models are much worse according to CERs. However, we inspected the produced audio samples manually and the difference is less striking: for example, the Generated model trained on 600 examples per language seems to produce intelligible natural-sounding speech for German or French (however, with numerous mispronunciations).

For the sake of completeness, we also present the MCD evaluation. See Table 6.7.

[24]Note that the other two models did not train successfully in this setting.

We can notice that the Generated model reaches significantly lower values for four languages (French, Dutch, Spanish, and Japanese). These lower numbers might reflect less frequent mispronunciations or a more intelligible or natural speech.

Conclusions: To summarize conclusions from this experiment, we found out that it is beneficial to train Tacotron on multilingual data if we do not have a large monolingual dataset. We realized that the Generated model has some advantages, for example, for French, Dutch, or Russian, over other architectures and that the Separate model does not scale to more languages very well. We also found out that it is possible to train a multilingual TTS model that can synthesize imperfect but intelligible speech with less than two hours of aligned data per language.

Table 6.7: *Summary of MCDs of Shared and Generated models trained on reduced training datasets (with 900 examples and 600 examples per language). Bold texts highlight models with the lowest average MCD and "*" marks a statistically significant difference against another model (i.e., the better Generated model is compared with the better Shared model) according to a one-sided paired t-test with 95% confidence level.*

Language	Shared 600 utterances	Shared 900 utterances	Generated 600 utterances	Generated 900 uterances
German	4.91 ± 1.17	4.80 ± 1.04	4.70 ± 1.19	$\mathbf{4.53 \pm 1.08}$
Greek	3.87 ± 0.54	3.84 ± 0.52	$\mathbf{3.78 \pm 0.51}$	3.89 ± 0.56
Spanish	4.34 ± 0.61	4.21 ± 0.65	4.08 ± 0.54	$\mathbf{*3.96 \pm 0.54}$
Finnish	3.61 ± 0.67	3.70 ± 0.77	$\mathbf{3.52 \pm 0.77}$	3.57 ± 0.81
French	4.73 ± 0.55	4.65 ± 0.53	4.41 ± 0.47	$\mathbf{*4.40 \pm 0.51}$
Hungarian	3.57 ± 0.63	3.47 ± 0.51	3.40 ± 0.53	$\mathbf{3.34 \pm 0.54}$
Japanese	3.63 ± 0.34	3.53 ± 0.35	3.47 ± 0.36	$\mathbf{*3.41 \pm 0.34}$
Dutch	4.42 ± 0.39	4.35 ± 0.39	4.27 ± 0.34	$\mathbf{*4.14 \pm 0.35}$
Russian	7.44 ± 1.70	$\mathbf{6.82 \pm 1.81}$	7.25 ± 1.95	7.44 ± 1.87
Chinese	3.72 ± 0.51	3.83 ± 0.66	3.70 ± 0.56	$\mathbf{3.59 \pm 0.56}$

6.4 Code-switching and Voice Cloning

The second and last experiment that we would like to present examined code-switching and voice-cloning abilities of the three models. Base Tacotron is excluded since monolingual models are not capable of code-switching in principle. We again refer to the GitHub repository of this work that contains links to many synthesized samples and interactive demos.[25]

[25]https://github.com/Tomiinek/Multilingual_Text_to_Speech

6.4.1 Experiment Setup

Training setup: Contrary to the previous experiment, we trained just the Shared, Separate, and Generated models. We used the dataset that includes a combination of cleaned Common Voice and cleaned CSS10 data (see Section 6.2). It contains five languages, namely German, French, Dutch, Russian, and Chinese. For vocoding, we used the same WaveRNN model as in Section 6.3.

All three models were trained for approximately 50k training steps. We used hyperparameters similar to those described in Section 6.3.1,[26] i.e., we halved the learning rate every 10k steps and in the case of the Separate model, we used an initial learning rate equal to 10^{-4}. We set the size of the speaker embeddings to 32 and we used the language embedding of size 4 in the case of the Shared model. The Generated model uses language embeddings of size 10 and generator layers of size 4. We enabled the adversarial speaker classifiers of the Shared and Generated models. They use a hidden layer of size 256 and their losses have weights 0.5 and 0.125, respectively. We used mini-batches of size 50 for all models.

Evaluation dataset: An objective evaluation of voice cloning or code-switching is difficult. First, our datasets do not include code-switching sentences, so there are no ground-truth recordings and the computation of the MCD is impossible. Secondly, a calculation of the CER requires an ASR engine that can handle code-switching. However, we do not have such an engine at disposal. Thus we had to settle for a subjective comparison. For the evaluation, we used bilingual sentences scraped from Wikipedia. We collected 400 sentences in total. For each language in the dataset, we picked 80 sentences with a few foreign words (20 sentences for each secondary language out of the four remaining ones). We romanized Chinese using the pipeline described in a previous section. In most cases, the foreign words are names of cities, places, or celebrities. It is common to transcribe Russian and Chinese names to Latin, but also vice-versa, i.e., Russian and Chinese transcribe German, Dutch, or French names into Cyrillic or Chinese scripts. That is why we replaced foreign names with their native forms. For example, we changed "délākèluówǎ shì fǎguó ..." to "Delacroix shì fǎguó ...", etc., see Figure 6.7.

Figure 6.7: *Examples of code-switching evaluation sentences.*

Similarly as in Section 6.3, we only collected clean sentences without any numbers and non-standard characters, thus we did not have to do any cleaning or normalization. The whole test set is available on GitHub.[27] We synthesized the evaluation sentences with speaker embeddings of CSS10 speakers for the corresponding base language.

[26] See https://github.com/Tomiinek/Multilingual_Text_to_Speech/tree/master/params for all parameter configurations used in the experiments described here.

[27] https://github.com/Tomiinek/Multilingual_Text_to_Speech/tree/master/evaluation

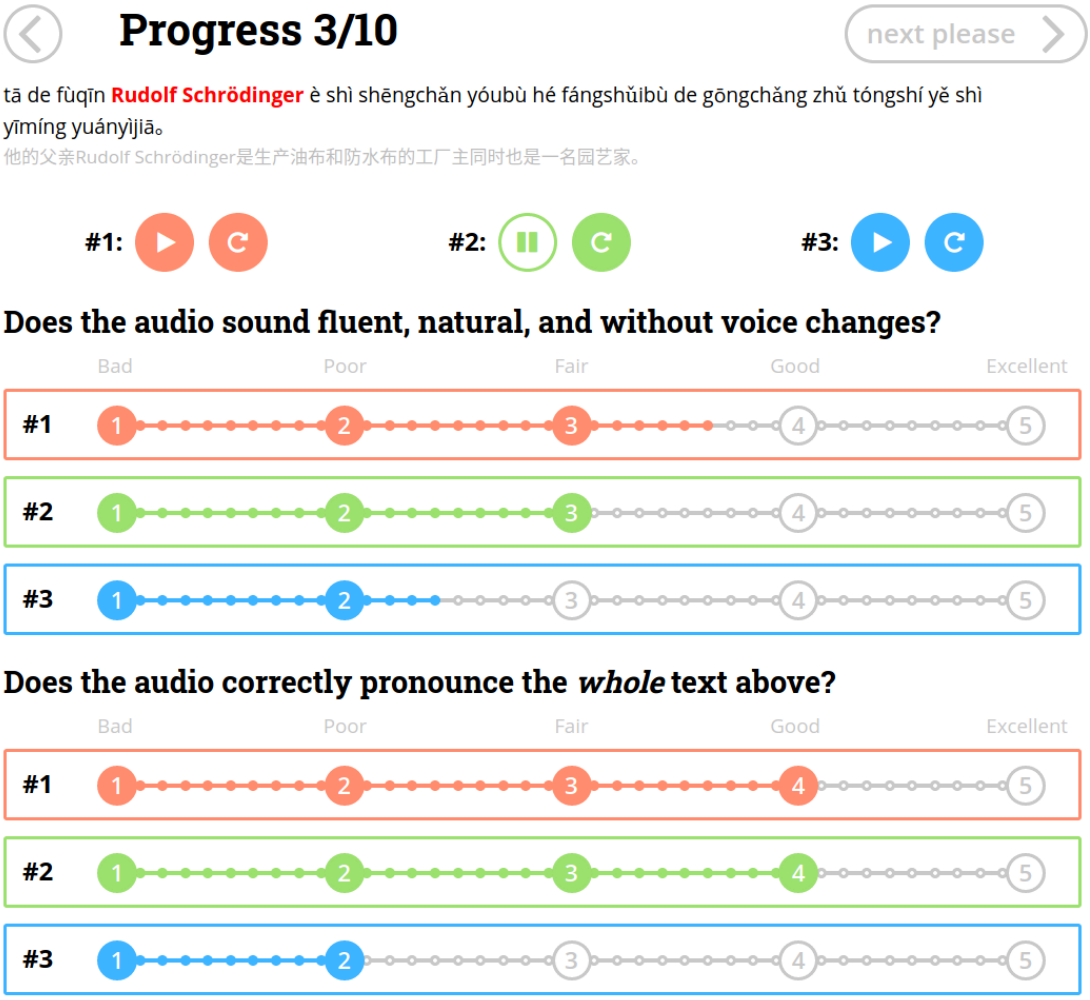

Figure 6.8: *User interface for the survey. There is a particular code-switching sentence at the top of the page. The three recordings can be controlled by red, green, and blue buttons. The recordings are randomly shuffled, so #1, #2, #3 do not correspond to the outputs of the same model throughout the survey. Note the two questions and their scales. We highlighted the word "whole" of the second question to encourage participants to penalize recordings because of incompleteness. The scales have labels ("Bad", "Poor", "Fair", "Good", and "Excellent") and five emphasized points. Two consecutive points correspond to 0.1 increments.*

Subjective evaluation setup: We realized the evaluation itself via an online survey. Participants were first asked about their language abilities. For each of the five languages, they had to fill in one of these options: "I am native", "I understand spoken", "I am able to read", "I have no experience". Afterward, they were given a set of samples to rate. We used a custom rating approach which is a combination of the traditional Mean opinion score with five-point scale and MUSHRA (see Chapter 3). For each sample, the transcript (with its romanized variant) and outputs of all three compared models were shown at the same time. Participants were allowed to replay the recordings and move between the samples in order to refine their ratings. They were asked to rate the recordings on two scales from 1 to 5 with 0.1 increments and with the usual labels "Bad", "Poor", "Fair", "Good", and "Excellent". The first scale concerned fluency, naturalness,

and stability of the voice and the second aimed at pronunciation accuracy. See Figure 6.8 for further details. The participants were also asked to use headphones during the evaluation. They could leave us a note at the end of the survey.

We decided to separate the two ratings because we would like to distinguish two potential types of errors: (1) when the foreign words cause a change of speaker's voice and (2) when they are read fluently but the model does not respect their correct pronunciation. These could be roughly described as fluency and adequacy ratings. We make this distinction even though we are aware that these two ratings are hard to disentangle. For example, if a model skips a part of a sentence (or in an extreme case the whole sentence), we would like to strongly penalize the pronunciation accuracy, but it is questionable whether to also reduce the fluency rating. On one hand, the model did not say that part and could not show us its abilities, on the other, all the words it said could be correct and natural. We left the decision about how to rate in these situations to the study participants.

Participants: For each language, we hired ten native speakers who speak fluently at least one of the other languages (for example, we hired Chinese participants that are fluent in German, French, Dutch, or Russian) via the Prolific crowdsourcing platform.[28] Each native speaker of a particular language was given twelve sentences where their native language was primary. Each of the secondary languages was represented by three sentences. Besides the hired native speakers, we asked 21 volunteers that are not native speakers of any of those languages to also participate in the evaluation. We gave them ten sentences where the primary and secondary languages were chosen at random.

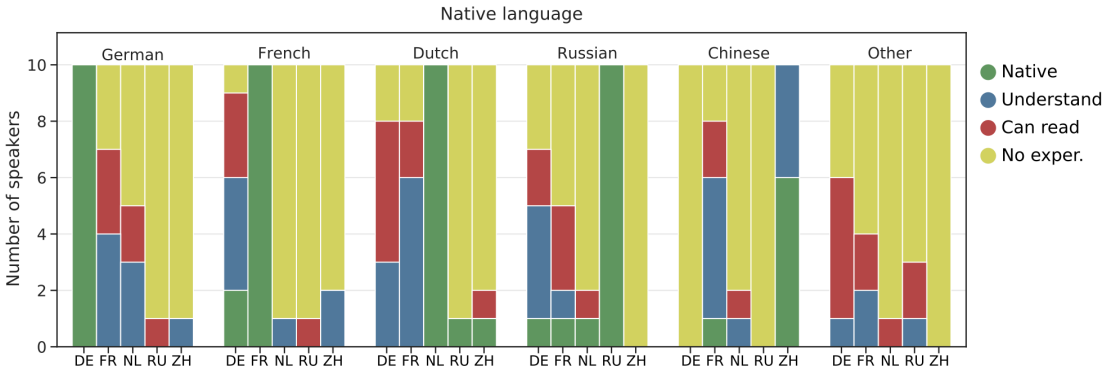

Figure 6.9: *Bar chart showing the language abilities of the participants of our survey. The six groups gather participants with different native languages (the Chinese group does not have ten native speakers, but all of them report as native Chinese to Prolific). The "Other" group consists mostly of Czechs. All the groups include five bars, each of them is labeled with a language code and shows the distribution of the abilities of the ten corresponding participants (as they stated at the start of our survey).*

During the rating for both participant groups, we replaced three random system outputs with their severely distorted versions that acted as sanity checks – any participants that fail to rank the distorted system outputs as the lowest would

[28]Prolific, https://www.prolific.co, accessed May 25, 2020

be removed from the study. We, of course, removed the ratings associated with these checks during evaluation. All hired participants passed this check, but three volunteers did not. We also removed ratings of 8 volunteers who stated poor language abilities (for example, they were able to read only German).

Figure 6.9 summarizes the language abilities of the participants who took part in our survey. Even though we hired ten Chinese speakers who reported as natives on Prolific, just six of them stated in our survey that they are natives (the rest declared understanding spoken language). Some Dutch native participants could also speak both German and French simultaneously. Nobody who would understand both Chinese and Russian at the same time participated in the survey.

6.4.2 Discussion of Results

Table 6.8 summarizes the results of the survey. First, consider the two bottom rows labeled "Total" and "Others". They show means and variances of the ratings of all hired native speakers and the volunteers, respectively. See Figure 6.10 that compares ratings in these two groups. The Generated model is significantly better in comparison with the two other models, except for the case of pronunciation ratings of volunteer non-native participants. It is noticeable that the fluency or naturalness ratings of the hired participants are slightly lower (and have lower variance). Also, the accuracy ratings are lower, but more profoundly. Thus we conclude that the non-native participants cannot distinguish between a correct and slightly incorrect pronunciation. For example, a Dutch and a few Russian speakers noted at the end of the survey that they encountered stresses on inappropriate parts of words and so they reduced the pronunciation ratings. We think that the non-native participants that are most often able just to read the text could not catch these nuances.

Table 6.8: *Summary of mean "Naturalness, fluency, stability" and "Pronunciation accuracy" ratings collected in our survey. The upper part of the table groups ratings of hired participants by their native languages. The lower part reports the statistics of all hired participants' ratings and also of all ratings of the other participants who are not native speakers of any of the five languages. Besides the means, we show also corresponding standard deviations. Bold font highlights models with the lowest mean rating and "*" marks a statistically significant difference against other models according to a paired t-test with 95% confidence level.*

Participant group	**Naturalness, fluency, stability**			**Pronunciation accuracy**		
	Generated	**Shared**	**Separate**	**Generated**	**Shared**	**Separate**
German	***3.4±0.9**	3.0 ± 1.1	2.6 ± 1.0	***3.7±1.0**	3.3 ± 1.1	3.1 ± 1.2
French	***3.5±0.9**	2.8 ± 1.0	2.6 ± 1.0	***3.7±0.9**	3.1 ± 1.1	2.7 ± 1.2
Dutch	***3.7±1.0**	3.1 ± 0.9	2.5 ± 1.1	***3.9±1.1**	3.4 ± 1.0	2.5 ± 1.2
Russian	***3.4±0.9**	2.8 ± 1.0	2.5 ± 1.0	***3.6±1.0**	3.0 ± 1.2	2.6 ± 1.2
Chinese	***3.5±1.2**	2.7 ± 1.3	2.6 ± 1.2	***3.5±1.2**	2.9 ± 1.4	2.8 ± 1.4
Total	***3.5±1.0**	2.9 ± 1.1	2.5 ± 1.1	***3.7±1.1**	3.1 ± 1.2	2.7 ± 1.2
Others	***3.7±1.1**	3.2 ± 1.1	2.9 ± 1.3	**3.9±1.1**	3.6 ± 1.2	3.2 ± 1.4

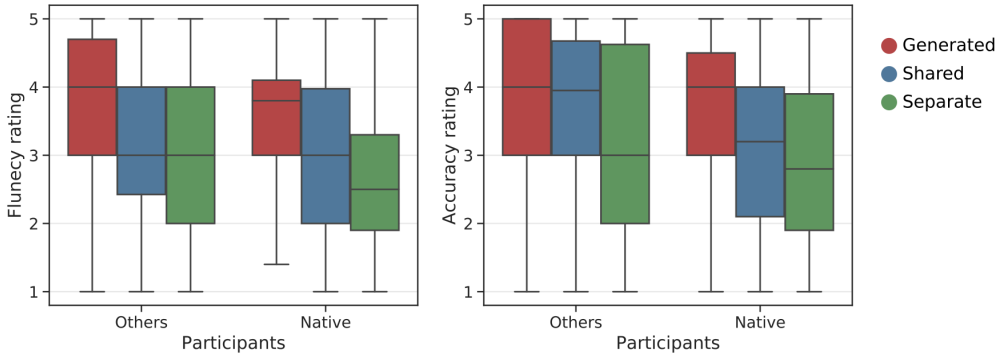

Figure 6.10: *Comparison of all ratings of naturalness or fluency and accuracy by different groups of participants, i.e., hired native speakers and volunteers that are not native speakers of any of the five languages. The whiskers show 5% and 95% quantiles and boxes represent quartiles. Solid lines inside boxes depict medians.*

We are going to take a closer look at the ratings of the hired participants. Table 6.8 shows that the Generated model has significantly higher mean ratings. In contrast, the Separate model seems to have the worst ratings constantly. Both the Generated and the Shared model have the best mean rating on Dutch sentences and the biggest gap is between their Chinese mean ratings. The lowest difference is present in the case of German. Again, the accuracy ratings are slightly higher in the comparison with the fluency ratings. See also Figure 6.11 and Figure 6.12 that group ratings by the primary language. We omit graphs that describe the ratings grouped by the secondary languages because we found out that the ratings across these groups do not vary very much. It seems that the performance depends more on the particular primary language.

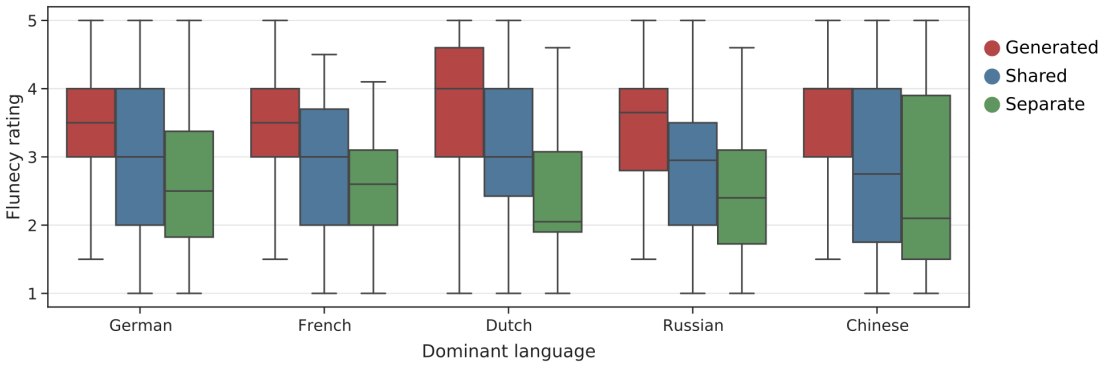

Figure 6.11: *Comparison of the "fluency, naturalness, or stability" ratings of native speakers grouped by their native language (which is identical with the dominant languages of the rated sentences). Red, blue, and green colors correspond to the Generated, Shared, and Separate model, respectively. The whiskers show 5% and 95% quantiles and boxes represent quartiles with the median.*

Manual error analysis: We manually inspected all the 400 synthesized recordings of all three models. We think that any lower ratings in case of of German, French, Russian, and Dutch are mainly caused by inappropriate pronunciations

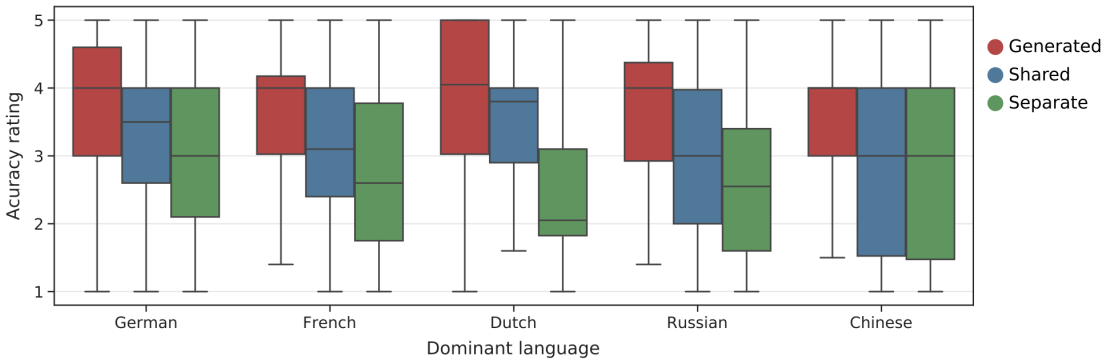

Figure 6.12: *Comparison of the "pronunciation accuracy" ratings of native speakers grouped by their native language (which is identical with the dominant languages of the rated sentences). Red, blue, and green colors correspond to the Generated, Shared, and Separate model, respectively. The whiskers show 5% and 95% quantiles and boxes represent quartiles with the median.*

of the foreign words. We should note the higher variance of ratings in the case of Chinese. This is probably because of the bad stability of the Shared and Separate models. We found out that these models when reading Chinese sometimes completely lose attention and skip some words or parts of synthesized sentences. This happens especially when reaching foreign words. To further examine this problem, we counted the incomplete sentences (including those with a single missing word) for each model (see Table 6.9). The errors in French system's outputs are less serious (such as a single or a couple of missing words) than in the Chinese ones (a completely skipped part of a sentence). Possibly, these issues would not be present if we used phoneme inputs.

Dominant language	**Number of incomplete sentences**		
	Generated	**Shared**	**Separate**
German	**0**	2	3
French	**5**	10	8
Dutch	**1**	3	3
Russian	**2**	4	3
Chinese	**6**	22	21
Total	**14**	41	38

Table 6.9: *Summary of the number of incomplete sentences (i.e., synthesized recordings with missing words) that were produced by the three models.*

Conclusions: To sum up our findings, we found that the Generated model performs significantly better than the other two models on the code-switching task. It is more stable and reaches higher ratings. This experiment partially answers the question on the possibility of code-switching from the beginning of this chapter. Even the Shared and Separate models can code-switch to an extent when similar languages are concerned. However, they have problems with more diverse inputs that mix languages with very diverse pronunciation. The

Generated model seems to be more suitable for the grapheme-based synthesis. However, it is still not ideal and occasionally produces imperfect outputs.

Finally, we would like to mention that the Generated model allows cross-lingual mixing of encoder outputs, which is not possible with the Shared or the Separate model. This allows us to smoothly control pronunciation, so, for example, we can weight the outputs of German and French encoders and successively obtain different pronunciations of some words. Let us consider the name "Jean-Paul". By the weighting of encoders, we can synthesize pronunciations from German [jeːʔan paʊ̯l] through [ʒɑ̃n paʊ̯l] to French [ʒɑ̃ pɔl].

Conclusion

As described in the Introduction, the goal of this work is to implement a system for speech synthesis based on neural networks, train it on multiple languages, evaluate its performance, and compare it with monolingual or other approaches.

In view of that, we implemented the Tacotron 2 model [Shen et al., 2018] for spectrogram generation from input alphabetic characters that utilizes a few ideas from the DCTTS architecture [Tachibana et al., 2017]. On top of that, we propose a novel approach to multilingual speech synthesis. Our meta-learning based model, which we call *Generated*, uses a parameter generator network to enable cross-lingual parameter sharing (Section 6.1.4). We compared it with two other approaches which perform a full and no parameter sharing. We call these models *Shared* (Section 6.1.2) and *Separate* (Section 6.1.3), respectively. We explored the code-switching or voice cloning abilities of the three models that we extended to support multi-speaker multilingual speech synthesis. The enhancement is, in the case of the Shared and Generated models, based on domain adversarial training.

For the purpose of converting the synthesized spectrograms into audio waves, we trained an open-source implementation of the WaveRNN vocoder Kalchbrenner et al. [2018], see Section 6.1.5 for more details.

We compared our models on two tasks. The first one aimed at joint multilingual training (Section 6.3). We trained the models on our cleaned version of the CSS10 dataset (Section 6.2.1) that includes ten languages, namely German, Greek, Spanish, Finnish, French, Hungarian, Japanese, Dutch, Russian, and Chinese. We also examined the models' behavior when training on small subsets of the dataset, in the most extreme case just using approximately 80 minutes of recordings per language. We found out that the sharing of encoder parameters is helpful and that the Generated model shows a slightly better performance (with respect to a character error rate and mel cepstral distortion evaluation) in comparison with Shared and Separate models and individual monolingual baselines.

The second evaluation task concerned code-switching (Section 6.4). For training, we used a subset of the CSS10 dataset that includes German, French, Dutch, Russian, and Chinese. Moreover, we enriched it by a multi-speaker dataset that we created from Common Voice data (Section 6.2.2). For the purpose of evaluation, we created a dataset that includes four hundred code-switching sentences. We arranged a survey for subjective evaluation where sixty participants took part. The Generated model was rated significantly better than the other two architectures.

To summarize the whole work, we introduced a novel model for multilingual text-to-speech synthesis based on meta-learning. It is capable of more effective multilingual training, it enables a more stable and natural code-switching, voice cloning, and pronunciation control, and it works with raw texts that do not require expensive preprocessing such as phonemicization.

Future work: We consider multiple experiments with the Generated model.

First, it would be interesting to incorporate the TUNDRA dataset (Section 4.2) into the experiments with multilingual training. It would allow us to use seven more languages, and as the experiments with training in data-stress situations shown, we would probably be able to use all of the languages including Danish with less than one hour of aligned data.

Secondly, it would be great to examine fine-tuning of the Generated model (e.g., with fixed encoder and decoder parameters, but with trainable speaker and language embeddings) to new languages. We think that the multilingual character of the model could enable an easy adaptation with a low amount of data.

Finally, we think that it would be interesting to further investigate the stability of code-switching synthesis. We also would like to incorporate more languages into the code-switching model. The Common Voice dataset grows quickly and we think that (at the time of writing this text) it would be probably possible to gather also some Spanish and Japanese multi-speaker data.

Besides these straightforward experiments, we think about extending the Generated model to enable a few-shot speaker adaptation.

Bibliography

Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M. Tyers, and Gregor Weber. Common Voice: A Massively-Multilingual Speech Corpus. *ArXiv*, abs/1912.06670, 2019.

Sercan Arik, Mike Chrzanowski, Adam Coates, Gregory Frederick Diamos, Andrew Gibiansky, Yongguo Kang, Xiongmin Li, John Miller, Andrew Ng, Jonathan Raiman, Shubho Sengupta, and Mohammad Shoeybi. Deep Voice: Real-time Neural Text-to-Speech. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 195–204, 2017.

Alan Baddeley. *Your Memory, a User's Guide*. Collier, 1984. ISBN 9780020753100.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations*, 2015.

Benjamin Barras. SoX : Sound eXchange. *Flash informatique*, 2012.

Eric Battenberg, RJ Skerry-Ryan, Soroosh Mariooryad, Daisy Stanton, David Kao, Matt Shannon, and Tom Bagby. Location-Relative Attention Mechanisms For Robust Long-Form Speech Synthesis. *ArXiv*, abs/1910.10288, 2019.

Mikołaj Bińkowski, Jeff Donahue, Sander Dieleman, Aidan Clark, Erich Elsen, Norman Casagrande, Luis C. Cobo, and Karen Simonyan. High Fidelity Speech Synthesis with Adversarial Networks. In *International Conference on Learning Representations*, 2020.

Alan W. Black. CMU Wilderness Multilingual Speech Dataset. In *ICASSP 2019 - IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5971–5975, 2019.

Marcely Zanon Boito, William N. Havard, Mahault Garnerin, Éric Le Ferrand, and Laurent Besacier. MaSS: A Large and Clean Multilingual Corpus of Sentence-aligned Spoken Utterances Extracted from the Bible. *ArXiv*, abs/1907.12895, 2019.

E. O. Brigham and R. E. Morrow. The fast Fourier transform. *IEEE Spectrum*, 4(12):63–70, 1967.

M. Brown and L. Rabiner. Dynamic time warping for isolated word recognition based on ordered graph searching techniques. In *ICASSP '82 - IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 7, pages 1255–1258, 1982.

Alan W. Black Nick Campbell. Optimising selection of units from speech databases for concatenative synthesis. In *EUROSPEECH-1995*, pages 581–584, 1995.

Yuewen Cao, Xixin Wu, Songxiang Liu, Jianwei Yu, Xu Li, Zhiyong Wu, Xunying Liu, and Helen M. Meng. End-to-end Code-switched TTS with Mix of Monolingual Recordings. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6935–6939, 2019.

Yuan-Jui Chen, Tao Tu, Cheng chieh Yeh, and Hung-Yi Lee. End-to-End Text-to-Speech for Low-Resource Languages by Cross-Lingual Transfer Learning. In *Proc. Interspeech 2019*, pages 2075–2079, 2019.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

Jan K. Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-Based Models for Speech Recognition. In *Advances in Neural Information Processing Systems 28*, pages 577–585. 2015.

Y. Chung, Y. Wang, W. Hsu, Y. Zhang, and R. J. Skerry-Ryan. Semi-supervised Training for Improving Data Efficiency in End-to-end Speech Synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6940–6944, 2019.

Paul Andrew Denisowski. CC-CEDICT editor. https://cc-cedict.org/editor/editor.php, 2020. Accessed February 29, 2020.

Homer Dudley and T. H. Tarnoczy. The Speaking Machine of Wolfgang von Kempelen. *The Journal of the Acoustical Society of America*, 22(2):151–166, 1950.

Peter Ebden and Richard Sproat. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21:333–353, 2015.

Fatchord. WaveRNN Vocoder + TTS. https://github.com/fatchord/WaveRNN, 2019. Accessed March 31, 2020.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-Adversarial Training of Neural Networks. *Journal of Machine Learning Research (JMLR)*, 17(1):2096–2030, 2016.

Andrew Gibiansky, Sercan Arik, Gregory Diamos, John Miller, Kainan Peng, Wei Ping, Jonathan Raiman, and Yanqi Zhou. Deep Voice 2: Multi-Speaker Neural Text-to-Speech. In *Advances in Neural Information Processing Systems 30*, pages 2962–2970. 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

D. Griffin and J. Lim. Signal estimation from modified short-time Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32(2): 236–243, 1984.

Casper Hansen. Activation Functions Explained – GELU, SELU, ELU, ReLU and more, 2019. URL https://mlfromscratch.com/activation-functions-explained/.

Fredric J. Harris. On the Use of Windows for Harmonic Analysis With the Discrete Fourier Transform. In *Proc. IEEE*, pages 51–83, 1978.

K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Hee-Soo Heo, Jee weon Jung, Hye jin Shim, Il-Ho Yang, and Ha-Jin Yu. Cosine similarity-based adversarial process. *ArXiv*, abs/1907.00542, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural Computation*, 9:1735–80, 1997.

W. Hsu, Y. Zhang, R. J. Weiss, Y. Chung, Y. Wang, Y. Wu, and J. Glass. Disentangling Correlated Speaker and Noise for Speech Synthesis via Data Augmentation and Adversarial Factorization. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5901–5905, 2019.

W. Hsu, Y. Zhang, R. J. Weiss, H. Zen, Y. Wu, Y. Cao, and Y. Wang. Hierarchical Generative Modeling for Controllable Speech Synthesis. In *International Conference on Learning Representations*, 2019.

Keith Ito. The LJ Speech Dataset. https://keithito.com/LJ-Speech-Dataset/, 2017. Accessed February 29, 2020.

Keith Ito. A TensorFlow implementation of Google's Tacotron speech synthesis with pre-trained model. https://github.com/keithito/tacotron, 2018. Accessed February 29, 2020.

ITU-T. Pulse code modulation (PCM) of voice frequencies. Recommendation G.711, International Telecommunication Union, November 1988.

ITU-T. Subjective video quality assessment methods for multimedia applications. Recommendation P.910, International Telecommunication Union, April 2008.

ITU-T. Method for the subjective assessment of intermediate quality levels of coding systems. Recommendation BS.1534, International Telecommunication Union, October 2015.

ITU-T. Mean opinion score (MOS) terminology. Recommendation P.800.1, International Telecommunication Union, July 2016a.

ITU-T. Mean opinion score interpretation and reporting. Recommendation P.800.2, International Telecommunication Union, July 2016b.

Ye Jia, Yu Zhang, Ron Weiss, Quan Wang, Jonathan Shen, Fei Ren, Zhifeng Chen, Patrick Nguyen, Ruoming Pang, Ignacio Lopez Moreno, and Yonghui Wu. Transfer Learning from Speaker Verification to Multispeaker Text-To-Speech Synthesis. In *Advances in Neural Information Processing Systems 31*, pages 4480–4490. 2018.

Dan Jurafsky and James H. Martin. *Speech and language processing*. Prentice Hall, Pearson Education International, 2014.

Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aaron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient Neural Audio Synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 2410–2419, 2018.

Simon J. King. An introduction to statistical parametric speech synthesis. *Sadhana-Academy proceedings in engineering sciences*, 36:837–852, 2011.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, 2015.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations*, Banff, AB, Canada, 2014.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing RNNs by Randomly Preserving Hidden Activations. *ArXiv*, abs/1606.01305, 2018.

R. Kubichek. Mel-cepstral distance measure for objective speech quality assessment. In *Proceedings of IEEE Pacific Rim Conference on Communications Computers and Signal Processing*, volume 1, pages 125–128, 1993.

Taku Kudo. MeCab: Yet Another Part-of-Speech and Morphological Analyzer. https://taku910.github.io/mecab/, 2015. Accessed March 31, 2020.

Kundan Kumar, Rithesh Kumar, Thibault de Boissiere, Lucas Gestin, Wei Zhen Teoh, Jose Sotelo, Alexandre de Brébisson, Yoshua Bengio, and Aaron C Courville. MelGAN: Generative Adversarial Networks for Conditional Waveform Synthesis. In *Advances in Neural Information Processing Systems 32*, pages 14910–14921. 2019.

Javier Latorre, Jakub Lachowicz, Jaime Lorenzo-Trueba, Thomas Merritt, Thomas Drugman, Srikanth Ronanki, and Klimkov Viacheslav. Effect of Data Reduction on Sequence-to-sequence Neural TTS. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7075–7079, 2019.

Younggun Lee and Taesu Kim. Robust and Fine-grained Prosody Control of End-to-end Speech Synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5911–5915, 2019.

Younggun Lee, Suwon Shon, and Taesu Kim. Learning pronunciation from a foreign language in speech synthesis networks. *ArXiv*, abs/1811.09364, 2018.

Francis F. Li and Trevor J. Cox. *Digital signal processing in audio and acoustical engineering*. CRC Press, Taylor & Francis Group, 2019.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*, 2019.

M-AILABS. The M-AILABS Speech Dataset. https://www.caito.de/2019/01/the-m-ailabs-speech-dataset/, 2019. Accessed February 29, 2020.

Rayhane Mama. DeepMind's Tacotron-2 Tensorflow implementation. https://github.com/Rayhane-mamah/Tacotron-2, 2018. Accessed February 29, 2020.

Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi. In *Proc. Interspeech 2017*, pages 498–502, 2017.

Brian McFee, Colin Raffel, Dawen Liang, Daniel Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and Music Signal Analysis in Python. In *Proceedings of the 14th Python in Science Conference*, 2015.

David R. Mortensen, Siddharth Dalmia, and Patrick Littell. Epitran: Precision G2P for Many Languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

E. Nachmani and L. Wolf. Unsupervised Polyglot Text-to-speech. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

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

Kyubyong Park. A TensorFlow implementation of Tacotron: A fully end-to-end text-to-speech synthesis model. https://github.com/Kyubyong/tacotron, 2018. Accessed February 29, 2020.

Kyubyong Park and Thomas Mulc. CSS10: A Collection of Single Speaker Speech Datasets for 10 Languages. In *Proc. Interspeech 2019*, pages 1566–1570, 2019.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the Difficulty of Training Recurrent Neural Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, page III–1310–III–1318, 2013.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie

Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of North American Chapter of the Association for Computational Linguistics*, 2018.

Alberto Pettarin. Aeneas. https://github.com/readbeyond/aeneas/, 2017. Accessed February 29, 2020.

Luis Pineda, Hayde Castellanos, Javier Cuetara, Lucian Galescu, Janet Juárez, Joaquim Llisterri, Patricia Pérez, and Luis Villaseñor-Pineda. The Corpus DIMEx100: Transcription and evaluation. *Language Resources and Evaluation*, 44:347–370, 2009.

Wei Ping, Kainan Peng, Andrew Gibiansky, Sercan O. Arik, Ajay Kannan, Sharan Narang, Jonathan Raiman, and John Miller. Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning. In *International Conference on Learning Representations*, 2018.

Emmanouil Antonios Platanios, Mrinmaya Sachan, Graham Neubig, and Tom Michael Mitchell. Contextual Parameter Generation for Universal Neural Machine Translation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 425–435, 2018.

Anusha Prakash, Anju Leela Thomas, S. Umesh, and Hema A Murthy. Building Multilingual End-to-End Speech Synthesisers for Indian Languages. In *Proc. 10th ISCA Speech Synthesis Workshop*, pages 194–199, 2019.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A Flow-based Generative Network for Speech Synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621, 2019.

Devendra Sachan and Graham Neubig. Parameter Sharing Methods for Multilingual Self-Attentional Translation Models. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 261–271, 2018.

David Samuel, Aditya Ganeshan, and Jason Naradowsky. Meta-learning Extractors for Music Source Separation. *ArXiv*, abs/2002.07016, 2020.

Jonathan Shen, Ruoming Pang, Ron Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, R. J. Skerrv-Ryan, Rif Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural TTS Synthesis by Conditioning Wavenet on MEL Spectrogram Predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4779–4783, 2018.

RJ Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Rob Clark, and Rif A. Saurous. Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron. *ArXiv*, abs/1803.09047, 2018.

L. N. Smith. Cyclical Learning Rates for Training Neural Networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 464–472, 2017.

R. William Soukoreff and I. Scott MacKenzie. Measuring errors in text entry tasks: an application of the Levenshtein string distance statistic. In *CHI EA '01*, 2001.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training Very Deep Networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, page 2377–2385, 2015.

Adriana Stan, Oliver Watts, Y. Mamiya, Mircea Giurgiu, R. Clark, and Junichi Yamagishi. TUNDRA: A multilingual corpus of found data for TTS research created with light supervision. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2013.

S. S. Stevens, J. Volkmann, and E. B. Newman. A Scale for the Measurement of the Psychological Magnitude Pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

Junyi Sun. Jieba. https://github.com/fxsjy/jieba, 2018. Accessed February 29, 2020.

Surfingtech. ST-CMDS-20170001_1, Free ST Chinese Mandarin Corpus. https://www.openslr.org/38/, 2017. Accessed February 29, 2020.

Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28, pages 1139–1147, 2013.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to Sequence Learning with Neural Networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. 2014.

Hideyuki Tachibana, Katsuya Uenoyama, and Shunsuke Aihara. Efficiently Trainable Text-to-Speech System Based on Deep Convolutional Networks with Guided Attention. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4784–4788, 2017.

Yaniv Taigman, Lior Wolf, Adam Polyak, and Eliya Nachmani. VoiceLoop: Voice Fitting and Synthesis via a Phonological Loop. In *International Conference on Learning Representations*, 2018.

Paul Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009. ISBN 9780521899277.

Rafael Valle. Tacotron 2 - PyTorch implementation with faster-than-realtime inference. https://github.com/NVIDIA/tacotron2, 2020. Accessed March 31, 2020.

Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alexander Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. WaveNet: A Generative Model for Raw Audio. *Arxiv*, abs/1609.03499, 2016.

Sean Vasquez and Mike Lewis. MelNet: A Generative Model for Audio in the Frequency Domain. *Arxiv*, abs/1906.01083, 2019.

Christophe Veaux, Junichi Yamagishi, and Kirsten Macdonald. CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit, 2017. URL https://doi.org/10.7488/ds/1994.

Yuxuan Wang, R.J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif Saurous. Tacotron: Towards End-to-End Speech Synthesis. In *Proc. Interspeech 2017*, pages 4006–4010, 2017.

Yuxuan Wang, Daisy Stanton, Yu Zhang, RJ-Skerry Ryan, Eric Battenberg, Joel Shor, Ying Xiao, Ye Jia, Fei Ren, and Rif A. Saurous. Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 5180–5189, 2018.

Yuxin Wu and Kaiming He. Group Normalization. In *The European Conference on Computer Vision (ECCV)*, 2018.

Mort Yao. A Romaji/Kana conversion library for Python. https://github.com/soimort/python-romkan, 2013. Accessed February 29, 2020.

Lixin Yu. Pinyin. https://github.com/lxyu/pinyin, 2016. Accessed February 29, 2020.

Heiga Zen. Acoustic Modeling in Statistical Parametric Speech Synthesis - From HMM to LSTM-RNN. In *Proceedings of Machine Learning in Speech and Language Processing*, 2015.

Heiga Zen, Keiichi Tokuda, and Alan W. Black. Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064, 2009.

Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J. Weiss, Ye Jia, Zhifeng Chen, and Yonghui Wu. LibriTTS: A Corpus Derived from LibriSpeech for Text-to-Speech. In *Proc. Interspeech 2019*, pages 1526–1530, 2019.

Jing-Xuan Zhang, Zhen-Hua Ling, and Li-Rong Dai. Forward Attention in Sequence- To-Sequence Acoustic Modeling for Speech Synthesis. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4789–4793, 2018.

Yu Zhang, Ron Weiss, Heiga Zen, Yonghui Wu, Zhifeng Chen, R.J. Skerry-Ryan, Ye Jia, Andrew Rosenberg, and Bhuvana Ramabhadran. Learning to Speak Fluently in a Foreign Language: Multilingual Speech Synthesis and Cross-Language Voice Cloning. In *Proc. Interspeech 2019*, pages 2080–2084, 2019.

E. Zwicker. Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen). *The Journal of the Acoustical Society of America*, 33(2): 248–248, 1961.