End-to-End Neural based Greek Text-to-Speech Synthesis

Sisamaki Eirini

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Stylianou Yannis*

This work has been performed at the University of Crete, School of Sciences and Engineering, Computer Science Department.

University of Crete
Computer Science Department

End-to-End Neural based Greek Text-to-Speech Synthesis

Thesis submitted by

in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author:	Sisamaki Eirini
Committee approvals:	Y.Stylianou Professor, Thesis Supervisor
	G.Tziritas Professor, Committee Member
	Y.Pantazis Researcher, Committee Member
Departmental approval:	Prof4 Professor, Director of Graduate Studies

Heraklion, October 2019

End-to-End Neural based Greek Text-to-Speech Synthesis

Abstract

Text-to-speech (TTS) synthesis is the automatic conversion of written text to spoken language. TTS systems play an important role in natural human-computer interaction. Concatenative speech synthesis and statistical parametric speech synthesis were the prominent methods used for decades. In the era of Deep learning, end-to-end TTS systems have dramatically improved the quality of synthetic speech. The aim of this work was the implementation of an end-to-end neural based TTS system for the Greek Language. The neural network architecture of Tacotron-2 is used for speech synthesis directly from text. The system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to acoustic features, followed by a modified WaveNet model acting as a vocoder to synthesize time-domain waveforms from the predicted acoustic features. Developing TTS systems for any given language is a significant challenge and requires large amount of high quality acoustic recordings. Because of this, these systems are only available for the most commonly and widely spoken languages. In this work, experiments are described for various languages and databases which are freely available. A Greek database, initially created for speech recognition, has been obtained from ILSP (Institute for Language and Speech Processing). In our first experiment, only 3 hours of recorded speech in Greek have been used. Then the technique of language adaptation has been applied, using 3 hours in Greek and 18 hours in Spanish. We also have applied speaker adaptation in order to produce speech with specific speakers from our database. Our TTS system for Greek can generate good quality of speech with very natural prosody. An evaluation with a listening test by 30 volunteers gave a score in MOS (Mean Opinion Score) of 3.15 to our model and 3.82 to the original recordings.

Από-άκρη-σε-άκρη Νευρωνική σύνθεση ομιλίας από κείμενο για την Ελληνική Γλώσσα

Περίληψη

Σύνθεση ομιλίας από κείμενο (TTS) είναι η αυτόματη μετατροπή του γραπτού λόγου σε προφορικό. Τα συστήματα σύνθεσης ομιλίας από κείμενο παίζουν σημαντικό ρόλο στη διάδραση ανθρώπου-υπολογιστή. Η συνενωτική σύνθεση ομιλίας και η στατιστική παραμετρική σύνθεση ομιλίας ήταν οι μέθοδοι που εφαρμόστηκαν για δεκαετίες. Στην εποχή της Βαθιάς Μάθησης τα από-άκρη-σε-άκρη συστήματα έχουν βελτιώσει δραματικά την ποιότητα της συνθετικής ομιλίας. Ο στόχος αυτής της εργασίας είναι η υλοποίηση ενός νευρωνικού από-άκρη-σε-άκρη συστήματος σύνθεσης ομιλίας από κείμενο, για την ελληνική γλώσσα. Η αρχιτεκτονική νευρωνικού δικτύου του Tacotron-2 χρησιμοποιείται για σύνθεση ομιλίας κατευθείαν από κείμενο. Το σύστημα αποτελείται από ένα αναδρομικό από-ακολουθία-σε-ακολουθία δίκτυο πρόβλεψης χαρακτηριστικών, που αντιστοιχίζει ενσωματώσεις χαρακτήρων σε φασματογράμματα κλίμακας Μελ που ακολουθείται από ένα τροποποιημένο μοντέλο WaveNet, που λειτουργεί ως συνθεσάϊζερ ομιλίας για να συνθέσει κυματομορφές στο πεδίο του χρόνου από αυτά τα ακουστικά χαρακτηριστικά. Η ανάπτυξη συστημάτων σύνθεσης ομιλίας από κείμενο για μια δεδομένη γλώσσα είναι μια σημαντική πρόκληση και απαιτεί μεγάλη ποσότητα ηχογραφήσεων υψηλής ποιότητας. Γι' αυτό, αυτά τα συστήματα είναι διαθέσιμα μόνο για τις πιο ευρέως ομιλούμενες γλώσσες. Σε αυτή την εργασία περιγράφονται πειράματα με διάφορες γλώσσες και βάσεις δεδομένων που είναι ελεύθερα διαθέσιμες. Μια ελληνική βάση δεδομένων, αρχικά δημιουργημένη για αναγνώριση ομιλίας, μας δόθηκε από το Ινστιτούτο Επεξεργασίας Λόγου. Στο πρώτο μας πείραμα χρησιμοποιήθηκαν μόνο 3 ώρες ηχογραφήσεων στα Ελληνικά. Έπειτα, εφαρμόστηκε η τεχνική της προσαρμογής γλώσσας, χρησιμοποιώντας 3 ώρες Ελληνικά και 18 ώρες Ισπανικά. Επίσης εφαρμόσαμε την προσαρμογή ομιλητή για να παράγουμε ομιλία με συγκεκριμένους ομιλητές από τη βάση δεδομένων μας. Το σύστημά μας για τα Ελληνικά μπορεί να συνθέτει καλής ποιότητας ομιλία με πολύ φυσική προσωδία. Μια αξιολόγηση με ένα ακουστικό τεστ με 30 εθελοντές έδωσε Μέσο Βαθμό Προτίμησης 3.15 στο μοντέλο μας και 3.82 στις ηχογραφήσεις.

Acknowledgements

I would like to express my sincere gratitude to my advisor and excellent teacher Prof. Yannis Stylianou for the support, motivation, enthusiasm and the opportunity he gave me to explore Greek TTS. I feel very lucky that I met Mr. Stylianou during my studies and started getting knowledge in the field of Speech Processing, and specifically in Speech Synthesis. He also provided me with the necessary resources to run my experiments.

My sincere thanks also goes to my co-supervisor of the thesis Dr. Vassilis Tsiaras from Technical University of Crete, Chania. He spent a lot of his valuable time guiding and supporting me in all the time of research and writing of this thesis. With his immense knowledge in the fields of computer science and mathematics, has helped me in clearing doubts to my questions.

I am grateful to Prof. Georgios Tziritas (Computer science Department, University of Crete) and Dr. Yannis Pantazis (researcher at Institute of Applied and Computational Mathematics, FORTH) for accepting to be members of the committee and helping me to schedule my thesis presentation.

I would also like to acknowledge Vassilis Katsouros, Giorgos Tambouratzis and Evita Fotinea from ILSP (Institute for Language and Speech Processing). They showed their interest and trust in our work and experiments and provided us the Greek database. I also appreciate that Evita and Vassilis visited our lab and did a small test with our models.

Another sincere thanks goes to Adjunct Prof. Giorgos Kafentzis (Computer science Department, University of Crete) for his constant presence and support in the lab (Multimedia Informatics, Speech Signal Processing lab) and for his great teaching that gave inspiration to me and other students during our studies. I thank my labmates Nagaraj Adiga, Muhammed Shifas PV, Dipjyoti Paul, Dora Yakoumaki, for sharing with me time, knowledge, experience, suggestions, hints and fun.

I must express my profound gratitude to my family and especially my nephews for the encouragement and the time that we spend together. I also thank my friends and students from the Evening Vocational Highschool of Heraklion, who have shown that working and studying can be combined.

Contents

List of Tables

List of Figures

Table 1: Abbreviations.

ADAM	Adaptive Moment Estimation
ARSG	Attention-based Recurrent Sequence Generator
BLSTM	Bidirectional LSTM
BPTT	Backpropagation Through Time
CBHG	1-D convolution bank + highway network + bidirectional GRU
DNN	Deep Neural Network
GPU	Graphics Processing Unit
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
NSW	Non-Standard Word
OOM	Out of Memory
POS	Part of Speech
RNN	Recurrent Neural Network
SPSS	Statistical Parametric Speech Synthesis
STFT	Short Time Fourier Transform
TTS	Text-to-Speech

Chapter 1

Introduction

Speech Synthesis is called the artificial production of human speech. Its aim is to synthesize intelligible and natural audio which is indistinguishable from human recordings. A speech synthesizer is a computer system used for this purpose. One famous person that used a speech computer to communicate was Stephen Hawking. Speech synthesis has applications that can be used by people with a wide range of disabilities (screen readers for people with visual impairment), or people with dyslexia. Other significant applications of speech synthesis are used for entertainment productions such as games and animations and for creation of educational tools for foreign languages. Speech Synthesis combined with Speech Recognition allows interaction with mobile devices via natural language processing interfaces.

Text-To-Speech(TTS) synthesis is called the automatic conversion of written to spoken language. The input is text and the output is speech waveform [19]. A TTS system is divided into two parts. The first part is called the front-end and converts text into linguistic specifications. The second part is called the back-end and uses these specifications to generate a waveform. These two modules are conventionally constructed independently. The linguistic specification comprises whatever factors might affect the acoustic realisation of the speech sounds making up the utterance. Many tasks performed by the front end (e.g. predicting pronunciation from spelling) are quite specific to one language or one family of languages (Taylor (2009)) [42]. The text-analysis module is trained using text corpora and often includes statistical models to analyze text, e.g., the phrasing boundary, accent, and POS. The waveform generation module, on the other hand, is trained using a labeled speech database. In statistical parametric synthesis, this module includes acoustic models.

In the era of Deep Learning, end-to-end neural networks, which are the state of the art for speech recognition tasks, have become highly competitive with the conventional TTS systems. An end-to-end TTS system can be trained on <text, audio> pairs without hand-crafted feature engineering. All the modules are trained together to optimize a global performance criterion, without manual integration of separately trained modules. It allows rich conditioning on various attributes such as, speaker, language, sentiment, etc. It is more robust than a multi-component system. It has the potential for transfer learning and it can adapt to new data. It may be trained on huge amount of often noisy data found in the real world.

Neural TTS systems proposed nowadays, are: Deep Voice 1 (Arik et al., 2017a), Deep Voice 2 (Arik et al., 2017b), Deep Voice 3 (Ping et al., 2018), Tacotron (Wang et al., 2017), Tacotron 2 (Shen et al., 2018), Char2Wav (Sotelo et al., 2017), VoiceLoop (Taigman et al., 2018), and ClariNet (Ping et al., 2018) Deep Voice 1 and 2 retain the traditional TTS pipeline, which has separate grapheme-to-phoneme, phoneme duration, frequency, and waveform synthesis models. In contrast, Tacotron, Deep Voice 3, and Char2Wav employ the attention based sequence-to-sequence models (Bahdanau et al., 2015). These models depend on a traditional vocoder, or a separately trained neural vocoder to convert the predicted spectrogram to raw audio. ClariNet, which is based on Deep Voice 3, seems to be the first text-to-wave neural architecture for TTS.

Tacotron (1, 2) take characters as input and produce spectrogram frames, which are then converted to waveforms. Tacotron 1 produces linear-scale spectrograms and uses the Griffin-Lim algorithm to synthesize waveform from the predicted spectrogram. Tacotron 2 produces mel-scaled spectrograms, which are used as local conditioning to a WaveNet vocoder. In contrast to Tacotron 1, Tacotron 2 uses simpler building blocks, using vanilla LSTM and convolutional layers in the encoder and decoder instead of CBHG stacks and GRU recurrent layers. Tacotron 2 will be described in this work, and experiments with it will be presented later.

1.1 History

In the past decades, the mainstream techniques were concatenative and parametric speech synthesis. Both of them had complex pipelines, required a lot of resource and manpower. They also were language specific.

An exemplar-based speech synthesis system simply stores the speech corpus itself. The stored speech data are labelled so that appropriate parts of them can be found, extracted and then concatenated during the synthesis phase. The most prominent exemplar based technique and one of the dominant approaches to speech synthesis is Unit-Selection (Fig. 1.1). In this technique, units from a large speech database are selected according to how well they match a specification and how well they join together. The specification and the units are completely described by a feature structure, which can be any mixture of linguistic and acoustic features. The quality of output derives directly from the quality of the recordings and it appears that the larger the database, the better the coverage. Commercial systems have exploited these techniques to bring about a new level of synthetic speech. However, these techniques limit the output speech to the same style as that in the original recordings. In addition, recording large databases with variations is very difficult and costly.

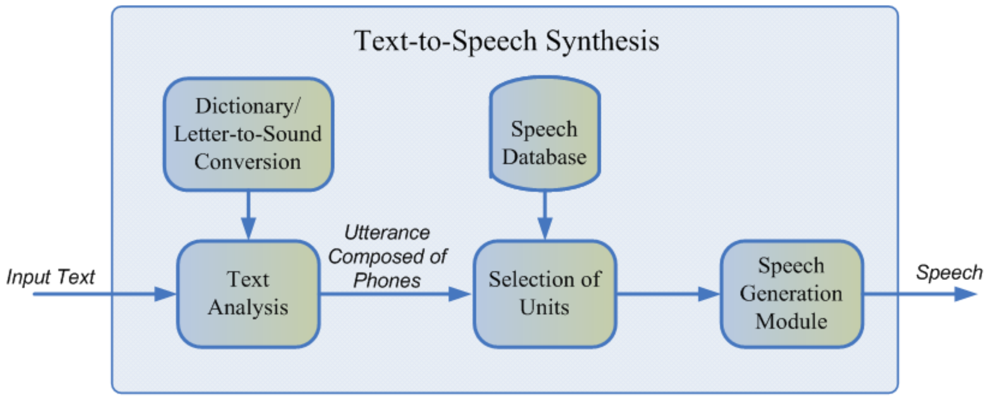


Figure 1.1: Modules of a unit selection TTS system.

A model-base speech synthesis system fits a model to the speech corpus (during the training phase) and stores this model. Due to the presence of noise and unpredictable factors in speech training data the models are usually statistical. **Statistical Parametric** speech synthesis [19] has also grown in popularity over the last years, in contrast to the selection of actual instances of speech. These models don't use stored exemplars. They describe the parameters of models using statistics (e.g. means and variances of probability density functions) which capture the distribution of parameter values found in the training data (Fig 1.2).

Most of the advantages of statistical parametric synthesis against unit-selection synthesis are related to its flexibility due to the statistical modelling process. Some of these advantages are:

- Transforming voice characteristics, speaking styles and emotions (by transforming its model parameters).
- Multilingual support (only the contextual factors to be used depend on each language).
- Small footprint compared with unit-selection (because statistics of acoustic models are stored rather than the multi-templates of the speech units). So statistical parametric speech synthesis seemed to be suitable for embedded applications.

The quality of speech produced by the initial statistical parametric systems was significantly lower than this of unit-selection systems. Three factors degrade the

quality of speech of SPSS: a) the vocoder, b) the acoustic modelling accuracy and c) the over-smoothing of the synthesized speech parameters. Researchers tried to improve all these factors. The most frustrating task was the improvement of the vocoder. After decades of research the proposed vocoders [24], [23] introduced little or no improvement compared to straight vocoder [18] which existed since the early days of SPSS [45]. On the other hand, the invention of more advanced statistical models, such as the trajectory HMMs, leaded to smoother speech trajectories and increased user opinion scores [50].
Finally, the most rewarding task was the addressing of over-smoothing. Toda applied the Global-Variance (GV) heuristic, which is a post-processing algorithm that scales the variance of the trajectories of the synthesized speech parameters to be equal to the variance of the original speech [44]. As a result, the naturalness of the synthesized speech was greatly improved and approached the quality of the unit-selection systems but did not surpassed it. Up to the begging of 2016, the best quality TTS systems were either unit-selection or hybrid systems where the global structure of the trajectories is created by statistical models and the segments of actual instances of speech were chosen from a database.

There are many possible approaches to statistical synthesis.

The Hidden Markov Models (HMMs) has been proven a powerful model of speech [51]. An important reason for this, is the availability of effective and efficient learning algorithms (Expectation-Maximization algorithm that is used for the parameter estimation) [31, 28, 32]. However, there are several assumptions in HMMs that do not apply to the properties of speech.
Disadvantages of HMM synthesis are:

- The speech has to be generated by a parametric model, so no matter how naturally the models generate parameters, the final quality is very much dependent on the model used.
- Even with the dynamic constraints, the models generate somewhat 'safe' observations and fail to generate some of the more interesting and delicate phenomena in speech.
- The assumptions such as the observations are conditionally independent given the state sequence.
- The speech statistics of each state do not change dynamically.
- The Markov assumption itself: that the probability of being in a given state at time t only depends on the state at time $t-1$, is inappropriate for speech sounds where dependencies often extend through several states.

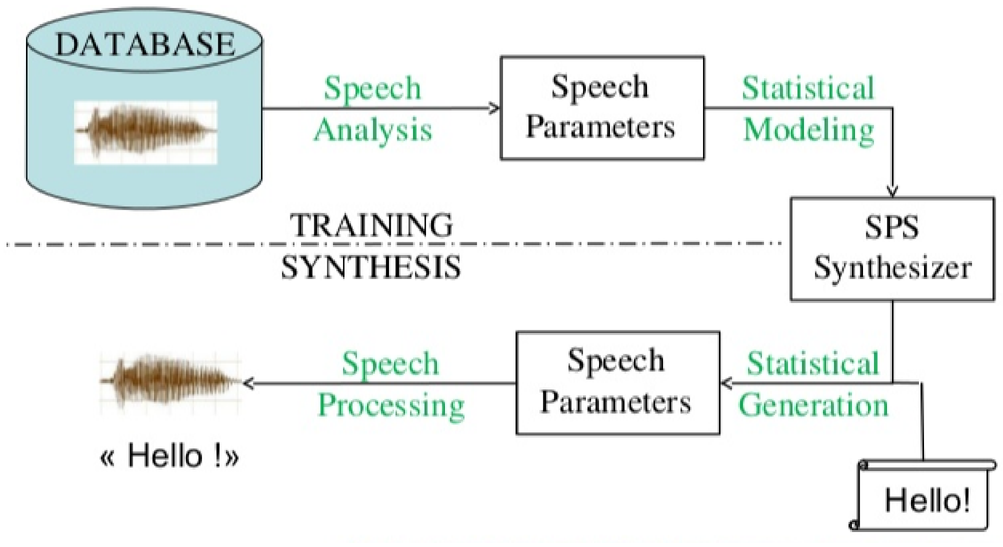


Figure 1.2: Statistical parametric speech synthesis.

Trajectory HMMs [50] can alleviate two limitations of the standard HMM, which are (i) piece-wise constant statistics within a state and (ii) conditional independence assumption of state output probabilities, without increasing the number of model parameters. In the same paper, a Viterbi-type training algorithm based on the maximum likelihood criterion was also derived. The performance of the trajectory HMM was evaluated both in speech recognition and synthesis. In a speaker-dependent continuous speech recognition experiment, the trajectory HMM achieved an error reduction over the corresponding standard HMM. Subjective listening test results showed that the introduction of the trajectory HMM improved the naturalness of synthetic speech. The formulation of the trajectory HMM is closely related to the technique for speech parameter generation from the standard HMM (Tokuda et al., 1995a, Tokuda et al., 1995b, Tokuda et al., 2000), in which the speech parameter sequence is determined so as to maximize its output probability for the standard HMM under the constraints between the static and dynamic features. While the speech parameter generation algorithm was derived to construct HMM-based speech synthesizers (Yoshimura et al., 1999) which can synthesize speech with various voice characteristics (Tamura et al., 2001, Yoshimura et al., 1997, Shichiri et al., 2002), the generation algorithm was also applied to speech recognition (Minami et al., 2002, Minami et al., 2003).
Statistical parametric speech synthesis can benefit from a stronger model of the speech signal, with perhaps a more explicit representation of the physical and linguistic specification from text.

Linear Dynamical Models (LDMs) , also known as Kalman filter models, had been proposed to explicitly capture the dynamics of speech [27, 46, 47]. LDMs are probabilistic, state space models, which explicitly model some of the dynamics of speech and introduce the continuity and context dependence needed for good quality synthesis. These models have also the property that can be trained via the EM algorithm in a maximum likelihood framework. LDMs can produce a smoother trajectory of the synthesized speech (closer to the natural).

Chapter 2

Text Normalization

2.1 The 'front-end'

The two stages of a common TTS system are Text Analysis (text into intermediate representation) and Waveform synthesis (from the intermediate representation into waveform). These stages are known as the front-end and the back-end respectively.
The 'front-end' of the TTS (From input text to linguistic specification): Text Processing sees the text as the input to the synthesizer and tries to rewrite any "non-standard" text as proper "linguistic" text. It takes arbitrary text and performs the task of classifying the written signal with respect to its semiotic type (natural language or other) decoding the written signal into an unambiguous, structured, representation and in the case of non-natural language, verbalising this representation to generate words. The tasks that can be included in the 'front-end' [25] are the following:

1. Pre-processing: possible identification of text genre, character encoding issues, possible multi-lingual issues.
2. Sentence splitting: segmentation of the Document into a list of sentences.
3. Tokenisation : segmentation of each sentence into a number of tokens, possible processing of XML.
4. Text-Analysis:
   (a) Semiotic classification : classification of each token as one of the semiotic classes of natural language, abbreviation quantity, date, time etc.
   (b) Decoding/parsing : finding the underlying identities of tokens using a decoder or parser that is specific to the semiotic class.
   (c) Verbalisation : Conversion of non-natural language semiotic classes into words which can be spoken.

5. Homograph resolution: Determination of the correct underlying word for any ambiguous natural language token.
6. Parsing: Assigning a syntactic structure to the sentence.
7. Prosody prediction: Attempting to predict a prosodic form for each utterance from the text.

Text Decoding (finding the words from the text) is a process of resolving ambiguity. Take a tokenised sentence and determine the best sequence of words. There are many types of linguistic ambiguity: word identity, grammatical and semantic. In **TTS** we need only to concentrate on the type of ambiguity which affects the **actual sound produced**.
Text Normalization removes capital letters, spells out numbers or separates punctuation. In addition to ordinary words and names, real text contains non-standard words (NSWs), including numbers, abbreviations, dates, currency amounts and acronyms [37]. Typically, one cannot find NSWs in a dictionary, nor can one find their pronunciation by an application of ordinary letter-to-sound rules. Non-standard words also have a greater propensity than ordinary words to be ambiguous with respect to their interpretation or pronunciation. In many applications, it is desirable to normalize text by replacing the NSWs with the contextually appropriate ordinary word or sequence of words. Typical technology for text normalization involves sets of ad hoc rules tuned to handle one or two genres of text (often newspaper-style text) with the expected result that the techniques do not usually generalize well to new domains. In general texts may include non-standard word sequences from a variety of different semiotic classes (Taylor, 2009 [25]):
NON-NATURAL LANGUAGE TEXT SEMIOTIC CLASSES

- cardinal numbers
- ordinal numbers
- telephone numbers
- years
- dates
- money
- percentages
- measures
- emails
- urls
- computer programs

- addresses
- real estate

2.1.1 Examples where we see the need of text normalization

1. date (day, month ,year)
   (a) 10 December 1967
   (b) December 10 1967
   (c) 10th of December 67
   (d) 10/12/1967
   (e) 10/12/67
   (f) 12/10/67

   In each of these cases, the date is exactly the same and we can pronounce it in various ways.

2. abbreviations: In text, abbreviations are often used, but conventionally they are read out fully. Even simple abbreviations can be ambiguous:
   - Dr. Livingston vs. Livingston Dr.
   - St. James vs. James St.
   - V can be a roman numeral or Volts
   - 100m could be 100 million or 100 metres or 100 miles,

   The system must recognise abbreviations then expand them. Simple rules can be used to expand most of these into words, although writing such rules is pretty tedious, and often language dependent(Simon King, [20]). A common implementation of rules used to recognise abbreviations, for example, is as regular expressions (or their equivalent finite state machine).

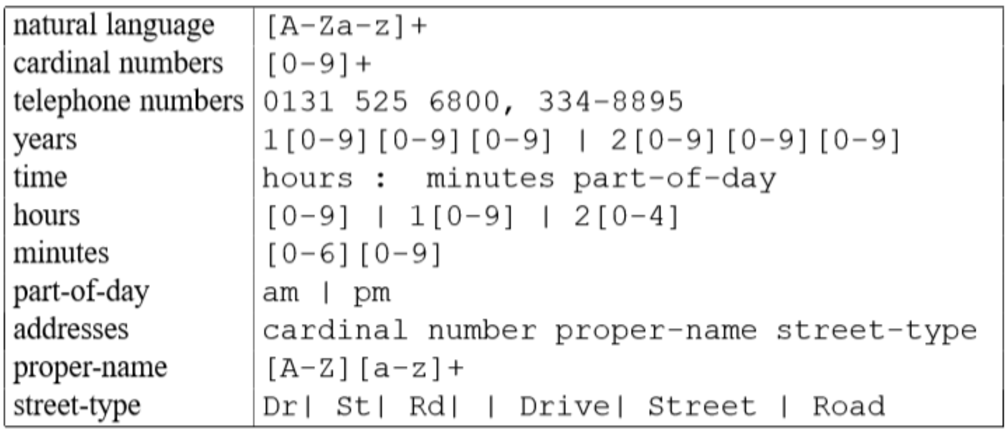
3. NUMBERS
   The interpretation of numbers is context sensitive.
   - 2.16pm
   - 15:22
   - 2.1
   - 20/11/05
   - The 2nd
   - $100bn
   - 99p
   - 0131 651 3174

2.1.2 Grapheme and Phoneme form models

Other well known models in TTS are the Grapheme and Phoneme form models [25]. First a grapheme form of the text input is found and this is then converted to a phoneme form for synthesis. Words are not central to the representation. This approach is particularly attractive in languages where the grapheme-phoneme correspondence is relatively direct; finding the graphemes often means the phonemes and hence pronunciation can accurately be found.

2.2 Text normalization for an end-to-end system

With an end-to-end system [36], [26] we will need only a small part of the above tasks, simple Text Decoding (Word Tokenization and Normalization). Word tokenisation and normalization are generally done by cascades of simple regular expressions substitutions or finite automata (Kleene, 1951). Though recent advancements in learned text normalization (Sproat and Jaitly, 2016, [38]) may render this unnecessary in the future. There are also machine learning approaches to text normalization (DeepNorm [30]). We will use regular expressions to work on our task. **Regular expression (RE)** is a language for specifying text search strings [15].



natural language	`[A-Za-z]+`
cardinal numbers	`[0-9]+`
telephone numbers	`0131 525 6800, 334-8895`
years	`1[0-9][0-9][0-9] | 2[0-9][0-9][0-9]`
time	`hours :  minutes part-of-day`
hours	`[0-9] | 1[0-9] | 2[0-4]`
minutes	`[0-6][0-9]`
part-of-day	`am | pm`
addresses	`cardinal number proper-name street-type`
proper-name	`[A-Z][a-z]+`
street-type	`Dr| St| Rd| | Drive| Street | Road`

Figure 2.1: Regular Expressions

Chapter 3

Background

Many basic concepts needed to understand Tacotron-2, will be presented briefly in this chapter: RNNs, LSTMs, Sequence to Sequence Models, Attention, Dropout, Zoneout, Batch Normalization.

3.1 RNNs

A recurrent neural network (RNN) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence (Fig 3.1). This allows it to exhibit temporal dynamic behavior for a time series. Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. RNNs are relatively old, like many other deep learning algorithms. They were initially created in the 1980s, but can only show their real potential since a few years, because of the increase in available computational power, the massive amounts of data that we have nowadays and the invention of LSTM in the 1990s. Because of their internal memory, RNNs are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next. This is the reason why they are the preferred algorithm for data like time series, speech, text, financial data, audio, video, weather, etc. In a RNN, the information cycles through a loop. When it

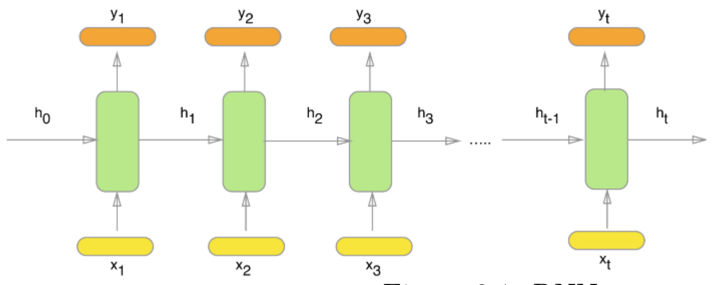


Figure 3.1: RNNs

makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously. All RNNs have infinite memory. However, the information decays exponential with time. The LSTM cell (Fig. 3.4) may keep the information for longer periods than the basic RNN cell.

3.1.1 RNN basics

A recurrent neural network (RNN) consists of a hidden state h and an optional output y, and it operates on a variable length sequence $x = (x_1, ..., x_T)$. At each time step t, the hidden state h_t of the RNN is updated by

$$h_t = f(h_{t-1}, x_t)$$

, where f is a non-linear activation function. The function f may be as simple as an element wise logistic sigmoid function and as complex as a long short-term memory (LSTM) unit (Hochreiter and Schmidhuber, 1997).

An RNN can learn a probability distribution over a sequence by being trained to predict the next symbol in a sequence. In that case, the output at each time step t is the conditional distribution

$$p(x_t | x_{t-1}, ..., x_1)$$

For example, a multinomial distribution (1-of-K coding) can be output using a softmax activation function

$$p(x_{t,j} = 1 \quad | x_{t-1}, ..., x_1) = \frac{exp(w_j \cdot h_t)}{\sum_{n=1}^{K} exp(w_n \cdot h_t)}$$

for all possible symbols $j = 1, ..., K$, where w_j are the rows of a weight matrix W. By combining these probabilities, we can compute the probability of the sequence x using

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

From this learned distribution, it is straight forward to sample a new sequence by iteratively sampling a symbol at each time step.

3.1.2 Backpropagation Through Time

Backpropagation Through Time (BPTT) is basically Backpropagation on an unrolled Recurrent Neural Network. Unrolling is a visualization and conceptual tool, which helps you to understand what's going on within the network (Fig. 3.3). A RNN can be viewed as a sequence of Neural Networks that are trained one after another with backpropagation. If Back-propagation Through Time is used, it is required to do the conceptualization of unrolling, since the error of a given time-step depends on the previous time-step.

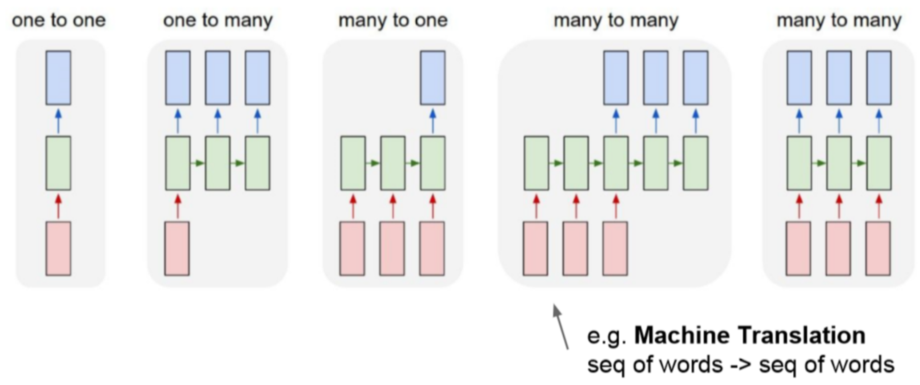


Figure 3.2: RNNs process Sequences.

Within BPTT the error is back-propagated from the last to the first time-step, while unrolling all the time-steps. This allows calculating the error for each time-step, which allows updating the weights. Note that BPTT can be computationally expensive when there is a high number of time-steps.

Truncated Backpropagation Through Time (truncated BPTT) [40] is a widespread method for learning recurrent computational graphs. Truncated BPTT keeps the computational benefits of Backpropagation Through Time (BPTT) while relieving the need for a complete backtrack through the whole data sequence at every step. However, truncation favors short-term dependencies: the gradient estimate of truncated BPTT is biased, so that it does not benefit from the convergence guarantees from stochastic gradient theory. Truncated BPTT heuristically solves BPTT deficiencies by chopping the initial sequence into evenly sized subsequences. Truncated BPTT truncates gradient flows between contiguous subsequences, but maintains the recurrent hidden state of the network. Truncation biases gradients, removing any theoretical convergence guarantee. Intuitively, truncated BPTT has trouble learning dependencies above the range of truncation.

The exploding gradient problem is an issue found in training artificial neural networks with gradient-based learning methods and backpropagation. An error gradient is the direction and magnitude calculated during the training of a neural network that is used to update the network weights in the right direction and by the right amount. When the magnitudes of the gradients accumulate, like in RNNs, the network becomes unstable and it is unable to learn from training data. At an extreme, the values of weights can become so large as to overflow and result in NaN values. Gradient clipping, weight regularization and weight normalization (Salimans and Kingma) [34] and truncated BPTT may reduce this problem. However, the best practice to reduce the exploding gradient problem is to use gated RNN Cells, like LSTM and GRU.

Related to exploding gradient problem is the vanishing gradient problem, where the gradient will be vanishingly small, effectively preventing the weight from changing its value. In the worst case, this may completely stop the neural network from further training. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM cell by Sepp Hochreiter and Juergen Schmidhuber. Later, the GRU cell also proved efficient in training RNNs without vanishing gradients.

Basic RNNs are not good at capturing long term dependencies. This is because during backpropagation, gradients from an output y would have a hard time propagating back to affect the weights of earlier layers. So, in basic RNNs, the output is highly affected by inputs closer to that word.

3.1.3 LSTMs

Long Short-Term Memory (LSTM) networks are RNNs capable of learning long-term dependencies [Hochreiter and Schmidhuber, 1997]. A memory cell using logistic and linear units with multiplicative interactions. Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

The units of an LSTM are used as building units for the layers of a RNN, which is then often called an LSTM network.

LSTMs enable RNNs to remember their inputs over a long period of time. This is because LSTMs contain their information in a memory, that is much like the memory of a computer because the LSTM can read, write and delete information from its memory. This memory can be seen as a gated cell, where gated means that the cell decides whether or not to store or delete information (e.g if it opens the gates or not), based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm. This simply means that it learns over time which information is important and which not.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate). You can see an illustration of a RNN with its three gates in figure 3.4. The gates in a LSTM are analog, in the form of sigmoids, meaning

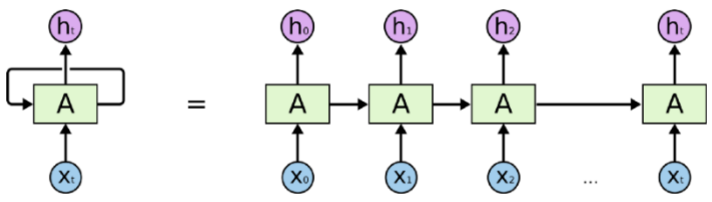


Figure 3.3: RNN unrolled

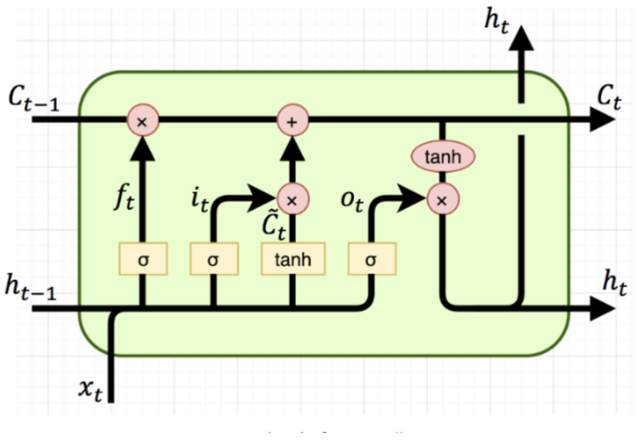


Figure 3.4: LSTM cell

that they range from 0 to 1. The fact that they are analog, enables them to do backpropagation with it.

The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.

Variants of LSTM GRU (Gated Recurrent Unit) [Cho et al., 2014]:

- Combine the forget and input gates into a single update gate.
- Merge the memory cell and the hidden state.

Bidirectional LSTMs are an extension of traditional LSTMs that can incorporate temporal dynamics and improve model performance on sequence classification problems, where all timesteps of the input sequence are available (Fig. 3.5). BLSTMs are trained simultaneously in positive and negative time direction. This can provide additional context to the network both from past (forward) and future (backwards) states simultaneously. Bidirectional LSTMs were introduced by Schuster and Paliwal, 1997 [35] and a detailed discussion of the various architectures was presented in the paper [12] by Graves and Schmidhuber, 2005. They have applications in domains such as speech recognition, because there is evidence that the context of the whole utterance is used to interpret what has being said. BLSTMs are also well suited for other speech processing tasks, where context is vitally important.

3.2 Sequence-to-Sequence models

Deep Neural Networks (DNNs)[39] are extremely powerful machine learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition. DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps.

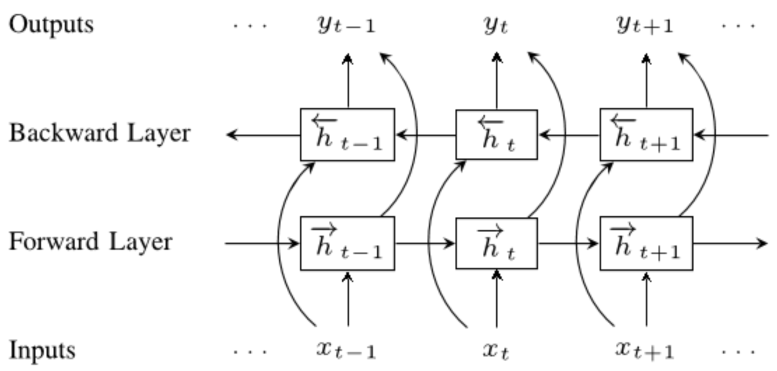


Figure 3.5: Bidirectional LSTM

Furthermore, large DNNs can be trained with supervised backpropagation whenever the labeled training set has enough information to specify the networks parameters. Thus, if there exists a parameter setting of a large DNN that achieves good results (for example, because humans can solve the task very rapidly), supervised backpropagation will find these parameters and solve the problem. Despite their flexibility and power, DNNs can only be applied to problems whose inputs and targets can be sensibly encoded with vectors of fixed dimensionality. It is a significant limitation, since many important problems are best expressed with sequences whose lengths are not known a-priori. For example, speech recognition and machine translation are sequential problems. Likewise, question answering can also be seen as mapping a sequence of words representing the question to a sequence of words representing the answer. It is therefore clear that a domain-independent method that learns to map sequences to sequences would be useful.

Sequences pose a challenge for DNNs because they require that the dimensionality of the inputs and outputs is known and fixed. It is shown that a straightforward application of the Long Short-Term Memory (LSTM) architecture can solve general sequence to sequence problems. The idea is to use one LSTM to read the input sequence, one timestep at a time, to obtain large fixed-dimensional vector representation, and then to use another LSTM to extract the output sequence from that vector (Fig. 3.6). The second LSTM is essentially a recurrent neural network language model except that it is conditioned on the input sequence.

There have been a number of related attempts to address the general sequence to sequence learning problem with neural networks. Kalchbrenner and Blunsom were the first to map the entire input sentence to vector (their approach is very similar to Cho et al). Graves introduced a novel differentiable attention mechanism that allows neural networks to focus on different parts of their input, and an

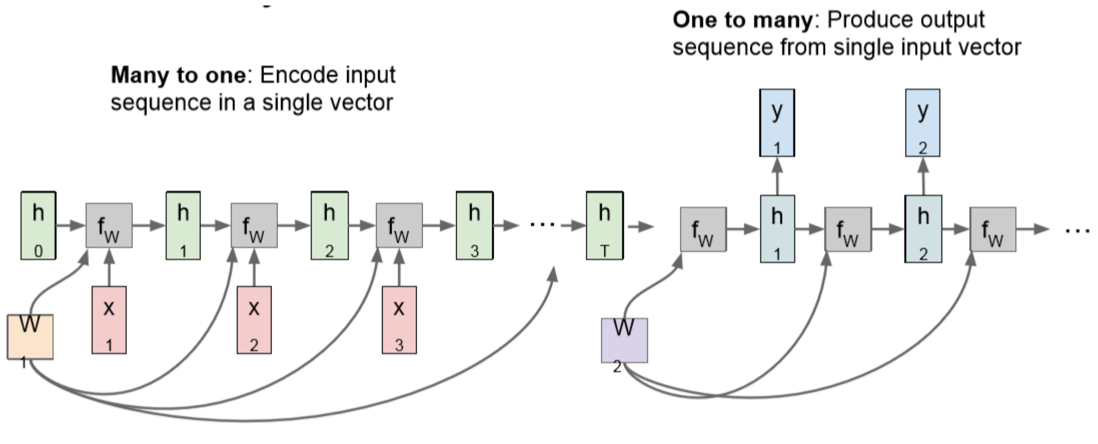

Many to one: Encode input sequence in a single vector
One to many: Produce output sequence from single input vector
h 0
h 1
h 2
h 3
h T
f W
x 1
x 2
x 3
W 1
W 2
y 1
y 2
...

Figure 3.6: Sequence-to-sequence models

elegant variant of this idea was successfully applied to machine translation by Bahdanau. The Connectionist Sequence Classification is another popular technique for mapping sequences to sequences with neural networks, although it assumes a monotonic alignment between the inputs and the outputs. The Sequence to sequence models have enjoyed great success in a variety of tasks such as machine translation, speech recognition, text summarization and Text-To-Speech Synthesis. Sequence-to-sequence models with attention is the central idea behind Tacotron that we will describe later. In machine translation a single neural network takes as input a source sentence X and generates its translation Y (let $X = x_1, x_2, ..., x_{T_x}$ and $Y = y_1, y_2, ..., y_{T_y}$ be two variable length sequences, where x_t and y_t are source and target symbols).
During training, the system learns the conditional probability

$$P(y_1, ..., y_{T_y}/x_1, x_2, ..., x_{T_x})$$

During generation, given a source sequence X the system samples Y according to the above probability. The neural machine translation models have three components: an encoder, a decoder and an attention mechanism. A basic architecture has two components: an encoder and a decoder (The encoder is an RNN and the decoder is another RNN which is trained to generate the output sequence by predicting the next symbol y_t given the hidden state s_t).

3.2.1 RNN Encoder - Decoder

In the Encoder - Decoder framework, an encoder is used to encode a variable-length sequence into a fixed-length vector representation and a decoder is used to decode a given fixed-length vector representation back into a variable-length sequence [8] (Fig. 3.7). From a probabilistic perspective, this model is a general method to learn the conditional distribution over a variable-length sequence conditioned on yet another variable-length sequence $X = (x_1, ..., x_{Tx})$. The input and output

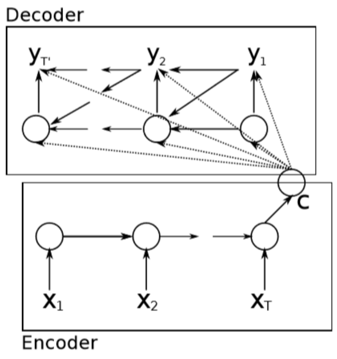


Figure 3.7: RNN Encoder - Decoder proposed by Cho

sequences may have different lengths. The most common approach is to use an RNN that reads each symbol of an input sequence X sequentially. As it reads each symbol, the hidden state of the RNN changes according to equation 3.1. After reading the end of the sequence (marked by an end-of-sequence symbol), the hidden state of the RNN is a summary c of the whole input sequence. The decoder of the proposed model is another RNN which is trained to generate the output sequence by predicting the next symbol y_t given the hidden state h_{t-1}. However, both y_t and h_{t-1} are also conditioned on y_{t-1} and on the summary c of the input sequence. Hence, the hidden state of the decoder at time t is computed by,

$$h_t = f(x_t, h_{t-1}) \tag{3.1}$$

and

$$c = q(h_1, ..., h_{Tx})$$

where $h_t \in R^n$ is a hidden state at time t, and c is a vector generated from the sequence of the hidden states. f and q are some non linear functions. Sutskever et al.(2014) used an LSTM as f and $q(h_1, h_2, ..., h_T) = h_T$ for instance.

RNN Encoder Decoder networks proposed by various researchers usually differ in terms of which RNN architectures are used for the decoder and how the encoder computes the source sentence representations.

3.2.2 Alignment mechanism

In existing encoder - decoder frameworks there is a potential bottleneck: the fixed-length representation. In the context of Neural Machine Translation [7] Bahdanau st al. (2015) has successfully applied an attentional mechanism to jointly align and translate words. The most important distinguishing feature of this approach from the basic encoder - decoder is that it does not attempt to encode a whole input sentence into a single fixed-length vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively while decoding the translation (Fig. 3.8). This frees a neural translation model from having to squash all the information of a source sentence, regardless of its length, into a fixed - length vector. This allows a model to cope better with long sentences.

For each generated word y_t in the translation, soft-searches for a set of positions $(1, ..., T)$ in a source sentence $x = (x_1, ..., x_T)$ where the most relevant information is concentrated. The predicted target word y_t is based on the context vectors

$$c_t = \sum_{j=1}^{T_x} \alpha_{tj} \cdot h_j$$

associated with these source positions and all the previous generated target words s_{t-1}, y_{t-1}. The context vector c_i depends on a sequence of annotations $(h_1, ..., h_{Tx})$ to which an encoder maps the input sentence (the encoder is a Bidirectional RNN). Each annotation summarizes the preceding words and the following words.

The Decoder has to do the following steps:

- Compute alignment a_{ij} (How well the inputs around position j and the output at position i match).

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=i}^{T_x} \exp(e_{ik})}$$

- Use simple feed-forward NN to compute e_{ij} based on s_{i-1} and h_j.

$$e_{ij} = v^T \cdot \tanh(W s_{i-1} + V h_j)$$

  where v, W, V are trainable weights.

- Compute context c_i

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} \cdot h_j$$

- Compute new decoder state s_i.

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

- Generate new output y_i

$$\text{argmax}\, p(y_i / y_1, ..., y_{i-1}, x) = g(y_{i-1}, s_i, c_i)$$

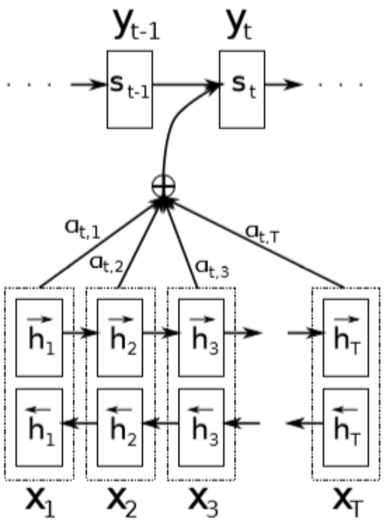


Figure 3.8: Alignment mechanism proposed by Bahdanau

The alignment model α is parametrized as a feed forward neural network which is jointly trained with all the other components of the proposed system. Note that unlike in traditional machine translation, the alignment is not considered to be a latent variable. Instead, the alignment model directly computes a soft alignment, which allows the gradient of the cost function to be back-propagated through. This gradient can be used to train the alignment model as well as the whole translation model jointly. The approach of taking a weighted sum of all the annotations can be viewed as computing an expected annotation, where the expectation is over possible alignments. Let α_{ij} be a probability that the target word y_i is aligned to, or translated from, a source word x_j. Then, the i^{th} context vector c_i is the expected annotation over all the annotations with probabilities α_{ij}. The probability α_{ij}, or its associated energy e_{ij}, reflects the importance of the annotation h_j with respect to the previous hidden state s_{i-1} in deciding the next state s_i and generating y_i. Intuitively, this implements a mechanism of attention in the decoder. The decoder decides parts of the source sentence to pay attention to. By letting the decoder have an attention mechanism, we relieve the encoder from the burden of having to encode all information in the source sentence into a fixed length vector. With this new approach the information can be spread throughout the sequence of annotations, which can be selectively retrieved by the decoder accordingly.

3.3 Attention

Informally, a neural attention mechanism equips a neural network with the ability to focus on a subset of its inputs (or features): it selects specific inputs.

We can talk about soft attention, which multiplies features with a (soft) mask of values between zero and one, or hard attention when those values are constrained to be exactly zero or one. Neural network is a function approximator. Its ability to approximate different classes of functions depends on its architecture.

A typical neural net is implemented as a chain of matrix multiplications and element-wise non-linearities, where elements of the input or feature vectors interact with each other only by addition.
Attention mechanisms compute a mask which is used to multiply features. This seemingly innocent extension has profound implications: 'suddenly', the space of functions that can be well approximated by a neural net, is vastly expanded, making entirely new use-cases possible. The intuition is the following: the theory says that neural networks are universal function approximators and can approximate an arbitrary function to arbitrary precision, but only in the limit of an infinite number of hidden units. In any practical setting, that is not the case: we are limited by the number of hidden units we can use. Consider the following example: we would like to approximate the product of N inputs (where N is a large integer). A feed forward neural network can do it only by simulating multiplications with (many) additions (plus non-linearities), and thus it requires a lot of complex computations. If we introduce multiplicative interactions, it becomes simple and compact.

3.3.1 Encoder - Decoder - Attention for Machine Translation

In the encoder-decoder architecture for machine translation, when we get very long sentences as input, it becomes very hard for the model to memorize the entire sentence. What attention models do is they take small samples from the long sentence and translate them, then take another sample and translate them, and so on. Attention summarizes the encoder, focusing on specific parts/words.

Attention Formalization. Attention computes the affinity between the decoder state and all encoder states. There are many affinity computation methods, but they 're all like a dot product.
Let there are n encoder states. The affinity between encoder state i and the decoder state is a_i. The encoder states are $h_{1:n}$, and the decoder state is s_{t-1}.
Let

$$\alpha_i = f(h_i, s_{t-1}) = h_i T s_{t-1}$$

, let weights

$$a = softmax(a)$$

, let the context

$$c = \sum_{i=1:n} h_i a_i$$

(Note that this is a weighted average.)
Attention is used at prediction as extra information in the final prediction. The

only difference is that the final prediction uses **the context vector concatenated to the decoder state** to make the prediction.

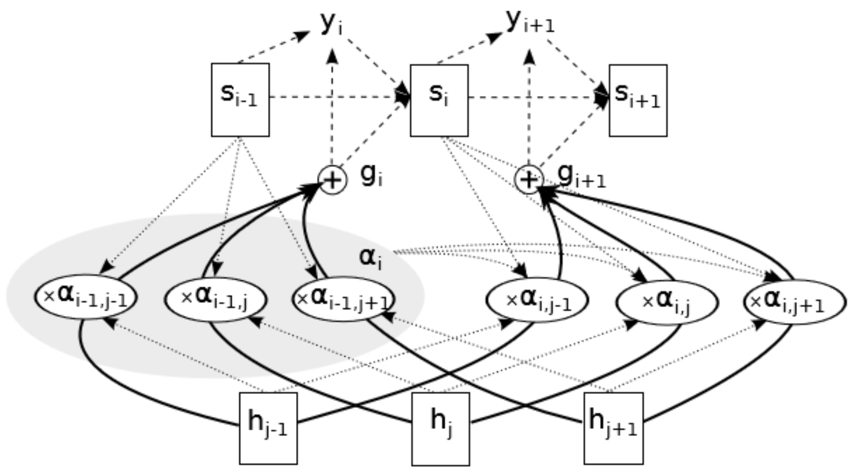


Figure 3.9: Chorowsky: Two steps of the proposed Attention-based Recurrent Sequence Generator (ARSG) with a hybrid attention mechanism (computing α based on both content (h) and location (previous α) information).

3.3.2 Attention based recurrent sequence generator

Attention-based recurrent networks have been successfully applied to a wide variety of tasks, such as handwriting synthesis, machine translation, image caption generation and visual object classification. Such models iteratively process their input by selecting relevant content at every step. This basic idea significantly extends the applicability range of end-to-end training methods, for instance, making it possible to construct networks with external memory.

An attention-based recurrent sequence generator (ARSG) is a recurrent neural network that stochastically generates an output sequence $(y_1, ..., y_T)$ from an input X [9]. In practice, X is often processed by an encoder which outputs a sequential input representation $h = (h_1, ..., h_L)$ more suitable for the attention mechanism to work with. In the context of speech recognition task, the output Y is a sequence of phonemes, and the input $X = (x_1, ..., x_{L'})$ is a sequence of feature vectors. Each feature vector is extracted from a small overlapping window of audio frames. The encoder is implemented as a deep bidirectional recurrent network (BiRNN), to form a sequential representation h of length $L = L'$. At the i-th step an ARSG generates an output y_i by focusing on the relevant elements of h:

$$\alpha_i = Attend(s_{i-1}, \alpha_{i-1}, h) \tag{3.2}$$

$$g_i = \sum_{j=1}^{L} \alpha_{i,j} h_j \tag{3.3}$$

$$y_i \sim Generate(s_{i-1}, g_i) \tag{3.4}$$

where s_{i-1} is the $(i-1)$-th state of the recurrent neural network to which we refer as the generator, $\alpha_i \in R^L$ is a vector of the attention weights, also often called the alignment. Also α_i is called a glimpse. The step is completed by computing a new generator state:

$$s_i = Recurrency(s_{i-1}, g_i, y_i) \tag{3.5}$$

Long short-term memory units (LSTM) and gated recurrent units (GRU) are typically used as a recurrent activation, to which we refer as a recurrency. The process is graphically illustrated in figure 3.9.

There are location-based, content-based and hybrid attention mechanisms. Attend in Eq. 3.2 describes the most generic, hybrid attention. If the term α_{i-1} is dropped from Attend arguments, i.e.,

$$\alpha_i = Attend(s_{i-1}, h) \tag{3.6}$$

, we call it content-based (see [7]). In this case, Attend is often implemented by scoring each element in h separately and normalizing the scores:

$$e_{i,j} = Score(s_{i-1}, h_j) \tag{3.7}$$

$$\alpha_{i,j} = \frac{\exp(e_{i,j})}{\sum_{j=1}^{L} \exp(e_{i,j})} \tag{3.8}$$

The main limitation of such scheme is that identical or very similar elements of h are scored equally regardless of their position in the sequence. This is the issue of 'similar speech fragments'. Often this issue is partially alleviated by an encoder such as e.g. a BiRNN or a deep convolutional network that encode contextual information into every element of h. However, capacity of h elements is always limited, and thus disambiguation by context is only possible to a limited extent. Alternatively, a location-based attention mechanism computes the alignment from the generator state and the previous alignment only such that

$$\alpha_i = Attend(s_{i-1}, \alpha_{i-1}). \tag{3.9}$$

For instance, Graves used the location-based attention mechanism using a Gaussian mixture model in his handwriting synthesis model. In the case of speech recognition, this type of location-based attention mechanism would have to predict the distance between consequent phonemes using s_{i-1} only, which we expect to be hard due to large variance of this quantity.

For these limitations associated with both content-based and location-based mechanisms, a hybrid attention mechanism seems a natural candidate for speech recognition. Informally, It is needed an attention model that uses the previous alignment α_{i-1} to select a short list of elements from h, from which the content-based attention, in Eqs. 3.6 and 3.7, will select the relevant ones without confusion.

Proposed Model: ARSG with Convolutional Features
Start from the ARSG-based model with the content-based attention mechanism, this model can be described by Eqs. 3.6 and 3.7, where

$$e_{i,j} = w^T \cdot tanh(W \cdot s_{i-1} + V \cdot h_j + b) \tag{3.10}$$

w and b are vectors, W and V are matrices. This content-based attention mechanism of the original model is extended to be location-aware by making it take into account the alignment produced at the previous step. First, k vectors are extracted $f_{i,j} \in R^k$ for every position j of the previous alignment α_{i-1} by convolving it with a matrix $F \in R^{k \times r}$:

$$f_i = F * \alpha_{i-1} \tag{3.11}$$

These additional vectors $f_{i,j}$ are then used by the scoring mechanism $e_{i,j}$:

$$e_{i,j} = w^T \cdot tanh(W \cdot s_{i-1} + V \cdot h_j + U \cdot f_{i,j} + b) \tag{3.12}$$

where U is also a matrix.

3.4 Character Embedding

Character Embedding is a brilliant design for solving lots of text classification problems. Difference between Character Embedding and Word Embedding is that Character Embedding can build any word as long as those characters are included.

3.5 Techniques for improvement in Neural Networks

Two common techniques used for regularization in a Neural Network are dropout and Zoneout. Batch Normalization also is used for better accuracy and speed.

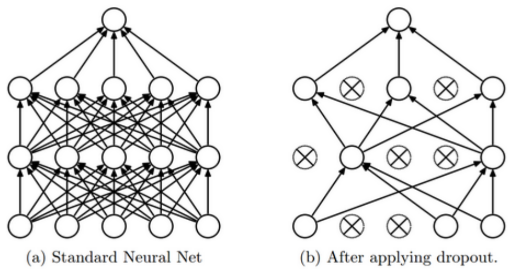


Figure 3.10: Dropout.

- **Dropout:** The term dropout refers to dropping out units (both hidden and visible) in a neural network. At each training stage, individual nodes are either dropped out of the net with probability $1-p$ or kept with probability p, so that a reduced network is left. Incoming and outgoing edges to a dropped-out node are also removed (Fig 3.10). Dropout is a way of regularization (prevents over-fitting).
- **Zoneout:** is like dropout, but uses identity masks instead of zero masks. At each timestep, zoneout stochastically forces some hidden units to maintain their previous values. Like dropout, zoneout uses random noise to train a pseudo-ensemble, improving generalization. But by preserving instead of dropping hidden units, gradient information and state information are more readily propagated through time, as in feed-forward stochastic depth networks (Fig. 3.11).

3.5.1 Batch Normalization

Batch Normalization is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem. Usually in order to train a neural network, we do some preprocessing to the input data. For example, we could normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). There are many reasons for that preprocessing, some of them being:

- preventing the early saturation of non-linear activation functions like the sigmoid function,
- assuring that all input data is in the same range of values, etc.

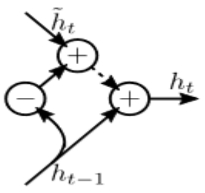


Figure 3.11: Zoneout.

But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt itself to a new distribution in every training step. This problem is known as internal covariate shift. So the solution is to force the input of every layer to have approximately the same distribution in every training step.
Batch Normalization must be used before the activation layer.

Chapter 4

Tacotron-2

Tacotron 2 [36] is a neural network architecture for speech synthesis directly from text. The system is composed of a recurrent sequence-to-sequence feature prediction network that maps character embeddings to mel-scale spectrograms, followed by a modified WaveNet model acting as a vocoder to synthesize time-domain waveforms from those spectrograms. The resulting system synthesizes speech with Tacotron-level prosody and WaveNet-level audio quality. This system can be trained directly from data without relying on complex feature engineering, and achieves state-of-the-art sound quality close to that of natural human speech.

4.1 Model Architecture

The proposed system consists of two components, shown in figure 4.1:

1. a recurrent sequence-to-sequence feature prediction network with attention which predicts a sequence of mel spectrogram frames from an input character sequence, and

2. a modified version of WaveNet which generates time-domain waveform samples conditioned on the predicted mel-spectrogram frames.

4.2 Intermediate feature representation

A low-level acoustic representation is chosen for this work: mel frequency spectrograms, to bridge the two components. Using a representation that is easily computed from time-domain waveforms allows to train the two components separately. This representation is also smoother than waveform samples and is easier to train using a squared error loss because it is invariant to phase within each frame. A mel-frequency spectrogram is related to the linear-frequency spectrogram, i.e. the short-time Fourier transform (STFT) magnitude. It is obtained by applying a non linear transform to the frequency axis of the STFT, inspired by measured responses from the human auditory system, and summarizes the frequency content with fewer dimensions. Using such an auditory frequency scale has the effect of emphasizing details in lower frequencies, which are critical to speech intelligibility, while de-emphasizing high frequency details, which are dominated by fricatives and other noise bursts and generally do not need to be modeled with high fidelity. Because of these properties, features derived from the mel scale have been used as an underlying representation for speech recognition for many decades.

While linear spectrograms discard phase information (and are therefore lossy), algorithms such as Griffin-Lim are capable of estimating this discarded information, which enables time-domain conversion via the inverse short-time Fourier transform. Mel spectrograms discard even more information, presenting a challenging inverse problem. However, in comparison to the linguistic and acoustic features used in WaveNet, the mel spectrogram is a simpler, lower level acoustic representation of audio signals. It should therefore be straightforward for a similar WaveNet model conditioned on mel spectrograms to generate audio, essentially as a neural vocoder. Indeed, we will show that it is possible to generate high quality audio from mel spectrograms using a modified WaveNet architecture.

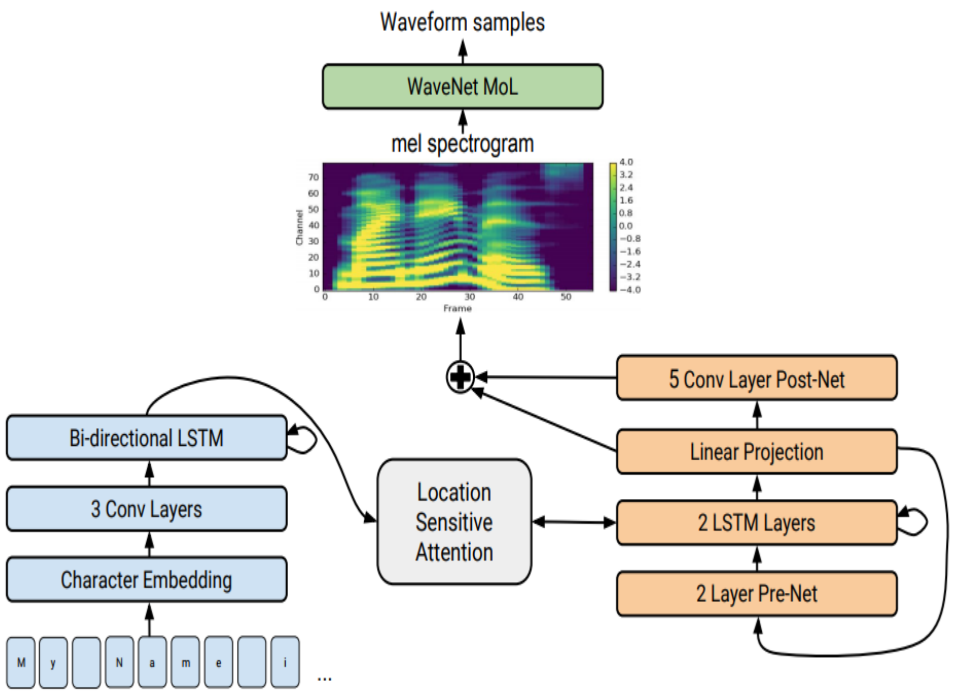


Figure 4.1: Tacotron-2.

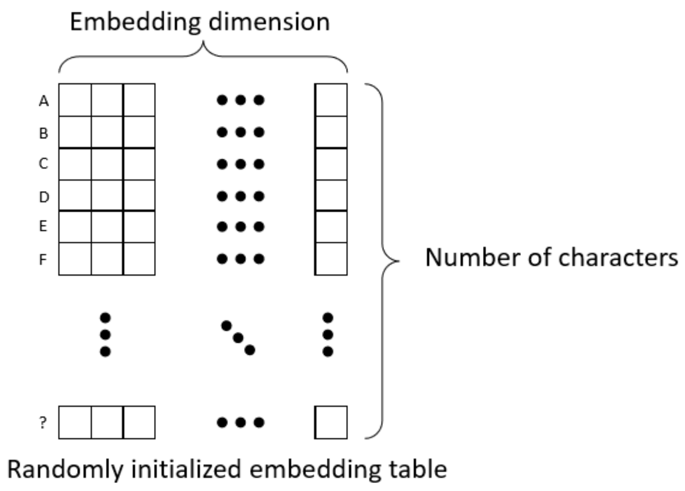


Figure 4.2: Character Embeddings

4.3 Encoder - Character Embeddings

Input characters are represented using a learned 512-dimensional character embedding. See figure 4.2.

4.3.1 Encoder

The output of the character embedding layer, is passed through a stack of 3 convolutional layers each containing 512 filters with length 5 (fig. 4.3). The output of each convolutional layer is batch normalized and then ReLU activations are applied. The output of the final convolutional layer is passed into a single bi-directional LSTM layer containing 512 units (256 in each direction). The forward and backward results are concatenated to generate the encoded features.

4.4 Attention and Decoder

The decoder has a recurrent architecture, that is, at each subsequent step, the output (one frame of the spectrogram) from the previous step is used. Another important element of the system is the mechanism of soft Attention. The idea of attention is to find what part of the encoder data should be used at the current decoder step. The encoder output is consumed by an attention network which summarizes the full encoded sequence as a fixed-length context vector for each decoder output step (Fig. 4.5, 4.4). Tacotron uses the location-sensitive attention proposed by Chorowski [9], which extends the additive attention mechanism of Bahdanau

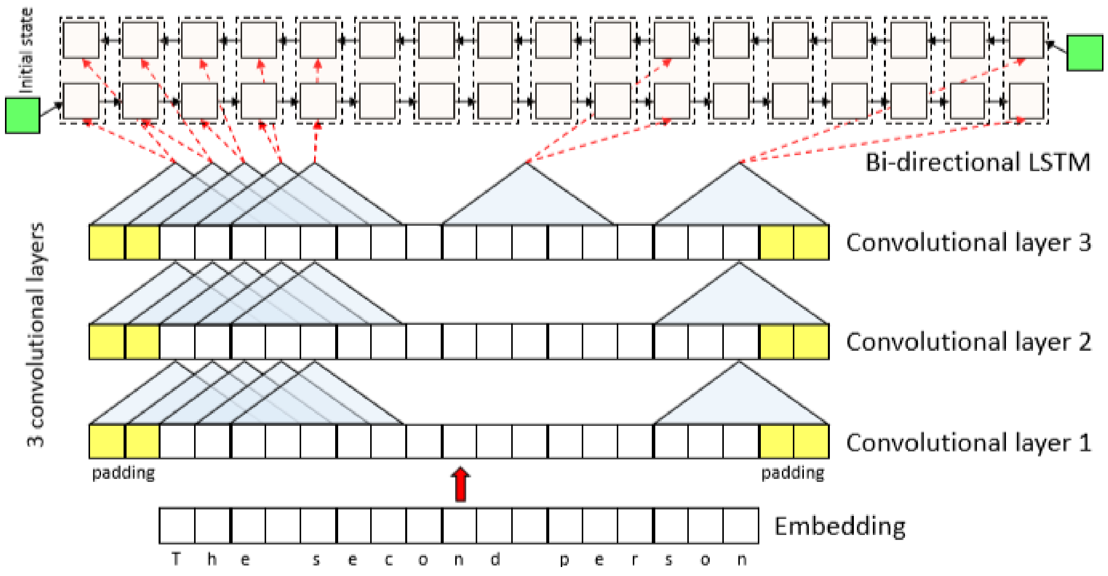


Figure 4.3: Convolutional Layers and Bidirectional LSTM

[7] to use cumulative attention weights from previous decoder time steps as an additional feature. Attention probabilities are computed after projecting inputs and location features to 128-dimensional hidden representations. The decoder is an autoregressive recurrent neural network which predicts a mel spectrogram from the encoded input sequence one frame at a time. The prediction from the previous time step is first passed through a small pre-net containing 2 fully connected layers of 256 hidden ReLU units. The pre-net acting as an information bottleneck is essential for learning attention. The prenet output and attention context vector are concatenated and passed through a stack of 2 uni-directional LSTM layers with 1024 units. The concatenation of the LSTM output and the attention context vector is projected through a linear transform to predict the target spectrogram frame. Finally, the predicted mel spectrogram is passed through a 5-layer convolutional post-net which predicts a residual to add to the prediction to improve the overall reconstruction. Each post-net layer is comprised of 512 filters with shape 5×1 with batch normalization, followed by tanh activations on all but the final layer. To return to the spectrogram dimension, the output of the PostNet is skiped through a fully connected layer with 80 neurons and the obtained data is added to the initial result of the decoder. In parallel to spectrogram frame prediction, the concatenation of decoder LSTM output and the attention context is projected down to a scalar and passed through a sigmoid activation to predict the probability that the output sequence has completed. This stop token prediction is used during inference to allow the model to dynamically determine when to terminate generation instead of always generating for a fixed duration. Specifically, generation completes at the first frame for which this probability exceeds a threshold of 0.5.
The convolutional layers in the network are regularized using dropout with probability 0.5, and LSTM layers are regularized using zoneout with probability 0.1.

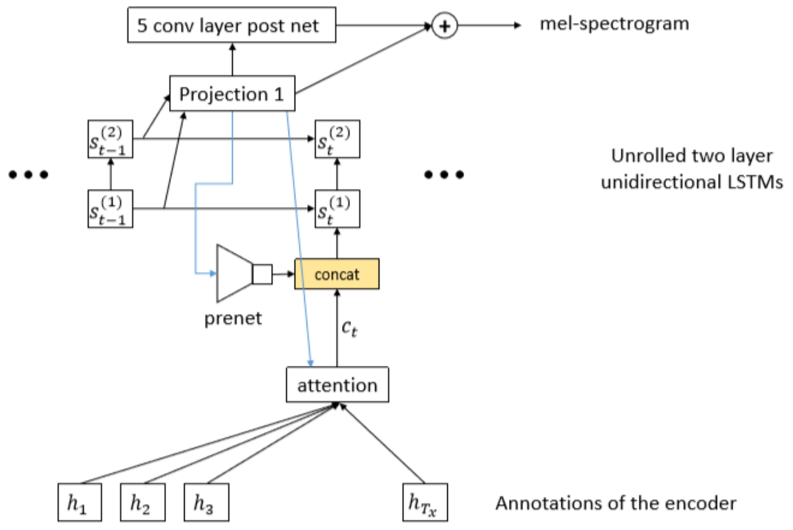


Figure 4.4: Attention - Decoder

In order to introduce output variation at inference time, dropout with probability 0.5 is applied only to layers in the pre-net of the autoregressive decoder.

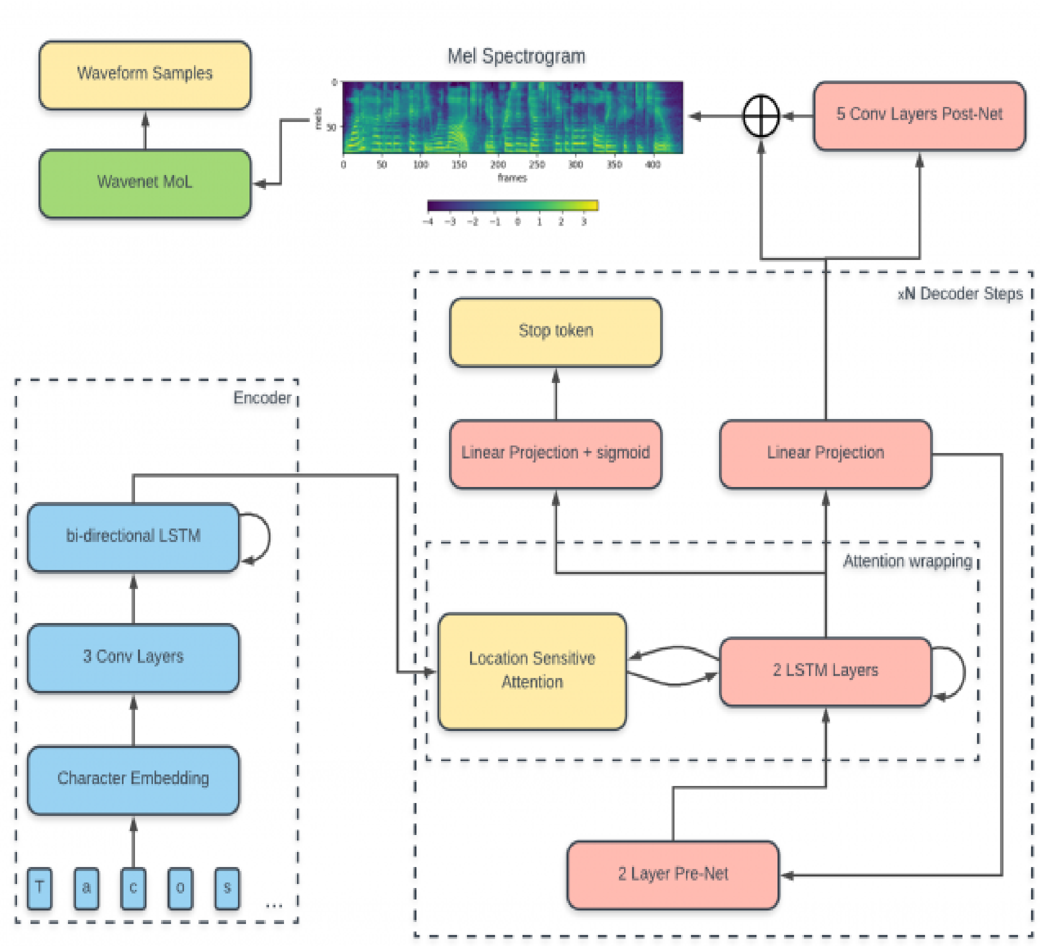


Figure 4.5: Tacotron-2.

Chapter 5

Neural Vocoder

Neural vocoders are neural networks that generate (speech) waveforms from acoustic feature inputs. WaveNet has been suggested for Text-To-Speech synthesis showing that a non-linear autoregressive system can mimic speech generation very well, if it is appropriately locally conditioned with linguistic information. If the local conditioning changes from linguistic information to acoustic information, then the WaveNet system is mainly a statistical vocoder. The second component of Tacotron-2 is a modified version of WaveNet used as a vocoder, to generate the synthesized speech from the spectrogram. During training our model, it would be a computationaly expensive choice to use WaveNet, so the Griffin-Lim algorithm is used and it is presented in the first section of this chapter.

5.1 Griffin-Lim Algorithm

The Griffin and Lim's [13] algorithm 5.1, recovers an audio signal given only the magnitude of its Short-Time Fourier Transform (STFT), also known as the spectrogram. It is an iterative algorithm that attempts to find the signal having an STFT such that the magnitude part is as close as possible to the modified spectrogram. The Griffin-Lim algorithm converges after 30 to 50 iterations. The Griffin-Lim produces characteristic artifacts and lower audio quality than approaches like WaveNet. Nevertheless, the Griffin-Lim spectrogram inversion is efficient and allows back propagation of derivatives since it is differentiable. Therefore, it could be the initial choice when debugging a new end-to-end system.

5.2 WaveNet

WaveNet is an autoregressive network, which generates a probability distribution of the next sample given some segment of previous samples. The next sample is produced by sampling from this distribution. An entire sequence of samples is produced by feeding previously generated samples back into the model. In order to make the training and generation tasks computationally tractable, the discrete

Algorithm 1 Griffin-Lim Algorithm

```
Input: $s$ spectrogram
$x$ =random($n$ samples)
for $i = 1 : n\_$ iterations
        $c_1 = STFT(x)$
        $a = angle(c_1)$
        $c_2 = s \cdot e^{ja}$
        $x = ISFTF(c_2)$
```

softmax is chosen as the probability distribution. Additionally, the speech samples are dynamic range compressed via μ-law transformation and then quantized to 8-bits. In order to convey verbal and prosodic information, WaveNet is conditioned on linguistic and/or acoustic features [10, 41]. The conditioning features are upsampled to the desired frequency and fed into the basic WaveNet through a conditioning network.

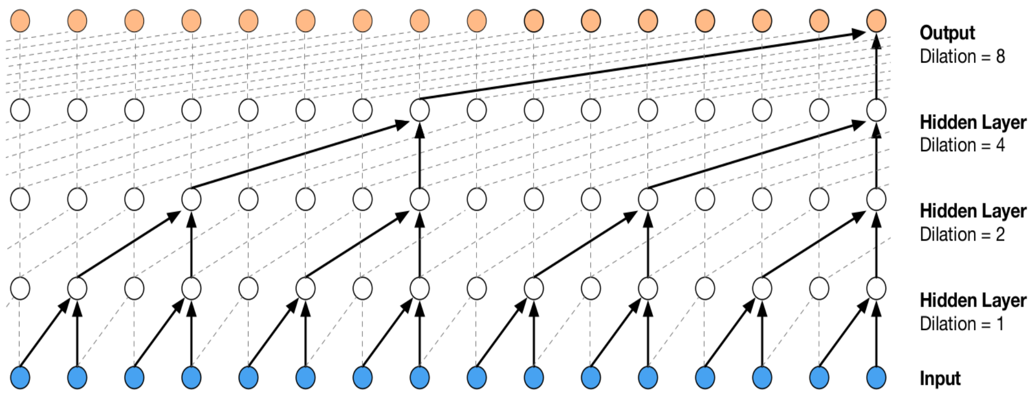


Figure 5.1: Visualization of a stack of dilated causal convolutional layers.

5.2.1 WaveNet architecture

Wavenet architecture has two parts: a convolution stack and a post-processing module. The convolution stack consists of dilated convolution residual blocks and acts as a multi-scale feature extractor, while the post-processing module combines the information from the residual blocks to predict the next sample. Let r be the receptive field of WaveNet, $x = \{x_1, x_2, \ldots, x_n\}$ be a sequence of quantized speech samples and $h = \{h_1, h_2, \ldots, h_n\}$ be the corresponding sequence of upsampled conditioning features. Assuming that $n > r$, the output of the conditioned WaveNet is described by the following conditional probability distribution.

$$P(x_n | x_{n-1}, x_{n-2} ..., x_{n-r}, h_n) \tag{5.1}$$

WaveNet is implemented as a stack of residual blocks, where each block contains expert and gate one-dimensional dilated causal convolutions (Fig. 5.2). The output of the expert and the gate are combined via element-wise multiplication. A block, i, computes a hidden state vector $z^{(i)}$, Eq.(5.2), and then (due to the residual connections between layers) adds to its input $x^{(i-1)}$ to generate its output $x^{(i)}$.

$$z^{(i)} = \tanh(W_f^{(i)} * x^{(i-1)} + L_f^{(i)}) \odot \sigma(W_g^{(i)} * x^{(i-1)} + L_g^{(i)}) \tag{5.2}$$

In Eq. (5.2), $L_f^{(i)}$ and $L_g^{(i)}$ are the outputs for block i of the conditioning network when it is fed with h. Symbol $*$ denotes convolution and symbol $\odot$ denotes element-wise multiplication. Figure 5.1 gives a visualization of a stack of dilated causal convolutional layers, the building blocks of the WaveNet Deep Learning Model. Figure 5.3 depicts the integration of acoustic features in the WaveNet architecture. The acoustic features, which are computed framewise, are of lower sampling frequency and are up-sampled to the frequency of the raw waveform.

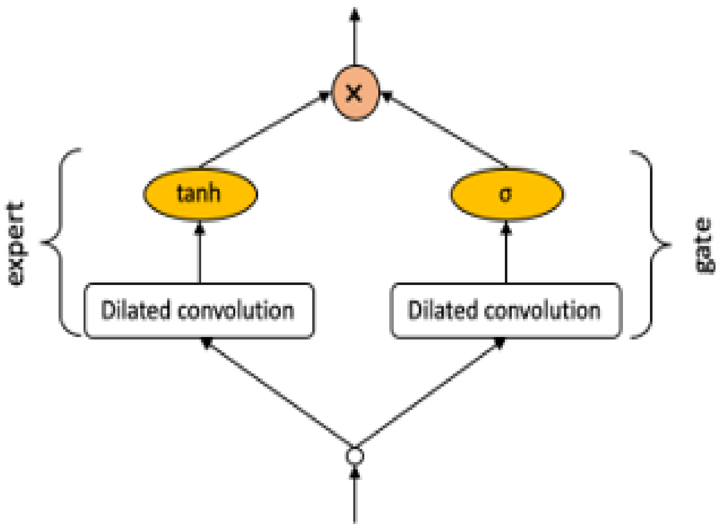


Figure 5.2: WaveNet is implemented as a stack of residual blocks, where each block contains expert and gate one-dimensional dilated causal convolutions.

5.2.2 Acoustic features for conditioning WaveNets

A classical source-filter vocoder widely used in parametric TTS, decomposes the speech signal into a smooth spectral envelope and a spectrally white excitation. The excitation is further characterized by spectral harmonics arising from the fundamental frequency and an aperiodicity spectrum that measures deviation from an ideal harmonic series structure. A neural vocoder is a neural network that

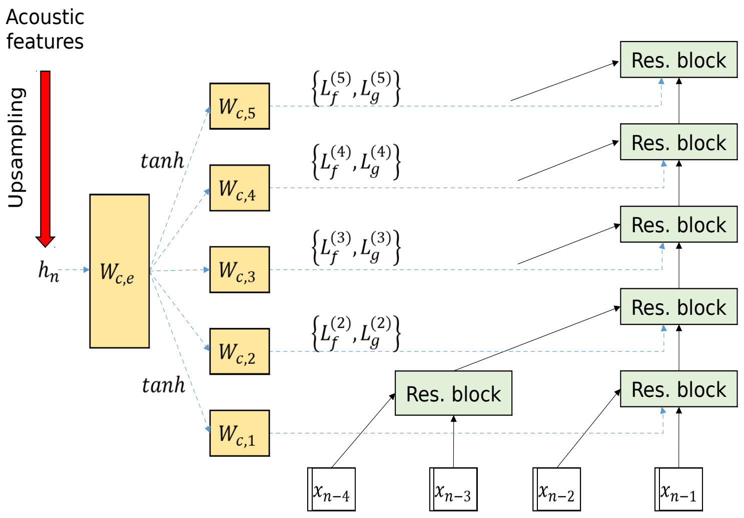


Figure 5.3: Local conditioning using Acoustic features

generates (speech) waveforms from acoustic feature inputs. WaveNet-type models are proposed as neural vocoders. One of the features of WaveNet is that it does not depend on the characteristics of the data to be applied, and can build a generative model in a data-driven manner. In case of speech, various assumptions based on the prior knowledge specific to speech can be avoided. Further, WaveNet can even capture the characteristics of non-speech sounds like breathing and mouth movements, which shows the greater flexibility of this model. In the original literature, WaveNet was applied to text-to-speech (TTS), and the quality of synthesized speech exceeded that of state-of-the-art approaches. The input to the WaveNet was the linguistic feature, the fundamental frequency (F0), and the phoneme duration, except for the waveform samples that it generated in the past. However, it was not specifically clarified what kind of features works effectively other than those.

There is an increasing interest in using WaveNet as statistical vocoder for generating speech waveforms from various acoustic features (Fig. 5.4, 5.3). A method that uses the acoustic features of existing vocoders as auxiliary features of WaveNet is proposed in [41]. The advantage of this method is that it does not require explicit modelling of excitation signals and various assumptions specific to speech generation process and speech analysis. The acoustic correlated/uncorrelated features are exlored as local conditioning in WaveNet vocoder, in the paper [6]. Filterbank coefficients and MCEP are used as correlated and uncorrelated local conditioning, respectively. In speech synthesis, MCEPs are mainly used in GMM and HMM due to the assumption of conditional independence and to avoid the computationally intensive algorithm. However, due to advanced algorithm like WaveNet, these assumptions are not required and can produce more natural speech than the conventional source-filter based vocoder. Only 1 hour of training data was enough for producing very good quality of speech. It is shown that filter-bank features are providing better local conditioning than cepstrum coefficients, allowing to produce good quality of speech for both male and female speakers.

Glottal vocoder is a kind of vocoder, that is trained to generate glottal source excitation waveforms conditioned on the acoustic features of a SPSS system. The generated excitation waveforms are then filtered using the vocal tract filter in order to produce synthetic speech. In [16] the authors extend the glottal waveform modelling domain to use the more powerful WaveNet-like architecture, presenting a raw waveform glottal excitation model, called GlotNet.

5.3 WaveNet vocoder in Tacotron-2

A modified version of the WaveNet architecture from [10] is used to invert the mel spectrogram feature representation into time-domain waveform samples. As in the original architecture, there are 30 dilated convolution layers, grouped into 3 dilation cycles, i.e., the dilation rate of layer k (k = 0...29) is 2k (mod 10). To work with the 12.5 ms frame hop of the spectrogram frames, only 2 upsampling layers are used in the conditioning stack instead of 3 layers. Instead of predicting discretized buckets with a softmax layer, we follow PixelCNN++ [43] and Parallel WaveNet [48] and use a 10 component mixture of logistic distributions (MoL) to generate 16-bit samples at 24 kHz. To compute the logistic mixture distribution, the WaveNet stack output is passed through a ReLU activation followed by a linear projection to predict parameters (mean, log scale, mixture weight) for each mixture component. The loss is computed as the negative log-likelihood of the ground truth sample.

5.4 Real-time synthesis with WaveNet

Autoregressive models, such as the WaveNet, which are trained with the cross-entropy loss criterion, have parallel training and sequential generation (Fig. 5.5). This makes the generation of samples very slow and inappropriate for real time applications. The first implementation of WaveNet required many hours of computations to synthesize 3 seconds of speech at 16000 sampling frequency. A PhD student, Tom Le Paine [29], noticed that most of the calculations during synthesis are redundant and proposed to cache past samples using queues (Fig. 5.6). This algorithm, which is called fast WaveNet, greatly accelerates the generation of samples (Fig. 5.7). For example, a well written implementation of fast WaveNet in TensorFlow or PyTorch requires about 3 minutes to synthesize 3 seconds of speech at 16000 sampling frequency on an nvidia 1080ti GPU. There have been many attempts to further accelerate the generation of WaveNet, including reducing the number of residual layers, altering the architecture and implementing the algorithm in CUDA. NVIDIA claimed that a WaveNet implementation in a persistent CUDA kernel using float16 precision arithmetic can generate samples slightly faster than real time possibly at the cost of a slightly reduced quality of the synthesized audio [4]. However, this acceleration is still not enough because in real time applications the sampling frequency is at least 24000 and because the neural vocoder should synthesize speech at least 6 times faster than real time so that within the real time window the Tacotron to predict Mel filter banks the WaveNet to synthesize the waveform and the communication between a device and a server to complete.

In order to address the real time requirements, Deep Mind, presented the Parallel WaveNet [48] which is a feed forward neural network that transforms a sequence of uncorrelated random samples drawn from the logistic distribution, Logistic(0, 1), into speech (Fig. 5.8). Parallel WaveNet is based on inverse autoregressive flows [21]. Contrary to autoregressive models, the inverse autoregressive models have sequential training and parallel generation (Fig. 5.9). Due to parallel generation, even a non-optimized implementation of Parallel WaveNet synthesize speech at least 20 times faster than real time. The major issue with Parallel WaveNet is the slow training using cross-entropy. For this reason, Deep Mind trained the Parallel WaveNet using knowledge distillation [14]. Originally knowledge distillation was used to transfer the knowledge of a huge neural network to a smaller network at the cost of some performance degradation and at the benefit of huge reduction in the computational requirements. However, in the case of Parallel WaveNet, the teacher network is smaller than the student network. The teacher is a WaveNet with output the parameters of a continuous distribution such as a mixture of logistics [48] or a Gaussian distribution [14]. The teacher is trained with cross-entropy. The student is a stack of 5 autoregessive flows, each of which is implemented with a WaveNet. The student is trained with Kullback-Leibler divergence between the distribution that predicts and the distribution that the teacher predicts. During the training of student network the weights of the teacher network do not change.

Another attempt to address the real time generation of audio, was the WaveRNN [17] which was also proposed by Deep Mind. WaveRNN substitutes the residual stack of WaveNet with a GRU recurrent cell (Fig. 5.11 and 5.10). Efficient implementations of WaveRNN can run more than 6 time faster than real time on GPUs and even faster than real time on devices with low computational power such as the mobile phones. As far as the quality of the synthesized speech is concerned, the best quality is produced by WaveNet, followed closely by WaveRNN and then comes the Parallel WaveNet.

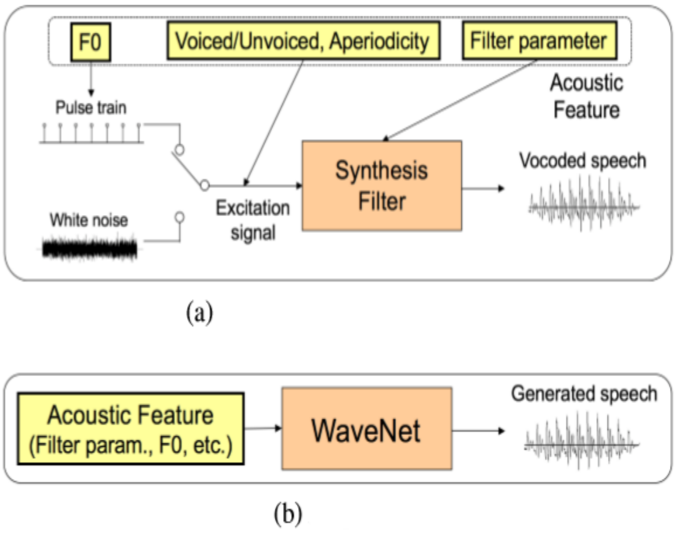


Figure 5.4: (a) conventional Vocoder, (b) WaveNet Vocoder

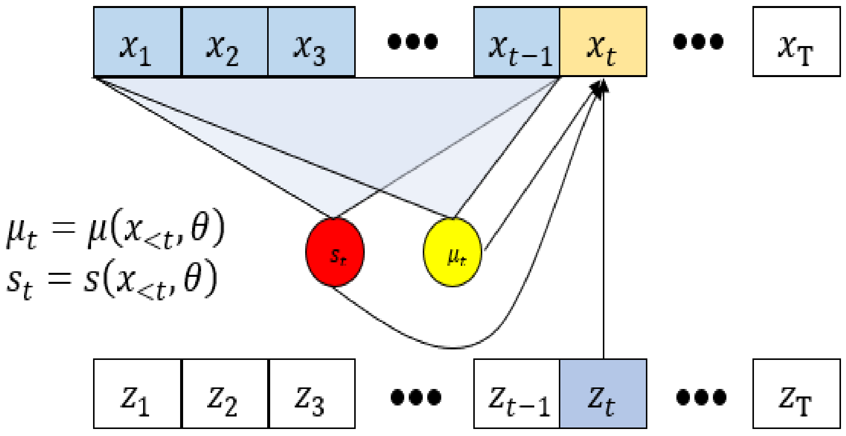


Figure 5.5: The WaveNet allows parallel training but has sequential generation.

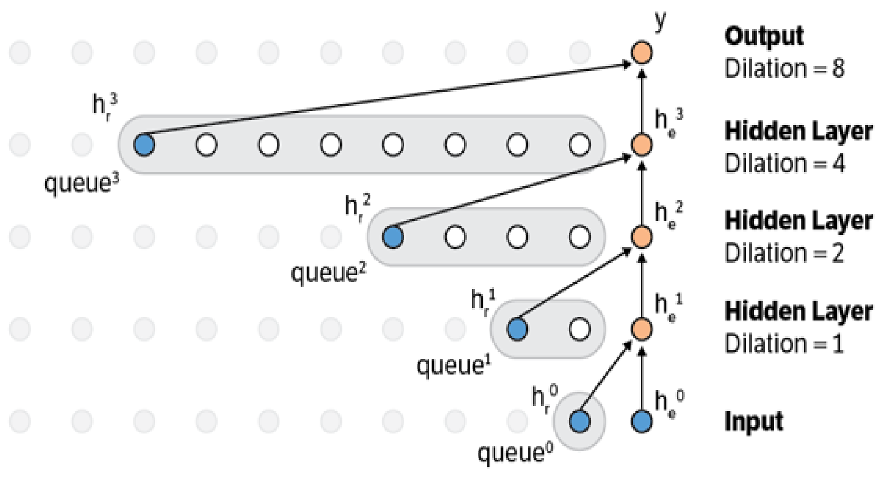


Figure 5.6: The caching scheme for efficient generation. Due to dilated convolutions, the size of the queue at the l-th hidden layer is 2^l.

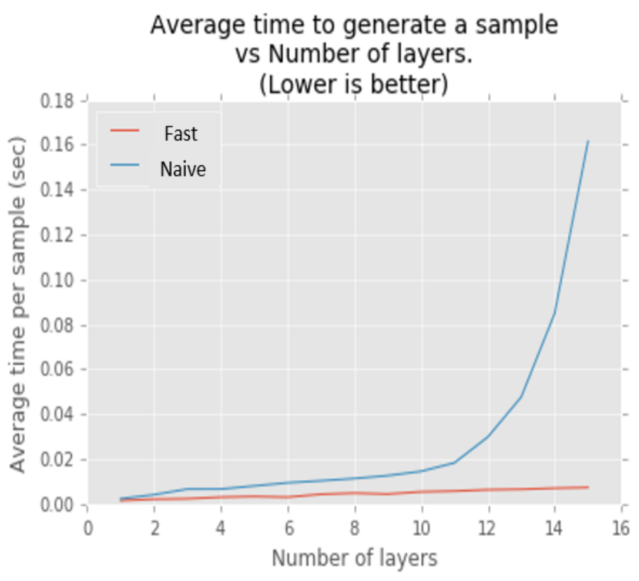


Figure 5.7: Timing experiments comparing the generation speeds of the naive algorithm and Fast WaveNet.

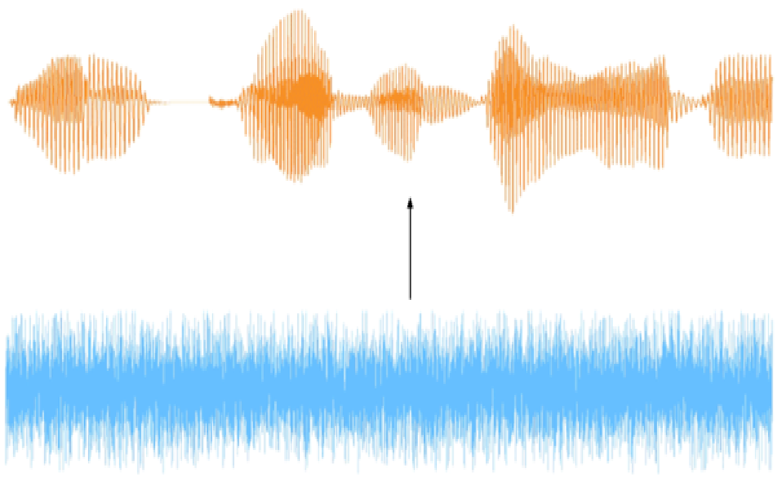


Figure 5.8: Parallel WaveNet converts noise into speech.

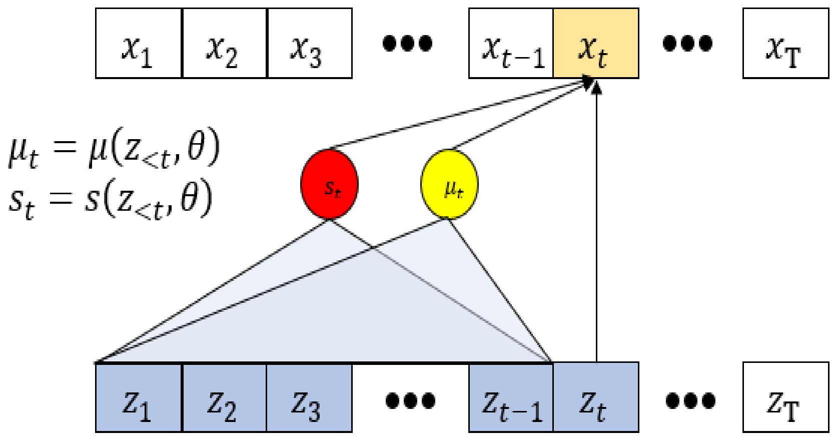


Figure 5.9: Parallel WaveNet allows parallel generation but training the model with cross-entropy is sequential and slow.

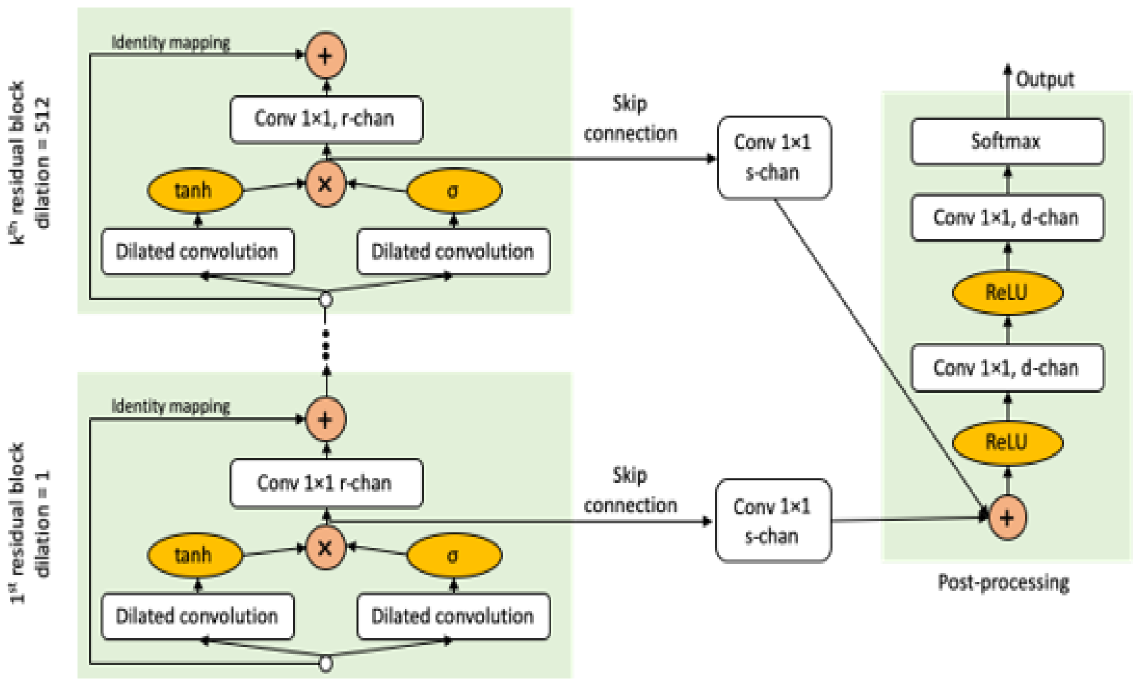


Figure 5.10: WaveNet

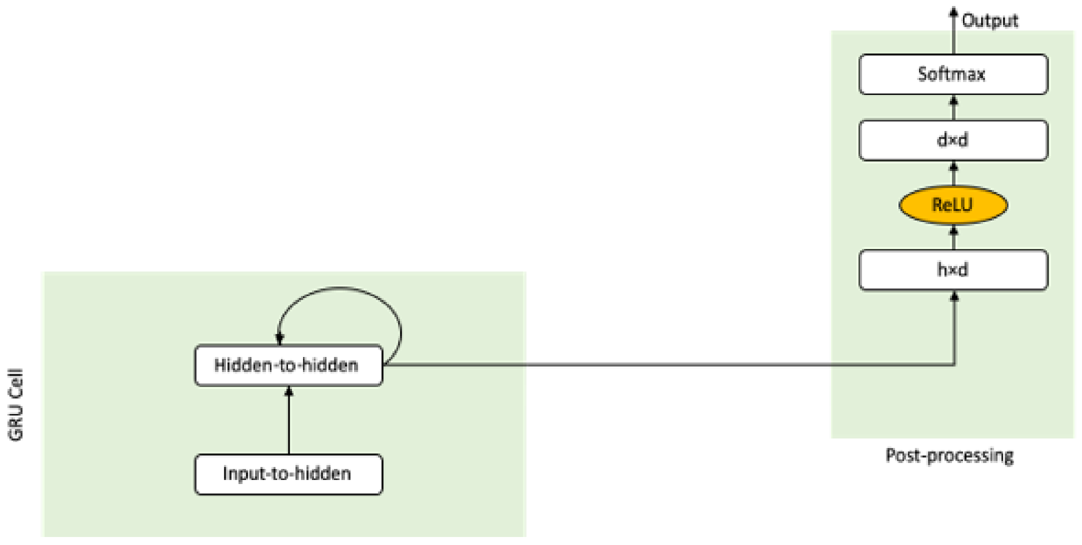


Figure 5.11: WaveRNN

Chapter 6

Experiments with various languages and Results

The training process involves first training the feature prediction network on its own, followed by training a modified WaveNet independently on the outputs generated by the first network. The Experiments were done using code from github [5] with small changes of the hyperparameters. Only one GPU is used (GeForce GTX 1080 ti with tensorflow-gpu 1.12). The network is trained using backpropagation. ADAM is used as an optimizer, Mean Square Error is used as an error function before and after PostNet, and also binary Cross Entropy above the actual and predicted values of the Stop Token Prediction layer. The sum of these three is the resulting error. Tacotron training (150000 steps) needed 3.5 days and Wavenet training 750000 steps took almost 7 days. The hyperparameter batchsize had to be changed (batchsize=16) because of memory problems (OOM error occurred when the batchsize was 32). Some more hyperparameters that depend on the sampling rate, were changed (table 6.1).

Table 6.1: Hyperparameters that depend on the sampling rate of the recordings.

Hyperparameters	**fs=22500Hz**	**fs=16000Hz**
win_size	1100	800
hop_size	275	200
frame_shift_ms	12.5	12.5
n_fft	2048	1024
num_freq	1025	513
upsample_scales	[5 5 11]	[10 20]

6.0.1 Preprocessing the data

The input data is the text, and the output is the Mel-spectrogram, a low level representation obtained by applying a fast Fourier transform to a discrete audio signal. The spectrograms obtained in this way still need to be normalized by compressing the dynamic range. This allows reducing the natural ratio between the loudest and quietest sound on the record. Based on the above-mentioned transformations, the sound is encoded into the Mel-spectrogram. The text is tokenized and turned into a sequence of integers. All texts are normalized. After preprocessing, we receive sets of numpy arrays of numerical sequences and Mel-Spectrograms recorded in numpy files (.npy). In order to match all the dimensions in the tensor batches at the learning stage, we add paddings to short sequences.

6.1 Datasets

Developing TTS systems for any given language is a significant challenge, and requires large amounts of high quality acoustic recordings. Because of this, these systems are only available for a tiny fraction of the world's languages. However, in languages other than English, there has been little exploration in this direction. Both the scarcity of annotated data and the complexity of the language increase the difficulty of the problem.

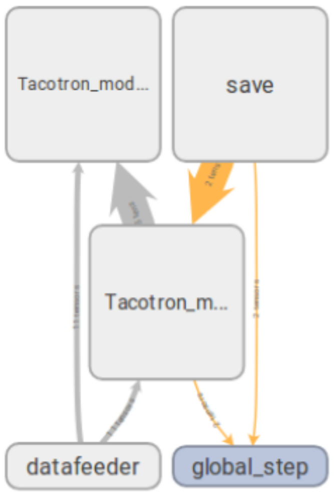


Figure 6.1: Main Graph

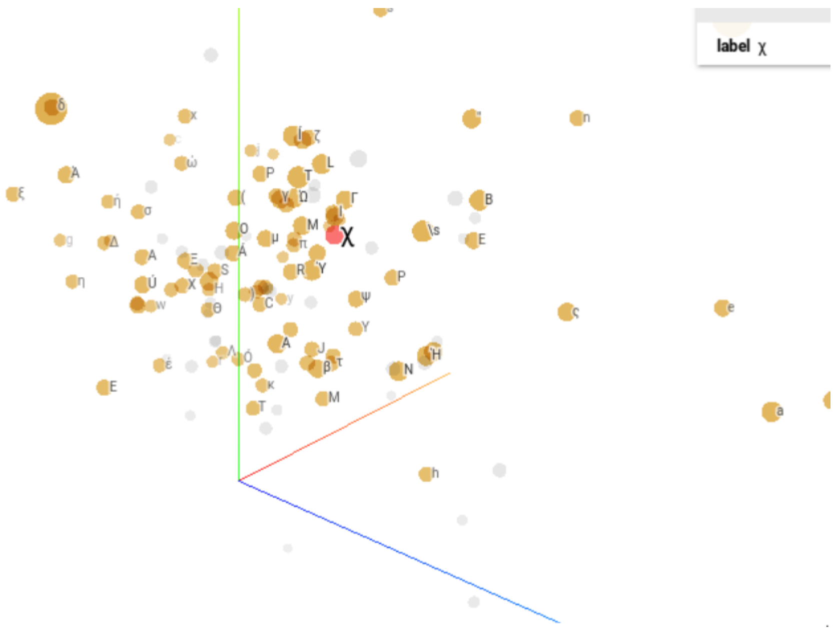


Figure 6.2: Model Character Embeddings Visualization (Spanish - Greek)

6.1.1 English, Spanish and Italian

The first datasets that we used in this work come from M-AIlabs [3], (Munchen artificial Intelligence Laboratories):

- English: LJSpeech-1.1 Dataset [2] (24 hours of labeled single actress voice recording, 22000Hz).
- Spanish: book = enedia, male speaker tux (11.00 hours, 16000 Hz).
- Italian: book = imalavoglia, female voice lisa_caputo (5856 examples, 6.73 hours,16000Hz)

All audio-files are in wav-format, mono.The models are trained on normalized text (all text in the datasets is spelled out).

6.1.2 Greek

The dataset for the Greek Language is obtained from ILSP [1]. It is called elenet (3.0 hours, 13 male and female speakers, 16000Hz, 2154 utterances). This dataset was created for speech Recognition. So it contains different kinds of noise (paff-noise, breaths, clear-throat, rarely wrong pronunciation). We trained Tacotron-2 with the above dataset on two different machines, with various hyperparameters (transliteration cleaners, basic cleaners, GTA=true, GTA=false). After each one of these experiments the model that was obtained could not generalise. It could synthesize only sentences that were seen in the database.

Some months later we received more Greek data (FemaleVoice2, 44100Hz, 413 sentences with multiple recordings of each sentence). The new voice had not naturalness. These new recordings were created for concatenative speech synthesis. We used both Greek datasets (4.94 hours) for a new experiment. The model could generalize but with noise. The stop token prediction failed, so in some synthesized sentences, it cut the last phoneme, or it put noise at the end (Fig 6.5).

6.1.3 Related Work in Greek

There is not any known work on end-to end neural text-to-speech for the Greek language. The previous years, Unit-Selection and Parametric Speech Synthesis were the dominant techniques for example [33], [11]. Researchers in ILSP (Institute for Language and Speech Processing - Research Center 'Athena') have published important work on the Greek Language.

6.2 Spanish-Greek Language Adaptation

We tried the following two choices:

1. Tacotron trained with both datasets and WaveNet trained only with the Greek one.

2. Both Tacotron and WaveNet were trained with both datasets.

For the first experiment with Spanish and Greek Language, the Spanish Database was used for 50000 steps and then the Greek Database for 100000 steps. The Spanish Dataset that was used, is obtained from [3], (18.06 hours), book='don_quijote', male and female speakers. (11133 utterances, 5202428 mel frames, 1040485600 audio timesteps, 16000Hz). The model could generalize (it was tested also with out-of-domain text).

For the second experiment with Spanish and Greek Language, Tacotron was trained with Spanish Database for 95000 steps and then the Greek Database for 55000 steps. WaveNet was also trained with both datasets From 0 steps to 170000 steps Greek, from 170000 steps to 630000 steps Spanish, from 630000 steps to 750000 steps Greek. Our results were improved (Fig. 6.6). This model is chosen for the listening test that will be described later.

Before training, preprocessing was done for both languages. Basic _ cleaners were used (the parameter was changed in hparams.py). The characters that we used are: ASCII + special spanish characters + greek characters. Greek Dataset:

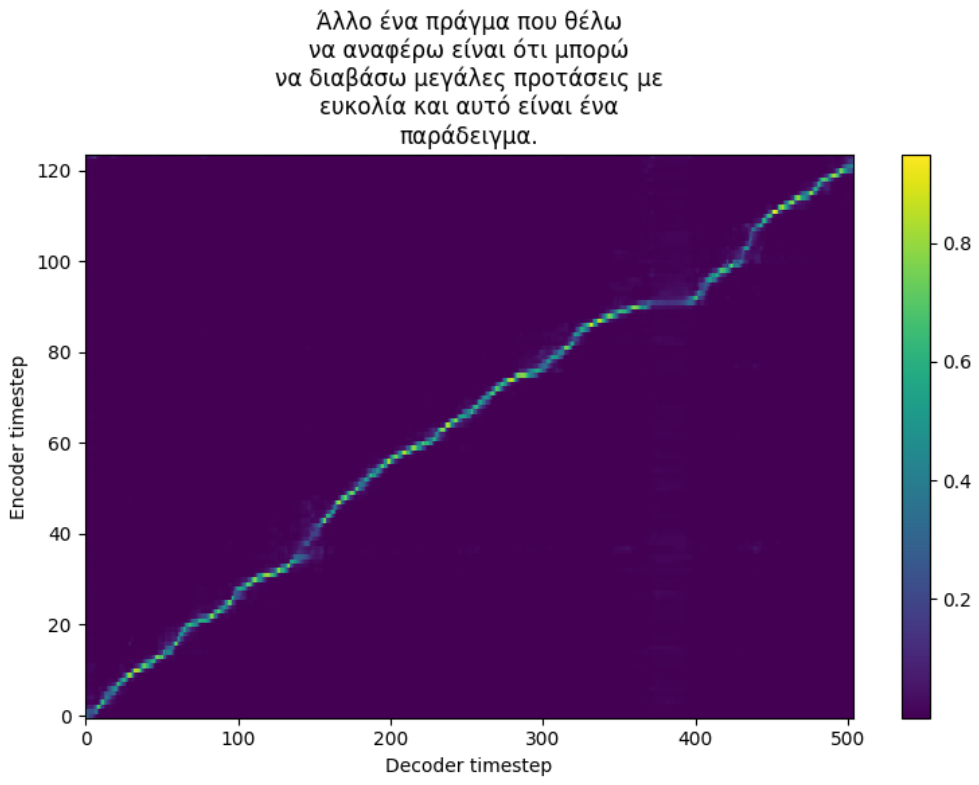


Figure 6.3: Attention is a key element of the entire system.

elenet, (3.00h). It is described earlier.

6.2.1 Speaker Adaptation

With a multi-speaker database like ours, another experiment that has been done is training the last 5000 steps of tacotron with data of one speaker in order to synthesize wavs only with his/her voice. If the data of one speaker is very little, the hyperparameter test_size must be changed to None. Otherwise we can choose data from two or three speakers that we prefer (Fig. 6.7).

1. We trained the previous model again from step 145000 to 150000 with 239 examples (0.32 hours of our dataset from speaker s027 male). It managed to synthesize speech with this speaker, but it did mistakes and sometimes produced noise. The quality was degraded.

2. We trained with 947 examples (1.33hours) from speakers s023 female and speaker s027 male. During synthesis the quality was good and no mistakes

appeared.

3. We received a new dataset. Female voice, perfect recordings 44100Hz, but very slow. Speaker adaptation with 413 examples (960 with multiple recordings each) 1.94 hours had not good results. It could synthesize sentences with the new voice but in long sentences it produced noise. We can see that we need more sentences recorded with the same speaker not more recordings of the same sentences.

6.3 Results

Tacotron-2 can generate natural Greek speech, with the prosody of the speakers of the database. The stresses sound correct and no pronunciation difficulties are found. It can read complex words easily. It can read the same sentence differently each time it is asked (this is due to the pre-net drop-out even at inference time). Sometimes noisy wavs are produced. This is because of the data that we use. Other problems is that some wavs begin suddenly. In some rare cases, end point prediction fails (the system cannot find where it must stop -it cuts the last syllable or adds noise at the end- when the sentence is longer than 5 seconds).
We tested the generalization ability of our system to out-of-domain text. We evaluated samples generated from sentences from books and sentences that other people asked us to synthesize.

This work was presented during the Researcher's night at FORTH, Heraklion, Crete on 27th of September 2019. Groups of students and families tested our model with their own sentences. Most of them were surprised because of the naturalness of the produced speech!

6.3.1 Evaluation

When generating speech in inference mode, the ground truth targets are not known. Therefore, the predicted outputs from the previous step are fed in during decoding, in contrast to the teacher-forcing configuration used for training. We randomly selected 16 fixed examples from the test set of our internal dataset as the evaluation set. We created a listening test. Each sentence in the test appears in two samples (one original and one synthesized) in random order. We used two voices from our multi-speaker database one male and one female. The goal was to evaluate naturality and intelligibility (but also the criteria should include presence of artifacts and quality of the sound). We asked 30 volunteers to rate each sample on a scale from 1 to 5 with 1 point increments (perfect, very good, good, bad, very bad), from which a subjective mean opinion score (MOS) is calculated. Using this set allows us to compare to the ground truth. The results are shown in the table 6.2.

Table 6.2: Mean Opinion Scores

Systems	MOS
Original recordings	3.82 ± 0.19
Tacotron-2 (Spanish/Greek)	3.15 ± 0.20

6.4 Future Work

This work must be continued.

- Self - Attention is the next aim. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence. It has been used successfully in a variety of tasks including reading comprehension, abstractive summarization, textual entailment and learning task-independent sentence representations. A new method [22] based on self Attention outperforms Tacotron-2 in efficiency and Performance (with english dataset). Inspired by the success of Transformer network [49] in neural machine translation (NMT), in this paper, the multi-head attention mechanism is adapted to replace the RNN structures and also the original attention mechanism in Tacotron 2. With the help of multi-head self-attention, the hidden states in the encoder and decoder are constructed in parallel, which improves training efficiency. Meanwhile, any two inputs at different times are connected directly by a self-attention mechanism, which solves the long range dependency problem effectively. Using phoneme sequences as input, their Transformer TTS network generates mel spectrograms, followed by a WaveNet vocoder to output the final audio results.

- More experiments for the Greek and other languages, sizes of datasets, batch-sizes, combinations of hyperparameters must be done to explore the benefits of these new techniques.

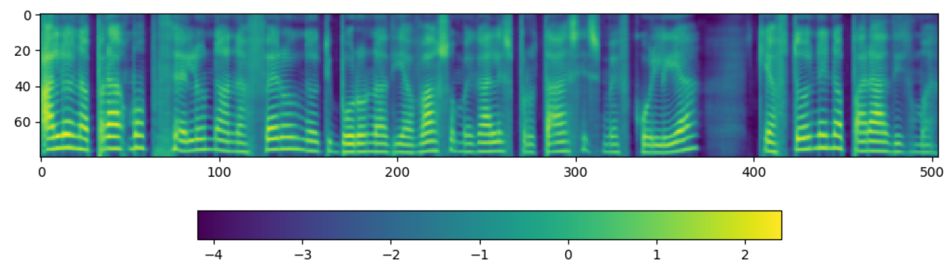
- Experiments with WaveRNN can be done, in order to improve speed during synthesis.

- Multi-GPU experiments must also be tested.

- Speech Enhancement can be applied for our Greek database.

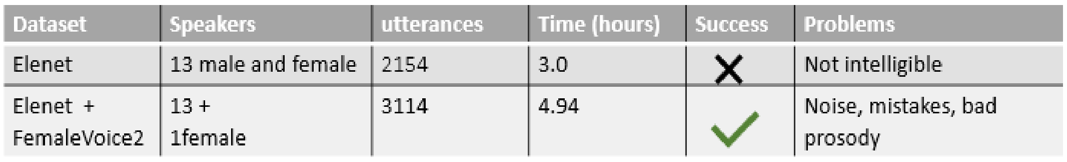
- Experiments with another Greek dataset, created for speech synthesis, will be tested soon.

- Instead of attempting to build a high quality voice for a single language using monolingual data from multiple speakers, can we somehow combine the limited monolingual data from multiple speakers of multiple languages to build a single multilingual voice that can speak ’any’ language?



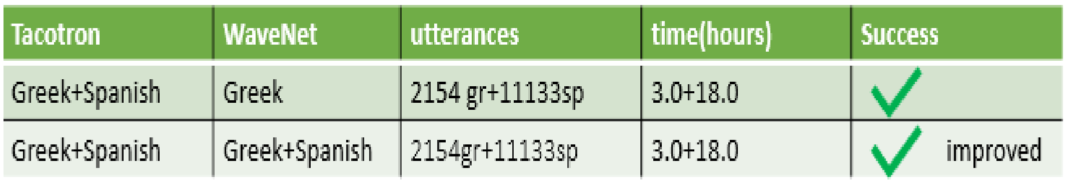
Άλλο ένα πράγμα που θέλω
να αναφέρω είναι ότι μπορώ
να διαβάσω μεγάλες προτάσεις με
ευκολία και αυτό είναι ένα
παράδειγμα.

Figure 6.4: Mel Spectrogram of a synthesized sentence



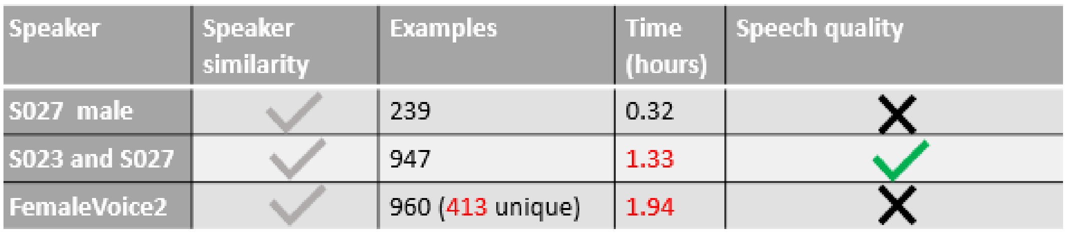
Dataset	Speakers	utterances	Time (hours)	Success	Problems
Elenet	13 male and female	2154	3.0	✗	Not intelligible
Elenet + FemaleVoice2	13 + 1female	3114	4.94	✓	Noise, mistakes, bad prosody

Figure 6.5: Results of experiments with Greek Database.



Tacotron	WaveNet	utterances	time(hours)	Success
Greek+Spanish	Greek	2154 gr+11133sp	3.0+18.0	✓
Greek+Spanish	Greek+Spanish	2154gr+11133sp	3.0+18.0	✓ improved

Figure 6.6: Results of experiments with Spanish and Greek datasets.



Speaker	Speaker similarity	Examples	Time (hours)	Speech quality
S027 male	✓	239	0.32	✗
S023 and S027	✓	947	1.33	✓
FemaleVoice2	✓	960 (413 unique)	1.94	✗

Figure 6.7: Results of Speaker Adaptation experiments.

Appendix A

hparams.py

```
    # Default hyperparameters
hparams = tf.contrib.training.HParams(
    cleaners='basic_cleaners',
    tacotron_gpu_start_idx = 0,
    tacotron_num_gpus = 1,
    wavenet_gpu_start_idx = 0,
    wavenet_num_gpus = 1,
    split_on_cpu = False,
#Audio

num_mels = 80,
num_freq = 513,
rescale = True,
rescaling_max = 0.999,
trim_silence = True,
clip_mels_length = True,
max_mel_frames = 1000,
use_lws=False,

# preprocessing
#Mel spectrogram
    n_fft = 1024,
hop_size = 200,
win_size = 800,
sample_rate = 16000,
frame_shift_ms = None, (Recommended: 12.5)
trim_hop_size = 128,
trim_top_db = 23,
signal_normalization = True,
allow_clipping_in_normalization = True,
symmetric_mels = True,
max_abs_value = 4.,
normalize_for_wavenet = True,
clip_for_wavenet = True, #whether to clip [-max, max] before
preemphasize = True, #whether to apply filter
preemphasis = 0.97, #filter coefficient.

#Limits
```

```
min_level_db = -100,
ref_level_db = 20,
fmin = 55,
fmax = 7600, #To be increased/reduced depending on data.

#Griffin Lim
power = 1.5,
griffin_lim_iters = 60, #Number of G&L iterations,

#Tacotron
outputs_per_step = 1,
stop_at_any = True,
embedding_dim = 512, #dimension of embedding space

#Encoder parameters
enc_conv_num_layers = 3,
enc_conv_kernel_size = (5, ),
enc_conv_channels = 512,
encoder_lstm_units = 256,

#Attention mechanism
smoothing = False,
attention_dim = 128,
attention_filters = 32,
attention_kernel = (31, ),
cumulative_weights = True,
%#Whether to cumulate (sum) all previous attention weights or
%#simply feed previous weights (Recommended: True)

#Decoder
prenet_layers = [256, 256],
decoder_layers = 2,
decoder_lstm_units = 1024,
max_iters = 2000,
#Max decoder steps during inference (Just for safety from infinite loop cases)

#Residual postnet
postnet_num_layers = 5,
postnet_kernel_size = (5, ),
postnet_channels = 512,

#CBHG mel->linear postnet
cbhg_kernels = 8,
cbhg_conv_channels = 128,
cbhg_pool_size = 2,
cbhg_projection = 256,
cbhg_projection_kernel_size = 3,
cbhg_highwaynet_layers = 4,
cbhg_highway_units = 128,
cbhg_rnn_units = 128,

#Loss params
mask_encoder = True,
mask_decoder = False,
cross_entropy_pos_weight = 20,
predict_linear = False,

#Wavenet
# Input type:
# 1. raw [-1, 1]
# 2. mulaw [-1, 1]
# 3. mulaw-quantize [0, mu]
# If input_type is raw or mulaw, network assumes scalar input and
# discretized mixture of logistic distributions output,
# otherwise one-hot
# input and softmax output are assumed.
#Model generatl type
input_type="raw",
quantize_channels=2 ** 16,

#Minimal scales ranges for MoL and Gaussian modeling
log_scale_min=float(np.log(1e-14)),
#Mixture of logistic distributions minimal log scale
log_scale_min_gauss = float(np.log(1e-7)),
#Gaussian distribution minimal allowed log scale

#model parameters
out_channels = 2,
layers = 20,
stacks = 2,
residual_channels = 128,
gate_channels = 256,
skip_out_channels = 128,
kernel_size = 3,

cin_channels = 80,
upsample_conditional_features = True,

upsample_type = '1D',
upsample_activation = 'LeakyRelu', #Activation function used during upsampling.
upsample_scales = [10, 20],
freq_axis_kernel_size = 3,
leaky_alpha = 0.4,

#global conditioning
gin_channels = -1,
#Set this to -1 to disable global conditioning,
use_speaker_embedding = True,
n_speakers = 5,
use_bias = True,

max_time_sec = None,
max_time_steps = 11000,

#Tacotron Training
#Reproduction seeds
tacotron_random_seed = 5339,
#Determines initial graph and operations
```

```
tacotron_data_random_state = 1234,

#performance parameters
tacotron_swap_with_cpu = False,
tacotron_batch_size = 16,
#number of training samples on each training steps

tacotron_synthesis_batch_size = 1,
tacotron_test_size = 0.05,
tacotron_test_batches = None,

#Learning rate schedule
tacotron_decay_learning_rate = True,
#boolean, determines if the learning rate will
#follow an exponential decay
tacotron_start_decay = 50000,
#Step at which learning decay starts
tacotron_decay_steps = 50000,
#Determines the learning rate decay slope (UNDER TEST)
tacotron_decay_rate = 0.5,
#learning rate decay rate (UNDER TEST)
tacotron_initial_learning_rate = 1e-3,
#starting learning rate
tacotron_final_learning_rate = 1e-5,
#minimal learning rate

#Optimization parameters
tacotron_adam_beta1 = 0.9,
#AdamOptimizer beta1 parameter
tacotron_adam_beta2 = 0.999,
#AdamOptimizer beta2 parameter
tacotron_adam_epsilon = 1e-6,
#AdamOptimizer Epsilon parameter

#Regularization parameters
tacotron_reg_weight = 1e-7,
#regularization weight (for L2 regularization)
tacotron_scale_regularization = False,
tacotron_zoneout_rate = 0.1,
#zoneout rate for all LSTM cells in the network
tacotron_dropout_rate = 0.5,
#dropout rate for all convolutional layers + prenet
tacotron_clip_gradients = True,
#whether to clip gradients

#Evaluation parameters
natural_eval = False,

#Decoder RNN learning can take be done in one of two ways:
#Teacher Forcing: vanilla teacher forcing
#(usually with ratio = 1). mode='constant'
#Curriculum Learning Scheme: From Teacher-Forcing
#to sampling from previous outputs is
#function of global step. (teacher forcing ratio decay)
```

```
#mode='scheduled'
#The second approach is inspired by:
#Bengio et al. 2015: Scheduled Sampling for Sequence
#Prediction with Recurrent Neural Networks.
tacotron_teacher_forcing_mode = 'constant',
#Can be ('constant' or 'scheduled'). 'scheduled'
# mode applies a cosine teacher forcing ratio decay.
#(Preference: scheduled)
tacotron_teacher_forcing_ratio = 1.,
tacotron_teacher_forcing_init_ratio = 1.,
#initial teacher forcing ratio. Relevant if mode='scheduled'
tacotron_teacher_forcing_final_ratio = 0.,
#final teacher forcing ratio. Relevant if mode='scheduled'
tacotron_teacher_forcing_start_decay = 10000,
#starting point of teacher forcing ratio decay.
#Relevant if mode='scheduled'
tacotron_teacher_forcing_decay_steps = 280000,
#Determines the teacher forcing ratio decay slope.
# Relevant if mode='scheduled'
tacotron_teacher_forcing_decay_alpha = 0.,
#teacher forcing ratio decay rate.
#Relevant if mode='scheduled'

#Wavenet Training
wavenet_random_seed = 5339, # S=5, E=3, D=9 :)
wavenet_data_random_state = 1234,
#random state for train test split repeatability

#performance parameters
wavenet_swap_with_cpu = False,

#train/test split ratios, mini-batches sizes
wavenet_batch_size = 8, #batch size used to train wavenet.
wavenet_synthesis_batch_size = 10 * 2,
wavenet_test_size = 0.0441,
wavenet_test_batches = None, #number of test batches.

#Learning rate schedule
wavenet_lr_schedule = 'exponential',
#learning rate schedule. Can be ('exponential', 'noam')
wavenet_learning_rate = 1e-4, #wavenet initial learning rate
wavenet_warmup = float(4000),
#Only used with 'noam' scheme. Defines the number of ascending
#learning rate steps.
wavenet_decay_rate = 0.5,
#Only used with 'exponential' scheme. Defines the decay rate.
wavenet_decay_steps = 300000,
#Only used with 'exponential' scheme. Defines the decay steps.

#Optimization parameters
wavenet_adam_beta1 = 0.9, #Adam beta1
wavenet_adam_beta2 = 0.999, #Adam beta2
wavenet_adam_epsilon = 1e-8, #Adam Epsilon
```

```
#Regularization parameters
wavenet_clip_gradients = False,
#Whether the clip the gradients during wavenet training.
wavenet_ema_decay = 0.9999,
wavenet_weight_normalization = False,
wavenet_init_scale = 1.,
#Only relevent if weight_normalization=True. Defines the initial scale
#in data dependent initialization of parameters.
wavenet_dropout = 0.05, #drop rate of wavenet layers

#Tacotron-2 integration parameters
train_with_GTA = False,
#Whether to use GTA mels to train WaveNet instead of ground truth mels.

#Eval sentences
sentences = ['WRITE YOUR SENTENCES HERE.'],
# From July 8, 2017 New York Times:

def hparams_debug_string():
values = hparams.values()
hp = ['  %s: %s' % (name, values[name])
for name in sorted(values) if name != 'sentences']
return 'Hyperparameters:\n' + '\n'.join(hp)
```

Bibliography

[1] Instute for language and speech processing. `www.ilsp.gr`.

[2] Lj-speech-dataset. `https://keithito.com/LJ-Speech-Dataset`.

[3] Munich artificial inteligence laboratories. `http://www.m-ailabs.bayern/en`.

[4] Nvidia developer blog. `https://devblogs.nvidia.com/nv-wavenet-gpu-speech-synthesis/`.

[5] Tacotron-2 implementation, tensorflow. `https://github.com/Rayhane-mamah/Tacotron-2`.

[6] N. Adiga, V. Tsiaras, and Y. Stylianou. On the use of wavenet as a statistical vocoder. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014.

[8] Kyunghyun Cho, van Merriënboer Bart, Gulcehre Caglar, Bahdanau Dzmitry, Bougares Fethi, Schwenk Holger, and Bengio Yoshua. 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, Doha, Qatar, Oct 2014. Association for Computational Linguistics.

[9] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in Neural Information Processing Systems*, volume 2015-January, pages 577–585. Neural information processing systems foundation, 2015.

[10] Van den Oord Aaron, Dieleman Sanderand, Zen Heiga, Simonyan Karen, Vinyals Oriol, Graves Alex, Kalchbrenner Nal, Senior Andrew, and Kavukcuoglu Koray. Wavenet: A generative model for raw audio. 2015.

[11] Fotinea S. E., Tambouratzis G., and Carayannis G. Constructing a segment database for greek time-domain speech synthesis. volume 3.

[12] Alex Graves and Jürgen Schmidhuber. 2005 special issue: Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Netw.*, 18(5-6):602–610, June 2005.

[13] Daniel W. Griffin and Jae S. Lim. Signal estimation from modified short-time fourier transform. In *ICASSP*, 1983.

[14] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

[15] Jurafsky and Martin. Speech and language processing, computational linguistics and speech recognition. 2009.

[16] L. Juvela, B. Bollepalli, V. Tsiaras, and P. Alku. Glotnet—a raw waveform model for the glottal excitation in statistical parametric speech synthesis. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(6):1019–1030, 2019.

[17] Nal Kalchbrenner, Erich Elsen, Karen Simonyan, Seb Noury, Norman Casagrande, Edward Lockhart, Florian Stimberg, Aäron van den Oord, Sander Dieleman, and Koray Kavukcuoglu. Efficient neural audio synthesis. *CoRR*, abs/1802.08435, 2018.

[18] H. Kawahara, I. Masuda-Katsuse, and A. Cheveigne. Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneousfrequency-based f0 extraction. *Speech Commun.*, 27(3-4):187–207, 1999.

[19] Simon King. An introduction to statistical parametric speech synthesis. *Sadhana, Indian Academy of Sciences*, 36(5):837–852, 2011.

[20] Simon King. A text-to-speech system: Festival. 2018.

[21] Diederik P. Kingma, Tim Salimans, and Max Welling. Improved variational inference with inverse autoregressive flow. *ArXiv*, abs/1606.04934, 2017.

[22] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, Ming Liu, and Ming Zhou. Neural speech synthesis with transformer network. 2019.

[23] Ranniery Maia, Masami Akamine, and Mark J.F. Gales. Complex cepstrum for statistical parametric speech synthesis. *Speech Commun.*, 55(5):606–618, June 2013.

[24] Masanori MORISE, Fumiya YOKOMORI, and Kenji OZAWA. World: A vocoder-based high-quality speech synthesis system for real-time applications. *IEICE Transactions on Information and Systems*, E99.D(7):1877–1884, 2016.

[25] Taylor Paul. Text-to-speech synthesis. 2009.

[26] Wei Ping, Kainam Peng, and Jitong Chen. Clarinet: Parallel wave generation in end-to-end text-to-speech. 30 July 2018.

[27] Carl Quillen. Kalman filter based speech synthesis. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2010*, pages 4618–4621. IEEE, 2010.

[28] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE.* IEEE, 1989.

[29] Prajit Ramachandran, Tom Le Paine, Pooya Khorrami, Mohammad Babaeizadeh, Shiyu Chang, Yang Zhang, Mark A. Hasegawa-Johnson, Roy H. Campbell, and Thomas S. Huang. Fast generation for convolutional autoregressive models. *CoRR*, abs/1704.06001, 2017.

[30] Shautya Rohatgi and Maryam Zare. Deepnorm - a deep learning approach to text normalization. 2017.

[31] A-V. I. Rosti and M. J. F. Gales. Generalized linear gaussian models. Technical Report CUED/F-INFENG/TR.420, University of Cambridge, Department of Engineering, 2001.

[32] Sam Roweis and Zoubin Ghahramani. A unifying review of linear gaussian models. *Neural Computation*, 11(2), 1999.

[33] Karabetsos S., Tsiakoulis P., A. Chalamandaris, and Raptis. *Text, Speech and Dialogue.* Springer, 2008.

[34] Tim Salimans and Diederik P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *CoRR*, abs/1602.07868, 2016.

[35] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45:2673–2681, 1997.

[36] J. Shen, R. Pang, Ron J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, RJ Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. 2017.

[37] Richard Sproat, Alan W. Black, S. Kumar S. Chen, M. Ostendorfk, and C. Richards. Normalization of non-standard words. 2001.

[38] Richard Sproat and Navdeep Jaitly. Rnn approaches to text normalization: A challenge. 2017.

[39] Ilya Sutskever, Oriol Vinyals, and Quoq V. Le. Sequence to sequence learning with neural networks.

[40] Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *CoRR*, abs/1705.08209, 2017.

[41] Akira Tamamori, Tomoki Hayashi, Kazuhiro Kobayashi, Kazuya Takeda, and Tomoki Toda. Speaker-dependent wavenet vocoder. In *INTERSPEECH*, 2017.

[42] Paul Taylor. *Text-to-Speech Synthesis*. Cambridge University Press, 2009.

[43] Salimans Tim, Karpathy Andrej, Chen Xi, and Kingma Diederik P. Pixel-cnn++:improving the pixelcnn with disretized logistic mixture likelihood and other modifications. 2017.

[44] Tomoki Toda and Keiichi Tokuda. Speech parameter generation algorithm considering global variance for hmm-based.

[45] K Tokuda, T Kobayashi, and S Imai. Speech parameter generation from hmm using dynamic features. *ICASSP*, 1995.

[46] Vassilis Tsiaras, Ranniery Maia, Vassilis Diakoloukas, Yannis Stylianou, and Vassilis Digalakis. Linear dynamical models in speech synthesis. Technical report, Technical University of Crete, School of Electronic Technical University of Crete and Toshiba Research Europe Limited, Cambridge Research Laboratory, Cambridge, UK, 2014.

[47] Vassilis Tsiaras, Ranniery Maia, Vassilis Diakoloukas, Yannis Stylianou, and Vassilis Digalakis. Towards a linear dynamical model based speech synthesizer. In *Proceedings of the Interspeech*. IEEE, 2015.

[48] Aäron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis C. Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. Parallel wavenet: Fast high-fidelity speech synthesis. *CoRR*, abs/1711.10433, 2017.

[49] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jacob Uszkoreit, Llion Jones, Aidan N. Gomez, and Lukasz Kaiser. Attention is all you need. 2017.

[50] H Zen, K Tokuda, and T Kitamura. Reformulating the hmm as a trajectory model by imposing explicit relationships between static and dynamic feature vector sequences. *Computer Speech and Language*, 21(1):153–173.

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