POLITECNICO DI MILANO

Industrial and Information Engineering School

Master's Degree Course in
Computer Engineering

Emotron: an expressive Text-To-Speech

Supervisor:	Prof. Licia Sbattella
Co-supervisors:	Ing. Roberto Tedesco, Ing. Vincenzo Scotti

Degree thesis by

Alexander Sukhov	Matr. 916187
Cristian Regna	Matr. 920977

Academic Year 2019 - 2020

Politecnico di Milano:
www.polimi.it

Industrial and Information Engineering School:
www.deib.polimi.it/ita/home

Contents

List of Figures

List of Tables

Listings

Abstract

The goal of this work is the design of a Text-To-Speech (TTS) tool, able to express emotions. In the thesis, we will present the various methodologies for the development of a classical speech synthesis. We will continue by analyzing the new models of TTS based on Neural Networks. We will see how elements such as prosody and intonation can be integrated within a Neural Network and which results can be obtained. Then, we will dive into expressive speech analysis and will justify several approaches through which we can influence expressiveness into neutral speech. We will continue with the presentation of our model created for the transfer of prosody able to make the spoken text very fluent and with some hints of emotion. The result obtained therefore makes us understand how one of the possible solutions for the implementation of a Text-To-Speech with emotions is through the transfer of these rhythmic elements of the accents. The model obtained can generate, given the input the sentence to be pronounced and the type of emotion, a very fluent and expressive voice.

Abstract

L'obiettivo di questo lavoro è la progettazione di uno strumento Text-To-Speech (TTS), in grado di esprimere emozioni. Nella tesi presenteremo le varie metodologie per lo sviluppo di un Text-To-Speech classico. Proseguiremo analizzando i nuovi modelli di Text-To-Speech basati sulle Reti Neurali.Vedremo come elementi come la prosodia e l'intonazione possono essere integrati all'interno di una rete neurale e quali risultati possono essere ottenuti. Quindi, in particolare faremo un analisi espressiva del discorso e giustificheremo i diversi approcci attraverso i quali è possibile influenzare l'espressività in un discorso neutro. Proseguiremo con la presentazione di un nostro modello creato per il trasferimento della prosodia in grado di rendere il testo parlato molto fluente e con qualche accenno di emozione. Il risultato ottenuto quindi ci fa capire come una delle possibili soluzioni per l'implementazione di un Text-To-Speech con le emozioni sia attraverso il trasferimento di questi elementi ritmici degli accenti. Il modello ottenuto è in grado di generare, dato in input la frase da pronunciare e il tipo di emozione, della voce molto fluente ed espressiva.

Chapter 1

Introduction

1.1 Motivations

Since the beginning of the 2000s, we have observed an explosion in the computer science field that has led us to experience a new industrial revolution. Technology has now become an integral part of everyday life. Whatever task we accomplish, whether inside or outside our homes, is completely different from 10 years ago. AI[1] is rapidly changing every market sector and it is legitimate to wonder how much it can help us in important areas as Natural Language Processing. There are several applications of Natural Language Processing and we have decided to focus on text and speech processing, and in particular Expressive Text-to-speech task.

Text-To-Speech makes the online world and beyond available to people with learning disabilities, visual impairments, and literacy challenges. Well-designed applications, websites, and services give all users equal access to information and functionality. One of the tools that help make this happen is Text-To-Speech (TTS[2]). Deploy this kind of applications that turn text-based content into audio and enable all your users to access information and communicate with others, there are many other fields where a TTS is very useful including, Health Care, Home Automation, Learning, Robotics, and Telecommunication.

Expressive Text-To-Speech is a hot topic in the NLP area because there is still no gold standard of how to implement it and there is much application for this area because only expressive TTS can in a full way duplicate human voice.

[1]Artificial Intelligence
[2]Text-To-Speech

1.2 Aims

The goal is to develop a Text-To-Speech tool able to produce a very fluent and understandable spoken representation and above all capable to express emotion. This need arose when we listen to the result produced by the recent Text-To-Speech they give the appearance of having a neutral and emotionless voice.

As this technology is becoming more and more popular, making it capable of transmitting emotions would take the user experience on another level. However, the task of influencing neutral speech with emotions is challenging and requires a lot of time and effort as there are many baseline speech synthesis models and we have several ideas on how to adapt the emotional module to them. All these combinations should be trained and tested and many of them might not give good results or they can completely fail, so there is enormous work ahead of us.

1.3 Outline

Below are reported the different contents of each chapter.

In the second chapter, we will explore the fundamental concepts and the first technologies developed not based on today's neural networks, we will also describe in detail and how the neural networks used in today's TTS are composed.

In the third chapter we will talk about the state of the art. We will go on to talk in detail about today's technologies used for the construction of a TTS, currently, the best systems are those that use neural networks and we will give an explanation of the various layouts that form these neural networks and why they are particularly effective in this task.

In the fourth chapter, we will analyze the datasets used for the training of neural networks. These datasets contain audio file phrases spoken by different speakers with the accompanying transcription. These datasets can be of various types, from containing audios that are spoken by professional speakers or taken from famous TV series, or simple conversations. We will highlight which datasets are best suited to our use and we will explore the preprocessing techniques we had to apply to the different datasets.

In the fifth chapter, we will explore the possible emotional architectures to be used together with the different models to produce speech. We will explain the choices made on one module rather than another, arriving at the final model.

In the sixth chapter we will present the model we created specifying the different layouts and the operations performed. We will introduce and explain a new metric used specifically to manage emotions called style loss.

In the seventh chapter we will show all the steps applied to our model for training, both for speech generation and the classification of emotions. We will calculate the necessary metrics and tuned the hyperparameters to be able to comment the results obtained.

In the eighth chapter, we will explain the approach used to evaluate our model to compare it with the current state of the art and we will show the results obtained.

Finally, the ninth and final chapter will cover the conclusions commenting on the results obtained and possible future works.

Chapter 2

Background

In this section, we will provide all necessary background information required for understanding the TTS area. Firstly we will shortly discuss general Text-To-Speech technics. Then, classical technics such as diphone and unit selection will be described in detail. For understanding State-of-the-art technics, we will provide an overview of Neural networks, Recurrent Neural networks, Highway networks, and residual connections.

Finally, we want to discuss the Tacotron model as it was state-of-the-art before the updated Tacotron2 was introduced. It is a really important model as many components and brand new technics were implemented by Google researchers and the general pipeline of Tacotron 2 was adopted from this model.

2.1 General overview of TTS

The modern task of speech synthesis, also called Text-To-Speech or TTS, is to produce speech (acoustic waveforms) from text input.

Synthesizers are used to conduct dialogues with people and are very important in non-conversational applications that speak to people, such as in devices that read out loud for the blind, or in video games or children's toys. Finally, speech synthesis can be used to speak for sufferers of neurological disorders, such as astrophysicist Steven Hawking who, having lost the use of his voice due to ALS[1], speaks by typing to a speech synthesizer and having the synthesizer speak out the words. State-of-the-art systems in speech synthesis can achieve remarkably natural speech for a very wide variety of input situations, although even the best systems still tend to sound wooden and are limited in the voices they use.

In an overview of speech synthesis technology[2] a generic description of TTS is exposed and it also explained the current structure of TTS with all its variants. When we approach TTS, we can divide it into two essential blocks: NLP[2] and DSP[3] (Figure 2.1).

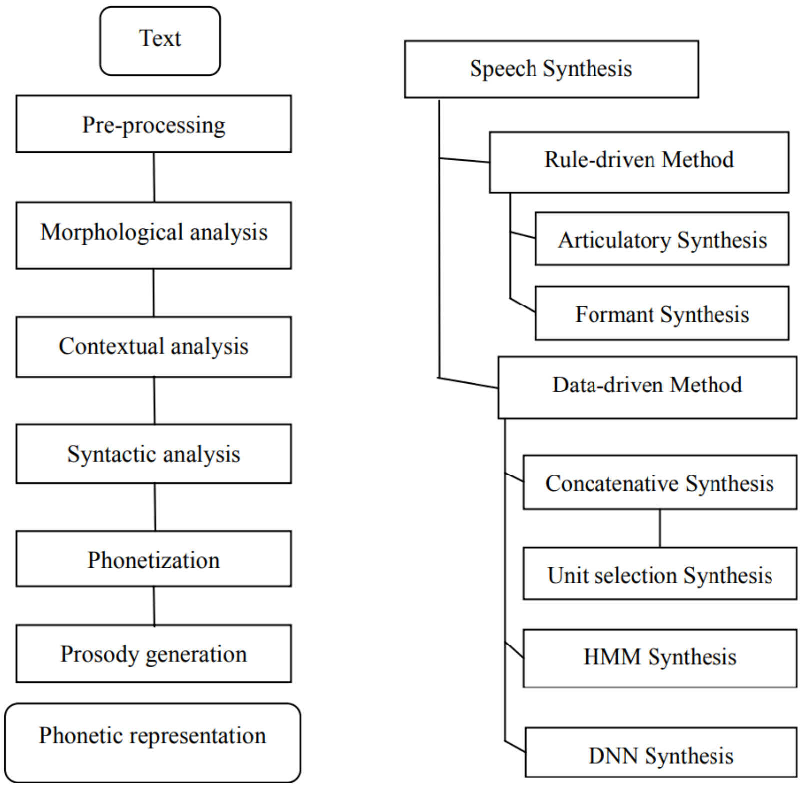


Figure 2.1: NLP and DSP Block.

[1]Amyotrophic lateral sclerosis
[2]Natural Language Processing
[3]Digital Signal Processing

The first one is the Natural language processing block; his role is to take the input text and transcribed it into a phonetic representation. The second block is Digital signal processing and his role is to generate the speech sound, this element is generated from the phonetic representation of the NLP block. The NLP block can be considered as the number of steps followed to get the phonetic representation and is composed of these elements:

1. The Pre-Processing where "Regularization" operations are done (convert abbreviation, acronyms turn into full text, etc.)
2. The Morphological Analysis and Contextual Analysis that work together, in this step the tokenization is applied. if we have some problem related to some particular language the contextual analysis is performed.
3. The Syntactic analysis and his main goal is to use the information of the precedent step to predict the prosody.
4. The phonetization block generates the sequence of phonetic symbols for each word. A pronunciation lexicon should be constructed first, and the phonetic symbols of each word should be produced according to the lexicon.
5. The Prosody generation that will generate the prosodic features which include pitch curve, duration and pause information, stress and rhythm features. This will influence the smoothness of the TTS.

Then we have the DSP block, which is composed of two main blocks: the rule-driven methods and data-driven methods. Rule-driven methods simulate the speech voice according to the rules deduced from the acoustic process while the principle of data-driven methods is to generate the speech by recorded speech data or by the statistical parameters got from speech data. The two approaches of Rule-driven synthesis methods are articulatory synthesis and formant synthesis while the main approaches of Data-driven synthesis methods are concatenative synthesis, unit selection synthesis, HMM[4] synthesis, and DNN[5] synthesis.

ě5mm **Rule-based**
Articulatory Synthesis is a direct simulation of the physical process of human pronunciation. Its advantage is very flexible, but its main disadvantage is the sound quality is not good. The second one is the Formant Synthesis that is a simulation of the acoustic process. Formant is the resonance frequency area of the vocal tract and it is based on the source-filter model of speech production. In this model, speech sound is produced by two kinds of sound sources: the source of a voiced sound and the source of unvoiced sounds is a random noise generator. Then, the voiced and unvoiced sounds generated from sound sources are modified by the vocal tract model.

[4]Hidden Markov model
[5]Deep neural networks

Data-driven
The Concatenative system generates the voice connecting small recorded voice, then the utterance is modified by the prosodic model, the sound quality is very good. there can be some problem with memory because the system is more natural if we have longer units since less concatenation is needed, but if we have shorter units the sound quality will become bad, the disadvantages are the discontinuities of unit boundaries and can lead to some unstable result. In this case, the Unit Selection can fix the problem.

Unit Selection is similar to Concatenative Synthesis but solves unnaturalness by storing multiple instances of each unit with a different prosodic feature, However, this method also has the same problems of selecting error units and a huge speech corpus is needed.

Since we can have a problem of database size in Unit Selection (need a big corpus) we can use a parametric synthesis to infer the acoustic parameter from speech data. in this case, less memory is needed to store the parameter (we're not storing anymore all the corpus). HMM is a parametric model and consist of two phases:

1. The training phase is similar to the one used in speech recognition, we extract the feature and we model this vector of features depending on the context (grammatic, prosody).

2. Synthesis filter transforms the feature vector in acoustic signal (Synthesis phase). this allows more flexibility with respect to Unit Selection but can lead to some limitations on synthesized speech.

In an HMM-based speech synthesis system, some limitations would degrade the accuracy of acoustic models and the quality of synthesized speech. DNN was introduced to solve this problem, thanks to the hidden layer we are trying to learn the more useful feature. DNN could be used to models the features of text and speech sounds and output the vectors of phonetic signals, but on the other side, the power computation is bigger than HMM.

Rule-driven synthesis methods are used less today because the synthesized speech sounds are not good enough. But their advantages are the low processing costs and high flexibility. Data-driven synthesis methods nowadays are the dominant methods because they can provide a higher quality of synthesized speech. Unit selection synthesis could generate the best speech but it also cost the largest memory and data. Compared with Concatenative synthesis and Unit Selection synthesis, HMM, DNN, and other statistical synthesis techniques could produce similar natural speech but only need fewer speech data and memory, they achieve the best balance at the present time.

2.2 Classical approaches for speech synthesis

Diphone Waveform synthesis

Given an internal representation of the input text, the purpose is to generate a waveform. The diphone concatenative synthesis generates waveform by concatenating phones which are taken from a prerecorded database of diphones.[38] a Diphone is a unit similar to phoneme with the difference that it starts somewhere in the middle of one phoneme and ends in the middle of the following one.
The process can be characterized by two processes:

- The training process consists of recording each diphone with a single speaker and cutting each diphone out of speech and store them into the diphone database.
- The synthesis process consists of retrieving diphones from a database, concatenate them, and use signal processing.

We tend to use diphones instead of phones because each phone slightly depends on the previous and following one. This phenomenon is called coarticulation.It can cause discontinuity and as it can be seen from the figure below, the same phoneme "eh" has a different structure in its beginning and ending, but in the middle part it is more or less stable, so it is better to concatenate from the middle of the phone (Figure 2.2).

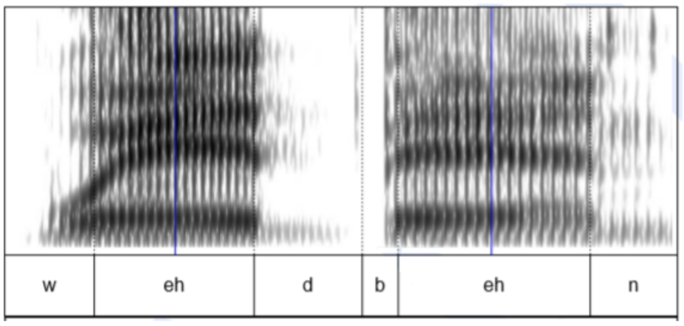


Figure 2.2: Waveform example.

To build a diphone database we need to perform the following steps:

1. Create a diphone inventory
2. Hire a speaker
3. Develop text for the speaker to read for each diphone
4. Record the speaker reading each diphone
5. Segment, label, and pitch-mark the diphones
6. Excise the diphones

Diphone inventory is the process of eliminating part of diphones. The idea is that by combining each possible phoneme with the same number of phones, we will get a huge number of possible diphones (pairwise phonemes). However, some of the pairs of phonemes are never used with each other in speech. So diphone inventory is the process of deleting this useless pair. Steps from second to fourth are required only a professional speaker, which can be called voice talent, professional microphone, and studio or silent location. Voice talent should be careful with recording as the same diphones in the speech should have a constant pitch, energy, and duration. In step five we need to label and segment two phones to make up each diphone. If we run speech recognizer in force alignment mode, we can achieve this. Usually, they are not quite precise in finding phoneme boundaries, so hand-correction is needed. After having all the phonemes in the text, we can split it up on diphones and set boundaries at 50% at all the phonemes, except for stop ones, which will have 30% of the way into the phone.

For getting to the next step, the following assumptions must be followed: input text is normalized and we have it as a sequence of diphones and prosodic targets. And also, we got the appropriate sequence of diphones from a database, the next task is to concatenate the diphones and modify pitch, energy, and duration to match intermediate representation.

To concatenate two diphones we firstly need to apply the windowing function as if the edges of them are different, we will be hearing the sound of a click. The second step is to be sure that the pitch period of the end of the first diphone should be the same as the beginning of the second one.

For detecting pitch period, we need an accurate way and it is done by computing glottal pressure. The traditional and most accurate way is to use an electroglottograph while recording speech. However, it is impossible to pitch-mark something that has already been collected, if it was recorded without EGG[6]. Some modern approaches are used to avoid using EGG, and nowadays their accuracy almost as good as classical EGG.

Finally, given pitch-marked corpus, with TD-PSOLA[7] we can modify the pitch and duration of waveform via extraction of frame per pitch. For example, to change the duration of the diphone, we just insert extra copies of some pitch-synchronous frames and duplicating the piece of a signal. It can be seen in the Figure 2.3.

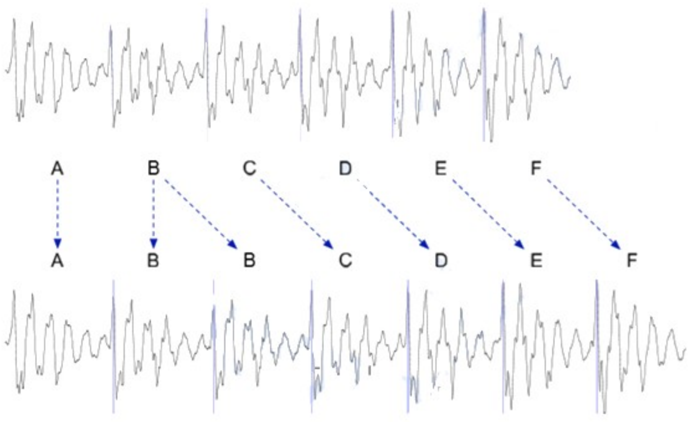


Figure 2.3: TD-PSOLA.

[6]Electroglottograph
[7]Time-Domain Pitch-Synchronous Overlap and Add

Unit Selection

Unit selection is introduced since diphone waveform synthesis has some problems. The first problem is that the stored diphone inside the database after pitch modification may contain some artifacts which make the speech unnatural. The second problem is related to what the diphone analysis can capture, the result is only the coarticulation but there are many global effects on phonetic realization.

Modern synthesizers are based on Unit Selection synthesis[38], is a kind of concatenative synthesis algorithm like diphone synthesis, there are different step respect the standard concatenative:

- The database of diphone synthesis stores exactly one copy of each diphone, while in Unit Selection the database contains many copies of each diphone.
- In diphone synthesis, the prosody is modified by TD-PSOLA while in unit selection signal processing is not applied.

Unit selection requires a bigger amount of space, but the benefits are high. Entire words or phrases of the utterance may be present in the database leading to a natural waveform. Besides, in cases we can't find large chunks an individual diphone can be used and since there are many copies of each diphone, the one that can fit naturally is probably found.
The architecture of unit selection can be summarized assuming that:

- A database of units is given (assume that are all diphone but can be half-phones and syllables).
- Characterization of the target known as "internal representation" is given.

The goal is to select the best sequence of diphone units from a database that corresponds to the target representation. The best sequence can be the one that meets these conditions:

- Each diphone unit we select coincide with the specification of the target (F0, stress level, etc)
- Each diphone unit concatenates smoothly with its neighboring units

Only ideally a unit can exactly meet one of these requirements. A gradient version of these constraints is used to find the sequence of the unit which minimize the target cost and the join cost.

- Target cost $T(u_t, s_t)$: how well the target specification s_t matches the potential unit u_t
- Join cost $J(u_t, u_{t+1})$: how well the potential unit u_t matches the potential neighbor unit u_{t+1}

High values of cost mean bad matches and join. The task of unit selection is:

$$U = \arg\min_{U} \sum_{t=1}^{T} T(u_t, s_t) + \sum_{t=1}^{T-1} J(u_t, u_{t+1}) \tag{2.1}$$

The target cost measure how well the unit matches the target diphone specification. Can be represented as a vector of diphone specification (feature) like stress, F0 value, content/function word, etc. Assume that for each dimension p we can come up with some sub cost. Each sub cost is related to a feature (can be 0 or 1) and this feature is weighted (w_p) since some feature is more important than the other one, assume that they are independent and additive. Now the target cost should be:

$$T(u_t, s_t) = \sum_{p=1}^{p} w_p T_p(s_t(p), u_j(p)) \tag{2.2}$$

The joining cost is a function of two units from the database, but to get this value we measure the acoustic similarity of the edges of the two units that will be joined. We compute the join cost as the sum of sub cost weight:

$$J(u_t, u_{t+1}) = \sum_{p=1}^{p} w_p J_p(u_t(p), u_{t+1}(p)) \tag{2.3}$$

Hunt and Black algorithm use three sub costs, the cepstral distance, the absolute differences in log power, and F0. If two units selected were consecutive diphones in the unit database the join cost is set to 0. This is an important feature of unit selection synthesis since it encourages large natural sequences of units to be selected from the database.

The standard method is to think of the unit selection problem as a Hidden Markov Model. The target units are the observed outputs, and the units in the database are the hidden states. Our job is to find the best-hidden state sequence.

Machine learning can be used to find the best weight for join and target cost but setting the weight by hand reaches a better result. The method of Hunt and Black holds out a test set of sentences from the unit selection database. For each of these test sentences, we take the word sequence and synthesize a sentence waveform (using units from the other sentences in the training database). Now we compare the acoustics of the synthesized sentence with the acoustics of the true human sentence.

Now we have a sequence of synthesized sentences, each one associated with a distance function to its human counterpart. Now we use linear regression based on these distances to set the target cost weights to minimize the distance. An alternative approach is to map the target unit into some acoustic space, and then find a unit that is near the target in that acoustic space.

2.3 Neural Network

Feed Forward Neural Network

Artificial Neural Network[1] is a mathematical model inspired by the biological neural network, usually, it is used to solve classification or regression problems. The imitation of the human brain has been considered as a solution because it is:

- Able to learn
- Fault-tolerant
- Highly parallel
- Able to deal with noise

The neurons are electrically excitable and the information is transported between them in form of electrical stimulation. The incoming information is added up, and if the stimulus has reached a threshold, the information will be passed to other neurons. In this case, the neuron is said to be activated. Otherwise, if the stimulation is too low, the information will not be passed to other neurons. The first model inspired by a single neuron is the perceptron, it is composed of inputs, weights, an activation function, and an output (Figure 2.4).

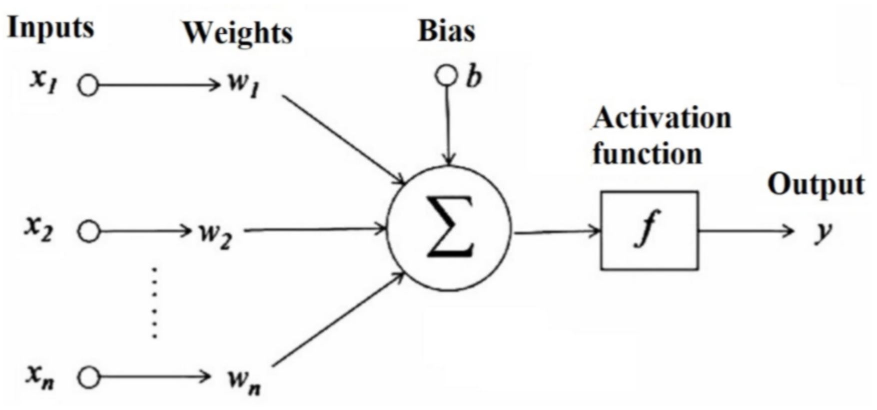


Figure 2.4: Perceptron architecture.

The perceptron works in the same way as a biological neuron. The input is sent to the perceptron and multiplied by the respective weight, then the bias is added. Finally, the activation function compares the resulting value z with a threshold and if the value exceeds the threshold the neuron will be activated and it will send an output to the connected neurons. Here is the formula:

$$z = \sum_j w_i \cdot x_j + b \tag{2.4}$$

$$y = g(z) \tag{2.5}$$

There are many activation function and the most used are shown below.
Step function:

$$g(z) = \begin{cases} 1 & \text{if z} \geq 0 \\ 0 & \text{if z} < 0 \end{cases} \tag{2.6}$$

Sign Function:

$$g(z) = \begin{cases} 1 & \text{if z} > 0 \\ 0 & \text{if z} = 0 \\ -1 & \text{if z} < 0 \end{cases} \tag{2.7}$$

Sigmoid function:

$$g(z) = \frac{1}{1+e^{-z}} \tag{2.8}$$

Hyberbolic tangent:

$$g(z) = \frac{e^z + e^{-z}}{e^z + e^{-z}} \tag{2.9}$$

The learning process for Perceptron is called Hebbian learning and can be summarized by the following rule:

$$\begin{aligned} w_i^{k+1} &= w_i^k + \boldsymbol{\Delta} w_i \\ \boldsymbol{\Delta} w_i &= \eta \cdot t \cdot x_i \end{aligned} \tag{2.10}$$

Where:
η : learning rate
x_i: i_{th} perceptron's input
t: the desired ouput
k: current step of the algorithm

And the steps are the following:

1. Start from random weights
2. Show one record (i.e. set x_i values to the ones of the record)
3. If wrong $w_i = w_i + \Delta w_i$
4. Go to next record (point 2)

The most used activation function for the perceptron is the sign function, but the problem is that perceptron allows the classification of only linearly-separable elements. Feed-Forward Neural Network was introduced to resolve the perception problem. It is a set of neurons connected according to the topology. There are different elements inside the network:

- Layer: neurons at the same distance from the input neurons.
- Input layer: the layer of neurons that receives as input the data to process.
- Output layer: the layer of neurons that gives the final result of the network.

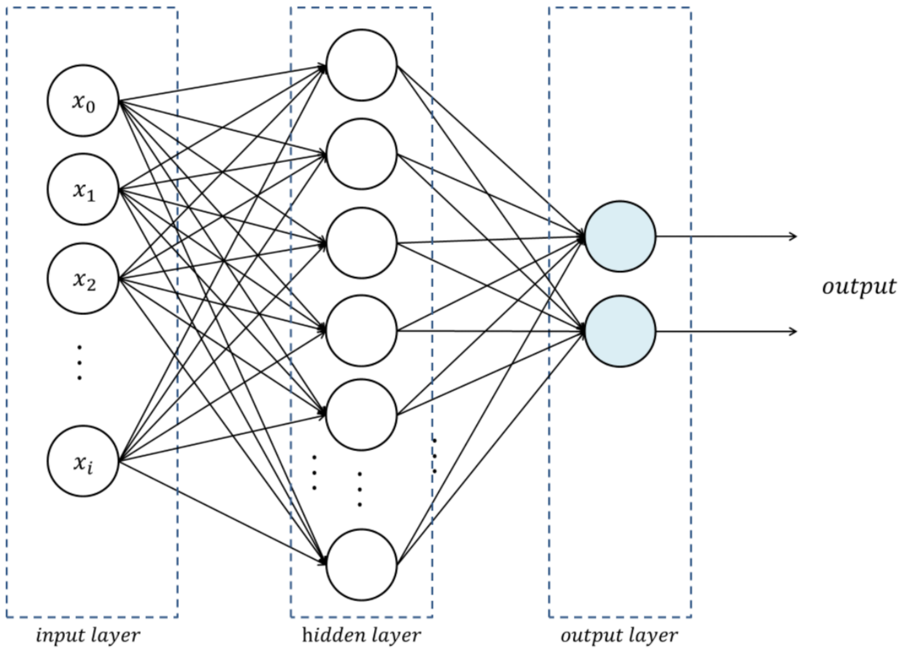


Figure 2.5: Architecture design.

- Hidden layer: a layer of neurons that processes data from other neurons to be processed by other neurons.

Sometimes it is not easy to minimize or maximize analytically a function, so we need to apply a numerical approach: the gradient descent.
It performs the following steps:

1. Pick up a possible solution w_0 (at random)
2. Compute the function derivative ∂E ∂w
3. Update the solution $w_{k+1} = w_k - \eta \frac{\partial E}{\partial w}\Big|_k$
4. Repeat 2 and 3 until convergence

The process of learning in a Feedforward Neural Network is called Backpropagation, which is the use of gradient descent to iteratively minimize the network error (Figure 2.6).

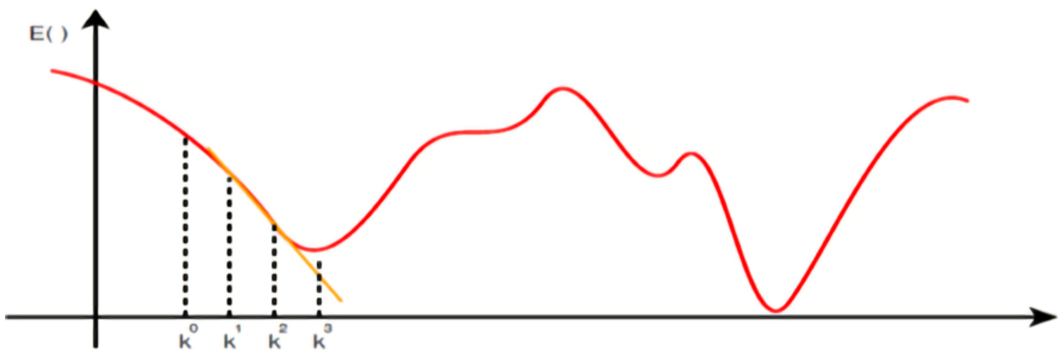


Figure 2.6: Error trend.

Backpropagation algorithm

Backpropagation is a core algorithm of how neural networks learn (Figure 2.7). The core of the algorithm is an expression of the partial derivative of a cost function concerning weights and bias. The result tells us how quickly cost changes when we are tweaking the weight.

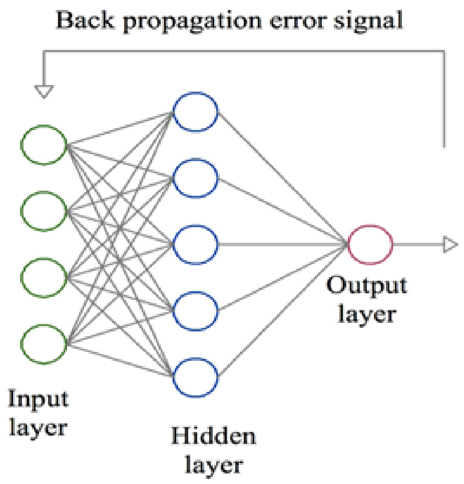


Figure 2.7: Backpropagation.

But activations could not be changed directly, they are direct calculations of weights and biases. This means that if we indirectly correctly adjust these parameters, we could get desired output. Before considering the algorithm we need to define some notations:

- L: Last layer
- L-1: Second Last layer
- l: layer
- y: output value we wish to predict
- C: cost function
- w: weights
- a: activations
- b: bias

We need to move backward in the network and update weights and biases:

$$
\begin{aligned}
\frac{\partial C}{\partial w^{(L)}} &= \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}} \\
\frac{\partial C}{\partial b^{(L)}} &= \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial b^{(L)}} \\
\frac{\partial C}{\partial a^{(L-1)}} &= \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}}
\end{aligned}
\tag{2.11}
$$

All of these equations are the measure of the ratio of how a particular weight, bias, activation impact the final cost function, which we are optimizing. During learning, in each part, we are stepping in the direction of the result of these derivations. Gradient vector keeps these partial derivatives of weights and biases. Activations are also a good measure of weights, but we do not keep them in gradient vectors. Then we compute the gradient using a mini-batch of our data. For each observation in the mini-batch, we averaging the output of each weight and bias. This averaged result becomes the output of the gradient, which in turn, makes a step towards the best direction over the mini-batch size.

Then, after each mini-batch, we update weight and bias in the following way:

$$\begin{aligned} w(l) &= w(l) - \eta \frac{\partial C}{\partial w(l)} \\ b(l) &= b(l) - \eta \frac{\partial C}{\partial b(l)} \end{aligned} \tag{2.12}$$

However, the three equations shown before (2.11), are used only for the output layer, to solve backpropagation, we should do the same procedure for all previous layers making more partial derivatives and adjust weights and biases for them.

Convolutional Neural Network

A CNN[8] is used for image processing. The input of a convolutional neural network is an image in 3 dimensions (width, height, and depth). It is composed of hidden layers and an output layer but the layers have neurons arranged in 3 dimensions.

The typical architecture of a convolutional neural network is shown in Figure 2.8. CNN are typically made of blocks that include:

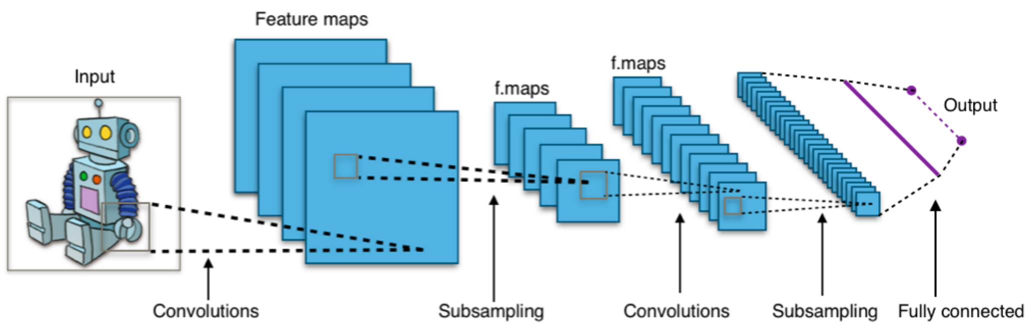


Figure 2.8: CNN architecture.

- Convolutional Layers
- Nonlinearities (activation function)
- Maxpooling

An image passing through a CNN is transformed in a sequence of volumes, as the deep increases, the height, and width of the volume decrease. The convolutional layers apply the convolution operation, in which a kernel slides over the input feature map, and at each kernel position, the elementwise product is computed between the kernel and the overlapped input subset, then the result is summed up (Figure 2.9). Using different kernel lead to different effect. Therefore, convolutional layers are described by a set of filters, that represent the weights of these linear combinations.

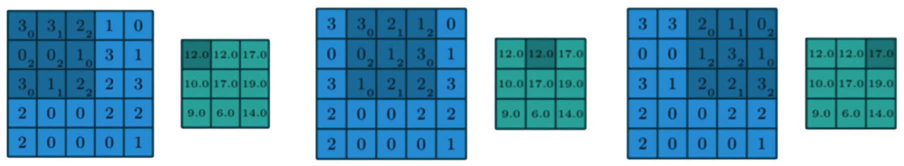


Figure 2.9: Computation details.

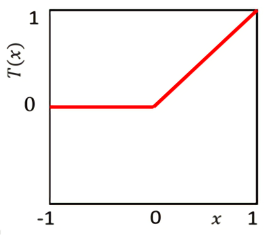
The filter is smaller than the input data and the type of multiplication applied between a filter-sized patch of the input and the filter is a dot product. A dot product is an element-wise multiplication between the filter-sized patch of the input and filter, which is then summed, always resulting in a single value. Because it results in a single value, the operation is often referred to as the "scalar product".

[8]Convolutional Neural Network

The activation function used in CNN is the ReLu[9] (Rectifier Linear Units)(Figure 2.10).

$$T(x) = \begin{cases} x & \text{if x} \geq 0 \\ 0 & \text{if x} < 0 \end{cases}$$

(a) ReLu equation.



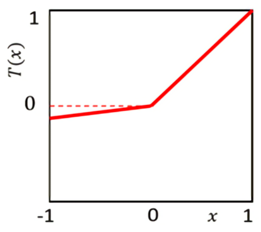
(b) ReLu graphic.

Figure 2.10: Relu activation function

The use of ReLU has a disadvantage: if the weights learned are such that the x is negative for the entire domain of inputs, the neuron never learns. This is known as the dying ReLU problem. The dying ReLU problem can be tackled by using a modified version of the activation function, called Leaky ReLU (Figure 2.11).

$$T(x) = \begin{cases} x & \text{if x} \geq 0 \\ 0.01 \cdot x & \text{if x} < 0 \end{cases}$$

(a) Leaky ReLu equation.



(b) Leaky ReLu graph.

Figure 2.11: Leaky ReLu activation function

Pooling Layers reduce the spatial size of the volume and can operate independently on every depth slice of the input and hence control overfitting, often using the MAX operation obtaining a scaled-down image which keeps the convolution effect. In the end, there is a Fully connected layer that has the same size as the number of classes and provides a score for the input image to belong to each class.

[9]Rectifier Linear Unit

2.4 Recurrent Neural Network

Classical neural networks can deal only with static datasets. When we want to deal with dynamic data, we can use either memoryless models such as autoregressive or feedforward neural networks or we can use memory-based models such as linear dynamic systems, hidden Markov models, or recurrent neural networks(RNN[10]).

In RNN, memory is introduced through a context vector and output depends on both current input and previous context. It can help when we need to remember something from the past. RNN is proved to be Turing-complete.

The output of an RNN at step t is passed to the network at step $t + 1$. In such a way, an RNN can be seen as multiple copies of the same network, each one passing information to the network at the next time step. This characteristic is shown in Figure 2.12. Therefore, the output at step t is predicted considering the information at step $t - 1$.

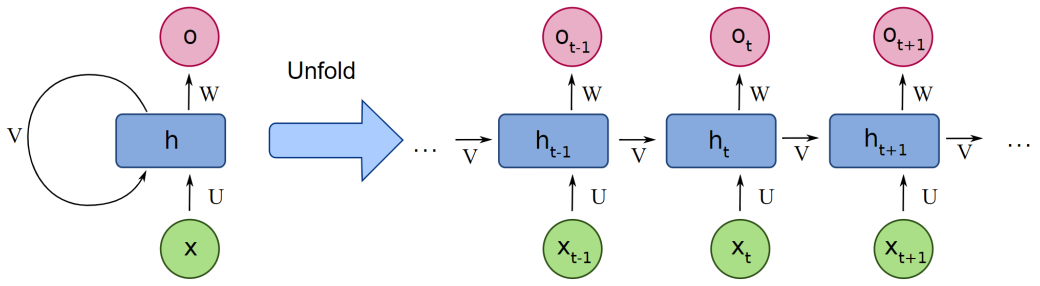


Figure 2.12: Recurrent NN.

Taken together, all of these considerations lead to a two-pass algorithm for training the weights in RNNs [28]. In the first pass, we perform forward inference, computing h_t, y_t, and accumulating the loss at each step in time, saving the value of the hidden layer at each step for use at the next time step. In the second phase, we process the sequence in reverse, computing the required error terms gradients as we go, computing and saving the error term for use in the hidden layer for each step backward in time. This general approach is commonly referred to as Backpropagation Through Time. Due to its properties, different type of RNN (Figure 2.13 on the next page) can solve a big spectrum of problems such as simple image classification, music generation, sentiment classification, machine translating, and many others.

The main advantages of RNN are the possibility to process data of any length, model size is not increasing with increasing size of the input, the computation can take historical information into account, weights are shared across time. However, during long backpropagation, with the classical one-gate model, there appears one problem, which could not be solved for a long time. This problem is a vanishing/exploding gradient. The reason for this is that it is difficult to capture long-term dependencies due to multiplicative gradients which will change a lot in some layers.

[10]Recurrent neural network

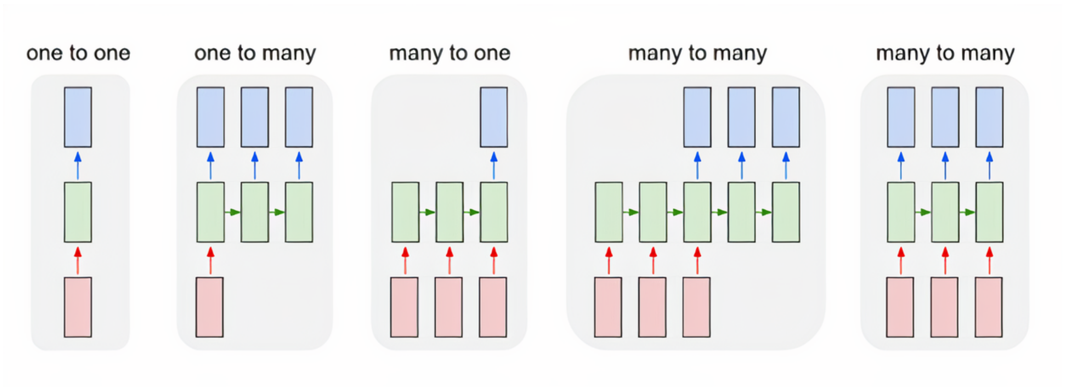


Figure 2.13: RNN types.

One way to solve gradient explosion is to use gradient clipping in which we are setting the maximum value for the gradient as shown in Figure 2.14.

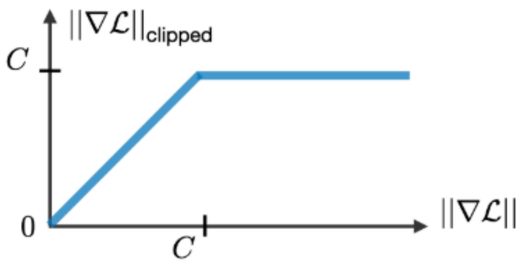


Figure 2.14: Gradient clipping.

Luckily, all the gradients problems can be solved using Long-Short-Term memory networks. (Figure 2.15) Instead of using only one gate, it is using several gates, which can filter out info passing through. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions.

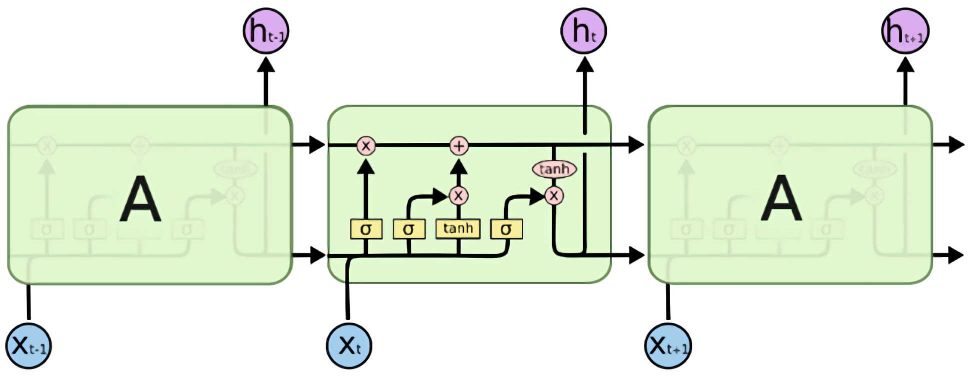


Figure 2.15: LSTM architecture.

Following the pipeline, the flow of information will encounter four "filters": forget, input-output gates, and cell state.

Forget gate is responsible for the decision of keeping or throwing away knowledge. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Input gate is getting previous hidden state and current input into a sigmoid function which will output the importance of values, where 0 is not important and 1 is important. Cell state is pointwise multiplied by the forget vector. This opens a possibility of dropping values in the cell state if it gets multiplied by values near 0.

Then we perform pointwise addition which will update cell state. The output gate is responsible for the next hidden state to which we will go to. The hidden state is used for predictions and also contains information about previous input. The new cell state and the new hidden state are then carried over to the next time step. Gated recurrent unit (Figure 2.16) combines the forget and input gates into a single update gate. This means that the usage of previous memory and getting the new information into the memory in the GRU[11] is not independent of LSTM[12].

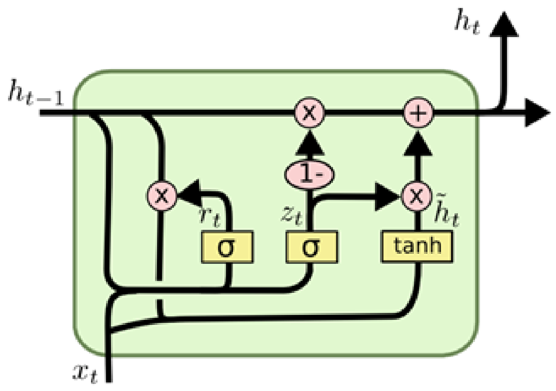


Figure 2.16: GRU cell.

Another difference between the structures is the lack of the cell state in the GRU. While the LSTM stores its longer-term dependencies in the cell state and short-term memory in the hidden state, the GRU stores both in a single hidden state. However, in terms of effectiveness in retaining long-term information, both architectures have been proven to achieve this goal effectively.

[11]Gated recurrent units
[12]Long short-term memory

2.5 Seq2Seq and Attention mechanism

Sequence-to-sequence neural network architectures involve two RNNs [27]. We start with the source sentence and after proceeding with word embedding, which is process of converting textual input into representational vector space, we feed this to encoder RNN. Encoder RNN produces an encoding of the source sentence and stores this encoding in the final hidden state.

The next step is to pass this encoding to decoder RNN in the initial state. Firstly, we feed the start token to the decoder, and then we can produce the first state of the decoder, as we are using the last hidden state of the encoder as the initial state of the decoder.

Training and testing processes for the encoder part are the same and involves creating contextualize representation of the input sequence. However, the Decoder part is different for training and testing. During the training part, we feed to the decoder also target sentence as the ground truth. Everything that produces a decoder is used only for computing loss J, where J is weighed sum over each loss of each decoder output. Loss of each output is computed as negative log probability of next word. The training process is depicted in the Figure 2.17.

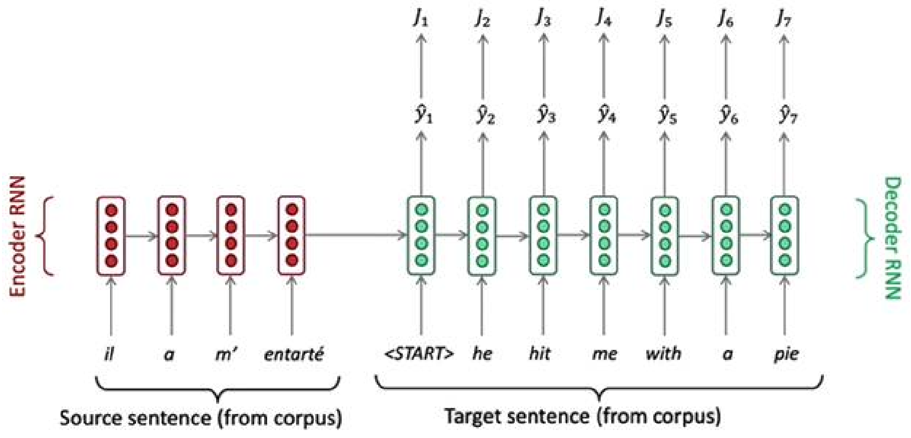

J_1 J_2 J_3 J_4 J_5 J_6 J_7
$\hat{y}_1$ $\hat{y}_2$ $\hat{y}_3$ $\hat{y}_4$ $\hat{y}_5$ $\hat{y}_6$ $\hat{y}_7$
Encoder RNN
Decoder RNN
il
a
m'
entarté
<START>
he
hit
me
with
a
pie
Source sentence (from corpus)
Target sentence (from corpus)

Figure 2.17: Seq2Seq training.

The testing part does not have a target sentence and in every state of decoder, using a probability distribution, trying to predict word which can come next using argmax of this distribution. Then we feed chosen word to the next step of the decoder and repeat this procedure until we will get the end token. The choice of the next word is conditioned by the previous state of a decoder and we use a word, which was produced by the previous step, as an input of the current step of the decoder. This is not generally true as in the attention mechanism which will be described later. The test process is depicted below (Figure 2.18 on the following page).

One of the problems of the seq2seq model described above is that it chooses words greedily, which means that in long-term games it can produce bad results. To solve this problem, when choosing a word to generate, we choose K's most

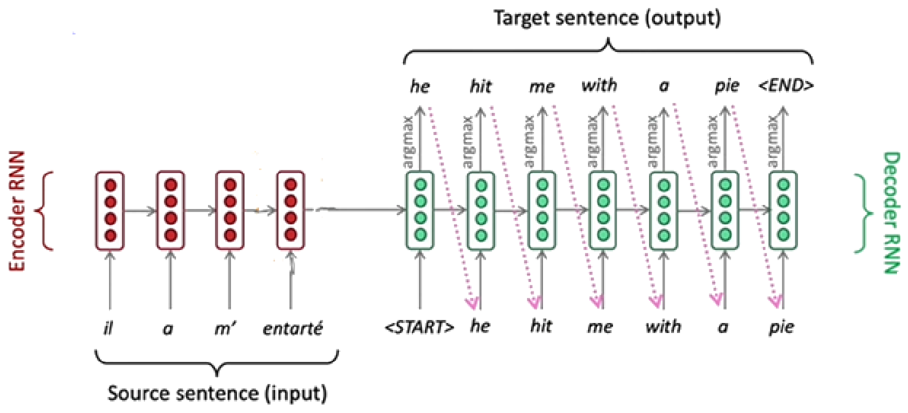


Figure 2.18: Seq2Seq testing.

promising words at each state of the decoder. It will generate a tree in which we need to choose a branch with a smaller normalized error value. This approach is called Beam search. Both encoder and decoder could be also vanilla or bidirectional RNN, GRU, or LSTM depending on the problem we are solving.

There is a drawback when we use this architecture and is related to interpretability, it is more difficult to debug and control. Worldwide there are many types of research in this kind of analysis of RNN, and this problem is still not resolved.

One improvement was created and it was so crucial that it is currently considered as new vanilla and it is called Attention. In standard seq2seq all the encoding of the sentence is on the last block of the Encoder RNN and this can lead to a bottleneck (Figure 2.19) as this state is overwhelmed with information.

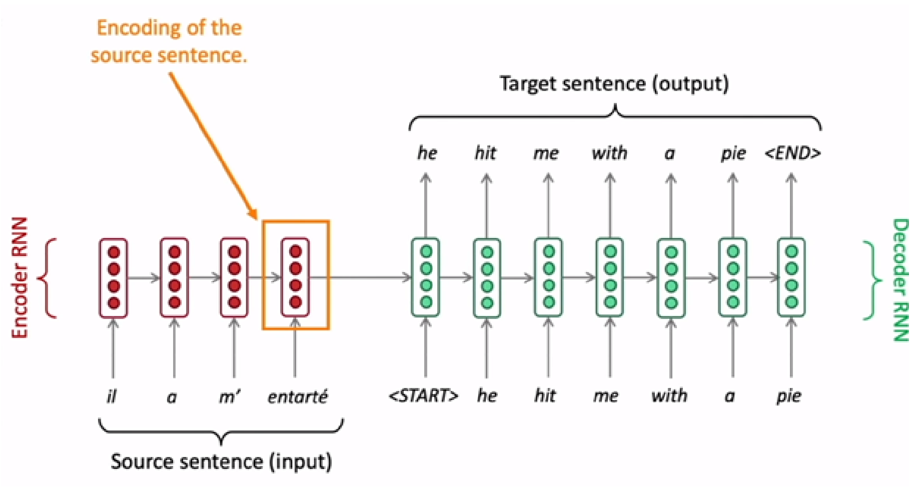


Figure 2.19: Seq2Seq bottleneck.

Attention provides a solution to the bottleneck problem. The idea is to use for each step of the encoder a direct connection to the encoder to focus on a particular part of the source sequence (Figure 2.20 on the facing page).

At the beginning we take the dot product between the first decoder hidden state and the first encoder hidden state, the result is called attention score. This value is

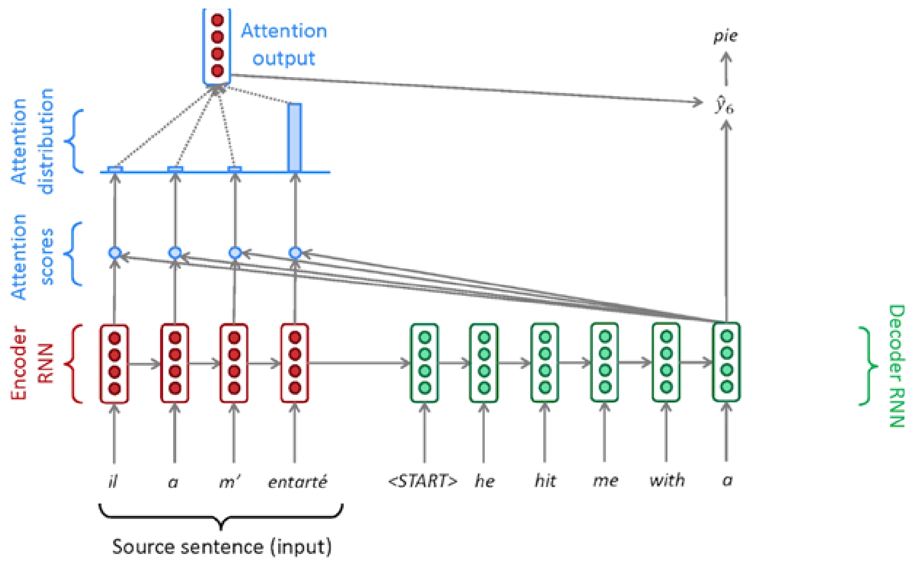


Figure 2.20: Seq2Seq attention.

obtained with all the other encoder hidden states, and we will have one attention score for each hidden state of the source sentence.

Using these attention scores, we apply the softmax function and we get a probability distribution for each attention score, and this will be the attention distribution. This value will be used to produce the attention output It is the weighted sum of the encoder hidden state.

The Attention output which is a single vector is going to contain mostly information from the hidden states that received high attention. Then we use the Attention output to influence the output of the next word, we concatenate the attention output with the decoder hidden state.

Then for each element of the decoder, we compute the same step as before for getting the output. Notice that in some cases attention output from the previous step is feed into the decoder.

General improvements:

- Attention significantly improves the performance
- Attention solves the bottleneck problem
- Attention helps with the vanishing gradient problem
- Attention provides some interpretability by inspecting the attention distribution we can see what the decoder was focusing on.
- Soft alignment is for free

2.6 Highway Network

Highway Network tries to resolve the same problem as Residual Connection (see Section 2.7) tried to resolve[7]. Training becomes difficult as the depth of the network increase. highway networks allow unimpeded information flow across several layers on information highways. The architecture is characterized by the use of gating units that learn to regulate the flow of information through a network.

Highway networks with hundreds of layers can be trained directly using stochastic gradient descent and with a variety of activation functions, opening the possibility of studying extremely deep and efficient architectures.

A plain feedforward neural network typically consists of L layers where the layer l (l ⊆ 1, 2, ..., L) applies a nonlinear transform H (parameterized by WH,l) on its input x_l to produce its output y_l. Thus, x_1 is the input to the network, and y_L is the network's output. Omitting the layer indices and biases for clarity, we are getting the following equation:

$$y = H(x, WH) \tag{2.13}$$

Where H is usually an affine transform followed by a non-linear activation function, but in general, it may take other forms. For a highway network, we additionally define two nonlinear transforms $T(x, WT)$ and $C(x, WC)$ such that:

$$y = H(x, WH) \cdot T(x, WT) + x \cdot C(x, WC) \tag{2.14}$$

We refer to T as the transform gate and C as the carry gate, since they express how much of the output is produced by transforming the input and carrying it, respectively.

$$y = \begin{cases} x & \text{if } T(x, Wt) = 0 \\ H(x, W_H) & \text{if } T(x, Wt) = 1 \end{cases} \tag{2.15}$$

2.7 Residual Connection

Deeper neural networks are more difficult to train. A residual learning framework allows the training of networks that are substantially deeper than those used previously. The layers are reformulated as learning residual functions concerning the layer inputs, instead of learning unreferenced functions. The depth of representation is very important for many visual recognition tasks.[8]

But is learning better networks as easy as stacking more layers? An obstacle to answering this question was the notorious problem of vanishing/exploding gradient which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization, with the network depth increasing, accuracy gets saturated and then degrades rapidly.

Instead of hoping every few stacked layers directly fit a desired underlying mapping, we explicitly let these layers fit a residual mapping. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping.

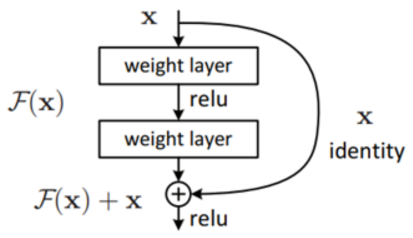

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.21: Residual connection.

Residual Learning

In details. we can consider $H(x)$ as an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers. So rather than expect stacked layers to approximate $H(x)$, we explicitly let these layers approximate a residual function $F(x) := H(x) - x$. The original function thus becomes $F(x) + x$. Although both forms should be able to asymptotically approximate the desired functions (as hypothesized), the ease of learning might be different. If the added layers can be constructed as identity mappings, a deeper model should have a training error no greater than its shallower counterpart. With the residual learning reformulation, if identity mappings are optimal, the solvers may simply drive the weights of the multiple nonlinear layers toward zero to approach identity mappings.

If the optimal function is closer to an identity mapping than to a zero mapping, it should be easier for the solver to find the perturbations concerning an identity mapping, than to learn the function like a new one. Here we can see a piece of the different network architecture and how are composed. In the last network, shortcut connections are inserted which turns the network into a residual version. (Figure 2.22 on the next page).

Highway network presents shortcut connections with gating function that are data-dependent gates and have parameters, in contrast to identity shortcuts that are parameter-free. When a gated shortcut is “closed” (approaching zero), the layers

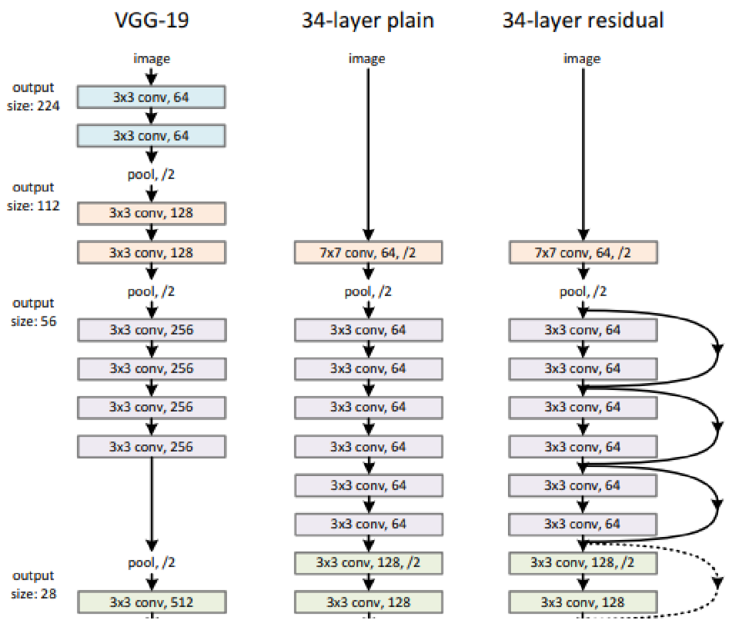


Figure 2.22: Residual network.

in highway networks represent non-residual functions. On the contrary, residual connection formulation always learns residual functions; our identity shortcuts are never closed, and all information is always passed through, with additional residual functions to be learned. Besides, highway networks have not demonstrated accuracy gains with extremely increased depth.

2.8 Tacotron

Tacotron is an end-to-end character-based architecture for Text-To-Speech synthesis based on seq2seq with an attention mechanism. Word-level neural machine translation models, despite their popularity, have some major weaknesses such as difficulty with modeling out-of-vocabulary words, and with huge dictionaries, its complexity becomes a problem. Character-based models can avoid these problems and they perform better for multilingual translation [11].

The advantage of end-to-end models is that they require much less labor feature engineering, which may require some heuristics and difficult design choices. Also, it more easily allows for rich conditioning on various attributes, such as speaker or language, or high-level features like the sentiment.

2.8.1 Model description

The model of the architecture is shown in Figure 2.23. At a high level, the model takes characters as input and produces spectrogram frames, which are then converted to waveforms.

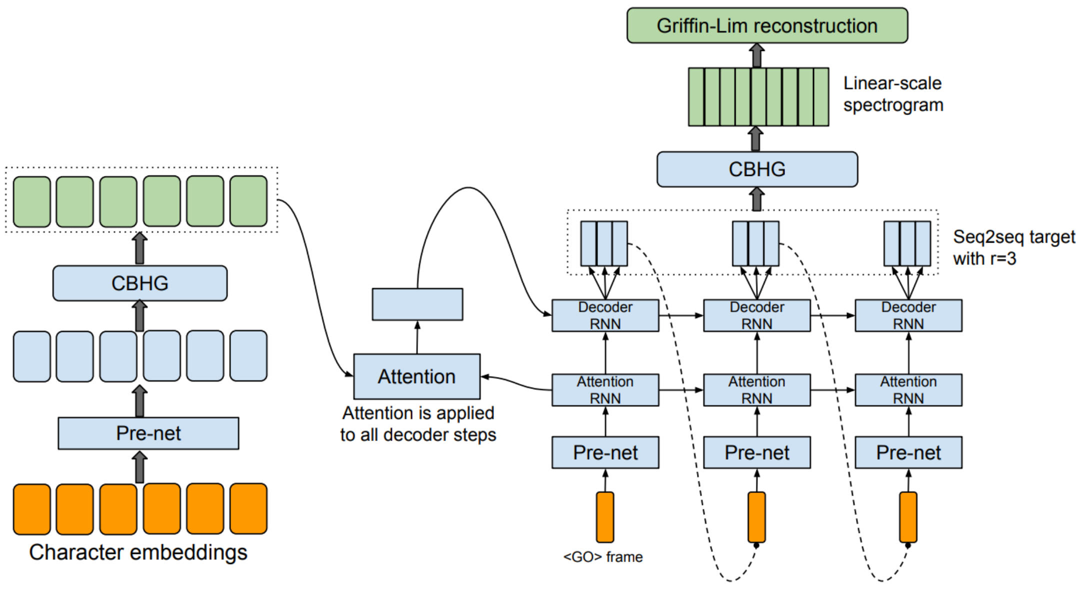


Figure 2.23: Tacotron architecture

The left part of the figure represents the encoder which is used to extract a contextual representation of the input sequence. The input sequence of characters is firstly embedded into a continuous vector. Then, for each embedding, we apply non-linear transformation using two fully connected layers with dropout. This transformation is called “pre-net” and it is used for improving generalization. Then we pass output to the CBHG module.

The right part of the figure represents the decoder. More precisely it uses a content-based attention decoder, which was described in the previous chapter. The score is computed with the tanh function. The task is to concatenate the context vector and the attention RNN cell output to form the input to the decoder RNNs.

The target of the decoder is an important component. In the Tacotron case, the compressed target might be sufficient as far as it provides sufficient intelligibility and prosody information for an inversion process. In the work, they use 80-band Mel-scale spectrogram as the target. One important thing which was done is that decoder output several frames at once. It is done as it speeds up convergence and training and one character is usually represented by more than one frame, so quality does not get worse. The first decoder step is conditioned on an all-zero frame, which represents a frame. In inference, at decoder step t, the last frame of the r predictions is fed as input to the decoder at step $t + 1$.

The next step is to pass all decoder outputs to CBHG, which in this case is responsible for the post-processing net. Post-processing net learns to predict spectral magnitude sampled on the linear frequency scale. Also, this net can see the whole decoder output at once and make some adjustments. The last step is a synthesis of the waveform, which was done using the Griffin-Lim algorithm. This algorithm was chosen for simplicity as it is differentiable and can be tuned in any way.

2.8.2 CBHG module

CBHG module takes into input the result of the pre-net transformation. CBHG consist of a bank of 1-D convolutional filters, the input sequence is convolved with K sets of 1-D convolutional filter, where the k^{th} set contains C_k filters of width k. As we can see from the Figure 2.24, we have a different filter of a different dimension.

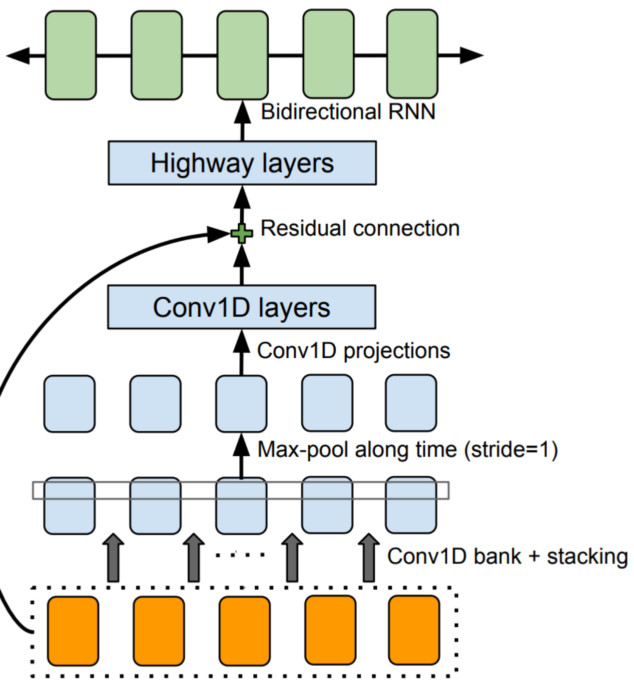


Figure 2.24: CBHG architecture

The convolutional output is stacked together, and a further max pooled along time to increase the local invariances.

After these steps, we further pass the processed sequence to a few fixed-width 1-D convolutions whose output is added with the original input sequence. This can be done by using Residual Connection. Notice that Batch normalization is used for all convolutional layers. The convolution outputs are fed into a multi-layer highway network to extract the high-level feature.

Finally, we stack a bidirectional GRU RNN on top to extract sequential features from both forward and backward contexts. The CBHG module improves the generalization.

Chapter 3

State of the Art

In this chapter we will talk about the state of the art, so what are the current best TTS systems developed so far. We will first explain architectures that we used as a final baseline for comparison with our model, then we will present alternatives approaches for vocoder and speech synthesis.

First, we will talk about Tacotron 2 (see Section 3.1), a neural network architecture for speech synthesis directly from the text. We will go into detail explaining its architecture and the differences from its previous version and why it is one of the best TTS systems at the moment.

Subsequently, we will talk about one of the approaches that had excellent results for style transfer. is an extension of Tacotron 2 and is called GST (see Section 3.2). Global style tokens are a bank of embeddings that are jointly trained within Tacotron, a state-of-the-art end-to-end speech synthesis system. The embeddings are trained with no explicit labels, yet learn to model a large range of acoustic expressiveness.

Later, we are going to talk about Wavenet (see Section 3.3), a deep generative model of raw audio waveforms. WaveNets can generate speech that mimics any human voice and which sounds more natural than the best existing Text-to-Speech systems, reducing the gap with human performance.

When we want to generate speech, we usually refer to the TTS concatenative, but in this system, it is very difficult to change the voice without registering a new database. With this new need, parametric TTS were introduced, but they tend to have a less natural sound than concatenative TTS. Wavenet changes this paradigm because it works directly on the waveform of the audio signal, one sample at a time.

However, Wavenet also has drawbacks. To solve these problems we will talk about an alternative solution, its name is WaveGlow (see Section 3.4) that allows having a generation of speech almost in real-time at the cost of a reduction in quality.

Finally, we will introduce and explain another architecture that compared to Tacotron 2 has a higher training speed, the so-called Transformers (see Section 3.5). We will talk about the differences and their advantages and disadvantages compared to Tacotron 2.

3.1 Tacotron 2

During these years, different techniques have dominated the field of TTS. Initially, Concatenative synthesis with unit selection was the state of the art, later some methods based on statistical parameters have solved some known problems of concatenative synthesis, but the audio produced was still muffled and not natural.

Tacotron 2 [17] combines the two best approaches: a sequence-to-sequence Tacotron-style model that generates Mel-spectrograms, followed by a modified WaveNet vocoder.

Vocoders are the neural systems which used to generate human voice from mathematical models of the vocal tract or Mel-spectrograms. During the analysis phase, vocoder parameters are extracted from the speech waveform which represent the excitation speech signal and filter transfer function

WaveNet vocoder (see Section 3.3) was later introduced, a generative model of time-domain waveforms that produce the audio quality that begins to rival that of real human speech, however, requires significant domain expertise to produce, involving elaborate text-analysis systems as well as a robust lexicon.

The architecture of Tacotron 2 is depicted in Figure 3.1 and is composed of two main components:

- Recurrent sequence-to-sequence feature prediction network with attention.
- Modified version of WaveNet.

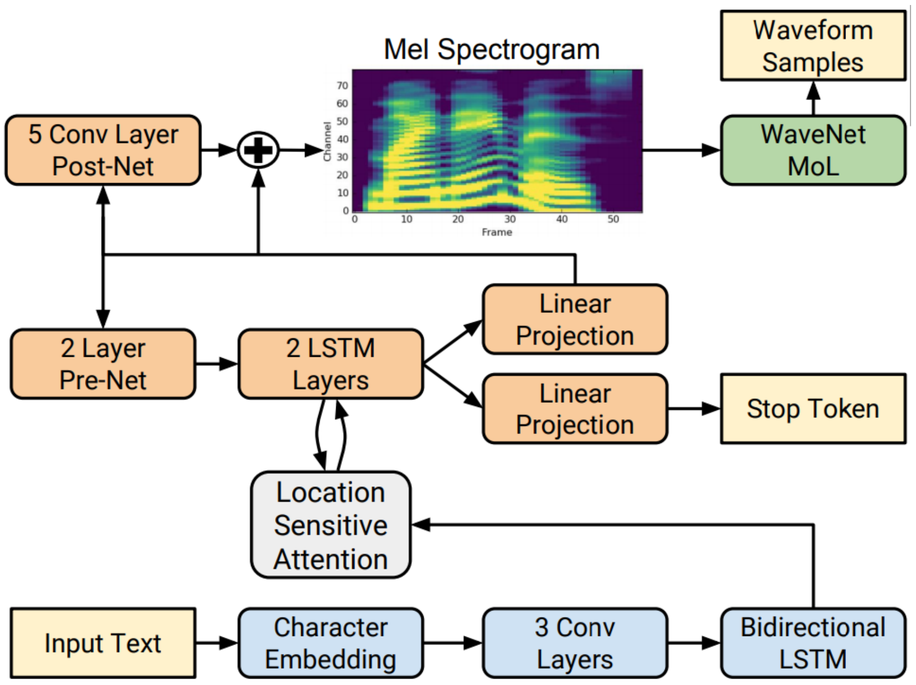


Figure 3.1: Tacotron 2 Model Architecture.

3.1.1 Intermediate Feature Representation

Mel frequency spectrograms are chosen as a low-level acoustic representation. With this representation is easier to operates on time-domain waveforms, furthermore is smoother than waveform samples and easier to train.

The Mel frequency spectrograms can be obtained by applying a nonlinear transform to the frequency of the STFT[1]. There are properties in high frequency and low frequency that can be summarized as features that have been always used in previous TTS systems.

Linear spectrograms discard this kind of information but algorithms like Griffin-Lim can estimate this discarded information. Mel-spectrogram discards even more information and can lead to a difficult problem to resolve. However, the Mel-spectrogram is simple since it contains a lower-level acoustic representation of the audio signals. Since the input of WaveNet is simpler is possible to generate high-quality audio from Mel-spectrograms with the modified WaveNet.

3.1.2 Spectrogram Prediction Network

As in Tacotron, this model use encoder and decoder with attention. However, there are several significant differences. Instead of using pre-net and CBHG layers at the embedding stage, we pass input characters through 512- dimensional embedding, and then it passed to 3 convolutional layers, each having 512 filters 5-by-1 each.

This dimensionality of the filter is useful as it can look at the current character, 2 previous, and two followings. Each convolution is followed by batch normalization and ReLu activation function. Afterward, the output is passed to the single bi-directional LSTM.

The encoder output is passed to a location-sensitive attention network as a context vector, which summarizes the full encoded sentence at each decoder output step. The difference of this attention mechanism with respect to the classical one is that it also considers cumulative attention weights from the previous step of the decoder.

The decoder of Tacotron 2 is an autoregressive recurrent neural network, which learns to predict Mel-spectrogram from the given encoder context. The prediction from the previous step is firstly passed to pre-net which has the same structure as in original Tacotron as it helps for learning attention. Then the output of pre-net and attention context is concatenated and passed through 2 LSTM. Then we project the output via linear transform to predict the target spectrogram frame.

Lastly, we use 5 layers of CNN post-net with dropout which predicts a residual and improves overall reconstruction. In parallel to frame prediction, we pass decoder LSTM output and context of the attention, after projecting to a scalar, through sigmoid function which helps to determine whether input sentence is finished via "stop token".

[1]Short-time Fourier transform

3.1.3 Wavenet Vocoder

The vocoder is a modified version of the standard WaveNet architecture. As in the original one, there are 30 dilated convolution layers grouped into 3 dilation cycles In order to work with the 12.5 ms frame hop of the spectrogram frames, only 2 upsampling layers are used in the conditional stack (original Wavenet used 3 layers).

PixelCNN and Parallel Wavenet are used to generate 16-bit samples at 24 kHz, there is a 10 component mixture of logistic distributions instead of the softmax layer. The logistic mixture is computed by passing through a ReLu activation followed by a linear project to predict parameters for each mixture component. Loss is computed as the negative log-likelihood of the ground truth.

Evaluations

Table 3.1 shows a comparison of Tacotron2 against various prior systems. In order to better isolate the effect of using Mel-spectrograms as features, they compare to a WaveNet conditioned on linguistic features with similar modifications to the WaveNet architecture.

They also compare to the original Tacotron that predicts linear spectrograms and uses Griffin-Lim to synthesize audio, as well as concatenative and parametric baseline systems, both of which have been used in production at Google. they find that the proposed system significantly outpeforms all other TTS systems, and results in an MOS[2] comparable to that of the ground truth audio.

Table 3.1: Mean Opinion Score (MOS) evaluations

System	MOS
Parametric	3.492 ± 0.096
Concatenative	4.166 ± 0.091
WaveNet (Linguistic)	4.341 ± 0.051
Ground truth	4.582 ± 0.053
Tacotron 2	4.526 ± 0.066

[2]Mean opinion score

3.2 Global style tokens - GST

To lead to an increasingly human experience of speech, a TTS system must learn to model prosody. Prosody is the confluence of a number of phenomena in speech, such as paralinguistic information, intonation, stress, and style [24]. The goal is to create a model in which a speech style can be chosen according to the context, but one of the problems that may arise is related to the objective measure of correctness in the prosodic style. One of the strengths of GTS is that it can also be applied to unlabeled and noisy data.

3.2.1 Model Architecture

The model is based on Tacotron that uses a sequence-to-sequence model that predicts Mel-spectogram. These Mel-spectograms are converted to waveforms by WaveNet. We have two different approaches to using the GST model, during a training and during an inference as we can see from Figure 3.2.

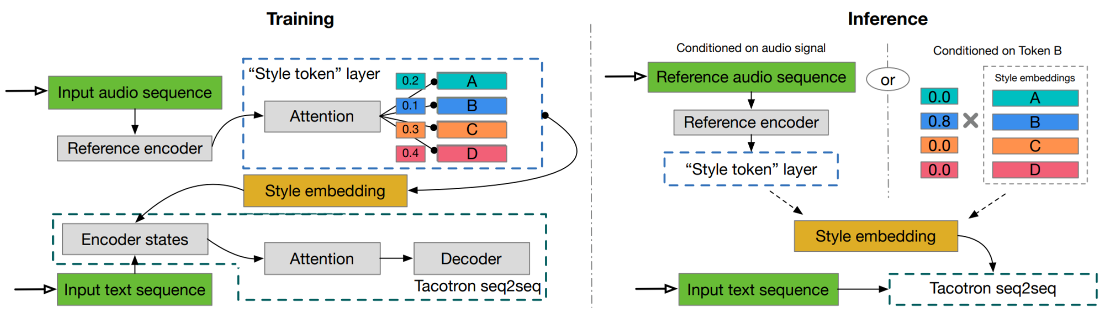


Figure 3.2: GST Model Diagram

Training

During the training the information flows in the following way:

1. The **reference encoder**, compresses the prosody of a variable-length audio signal into a vector, we will call him *reference embeddings*.

2. The reference embedding passes through a style token layer, where it is used as a query vector to an **attention module**, but in this case, we are not learning alignment but the similarity measure between the reference embedding and each token (randomly initialized embeddings).

3. The attention model produces a set of combination weights that represents the contribution of each style token to the encoded reference embedding

4. The style token layer are jointly trained with the rest of the model taking only into account the reconstruction loss from the Tacotron decoder. No explicit style is required.

Inference

GST was designed to be flexible and allow great control in inference mode, there are two methods to get a sample from the model:

- We can directly condition the text encoder on a certain token without using a reference signal.
- We can feed a different audio signal to achieve style transfer.

For The GST-augmented Tacotron systems, they use the same architecture and hyperparameters as [17] except for a few details. They use phoneme inputs to speed up training, and slightly change the decoder, replacing GRU cells with two layers of 256-cell LSTMs; these are regularized using zone-out with probability 0.1.

The decoder outputs 80-channel log-Mel spectrogram energies, two frames at a time, which are run through a dilated convolution network that outputs linear spectrograms. They run these through Griffin-Lim for fast waveform reconstruction. It is straightforward to replace Griffin-Lim by a WaveNet vocoder to improve the audio fidelity. The baseline model achieves a 4.0 mean opinion score.

3.2.2 Multi-headed attention

The idea behind the Scaled Dot product (Figure 3.3) is to convert the input vector using Keys, Queries, and Values to extract context information and put it alongside input as a weighted vector. For example, we can imagine that input is a sentence, Query is a particular word in a sentence, Keys are all the words in a sentence. By making dot product between Query and Keys we normalize them and then we use SoftMax to obtain a vector of weights for this particular query. Finally, we multiply this vector on original Values and obtain the final result.

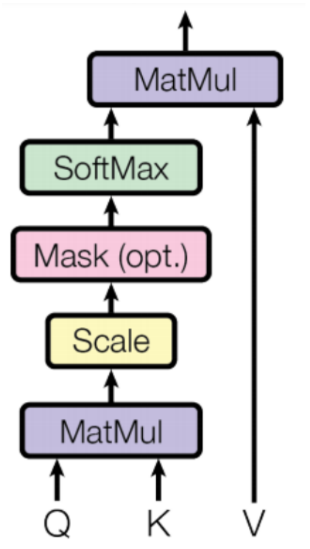


Figure 3.3: Scaled Dot product attention.

Multi-headed attention (Figure 3.4) is a stack of Scaled Dot-Product attentions to which gives an advantage of parallelization of computations. However, when we send as input set of Queries, Keys, and Values we get dimensionality mismatch in output comparing to the input vector. This can be solved using the concatenation layer after Scaled Dot-Product attentions and then run it through the Dense layer which will result in the same dimensionality as the original input vector.

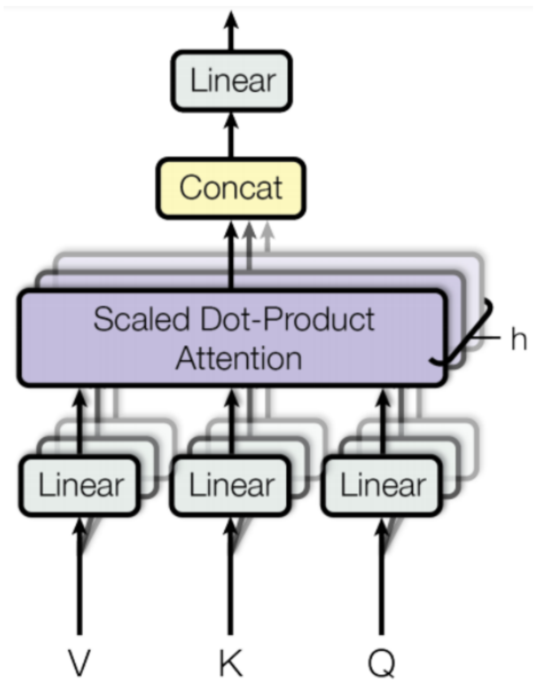


Figure 3.4: Multi-headed attention.

3.2.3 Style Token Architecture

The architecture consists of a reference encoder and style token layer. The former extracts a high-level prosody embedding vector from the input audio. After, in the style token layer, They use a multi-headed attention module to learn the similarity between each style token and prosody embedding.

The weights which we will find during training represent how much each token contributes to some specific speaking style. In the end, we generate a style embedding vector by use of a linear combination of learned weights and global style tokens.

Reference encoder

The reference encoder is made up of a convolutional stack, followed by an RNN. It takes as input a log-Mel spectrogram, which is first passed to a stack of six 2-D convolutional layers with 3×3 kernel, 2×2 stride, batch normalization and ReLU activation function. They use 32, 32, 64, 64, 128 and 128 output channels for the 6 convolutional layers, respectively.

The resulting output tensor is then shaped back to 3 dimensions (preserving the output time resolution) and fed to a single-layer 128-unit unidirectional GRU.

The last GRU state serves as the reference embedding, which is then fed as input to the style token layer.

Style token layer

The style token layer is made up of a bank of style token embeddings and an attention module. Unless stated otherwise, our experiments use 10 tokens, which we found sufficient to represent a small but rich variety of prosodic dimensions in the training data. To match the dimensionality of the text encoder state, each token embedding is 256-D. Similarly, the text encoder state uses a tanh activation; applying a tanh activation to GSTs before applying attention led to greater token diversity.

The content-based tanh attention uses a softmax activation to output a set of combination weights over the tokens; the resulting weighted combination of GSTs is then used for conditioning. We experimented with different combinations of conditioning styles and found that replicating the style embedding and simply adding it to every text encoder state performed the best. While they use content-based attention as a similarity measure in this work, it is trivial to substitute alternatives. Dot product attention, location-based attention, or even combinations of attention mechanisms may learn different types of style tokens.

In their experiments, they found that using multi-head attention significantly improves style transfer performance, and, moreover, is more effective than simply increasing the number of tokens. When using h attention heads, we set the token embedding size to be $256/h$ and concatenate the attention outputs, such that the final style embedding size remains the same.

3.3 WaveNet

Wavenet is a generative model for audio waveform [16]. The main idea is to predict for each time point a bit of depth. However, this is a huge classification problem as for each time point, we need to predict one of the 65536 values in stereo sound since the bit depth is 16. The first measure, which was taken to solve this problem is compounding transformation, which transforms these 65536 values in the range of 256 values, which at the end we will invert to get the original value back.

As the authors of the paper referred to PixelCNN [21] during the development of Wavenet, they had to deal with training procedure in the following way: during the training in PixelCNN we can train all the pixels in parallel with the assumption, that we can look only at previous pixels in the image, they solved this problem with masked convolutions. In Wavenet they solved this issue via causal convolutions, by shifting output cells on the right, which helps to not look at the future values.

Another innovation in the Wavenet is dilated convolutions. The problem of causal convolutions is that they need to have a big number of layers to increase the receptive field. The bigger receptive field allows us to look for a dependency of data in long term. A dilated convolution is a convolution where the filter is applied over an area larger than its length by skipping input values with a certain step. If in traditional linear filters the output is the weighted sum of the inputs, with causal convolutions extra non-linear operations are applied after each convolution.

Dilated causal convolutions are implemented in WaveNet by doubling the dilation factor until we will get to the given limit after we start from the first value of the set. In the end, we stack dilated convolution operations one over another as illustrated in the Figure 3.5.

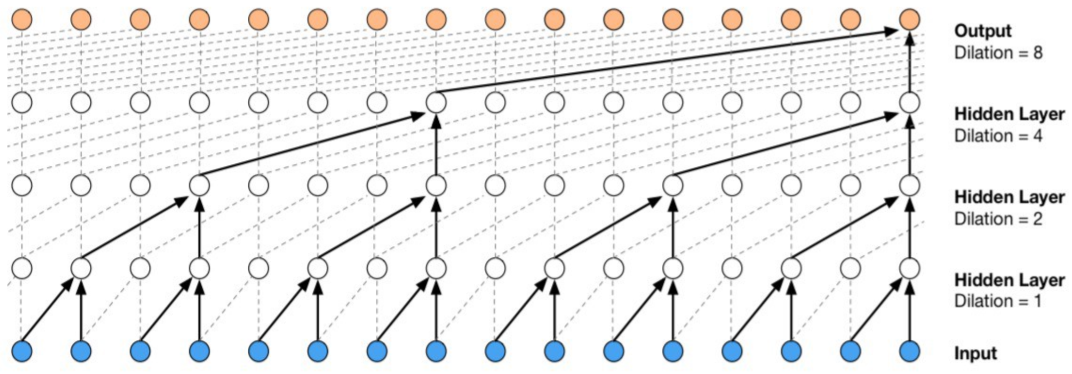


Figure 3.5: Visualization of a stack of dilated causal convolutional layers.

3.3.1 Residual Learning Framework

The overall architecture is depicted in the Figure 3.6, where causal convolution is applied at the beginning and the residual unit represent one hidden layer in the figure, and as we can see there is a stack of them, which correspond to a set of features of the input. All skip connections are summed up to extract relational information through all residual layers. Then two (ReLu - 1x1 convolution) operations are performed to return the number of the features to the original one. Finally, we predict non-normalized probability distribution and we normalize it using the SoftMax function.

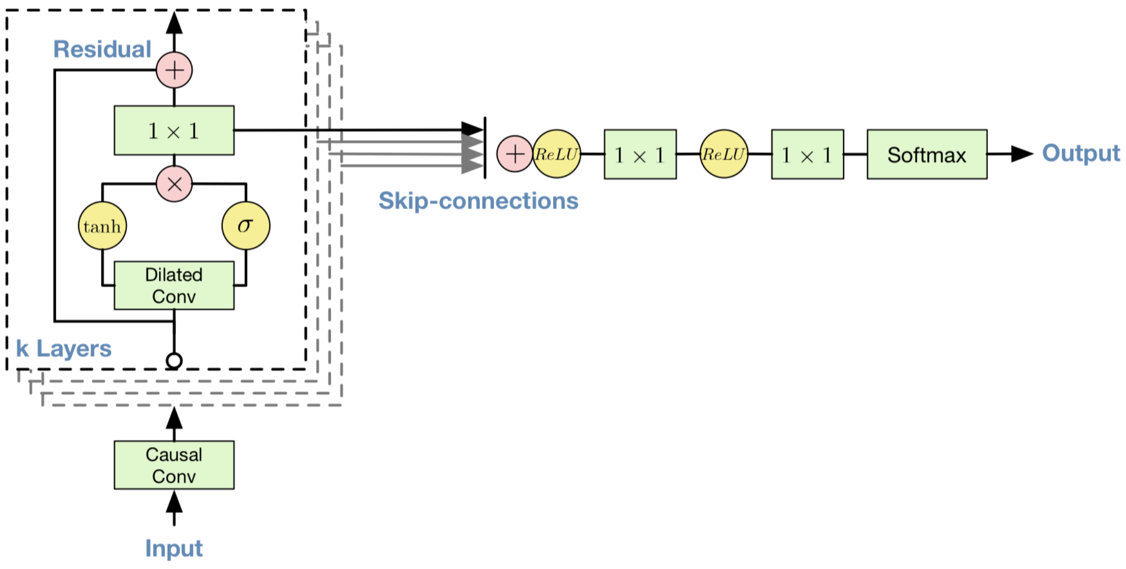


Figure 3.6: Overview of the residual block and the entire architecture.

3.3.2 Residual Unit

WaveNet implements Residual Unit to avoid the vanishing gradient problem and to reduce the training time. The structure of the residual units used in Wavenet is shown in the Figure 3.7. There are shortcut connections implemented as skip-connection and identity mapping. Skip connection $F2$(r) bypasses the residual layer while Identity mappings consist of an element-wise addition between r and the non-linear output $F1$(r).

Thanks to the shortcut connection we can speed up the gradient propagation through all layers. The convolution 1x1 is used to produce in the output the same input matrix but with a different number of features.

There are the same gated activation units as in the PixelCNN network. In PixelRNN [22] is used LSTM and outperform normal PixelCNN, and to resolve this problem instead of using a simple RELU between the masked convolutions the following gated activation unit is used:

$$\mathbf{z} = tanh(W_{f,k} \cdot \mathbf{x}) \odot \sigma(W_{g,k} \cdot x) \tag{3.1}$$

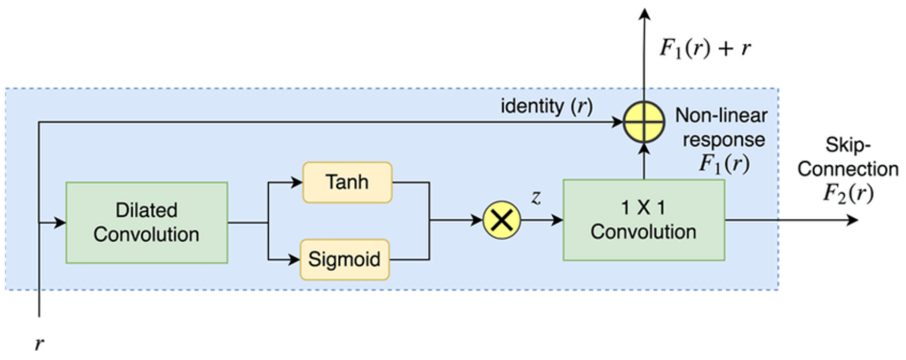


Figure 3.7: Residual Unit.

where σ is the sigmoid non-linearity, k is the number of the layer, $\odot$ is the element-wise product, and $\cdot$ is the convolution operator.

This unit is used as a residual learning framework and r is the input of the residual unit, k is the layer index. f and g are respectively the filter and gate, W indicates learnable convolution filters. This kind of layer works significantly better than a rectified linear unit activation function for modeling audio signals.

3.3.3 Conditional Wavenet

Given an additional input h, WaveNets can model the conditional distribution $p(\boldsymbol{x}|\boldsymbol{h})$.

$$p(x|h) = \prod_{t=1}^{T} p(x_t|x_1, ..., x_{t-1}, \mathbf{h}) \tag{3.2}$$

This parameter allows modifying the probability considering not only the previously generated samples but also some variables that describe the audio to be generated. Without h, the sample x would have the most probable value depending on the previously generated samples, meaning that the final result will be a generalization of what WaveNet learned.

Global Conditioning

Global conditioning uses conditioners that describe the audio samples to be generated through all the time steps (speaker's identity for example). Now the distribution not only depends on the previously generated samples but also the global characteristics of the audio signal.

$$\mathbf{z} = tanh(W_{f,k} \cdot \mathbf{x} + V_{f,k}^T \cdot \mathbf{h}) \odot \sigma(W_{g,k} \cdot x + V_{g,k}^T \cdot \mathbf{h}) \tag{3.3}$$

Where $V_{*,k}$ is a learnable transformation of the global conditioner h, projected to all time steps.

Local Conditioning

Local conditioners direct act the audio samples to be generated over a window of time (text characters, encoder speech parameters, phonemes). Usually, this type of conditioner has a lower sampling rate than the desired audio sequence, therefore it needs to be mapped into the same time resolution.

This can be done by using a transposed convolution operation that applied upsampling. The conditioner h becomes $\mathbf{y} = f(\mathbf{h})$ which is set to the same resolution of the audio sequence. Now we can apply this new conditioner to the gated activation unit as follows:

$$\mathbf{z} = tanh(W_{f,k} \cdot \mathbf{x} + V_{f,k}^{T} + \mathbf{y}) \odot \sigma(W_{g,k} \cdot x + V_{g,k}^{T} \cdot \mathbf{y}) \tag{3.4}$$

Where $V_{*,k} \cdot y$ is a 1x1 convolution that upsamples the local conditioner.

3.4 WaveGlow

WaveGlow is a flow-based network capable of generating high-quality speech from Mel-spectrograms [12]. Basically, it combines the insights of Glow [13] and WaveNet (see Section 3.3), the great advantage of this network is in its speed of speech generation without losing great quality. Waveglow is formed by a single network trained using only a single cost function, auto-regression was not needed.

In detail, we have a generative model that generates audio by sampling from a distribution. To use a neural network as a generative model, the samples are taken from a simple distribution, a zero-mean spherical Gaussian z with the same number of dimensions as the desired output, and put those samples through a series of layers x that transforms the simple distribution into one which has the desired distribution.

$$\begin{gathered}\mathbf{z} \sim \mathcal{N}(\mathbf{z}, 0, \mathbf{I}) \\ x = \boldsymbol{f_0} \cdot \boldsymbol{f_1} \cdot ...\boldsymbol{f_k}(z)\end{gathered} \tag{3.5}$$

The training was carried out with the aim of minimizing the negative log-likelihood. In most cases, it would be very difficult to deal with this type of network with this type of loss, which is why Flow-based networks have been used. We can solve this problem by making reverse mapping possible, the likelihood can be calculated directly using a change of variables:

$$\begin{gathered}logp_\Theta(x) = logp_\Theta(z) + \sum_{i=1}^{k} log\left|det(\boldsymbol{J}(\boldsymbol{f_i^{-1}}(x)))\right| \\ z = \boldsymbol{f_k^{-1}} \cdot \boldsymbol{f_{k-1}^{-1}} \cdot ...\boldsymbol{f_0^{-1}}(x)\end{gathered} \tag{3.6}$$

The first term is the log-likelihood of the spherical Gaussian. This term penalizes the L2-norm of the transformed sample. The second term arises from the change of variables, and the J is the Jacobian. The log-determinant of the Jacobian rewards any layer for increasing the volume of the space during the forward pass. This term also keeps a layer from just multiplying the x terms by zero to optimize the L2-norm. The sequence of transformations is also referred to as a normalizing flow.

The model is most similar to Glow and is depicted in Figure 3.8. For the forward pass through the network, groups of 8 audio samples are taken as vectors, and then the "squeeze" operation is applied, as in Glow. We then process these vectors through several "steps of flow". A step of flow here consists of an invertible 1×1 convolution followed by an affine coupling layer.

Affine Coupling Layer

Invertible neural networks are typically constructed using coupling layers. In the WaveGlow case, an affine coupling layer is used. Half of the channels serve as inputs, which then produce multiplicative and additive terms that are used to scale and translate the remaining channels:

$$\begin{gathered}\mathbf{x_a}, \mathbf{x_b} = split(\mathbf{x}) \\ (log\mathbf{s}, \mathbf{t}) = WN(\mathbf{x_a}, Mel - spectogram) \\ \mathbf{x_b}' = \mathbf{s} \odot \mathbf{x_b} + \mathbf{t} \\ \mathbf{f}^{-1}_{\mathbf{coupling}}(\mathbf{x}) = concat(\mathbf{x_a}, \mathbf{x_b}')\end{gathered} \tag{3.7}$$

WN is a layers of dilated convolutions with gated-*tanh* nonlinearities, as well as residual connections and skip connections. This WN architecture is similar to the one present in WaveNet. In this way, the coupling layer allows to preservers invertibility for the overall network. Now that we inverted the network we are able to compute s and t starting from output x_a, and then we can obtain x_b using $x_b{}'$.

This W N architecture is similar to WaveNet, but the convolutions here have 3 taps and are not causal. The affine coupling layer is also included in the Mel-spectrogram in order to condition the generated result on the input. The upsampled Mel-spectrograms are added before the gated-tanh nonlinearities of each layer as in WaveNet.

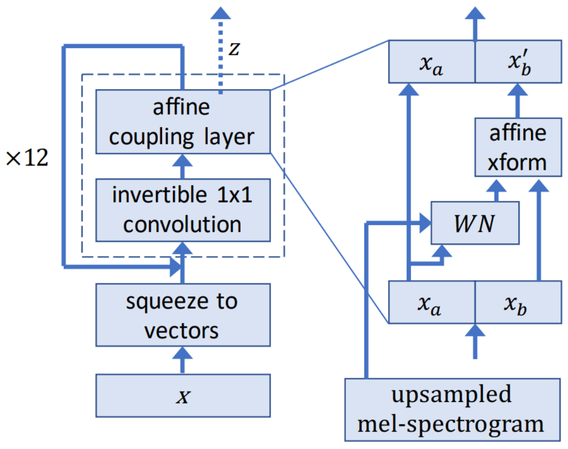


Figure 3.8: WaveGlow arhitecture.

1x1 Invertible Convolution

They were introduced because normally you would have many restrictions without mixing information across channels, but with invertible convolutions, this is no longer the case. The W weights of these convolutions are initialized to be orthonormal and hence invertible. The log-determinant of the Jacobian of this transformation joins the loss function due to the change of variables, and also serves to keep these convolutions invertible as the network is trained.

Early outputs

Rather than having all channels go through all the layers, a different approach was used, it consists to output 2 of the channels from the loss function after every 4 coupling layers. After passing all over the network, the final vector is concatenated with the previous channel. Outputting some dimensions early makes it easier for the network to add information at multiple time scales, and helps gradients propagate to earlier layers, this is very similar to skip connections.

Inference

In order to compute the inference, we should randomly sample z values from a Gaussian and running them through the network. During training was used $\sigma = \sqrt{0.5}$ and during inference, zs was sampled from a Gaussian with a standard deviation of 0.6. Inverting the 1x1 convolutions is just a matter of inverting the weight matrices. The inverse is guaranteed by the loss.

3.5 Transformers

For RNN and LSTM networks data should be passed sequentially. We need to have the input of the previous state to perform any operations in the current state. They are not able to use all power of modern GPU‘s which are designed to make parallel computations. That is why RNN architecture is slow and LSTM even slower due to its high complexity.

To deal with this problem, Transformer Architecture was introduced [18]. This architecture also uses the encoder-decoder model as in RNN‘s but its main feature is that they take the whole sentence as input and perform operations in parallel relying only on self-attention mechanisms. Self-attention, sometimes called intra-attention is an attention mechanism relating different positions of a single sequence to compute a representation of the sequence. It means that we can extract some context from a given embedding vector and see relations within this vector.

As most of the Text-To-Speech architectures that use Neural Networks, the Transformers architecture has an encoder-decoder structure. The encoder maps an input sequence of symbol representation to a sequence of continuous representation. Given that sequence, the decoder generates an output sequence of symbols one element at a time. It is autoregressive, which means that at each step the model consumes the previously generated symbols as additional input when generating the next one. The transformers use a stacked self-attention and point-wise, fully connected layers for both encoder and decoder. The overall architecture of the TTS model which based on transformers [19] is depicted in the Figure 3.9

Encoder

The encoder is composed of a stack of N = 6 identical layers and each layer has two sub-layers. The first is the multi-head attention (see Section 3.2.2) mechanism and the second is a position-wise fully connected feed-forward network. Residual connections are used around each of the two sub-layers followed by layer normalization.

Decoder

The decoder is also composed of a stack of N = 6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer which performs multi-head attention over the output of the encoder stack. The masking ensures that the prediction can depend on the known outputs.

In addition to attention sub-layers, each of the layers in the encoder and decoder contains a fully connected feed-forward network, which consists of two linear transformations with a ReLU activation. A learned embedding is used to convert the input tokens and output tokens to vectors.

Text to phoneme Converter

The neural networks are responsible to find regularities in the training process; however, it is difficult to learn all regularities. A rule-based system is implemented and allows to find all the regularities and convert text-to-phoneme

Encoder Pre-net

In Tacotron2 a 3-layer CNN is applied to the input text embeddings. In the Transformers TTS architecture, the phoneme sequence is the input of the same network called “encoder pre-net". Each phoneme has a trainable embedding of 512 dims, and the output of each convolution layer has 512 channels, followed by batch normalization and ReLU activation, and a dropout layer as well. A final linear projection is applied after the ReLU.

Decoder Pre-net

The Mel-spectrogram is first consumed by a neural network composed of two fully connected layers with ReLU activation. Phonemes have trainable embeddings thus their subspace is adaptive. The decoder “pre-net” is the one responsible for a projection of the Mel-spectrogram into the same subspace as phoneme embeddings to measure the similarity of $<phoneme, mel - frame>$. An additional linear projection is also added to the encoder pre-net to keep consistency and also to obtain the same dimension as the triangle positional embeddings.

Mel Linear, Stop Linear and Post-net

It is the same technique as in Tacotron 2 in which we use two linear projections to predict the Mel-spectrogram and the stop token. Then produce residuals using 5 layers CNN to refine reconstruction of the spectrogram. Also, positive weights were imposed on the tail of the positive token as it was imbalanced due to hundreds of negative frames before the stop token.

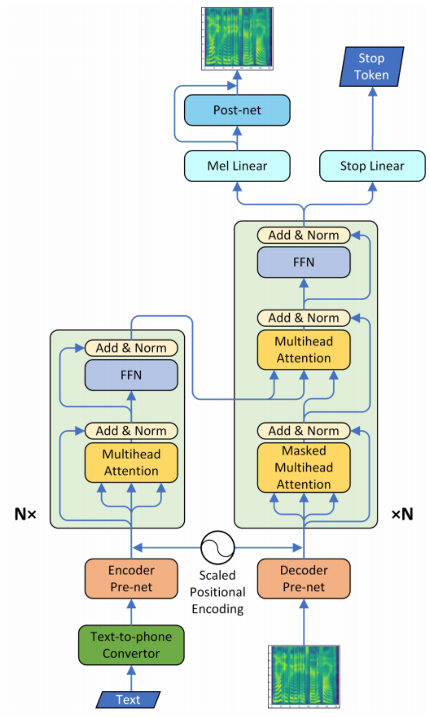


Figure 3.9: TTS transformers arhitecture.

3.6 Discussion

Concerning spectrogram synthesis models, it is worth mentioning that Tacotron 2 and Transformers are not the only two models which we analyze during our thesis work. For example, almost at the same time as Tacotron 2 was published, DeepVoice 3 [14] was also introduced. However, in our opinion, even the demo samples on their official website having a robotic accent, which is incomparably worse than Tacotron2 samples.

Transformers are one of the most trendy and powerful architectures nowadays and they also have shown an excellent quality in terms of neutral speech synthesis. Also, its big advantage is computation speed as it is six times faster for training comparing to Tacotron 2. However, there is one point that has made our decision about the baseline speech synthesis model.

The problem with transformers is that by the time of writing this word, there was no strong evidence that they can handle expressive speech. In Seq2Seq models, the idea of bringing emotions into speech is pretty straightforward which could not be said about transformers with their high parallelization.

During inference time, after have trained the spectrogram generation model, we did not choose a particular vocoder as a final one. As Wavenet and Waveglow have their advantages and disadvantages and since we are in possession of both pretrained versions we could use them without any problems depending on the particular goal we wanted to achieve. To get the best quality we used WaveNet, and to be faster in testing and be more productive in terms of speed we used WaveGlow.

Chapter 4

Datasets and Preprocessing

In this chapter, we will discuss the datasets and corpus used for the tested models and our model EMOTRON. We will also talk about the preprocessing operations carried out for each dataset. We will start by talking about the first dataset we used and it is LJSpeech (see Section 4.1). This dataset is one of the most used as it contains short audio clips of a single professional speaker and is easy to find. It is also one of the most referenced datasets for TTS tasks with excellent results.

Next, we will talk about the MELD[1] (see Section 4.2), After testing the first dataset we were looking for an emotional dataset, this is because LJ Speech is a very good dataset for the TTS task but it is not very emotional. this dataset was created by taking phrases from the television series *Friends*. Later we searched for other datasets, but in this case, we preferred to look for samples obtained from audiobooks since the quality of the audio file is higher. We found LibriTTS (see Section 4.3), a speech corpus designed for Text-To-Speech use. It is derived from the original audio and text materials of the LibriSpeech corpus.

Finally, we wanted to look for a dataset with the same characteristics as the last one but that was only spoken by a single speaker. And that's how we found the dataset used for the Blizzard Challenge 2013 (see Section 4.5). This dataset is used in recent works in Text-To-Speech focuses on generating an expressive and interesting speech. Another dataset we used was IEMOCAP (see Section 4.6). this dataset is mostly used for emotion recognition, it was mainly used to test the performance of our model for the part concerning the classification of emotions.

[1]Multimodal EmotionLines Dataset

4.1 LJ Speech

This is a public domain speech dataset [34] consisting of 13,100 short audio clips of a single speaker reading passages from 7 non-fiction books. A transcription is provided for each clip. Clips vary in length from 1 to 10 seconds and have a total length of approximately 24 hours. The texts were published between 1884 and 1964, and are in the public domain. The audio was recorded in 2016-17 by the LibriVox project and is also in the public domain.

File Format Metadata is provided in transcripts.csv. This file consists of one record per line, delimited by the pipe character (0x7c). The fields are:

- ID: this is the name of the corresponding .wav file
- Transcription: words spoken by the reader (UTF-8)
- Normalized Transcription: transcription with numbers, ordinals, and monetary units expanded into full words (UTF-8).

Following preprocessing were applying by the creator of the dataset:

- The audio clips range in length from approximately 1 second to 10 seconds. They were segmented automatically based on silences in the recording. Clip boundaries generally align with sentence or clause boundaries, but not always.
- The text was matched to the audio manually and a QA pass was done to ensure that the text accurately matched the words spoken in the audio.
- The original LibriVox recordings were distributed as 128 kbps MP3 files. As a result, they may contain artifacts introduced by the MP3 encoding.
- abbreviations can appear in the text. An appropriate solution is to expand them.
- Each audio file is a single-channel 16-bit WAV with a sample rate of 22050 Hz.

4.2 MELD

In the paper [33] which introduce Multimodal EmotionLines Dataset(MELD) dataset they extend and improved EmotionalLines dataset [25]. It includes textual dialogues, visual and audio parts. Dataset has the following characteristics:

- More than 13000 utterance
- 13 hours of total audio
- Emotion recognition baseline was established
- Average duration of audio is 3.59 seconds
- Seven emotions: anger, disgust, fear, joy, neutral, sadness, surprise

EmotionalLines dataset consists of dialogues from the sitcom Friends. Each dialog contains utterances from multiple speakers. The final dataset contains 1000 dialogues.

The utterances in each dialogue were annotated with the most appropriate emotion category by 5 workers from Amazon Mechanical Turk.

However, the utterances in the original EmotionLines were annotated by looking only at the textual part. In the proposed paper, they re-annotate all the utterances by asking the three annotators to also look at the available video clip of the utterances. And then use majority voting to obtain the final label for each utterance.

In Figure 4.1 we can see more detailed statistics on emotional distribution between characters. As we can notice for each character we have approximately 50 percent of neutral samples.

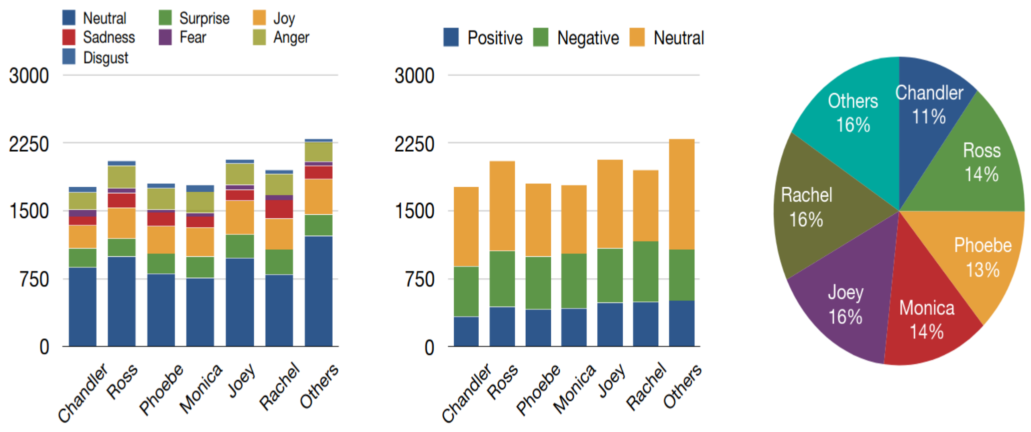


Figure 4.1: MELD statistics.

The following preprocessing steps were applied to MELD dataset in order to being able to train TTS model:

- Dataset was reconverted in format:"audiopath, text of audio" in .csv extension.
- Original dataset contained only video. Convertion to Wav format was performed.
- Number of channels were down-sampled to mono format.
- Sampling rate was converted to 22050Hz

4.3 LibriTTS

LibriTTS corpus is derived from the original materials of the LibriSpeech corpus in audio book domain and is distributed under the same non-restrictive license.[26]
It has the following characteristics:

- Audiofiles has a 24Khz sampling rate. The raw samples which are less than 24KHz were excluded from the dataset.
- The speech is split at sentence breaks; book-level texts are split into sentences using Google's proprietary sentence splitting engine then the audio was split at these sentence boundaries.
- Original and normalized texts are included.
- Utterances with big background noise are excluded
- There are 586 hours of audio.
- 2456 professional speakers used.

The data processing pipeline which was used for dataset generation consist of several steps:
1. Text pre-processing: Book texts are split into paragraphs as a set of newlines. Paragraphs tokenized into sentences non-standard words are detected and normalized.
2. Run ASR on audio data
3. Extract chapter-level text from the book-level text by matching the transcription with the book-level text.
4. Align audio and text.
5. For the TTS train we down-sampled audio to be 22050 Hz each and converted to mono channel.

Below in Table 4.1 there is the number of hours and the speakers obtained following all the preprocessing operations. We can note that in the subset trains the number of hours has decreased compared to the number marked in the name of the subset.

Table 4.1: LibriTTS data subsets

	Hours	Female speaker	Male speaker	Total Speaker
dev-clean	8.97	20	20	40
test-clean	8.56	19	20	39
train-clean-100	53.78	123	124	247
train-clean-360	191.29	430	474	904
train-other-500	310.08	560	6004	1160
total	572.38	1152	1238	2390

4.4 SEMAINE

Semaine corpus derived from a large audiovisual database as part of an iterative approach to building Sensitive Artificial Listener agents that can engage a person in a sustained, emotionally coloured conversation [32].

- The dataset features 150 participants.
- The youngest participant was 22, the oldest 60, and the average age is 32.8 years old (std. 11.9).
- Participants come from 8 different countries
- Typical session duration was about thirty minutes

The concept of recording this dataset was by recording chat between 'user' and 'operator', where operator should behave as one of the SAL[2] characters.
SAL characters:

- Poppy is a outgoing and optimistic female.
- Obadiah is a gloomy and depressed man.
- Prudence is a pragmatic and practical female.
- Spike is an angry and argumentative man.

Three basic scenarios were used: First is when human operators play the roles of the SAL characters; Semi-automatic SAL, where a human operator selects phrases from a pre-defined list but the system speaks them; and Automatic SAL, where an automated system chooses sentences and non-verbal signals. The overwhelming majority took part in only one scenario. In this dataset there are seven basic emotions: fear, anger, happiness, sadness, disgust, contempt and amusement. Furthermore, most of the utterances presents laugh at the end and are included in th aligned transcripts with the annotation <LAUGH>.
Each sessions are splits into different subsessions. Non-standard words are detected and normalized. The data processing pipeline which was used for dataset generation consist of several steps:

1. Text pre-processing.
2. Getting the aligned text from datasets.
3. Align audio and text to splits into sentences.
4. Number of channels were down-sampled to mono format.
5. Sampling rate was converted to 22050Hz

[2]Sensitive Artificial Listener

4.5 Blizzard

The Blizzard Challenge 2013 is a dataset of high-quality audiobooks which were provided by The Voice Factory and Lessac Technologies. The data-set contains unsegmented single-speaker audiobooks with a total duration of more than one hundred hours read in a highly expressive manner by Catherine Byers. The book present in this dataset are: Emma, Mansfield Park, Persuasion, Pride and Prejudice, Sense and Sensibility, The Emerald City of Oz, The Patchwork Girl of Oz, A Little Princess, The Secret Garden, Through the Looking Glass, The Awakening, Silas Marner, A Room with a View, Far from the Madding Crowd, The Scarlet Letter, The Gift of the Magi, Daisy Miller, Washington Square, The Jungle Book, Carmilla, Black Beauty, Treasure Island, Ethan Frome, Madame de Treymes, Summer.

There are several versions of the dataset to download but the two most useful for our purpose are two:

- Selected Version
- Full Version

The first is a reduced version of the dataset and contains audio already selected considered the best to be used for neural networks. These audio files are obtained from the different books available listed above and the overall duration of the dataset is approximately 35 hours.

The complete dataset, on the other hand, does not have audio files already divided into sentences but for each folder that represents a book there are audio files that include the entire chapter and its corresponding text divided by chapters.

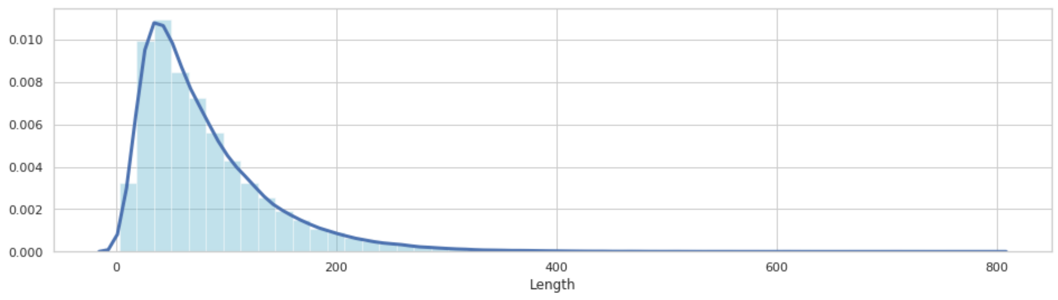


Figure 4.2: Utternace lenght distribution.

For the reduced dataset, it was only necessary to create a file that would combine its path and transcription for each sentence, while for the complete dataset the data processing pipeline which was used consist of several steps:

- Change of transcripts of audiobooks replaced with the one on Project Gutenberg site (some audio and transcripts do not match in the given dataset).
- Getting alignment in order to split the chapter into utterances with Aeneas forced aligner
- splitting chapter into utterance with FFmpeg plugin

- prepare train and validation couple (path, transcripts)
- discards audio less than 1 second and more than 14 seconds.

After this preprocessing step we obtained a subset of utterance lasting 120 hours we can see the utterance duration distribution in Figure 4.2 and the comparison of utterance length and segment duration in Figure 4.3

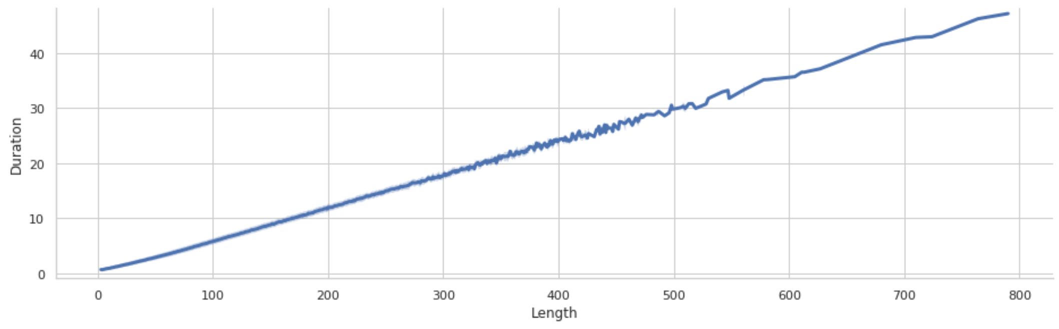


Figure 4.3: Utterance lenght vs segment duration.

4.6 IEMOCAP

The “interactive emotional dyadic motion capture database” (IEMOCAP) [35], is a database that was recorded from ten actors in dyadic sessions with markers on the face, head, and hands, which provide detailed information about their facial expression and hand movements during scripted and spontaneous spoken communication scenarios, it was collected by the Speech Analysis and Interpretation Laboratory (SAIL) at the University of Southern California (USC).

There are five sessions, each recorded session lasts approximately five minutes and consists of two actors interacting with each other. It contains approximately twelve hours of audio-visual data from five mixed-gender pairs of actors. Two acting styles were used: the improvisation of scripts and the improvisation of hypothetical scenarios. The dyadic sessions were manually segmented into utterances.The emotional content of each utterance was annotated by human annotators in categorical labels:

- angry
- happy
- sad
- neutral
- frustrated
- excited
- fearful
- surprised
- disgusted
- other

As the number of emotions is very high we had to restrict the number of emotions by eliminating some of them or by combining similar emotions in a single label. The final result obtained is composed of emotions: Neutral, frustrated, happy, angry, sad. We also later removed noise through the SoX tool using known profiles to remove background noise and clean the audio.

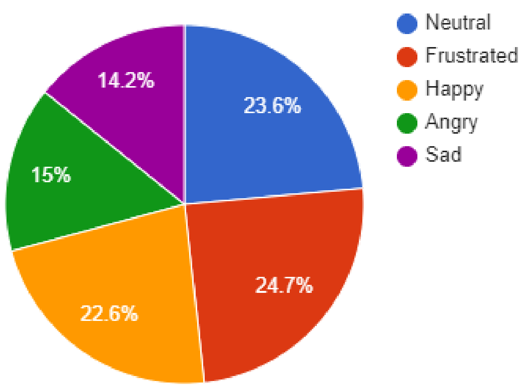


Figure 4.4: Iemocap emotion distribution.

Chapter 5

Emotional Architecture Creation Process

Before the beginning of the development process, there is a crucial need for research in any type of field. In our thesis work, we have been sure that we want to use DNN type of architecture for Text-To-Speech synthesis, however, even in this small area of TTS, there are several novel approaches that we should take into account when considering emotional module implementation.

The main goal of this chapter is to discuss and evaluate different modules capable of influencing expressive speech. Concerning GST module, we do not want to repeat ourselves and will not include this section in the chapter and it is already well described (see Section 3.2).

We will also not discuss our attempt of creating expressive speech with Transformers as it completely failed for us. We believe that it for the reason that Transformers are not Seq2Seq model and it is parallelizing a lot. This speeds up the process of training but making all our approaches infeasible.

5.1 Variational autoencoder

VAE[1] is already a well-known approach for emotion embedding which was declared and used in several papers implementing TTS [9, 10]. We want to give a quick overview of this module.

Autoencoders are an unsupervised learning technique in which we leverage neural networks for the task of representation learning. Basically, it is a compression technique where we map input data into a latent representation vector. The simple structure can be seen in Figure 5.1

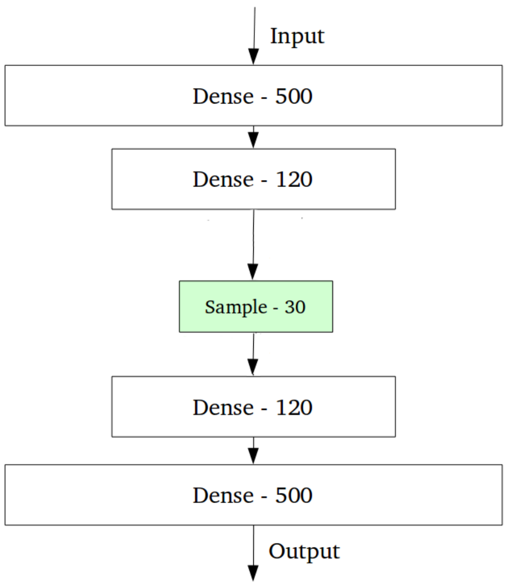


Figure 5.1: Autoencoder.

The problem with autoencoders is that they can not deal with inputs that the encoder has never seen before because during the inference, latent space in which they convert their inputs and where their encoded vectors lie, may not be continuous or allow easy interpolation. To tackle this problem, the variational autoencoder was created by adding a layer containing a mean and a standard deviation for each hidden variable in the middle layer (Figure 5.2).

Instead of mapping input to a fixed vector, we map the input to distribution. This solves an original issue but due to the stochasticity of distribution there is a problem of running the backpropagation through the sampling node. To solve this issue, the reparameterization trick is done from which the latent vector is derived. Reparameterization trick is when we remove the stochasticity of nodes by changing them into:

$$z = \mu + \sigma \odot \varepsilon \tag{5.1}$$

Where z is continuous random latent variable. μ and σ are two trainable parameters and the trick is that we are putting stochasticity into ε which is always

[1]Variational Autoencoders

going to be a standard Gaussian which will not cause any damage to the training process.

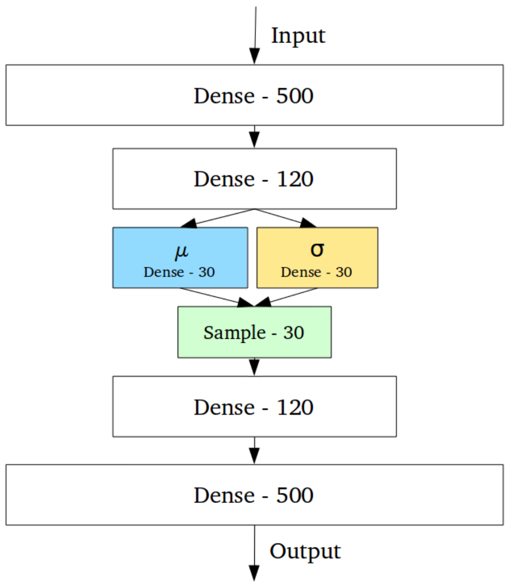


Figure 5.2: Variational autoencoder.

In the Text-To-Speech pipeline we drop decoder part of VAE and the remaining part behaves in the same way as GST and a latent vector is simply added to the output text encoder state after which it is consumed by a location-sensitive attention network of Tacotron 2.

5.2 Hard prosody transfer

There have been many works on prosody transfer and expressive speech synthesis. There are two main approaches:

- Supervised training, it used labels that annotate the accent type or linguistic description of the text.
- Unsupervised training, since labeling is costly and the annotation can be really subjective one way can be to decomposing the residual embedding with style tokens and obtained somewhat disentangled control over prosody attributes in an inductive manner.

Hard prosody transfer architecture has as a baseline the Tacotron 2 model. It used a seq2seq approach and what we are going to do is to modify the network with the same methodology used in GST. Speaker embedding has been introduced, in this way, it is possible to have a multi-speaker model.

First of all, however, we need to introduce new concepts. The first one is the Gradient Reversal Layer, the purpose of this layer is to leaves the input unchanged during forward propagation and reverses the gradient by multiplying it by a negative scalar during the backpropagation. Gradient reversal layer between prosody embedding and speaker classifier is adopted to solve the problem where certain speaker always speaks in a certain style.

In the first part, as shown in Figure 5.3, we have a prosody encoder. As input, we have the mel-spectrogram and the internal structure is composed of an autoencoder, style token, and multi-head attention. It would seem identical to GST but the result obtained is fed to the gradient reversal layer, later this value is fed into a speaker classifier to obtain a speaker embedding. The result is concatenated with the text encoder and sent to the decoder, precisely in the Location-sensitive Attention.

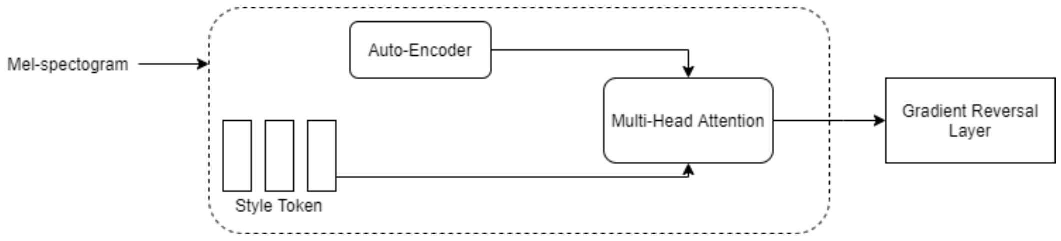


Figure 5.3: Hard prosody encoder.

The other difference that concerns the decoder input is that we no longer have mel-spectogram as input but we also have pitch contour, obtained from F0. The mel-spectrogram going through the mel-prenet, while F0 goes through a single convolution layer followed by a ReLU non-linearity. The two results are then concatenated.

The fact of being a multi-speaker TTS is the big advantage of this network, this particular would help to use different types of datasets and to train with more data.

5.3 Mel-spectogram converter

We have always influenced the text using an embedding of the emotion that has been passed to the decoder with the encoding of the text. An alternative method we thought was to have a different pipeline. In this new architecture, we use Tacotron2 as a baseline to obtain a mel-spectrogram and then we apply a conversion to the mel-spectrogram in the desired emotion. Here in the Figure 5.4 we can see the new pipeline.

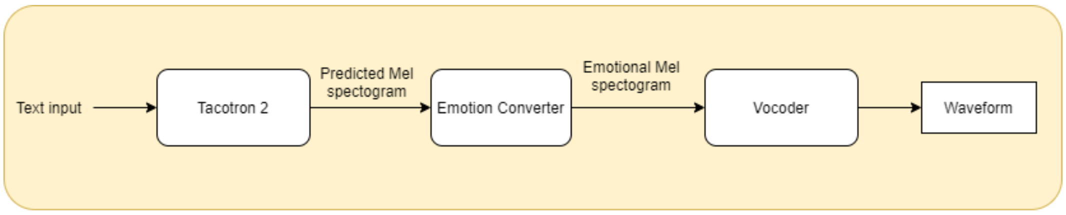


Figure 5.4: Mel converter pipeline.

As we can see we insert the Emotion Converter between Tacotron 2 and the Vocoder. As far as the generation of the waveform is concerned, one of the vocoders (see Section 3.3 and see Section 3.4) already explained in previous chapter can be used.

Emotional converter transforms Mel-spectrograms from neutral style to different emotion styles. we extract Mel-spectrograms from both neutral voice and different emotional voice and use DTW[2] (more information in appendix A) to align them. Then, the aligned Mel-spectrograms are used as input to a neural network (Converter Network) composed of 3 causal convolutional blocks.

The training process is shown in the Figure 5.5.

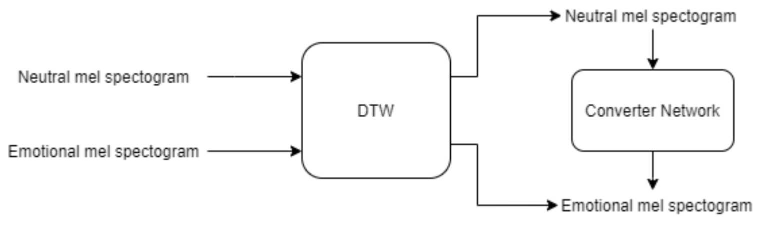


Figure 5.5: Train of emotion converter.

A parallel emotion dataset is needed to train the network. We used TESS [36] and RAVDESS [37] datasets.

[2]Dinamic time warping

5.4 Classification of emotions

After the various experiments carried out previously, we have decided to change our approach. The idea is to combine the Tacotron 2 model with an emotion classifier. First of all, we need to describe how this classifier works. We take inspiration from [39] where the goal was to use pure audio data considering Mel-frequency cepstral coefficients.

The model of classification of emotions here proposed is based on a deep learning strategy base on convolutional neural networks (CNN) and dense layers. The main idea is to consider Mel-frequency cepstral coefficients, commonly referred to as the ”spectrum of a spectrum”, as the only feature to train the model. Notice that MFCC is a different interpretation of the Mel-frequency cepstrum (MFC[3]), and it has been demonstrated to be the state of the art of sound formalization in automatic speech recognition tasks. For each audio file, 40 features have been extracted. The feature has been generated by converting each audio file to a floating-point time series. Then, an MFCC sequence has been created from the time series.

The network designed for the classification task is depicted in Figure 5.6.

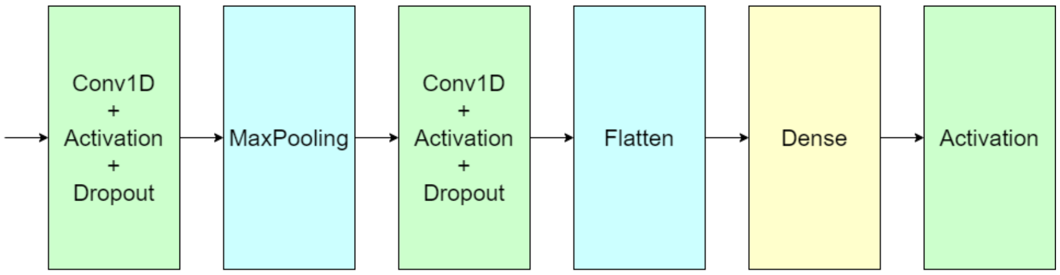


Figure 5.6: Audio feature extraction pipeline.

The network takes as input a vector of 40 features for each audio file. Consequently, we perform one round of 1D CNN with ReLu activation function, dropout of 0.2, and a max-pooling function 2x2. Following this, we have applied another 1D CNN, dropout and then flatten the output to make it compatible with the next layers. Finally, we applied one Dense layer with a softmax activation function, varying the output size to the number of emotion to classify and estimating the probability distribution of each of the classes properly encoded.

In order to combine Tacotron 2 with this model we have changed a part of the described network. We have introduced a dense inter-layer between last Flatten and Dense layer. This layer is used as compressed representation of audio feature which in combination with text encoder outputs should be able to generate expressive speech.

[3]Mel-frequency cepstrum

5.5 Discussion

Variational Autoencoder

We manually inspected the results of its inference in comparison with also well-known GST architecture and we have decided to use Tacotron 2+GST as a baseline model because, in the model which we have trained, the quality of speech and expressiveness was slightly worse than in the GST module. We are not declaring here superiority of Global Style Tokens, allowing that there could have been an option that with a different set of hyperparameters, reference audios, or longer train time the results could have been better for VAE.

Hard prosody transfer

The reason we did not choose this network is due to its inherent characteristic. It is a non-parallel prosody transfer which means that the input that will be generated during the inference will have a similar pitch to the reference audio. It would be impossible to generalize the emotional speech since we will have to test the reference audio with a given emotion and check the correctness.

Mel-spectrogram converter

We didn't get decent results when testing this network, this is due to the size of the datasets used that weren't big enough. Unfortunately, this architecture can only be used with a parallel emotional dataset making it impossible to test it further.

Classification of emotions

We trained using this module and the results were not satisfactory for us. We believed that there might be several reasons for this. In the current approach, we are using only MFCC features which might be not sufficient to represent the expressiveness of speech. The other reason might be the poorly designed neural architecture that we have used for latent vector representation. However, we were inspired by this idea and we decided to continue working in this direction and we used this approach as a base for our final model EMOTRON.

Chapter 6

EMOTRON

We have been inspired by the emotional classification idea and decided to move in this direction. We have combined ideas from the emotional classifier with the GST implementation reference encoder, which we will discuss later in this chapter.

We decided to use Tacotron 2 (see Section 3.1) as a baseline model for comparison quality of the final speech. As we have been working with expressive speech, this baseline is suitable only for speech quality comparison and concerning expressiveness baseline module, so we implemented the GST module (see Section 3.2). While using Tacotron 2 as a baseline, we change several details in implementation details. For example, we started using adaptive learning rates and gradient accumulation. Also during development, we have tried training it using GRU modules instead of LSTM, however, it did not bring many improvements.

Our proposed model is an alternative way to GST and seeks to improve some aspects of it. For example, the GST model uses a multi-head attention model which decides how to extract features from the reference audio. However, it might be worth trying to use a more deterministic approach. In EMOTRON we decided to use two classification modules which will learn to predict emotion of reference audio Mel-spectrogram and predicted Mel-spectrogram in encoder and decoder respectively.

6.1 Architecture

EMOTRON is a reference-based emotion speech synthesis approach based on the Tacotron 2 architecture. We used the same idea of GST in the way how it intersects with the Tacotron 2 pipeline. We have extended Tacotron 2 by introducing an Emotional Matcher which is a classical emotion classifier as we can deduce from the Figure 6.1.

Using this module we can learn the emotion embedding space. This result is used to condition Tacotron2 during learning by concatenating the result obtained from the Tacotron 2 encoder with the emotional matcher. Furthermore, the Mel-Spectrogram obtained from the decoder is inserted into another emotional matcher. This step is done to learn even better the ability to distinguish different emotions. Through the outputs obtained from the two matchers (respectively that of the reference Mel-spectrogram and that produced), we are able to calculate the style loss.

To obtain the style loss we have referred to the paper in which the style is transferred through convolutional networks. We adapted this idea by replacing the images with Mel-Spectrograms.

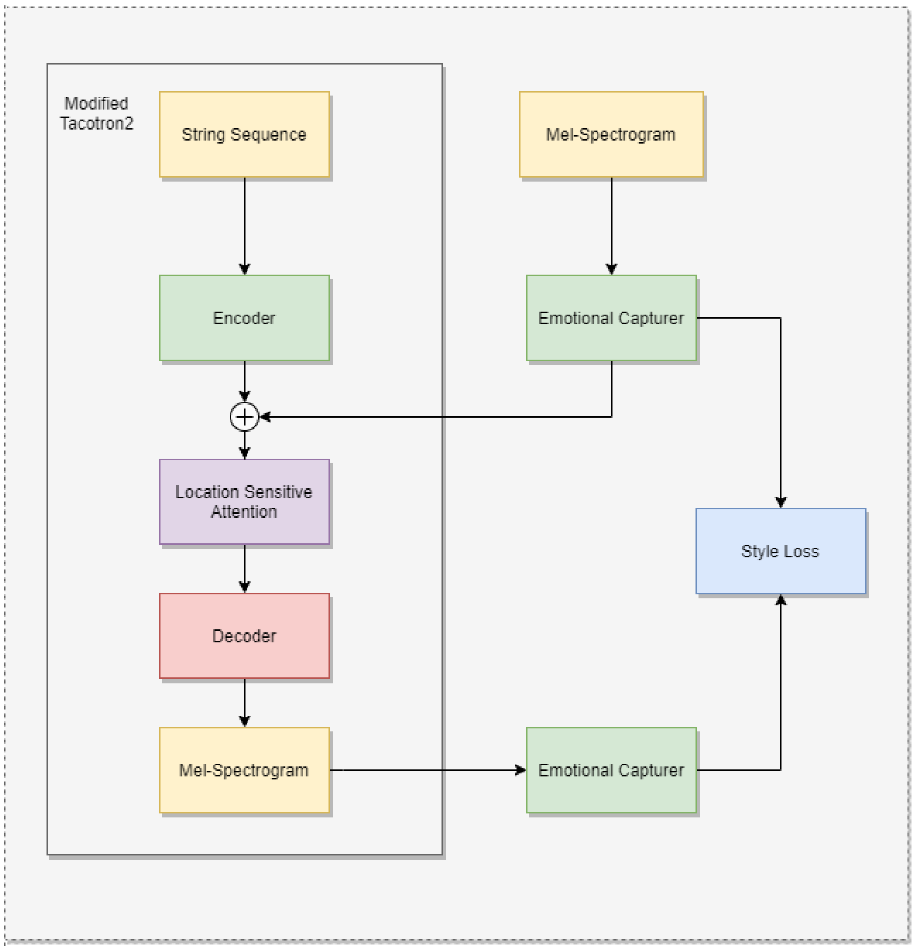


Figure 6.1: EMOTRON proposed model.

Baseline speech synthesis model

During the research and implementation of EMOTRON, several attempts were made. Modifications have also been made and tested on Tacotron 2 as it is the basic structure. As already mentioned above, the model uses encoders and decoders with an attention mechanism and we will go into a little more detail. The first step, which is applied before the encoder, is that of transforming the input text into a format that can be read by the encoder, this operation is the so-called word embedding, they are a set of modeling techniques in which words or phrases from a vocabulary are mapped into vectors of real numbers. With these operations the result will be a space consisting of one dimension per word is transformed into a continuous vector space of a much smaller dimension. We used the layer offered by the framework used and we create an embedding layer where the Tensor is initialised randomly. only in train process the similarity between similar words should appear.

Character input is passed through 256-dimensional embedding to the encoder network. It initially passes through 3 CNNs and finally, in an LSTM, we tried to use a GRU instead of the LSTM without an improvement in performance and results. The result of this embedding text (dimension 256) is combined with that of the emotion classifier which will be discussed specifically in the following paragraphs. This result is passed to the location-sensitive attention network, its purpose is to summarize the output of the encoder and the emotion classifier at each step of the decoder output.

The role of the decoder is to learn how to predict Mel-spectrogram from the encoder + emotional embedding context. a pre-net network is used to improve attention learning using the prediction of the previous step. the embedding context is concatenated with the result of the pre-net network and passed through 2 LSTM. Finally, we use the post-net network made up of 5 CNNs for a better reconstruction.

Emotional strenght

In the next section, we are going to discuss in detail the architecture of our expressive module including Emotional capturer and style loss. It worth mentioning that in implemented EMOTRON during the inference time there is the possibility of controlling emotional strength in a manual manner. This is done by multiply the weights of latent vector of features , extracted from reference audio, by a scalar value. The mentioned weights are the output retrieved from the emotional network.

6.2 Emotional modules

In this section, we will speak about two essential components of EMOTRON architecture which responsible for influencing neutral speech synthesis with expressiveness. These components are classification network and style loss.

6.2.1 Classification Networks

Emotional capturer architecture

The general idea of the proposed architecture is to take a mel spectrogram which can be extracted either from input audio or from predicted by the decoder, where this mel spectrogram is represented as a multidimensional tensor and extract emotional state via classification network.

The Emotional capturer architecture block is depicted in Figure 6.3 and it is worth mentioning that this architecture is applied on both encoder and decoder part of the overall system. We will elaborate this architecture in details in following chapters.

Reference encoder

The reference encoder is consists of a convolutional stack, followed by an RNN which is adopted from GST architecture [4, 5] and can be seen on Figure 6.2.

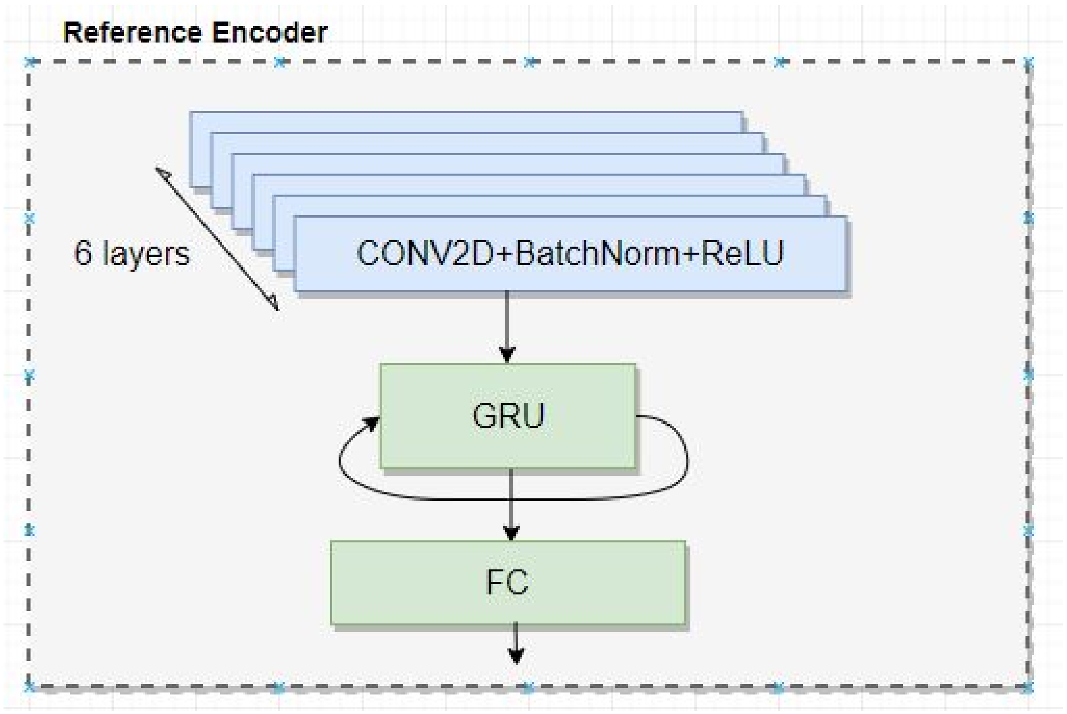


Figure 6.2: Reference encoder.

It takes as input a mel spectrogram, which is, in the beginning, passed to a set of six 2-D convolutional layers with 3×3 kernel, 2×2 stride, batch normalization, and ReLU activation function. Output channels with sizes 32, 32, 64, 64, 128, and

128 were chosen for the 6 convolutional layers. Then the generated output tensor is shaped into 3 dimensions and passed to a single-layer 128-unit unidirectional GRU.

In more detail, the architecture downsampling the reference signal 64 times in both dimensions, the feature size and the 128 channels of the final convolution. The expanded layer serves as the internal size of the resulting matrix, which is used to compress the length sequence generated by the CNN layer into a single fixed-length vector. The RNN used is a single 128-width gated recurrent unit (GRU) layer.

We use the final 128 dimensions output of the GRU as input for a fully connected layer which instead serves as a summary of the whole original sequence.

Emotional matcher

An emotional matcher is a classification neural network which can be seen in Figure 6.3 and we attached it right after the reference encoder. Its task is to learn more discriminative emotion embedding which in turn will be able to better way distinguish different emotion categories. It consists of three fully connected layers, where the first layer has 128 units and the two following have 256 units. We applying Leaky ReLu and Dropout for the first two fully connected layers.

The reason for the choice of Leaky Relu was the dying neuron problem: after training the model with classical Relu, during the inference, we have got almost all the weights equal to zero. Then we used the Softmax layer to distinguish between different emotions. The bottleneck of this approach is the labeling of emotions in the train data that will be discussed in next chapter.

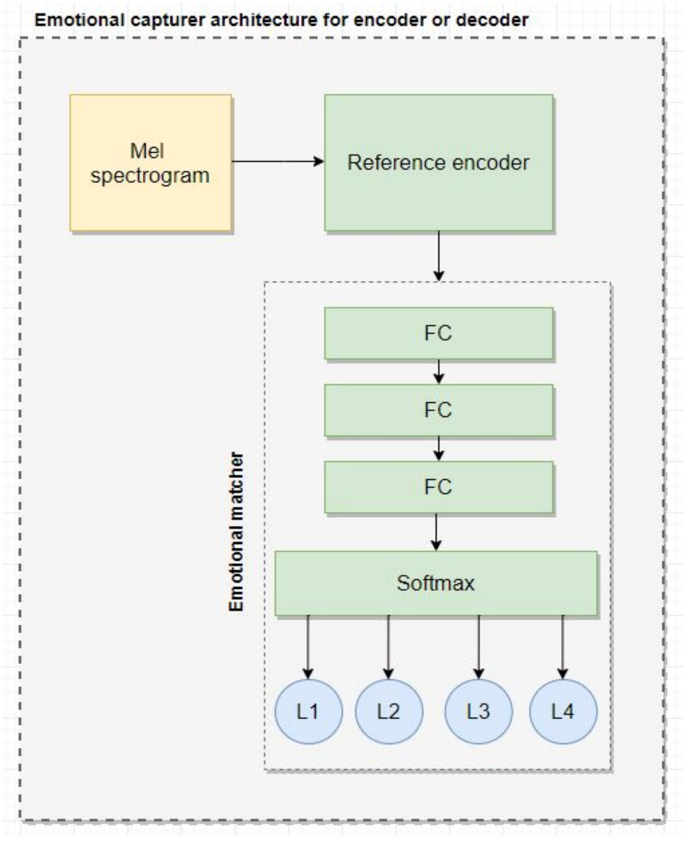


Figure 6.3: Emotional capturer architecture.

6.2.2 Classification Networks + Style Loss

The last module which was implemented in this architecture is style loss. Originally it was used in the computer vision field for stylization task[20] example of which can be seen in Figure 6.4 and one of the losses in this task is style reconstruction loss. It was used for penalizing differences in styles like colors and shapes. The style reconstruction loss is then the squared Frobenius norm of the difference between the Gram matrices of the output and target images. The style reconstruction loss is well-defined even when target and source have different sizes since their Gram matrices will both have the same shape.

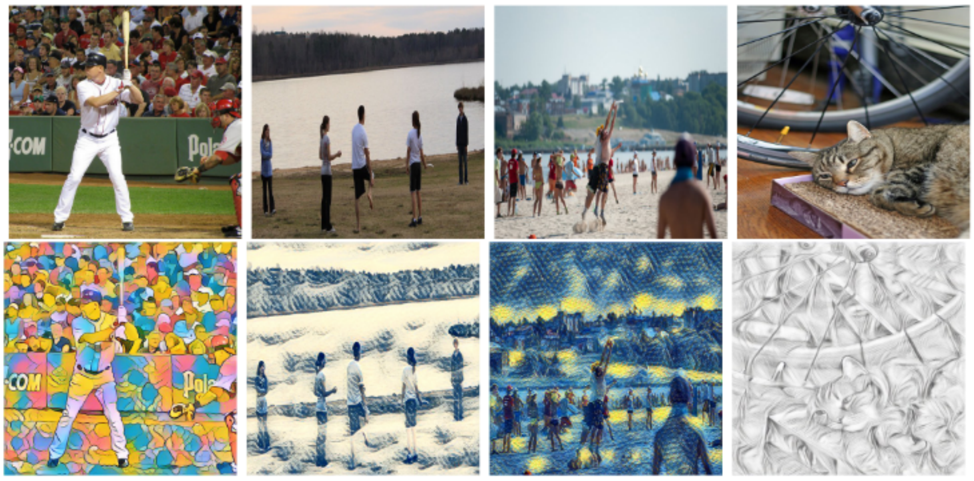


Figure 6.4: Image stylization.

Lucky for us, style loss was adapted for the TTS stylization task[3]. The gram matrix computes patch-level appearance statistics, such as texture, in a location-invariant manner. It is normal to expect that a gram matrix of feature maps calculated from a mel-spectrogram captures local statistics of an audio signal in the frequency-time domain, representing low-level characteristics of sound, e.g. loudness, stress, speed, pitch, etc.

In order to compute Style loss in our architecture, we need to have two representations of mel spectrograms: the ground truth and the predicted one. The first one can be extracted from the latent vector of the input data (result of emotional module) and for the second one we need to use an additional structure in the decoder. This structure is repeating encoder classification: mel-spectrogram predicted by the decoder is passed through the reference encoder and after through emotional matcher.

To be more exact, we are not using mel-spectrograms per se, but we are using their encoded and compressed representations which we are getting when running through a reference encoder and two fully connected layers. Latent vectors which we received from the output of second fully connected layers of encoder and decoder are representing feature maps and they will be used for computing gram matrix for style loss.

Formulas for calculations are shown below.

$$\begin{gathered}\mathbf{G}(\mathbf{y}) = V^T V \\ \mathbf{G}(\mathbf{y_{pred}}) = X^T X \\ \mathbf{l_{style}}(\mathbf{y_{pred}}, \mathbf{y}) = \parallel G(y_{pred}) - G(y) \parallel_2\end{gathered} \tag{6.1}$$

Where V and X are the feature maps of the mel-spectrograms from the reference and the synthesized audio samples, respectively. We compute the gram matrices $G(y)$ and $G(y_{pred})$ as the inner-product of vectorized feature maps V and X, respectively. The Gram matrix captures a feature distribution from the set of feature maps. Our style loss l_{style} is then the $L2$ distance between $G(y)$ and $G(y_{pred})$.

During development of this model we have encountered different datasets and for each of this datset we were trying to train the model with different hyperparameters and slightly different versions of architecture. The last and the best set of hyperparameters can be found in Appendix A.

Chapter 7

EMOTRON: Training Process Details

In this chapter, we will discuss the datasets and corpus used during our work and we will give details for the implementation of our model. We will also talk about the preprocessing operations carried out for each dataset. This chapter is mainly divided into two parts. In the first, we will talk about the speech generation task We will start by talking about the first dataset we used and it is LJSpeech (see Section 7.1.1). This dataset is one of the most used as it contains short audio clips of a single professional speaker and is easy to find. It is also one of the most referenced datasets for TTS tasks with excellent results.

We then moved on to a dataset that also focuses on emotions. We found and used the MELD dataset (see Section 7.1.2). After some preprocessing and testing phases, we moved on to a dataset used specifically for the training of TTS systems, namely LibriTTS (see Section 7.1.3).

Finally, after using these multi-speaker datasets we noticed that the model cannot correctly distinguish the different types of speakers and we looked for a single speaker dataset. We found and tested the Blizzard dataset (see Section 7.1.4), which contains a collection of audiobooks recorded by one person in a very expressive way. We will show the graphs and information related to the training including training and validation error, Mel-Spectrogram, and alignment for each dataset.

In the second part, we will talk about the task of the classification of emotions (see Section 7.2). We will also talk about the techniques applied to obtain the labels that we fed to the network that classifies emotions. Finally, We will see the results obtained in terms of accuracy and confusion matrix with the different datasets.

7.1 Speech Generation Task

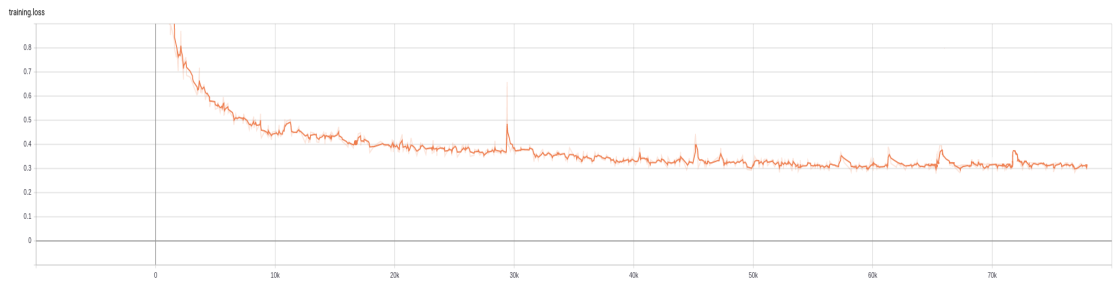
Once the model was developed and created, we tested our model with different dataset: LJSpeech, MELD, LibriTTS and Blizzard.

During this phase for each dataset, we have tried different parameters and small changes to the network. the following images correspond to the results with the best parameters obtained following their tuning.

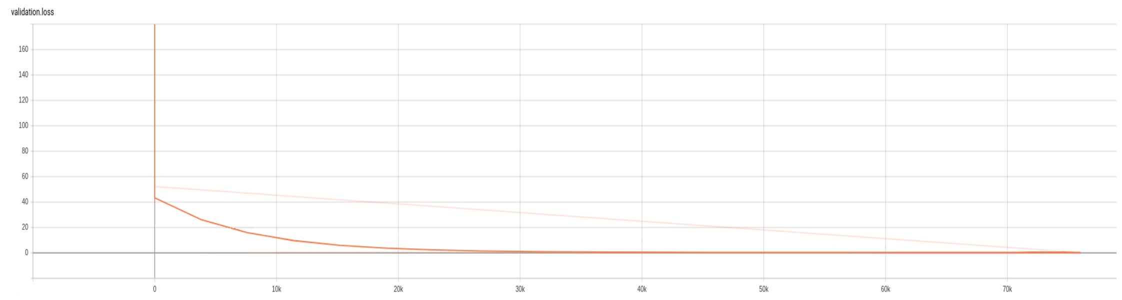
7.1.1 LjSpeech

The first dataset used was LJSpeech. We decided to use this dataset because it is one of the most used datasets in the Text-To-Speech field with currently the best open-source results. It was not necessary to carry out any pre-processing operations since the data had already been divided into utterances and the additional audio parameters were the optimal ones.

There was no need to make any changes to the network structure and not even a major phase of parameter tuning. Decent results were obtained relatively early. On the other hand, as far as expressiveness is concerned, the results are fair and not so excellent. This is most likely due strictly to the dataset, with a number of hours sufficient to generate good speech but with a not so wide variety of expression. We were, therefore, able to show the training phase. The details of the training process can be seen in Figure 7.1.



(a) Training Loss

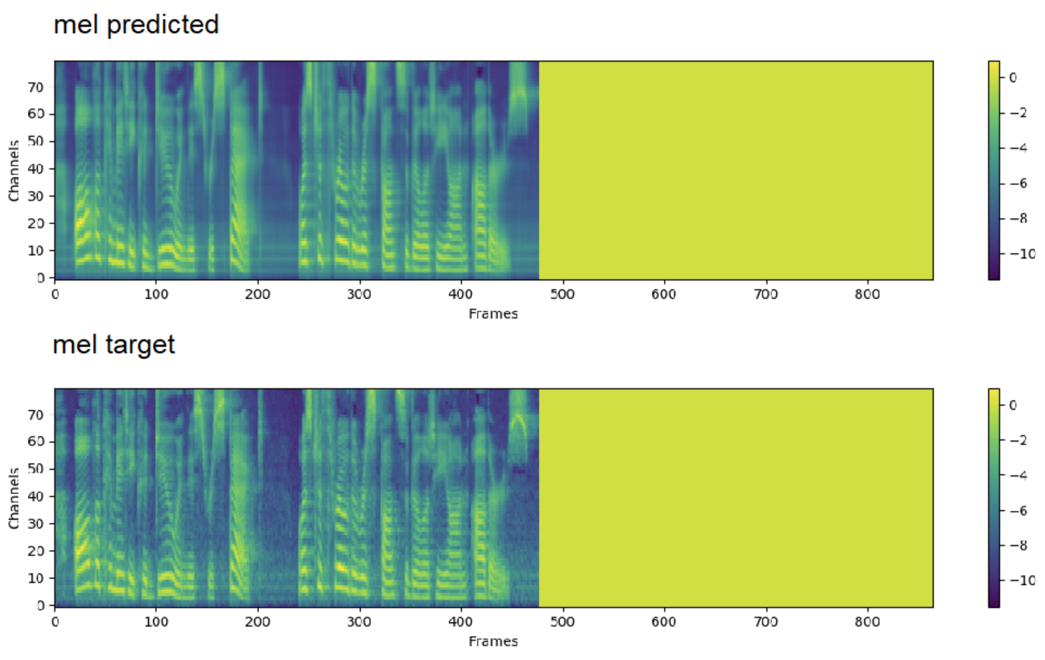


(b) Validation Loss

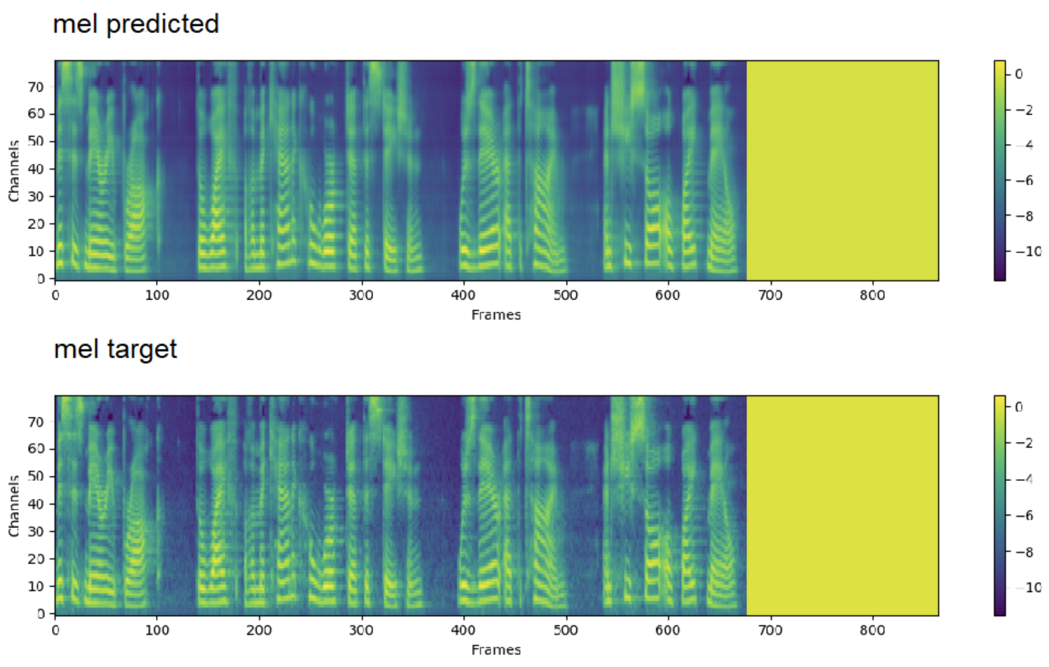
Figure 7.1: LJSpeech Validation and Training Error

Training had 70k iterations with a learning rate 10^{-3} and batch size 32. As we can see training curve has reached its plateau at 0.3 training loss. Validation loss is sufficiently small and the value is similar to the training loss. We can also see how the curve descends quite rapidly towards its plateau value.

To further analyze the result of the model we can check target and predicted Mel-spectrograms. First we can see them at iteration 15k (Figure 7.2a). As we can see, the predicted spectrogram is almost accurate and will produce a decent result. If we will check the result at iteration 64K (Figure 7.2b) we can see that results are better, especially in areas where there is a break it can also capture more details than in the previous steps.



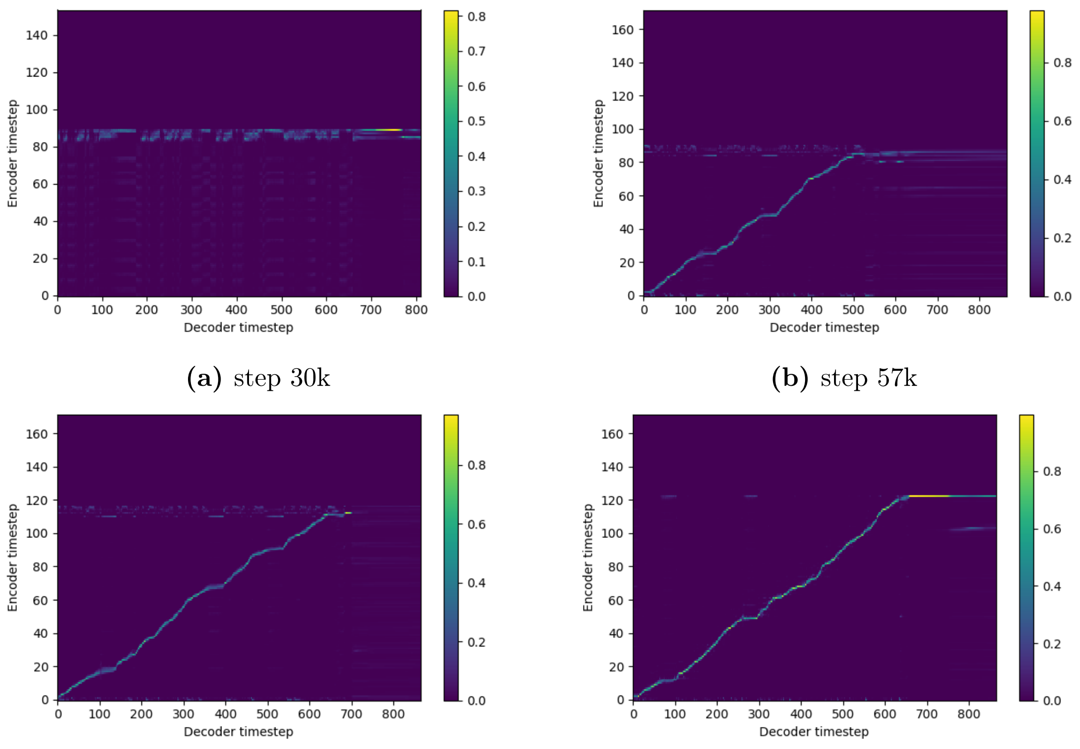
(a) Mel-Spectogram at step 15k



(b) Mel-Spectogram at step 64k

Figure 7.2: LJSpeech Mel-Spectogram

The last analysis point is alignment, we can see the trend of it in Figure 7.3. As we can see from these alignments, initially, there is not the usual diagonal behavior of a good model but it is flat. However, after some steps, the alignment begins to take the right shape leaving only some gaps at the end and some fluctuation. In the last alignment the graph is diagonal and there are barely those small bands near the peak.



(a) step 30k

(b) step 57k

(c) step 64k

(d) step 72k

Figure 7.3: LJSpeech Alignment details

7.1.2 MELD

After success with neural speech synthesis, we decided to look for emotional datasets in order to learn the expressiveness of speech. The first dataset which was found was the MELD dataset. As described in Section 4.2 MELD consists of utterances from the famous sitcom "Friends". It is worth mentioning that the original dataset is not appropriate for TTS usage. During the first training round we found out that there are several reasons for this:

- The data contains a lot of noise
- Not annotated laughing in data
- Many inaccurate audio-to-sentence alignments
- Some utterances contain multiple character speech

These issues cause training to completely fails to produce any alignments and in inference, time was only white noise. To handle these issues following cleaning steps were performed:

- Denoising data using SoX library
- Manually inspection of each out of 10k files to split up files with multiple speakers
- Manual fix of errors in the text
- Change laugh in audio into [laugh] token

After these steps training process seemed to have started working. In this phase, in addition to the preprocessing, several parameters of the network have been changed.

For example, we tried to play with the size of the speaker embedding (part of the encoder) this because we thought that with a larger size the network would be able to distinguish the different types of speakers present. But that wasn't enough by just making training processing slower without any benefit.

Training had 780k iterations with a learning rate of 10^{-3} and batch size 4. Such low batch size was due to training on local PC. As we can see training curve has reached its plateau at 0.4 which is bigger than Ljspeech training loss. Validation loss is sufficiently small and has the same trend as the training loss. The details on the training process can be seen in Figure 7.4

To further analyze the result of the model we can check target and predicted Mel-spectrograms. First we can see them at iteration 36k in Figure 7.5. As we can see, the predicted spectrogram is not accurate and it will produce bad results. From the images, we can see how the spectrograms are not well defined with not well recognizable waveforms. But this is not only the case for the spectrogram produced but also for the target audio file and consequently predicted spectrogram is not accurate and it will produce bad results.

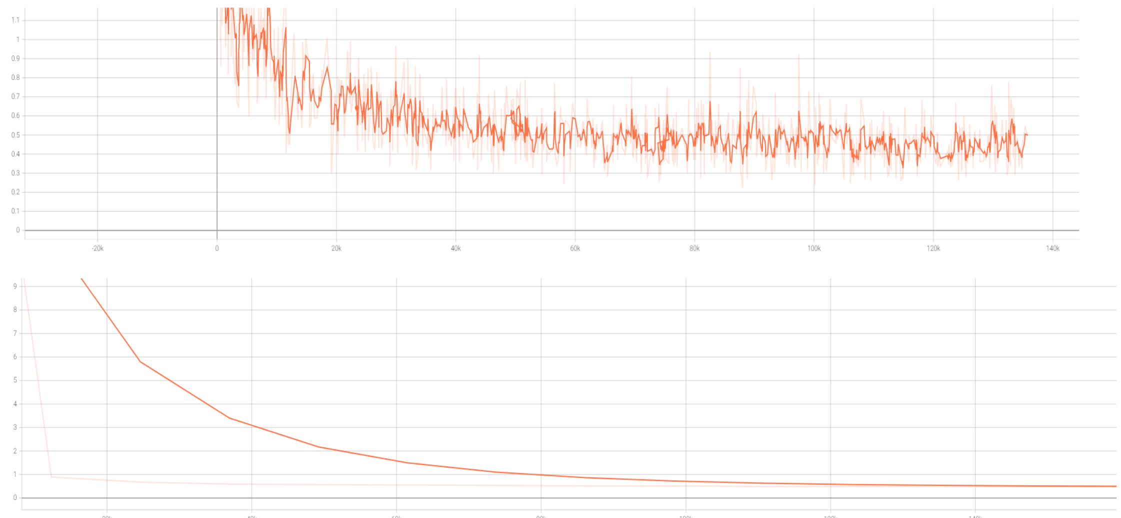


Figure 7.4: Training and validation process with MELD.

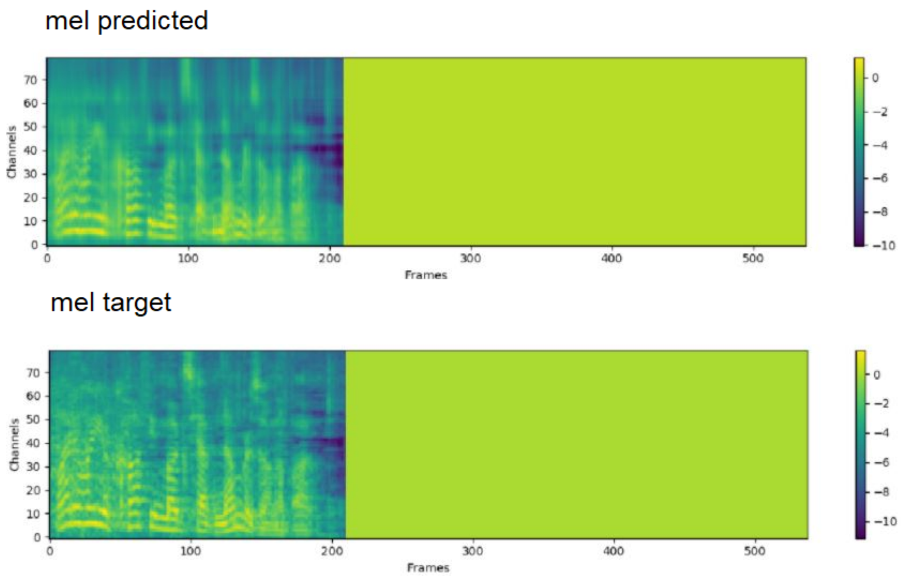


Figure 7.5: Target and predicted Mel-spectrogram MELD at 36K iteration.

If we will check result at iteration 738K depicted in Figure 7.6 we can see that results are slightly better,Although the network has been trained for quite a long time, it has not been able to give a minimally satisfactory result in terms of spectrograms and consequently there is noise and inaccuracy.

The last analysis point is alignment, which completely failed with the MELD dataset. We can see the example of it in Figure 7.7. As we can see from these alignments, there is always the presence of disturbance, and alignment never even tries to form the diagonal. We believe that the current dataset fails due to lack of data and the amount of remaining noise in data and probably this dataset does not suit our architecture well.

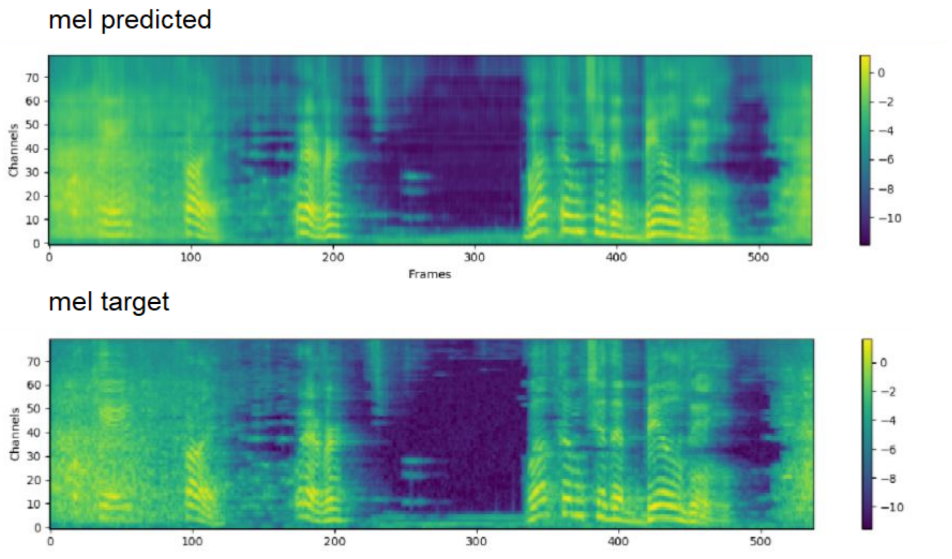
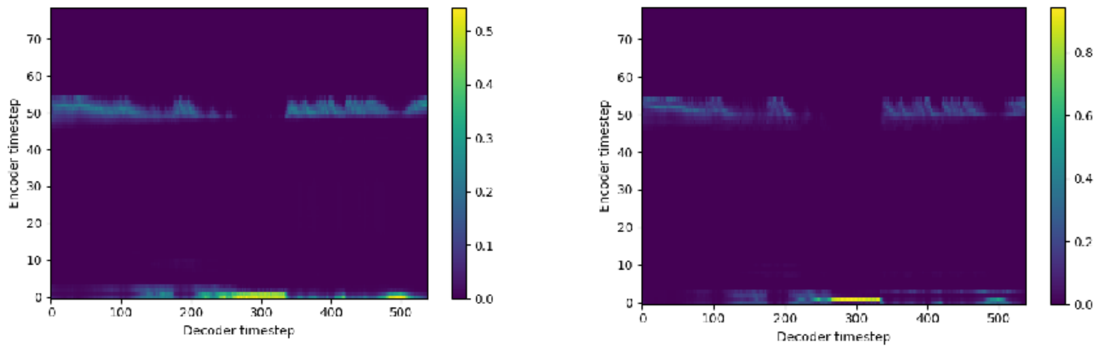
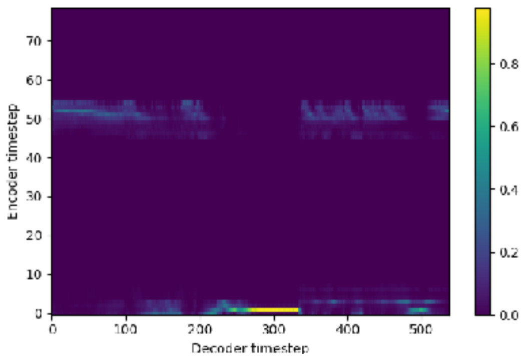


Figure 7.6: Target and predicted Mel-spectrogram MELD at 738K iteration.



(a) step 49k **(b)** step 295k



(c) step 738k

Figure 7.7: MELD Alignment details

7.1.3 LibriTTS

From the analysis of MELD dataset results, we understood that we need to have a larger dataset with cleaner quality. And we have found LibriTTS which consists of 586 hours of audio and has 2456 professional speakers.

We used two different approaches for training this network. Furthermore, as regards the dataset we used a reduced portion of the complete dataset, in particular, we used the version reduced to a number of hours equal to 54. This is because it is true that with such a high number of hours we reduce the risk of overfitting but it is also true that the memory used increases considerably.

With the first approach, we performed training without any preloading of the weights obtained from other datasets. So we still had the possibility to modify the network structure (mostly the encoder module), this solution is not the one that will be shown later, since, following the comparison between the two approaches, the second slightly speeds up the process compared to the current one.

In the second approach, the model was trained using warm start, and in this case, we had to use the same parameters as the checkpoint used. The baseline pretrained model was LJspeech, which was trained before. It's the first time that the adaptive learning rate was implemented to the model and batch size was set to 16.

And at first glance on the resulting plots depicted in Figure 7.8, we can see that the training error has dropped and quite fast. Now it is 0.25 with respect to 0.4 of the MELD dataset, it seems promising and we can analyze the results of target and predicted Mel-spectrograms.

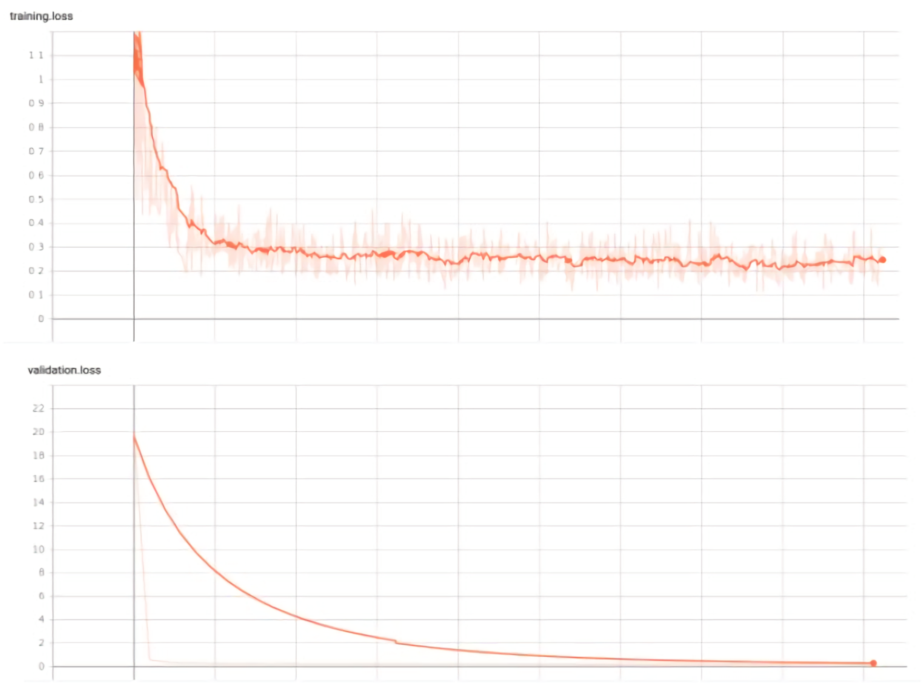
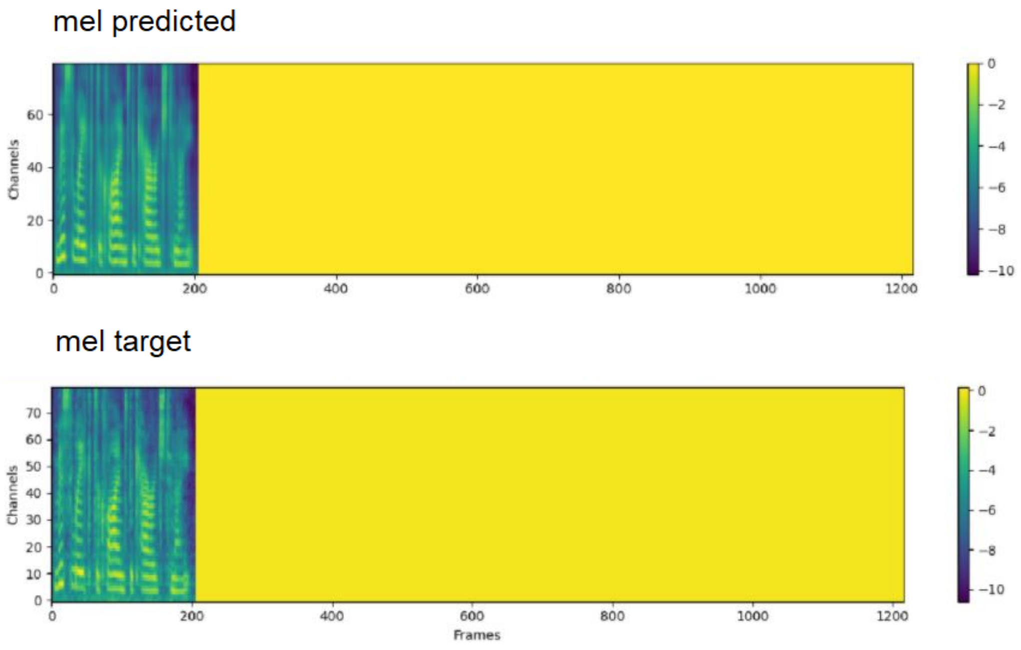
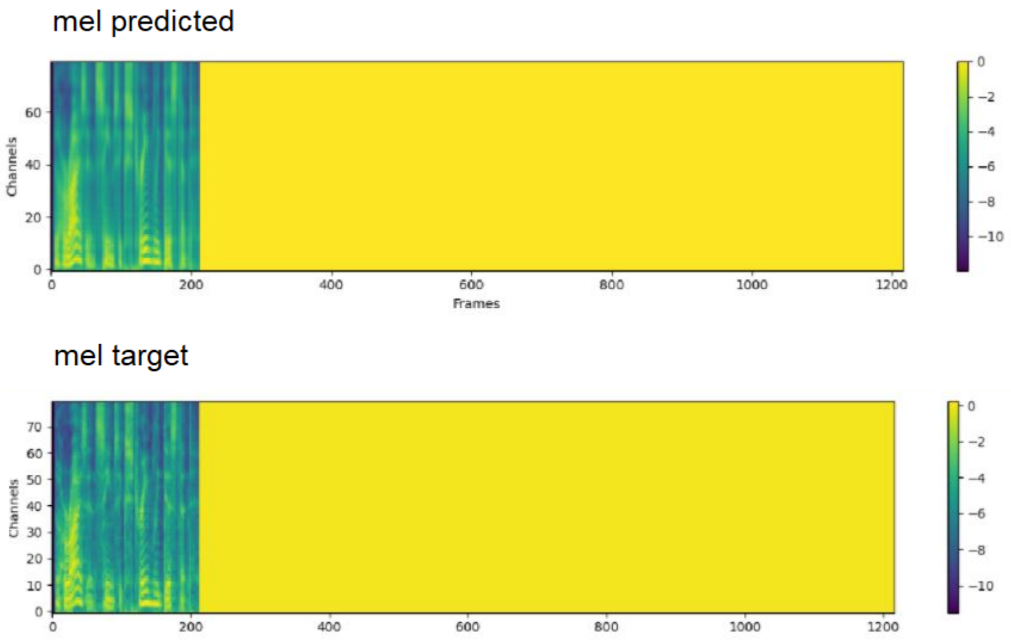


Figure 7.8: Training and validationprocess with LibriTTS.

In Figures 7.9a and 7.9b we can see target and predicted spectrograms. Here the images both for reference and those obtained certainly seem better than the dataset MELD. However, they don't seem to have a well-defined spectrum like LJSpeech's. Further metrics are needed to be evaluated to understand if this dataset is actually compatible with Emotron.



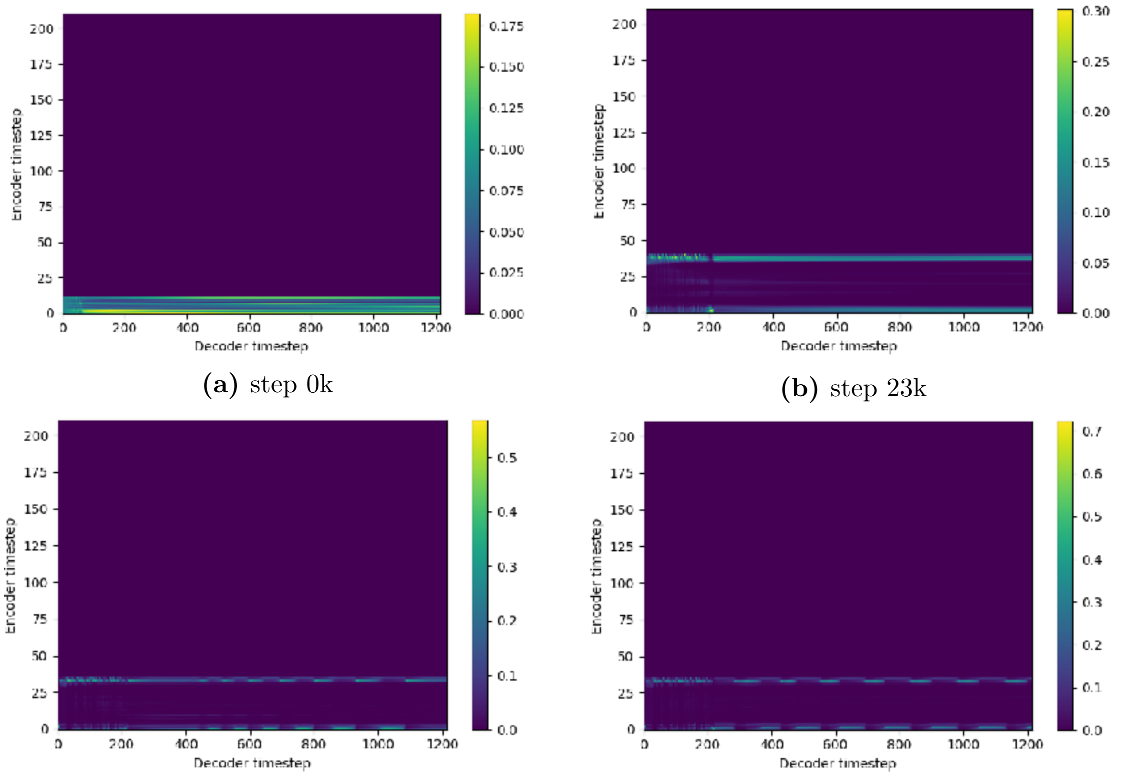
(a) Target and predicted Mel-spectrogram at 56K iteration.



(b) Target and predicted Mel-spectrogram at 182K iteration.

Figure 7.9: LibriTTS Mel-spectogram

Finally, we have the alignment in Figure 7.10 at different steps of the training process. It turns out completely flat and shows no sign of any improvement, the model cannot find the proper way to create a diagonal so at inference time it produces a random set of mumbling. We think that the main problem of the current dataset is the multi-speakers component (247 voices). The model simply cannot learn patterns to distinguish different voices. There is a need for an expressive single speaker dataset.



(a) step 0k

(b) step 23k

(c) step 68k

(d) step 182k

Figure 7.10: LibriTTS Alignment details

7.1.4 Blizzard

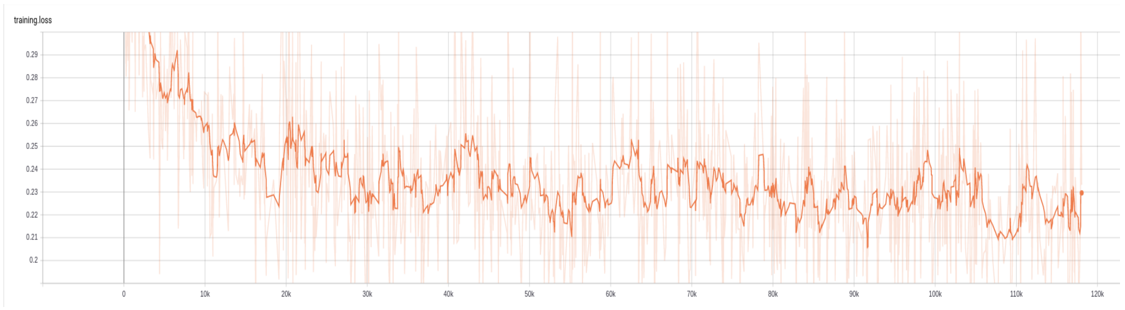
After testing several multi-speaker datasets we concluded that our model is not able to distinguish the different patterns of each person for each type of style. And that's why here we discuss the results obtained using a single speaker dataset instead. Unlike LJ Speech, the Blizzard dataset contains a similar number of hours for the segmented version but is much more expressive.

There are two versions of this dataset, an already segmented version reduced and a complete but not segmented. We initially used the already segmented version and a few preprocessing operations are enough to start the training (preparation of text and corresponding audio file and sampling rate conversion).

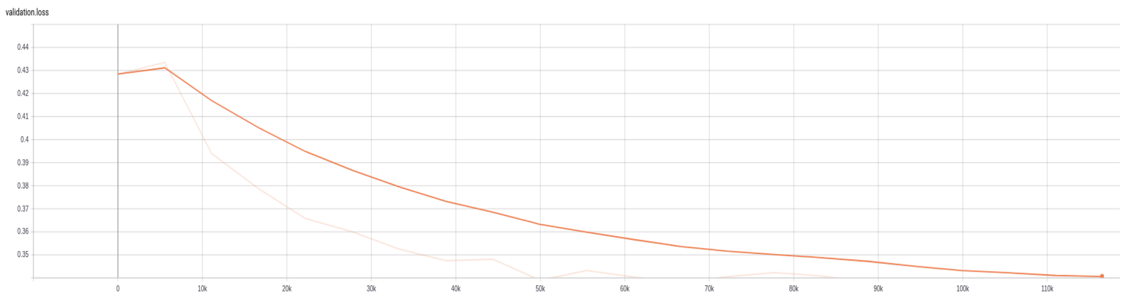
The training was done without using any warm start and the batch size used is 32, despite the size the dataset is well optimized to be able to have a much larger batch size than previous datasets. We decided not to use the LJSpeech weights to try right away to capture the pattern of the different styles without being influenced by the previous dataset.

Concerning this chapter and therefore to the creation of speech, the two versions of the dataset are both excellent if not the best. The abridged version already contains enough hours to produce good quality speech. The full version of Blizzard was used with warm start using the weights of the same reduced dataset. This allowed us to reach convergence faster. The data we are going to show belong to the best result obtained using the complete dataset with warm start.

Let's start by analyzing the train and validation error plot in Figure 7.11. We can see that compared to LJSpeech the error decreases much faster reaching a plateau of 0.21-0.22 similar to that of LibriTTS. As for the validation loss, it also falls much faster and reaches the lowest value among all the previous datasets.



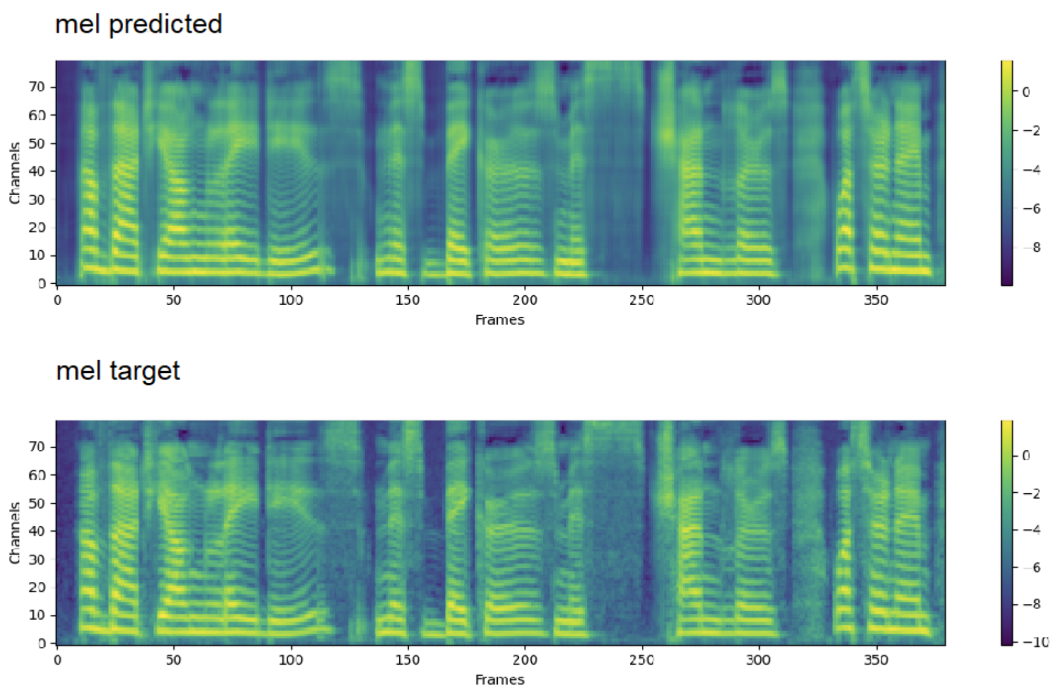
(a) Training Loss



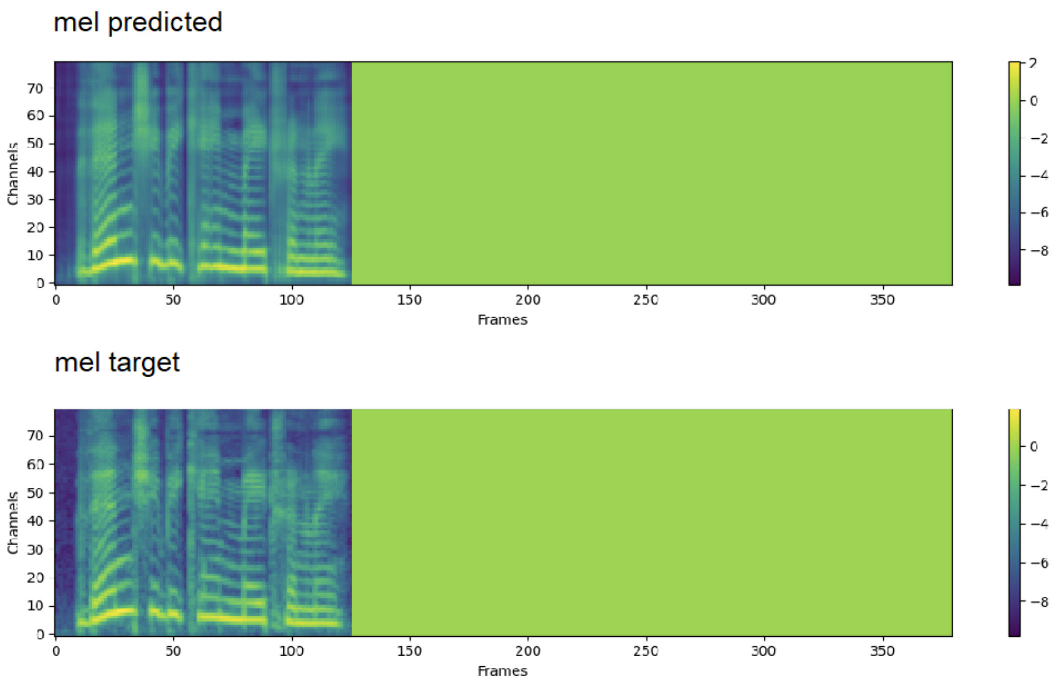
(b) Validation Loss

Figure 7.11: Blizzard Validation and Training Error

Continuing with the analysis we can see the target and predicted spectrograms on Figure 7.12. Here we can notice that immediately the results produced are very good and continue to improve with the increase in steps. We can also notice how the waveforms are really defined both in short and relatively long sentences.



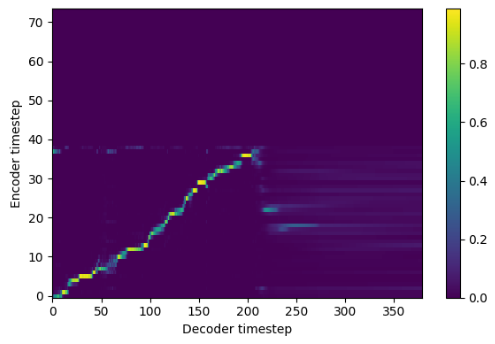
(a) Mel-Spectogram at step 16k



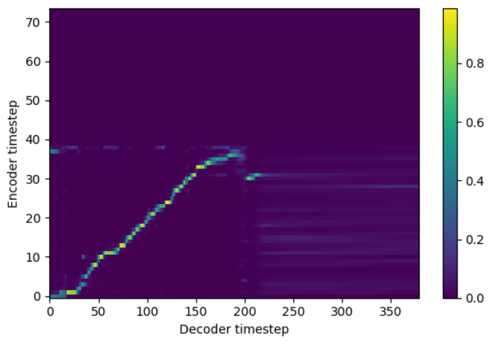
(b) Mel-Spectogram at step 116k

Figure 7.12: Blizzard Mel-Spectogram

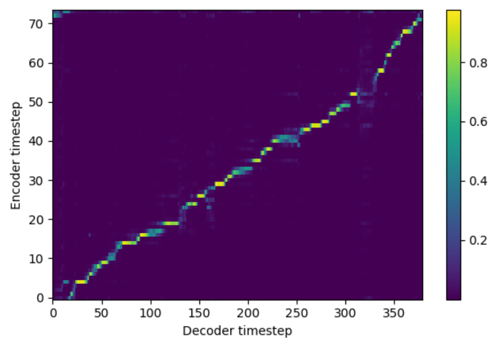
The last point to analyze is the alignment. As we can see from Figure 7.13 it is noted from the first steps that the alignment takes the typical diagonal shape, but initially, there are some "falls" and interruptions. As the training continues, the alignment assumes precisely the diagonal shape without any kind of "hole". Compared to the values obtained with the other datasets, the alignment here is excellent right from the start, partly due to the warm start procedure.



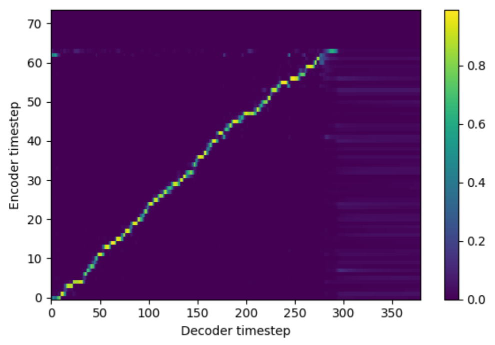
(a) step 5k



(b) step 11k



(c) step 44k



(d) step 61k

Figure 7.13: Blizzard Alignment details

7.2 Classification Task with Emotron

The second task we worked on was emotion classification as one of the problems of GST is that it is not being able to obtain a speech in which you can choose the type of emotion a priori, but what you get is a generation of different styles that do not always correspond to different types of emotions.

Our idea was to create a Text-To-Speech model in such a way that you can choose the type of emotion (or reference audio) before running the code and it will modify audio during the inference. This modification is done mostly by the emotional classifier which we have developed. To achieve this we have created a network in which, through the study of the features of the audio files, it is possible to classify emotions. The values that should represent an embedding of emotions are then linked to the embedding of the text that will be fed to an attention model.

During the development of the network for the recognition of emotions, we looked for the most used datasets in this task. We then found some datasets, including RADVESS and TESS. Several tests have been carried out with these datasets, but one of the problems encountered is the number of hours of speech.

In a case like that of the recognition of emotions alone, they are very valid datasets, unfortunately in the speech synthesis case, they are not enough. This is because EMOTRON performs training in parallel (emotion classifier + mel-spectrogram synthesizer) and the number of hours of the dataset is very low for the speech task. Furthermore, we have not achieved satisfactory results with different metrics that would keep us going further.

Two main datasets were tested. The first one used was IEMOCAP, which is a dataset with already labeled emotions. The second was Blizzard. In the second case with Blizzard, we have traveled different paths since this dataset is very expressive but does not already contain the labeled emotions. To get the labels we first used different networks to classify emotions with accuracy greater than 80%, experiencing a different set of emotions each time.

Later we also used some clustering techniques after extracting the important features that characterize emotions. To evaluate the model training and understand how efficient the training was proceeding we used several metrics. Later for each variant of the dataset used we will show the confusion matrix obtained (from the validation set) and the accuracy achieved. In this phase, we have decided to exclude MELD and LibriTTS, following the bad results obtained with our network in the part of speech generation.

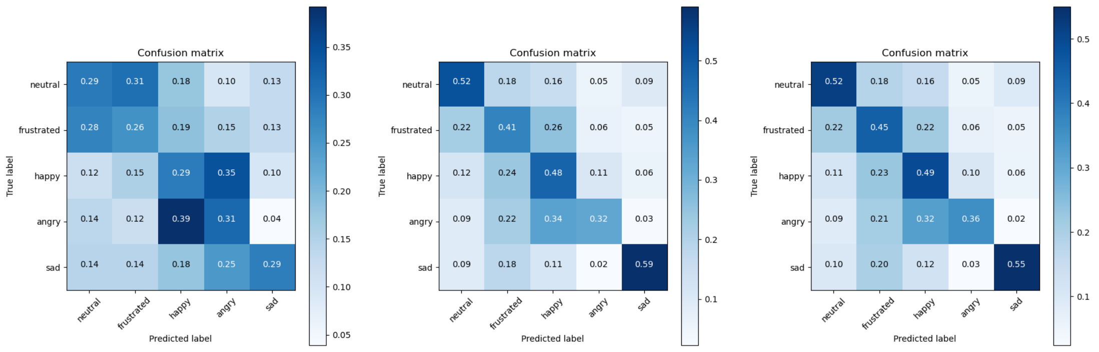
7.2.1 IEMOCAP classification

In this section, therefore, we will discuss all the operations carried out and the results obtained with the IEMOCAP dataset. We have not previously considered this dataset as it was not the most suitable for the sole purpose of speech generation. The number of emotions that people have tagged is very high even for networks that classify emotions (see Section 4.6).

We did several operations with IEMOCAP, and one of the first choices we had to make was the number of emotions our classifier needed to recognize. Initially, we decided to use 5 emotions, the most common emotions were chosen which can also represent a sub-category of a specific emotion. We also take into account the proportion between the various labels to make the dataset unbalanced. The 5 emotions chosen are: *neutral*, *frustrated*, *happy*, *angry* and *sad*.

With this configuration, we were able to obtain a more or less balanced dataset. We have decided to include the frustrated emotion because it has great relevance in terms of hours within IEMOCAP. Given the shortage of hours compared to a dataset used exclusively for speaking, we felt it was necessary to combine different emotions.

The emotion *fear* was added with *sad*, *disgust* with *angry* and *excited* with *happy*. During the training phase, we made several changes to the network (without modifying the part for the speech creation) and we did hyperparameter tuning. After these steps, it was possible to move on to the testing phase to analyze the result obtained.

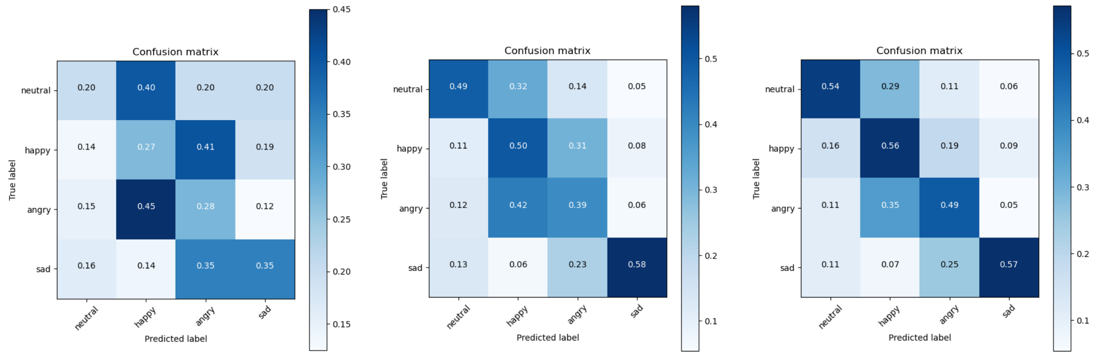


(a) step 6k **(b)** step 28k **(c)** step 48k

Figure 7.14: IEMOCAP Confusion Matrix 5 Emotion

First, we can analyze the confusion matrix obtained in the different steps from the results. Initially, we can see in Figure 7.14 how the values are quite random and there is no real distinction between the various emotions. But over time, we can see how the network begins to recognize the different emotions, albeit not always correctly. The accuracy value is around 50.5%, very particular is the fact that the "angry" emotion is labeled many times as "happy".

Not considering the results satisfactory enough, we decided to facilitate the work of the network by decreasing the number of emotions to be recognized. We have removed the *frustrated* emotion by having a slightly more unbalanced dataset than before but made the classification problem easier. Before starting the training with this new label there was a need to find the new proportion between the different labeled emotions. In doing so we applied a vector weight during the computation of the loss to give a greater weight when the emotions present in smaller quantities in the dataset were recognized.



(a) step 6k **(b)** step 28k **(c)** step 48k

Figure 7.15: IEMOCAP Confusion Matrix 4 Emotion

Analyzing now the confusion matrix in Figure 7.15 we have a behavior very similar to that described previously. However, in this case, we notice that from the beginning of the training the network already recognizes some emotions. With the continuation of the training, the results improve, and compared to before we have results that are on average better especially for sad and happy emotions. In fact, now, the accuracy value on the validation set has increased to 55%. If we analyze the results that the network predicts, we can still see how the network struggles to predict angry emotion by confusing it with happiness. Before making major changes to the network, we used another dataset that we will discuss in the next section.

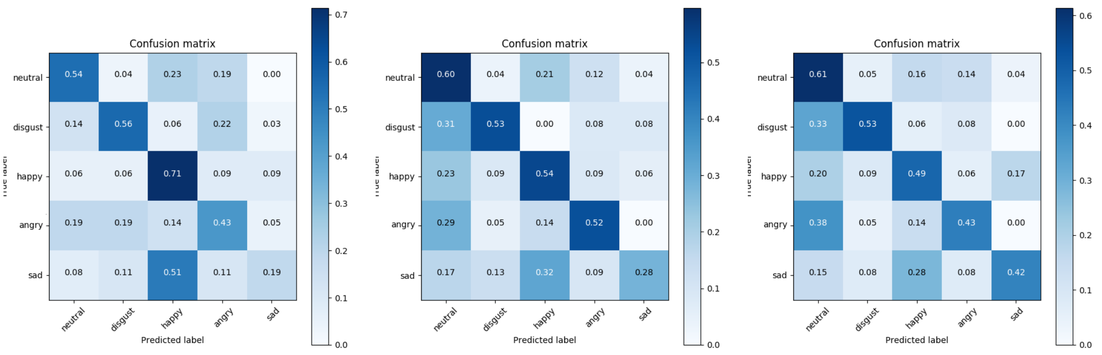
7.2.2 Blizzard classification

The dataset we decided to test following IEMOCAP was Blizzard. One of the main reasons why it was chosen was due to its excellent results obtained in the generation of speech. This dataset is characterized by being very expressive, so we decided to get the labels by ourselves to use on the network. We used several methodologies.

Our first approach was to use a neural network developed for the recognition of emotions given in input an audio and the corresponding text. We were kindly allowed to use PathOSNet with its pre-trained checkpoint. This network was developed by our colleague Federico Galati, using the IEMOCAP dataset during training, reaching an accuracy percentage of about 80%. The network output for each audio corresponds to one of the four emotions between *neutral*, *happy*, *sad*, and *angry*. Once we got the results we went manually to test some predictions produced by the network and we noticed that with the blizzard dataset there was a certain bias towards the happy emotion.

To overcome this problem we found another network with an accuracy of around 80% as well [39]. This network recognizes a greater number of emotions: neutral, calm, happy, sad, angry, fearful, disgust, surprised. This allowed us to make a direct comparison with IEMOCAP having the same number of emotions. After obtaining the results of this new model we combined the emotions calm and neutral, sad and fearful and we did not consider surprised as they were a small number. Finally, we used the ensemble of two models as a voter for choosing the emotion, more precisely:

- If the label is different from happy, take the result predicted from PathOSNet.
- If the label predicted on PathOSNet is equal to happy and
    1. The label predicted from the second model is happy the result label is happy.
    2. The label predicted from the second model is different from happy the result is the label predicted from the second model.



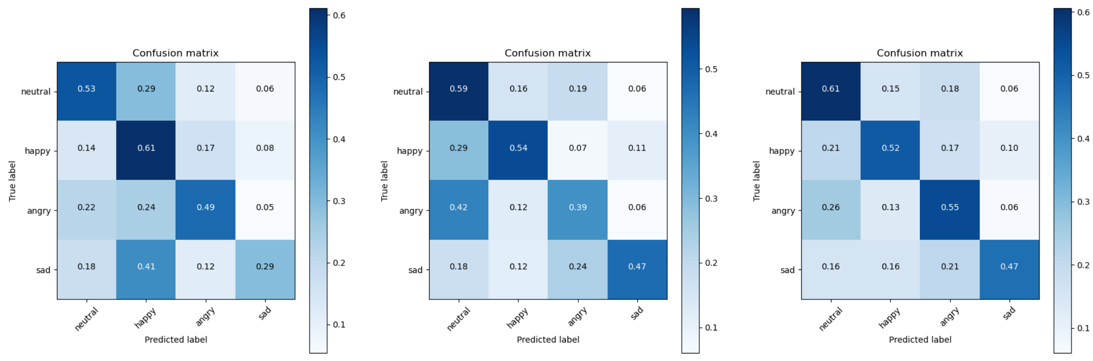
(a) step 25k **(b)** step 40k **(c)** step 50k

Figure 7.16: Blizzard Confusion Matrix 5 Emotion

We have thus obtained the labels for Blizzard and they are: *neutral*, *disgust*, *happy*, *angry*, *sad*. During the training phase, the same structure and hyper parameters used previously with IEMOCAP were initially used. Below are the best results after the various tuning.

Analyzing the confusion matrix in Figure 7.16 we note that now the number of steps necessary has increased but on the other hand also the dataset has a larger size, moreover we have slightly better results than the previously used dataset. We are not in the presence of a contradictory value for angry emotion as before, the score obtained on the validation set for what concern accuracy is equal to 54%.

As before we have reduced the number of emotions by removing disgust emotion and adding it to the angry labels. As we can see from Figure 7.17 now the results are even better, in fact, after removing the "disgust" label there has been an increase in the happy and sad labels. The accuracy value is 57%. Although the result has improved, we wanted to try an alternative approach. We decided not to use the labels obtained from the models based on neural networks.



(a) step 25k **(b)** step 40k **(c)** step 50k

Figure 7.17: Blizzard Confusion Matrix 4 Emotion

Final idea was to use the clustering technique. For this, we used the OpenSMILE [30] library to extract features responsible for the expressiveness of audio, in particular, we used the feature set called emobase. This set contains the acoustic features for emotion recognition including Intensity, Loudness, 12 MFCC, Pitch (F0), Probability of voicing, F0 envelope, 8 LSF (Line Spectral Frequencies), Zero-Crossing Rate. In the end, we have got 120 features, we apply PCA[1] to reduce it to 10 with 90 percent of variance explained and applying K-means clustering to get until 4 distinguishable groups.

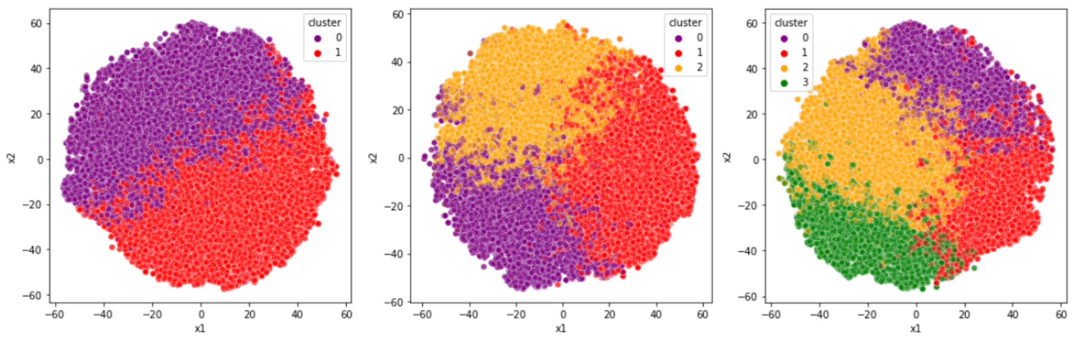
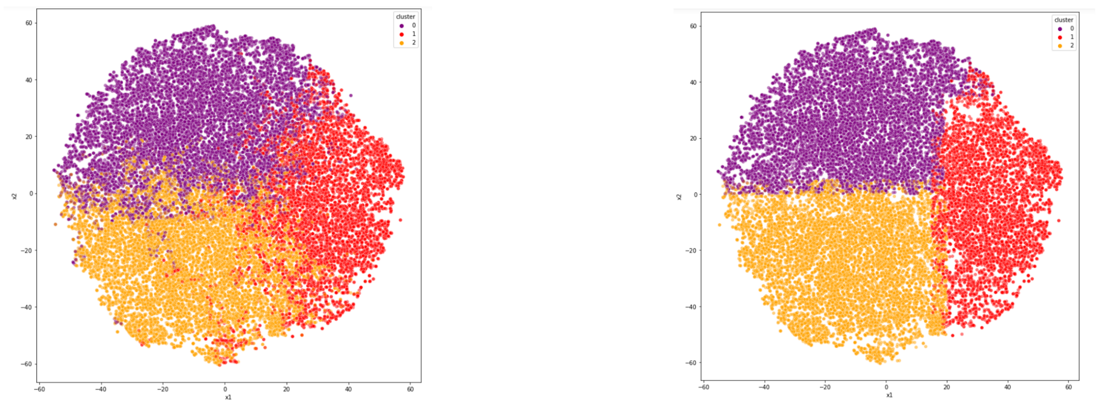


Figure 7.18: Different cluster dimension

As can be understood from the image in Figure 7.18, quite good clusters were obtained in almost all the experiments. We decided to use the experiment with 3 clusters as it is well divided and proportionate, 2 clusters are too few and 4 clusters are badly divided.

The only remaining problem is the presence of some points that are not easily distinguishable from one cluster to another. We have carried out some outlier removal operations and we have eliminated the points on the border between the various clusters. The final result is shown in Figure 7.19.



(a) Before filtering **(b)** After filtering

Figure 7.19: Cluster filtering

Now we no longer have labels that represent different emotions, but we have styles.

[1]Principal component analysis

Below are the results, first we can see in Figure 7.20 the value of train and validation accuracy. the curve grows fast enough and reaches its plateau almost immediately. The graph shows that the maximum accuracy value on the validation set is about 66%.

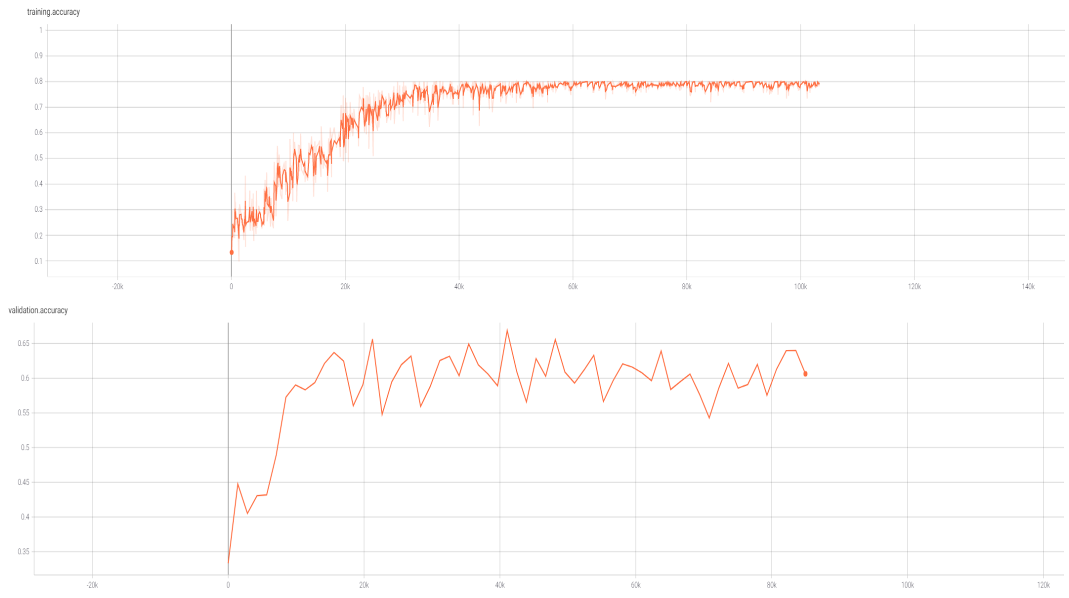
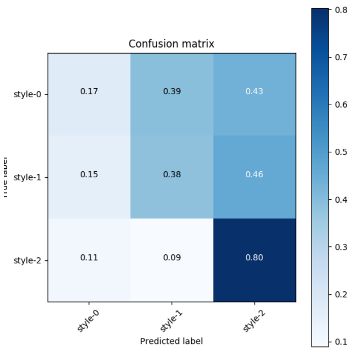
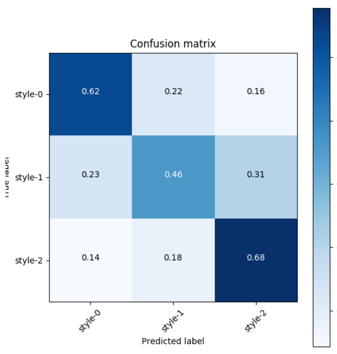


Figure 7.20: Training process with style.

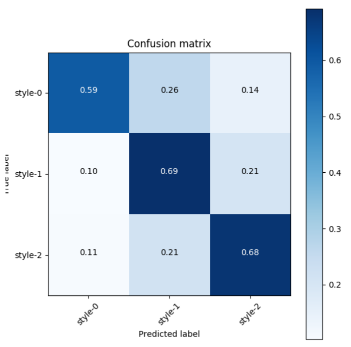
Continuing with the analysis we have the confusion matrix in Figure 7.21, where we can see how the network is able to distinguish the different styles. Especially the first and second styles hardly ever confuse it with style 0, while styles 1 and 2 sometimes get it wrong with each other.



(a) step 10k



(b) step 28k



(c) step 70k

Figure 7.21: Blizzard Confusion Matrix 3 Style

7.3 Discussion

In this chapter, we have retraced the main steps with our EMOTRON model. In the first part, we talked about the task regarding the generation of speech. In this section, we have explored several datasets understanding that our model is not suitable for multi-speaking. This is most likely because it is very difficult for our model to distinguish different speakers (as in the case of LibriTTS where there is a large number of speakers) and at the same time be expressive and capture the different patterns for each different speaker.

So the most compatible dataset with our model is Blizzard. It contains a very high number of hours and has one of the best qualities of all the datasets tested. The MELD dataset has had very bad results because it is not very suitable for Text-To-Speech where a high-quality spectrogram is required to generate expressive and fluent speech. Note that the network requires a large amount of memory and data to achieve optimal results.

While as regards the classification of emotions, we did not find the results obtained with the IEMOCAP dataset satisfactory. We expected better results, it is a dataset built specifically for this task. While the metrics obtained with Blizzard are quite good especially following clustering operations. Moreover, this dataset is very expressive but it is not characterized by having explicit emotions and in the end, the results are satisfactory as we can see from Table 7.1.

We believe that our model fails to classify very optimally because its architecture is based on parallel training of emotion predictor and Mel-spectrogram synthesizer. This means that we cannot have a very complex architecture for the classification of emotions in terms of size of embedding space. If the model will have embedding of reference audio which is overweight text embedding the attention part would no longer be able to distinguish the emotions from the embedding of the text.

Table 7.1: Validation Accuracy

System	Accuracy
IEMOCAP 5 emotion	51%
IEMOCAP 4 emotion	55%
Blizzard 5 emotion	54%
Blizzard 4 emotion	57%
Blizzard style cluster	66%

Chapter 8

Evaluation and Results

There might be millions of ideas and hypotheses in the world because it is just an observation that has not been proven to be true in practice. After having enough accumulated evidence which supports a hypothesis, it becomes a theory. As we propose a new idea on expressive TTS synthesis, in order to prove its usefulness this proposal must be theoretically substantiated, practically feasible, and tested in various ways.

In Section 3.1, and Chapter 6 we have already provided a theoretical explanation on point of our discussion concerning EMOTRON, and in Chapter 7 we have shown practical details for proposed architecture, however, there is still one important piece missing. This piece is a model evaluation in inference time.

In this chapter, we will initially describe what metrics we have used to evaluate the quality of our Text-To-Speech model and report details of experiments that we have organized (see Section 8.1). Later we will show the results obtained regarding the experiment carried out (See section 8.2). The experiments aim to evaluate two different aspects.

The first aspect is to check the quality of the EMOTRON audio in comparison with the competing model by comparing the MOS metric. The second objective is to evaluate the ability of the network to produce emotions. In this case, we used the confusion matrix already seen in Chapter 7. It allowed us to compare our emotional module with GST.

8.1 Metrics for Evaluation

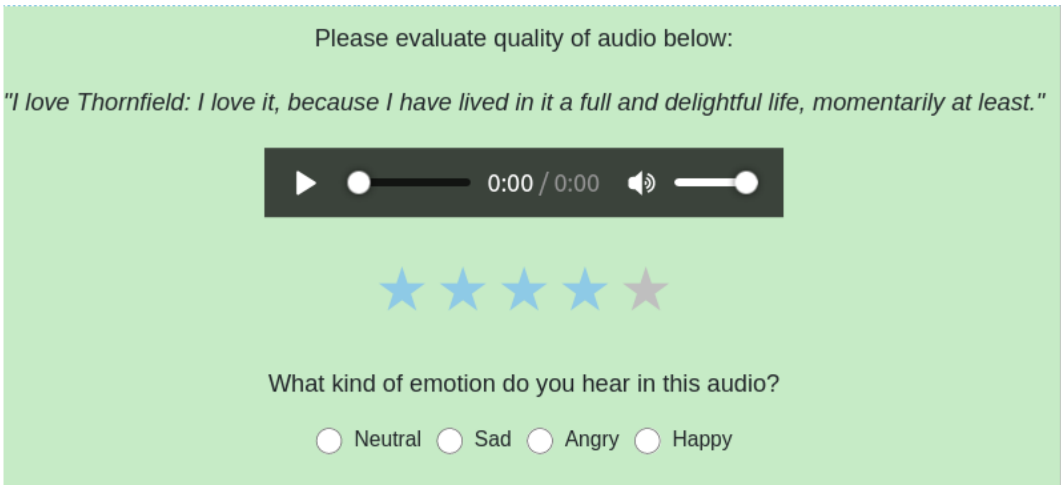
MOS is one of the most popular metrics for estimation the quality of synthesized audio. It was used for evaluating the quality of TTS models in many papers [11, 17, 23, 24]. The idea behind the MOS is simple: ask people to rate audio samples produced by the model and return an average value of these ratings.

Since there is no explicit evaluation of expressiveness in our model, we decided to ask for this data in the questionnaire. For convenience, we decided to collect data for MOS and emotional accuracy in one survey.

To conduct a survey with a nice and easy user experience we created a website using HTML/CSS with bootstrap so it can be adaptive for any kind of device.

We choose to test 4 types of emotions: neutral, angry, sad, and happy. For comparison reasons and objectiveness of results, we decided to collect scores not only for EMOTRON, but also for Tacotron2 with GST and original utterance.

The final structure of the survey consists of 12 questions: 3 synthesized audio for each emotion. The example of the question block can be seen in Figure 8.1.


Please evaluate quality of audio below:

"I love Thornfield: I love it, because I have lived in it a full and delightful life, momentarily at least."

What kind of emotion do you hear in this audio?

Neutral Sad Angry Happy

Figure 8.1: Question block of survey.

In desription of survey, we gave an guidance what each star represent in terms of audio quality:

- 1 - bad, unrecognizable speech
- 2 - poor, speech is barely recognizable
- 3 - fair, acceptable quality of speech, small errors allowed
- 4 - good, speech with proper pronounciation and quality
- 5 - excellent, perfect speech, sounds natural and expressive

MOS is estimated using data collected from stars and emotional accuracy score from the second question. This procedure is done on all twelve questions.

8.2 Achieved results

After launching the survey we have left it on the web for 3 days. During these days we have asked our colleagues and friends on social media to complete this survey. For three days we have received 54 completed tests, which is sufficient enough to analyze the data and make a conclusion on its results. As the test had 2 different purposes (MOS evaluation and Emotional classification evaluation) we will first speak about mean average opinion and then about the quality of expressive speech.

8.2.1 Speech quality

The final MOS scores achieved from the experiment is shown in Table 8.1.

Table 8.1: MOS scores.

	Neutral	Sad	Angry	Happy	Average
Ground Truth	4.5	4.75	4.2	4.6	4.53
Tacotron2 GST	4.12	3.89	4.08	3.77	3.96
EMOTRON	4.1	4.0	3.71	3.65	3.87

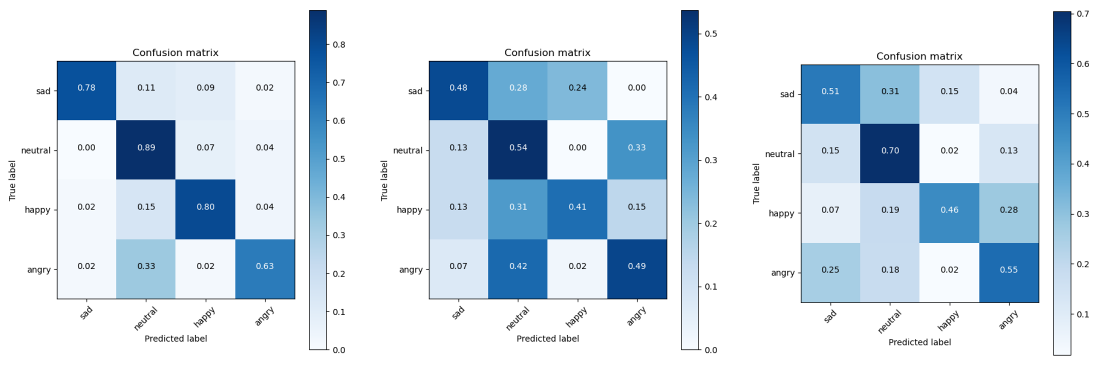
The overall results of synthetic speech are worse than ground truth which means that there is still big room for speech synthesis improvement. As we can see the results of Emotron have shown slightly worse quality than Tacotron2 with GST. We believe that it is due to the GST module and in particular multi-head attention which, we consider, can smooth the result in the crossroad of expressive part and Mel-spectrogram pipeline, during the training process. Emotron classifier is a more deterministic way of implying emotions, so it could have caused a slight decrease in quality during this process.

It is also in some way can be confirmed by looking at results of MOS for each emotion separately: the neutral evaluation score is almost identical between GST and EMOTRON which means that when there is not a big impact from the emotional module, the results are similar.

8.2.2 Emotional clarity

Moving to emotional classification evaluation we can see the results in Figure 8.2 obtained using again a confusion matrix with the survey data this time. We can observe how immediately we are in the presence of high results for the ground truth which is quite natural, moreover, we can also see that both models reproduce the neutral label better than the other emotions.

As for the other emotions instead, both models have decent results (see Table 8.2), and analyzing in detail we can understand that EMOTRON on average can reproduce slightly more truthful emotions than Tacotron2 GST. With the survey data, we also calculated the accuracy of the two models: Tacotron 2 with GST obtained an accuracy of 48% while EMOTRON obtained a slightly better result of 56% accuracy.



(a) Ground Truth **(b)** Tacotron2 GST **(c)** EMOTRON

Figure 8.2: Confusion Matrix Survey

Table 8.2: Emotion accuracy.

	Neutral	Sad	Angry	Happy	Average
Ground Truth	89%	78%	63%	80%	78%
Tacotron2 GST	54%	48%	49%	41%	48%
EMOTRON	70%	51%	55%	46%	56%

So overall two architectures showing similar performance: in terms of speech tacotron2 with GST showing slightly better results, but in terms of emotion transfer EMOTRON has a small advantage.

Chapter 9

Conclusion

In this chapter, we will summarize our work and the most important results. We will also talk about the future developments that can be implemented starting from what we have already done. The goal of this thesis was to develop a TTS capable of transmitting different emotions and during this task, the creation of an emotional module compatible with a baseline model was very challenging as there was a need to try different combinations.

At the beginning we will talk about the path we have undertaken during this thesis activity, highlighting the activities carried out and the skills acquired (see Section 9.1). Later we will briefly explain the results achieved if in line with the objectives set with a conclusion regarding the research work performed (see Section 9.2), Finally, we will talk about possible future works (see Section 9.3).

This field is important from various points of view and the research is very active. This technology is now present in everyday life, just think of the digital home assistants present in all homes, but not only. There are situations in which it is only possible to hear and it is not possible to see either in case of illness or because of certain situations.

On this occasion the Text-To-Speech is essential, to have the best experience it would be of great help to be able to provide fluent speech with some hints of emotion. At the moment there are solutions that produce speech in a truly fluent way and with great quality, but we are still in a research phase regarding the induction of emotions during the generation of speech. We think that our work can be a starting point for this specific task.

9.1 Concluding words

Deeper into the area

As we have not been working with Text-to-Speech and Speech synthesis before the thesis work, we started with the exploration of the fundamentals of this area. We have learned the theory on classical methods for TTS such as Unit Selection and Diphone synthesis. We also provided theoretical information which is required for modern DNN methods.

Modern technologies exploration

We have described in detail models for Text-To-Speech synthesis which is currently considered as one of the most promising and efficient. Considering Mel-spectrogram synthesis we reported details on Tacotron2 and Transformers architectures and reasoning for our choice. One of the most challenging parts of the work was the creation of the emotional module and we have considered and tested several ideas before making the final decision for EMOTRON. Also, we have analyzed the pros and cons of two types of vocoders: Waveglow and Wavenet.

EMOTRON

We have described EMOTRON, our model. We presented its main structure, explaining its modules in detail. For each module, we have defined its structure and the parameters used. We defined what style loss is and in which field it was originally used. We also explained in detail how we calculated the new loss within our network.

Dataset analysis

We carried out an analysis and study of the datasets and then tested them with the models, understanding the importance of data both in terms of quality and quantity, especially for the context of Text-To-Speech. We performed several preprocessing operations and also learned how to work with audio data by performing operations such as denoising, silence detection, etc

Training process

We have defined a pipeline to train our model with. We initially focused solely on generating high-quality Mel-spectograms. Later the focus shifted to the classification of the emotions. We repeated this procedure for the different datasets analyzed previously, for each of the different operations were applied, thus leading us to the final choice of the dataset with the best result.

Survey

Finally, we decided to test produced models and evaluate the MOS and emotional accuracy to see other people's opinions on this matter. We have created a website based on Firebase in order to make a survey. To attract people for completing the survey we used our and our friend's social media services and in the end, we managed to reach 54 submitted tests.

9.2 Final remarks on EMOTRON

In conclusion we have built a new Text-To-Speech model that speaks quite naturally and fluently. Regarding the emotional part, we are able to input an emotion of your choice and to produce a speech in which there is a certain note of emotion. Specifically, the speech quality of EMOTRON is good but does not reach the levels of some networks focused only on speaking. The classification of emotions is just discreet, it is not comparable with some models whose purpose is only to recognize the emotion. But in our case the emotion must not only be recognized but also "inserted" in the desired phrase. There are no metrics so immediate as to verify the results perfectly, and that is why we also had to rely on surveys. After the collection of all survey results, we have made several useful conclusions. The first conclusion is that EMOTRON provides a slightly worse quality of synthetic speech comparing to Tacotron2 + GST. The second is that expressiveness of emotions still far from ground truth audio samples but is a bit better in our proposed architecture than in Tacotron2 with global style tokens.

9.3 Future Works

Concatenation

During the training, EMOTRON uses the two pieces of information inherent to the embedding of the text and that of the emotions. These two pieces of information are added together and then fed to the attention part. The resulting problem is that the emotion is highly influenced by the input text, thus producing different results depending also on the length of the text. The solution is to no longer add the two pieces of information but instead concatenate them. In general, concatenation requires a vastly greater effort, especially the part of the attention that must be able to distinguish where to focus more the attention. On the other hand, the results obtained will be better.

Multispeaker adaptation

The current architecture does not support multi-speaker adaptation, however, it would be a really useful feature for several reasons. The first reason is straightforward and it is the ability of a model to use a trained subset of voices. Secondly, when this option is available, there is an open possibility to start playing with prosody and create new unique voices. Lastly, it is much easier to find a big multispeaker dataset for training purposes.

Testing new dataset

The currently used Blizzard dataset is a good option, but we believe that there is still room for improvement: speech can be more expressive and richer. So we are open to testing our model on brand new datasets. Also, a good source of audio is professionally recorded audiobooks. In the future, we can define a pipeline for cutting, filtering, and aligning audiobooks for getting excellent quality datasets for the TTS system.

Transformers are a good option?

We have already made an attempt for using expressive transformers TTS, however, our approach for emotion embedding acceptable only for Seq2Seq model, so we failed during testing. But we are sure that there might be a way to adapt emotion into transformers, so it can be one of the areas for future research.

Split speech and emotion procedure

EMOTRON now has a parallel training style, that is, while trying to learn to speak, it also learns to classify emotions. The result is a faster process, but at the same time, we have noticed some limitations in the use of datasets. If the two training processes were divided it is possible to use different datasets that give the best result in his task. If through some new method it was possible to combine the two components after training, the results obtained would improve considerably.

Appendix A

Appendix

A.1 Code for all the hyperparameters

Listing A.1: Hyperparameters

```
import tensorflow as tf
from text.symbols import kor_symbols as symbols

def create_hparams(hparams_string=None, verbose=False):
    """Create model hyperparameters. Parse nondefault
       from given string."""

    hparams = tf.contrib.training.HParams(
        ################################
        # Experiment Parameters        #
        ################################
        epochs=400,
        iters_per_checkpoint=5019,
        seed=1234,
        dynamic_loss_scaling=True,
        fp16_run=False,
        distributed_run=False,
        dist_backend="nccl",
        dist_url="tcp://localhost:54321",
        cudnn_enabled=True,
        cudnn_benchmark=False,
        #ignore_layers=['embedding.weight'],
        ignore_layers=['embedding.weight'],
        ################################
        # Data Parameters              #
        ################################
        load_mel_from_disk=False,
        training_files='filelists/
           Emotron_clustered_3_train.txt',
        validation_files='filelists/
           Emotron_clustered_3_vali.txt',
        text_cleaners=['english_cleaners'],
        sort_by_length=False,
        ################################
        # Audio Parameters             #
        ################################
        max_wav_value=32768.0,
        sampling_rate=22050,
        filter_length=1024,
        hop_length=256,
        win_length=1024,
        n_mel_channels=80,
        mel_fmin=0.0,
        mel_fmax=8000.0,
        ################################
        # Model Parameters             #
        ################################
        # n_symbols = 80,
        n_symbols = 65,
        symbols_embedding_dim= 512,
        # Transcript encoder parameters
        encoder_kernel_size = 5,
        encoder_n_convolutions = 3,
        encoder_embedding_dim = 512,
        encoder_embedding_final_dim = 544,
        # Prosody encoder parameters
        prosody_n_convolutions = 6,
        prosody_conv_dim_in = [1, 32, 32, 64, 64, 128],
        prosody_conv_dim_out = [32, 32, 64, 64, 128,
           128],
        prosody_conv_kernel = 3,
        prosody_conv_stride = 2,
        prosody_embedding_dim = 128,
        # Decoder parameters
        n_frames_per_step=1,
        decoder_rnn_dim=1024,
        prenet_dim=256,
        max_decoder_steps=1000,
        gate_threshold=0.5,
        p_attention_dropout=0.1,
        p_decoder_dropout=0.1,
        # Attention parameters
        attention_rnn_dim=1024,
        attention_dim=128,
        # Location Layer parameters
        attention_location_n_filters=32,
        attention_location_kernel_size=31,
        # Mel-post processing network parameters
        postnet_embedding_dim=512,
        postnet_kernel_size=5,
        postnet_n_convolutions=5,
        ################################
        # EMOTRON CLASSIFICATION #
        ################################
        ECL_hidden_size = [256, 32, 3],
        weights_loss = [[1, 1.6, 1.04]],
        ################################
        # Optimization Hyperparameters #
        ################################
        use_saved_learning_rate=False,
        learning_rate=1e-3,
        weight_decay=1e-6,
        grad_clip_thresh=1.0,
        batch_size=8,
        mask_padding=True,
        #######################3
        # reference encoder
        E = 512,
        ref_enc_filters = [32, 32, 64, 64, 128, 128],
        ref_enc_size = [3, 3],
        ref_enc_strides = [2, 2],
        ref_enc_pad = [1, 1],
        ref_enc_gru_size = 512 // 2,

    )

    if hparams_string:
        tf.logging.info('Parsing command line hparams: %s', hparams_string)
        hparams.parse(hparams_string)

    if verbose:
        tf.logging.info('Final parsed hparams: %s', hparams.values())

    return hparams
```

A.2 DTW - Dinamic Time Warping

DTW is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. It has been applied to temporal sequences of video, audio, and graphics data. Indeed, any data that can be turned into a linear sequence can be analyzed with DTW. A well known application has been automatic speech recognition, to cope with different speaking speeds. Other applications include speaker recognition and online signature recognition. In our specific case where we used the audio files, the inputs of the DTW were the MFCCs of the audio files.

Implementation

Two sequences s and t are numpy array of MFCCs. For two value x and y, d(x, y) is a distance between the symbols, in our case d(x, y) is the norm.

Listing A.2: DTW algorithm

```
int DTWDistance(s: array [1..n], t: array [1..m]) {
    DTW := array [0..n, 0..m]

    for i := 0 to n
        for j := 0 to m
            DTW[i, j] := infinity
    DTW[0, 0] := 0

    for i := 1 to n
        for j := 1 to m
            cost := d(s[i], t[j])
            DTW[i, j] := cost + minimum(DTW[i-1, j  ],
                                        DTW[i  , j-1],
                                        DTW[i-1, j-1])

    return DTW[n, m]
}
```

Acronims

AI	Artificial Intelligence Artificial Intelligence is intelligence demonstrated by machines, unlike the natural intelligence displayed by humans and animals, which involves consciousness and emotionality. `https://en.wikipedia.org/wiki/Artificial_intelligence`
TTS	Text-To-Speech A text-to-speech system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech.. `https://en.wikipedia.org/wiki/Speech_synthesis`
EGG	Electroglottograph is a device used for the noninvasive measurement of the degree of contact between the vibrating vocal folds during voice production. `https://en.wikipedia.org/wiki/Electroglottograph`
TD-PSOLA	Time-Domain Pitch-Synchronous Overlap and Add is an algorithm that allows to modify the pitch and the speed of a sound independently of each other. `https://docs.rs/tdpsola/0.1.0/tdpsola/`
RNN	Recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. `https://en.wikipedia.org/wiki/Recurrent_neural_network`
STFT	Short-time Fourier transform A short-time Fourier transform is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time.In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. `https://en.wikipedia.org/wiki/Short-time_Fourier_transform`
ReLu	Rectifier Linear Unit In the context of artificial neural networks, the rectifier is an activation function defined as the positive part of its argument. `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)`
SAL	Sensitive Artificial Listener

LSTM	Long short-term memory Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). https://en.wikipedia.org/wiki/Long_short-term_memory
CNN	Convolutional Neural Network In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery https://en.wikipedia.org/wiki/Convolutional_neural_network
MOS	Mean opinion score is a measure used in the domain of Quality of Experience and telecommunications engineering, representing overall quality of a stimulus or system. https://en.wikipedia.org/wiki/Mean_opinion_score
PCA	Principal component analysis PCA is used in exploratory data analysis and for making predictive models. It is commonly used for dimensionality reduction by projecting each data point onto only the first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. https://en.wikipedia.org/wiki/Principal_component_analysis
GRU	Gated recurrent units are a gating mechanism in recurrent neural networks, introduced in 2014 by Kyunghyun Cho.The GRU is like a long short-term memory (LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate. https://en.wikipedia.org/wiki/Gated_recurrent_unit
VAE	Variational Autoencoders a probabilistic manner for describing an observation in latent space. https://www.jeremyjordan.me/variational-autoencoders
MFC	Mel-frequency cepstrum In sound processing, the mel-frequency cepstrum (MFC) is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. https://en.wikipedia.org/wiki/Mel-frequency_cepstrum
NLP	Natural Language Processing NLP is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. https://en.wikipedia.org/wiki/Natural_language_processing
DTW	Dinamic time warping In time series analysis, dynamic time warping (DTW) is one of the algorithms for measuring similarity between two temporal sequences, which may vary in speed. A well known application has been automatic speech recognition, to cope with different speaking speeds. https://en.wikipedia.org/wiki/Dynamic_time_warping

DSP	Digital Signal Processing is the use of digital processing, such as by computers or more specialized digital signal processors, to perform a wide variety of signal processing operations. `https://en.wikipedia.org/wiki/Digital_signal_processing`
ALS	Amyotrophic lateral sclerosis Neuromuscular disease that results in the progressive loss of motor neurons that control voluntary muscles. `https://en.wikipedia.org/wiki/Amyotrophic_lateral_sclerosis`
DNN	Deep neural networks DNN is an artificial neural network with multiple layers between the input and output layers. `https://en.wikipedia.org/wiki/Deep_learning#Deep_neural_networks`
HMM	Hidden Markov model is a statistical Markov model in which the system being modeled is assumed to be a Markov process call it X with unobservable hidden states. `https://en.wikipedia.org/wiki/Hidden_Markov_model`
MELD	Multimodal EmotionLines Dataset MELD has more than 1400 dialogues and 13000 utterances from Friends TV series. `https://affective-meld.github.io/`

Bibliography

[1] Dan Jurafsky and James H. Martin. *Speech and Language Processing*. 3rd ed. stanford, 2020 (cit. on p. 11).

[2] Zhigang Yin. "An Overview of Speech Synthesis Technology". In: 2018. URL: https://www.researchgate.net/publication/340230997_An_Overview_of_Speech_Synthesis_Technology (cit. on p. 4).

[3] Justin Johnson, Alexandre Alahi, Li Fei-Fei. "Perceptual Losses for Real-Time Style Transferand Super-Resolution". In: 2016. URL: https://arxiv.org/pdf/1603.08155.pdf (cit. on p. 72).

[4] Yuxuan Wang, Daisy Stanton, Yu Zhang, RJ Skerry-Ryan, Eric Battenberg, Joel Shor, Ying Xiao, Fei Ren, Ye Jia, Rif A. Saurous. "Style Tokens: Unsupervised Style Modeling, Control and Transfer in End-to-End Speech Synthesis". In: 2018. URL: https://arxiv.org/pdf/1803.09017.pdf (cit. on p. 70).

[5] RJ Skerry-Ryan, Eric Battenberg, Ying Xiao, Yuxuan Wang, Daisy Stanton, Joel Shor, Ron J. Weiss, Rob Clark, Rif A. Saurous. "Towards End-to-End Prosody Transfer for Expressive Speech Synthesis with Tacotron". In: 2018. URL: https://arxiv.org/pdf/1803.09047.pdf (cit. on p. 70).

[6] Deana L. Pennell,Yang Liu. "Evaluating the effect of normalizing informal text on TTS output". In: 2012. URL: https://ieeexplore.ieee.org/document/6424271.

[7] Rupesh Kumar Srivastava, Klaus Greff, Jürgen Schmidhuber. "Highway Networks". In: 2015. URL: https://arxiv.org/abs/1505.00387 (cit. on p. 24).

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. "Deep Residual Learning for Image Recognition". In: 2015. URL: https://arxiv.org/pdf/1512.03385.pdf (cit. on p. 25).

[9] Yusuke Yasuda, Xin Wang†, Junichi Yamagishi. "END-TO-END TEXT-TO-SPEECH USING LATENT DURATION BASED ON VQ-VAE". In: 2020. URL: https://arxiv.org/pdf/2010.09602.pdf (cit. on p. 60).

[10] Ya-Jie Zhang, Shifeng Pan2, Lei He, Zhen-Hua Ling. "LEARNING LATENT REPRESENTATIONS FOR STYLE CONTROL AND TRANSFER INEND-TO-END SPEECH SYNTHESIS". In: 2019. URL: https://arxiv.org/pdf/1812.04342.pdf (cit. on p. 60).

[11] Yuxuan Wang, RJ Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss†, Navdeep Jaitly,Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio†, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, Rif A. Saurous. "TACOTRON: TOWARDS END-TO-END SPEECH SYNTHESIS". In: 2017. URL: https://arxiv.org/pdf/1703.10135.pdf (cit. on pp. 27, 98).

[12] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. *WaveGlow: A Flow-based Generative Network for Speech Synthesis*. 2018 (cit. on p. 43).

[13] Diederik P. Kingma and Prafulla Dhariwal. *Glow: Generative Flow with Invertible 1x1 Convolutions*. 2018 (cit. on p. 43).

[14] Wei Ping et al. *Deep Voice 3: Scaling Text-to-Speech with Convolutional Sequence Learning*. 2018 (cit. on p. 48).

[15] Jonathan Boilard, Philippe Gournay, R. Lefebvre. "A Literature Review of WaveNet: Theory, Application and Optimization". In: 2019. URL: https://www.researchgate.net/publication/333135603_A_Literature_Review_of_WaveNet_Theory_Application_and_Optimization.

[16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, Koray Kavukcuoglu. "WAVENET: A GENERATIVE MODEL FOR RAW AUDIO". In: 2016. URL: https://arxiv.org/pdf/1609.03499.pdf (cit. on p. 39).

[17] Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, RJ Skerry-Ryan, Rif A. Saurous, Yannis Agiomyrgiannakis, Yonghui Wu. "Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions". In: 2017. URL: https://arxiv.org/abs/1712.05884 (cit. on pp. 32, 36, 98).

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser. "Attention is all you need". In: 2017. URL: https://arxiv.org/pdf/1706.03762.pdf (cit. on p. 46).

[19] Naihan Li, Shujie Liu, Yanqing Liu, Sheng Zhao, Ming Liu, Ming Zhou. "Neural Speech Synthesis with Transformer Network". In: 2019. URL: https://arxiv.org/pdf/1809.08895.pdf (cit. on p. 46).

[20] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. 2016 (cit. on p. 72).

[21] Ron van den Oord, Nal Kalchbrenner, Oriol Vinyals,Lasse Espeholt, Alex Graves,Koray Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders". In: 2016. URL: http://arxiv.org/abs/1606.05328 (cit. on p. 39).

[22] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. *Pixel Recurrent Neural Networks*. 2016 (cit. on p. 40).

[23] Heejin Choi, Sangjun Park, Jinuk Park, Minsoo Hahn. "Emotional Speech Synthesis for Multi-Speaker Emotional Dataset Using WaveNet Vocoder". In: 2019. URL: https://ieeexplore.ieee.org/document/8661919 (cit. on p. 98).

[24] Ohsung Kwon, Inseon Jang, ChungHyun Ahn, Hong-Goo. “Emotional Speech Synthesis Based on Style Embedded Tacotron2 Framework”. In: 2019. URL: https://ieeexplore.ieee.org/abstract/document/8793393 (cit. on pp. 35, 98).

[25] Sheng-Yeh Chen, Chao-Chun Hsu, Chuan-Chun Kuo, Ting-Hao (Kenneth) Huang, Lun-Wei Ku. “EmotionLines: An Emotion Corpus of Multi-Party Conversations”. In: 2018. URL: https://arxiv.org/pdf/1802.08379.pdf (cit. on p. 51).

[26] Heiga Zen, Viet Dang, Rob Clark, Yu Zhang, Ron J. Weiss, Ye Jia, Zhifeng Chen, Yonghui Wu. “LibriTTS: A Corpus Derived from LibriSpeech for Text-to-Speech”. In: 2019. URL: https://arxiv.org/abs/1904.02882 (cit. on p. 53).

[27] Richard Socher. *Sequence-to-sequence with attention*. 2018. URL: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1184/lectures/lecture11.pdf (cit. on p. 21).

[28] Afshine Amidi and Shervine Amidi. *Recurrent Neural Networks cheatsheet*. 2018. URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (cit. on p. 18).

[29] Machine Learns. *Text to Speech Deep Learning Architectures*. 2018. URL: https://erogol.com/text-speech-deep-learning-architectures/.

[30] Open Smile. *AUDIO FEATURE EXTRACTION*. 2021. URL: https://www.audeering.com/opensmile/ (cit. on p. 93).

[31] Kion Kin. *WaveNet: Increasing reception field using dilated convolution*. 2018. URL: https://medium.com/@kion.kim/wavenet-a-network-good-to-know-7caaae735435#:~:text=WaveNet%20is%20an%20generative%20model,second%20for%20a%20reasonable%20quality..

[32] Gary Mckeown, Michel Valstar, Roddy Cowie, Maja Pantic. “The SEMAINE Database: Annotated Multimodal Records of Emotionally Colored Conversations between a Person and a Limited Agent”. In: 2013. URL: https://www.researchgate.net/publication/224248863_The_SEMAINE_Database_Annotated_Multimodal_Records_of_Emotionally_Colored_Conversations_between_a_Person_and_a_Limited_Agent (cit. on p. 54).

[33] Soujanya Poria et al. “MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations”. In: *CoRR* abs/1810.02508 (2018). URL: http://arxiv.org/abs/1810.02508 (cit. on p. 51).

[34] Keith Ito and Linda Johnson. *The LJ Speech Dataset*. https://keithito.com/LJ-Speech-Dataset/. 2017 (cit. on p. 50).

[35] Carlos Busso et al. “IEMOCAP: Interactive emotional dyadic motion capture database”. In: *Language Resources and Evaluation* 42 (Dec. 2008), pp. 335–359 (cit. on p. 57).

[36] M. Kathleen Pichora-Fuller and Kate Dupuis. *Toronto emotional speech set (TESS)*. Version DRAFT VERSION. 2020. URL: https://doi.org/10.5683/SP2/E8H2MF (cit. on p. 63).

[37] Steven R. Livingstone and Frank A. Russo. “The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English”. In: *PLOS ONE* 13.5 (May 2018), pp. 1–35. URL: https://doi.org/10.1371/journal.pone.0196391 (cit. on p. 63).

[38] Stanford. “Diphone TTS architecture”. In: 2019. URL: https://nlp.stanford.edu/courses/lsa352/lsa352.lec4.6up.pdf (cit. on pp. 7, 9).

[39] M. G. de Pinto et al. “Emotions Understanding Model from Spoken Language using Deep Neural Networks and Mel-Frequency Cepstral Coefficients”. In: (2020), pp. 1–5 (cit. on pp. 64, 91).